Correctness, Precision and Efficiency in the Sharing Analysis of Real Logic Languages

Enea Zaffanella

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy

The University of Leeds
School of Computing

The candidate confirms that the work submitted is his own
and that appropriate credit has been given
where reference has been made to the work of others.

September 2001

A Roberta e Mattia

It can be true, it can be false. You'll be given the same reward.
The Clash

Acknowledgements

I wish to thank Roberto Bagnara and Patricia M. Hill.

Since 1997, Roberto, Pat and myself have been working together as a team, investigating static analysis of (constraint) logic programs. It is difficult to imagine this thesis without them. As a matter of fact, all the results presented here have been obtained thanks to our collaboration; Roberto and Pat are also co-authors of all of my publications on sharing analysis. Honestly, it is *not* possible to distinguish my contributions from their own: it is joint work, as simple as that.

Roberto deserves very special thanks. We first met when we were PhD students, sharing an office in Pisa; then, when I started working outside the academic world, Roberto became a co-author and my only link to the research community; now that I am working in Parma as a fellow researcher, he is both a co-author and an office-mate. During these years, which have seen so many changes in my life, our friendship has been a firm constant.

I am also grateful to Roberto for having introduced me to Pat. Initially, my knowledge of Pat was limited to the hundreds of e-mail messages related to our work. While immediately appreciating her value as a researcher, this was rather "impersonal". Fortunately, as we found more and more opportunities to meet each other, this relation evolved in a true friendship. Pat has also been my PhD supervisor in Leeds and I wish to thank her for being such an excellent one. She has been a continual source of useful suggestions and encouragement. She also bore the burden of almost all the tedious administrative work related to my PhD, which was far beyond her duties as a supervisor.

Other people have contributed, perhaps indirectly, to this thesis. I wish to thank Giorgio Levi, my supervisor when I was a student in Pisa. Giorgio sparked my interest on logic programming and abstract interpretation, introducing me to academic research. I would like to thank Roberta Gori, who recently joined our research team and kindly helped me with some of the technical results. I also thank Prof. Giovanni Bassanelli, the Director of the Department of Mathematics at the University of Parma, for having allowed me to undertake this PhD at Leeds while working as a research fellow in Parma.

Without Roberta and Mattia, my beloved family to whom this thesis is dedicated, none of this would have been possible, meaningful or even desirable. They are my foundations, on which everything else rests.

Leeds, September 2001

Addendum

I would like to thank Prof. Anthony G. Cohn and Dr. Andrew M. King, respectively my internal and external PhD examiners, for the time spent reading this thesis and for their useful comments.

Parma, December 2001

Abstract

For programming languages based on logic, a knowledge of variable sharing is important; for instance, for their automatic parallelization and for many optimizations of the unification procedure, such as occurs-check reduction. Because of its usefulness, a considerable amount of research has been done on the design and development of techniques for the static analysis of variable sharing. Despite this fact, some of the most important issues related to the specification and implementation of a practical sharing analysis tool, such as the correctness, the precision and the efficiency of the analysis, have lacked satisfactory solutions. The thesis reports on our work in rectifying this situation and, thereby, enhancing the state-of-the-art on sharing analysis. Our contributions include: a correctness proof for the set-sharing domain of Jacobs and Langen that does not depend on the presence of the occurs-check in the concrete unification procedure; the definition of the simpler abstraction of set-sharing that is guaranteed to achieve the same precision on both variable independence and groundness; the specification, on this new domain, of an abstract unification operator having polynomial complexity, as opposed to the exponential complexity of the abstract unification operator defined on set-sharing; the generalization of all the above results to a combined abstract domain including set-sharing, freeness and linearity information; an extensive experimental evaluation, including both the validation of the above results as well as the implementation and comparison of many other recent proposals for more precise sharing analysis domains; and the specification of a new representation for set-sharing which, by allowing for the definition of a family of widening operators, solves all the scalability problems of the analysis, while having a negligible impact on its precision.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the execution of a logic program, two or more program variables can be bound to terms sharing a common variable. Since a knowledge of variable sharing is essential for triggering optimizations of logic languages, as well as providing good approximate information about many other interesting properties, in the recent past, sharing analysis has received much attention from the researchers working on the static analysis of logic languages. In this chapter, after presenting the goals of practical static analyses, we introduce sharing analysis for logic languages and briefly survey the main proposals appeared in the literature. We then discuss whether or not this state-of-the-art satisfies the requirements of a practical static analysis: we highlight how a key problem has been completely disregarded while others have been provided with unsatisfactory solutions. The remaining chapters then focus on these open issues, from both theoretical and practical perspectives, presenting new solutions to some of them and providing more insight on what should be the direction of future research in this area.

1.1 The Goals of Static Analysis

A static analyzer is a computer program that takes as input an arbitrary program and delivers some partial information about the run-time behavior of that program. This information is then made available to other semantics-based tools, including optimizing compilers, debuggers and (semi-)automatic program verification systems. Any static analyzer needs to fulfill the following tasks:

- it has to produce *correct* information; and
- it has to *terminate*.

From a theoretical point of view, the termination of the analysis is the real motivation for considering approximations of program properties, since the exact characterization of program behavior is, in general, not computable in finite time. Whenever the final goal is the development of a useful analysis tool, then the mandatory requirements specified above are not enough. Any *practical* static analyzer also needs to enjoy the following properties, which are as important as correctness and termination:

- it has to produce *precise* information; and
- it has to be *efficient*.

From a practical point of view, the property of correctness is too weak, because approximations that are not precise are very likely to be useless. On the other hand, if the analyzer is to be included in production tools, then termination should be achieved in a reasonable amount of time. The exact meaning of the word "reasonable" depends on the considered application and big variations have to be expected in this sense. For instance, a few seconds can be considered reasonable for the optimization phase of a compiler, particularly during the program development process; in contrast, hours of CPU time can be dedicated to the optimization of the production version of the software, since in this case thousands (or millions) of program runs will benefit from any resulting improvement. In no cases should a static analyzer take years to produce useful results.

Thus, a static analyzer should be correct, precise and efficient (note that the latter requirement also enforces termination). Abstract interpretation is a mathematical framework that allows the formal specification of correct analyses and a comparison of their relative precision. In this framework, the concrete semantics domain and operators of the language are replaced by correct abstract counterparts, which are then used to mimic all the possible concrete computations of the analyzed program. Termination and efficiency of the analysis are obtained by imposing suitable restrictions on the structure of the abstract domain and/or by dynamically adjusting the approximation level using widening operators. It follows that precision and efficiency are often contrapositive goals and that a satisfactory trade-off can often only be obtained by coupling the theoretical design phase with a deep experimental evaluation. Even though abstract interpretation concepts and techniques are independent from the particular programming language considered, declarative languages are probably one of the most studied fields of application. One reason for this is that, while enjoying a clean and elegant semantics, declarative languages are normally provided with just a few "basic" operations that are very general and powerful, such as matching for functional programming, unification for logic programming and constraint solving for constraint (logic) programming. The optimized implementation of these operations turns out to be a key issue where static analysis is really helpful.

1.2 Sharing Analysis for Logic Languages

> "Analysis of Variable Aliasing is the centerpiece of analysis of non-trivial Logic Programs." [Lan90]

In logic programming, the information provided by sharing analysis can be used for several applications, including the simplification of concrete unification in optimizing compilers, the automatic parallelization of logic programs, the identification of unifications where the occurs-check can be safely omitted (occurs-check reduction), and debugging. Besides practical applications, the study of abstract domains for sharing analysis is interesting also from a theoretical point of view. There is considerable research interest in the definition of operators that transform the abstract domains given as input, returning other domains that are more precise or have a more compact representation. Sharing analysis domains, since they encode many different kinds of information that interact with each other in complex ways, turn out to be useful test-cases for probing the applicability of these meta-level operators. In the following sections, we will explain the kind of information collected by a sharing analysis and how this is used in its applications. Then, we will survey the main literature on sharing analysis, trying to provide a picture of what was the state-of-the-art in this field when we started our research.

1.2.1 Information Gathered by Sharing Analysis

The terminology "sharing analysis" identifies several different kinds of information about the instantiation of logic variables at specific program points.

To avoid confusion, it is worth putting some emphasis on the meaning, in a static analysis context, of contrapositive adjectives like definite/possible and universal/existential. A property is said to *definitely* hold at a particular program point if the property holds for the set of computations reaching that program point, this set considered as a whole. Often, properties on sets of computations can be checked by looking at the individual elements in the set: in particular, a *universal* property is one that holds for all of the computations in this set; an *existential* property is one that holds for at least one of these computations. In contrast, in a static analysis context, when we say that a property *possibly* holds, formally we provide no information at all: due to the intrinsic approximation, it may well be the case that the property does not hold. However, a well-established convention in the field is to consider the *negation* of possible properties, so that if a property cannot possibly hold we correctly conclude that its negation definitely holds. This might cause some confusion, in particular when dealing with existential properties. Note however that throughout this thesis we will always be dealing with universal properties.

This stated, we now informally describe the many parts in which a sharing description can be decomposed.

Independence. Two variables are said to be *definitely independent* if they are always bound to terms that do not share a common variable. Equivalently, if we are not able to prove that two variables are definitely independent, we say that they *possibly share* (we can also say that they are *possibly aliased*). Given two *sets* of variables, they are said to be independent if they are pair-wise independent, meaning that each variable in the first set is independent from each variable in the second set. Thus, definite independence corresponds to (the negation of) possible *pair*-sharing.

Groundness. A variable is said to be definitely *ground* if it is always bound to a term containing no variables. Ground variables cannot share with other variables, so that they are always independent from all the variables; moreover, they cannot be further instantiated, ensuring that this independence property will hold as the computation progresses.

Freeness. A variable is said to be definitely *free* if it cannot be bound to a non-variable term. Two free variables that are not independent are said to be possibly *aliased*: note that in this case the terminology is really apt, since if the two variables do share they provide different names denoting the same term.

Linearity. A variable is said to be definitely *linear* if it can only be bound to terms containing single occurrences of variables. Note that a variable that is either definitely ground or definitely free is also definitely linear.

Besides the ones above, which nowadays can be considered the classical ingredients of a sharing analysis, recent work has highlighted how other properties can contribute to the accuracy of sharing information. These will be introduced later in the thesis, when discussing the potential benefits arising from considering them.

1.2.2 Applications of Sharing Analysis

Sharing analysis was initially proposed for occurs-check reduction and for the automatic parallelization of logic programs. However, sharing analysis has been shown to provide useful information for a number of other useful applications. For example, a classical application of many static analyses for logic programs is the optimized compilation of the concrete unification procedure [Tay91, VD92]. Sharing analysis provides useful information in this respect. For instance, knowledge about the freeness or groundness of the unified terms enables the replacement of the general unification procedure by a combination of simple tests and assignments. With the advent of concurrent logic languages, sharing analysis information has also been used for the optimization of distributed unification algorithms: in this cases, important performance improvements can be achieved by identifying those cases where unification can be computed locally, i.e., when variable bindings do not need to be communicated to the other processes running concurrently.

Occurs-check reduction [CKS96, Pla84, Søn86] is an interesting application of static analysis that, in our opinion, has not received the attention it deserves. It is well-known that many implemented logic programming languages (in particular, almost all Prolog systems) omit the *occurs-check* from the unification procedure. Occurs-check reduction amounts to identifying all the unifications where such an omission is potentially unsafe, so that these can be replaced by calls to a safe, but more expensive, variant of the unification procedure, performing the occurs-check. For this purpose, as pointed out by [Pla84, Søn86], information on variable independence and linearity is crucial.

Variable independence is also essential for the efficient exploitation of strict independent AND-parallelism [CDD85, HG90, HR95, JL92, MH92]. Informally, two atoms in a goal are executed in parallel if, by a mixture of compile-time and run-time checks, it can be guaranteed that they do not share any variable. This form of independence implies the absence of *binding conflicts* at run-time, that is, it will never happen that the processes associated to the two atoms try to bind the same variable. The goal of the analysis is thus to gather enough information so that as many as possible of the run-time checks can be replaced by compile-time checks, therefore reducing the overhead for parallelization. In the more general case of non-strict independent AND-parallelism [CH94, HR95], the requirements for running two processes in parallel are weakened by permitting some restricted form of variable sharing to occur. By coupling independence with definite freeness information, it is possible to identify at compile-time some of the cases when such a transformation is allowed.

A recent line of research shows that sharing analysis can be usefully exploited for the efficient refinement of induced predicate definitions in Inductive Logic Programming [BDJ+00]. In this case, independence is exploited to avoid the expensive recomputation of those parts of the refined predicates that are not influenced by the considered refinement.

1.2.3 Domains for Sharing Analysis: the State-of-the-Art

Sharing analysis for logic programs has a 15 year history. After the first simple proposals, Søndergaard [Søn86] outlined a sharing analysis domain, called ASub, that derived useful information for the occurs-check reduction problem. The domain ASub encodes, in their simplest form, three of the basic properties that we introduced in Section 1.2.1, namely independence, groundness and linearity. Groundness information is represented by the set of definitely ground variables. Variable independence is represented by the set of all the pairs (v, w) of distinct program variables that possibly share a variable: for this reason, the domain ASub is said to be a *pair-sharing* domain. Note that, according to the discussion in Section 1.2.1, the independent pairs of variables are those pairs not occurring in such a set. Similarly, linearity is represented by the set of all the variables that are possibly non-linear. Actually, in [Søn86], possible pair-sharing and possible non-linearity information are mixed in the representation of abstract elements, by letting pairs of the form (v, v) denote the possible non-linearity of variable v. Note that any element of the domain ASub can be represented in a polynomial amount of space (in the number of program variables). A similar polynomial bound is enjoyed by the abstract unification algorithm informally proposed in [Søn86], which has been formalized in [CDY91].

Jacobs and Langen [JL89] proposed a different domain for sharing analysis, called *Sharing*. An element of this domain is a set of sets of variables, as opposed to the set of pairs of variables characterizing the independence component of the domain ASub; for this reason, the domain *Sharing* is said to be a *set-sharing* domain. The intuition behind this domain is that any set occurring in an abstract element corresponds to one or more variables that is possibly shared by all and only those variables occurring in the set. As we will see in the following chapters, both independence and groundness information can be represented in this way; however, the novelty with respect to the domain ASub is that it is possible to encode the *dependencies* between these properties, therefore obtaining a more precise description. When compared to ASub, the *Sharing* domain suffers from significant efficiency problems: the initial abstract description of a concrete element is still polynomial in the number of program variables, but an exponential amount of space may be needed to represent the domain elements computed as the analysis progresses. Moreover, the abstract unification operator has a worst case exponential complexity in the size of the representation. For these reasons, the simpler domain ASub, when compared to *Sharing* from a practical point of view, was rightfully considered a better trade-off between precision and efficiency.

Despite these practical problems, the set-sharing domain *Sharing* has received much attention by most of the subsequent research in the field, becoming a de-facto standard for sharing analysis. As a first improvement, in his PhD thesis [Lan90], Langen enriched the *Sharing* domain by adding linearity information. Muthukumar and Hermenegildo [MH92] exploited the synergistic effect of the combination of variable freeness with variable independence. A composition with both linearity and freeness was defined and proved correct in [HW92]. Since then several authors [BCM94a, CDFB96, Fil94, Kin94, KS94, MSJB95] worked on more or less different combinations for these properties, by including more information or resorting to more precise abstract unification operators. In many of them some kind of explicit structural information was also considered, by recording the concrete structure of terms up to a certain depth: more or less, these proposals can be seen as variations of the *abstract equation systems* presented in [CDFB96] or the generic structural domain constructor defined in [CLV94, CLV00].

1.3 Issues in Sharing Analysis

As discussed in Section 1.1, a practical static analysis tool has to enjoy three basic requirements: correctness, precision and efficiency. We will now discuss these requirements, arguing whether or not the available sharing analysis tools can be considered satisfactory solutions.

1.3.1 Correctness ...

In abstract interpretation, the concrete semantics of a program is approximated by mimicking its computation on the abstract domain. To this end, each elementary concrete operation is replaced by an abstract operation which is a correct approximation of the concrete one. For logic programming, the key elementary operation is *unification*, which computes a solution to a set of equations. For global correctness of the abstract semantics, there needs to be, therefore, a corresponding abstract operation that is sound with respect to unification.

In theoretical terms, once the abstract domain has been specified and formally related to the concrete one, the specification of the best correct approximation of a concrete operator is readily available and the implementer of the domain only needs to code it. Unfortunately, very often a direct implementation of this specification is not possible, since it is expressed as the composition of functions that are not finitely computable: thus, we need to formally manipulate such a specification, so as to turn it into an executable one. In many cases this is a simple task and the abstract operator obtained is, *by definition*, correct. There are important cases, however, when the concrete operator is really far from being simple: unification is one of these. As a consequence, abstract unification algorithms have traditionally been obtained by a trial-and-error process, where the theoretical specification is merely used as a hint. The drawback of this approach is that any abstract unification operator obtained in this way needs to be *proved* correct with respect to concrete unification.

The specification and proof of correctness of abstract unification for sharing analysis domains have been difficult and error-prone tasks, in particular when integrating the different components of sharing information: for instance, [CDFB96] reports that the first proposal for combining independence and freeness information [MH91] was corrected in [BdlBH94]; for the combination of independence with linearity, King [Kin93, Kin00] points out an error in [CDY91] affecting the abstract unification on the domain ASub. In both cases, no fundamental error was present: after simple corrections, both algorithms have been proved correct and are still used as base references in sharing analysis research.

In summary, after the application of simple patches, the available sharing analysis tools have been proved correct for the analysis of logic languages. However, in the next section we will explain why all of the above results are not satisfactory when considering the analysis of implemented logic languages.

1.3.2 ...and the Occurs-Check?

An important step in standard unification algorithms based on that of Robinson [Rob65] (such as the Martelli-Montanari algorithm [MM82]) is the *occurs-check*, which avoids the generation of infinite (or cyclic) data structures. With such algorithms, the resulting solution is both unique and idempotent. However, in computational terms, the occurs-check is expensive and the vast majority of Prolog implementations omit this test, although some Prolog implementations do offer unification with the occurs-check as a separate built-in predicate (in ISO Prolog [ISO95] the predicate is `unify_with_occurs_check/2`). If the unification algorithm is based on the Martelli-Montanari algorithm but without the occurs-check step, then the resulting solution may be non-idempotent. Consider the following example.

Example 1.1 *Consider the equation $p\big(z, f(x,y)\big) = p\big(f(z,y), z\big)$ and an empty initial substitution. We apply the steps in the Martelli-Montanari procedure but without the occurs-check:*

	equations	*substitution*
1	$p(z, f(x,y)) = p(f(z,y), z)$	$\varnothing$
2	$z = f(z,y), f(x,y) = z$	$\varnothing$

3	$f(x,y) = f(z,y)$	$\bigl\{z \mapsto f(z,y)\bigr\}$
4	$x = z, y = y$	$\bigl\{z \mapsto f(z,y)\bigr\}$
5	$y = y$	$\bigl\{z \mapsto f(z,y), x \mapsto z\bigr\}$
6	$\varnothing$	$\bigl\{z \mapsto f(z,y), x \mapsto z\bigr\}$

Then $\sigma = \bigl\{z \mapsto f(z,y), x \mapsto z\bigr\}$ *is the computed substitution; it is not idempotent since, for example,* $\sigma(x) = z$ *and* $\sigma\bigl(\sigma(x)\bigr) = f(z,y)$.

Non-standard equality theories and unification procedures are also available and used in many logic programming systems. In particular, there are theoretically coherent languages, such as Prolog II and its successors [Col82, Col90], SICStus Prolog [SIC95], and Oz [ST94], that employ an equality theory and unification algorithm based on a theory of *rational trees* (possibly infinite trees with a finite number of subtrees). As remarked in [Col82], complete (i.e., always terminating) unification with the omission of the occurs-check solves equations over rational trees. Complete unification is made available by several Prolog implementations and the substitutions computed by such systems are in *rational solved form*, which is a much weaker property than idempotence. As an example, the substitution $\{x \mapsto f(x)\}$, which is clearly non-idempotent, is in rational solved form and could itself be computed by the above algorithms.

It is therefore important that theoretical work in data-flow analysis makes no assumption that the computed solutions are idempotent. In spite of this, at the time we started our research, all the correctness proofs for sharing analysis of logic programs assume a domain of idempotent substitutions. This is the case both for the correctness results presented in [CDY91] for the domain ASub, and for the results presented in [BCM94a, CF99, HW92, JL89, Lan90], for domains based on set-sharing.

Therefore, one of the main targets of our research is the problem of providing a correctness proof for sharing analysis whose validity does not depend on the presence, or even the absence, of the occurs-check in the concrete unification procedure. It is worth stressing that such a contribution cannot be obtained, in general, by simply adapting the proofs developed for idempotent substitutions. As acknowledged in [LMM88], the possibility to restrict attention to idempotent substitutions, by itself, is a considerable simplification, since their semantics is that of mathematical equality. In contrast, using the words in [LMM88], proofs for non-idempotent substitutions are

> "...painful and tricky, as we are now dealing with a destructive assignment."

Note that the quoted sentence was still referring to substitutions that are consistent in the theory of finite trees: namely, non-idempotent substitutions defining finite trees only. Dealing with arbitrary substitutions in rational solved form, where cyclic bindings are allowed to occur, can be fairly expected to be even worse. For these reasons, in all of our correctness proofs, efforts are made to properly justify any non-trivial passage, possibly by making an explicit reference to a previously proved result. Probably, since we are trading space for clarity, such an approach results in longer proofs.

1.3.3 Precision *and* Efficiency

In static analysis, precision and efficiency issues are closely intertwined. As a matter of fact, if considered in isolation, efficiency poses no problem at all: the most efficient (and correct) static analysis always answers "don't know", regardless of the particular input program. Clearly, difficulties arise when precision and efficiency are joint goals.

In our opinion, precision should always have priority over efficiency. Obtaining precise information is the very motivation for performing a static analysis and, as new applications of this information are conceived and developed, the precision requirements keep increasing. In contrast, once the precision requirements have been fixed, efficiency problems become less important as the computer technology progresses. As a rule of thumb, any tiny precision improvement, by itself, outweighs the corresponding efficiency loss (provided there is such a loss; as we will see, it is also common that a precision improvement gives rise to speed-ups). The only and obvious exception to the above rule is that the results of the static analysis should always be obtained in a reasonable amount of time.

Strictly speaking, the only way to obtain an unbiased measure of the precision of a static analysis is the full implementation of the particular application for that static analysis. For instance, in the case of an analysis delivering information for an optimizing compiler, one should implement the analyzer *and* the module of the compiler performing the optimization phase, so that the precision of the analysis is evaluated by measuring the efficiency improvements obtained thanks to the considered optimizations.

Unfortunately, there are several good reasons why this approach is seldomly used in practice, at least while doing research. The most important one is that, in many cases, the overall process is a formidable task, requiring a substantial effort and adequate resources. Moreover, as is the case for the sharing analysis of logic languages, it might happen that the same information can be used by many different applications: to be fully compliant with the above approach, one should implement all of these applications and measure the improvements obtained by each of them. Finally, it must be considered that, potentially, the information provided by static analysis may be useful for a practical application that still has to be conceived. For instance, when researchers started studying the sharing analysis for logic programs, probably no one imagined that this information could be usefully exploited in the field of Inductive Logic Programming [BDJ$^+$00].

Therefore, as is usual in this research field, we will adopt a more pragmatic approach: precision will be evaluated by measuring the information delivered by the static analyzer, almost independently from the application using it. A questionable point is whether we should perform the comparison on *all* the information produced by the analyzers or, in contrast, we should project these results on the *observable properties*, i.e., those parts of the analysis results that will be actually provided to the final applications. We argue that the latter is a better alternative, since it allows us to abstract from the useless information that a particular abstract domain may contain. For instance, in the case of sharing analyses for logic languages, a suitable set of observable properties is probably given by variable independence, groundness, freeness and linearity.

Even when considering the above simplifications, the actual precision comparison can be performed by following two fundamentally different approaches. The first one, based on the theory of abstract interpretation, consists in checking whether one of the analyses is *uniformly* more precise than the other, meaning that it will always return more precise results. The main advantage of this approach is that it does not require the implementation of the two analyses. On the other hand, very often the considered analyses provide results that are not comparable in this sense, meaning that there exists cases, no matter how rare they can be, when each one outperforms the other. For instance, this is the case when comparing the sharing analysis domains ASub and *Sharing*, since the first one tracks linearity while the second one is more precise, e.g., for groundness [CF93]. It might come as a surprise the fact that, from this theoretical perspective, all the classical combinations of set-sharing with freeness and linearity, such as those specified in [BCM94a, HW92, Lan90], are still not comparable with respect to the pair-sharing domain ASub (when using the correct abstract unification operator as specified in [Kin00]). The reason, which was overlooked even by the experts, is that in all the above cases the integration of set-sharing with linearity is not as powerful as that originally suggested by Søndergaard. One of the contributions of this thesis will be the specification of a new abstract unification operator for the combination of set-sharing, freeness and linearity: the resulting analysis will be uniformly more precise than the analysis based on ASub.

Another possibility, based on a more pragmatic view, is to implement the two analyses and compare the actual results obtained when analysing a given set of benchmarks. By doing this, we can derive some useful hints on the relative precision of each analyses, even when they are not comparable from the formal point of view. Clearly, in this case the results also depend on the considered set of benchmarks, which is assumed to be a representative sample. This second approach, by requiring the implementation of the analyses, happens to be much less common. In particular, many of the recent proposals for sharing analysis have never been implemented. Unfortunately, even the few experimental comparisons of different analyses that have been done have been mainly performed on small benchmark suites. These have shown that, from this pragmatic perspective, the classical combinations of set-sharing with freeness and linearity provide better results than ASub, while incurring significant efficiency problems when considering the bigger programs. This scalability problem has been always disregarded. The implicitly suggested solution that has been adopted is to revert to using the simpler domain ASub. The drawbacks of such an approach are evident: to avoid efficiency problems in the analysis of a few programs, we potentially degrade the precision of even those analyses with no efficiency problems at all. Therefore, since we value precision more than efficiency, this solution is not really satisfactory.

In this thesis, when comparing the precision and the efficiency of different domains, we will consider both the formal and the pragmatic approach. In particular, for the set-sharing domain of Jacobs and Langen we will define a new abstract domain that, while formally obtaining the same precision on all the observable properties, is characterized by an abstract unification operator that can be computed in polynomial time. We will then consider our new domain combination for set-sharing, freeness and linearity and we will generalize the above results. The theoretical results will be also validated by an extensive experimentation, showing that even in practice it is possible to obtain significant improvements in the efficiency of the analysis with no precision loss. Furthermore, the implemented abstract domain will be used as a starting point for the systematic implementation and experimental evaluation of many different proposals for enhanced sharing analysis domains, therefore allowing an investigation of the corresponding precision improvements.

Even though the new domain for sharing analysis can be considered much more efficient with respect to the previous proposals based on set-sharing, it still suffers from scalability problems. To solve this problem, we will define and prove correct a new representation for set-sharing that supports the specification of many possible widening operators. Using this representation, time and space requirements can be dynamically adjusted during the analysis. The experimental evaluation will show that all termination problems are solved, while the precision losses due to the widenings are extremely rare.

1.4 Plan of the Thesis

The thesis is organized as follows.

Chapter 2 presents most of the notation and terminology to be used throughout the thesis, as well as providing the necessary background references on logic programming and abstract interpretation.

Chapter 3 introduces the set-sharing domain of Jacobs and Langen and proves the correctness of its abstract unification operator, for both finite-tree and rational-tree logic languages. (This chapter is mainly based on the results of [HBZ98, HBZ02].)

Chapter 4 shows that the set-sharing domain is over-complex when the goal is the computation of pair-sharing (that is, variable independence). By defining an equivalence relation on the domain elements, the domain *PSD* is characterized as the weakest abstraction of set-sharing able to achieve the same precision on pair-sharing. Abstract unification on *PSD* is shown to have a polynomial complexity, in contrast to the exponential complexity of the corresponding operator on the set-sharing domain. (This chapter is mainly based on the results of [BHZ97, BHZ02].)

Chapter 5 generalizes the results of the previous chapter by defining a hierarchy of abstractions of the set-sharing domain each one enjoying a completeness result. The *PSD* domain and the groundness domain *Def* are shown to be instances of this construction. By investigating the lattice structure of these domains and exploiting the concept of abstract domain complementation, *PSD* is factorized into three sub-domains, one of which is the domain representing the pair-sharing property. (This chapter is mainly based on the results of [ZHB99, ZHB02].)

Chapter 6 considers the integration of set-sharing with freeness and linearity information, denoted by *SFL*. A novel abstract unification operator is defined and proved correct for both finite-tree and rational-tree languages. By generalizing the results of Chapter 4, the non-redundant version of *SFL*, based on the domain *PSD*, is shown to achieve the same precision on groundness, independence, freeness and linearity.

Chapter 7 describes our implementation of the above mentioned domains. An extensive experimental evaluation is provided, confirming that the non-redundant version of *SFL* achieves the same precision while obtaining a significant efficiency improvement.

Chapter 8 considers a number of proposals for enhanced sharing analyses. In order to measure the corresponding precision gains, all but one of them are implemented and experimentally evaluated. (This chapter contains an extended and improved version of the results of [BZH00].)

Chapter 9 attacks the problem of obtaining a scalable sharing analysis without losing precision. The idea underlying a recent proposal for a simpler sharing analysis is turned into the definition of a suitable widening operator for the set-sharing domain, relying on a new representation for set-sharing. (This chapter is mainly based on the results of [ZBH99b].)

Chapter 10 presents a few of the more recent publications on sharing analysis. The relationship between these works and the results presented in this thesis are discussed.

We conclude in Chapter 11.

Chapter 2

Notation and Background

In this chapter we introduce most of the notation and terminology that will be needed throughout the thesis. Besides basic concepts, we provide the necessary background references for both logic programming semantics and abstract interpretation.

2.1 Basic Concepts

2.1.1 Sets, Multisets and Sequences

For a set S, $\wp(S)$ is the power set of S. The cardinality of S is denoted by $\# S$ and the empty set is denoted by $\varnothing$. The notation $\wp_{\mathrm{f}}(S)$ stands for the set of all the *finite* subsets of S, while the notation $S \subseteq_{\mathrm{f}} T$ stands for $S \in \wp_{\mathrm{f}}(T)$.

A *multiset* is a mathematical entity that is like a set except for the fact that it can contain multiple occurrences of identical elements. Note that we do not pose any cardinality restriction on the occurrences of an element: elements can occur infinitely often in a multiset. If an element a occurs more than once in a multiset M, we write $a \Subset M$.

For any set S, $S^\star$ is the set of all the finite sequences built from elements in S. The empty sequence is denoted by ϵ. For each $e \in S$ and $s \in S^\star$, $e \,.\, s \in S^\star$ denotes the sequence obtained by concatenating the sequence formed by element e with the sequence s. The length of sequence s is denoted by $|s|$ and defined as

$$|s| = \begin{cases} 0, & \text{if } s = \epsilon; \\ 1 + |s'|, & \text{if } s = e \,.\, s'. \end{cases}$$

2.1.2 Orders, Lattices and Closure Operators

A *preorder* $\preceq$ over a set P is a binary relation that is reflexive and transitive. If $\preceq$ is also antisymmetric, then it is called a *partial order*. A partial order $\preceq$ is a *linear order* or *total order* if any two elements are comparable, that is, for each $x, y \in P$, either $x \preceq y$ or $y \preceq x$. A set P equipped with a partial order $\preceq$ is said to be *partially ordered* and sometimes written $\langle P, \preceq \rangle$. Partially ordered sets are also called *posets*. An *increasing chain* over the poset $\langle P, \preceq \rangle$ is a subset X of P such that $\preceq$ is a linear order on X.

Given a poset $\langle P, \preceq \rangle$ and $S \subseteq P$, $y \in P$ is an *upper bound* for S if and only if $x \preceq y$ for each $x \in S$. An upper bound y for S is the *least upper bound* (or lub) of S if and only if, for every upper bound y' for S, $y \preceq y'$. The lub, when it exists, is unique. In this case we write $y = \operatorname{lub} S$. The terms *lower bound* and *greatest lower bound* (or glb) are defined dually. A *complete partial order*, or simply *cpo*, is a poset such that every increasing chain has a least upper bound.

A poset $\langle L, \preceq \rangle$ such that, for each $x, y \in L$, both $\operatorname{lub}\{x, y\}$ and $\operatorname{glb}\{x, y\}$ exist, is called a *lattice*. In this case, lub and glb are also called, respectively, the *join* and the *meet* operations of the lattice. A *complete lattice* is a lattice $\langle L, \preceq \rangle$ such that every subset of L has both a least upper bound and a greatest lower bound. The *top* element of a complete lattice L, denoted by $\top$, is such that $\top \in L$ and $\forall x \in L : x \preceq \top$. The *bottom* element of L, denoted by $\bot$, is defined dually.

As an alternative definition, a lattice is an algebra $\langle L, \wedge, \vee \rangle$ such that $\wedge$ and $\vee$ are two binary operations over L that are commutative, associative, idempotent, and satisfy the following *absorption laws*, for each $x, y \in L$: $x \wedge (x \vee y) = x$ and $x \vee (x \wedge y) = x$.

The two definitions of lattices are equivalent, as can be seen by defining:

$$x \preceq y \quad \overset{\text{def}}{\iff} \quad x \wedge y = x \quad \overset{\text{def}}{\iff} \quad x \vee y = y$$

and

$$\operatorname{glb}\{x, y\} \overset{\text{def}}{=} x \wedge y, \qquad\qquad \operatorname{lub}\{x, y\} \overset{\text{def}}{=} x \vee y.$$

If D and C are sets, then $f \colon D \to C$ denotes a total function f mapping elements of the domain D into elements of the co-domain C; for any $D' \subseteq D$, we write $f(D')$ to denote the set $\{\, f(d) \mid d \in D' \,\} \subseteq C$. The function f is injective if, for each $d, d' \in D$, $f(d) = f(d')$ implies $d = d'$; it is surjective if $f(D) = C$; it is a bijection if it is both injective and surjective. Let $\langle P^\sharp, \preceq^\sharp \rangle$ be a poset. Functions $f, g \colon D \to P^\sharp$ can be partially ordered by the point-wise extension of the order defined on their co-domain. Thus, we write $f \preceq^\sharp g$ to indicate that, for all $d \in D$, it holds $f(d) \preceq^\sharp g(d)$. Let $\langle P^\flat, \preceq^\flat \rangle$ be another poset. A function $f \colon P^\flat \to P^\sharp$ is called *monotonic* if, for each $x, y \in P^\flat$, $x \preceq^\flat y$ implies $f(x) \preceq^\sharp f(y)$; if $P^\flat$ and $P^\sharp$ are *cpo*'s, then f is *continuous* if, for every non-empty chain $X \subseteq P^\flat$, it holds $f(\operatorname{lub}^\flat X) = \operatorname{lub}^\sharp f(X)$; if $P^\flat$ and $P^\sharp$ are lattices, then f is *additive* if the previous condition holds for arbitrary non-empty subsets of $P^\flat$. The monotonic function $f \colon P^\flat \to P^\sharp$ is an *order isomorphism* (or, more simply, isomorphism) if it is also bijective. When the partial orders are clear from the context, the existence of an isomorphism between the two posets $\langle P^\flat, \preceq^\flat \rangle$ and $\langle P^\sharp, \preceq^\sharp \rangle$ is denoted by $P^\flat \equiv P^\sharp$.

A *Galois connection* is a pair of monotonic functions $\alpha \colon P^\flat \to P^\sharp$ and $\gamma \colon P^\sharp \to P^\flat$ such that

$$\forall x^\flat \in P^\flat : x^\flat \preceq^\flat \gamma\bigl(\alpha(x^\flat)\bigr), \qquad\qquad \forall x^\sharp \in P^\sharp : \alpha\bigl(\gamma(x^\sharp)\bigr) \preceq^\sharp x^\sharp.$$

The functions α and γ are said to be the lower adjoint and the upper adjoint, respectively. A *Galois insertion* is a Galois connection where $(\alpha \circ \gamma)\colon P^\sharp \to P^\sharp$ is the identity function.

A monotonic self-map $f\colon P \to P$ over a poset $\langle P, \preceq \rangle$ is *idempotent* if, for each $x \in P$, $f(x) = f\bigl(f(x)\bigr)$. A monotonic and idempotent self-map $\rho\colon P \to P$ is an *upper closure operator* if it is also *extensive*, namely $x \preceq \rho(x)$ for each $x \in P$. A lower closure operator $\varrho\colon P \to P$ is defined dually, by requiring that $\varrho(x) \preceq x$ for each $x \in P$.

Let $\langle C, \preceq \rangle$ be a complete lattice. Each upper closure operator ρ over C is uniquely determined by the set of its fixpoints, that is, by its image

$$\rho(C) \stackrel{\mathrm{def}}{=} \bigl\{\, \rho(x) \bigm| x \in C \,\bigr\}.$$

We will often denote upper closure operators by their images. The set of all upper closure operators over the complete lattice C, denoted by $\mathrm{uco}(C)$, forms itself a complete lattice ordered as follows: if $\rho_1, \rho_2 \in \mathrm{uco}(P)$, $\rho_1 \sqsubseteq \rho_2$ if and only if $\rho_2(C) \subseteq \rho_1(C)$. This partial order corresponds to the point-wise extension of the partial order defined on C.

For a detailed introduction to closure operators, the reader is referred to [GHK+80].

2.2 Abstract Interpretation

The theory of abstract interpretation has been developed by Patrick and Radhia Cousot [CC77a, CC77b, CC79, CC92a, CC92b]. Using their own words [CC92a],

> "Abstract interpretation is a method for designing approximate semantics of programs which can be used to gather information about programs in order to provide sound answers to questions about their run-time behaviours."

Therefore, static analysis is just one of the many possible applications of abstract interpretation theory. Other applications include, for instance, the formal comparison between different concrete semantics and the design of semi-automatic proof methods, where the conditions on the termination of the abstract semantics computation can be relaxed.

In all cases, we are provided with a set of concrete properties C, whose elements represent the possible behaviors of programs, and a set of abstract properties A, whose elements are "non-standard" descriptions of the concrete ones. These two sets are related by a *soundness relation* $\propto \subseteq A \times C$. Intuitively, $a \propto c$ means that any conclusion we derive by looking at the abstract description $a \in A$ corresponds to a valid observation that we could have done by looking at the concrete description $c \in C$.

A concrete semantics function $[\![\cdot]\!]$ associates each program P to its corresponding (concrete) meaning $[\![P]\!] \in C$. One of the goals of abstract interpretation is to define a *correct* abstract semantics function $[\![\cdot]\!]^\sharp$, mapping each program P into an abstract description $[\![P]\!]^\sharp \in A$ such that $[\![P]\!]^\sharp \propto [\![P]\!]$.

The traditional abstract interpretation framework is based on the existence of a Galois connection between the concrete and the abstract domains, which are both formalized as complete lattices. In such a context, the soundness relation is defined, for each $a \in A$ and

$c \in C$, by

$$a \propto c \quad \overset{\text{def}}{\Longleftrightarrow} \quad \alpha(c) \preceq_A a \quad \overset{\text{def}}{\Longleftrightarrow} \quad c \preceq_C \gamma(a).$$

Thus, the partial orders $\preceq_C$ and $\preceq_A$ can both be regarded as *approximation relations*, formalizing the intuitive notion that an element is more precise than another one. The functions $\alpha \colon C \to A$ and $\gamma \colon A \to C$ providing the Galois connection are called, respectively, the *abstraction* and the *concretization* function. The abstraction function α maps each concrete element $c \in C$ into its best possible approximation $\alpha(c) \in A$; in turn, we can say that the meaning of an abstract element $a \in A$ is precisely described by its concretization $\gamma(a) \in C$. Note that, by a standard result on Galois connection, the abstraction function uniquely defines the corresponding concretization function, and vice versa:

$$\begin{aligned} \forall a \in A : \gamma(a) &= \mathrm{lub}_C\{\, c \in C \mid \alpha(c) \preceq_A a \,\}; \\ \forall c \in C : \alpha(c) &= \mathrm{glb}_A\{\, a \in A \mid c \preceq_C \gamma(a) \,\}. \end{aligned}$$

Also note that any additive function $\alpha \colon C \to A$ always induces a Galois connection between the concrete domain C and the abstract domain A.

It is almost always the case that the concrete semantics function $[\![\cdot]\!]$ is defined as the least fixpoint of an operator $f_C \colon C \to C$; in its turn, the operator f_C can be specified modularly, as the composition of a given set of simpler concrete operators. A well-known and very useful result of abstract interpretation theory is that correctness is preserved by function composition. Therefore, it is possible to obtain a correct approximation of the concrete semantics function $[\![\cdot]\!]$ by providing a correct approximation f_A for each of the basic concrete operators. The correctness of the abstract computation can thus be specified as one of the following two equivalent conditions: for all $c \in C$ and $a \in A$,

$$\alpha(c) \preceq_A a \quad \Longrightarrow \quad (\alpha \circ f_C)(c) \preceq_A f_A(a); \tag{2.1}$$

$$c \preceq_C \gamma(a) \quad \Longrightarrow \quad f_C(c) \preceq_C (\gamma \circ f_A)(a). \tag{2.2}$$

In the Galois connection framework it is always possible to specify the *optimal* abstract semantics operator

$$f_A^\sharp \stackrel{\text{def}}{=} \alpha \circ f_C \circ \gamma, \tag{2.3}$$

corresponding to the best correct approximation, on the abstract domain A, of the concrete operator f_C. Note however that, in general, optimality results are not preserved by the composition of abstract semantics functions.

The function $\rho \stackrel{\text{def}}{=} \gamma \circ \alpha$ is an upper closure operator on the concrete domain C; similarly, $\varrho \stackrel{\text{def}}{=} \alpha \circ \gamma$ is a lower closure operator on the abstract domain A and we have $\rho(C) \equiv \varrho(A)$, where the isomorphism is provided by the functions α and γ. When dealing with a Galois insertion, ϱ is the identity function on A, so that $\rho(C) \equiv A$. The presentation of abstract interpretation in terms of Galois connections can be rephrased by using uco's. In particular, the partial order $\sqsubseteq$ defined on $\mathrm{uco}(C)$ formalizes the intuition of an abstract domain being more precise than another one.

Note that other frameworks, where the correspondence between the concrete and abstract domains is weaker than a Galois connection, have been considered in [CC92a].

Besides the above fundamental concepts and results, an interesting line of research in abstract interpretation theory studies how new abstract domains can be systematically derived from known ones [CC79]. The typical example is the *reduced product* operator between abstract domains: given two abstract domains, it returns the smaller (i.e., most abstract) domain which is as precise as their conjunctive combination. An elegant characterization of the reduced product can be obtained when adopting the upper closure operator approach: given two elements $\rho_1, \rho_2 \in \mathrm{uco}(C)$, their reduced product, denoted $\rho_1 \sqcap \rho_2$, is their glb on $\mathrm{uco}(C)$.

Another well-known example is the *disjunctive completion* operator, returning the minimal domain encoding all possible disjunctions of the given domain elements. The *complementation* operator of [CFG⁺97] can be interpreted as the reverse of the reduced product and it can be used to decompose an abstract domain into a set of minimal factors. Similarly, the *least disjunctive base* operator [GR96] can be seen as the inverse of the disjunctive completion operator. The *quotient* operator [CFW98] and the *least fully-complete extension* [GR97] can be used to compute the simplest abstract domain being as precise as a given reference domain with respect to a given observable property. Finally, the *Heyting completion* operator [GS97] takes an abstract domain encoding some properties and enriches it by adding new abstract elements representing relational dependencies between those properties.

2.3 Logic Programming

This thesis is not primarily about logic programming and it is assumed that the reader is familiar with the concepts and techniques of the, by now standard, concrete semantics constructions [Apt90, FLMP89, FLMP93, Llo87], how these can be extended for logic languages with constraints [JLM87, JM94] and, in particular, to logic languages computing on a domain of rational trees [Col82, Col84]. In the following, we will only review the basic concepts that are strictly necessary for the presentation and development of the formal results contained in this thesis.

2.3.1 Terms

Let *Sig* denote a possibly infinite set of function symbols, ranked over the set of natural numbers. Let *Vars* denote a denumerable set of variables, disjoint from *Sig*. Then *Terms* denotes the free algebra of all (possibly infinite) terms in the signature *Sig* having variables in *Vars*. Thus a term can be seen as an ordered labeled tree, possibly having some infinite paths and possibly containing variables: every inner node is labeled with a function symbol in *Sig* with a rank matching the number of the node's immediate descendants, whereas every leaf is labeled by either a variable in *Vars* or a function symbol in *Sig* having rank 0 (a constant). It is assumed that *Sig* contains at least two distinct function symbols, one having rank 0 (so that there exist finite terms having no variables) and one having rank greater than 0 (so that there exist infinite terms).

If $t \in \mathit{Terms}$ then $\mathrm{vars}(t)$ and $\mathrm{mvars}(t)$ denote the set and the multiset of variables occurring in t, respectively. By a slight abuse of notation, we will also write $\mathrm{vars}(o)$ to denote the set of variables occurring in the syntactic representation of an arbitrary object o. To prove a few of the results of this thesis, it is useful to assume a total ordering, denoted with '$\leq$', on Vars.

Suppose $s, t \in \mathit{Terms}$: s and t are *independent* if $\mathrm{vars}(s) \cap \mathrm{vars}(t) = \varnothing$; if $y \in \mathrm{vars}(t)$ and $\neg\bigl(y \Subset \mathrm{mvars}(t)\bigr)$ we say that variable y *occurs linearly in* t, more briefly written using the predication $\mathrm{occ\_lin}(y, t)$; t is said to be *ground* if $\mathrm{vars}(t) = \varnothing$; t is *free* if $t \in \mathit{Vars}$; t is *linear* if, for all $y \in \mathrm{vars}(t)$, we have $\mathrm{occ\_lin}(y, t)$; finally, t is a *finite term* (or *Herbrand term*) if it contains a finite number of occurrences of function symbols. The sets of all ground, linear and finite terms are denoted by GTerms, LTerms and HTerms, respectively.

The function $\mathrm{size}\colon \mathit{HTerms} \to \mathbb{N}$, for each $t \in \mathit{HTerms}$, is defined by

$$\mathrm{size}(t) \stackrel{\mathrm{def}}{=} \begin{cases} 1, & \text{if } t \in \mathit{Vars}; \\ 1 + \sum_{i=1}^{n} \mathrm{size}(t_i), & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

A path $p \in \bigl(\mathbb{N} \setminus \{0\}\bigr)^\star$ is any finite sequence of (non-zero) natural numbers. Given a path p and a (possibly infinite) term $t \in \mathit{Terms}$, we denote by $t[p]$ the subterm of t found by following path p. Formally,

$$t[p] = \begin{cases} t & \text{if } p = \epsilon; \\ t_i[q] & \text{if } p = i \,.\, q \wedge (1 \leq i \leq n) \wedge t = f(t_1, \ldots, t_n). \end{cases}$$

Note that $t[p]$ is only defined for those paths p actually corresponding to subterms of t.

2.3.2 Substitutions

A *substitution* is a total function $\sigma\colon \mathit{Vars} \to \mathit{HTerms}$ that is the identity almost everywhere; in other words, the *domain* of σ,

$$\mathrm{dom}(\sigma) \stackrel{\mathrm{def}}{=} \{\, x \in \mathit{Vars} \mid \sigma(x) \neq x \,\},$$

is finite. Given a substitution $\sigma\colon \mathit{Vars} \to \mathit{HTerms}$, we overload the symbol 'σ' so as to denote also the function $\sigma\colon \mathit{HTerms} \to \mathit{HTerms}$ defined as follows, for each term $t \in \mathit{HTerms}$:

$$\sigma(t) \stackrel{\mathrm{def}}{=} \begin{cases} t, & \text{if } t \text{ is a constant symbol;} \\ \sigma(t), & \text{if } t \in \mathit{Vars}; \\ f\bigl(\sigma(t_1), \ldots, \sigma(t_n)\bigr), & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

If $t \in \mathit{HTerms}$, we write $t\sigma$ to denote $\sigma(t)$. Note that, for each substitution σ and each finite term $t \in \mathit{HTerms}$, it holds $\mathrm{size}(t) \leq \mathrm{size}(t\sigma)$.

If $x \in \mathit{Vars}$ and $t \in \mathit{HTerms} \setminus \{x\}$, then $x \mapsto t$ is called a *binding*. The set of all bindings is denoted by *Bind*. Substitutions are denoted by the set of their bindings, thus a substitution σ is identified with the (finite) set

$$\{\, x \mapsto \sigma(x) \mid x \in \mathrm{dom}(\sigma) \,\}.$$

We denote by $\mathrm{vars}(\sigma)$ the set of variables occurring in the bindings of σ. We also define the sets $\mathrm{param}(\sigma)$ and $\mathrm{range}(\sigma)$ (the *parameter variables* and the *range variables* of σ, respectively) as

$$\mathrm{param}(\sigma) \stackrel{\mathrm{def}}{=} \mathrm{vars}(\sigma) \setminus \mathrm{dom}(\sigma),$$
$$\mathrm{range}(\sigma) \stackrel{\mathrm{def}}{=} \{\, y \in \mathrm{vars}(t) \mid (x \mapsto t) \in \sigma \,\}.$$

A substitution is said to be *circular* if, for $n > 1$, it has the form

$$\{x_1 \mapsto x_2, \ldots, x_{n-1} \mapsto x_n, x_n \mapsto x_1\},$$

where $x_1, \ldots, x_n$ are distinct variables. A substitution is in *rational solved form* if it has no circular subset. The set of all substitutions in rational solved form is denoted by *RSubst*. A substitution σ is *idempotent* if, for all $t \in \mathit{Terms}$, we have $t\sigma\sigma = t\sigma$. Equivalently, σ is idempotent if and only if $\mathrm{dom}(\sigma) \cap \mathrm{range}(\sigma) = \varnothing$. The set of all idempotent substitutions is denoted by *ISubst* and $\mathit{ISubst} \subset \mathit{RSubst}$.

Example 2.1 *Let x, y and z be distinct variables and $a \in \mathit{Sig}$, with rank 0, be a term constant. The following hold:*

$$\begin{aligned}
\{x \mapsto y, y \mapsto a\} &\in \mathit{RSubst} \setminus \mathit{ISubst}, \\
\{x \mapsto a, y \mapsto a\} &\in \mathit{ISubst}, \\
\{x \mapsto y, y \mapsto g(y)\} &\in \mathit{RSubst} \setminus \mathit{ISubst}, \\
\{x \mapsto y, y \mapsto g(x)\} &\in \mathit{RSubst} \setminus \mathit{ISubst}, \\
\{x \mapsto y, y \mapsto x\} &\notin \mathit{RSubst}, \\
\{x \mapsto y, y \mapsto x, z \mapsto a\} &\notin \mathit{RSubst}.
\end{aligned}$$

We have assumed that there is a total ordering '$\leq$' for *Vars*. We say that $\sigma \in \mathit{RSubst}$ is *ordered* (with respect to this ordering) if, for each binding $(x \mapsto y) \in \sigma$ such that $y \in \mathrm{param}(\sigma)$, we have $y < x$.

We will sometimes write $t[x/s]$ to denote $t\{x \mapsto s\}$.

The composition of substitutions is defined in the usual way. Thus $\tau \circ \sigma$ is the substitution such that, for all terms $t \in \mathit{HTerms}$, $(\tau \circ \sigma)(t) = \tau\bigl(\sigma(t)\bigr)$ and has the formulation

$$\tau \circ \sigma = \{\, x \mapsto x\sigma\tau \mid x \in \mathrm{dom}(\sigma), x \neq x\sigma\tau \,\} \cup \{\, x \mapsto x\tau \mid x \in \mathrm{dom}(\tau) \setminus \mathrm{dom}(\sigma) \,\}. \quad (2.4)$$

As usual, σ^0 denotes the identity function (i.e., the empty substitution) and, when $i > 0$, σ^i denotes the substitution $(\sigma \circ \sigma^{i-1})$.

For each $\sigma \in \mathit{RSubst}$, $s \in \mathit{HTerms}$, the sequence of finite terms $\sigma^0(s), \sigma^1(s), \sigma^2(s), \ldots$ converges to a (possibly infinite) term, denoted $\sigma^\infty(s)$ [IZ96, Kin00]. Therefore, the function $\mathrm{rt}\colon \mathit{HTerms} \times \mathit{RSubst} \to \mathit{Terms}$ such that

$$\mathrm{rt}(s, \sigma) \stackrel{\mathrm{def}}{=} \sigma^\infty(s)$$

is well defined. Note that, in general, this function is not a substitution: while having a finite domain, its "bindings" $x \mapsto \mathrm{rt}(x, \sigma)$ can map a domain variable x into a term $\mathrm{rt}(x, \sigma) \in \mathit{Terms} \setminus \mathit{HTerms}$. However, as the name of the function suggests, the term $\mathrm{rt}(x, \sigma)$ is granted to be *rational*, meaning that it can only have a finite number of distinct subterms. Rational terms, even though infinite in the sense that they admit paths of infinite length, can be finitely represented.

2.3.3 Equality Theories

An *equation* is of the form $s = t$ where $s, t \in \mathit{HTerms}$. Eqs denotes the set of all equations. A substitution σ may be regarded as the finite set of equations $\{\, x = t \mid (x \mapsto t) \in \sigma \,\}$. We say that a set of equations e is in *rational solved form* if $\{\, s \mapsto t \mid (s = t) \in e \,\} \in \mathit{RSubst}$. In the rest of the paper, we will often write a substitution $\sigma \in \mathit{RSubst}$ to denote a set of equations in rational solved form (and vice versa).

As is common in research work involving equality, we overload the symbol '=' and use it to denote both equality and to represent syntactic identity. The context makes it clear what is intended.

Let $\{r, s, t, s_1, \ldots, s_n, t_1, \ldots, t_n\} \subseteq \mathit{HTerms}$ and $f \in \mathit{Sig}$ with rank n. We assume that any equality theory T over Terms includes the *congruence axioms* denoted by the following schemata:

$$s = s, \tag{2.5}$$
$$s = t \leftrightarrow t = s, \tag{2.6}$$
$$r = s \land s = t \to r = t, \tag{2.7}$$
$$s_1 = t_1 \land \cdots \land s_n = t_n \to f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n). \tag{2.8}$$

In logic programming and most implementations of Prolog it is usual to assume an equality theory based on syntactic identity. This consists of the congruence axioms together with the *identity axioms* denoted by the following schemata, where $f, g \in \mathit{Sig}$ with ranks n and m, respectively, are such that f and g are distinct function symbols or $n \neq m$:

$$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \to s_1 = t_1 \land \cdots \land s_n = t_n, \tag{2.9}$$
$$\neg\bigl(f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)\bigr). \tag{2.10}$$

The axioms characterized by schemata (2.9) and (2.10) ensure the equality theory depends only on the syntax. The equality theory for a non-syntactic domain replaces these axioms by ones that depend instead on the semantics of the domain and, in particular, on the interpretation given to functor symbols.

The equality theory of Clark [Cla78], denoted $\mathcal{FT}$, on which pure logic programming is based, usually called the *Herbrand* equality theory, is given by the congruence axioms, the identity axioms, and the axiom schema

$$\forall z \in \mathit{Vars} : \forall t \in (\mathit{HTerms} \setminus \mathit{Vars}) : z \in \operatorname{vars}(t) \rightarrow \neg(z = t). \tag{2.11}$$

Axioms characterized by the schema (2.11) are called the *occurs-check axioms* and are an essential part of the standard unification procedure in SLD-resolution.

An alternative approach used in some implementations of logic programming systems, such as Prolog II, SICStus and Oz, does not require the occurs-check axioms. This approach is based on the theory of rational trees [Col82, Col84], denoted $\mathcal{RT}$. It assumes the congruence axioms and the identity axioms together with a *uniqueness axiom* for each substitution in rational solved form. Informally speaking these state that, after assigning a ground rational tree to each parameter variable, the substitution uniquely defines a ground rational tree for each of its domain variables. Note that being in rational solved form is a very weak property. Indeed, unification algorithms returning a set of equations in rational solved form are allowed to be much lazier than one would usually expect (e.g., see the first substitution in Example 2.1). We refer the interested reader to [JLM87, Kei94, Mah88] for details on the subject.

In the sequel we will use the expression "equality theory" to denote any consistent, decidable theory T satisfying the congruence axioms. We will also use the expression "syntactic equality theory" to denote any equality theory T also satisfying the identity axioms. Sometimes, when the equality theory T is clear from the context, it is convenient to adopt the notations $\sigma \implies \tau$ and $\sigma \iff \tau$, where σ, τ are sets of equations, to denote $T \vdash \forall(\sigma \rightarrow \tau)$ and $T \vdash \forall(\sigma \leftrightarrow \tau)$, respectively.

Given an equality theory T, and a set of equations in rational solved form σ, we say that σ is *satisfiable* in T if $T \vdash \forall \mathit{Vars} \setminus \operatorname{dom}(\sigma) : \exists \operatorname{dom}(\sigma) \mathrel{.} \sigma$. If T is a syntactic equality theory that also includes the occurs-check axioms, and σ is satisfiable in T, then we say that σ is *Herbrand*.

Given a satisfiable set of equations $e \in \wp_{\mathrm{f}}(\mathit{Eqs})$ in an equality theory T, then a substitution $\sigma \in \mathit{RSubst}$ is called a *solution for e in T* if σ is satisfiable in T and $T \vdash \forall(\sigma \rightarrow e)$. If $\operatorname{vars}(\sigma) \subseteq \operatorname{vars}(e)$, then σ is said to be a *relevant* solution for e. In addition, σ is a *most general solution for e in T* if $T \vdash \forall(\sigma \leftrightarrow e)$. In this thesis, a most general solution is always a relevant solution of e. When the theory T is clear from the context, the set of all the relevant most general solutions for e in T is denoted by $\operatorname{mgs}(e)$.

Given an equality theory T, a set of equations in rational solved form may not be satisfiable in T. For example, $\exists x \mathrel{.} \bigl\{x = f(x)\bigr\}$ is false in the equality theory $\mathcal{FT}$.

2.4 The Concrete Domain

In this thesis, we are interested in finding approximate information about the variable sharing occurring in the possibly infinite set of solutions computed by logic programs. Therefore, it is natural to consider a concrete domain whose elements contains a set of substitutions in rational solved form. In addition, to give a meaning to the concrete description, we require a knowledge of the finite set of *variables of interest.* In the Ph.D. thesis of Langen [Lan90] a set of variables of interest is implicitly defined, for each program clause being analyzed, as the finite set of variables occurring in that clause. In this thesis we follow the clearer approach introduced in [CFW94, CFW98] and also adopted in [BHZ97, BHZ02, CF99], where the set of variables of interest is provided as an explicit component of the concrete domain.

Definition 2.2 (The concrete domain.) *The concrete domain is the complete lattice*

$$\mathcal{D}^\flat \stackrel{\mathrm{def}}{=} \bigl(\wp(RSubst) \times \wp_\mathrm{f}(Vars)\bigr) \cup \{\bot^\flat, \top^\flat\},$$

ordered by '$\leq^\flat$' defined as follows: for each $d \in \mathcal{D}^\flat$, $\bot^\flat \leq^\flat d$ and $d \leq^\flat \top^\flat$; for each $(\Sigma_1, VI_1), (\Sigma_2, VI_2) \in \mathcal{D}^\flat$:

$$(\Sigma_1, VI_1) \leq^\flat (\Sigma_2, VI_2) \qquad \iff \qquad (VI_1 = VI_2) \wedge (\Sigma_1 \subseteq \Sigma_2).$$

The induced least upper bound, denoted 'lub' *corresponds to the* 'merge-over-all-paths' *operator* [CC77a]. *Note that the only reason for including the distinguished elements $\bot^\flat$ and $\top^\flat$ is to obtain a complete lattice structure. Since they are never used in actual computations, all the remaining operations will not be explicitly specified for these distinguished elements.*

The operation unify: $\mathcal{D}^\flat \times RSubst \to \mathcal{D}^\flat$ *first extends the concrete element it takes as an argument to the set of variables occurring in the substitution it is given as the second argument; then it computes unification in the context of the syntactic equality theory T. For each $(\Sigma, VI) \in \mathcal{D}^\flat$ and $\mu \in RSubst$, where* $\mathrm{vars}(\Sigma) \cap \mathrm{vars}(\mu) \subseteq VI$, *we have*

$$\mathrm{unify}\bigl((\Sigma, VI), \mu\bigr) \stackrel{\mathrm{def}}{=} \Bigl(\bigcup\{\, \mathrm{mgs}(\sigma \cup \mu) \mid \sigma \in \Sigma \,\}, VI \cup \mathrm{vars}(\mu)\Bigr).$$

The existential quantification of $W \in \wp_\mathrm{f}(Vars)$ in $(\Sigma, VI) \in \mathcal{D}^\flat$ is defined as follows:

$$\exists W \,.\, (\Sigma, VI) \stackrel{\mathrm{def}}{=} \left\{ \sigma' \in RSubst \;\middle|\; \begin{array}{l} \sigma \in \Sigma, \quad \Delta = \mathrm{vars}(\sigma) \setminus VI, \\ T \vdash \forall\bigl(\exists \Delta \,.\, (\sigma' \leftrightarrow \exists W \,.\, \sigma)\bigr) \end{array} \right\}.$$

The operation of projecting $(\Sigma, VI) \in \mathcal{D}^\flat$ on a new set $W \in \wp_\mathrm{f}(Vars)$ of variables of interest is defined as follows:

$$\mathrm{proj}\bigl((\Sigma, VI), W\bigr) \stackrel{\mathrm{def}}{=} \bigl(\exists(VI \setminus W) \,.\, (\Sigma, VI), W\bigr).$$

Note that, by a little abuse of terminology, our 'proj' operator not only projects, but also *extends* a concrete element to the new variables of interest.

If $(\Sigma, VI) \in \mathcal{D}^\flat$, then (Σ, VI) represents the possibly infinite set of first-order formulas

$$\left\{\, \exists \Delta \,.\, \sigma \;\middle|\; \sigma \in \Sigma,\, \Delta = \mathrm{vars}(\sigma) \setminus VI \,\right\}$$

where σ is interpreted as the logical conjunction of the equations corresponding to its bindings. Concrete domains for constraint logic languages would be similar. If the analyzed language allows the use of constraints on various domains to restrict the values of the variable leaves of rational trees, the corresponding concrete domain would have one or more extra components to account for the constraints (see [BHZ00] for an example).

To better highlight the role of the set VI of variables of interests, consider the concrete element

$$d \stackrel{\mathrm{def}}{=} \bigl(\{\sigma\}, VI\bigr),$$

where $\sigma = \bigl\{x \mapsto f(y)\bigr\}$. Then, if $VI = \{x, y\}$ we have that $d \in \mathcal{D}^\flat$ expresses a dependency between the variables x and y. In contrast, if $VI = \{x\}$ then the concrete element d only constrains x. The same concept can be expressed by saying that in the first case the variable name 'y' matters, but it does not in the second case.

This example shows that the set of variables of interest is crucial for defining the meaning of the concrete (and, as we will see, also the abstract) descriptions. During the computation process, this set expands (e.g., when solving the body of the clause) and contracts (e.g., when the computed substitutions are projected onto the variables occurring in the head of the clause). This technique has two advantages: first, a clear and unambiguous description of those semantic operators that modify the set of variables of interest can be provided; second, the definition of the concrete (and, hence, the abstract) domain is completely independent from any particular program. Note that other solutions are possible; we refer the interested reader to [CFW96, Section 7] and [Sco02, Section 10], where this problem is discussed in the context of groundness analysis.

For the abstract interpretation of (constraint) logic languages, several general frameworks [BGL93, Bru91, GDL95, JS87, Mel87, MS90, MSJ94] have been proposed which, although independent from the actual abstract domain, differ for the particular type of semantics construction: operational, denotational, or model-theoretic based; top-down or bottom-up; based on a concrete domain of syntactic substitutions or on a generic constraint domain. The common recipe, borrowed from the general theory of abstract interpretation, is that the concrete semantics is constructed by using a minimal set of concrete operators, which are then correctly approximated on the abstract domain. Using these, the construction of a correct abstract semantics can be automatically completed.

In this thesis, we adopt a bottom-up semantics construction which is very similar to the one proposed in [BGL93]. The semantics provides information about the success patterns of the analyzed logic program; information on the call-patterns can be derived by suitable transforming this program [DR94]. The concrete semantics is obtained by iteration of a

T_P-like operator defined in terms of 'lub', 'unify' and 'proj' (the other operations needed, such as the consistent renaming apart of concrete elements, are very simple). It is worth noting that the considered concrete operators given in Definition 2.2 are additive, therefore allowing a simplification of the statements of correctness. In particular, for the unification operator, any correctness results for *sets* of substitutions is a direct consequence of the corresponding correctness result for a single substitution.

2.5 Boolean Functions

Boolean functions have been extensively used for data-flow analysis of logic-based languages and, in particular, for groundness analysis.

Definition 2.3 (Boolean functions.) *Let $VI \in \wp_{\mathrm{f}}(Vars)$ and $Bool \stackrel{\mathrm{def}}{=} \{0,1\}$. The set of* Boolean functions over VI *is*

$$Bfun \stackrel{\mathrm{def}}{=} \wp(VI) \to Bool.$$

The set $m \in \wp(VI)$ is a model *of $\phi \in Bfun$ if and only if $\phi(m) = 1$. We denote by $[\phi]_{VI}$ the set of all the models of $\phi \in Bfun$. The set $Bfun$ is partially ordered by the relation $\models$ where, for each $\phi, \psi \in Bfun$, we have $\phi \models \psi$ if and only if $[\phi]_{VI} \subseteq [\psi]_{VI}$.*

Boolean functions are inductively constructed from the elementary functions corresponding to variables by means of the usual logical connectives. Thus, if $x \in VI$, we write x to denote the Boolean function identified by the set of models $[x]_{VI} = \{\, m \subseteq VI \mid x \in m \,\}$. Similarly, if $\phi, \psi \in Bfun$, then we write $\neg\phi$ and $\phi \wedge \psi$ to denote the Boolean functions identified by $[\neg\phi]_{VI} = \wp(VI) \setminus [\phi]_{VI}$ and $[\phi \wedge \psi]_{VI} = [\phi]_{VI} \cap [\psi]_{VI}$, respectively. The other connectives are derived from the two above, as usual.

Two important classes of Boolean functions used for tracking groundness dependencies are *Pos* and *Def* [AMSS98].

Definition 2.4 (*Pos* and *Def*). *The domain $Pos \subset Bfun$ of* positive *Boolean functions is defined as the set of functions having VI as a model; formally,*

$$Pos \stackrel{\mathrm{def}}{=} \{\, \phi \in Bfun \bigm| \phi(VI) = 1 \,\}.$$

The domain $Def \subseteq Pos$ of definite *Boolean functions is defined as the subset of those positive Boolean functions whose models are closed under set-intersection; formally,*

$$Def \stackrel{\mathrm{def}}{=} \{\, \phi \in Pos \bigm| m_1, m_2 \in [\phi]_{VI} \implies (m_1 \cap m_2) \in [\phi]_{VI} \}.$$

Therefore, by letting $VI = \{x, y\}$, we obtain that $(x \to y) \in Def$, $(x \vee y) \in Pos \setminus Def$ and $(\neg x) \in Bfun \setminus Pos$.

Chapter 3

Set-Sharing

In this chapter we introduce the set-sharing domain of Jacobs and Langen and the corresponding abstraction function defined on idempotent substitutions. After showing that this abstraction function is inadequate for rational-tree languages, we address the problem of a sound and precise approximation of the set-sharing information contained in a substitution in rational solved form. By introducing the new concept of variable idempotent substitutions, we provide a generalization of this abstraction function that can be applied to any logic language computing on domains of syntactic structures, with or without the occurs-check. Results for correctness, idempotence and commutativity for abstract unification using this abstraction function are proven.

> **Note:** this chapter is mainly based on the results of [HBZ98]; an extended version will appear in [HBZ02].

3.1 The Set-Sharing Lattice

In this section, we first recall the definition of the set-sharing domain and present the (classical) abstraction function used for dealing with idempotent substitutions. We will then give evidence for the problems arising when applying this abstraction function to the more general case of substitutions in rational solved form.

The set-sharing domain is due to Jacobs and Langen [JL89]. However, we use the definition as presented in [BHZ97] where the set of variables of interest is given explicitly. An element of the domain (a *sharing set*) is thus a set of subsets of the variables of interest (the *sharing groups*). Even though the literature on set-sharing is almost unanimous in defining sharing sets so that they always contain the empty set, we deviate from this *de facto* standard: in our approach sharing sets *never* contain the empty set. We do this because the definitions turn out to be easier and, moreover, they describe the implementation (where the empty set never appears in sharing sets) more faithfully. Clearly, no problem can arise from a coherent omission/inclusion of the empty set, since the two corresponding domain representations are isomorphic.

Definition 3.1 (The *set-sharing* lattice.) *The domain SH is given by*

$$SH \stackrel{\text{def}}{=} \wp(SG),$$

where the set of sharing groups *SG is given by*

$$SG \stackrel{\text{def}}{=} \wp_{\mathrm{f}}(\mathit{Vars}) \setminus \{\varnothing\}.$$

The set-sharing lattice *is given by the set*

$$SS \stackrel{\text{def}}{=} \bigl\{\, (sh, \mathit{VI}) \bigm| sh \in SH, \mathit{VI} \in \wp_{\mathrm{f}}(\mathit{Vars}), \forall S \in sh : S \subseteq \mathit{VI} \,\bigr\} \cup \{\bot, \top\},$$

ordered by $\preceq_{SS}$ defined as follows, for each $d, (sh_1, \mathit{VI}_1), (sh_2, \mathit{VI}_2) \in SS$:

$$\begin{aligned} \bot &\preceq_{SS} d, \\ d &\preceq_{SS} \top, \\ (sh_1, \mathit{VI}_1) \preceq_{SS} (sh_2, \mathit{VI}_2) \quad &\iff \quad (\mathit{VI}_1 = \mathit{VI}_2) \wedge (sh_1 \subseteq sh_2). \end{aligned}$$

It is straightforward to see that every subset of SS has a least upper bound with respect to $\preceq_{SS}$. Hence SS is a complete lattice (as a matter of fact, the only reason we have $\top \in SS$ is in order to turn SS into a lattice rather than a *cpo*). The lub operator over SS will be denoted by Alub.

3.1.1 The Classical Abstraction Function for *ISubst*

An element sh of SH encodes the set-sharing information contained in an idempotent substitution σ. Namely, each sharing group $S \in SG$ in the sharing set sh corresponds to one or more variables which are shared by all and only the variables occurring in S.

Definition 3.2 (Classical sg and abstraction functions.) *The* sharing group function, $\mathrm{sg}\colon \mathit{ISubst} \times \mathit{Vars} \to \wp_{\mathrm{f}}(\mathit{Vars})$ *is defined, for each $\sigma \in \mathit{ISubst}$ and each $v \in \mathit{Vars}$, by*

$$\mathrm{sg}(\sigma, v) \stackrel{\text{def}}{=} \bigl\{\, y \in \mathit{Vars} \bigm| v \in \mathrm{vars}(y\sigma) \,\bigr\}.$$

The concrete domain $\mathcal{D}_I^\flat \stackrel{\text{def}}{=} \wp(\mathit{ISubst}) \times \wp_{\mathrm{f}}(\mathit{Vars})$ is related to SS by means of the abstraction function $\alpha_I\colon \mathcal{D}_I^\flat \to SS$. *For each $\Sigma \in \wp(\mathit{ISubst})$ and each $\mathit{VI} \in \wp_{\mathrm{f}}(\mathit{Vars})$,*

$$\alpha_I\bigl((\Sigma, \mathit{VI})\bigr) \stackrel{\text{def}}{=} \mathrm{Alub}\bigl\{\, \alpha_I(\sigma, \mathit{VI}) \bigm| \sigma \in \Sigma \,\bigr\},$$

where $\alpha_I\colon \mathit{ISubst} \times \wp_{\mathrm{f}}(\mathit{Vars}) \to SS$ is defined, for each substitution $\sigma \in \mathit{ISubst}$ and each $\mathit{VI} \in \wp_{\mathrm{f}}(\mathit{Vars})$, by

$$\alpha_I(\sigma, \mathit{VI}) \stackrel{\text{def}}{=} \Bigl(\bigl\{\, \mathrm{sg}(\sigma, v) \cap \mathit{VI} \bigm| v \in \mathit{Vars} \,\bigr\} \setminus \{\varnothing\}, \mathit{VI}\Bigr).$$

The sharing group function 'sg' was first defined by Jacobs and Langen [JL89] and used in their definition of a concretization function for SH. The function α_I corresponds closely to the abstract counterpart of this concretization function, but explicitly includes the set of variables of interest as a separate argument. It is identical to the abstraction function for set-sharing defined by Cortesi and Filé [CF99].

In order to provide an intuitive reading of the sharing information encoded into an abstract element, it is worth stressing once again that the analysis aims at capturing *possible* sharing. The corresponding *definite* information (e.g., definite groundness or independence) can be extracted by observing which sharing groups are *not* in the abstract element. As an example, if we observe that there is no sharing group containing a particular variable of VI, then we can safely conclude that this variable is definitely ground (namely, it is bound to a term containing no variables). Similarly, if we observe that two variables never occur together in the same sharing group, then we can safely conclude that they are independent (namely, they are bound to terms that do not share a common variable).

Example 3.3 *Assume* $VI = \{x_1, x_2, x_3, x_4\}$ *and let*

$$\sigma = \bigl\{x_1 \mapsto f(x_2, x_3), x_4 \mapsto a\bigr\},$$

so that its abstraction is given by

$$\alpha_I(\sigma, VI) = \Bigl(\bigl\{\{x_1, x_2\}, \{x_1, x_3\}\bigr\}, VI\Bigr).$$

From this abstraction we can safely conclude that variable x_4 *is ground and variables* x_2 *and* x_3 *are independent.*

3.1.2 Toward an Abstraction Function for *RSubst*

To help motivate the approach we will take in adapting the classical abstraction function to non-idempotent substitutions, we now explain some of the problems that arise if we apply α_I, as it is defined on *ISubst*, to the non-idempotent substitutions in *RSubst*. Note that these problems are only partially caused by non-Herbrand substitutions (that is substitutions that are not satisfiable in a syntactic equality theory containing the occurs-check axioms). They are also due to the presence of non-idempotent but Herbrand substitutions that may arise because of the potential "laziness" of unification procedures based on the rational solved form.

We use the following substitutions to illustrate the problems, where it is assumed that the set of variables of interest is $VI = \{x_1, x_2, x_3, x_4\}$. Let

$$\begin{aligned}
\sigma_1 &= \bigl\{x_1 \mapsto f(x_1)\bigr\},\\
\sigma_2 &= \bigl\{x_3 \mapsto x_4\bigr\},\\
\sigma_3 &= \bigl\{x_1 \mapsto x_2, x_2 \mapsto x_3, x_3 \mapsto x_4\bigr\},\\
\sigma_4 &= \bigl\{x_1 \mapsto x_4, x_2 \mapsto x_4, x_3 \mapsto x_4\bigr\}
\end{aligned}$$

so that we have

$$\begin{aligned}
\alpha_I(\varnothing, VI) = \alpha_I(\sigma_1, VI) &= \Bigl(\bigl\{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}\bigr\}, VI\Bigr), \\
\alpha_I(\sigma_2, VI) = \alpha_I(\sigma_3, VI) &= \Bigl(\bigl\{\{x_1\}, \{x_2\}, \{x_3, x_4\}\bigr\}, VI\Bigr), \\
\alpha_I(\sigma_4, VI) &= \Bigl(\bigl\{\{x_1, x_2, x_3, x_4\}\bigr\}, VI\Bigr).
\end{aligned}$$

The first problem is that the concrete equivalence classes induced by the classical abstraction function on *RSubst* are much coarser than one would expect and hence we have an unwanted loss of precision. For example, in all the sets of rational trees that are solutions for σ_1, the variable x_1 is ground (formally, $\mathrm{rt}(x_1, \sigma_1) \in \mathit{GTerms}$). However, the computed abstract element fails to distinguish this situation from that resulting from the empty substitution, where all the variables are free and un-aliased. Similarly, we have the same abstract element for both σ_2 and σ_3 although, x_1, x_2 and x_3 are independent in σ_2 only.

The second problem is quite the opposite from the first in that the abstraction function distinguishes between substitutions that are equivalent (with respect to any equality theory). For example, σ_3 and σ_4 are equivalent although the abstract elements are distinct. Note that the two problems described here are completely orthogonal although they can interact and produce more complex situations.

3.2 Variable-Idempotence

In this section we define a new class of substitutions based on the concept of *variable-idempotence*, a generalization of the well-known concept of idempotence. By considering variable-idempotent substitutions only, it is possible to provide a simple solution to the problems outlined in the previous section. Moreover, there is no loss of generality, since we will show an algorithm transforming any substitution in rational solved form to an equivalent (with respect to any equality theory) variable-idempotent substitution.

3.2.1 Variable-Idempotent Substitutions

Recall that, for substitutions, the definition of idempotence requires that repeated applications of a substitution do not change the syntactic structure of a term. However, a sharing abstraction such as α_I is only interested in the variables and not in the structure that contains them. Thus, an obvious way to relax the definition of idempotence to allow for a non-Herbrand substitution is to ignore the structure and just require that its repeated application leaves the set of variables in a term invariant.

Definition 3.4 (Variable-idempotence.) *A substitution $\sigma \in \mathit{RSubst}$ is* variable-idempotent *if and only if for all $t \in \mathit{HTerms}$ we have*

$$\mathrm{vars}(t\sigma\sigma) = \mathrm{vars}(t\sigma).$$

The set of variable-idempotent substitutions is denoted VSubst.

Note that any idempotent substitution is also variable-idempotent, so that we have the inclusions $\mathit{ISubst} \subset \mathit{VSubst} \subset \mathit{RSubst}$.

Example 3.5 *Consider the following substitutions which are all in RSubst.*

$$
\begin{aligned}
\sigma_1 &= \{x \mapsto f(x)\} && \in \mathit{VSubst} \setminus \mathit{ISubst},\\
\sigma_2 &= \{x \mapsto f(y), y \mapsto z\} && \notin \mathit{VSubst},\\
\sigma_3 &= \{x \mapsto f(z), y \mapsto z\} && \in \mathit{ISubst},\\
\sigma_4 &= \{x \mapsto z, y \mapsto f(x, y)\} && \notin \mathit{VSubst},\\
\sigma_5 &= \{x \mapsto z, y \mapsto f(z, y)\} && \in \mathit{VSubst} \setminus \mathit{ISubst}.
\end{aligned}
$$

Note that σ_2 is equivalent (with respect to any equality theory) to the idempotent substitution σ_3; and σ_4 is equivalent (with respect to any equality theory) to the substitution σ_5 which is variable-idempotent but not idempotent.

The next result provides an alternative characterization of variable-idempotence.

Lemma 3.6 *Let $\sigma \in \mathit{RSubst}$. Then*

$$\sigma \in \mathit{VSubst} \iff \forall (x \mapsto r) \in \sigma : \mathrm{vars}(r\sigma) = \mathrm{vars}(r).$$

Proof. Suppose first that $\sigma \in \mathit{VSubst}$ and let $(x \mapsto r) \in \sigma$. Then

$$\mathrm{vars}(x\sigma\sigma) = \mathrm{vars}(x\sigma)$$

and hence, $\mathrm{vars}(r\sigma) = \mathrm{vars}(r)$.

Next, suppose that for all $(x \mapsto r) \in \sigma$, $\mathrm{vars}(r\sigma) = \mathrm{vars}(r)$ and consider $t \in \mathit{HTerms}$. We will show that $\mathrm{vars}(t\sigma\sigma) = \mathrm{vars}(t\sigma)$ by induction on the size of t. If t is a constant or $t \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$, then the result follows from the fact that $t\sigma = t$. If $t \in \mathrm{dom}(\sigma)$, then there exists $(y \mapsto s) \in \sigma$ such that $t = y$, so that $t\sigma = s$. Thus, we have

$$\mathrm{vars}(t\sigma\sigma) = \mathrm{vars}(s\sigma) = \mathrm{vars}(s) = \mathrm{vars}(t\sigma).$$

Finally, if $t = f(t_1, \ldots, t_n)$, then by the inductive hypothesis $\mathrm{vars}(t_i\sigma\sigma) = \mathrm{vars}(t_i\sigma)$ for $i = 1, \ldots, n$. Therefore we have

$$\mathrm{vars}(t\sigma\sigma) = \bigcup_{i=1}^{n} \mathrm{vars}(t_i\sigma\sigma) = \bigcup_{i=1}^{n} \mathrm{vars}(t_i\sigma) = \mathrm{vars}(t\sigma).$$

Thus, by Definition 3.4, as $\sigma \in \mathit{RSubst}$, $\sigma \in \mathit{VSubst}$. $\square$

Note that, as a consequence of Lemma 3.6, any substitution consisting of a single binding is variable-idempotent. Note though that we cannot assume that every subset of a variable-idempotent substitution is variable-idempotent.

Example 3.7 *Let*

$$\begin{aligned}
\sigma_1 &= \{x_1 \mapsto a, x_2 \mapsto f(x_1, x_3, x_4), x_3 \mapsto f(x_1, x_3, x_4)\}, \\
\sigma_2 &= \{x_1 \mapsto a, x_2 \mapsto f(x_1, x_3, x_4)\}, \\
\sigma_3 &= \sigma_1 \setminus \sigma_2 = \{x_3 \mapsto f(x_1, x_3, x_4)\}.
\end{aligned}$$

It can be observed that $\sigma_1, \sigma_3 \in \mathit{VSubst}$. Also note that $\sigma_2 \notin \mathit{VSubst}$, because we have $x_1 \in \operatorname{vars}(x_2\sigma_2)$ but $x_1 \notin \operatorname{vars}(x_2\sigma_2\sigma_2)$.

On the other hand, a variable-idempotent substitution does enjoy the following useful property with respect to its subsets.

Lemma 3.8 *Let $\sigma \in \mathit{VSubst}$ and $t \in \mathit{HTerms}$. Then, for all $\sigma' \subseteq \sigma$,*

$$\operatorname{vars}(t\sigma\sigma') \setminus \operatorname{dom}(\sigma) = \operatorname{vars}(t\sigma) \setminus \operatorname{dom}(\sigma).$$

Proof. Since $\sigma' \subseteq \sigma$, the relation $\operatorname{vars}(t\sigma) \setminus \operatorname{dom}(\sigma) \subseteq \operatorname{vars}(t\sigma\sigma')$ is trivial.

To prove the opposite relation, suppose that $y \in \operatorname{vars}(t\sigma\sigma') \setminus \operatorname{dom}(\sigma)$. Then there exists $x \in \operatorname{vars}(t\sigma)$ such that $y \in \operatorname{vars}(x\sigma')$. Now, if $x \notin \operatorname{dom}(\sigma')$, then $x = y$ and $y \in \operatorname{vars}(t\sigma)$. Otherwise, if $x \in \operatorname{dom}(\sigma')$, then $x\sigma' = x\sigma$ so that $y \in \operatorname{vars}(t\sigma\sigma)$ and hence, as $\sigma \in \mathit{VSubst}$, $y \in \operatorname{vars}(t\sigma)$. $\square$

Example 3.9 *Considering again the substitutions defined in Example 3.7, it can be observed that, for all $t \in \mathit{HTerms}$,*

$$\begin{aligned}
\operatorname{vars}(t\sigma_1) \setminus \operatorname{dom}(\sigma_1) &= \operatorname{vars}(t\sigma_1\sigma_2) \setminus \operatorname{dom}(\sigma_1), \\
\operatorname{vars}(t\sigma_1) \setminus \operatorname{dom}(\sigma_1) &= \operatorname{vars}(t\sigma_1\sigma_3) \setminus \operatorname{dom}(\sigma_1).
\end{aligned}$$

The next result provides a sufficient condition for a variable-idempotent substitution so that all of its subsets are variable-idempotent too.

Lemma 3.10 *Let $\sigma \in \mathit{VSubst}$ be such that $y \in \operatorname{dom}(\sigma) \cap \operatorname{range}(\sigma)$ implies $y \in \operatorname{vars}(y\sigma)$. Then, for all $\sigma' \subseteq \sigma$, $\sigma' \in \mathit{VSubst}$.*

Proof. Let $(x \mapsto t) \in \sigma' \subseteq \sigma$. We will prove that $\operatorname{vars}(t\sigma') = \operatorname{vars}(t)$, so that the thesis will follow from Lemma 3.6.

To prove the first implication, let $y \in \operatorname{vars}(t\sigma')$, so that $y \in \operatorname{range}(\sigma)$. If it also holds $y \in \operatorname{dom}(\sigma)$, then by the hypothesis $y \in \operatorname{vars}(y\sigma)$, so that $y \in \operatorname{vars}(t\sigma)$. Otherwise, if $y \notin \operatorname{dom}(\sigma)$, then again $y \in \operatorname{vars}(t\sigma)$. Thus, in both cases, since $\sigma \in \mathit{VSubst}$, by Lemma 3.6 we obtain $y \in \operatorname{vars}(t)$.

To prove the other implication, let $y \in \operatorname{vars}(t)$, so that $y \in \operatorname{range}(\sigma)$. If $y \notin \operatorname{dom}(\sigma')$ then $y \in \operatorname{vars}(t\sigma')$. Otherwise, if $y \in \operatorname{dom}(\sigma')$, then we have $y \in \operatorname{dom}(\sigma) \cap \operatorname{range}(\sigma)$. Thus, by hypothesis, $y \in \operatorname{vars}(y\sigma)$. Since $y\sigma = y\sigma'$, we have $y \in \operatorname{vars}(y\sigma')$, so that $y \in \operatorname{vars}(t\sigma')$. $\square$

We now state two technical results that will be needed later. Note that, when proving these results at the end of this section, we require that the equality theory also satisfies the identity axioms. They show that equivalent, ordered, variable-idempotent substitutions have the same domain and bind the domain variables to terms with the same set of parameter variables.

Lemma 3.11 *Let $\tau, \sigma \in VSubst$ be ordered and satisfiable in the syntactic equality theory T and suppose $T \vdash \forall(\tau \to \sigma)$. Then $\mathrm{dom}(\sigma) \subseteq \mathrm{dom}(\tau)$.*

Lemma 3.12 *Let $\tau, \sigma \in VSubst$ be satisfiable in the syntactic equality theory T and suppose $T \vdash \forall(\tau \to \sigma)$. In addition, let $s, t \in HTerms$ be such that $T \vdash \forall\bigl(\tau \to (s = t)\bigr)$ and $v \in \mathrm{vars}(s) \setminus \mathrm{dom}(\tau)$. Then there exists a variable $z \in \mathrm{vars}(t\sigma) \setminus \mathrm{dom}(\sigma)$ such that $v \in \mathrm{vars}(z\tau)$.*

It is worth noting that, thanks to the above results, we know that one of the problems outlined in Section 3.1.2, the possible "laziness" of the unification algorithm, does not affect variable-idempotent substitutions.

3.2.2 $\mathcal{S}$-transformations

A useful property of variable-idempotent substitutions is that any substitution in rational solved form can be transformed to an equivalent (with respect to any equality theory) variable-idempotent one. Thus, any result obtained for variable-idempotent substitutions can be systematically generalized to arbitrary substitutions in *RSubst*.

Definition 3.13 ($\mathcal{S}$-steps.) *The relation $\overset{\mathcal{S}}{\longmapsto} \subseteq RSubst \times RSubst$, called $\mathcal{S}$-step, is defined by*

$$\frac{(x \mapsto t) \in \sigma \qquad (y \mapsto s) \in \sigma \qquad x \neq y}{\sigma \overset{\mathcal{S}}{\longmapsto} \bigl(\sigma \setminus \{y \mapsto s\}\bigr) \cup \{y \mapsto s[x/t]\}}.$$

If we have a finite sequence of $\mathcal{S}$-steps $\sigma_1 \overset{\mathcal{S}}{\longmapsto} \cdots \overset{\mathcal{S}}{\longmapsto} \sigma_n$ mapping σ_1 to σ_n, then we write $\sigma_1 \overset{\mathcal{S}}{\longmapsto}{}^* \sigma_n$ and say that σ_1 can be rewritten, by $\mathcal{S}$-transformation, to σ_n.

Example 3.14 *Let*

$$\sigma_0 = \bigl\{x_1 \mapsto f(x_2), x_2 \mapsto g(x_3, x_4), x_3 \mapsto x_1\bigr\}.$$

Observe that substitution σ_0 is not variable-idempotent, since we have $\mathrm{vars}(x_1\sigma_0) = \{x_2\}$ but $\mathrm{vars}(x_1\sigma_0\sigma_0) = \{x_3, x_4\}$. By considering all the bindings of the substitution, one at a time, and applying the corresponding $\mathcal{S}$-step to all the other bindings, we produce a new substitution σ_3.

$$\begin{aligned}
\sigma_0 &= \bigl\{\underline{x_1 \mapsto f(x_2)}, x_2 \mapsto g(x_3, x_4), x_3 \mapsto x_1\bigr\}\\
\sigma_1 &= \bigl\{x_1 \mapsto f(x_2), \underline{x_2 \mapsto g(x_3, x_4)}, x_3 \mapsto f(x_2)\bigr\},\\
\sigma_2 &= \bigl\{x_1 \mapsto f(g(x_3, x_4)), x_2 \mapsto g(x_3, x_4), \underline{x_3 \mapsto f(g(x_3, x_4))}\bigr\},
\end{aligned}$$

$$\sigma_3 = \bigl\{x_1 \mapsto f(g(f(g(x_3, x_4)), x_4)),\\ x_2 \mapsto g(f(g(x_3, x_4)), x_4), x_3 \mapsto f(g(x_3, x_4))\bigr\}.$$

Then

$$\sigma_0 \xmapsto{\mathcal{S}}{}^{*} \sigma_1 \xmapsto{\mathcal{S}}{}^{*} \sigma_2 \xmapsto{\mathcal{S}}{}^{*} \sigma_3.$$

Note that σ_3 is variable-idempotent and $T \vdash \forall(\sigma_0 \leftrightarrow \sigma_3)$, for all equality theories T; moreover, for all $y \in \mathrm{dom}(\sigma_3) \cap \mathrm{range}(\sigma_3)$, we have $y \in \mathrm{vars}(y\sigma_3)$.

The next two theorems, which are proved at the end of this section, show that we need only consider variable-idempotent substitutions.

Theorem 3.15 *Let T be an equality theory, $\sigma \in RSubst$ and $\sigma \xmapsto{\mathcal{S}}{}^{*} \sigma'$. Then we have $\sigma' \in RSubst$, $\mathrm{vars}(\sigma) = \mathrm{vars}(\sigma')$, $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma')$ and $T \vdash \forall(\sigma \leftrightarrow \sigma')$.*

Theorem 3.16 *Let $\sigma \in RSubst$. Then there exists $\sigma' \in VSubst$ such that $\sigma \xmapsto{\mathcal{S}}{}^{*} \sigma'$ and $y \in \mathrm{dom}(\sigma') \cap \mathrm{range}(\sigma')$ implies $y \in \mathrm{vars}(y\sigma')$.*

The proof of this theorem formalizes the rewriting process informally described in Example 3.14.

Corollary 3.17 *Let T be an equality theory and $\sigma \in RSubst$. There exists a substitution $\sigma' \in VSubst$ such that $\mathrm{vars}(\sigma) = \mathrm{vars}(\sigma')$, $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma')$, $T \vdash \forall(\sigma \leftrightarrow \sigma')$ and $y \in \mathrm{dom}(\sigma') \cap \mathrm{range}(\sigma')$ implies $y \in \mathrm{vars}(y\sigma')$.*

Proof. It is a simple consequence of Theorems 3.15 and 3.16. □

Note that, thanks to the last property, we can also apply Lemma 3.10 to obtain that $\tau \in VSubst$, for all $\tau \subseteq \sigma'$. Thus, substitutions such as σ_1 in Example 3.7 can be disregarded.

The following result concerning composition of substitutions will be needed later.

Lemma 3.18 *Let $\sigma, \tau \in VSubst$, where $\mathrm{dom}(\sigma) \cap \mathrm{vars}(\tau) = \varnothing$. Then $\tau \circ \sigma$ has the following properties.*

1. $T \vdash \forall\bigl((\tau \circ \sigma) \leftrightarrow (\tau \cup \sigma)\bigr)$, *for any equality theory T;*
2. $\mathrm{dom}(\tau \circ \sigma) = \mathrm{dom}(\sigma) \cup \mathrm{dom}(\tau)$;
3. $\tau \circ \sigma \in VSubst$.

3.2.3 The Abstraction Function for *VSubst*

In the two previous sections we have seen how variable-idempotence is a general solution to one of the problems outlined in Section 3.1.2. It is now sufficient to address the other problem, that is the potential loss in precision due to the non-Herbrand substitutions. The simple solution is to define a new abstraction function for *VSubst* which is the same as that in Definition 3.2 but where any sharing group generated by a variable in the domain of the substitution is disregarded. This new abstraction function works for variable-idempotent substitutions and no longer suffers the drawbacks outlined in Section 3.1.2.

Therefore, at least from a theoretical point of view, the problem of defining a sound and precise abstraction function for arbitrary substitutions in rational solved form would have been solved. Given a substitution in *RSubst*, we would proceed in two steps: we first transform it to an equivalent substitution in *VSubst* and then compute the corresponding description by using the modified abstraction function. However, from a practical point of view, it is better to define an abstraction function that directly computes the description of a substitution in *RSubst* in a single step, thus avoiding the expensive computation of the intermediate variable-idempotent substitution. We present such an abstraction function in Section 3.3.

3.2.4 Proofs of the Results of Section 3.2

To prove Lemmas 3.11 and 3.12, we need a few auxiliary results.

The next Lemma shows that, given a substitution in rational solved form, satisfiability is maintained when binding a non-domain variable to a ground and finite term.

Lemma 3.19 *Let $\sigma \in RSubst$ be satisfiable in the equality theory T and consider $x \mapsto t$ such that $x \notin \mathrm{dom}(\sigma)$ and $t \in GTerms \cap HTerms$. Then, $\sigma' \stackrel{\mathrm{def}}{=} \sigma \cup \{x \mapsto t\} \in RSubst$ and σ' is satisfiable in T.*

Proof. As $x \notin \mathrm{dom}(\sigma)$, $\sigma \in RSubst$ and $t \in GTerms \cap HTerms$, then $\sigma' \in RSubst$.

Since σ is satisfiable in T, we have $T \vdash \forall Vars \setminus \mathrm{dom}(\sigma) : \exists\, \mathrm{dom}(\sigma) \,.\, \sigma$. Moreover, by the congruence axiom (2.5), $T \vdash \forall Vars \setminus \{x\} : \exists x \,.\, \{x = t\}$. Hence,

$$T \vdash \forall Vars \setminus \bigl(\mathrm{dom}(\sigma) \cup \{x\}\bigr) : \exists\bigl(\mathrm{dom}(\sigma) \cup \{x\}\bigr) \,.\, \sigma \cup \{x = t\}.$$

Thus $\sigma' = \sigma \cup \{x \mapsto t\}$ is satisfiable in T. □

Syntactically, any substitution in *RSubst* may be regarded as a set of equations in rational solved form and vice versa. The next lemma shows the semantic relationship between them.

Lemma 3.20 *Let T be an equality theory, $\sigma \in RSubst$ and $t \in HTerms$. Then*

$$T \vdash \forall\bigl(\sigma \to (t = t\sigma)\bigr).$$

Proof. We assume the congruence axioms hold and prove that, for any $t \in HTerms$, we have $\sigma \implies \{t = t\sigma\}$. The proof is by induction on the size of t.

Suppose, first that $\mathrm{size}(t) = 1$. If t is a parameter variable of σ or a constant, then $t\sigma = t$ and the result follows from axiom (2.5). If $t \in \mathrm{dom}(\sigma)$, then, for some $r \in HTerms$, $(t \mapsto r) \in \sigma$. Thus $\sigma \implies \{t = t\sigma\}$.

If $\mathrm{size}(t) > 1$, then t has the form $f(s_1, \dots, s_n)$, where $n > 0$ and, for each $i = 1, \dots, n$, $s_i \in \mathit{HTerms}$ and $\mathrm{size}(s_i) < \mathrm{size}(t)$. By the inductive hypothesis, for each $i = 1, \dots, n$, we have $\sigma \implies \{s_i = s_i\sigma\}$. Therefore, applying axiom (2.8), we have $\sigma \implies \{t = t\sigma\}$. □

Lemma 3.21 *Let $\sigma \in \mathit{VSubst}$, $r \in \mathit{HTerms}$ and suppose that $r\sigma^i \in \mathit{Vars}$ for all $i \geq 0$. Then $r\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$.*

Proof. As $\sigma \in \mathit{VSubst}$, for all $i \geq 0$ we have

$$\{r\sigma^{i+1}\} = \mathrm{vars}(r\sigma\sigma^i) = \mathrm{vars}(r\sigma) = \{r\sigma\}.$$

In particular, $r\sigma = r\sigma\sigma$, so that $r\sigma \notin \mathrm{dom}(\sigma)$. □

Lemma 3.22 *Let $\sigma \in \mathit{VSubst}$ be satisfiable in the syntactic equality theory T. Suppose that $v \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$ and $r \in \mathit{HTerms}$ are such that $T \vdash \forall\bigl(\sigma \to \{v = r\}\bigr)$. Then $v = r\sigma$.*

Proof. We assume that the congruence and identity axioms hold.

Let $t_1, t_2 \in \mathit{GTerms} \cap \mathit{HTerms}$ have distinct outer-most symbols so that, by the identity axioms, $T \vdash \forall(t_1 \neq t_2)$. By Lemma 3.21, either $r\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$ or, for some $j \geq 0$, $r\sigma^j \notin \mathit{Vars}$. We consider each case separately.

If, for some $j \geq 0$, $r\sigma^j \notin \mathit{Vars}$, then, as t_1 and t_2 have distinct outer-most symbols, there exists an $i \in \{1, 2\}$ such that the terms t_i and $r\sigma^j$ have distinct outer-most symbols. By the identity axioms, $T \vdash \forall(t_i \neq r\sigma^j)$. Let $\sigma' = \sigma \cup \{v = t_i\}$. It follows from Lemma 3.19 that, as $v \notin \mathrm{dom}(\sigma)$ and σ is satisfiable, $\sigma' \in \mathit{RSubst}$ and is satisfiable. By Lemma 3.20 and the congruence axioms, $\sigma \implies \{v = r\sigma^j\}$. However, we also have $\sigma' \implies \sigma$, so that we obtain $\sigma' \implies \{v = r\sigma^j, v = t_i\}$. Thus, by the congruence axioms, we have $\sigma' \implies \{t_i = r\sigma^j\}$, which is a contradiction.

Suppose then that $r\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$. If $v \neq r\sigma$, then it follows from Lemma 3.19 that $\sigma' = \sigma \cup \{v = t_1, r\sigma = t_2\} \in \mathit{RSubst}$ and, as σ is satisfiable, σ' is satisfiable. By Lemma 3.20, $\sigma \implies \{v = r\sigma\}$. However, we also have $\sigma' \implies \sigma$, so that we obtain $\sigma' \implies \{v = r\sigma, v = t_1, r\sigma = t_2\}$. Thus, by the congruence axioms, we have $\sigma' \implies \{t_1 = t_2\}$, which is a contradiction. Hence $v = r\sigma$ as required. □

Proof of Lemma 3.11 on page 49. We assume that the congruence and identity axioms hold. To prove the result, we suppose that there exists $v \in \mathrm{dom}(\sigma) \setminus \mathrm{dom}(\tau)$ and derive a contradiction.

By hypothesis, $\tau \implies \sigma$. Thus, using Lemma 3.20 and the congruence axioms, we have $\tau \implies \{v = v\sigma^i\}$, for any $i \geq 0$. By Lemma 3.22, for all $i \geq 0$, $v = v\sigma^i\tau$ so that $v\sigma^i \in \mathit{Vars}$. By Lemma 3.21, $v\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$, so that, as σ is ordered and $v \in \mathrm{dom}(\sigma)$, $v\sigma < v$. In particular, $v\sigma \neq v$, so that as $v\sigma\tau = v$ and τ is ordered, we would have $v < v\sigma$, which is a contradiction. □

Proof of Lemma 3.12 on page 49. We assume that the congruence and identity axioms hold. Note that, by the hypothesis, $\tau \implies \sigma$ and $\tau \implies \{s = t\}$ so that, using

Lemma 3.20 and the congruence axioms, we have $\tau \implies \{s = t\sigma^j\}$ and $\tau \implies \{t\sigma\tau^k = s\}$, for all $j, k \geq 0$.

Let $v \in \text{vars}(s) \setminus \text{dom}(\tau)$. We prove, by induction on size(s), that there exists a variable $z \in \text{vars}(t\sigma) \setminus \text{dom}(\sigma)$ such that $v \in \text{vars}(z\tau)$. The base case is when size$(s) = 1$, so that $s = v$. Now, for each $j \geq 0$, $\tau \implies \{v = t\sigma^j\}$ and hence, by Lemma 3.22, as $v \notin \text{dom}(\tau)$, $v = t\sigma^j\tau$. As a consequence, $t\sigma^j \in \mathit{Vars}$ for all $j \geq 0$ and $v = t\sigma\tau$. By Lemma 3.21, $t\sigma \in \mathit{Vars} \setminus \text{dom}(\sigma)$. Thus, we define $z = t\sigma$.

For the inductive step, we assume that size$(s) > 1$ so that, for some $n > 0$, we have $s = f(s_1, \ldots, s_n)$ and, for some $i \in \{1, \ldots, n\}$, $v \in \text{vars}(s_i)$ and size$(s_i) <$ size(s). By Lemma 3.21, either $t\sigma \in \mathit{Vars} \setminus \text{dom}(\sigma)$ or there exists a $j \geq 0$ such that $t\sigma^j \notin \mathit{Vars}$.

First, suppose that $t\sigma \in \mathit{Vars} \setminus \text{dom}(\sigma)$. Now, $\tau \implies \{t\sigma\tau = s\}$ so that, as $s\tau \notin \mathit{Vars}$, by Lemma 3.22, we have $t\sigma\tau \notin \mathit{Vars} \setminus \text{dom}(\tau)$. Thus, by Lemma 3.21, there exists $k > 1$ such that $t\sigma\tau^k \notin \mathit{Vars}$. Then, using the identity axioms, we have $t\sigma\tau^k = f(r_1, \ldots, r_n)$ and $\tau \implies \{s_i = r_i\}$. By the inductive hypothesis (letting σ be the empty substitution), we have $v \in \text{vars}(r_i\tau)$. However, $\text{vars}(r_i) \subseteq \text{vars}(t\sigma\tau^k)$ so that $v \in \text{vars}(t\sigma\tau^{k+1})$. As $\tau \in \mathit{VSubst}$ and $v \notin \text{dom}(\tau)$, $v \in \text{vars}(t\sigma\tau)$. Thus, in this case, let $z = t\sigma$.

Secondly, suppose $t\sigma^j \notin \mathit{Vars}$ for some $j \geq 0$. Then, as $\tau \implies \{s = t\sigma^j\}$, it follows from the identity axioms that $t\sigma^j = f(t_1, \ldots, t_n)$ and $\tau \implies \{s_i = t_i\}$. By the inductive hypothesis, there exists $z \in \text{vars}(t_i\sigma) \setminus \text{dom}(\sigma)$ such that $v \in \text{vars}(z\tau)$. However, $\text{vars}(t_i\sigma) \subseteq \text{vars}(t\sigma^{j+1})$ so that we must have $z \in \text{vars}(t\sigma^{j+1}) \setminus \text{dom}(\sigma)$. As $\sigma \in \mathit{VSubst}$, $z \in \text{vars}(t\sigma) \setminus \text{dom}(\sigma)$ as required. $\square$

To prove Theorem 3.15, we need to show that the result holds for a single $\mathcal{S}$-step.

Lemma 3.23 *Let T be an equality theory, $\sigma \in \mathit{RSubst}$ and $\sigma \overset{\mathcal{S}}{\longmapsto} \sigma'$. Then $\sigma' \in \mathit{RSubst}$,* $\text{dom}(\sigma) = \text{dom}(\sigma')$, $\text{vars}(\sigma) = \text{vars}(\sigma')$, *and $T \vdash \forall(\sigma \leftrightarrow \sigma')$.*

Proof. Since $\sigma \overset{\mathcal{S}}{\longmapsto} \sigma'$, there exists $x, y \in \text{dom}(\sigma)$, with $x \neq y$, such that $(x \mapsto t) \in \sigma$, $(y \mapsto s) \in \sigma$ and $\sigma' = \bigl(\sigma \setminus \{y \mapsto s\}\bigr) \cup \bigl\{y \mapsto s[x/t]\bigr\}$. If $x \notin \text{vars}(s)$, $\sigma = \sigma'$ and the result is trivial. Suppose now that $x \in \text{vars}(s)$. We define

$$\sigma_0 \overset{\text{def}}{=} \sigma \setminus \{x = t, y = s\}.$$

Hence, as it is assumed that $x \neq y$,

$$\sigma = \sigma_0 \cup \{x \mapsto t, y \mapsto s\}, \tag{3.1}$$

$$\sigma' = \sigma_0 \cup \{x \mapsto t, y \mapsto s[x/t]\}. \tag{3.2}$$

We first show that $\sigma' \in \mathit{RSubst}$ and $\text{dom}(\sigma) = \text{dom}(\sigma')$. If $s \notin \mathit{Vars}$, then $s[x/t] \notin \mathit{Vars}$ so that $\text{dom}(\sigma) = \text{dom}(\sigma')$. Also, as σ has no circular subset, σ' has no circular subset and $\sigma' \in \mathit{RSubst}$. If $s \in \mathit{Vars}$, then $s = x$ and $s[x/t] = t$. Thus, as $\sigma = \sigma_0 \cup \{x \mapsto t, y \mapsto x\}$ has no circular subset, $t \neq y$ so that $\text{dom}(\sigma) = \text{dom}(\sigma')$. Moreover, neither $\sigma_0 \cup \{x \mapsto t\}$ nor $\sigma_0 \cup \{y \mapsto t\}$ have circular subsets. Hence σ' has no circular subset. Thus $\sigma' \in \mathit{RSubst}$.

Now, since

$$\big(\mathrm{vars}(s) \cup \mathrm{vars}(t)\big) \setminus \mathrm{dom}(\sigma) = \mathrm{vars}\big(s[x/t] \cup \mathrm{vars}(t)\big) \setminus \mathrm{dom}(\sigma),$$

it follows that $\mathrm{vars}(\sigma) = \mathrm{vars}(\sigma')$.

Therefore, it remains to show that $T \vdash \forall(\sigma \leftrightarrow \sigma')$ for any equality theory T. To do this, we assume that the congruence axioms hold, and show that $\sigma \iff \sigma'$. By Lemma 3.20, we have

$$\{x = t\} \implies \{s = s[x/t]\}.$$

Thus, using the congruence axiom (2.7), we have

$$\begin{aligned}\{x = t, y = s\} &\implies \big\{x = t, y = s, s = s[x/t]\big\}\\ &\implies \big\{x = t, y = s[x/t]\big\}.\end{aligned}$$

Similarly, using congruence axioms (2.6) and (2.7), we have

$$\begin{aligned}\big\{x = t, y = s[x/t]\big\} &\implies \big\{x = t, y = s[x/t], s = s[x/t]\big\}\\ &\implies \{x = t, y = s\}.\end{aligned}$$

Thus

$$\{x = t, y = s\} \iff \big\{x = t, y = s[x/t]\big\}.$$

It therefore follows from (3.1) and (3.2) that $\sigma \iff \sigma'$. □

The condition $x \neq y$ in the proof of Lemma 3.23 is necessary. For example, suppose $\sigma = \big\{x \mapsto f(x)\big\}$ and $\sigma' = \big\{x \mapsto f(f(x))\big\}$. Then we do not have $\sigma' \implies \sigma$. Note however that this implication will hold as soon as we enrich the equality theory T with either the occurs-check axioms of the finite-tree theory or the uniqueness axioms of the rational-tree theory.

Proof of Theorem 3.15 on page 50. The proof is by induction on the length of the sequence of $\mathcal{S}$-steps transforming σ to σ'. The base case is the empty sequence. For the inductive step, the sequence has length $n > 0$ and there exists σ_1 such that $\sigma \overset{\mathcal{S}}{\longmapsto} \sigma_1 \overset{\mathcal{S}}{\longmapsto}{}^{*} \sigma'$ and $\sigma_1 \overset{\mathcal{S}}{\longmapsto}{}^{*} \sigma'$ has length $n - 1$. By applying Lemma 3.23, we obtain $\sigma_1 \in \mathit{RSubst}$, $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma_1)$, $\mathrm{vars}(\sigma) = \mathrm{vars}(\sigma_1)$ and $T \vdash \forall(\sigma \leftrightarrow \sigma_1)$. By the inductive hypothesis, $\sigma' \in \mathit{RSubst}$, $\mathrm{dom}(\sigma_1) = \mathrm{dom}(\sigma')$, $\mathrm{vars}(\sigma_1) = \mathrm{vars}(\sigma')$ and $T \vdash \forall(\sigma_1 \leftrightarrow \sigma')$. Hence we have $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma')$, $\mathrm{vars}(\sigma) = \mathrm{vars}(\sigma')$, and $T \vdash \forall(\sigma \leftrightarrow \sigma')$. □

Proof of Theorem 3.16 on page 50. To prove the theorem, we construct an $\mathcal{S}$-transformation and show that the resulting substitution has the required properties.

Suppose that $\{x_1, \dots, x_n\} = \operatorname{dom}(\sigma)$, $\sigma_0 = \sigma$ and, for each $j = 0, \dots, n$,

$$\sigma_j = \{x_1 \mapsto t_{1,j}, \dots, x_n \mapsto t_{n,j}\},$$

where, if $j > 0$, $t_{j,j} = t_{j,j-1}$ and, for each $i = 1, \dots, n$ with $i \neq j$, $t_{i,j} = t_{i,j-1}[x_j/t_{j,j}]$.

It follows from the definition of σ_j that, for $j = 1, \dots, n$, σ_j can be obtained from σ_{j-1} by two sequences of $\mathcal{S}$-steps of lengths $j-1$ and $n-j+1$:

$$\sigma_{j-1} = \sigma_{j-1}^0 \overset{\mathcal{S}}{\longmapsto} \cdots \overset{\mathcal{S}}{\longmapsto} \sigma_{j-1}^{j-1} = \sigma_{j-1}^j \overset{\mathcal{S}}{\longmapsto} \cdots \overset{\mathcal{S}}{\longmapsto} \sigma_{j-1}^n = \sigma_j,$$

where, for $i = 1, \dots, n$ with $i \neq j$,

$$\begin{aligned} \sigma_{j-1}^i &= \bigl(\sigma_{j-1}^{i-1} \setminus \{x_i \mapsto t_{i,j-1}\}\bigr) \cup \bigl\{x_i \mapsto t_{i,j-1}[x_j/t_{j,j}]\bigr\} \\ &= \{x_1 \mapsto t_{1,j}, \dots, x_i \mapsto t_{i,j}, x_{i+1} \mapsto t_{i+1,j-1}, \dots, x_n \mapsto t_{n,j-1}\}. \end{aligned}$$

Hence, by Theorem 3.15, $\sigma_1, \dots, \sigma_n \in \mathit{RSubst}$.

We next show, by induction on j, with $0 \leq j \leq n$, that, for each $i = 1, \dots, n$ and each $h = 1, \dots, j$, we have $\operatorname{vars}(t_{i,j}) = \operatorname{vars}\bigl(t_{i,j}[x_h/t_{h,j}]\bigr)$.

For the base case when $j = 0$ there is nothing to prove. Suppose, therefore, that $1 \leq j \leq n$ and that, for each $i = 1, \dots, n$ and $h = 1, \dots, j-1$,

$$\operatorname{vars}(t_{i,j-1}) = \operatorname{vars}\bigl(t_{i,j-1}[x_h/t_{h,j-1}]\bigr).$$

Now by the definition of $t_{k,j}$ where $1 \leq k \leq n$, $k \neq j$, we have

$$\operatorname{vars}(t_{k,j}) = \operatorname{vars}\bigl(t_{k,j-1}[x_j/t_{j,j}]\bigr). \tag{3.3}$$

Since a substitution consisting of a single binding is variable-idempotent,

$$\operatorname{vars}(t_{j,j}) = \operatorname{vars}\bigl(t_{j,j}[x_j/t_{j,j}]\bigr)$$

so that, as $t_{j,j} = t_{j,j-1}$,

$$\operatorname{vars}(t_{j,j}) = \operatorname{vars}\bigl(t_{j,j-1}[x_j/t_{j,j}]\bigr). \tag{3.4}$$

Thus, by (3.3) and (3.4), for all k such that $1 \leq k \leq n$, we have

$$\operatorname{vars}(t_{k,j}) = \operatorname{vars}\bigl(t_{k,j-1}[x_j/t_{j,j}]\bigr). \tag{3.5}$$

Therefore, for each $i = 1, \dots, n$ and $h = 1, \dots, j$, using (3.5) and the inductive hypothesis, we have

$$\operatorname{vars}\bigl(t_{i,j}[x_h/t_{h,j}]\bigr) = \operatorname{vars}\Bigl(t_{i,j-1}[x_j/t_{j,j}]\bigl[x_h/t_{h,j-1}[x_j/t_{j,j}]\bigr]\Bigr)$$

$$\begin{aligned}
&= \mathrm{vars}\big(t_{i,j-1}[x_h/t_{h,j-1}][x_j/t_{j,j}]\big)\\
&= \mathrm{vars}\big(t_{i,j-1}[x_j/t_{j,j}]\big)\\
&= \mathrm{vars}(t_{i,j}).
\end{aligned}$$

Letting $j = n$ we obtain, for each $i, h = 1, \ldots, n$,

$$\mathrm{vars}\big(t_{i,n}[x_h/t_{h,n}]\big) = \mathrm{vars}(t_{i,n}).$$

Therefore, for each $i = 1, \ldots, n$,

$$\mathrm{vars}(t_{i,n}\sigma_n) = \mathrm{vars}(t_{i,n}).$$

Thus, letting $\sigma' = \sigma_n$, by Lemma 3.6, we obtain $\sigma' \in \mathit{VSubst}$.

Finally, we show by induction on j, with $0 \leq j \leq n$, that

$$\forall h \in \{1, \ldots, j\} : x_h \in \mathrm{range}(\sigma_j) \implies x_h \in \mathrm{vars}(x_h\sigma_j). \tag{3.6}$$

For the base case when $j = 0$ there is nothing to prove. Suppose, therefore, that (3.6) holds for an index j (where $0 \leq j < n$) and consider the index $j + 1$. It is not difficult to observe that, by construction,

$$\mathrm{range}(\sigma_{j+1}) = \big(\mathrm{range}(\sigma_j) \setminus \{x_{j+1}\}\big) \cup \mathrm{vars}(t_{j+1,j}).$$

Thus, for each $h \in \{1, \ldots, j\}$, since $x_h \neq x_{j+1}$, we have

$$\begin{aligned}
x_h \in \mathrm{range}(\sigma_{j+1}) &\iff x_h \in \mathrm{range}(\sigma_j)\\
&\implies x_h \in \mathrm{vars}(x_h\sigma_j)\\
&\implies x_h \in \mathrm{vars}(x_h\sigma_{j+1}).
\end{aligned}$$

Consider now $h = j + 1$. Note that, if $x_{j+1} \notin \mathrm{vars}(x_{j+1}\sigma_{j+1}) = \mathrm{vars}(t_{j+1,j})$, then every occurrence of x_{j+1} in the terms $t_{i,j}$ (where $i \neq j$) will be replaced, in $t_{i,j+1}$, by term $t_{j+1,j}$, so that $x_h \notin \mathrm{range}(\sigma_{j+1})$. Formally, we have

$$x_h \in \mathrm{range}(\sigma_{j+1}) \implies x_{j+1} \in \mathrm{vars}(x_{j+1}\sigma_{j+1}),$$

therefore completing the inductive proof. By letting $j = n$ in (3.6), since $\sigma_n = \sigma'$, we obtain that $y \in \mathrm{dom}(\sigma') \cap \mathrm{range}(\sigma')$ implies $y \in \mathrm{vars}(y\sigma')$. □

Proof of Lemma 3.18 on page 50. We have that $(\tau \cup \sigma) \in \mathit{RSubst}$ because, by hypothesis, $\sigma, \tau \in \mathit{RSubst}$ and $\mathrm{dom}(\sigma) \cap \mathrm{vars}(\tau) = \varnothing$. It follows from (2.4) that $\tau \circ \sigma$ can be obtained from $(\tau \cup \sigma)$ by a sequence of $\mathcal{S}$-steps so that, by Theorem 3.15, we have properties 1 and 2.

To prove property 3, we will show that, for all terms $t \in \mathit{HTerms}$,

$$\mathrm{vars}(t\sigma\tau) = \mathrm{vars}(t\sigma\tau\sigma\tau).$$

- We start by proving the inclusion $\mathrm{vars}(t\sigma\tau) \subseteq \mathrm{vars}(t\sigma\tau\sigma\tau)$. Thus, let $z \in \mathrm{vars}(t\sigma\tau)$.

  First note that, if $z \notin \mathrm{dom}(\sigma) \cup \mathrm{dom}(\tau)$, then the result is trivial.

  Suppose $z \in \mathrm{dom}(\sigma)$. By hypothesis, $z \notin \mathrm{vars}(\tau)$ so that $z \in \mathrm{vars}(t\sigma)$. Since σ is variable-idempotent, $z \in \mathrm{vars}(t\sigma\sigma)$, so that there exists $v \in \mathrm{vars}(t\sigma) \cap \mathrm{dom}(\sigma)$ such that $z \in \mathrm{vars}(v\sigma)$. Thus $v \notin \mathrm{vars}(\tau)$, so that $v \in \mathrm{vars}(t\sigma\tau)$. Therefore $z \in \mathrm{vars}(t\sigma\tau\sigma)$ and, since $z \notin \mathrm{vars}(\tau)$, we can conclude $z \in \mathrm{vars}(t\sigma\tau\sigma\tau)$.

  Otherwise, let $z \in \mathrm{dom}(\tau)$, so that $z \notin \mathrm{dom}(\sigma)$. There exists $v \in \mathrm{vars}(t\sigma) \cap \mathrm{dom}(\tau)$ such that $z \in \mathrm{vars}(v\tau)$. Since τ is variable-idempotent, $z \in \mathrm{vars}(v\tau\tau)$ so that there exists $w \in \mathrm{vars}(v\tau) \cap \mathrm{dom}(\tau)$ such that $z \in \mathrm{vars}(w\tau)$. Since $w \notin \mathrm{dom}(\sigma)$ then $w \in \mathrm{vars}(t\sigma\tau\sigma)$. Therefore we can conclude $z \in \mathrm{vars}(t\sigma\tau\sigma\tau)$.

- To prove the other inclusion, let $z \in \mathrm{vars}(t\sigma\tau\sigma\tau)$, so that there exists $v \in \mathrm{vars}(t\sigma\tau\sigma)$ such that $z \in \mathrm{vars}(v\tau)$. Similarly, there exists $w \in \mathrm{vars}(t\sigma\tau)$ such that $v \in \mathrm{vars}(w\sigma)$.

  Suppose $v \neq w$. Then $w \in \mathrm{dom}(\sigma)$, so that by hypothesis $w \notin \mathrm{vars}(\tau)$. As a consequence, $w \in \mathrm{vars}(t\sigma)$, $v \in \mathrm{vars}(t\sigma\sigma)$ and $z \in \mathrm{vars}(t\sigma\sigma\tau)$. Thus, as $\sigma \in \mathit{VSubst}$, we obtain $z \in \mathrm{vars}(t\sigma\tau)$.

  Otherwise, if $v = w$, there exists $x \in \mathrm{vars}(t\sigma)$ such that $z \in \mathrm{vars}(x\tau\tau)$. Thus, $z \in \mathrm{vars}(t\sigma\tau\tau)$ and, since $\tau \in \mathit{VSubst}$, $z \in \mathrm{vars}(t\sigma\tau)$.

□

3.3 The Abstraction Function for *RSubst*

In this section we define a new abstraction function mapping arbitrary substitutions in rational solved form into their abstract descriptions. This abstraction function is based on a new definition for the notion of *occurrence*. The new occurrence operator 'occ' is defined on *RSubst* so that it does not require the explicit computation of intermediate variable-idempotent substitutions. In particular, it can be seen as a partial evaluation of the composition of the $\mathcal{S}$-transformation algorithm and the abstraction function for *VSubst*, as informally outlined in the previous section. The 'occ' operator, which is defined as the fixed point of a sequence of *occurrence functions*, generalizes the 'sg' operator, defined for *ISubst*, coinciding with it when applied to idempotent substitutions.

Definition 3.24 (Occurrence functions.) *For each $n \in \mathbb{N}$, the* occurrence function $\mathrm{occ}_n : \mathit{RSubst} \times \mathit{Vars} \to \wp_{\mathrm{f}}(\mathit{Vars})$ *is defined, for each $\sigma \in \mathit{RSubst}$ and each $v \in \mathit{Vars}$, by*

$$\mathrm{occ}_0(\sigma, v) \stackrel{\mathrm{def}}{=} \{v\} \setminus \mathrm{dom}(\sigma),$$
$$\mathrm{occ}_{n+1}(\sigma, v) \stackrel{\mathrm{def}}{=} \bigl\{\, y \in \mathit{Vars} \bigm| \mathrm{vars}(y\sigma) \cap \mathrm{occ}_n(\sigma, v) \neq \varnothing \,\bigr\}.$$

The following monotonicity property for occ_n is proved at the end of this section.

Lemma 3.25 *For each $n \in \mathbb{N}$, each $\sigma \in RSubst$ and each $v \in Vars$,*

$$\mathrm{occ}_n(\sigma, v) \subseteq \mathrm{occ}_{n+1}(\sigma, v).$$

Note that, by considering the substitution $\{u \mapsto v, v \mapsto w\}$, it can be seen that, if we had not excluded the domain variables in the definition of occ_0, then this monotonicity property would not have held.

For any n, the set $\mathrm{occ}_n(\sigma, v)$ is restricted to the set $\{v\} \cup \mathrm{dom}(\sigma)$. Thus, it follows from Lemma 3.25, that there is an index $\ell \leq \# \sigma$ such that $\mathrm{occ}_\ell(\sigma, v) = \mathrm{occ}_n(\sigma, v)$ for all $n \geq \ell$.

Definition 3.26 (Occurrence operator.) *For each $\sigma \in RSubst$ and each $v \in Vars$, the* occurrence operator $\mathrm{occ} \colon RSubst \times Vars \to \wp_{\mathrm{f}}(Vars)$ *is defined by*

$$\mathrm{occ}(\sigma, v) \stackrel{\mathrm{def}}{=} \mathrm{occ}_{\# \sigma}(\sigma, v).$$

Note that, by combining Definitions 3.24 and 3.26, we obtain

$$\mathrm{occ}(\sigma, v) = \bigl\{\, y \in Vars \bigm| \mathrm{vars}(y\sigma) \cap \mathrm{occ}(\sigma, v) \neq \varnothing \,\bigr\}. \tag{3.7}$$

The following simpler characterizations for 'occ' can be used when the variable is in the domain of the substitution, the substitution is variable-idempotent or the substitution is idempotent.

Lemma 3.27 *If $\sigma \in RSubst$ and $v \in \mathrm{dom}(\sigma)$, then $\mathrm{occ}(\sigma, v) = \varnothing$.*

Lemma 3.28 *If $\sigma \in VSubst$ then, for each $v \in Vars$,*

$$\begin{aligned} \mathrm{occ}(\sigma, v) &= \mathrm{occ}_1(\sigma, v) \\ &= \bigl\{\, y \in Vars \bigm| v \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma) \,\bigr\}. \end{aligned}$$

Lemma 3.29 *If $\sigma \in ISubst$ and $v \in Vars$ then $\mathrm{occ}(\sigma, v) = \mathrm{sg}(\sigma, v)$.*

The next result shows that the 'occ' operator precisely captures the intended property. All of these results are proved at the end of this section.

Proposition 3.30 *Let $\sigma \in RSubst$ and $y, v \in Vars$. Then*

$$y \in \mathrm{occ}(\sigma, v) \qquad \Longleftrightarrow \qquad v \in \mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr).$$

Example 3.31 *Consider* Example 3.14. *For all $i \geq 0$, we have* $\mathrm{dom}(\sigma_i) = \{x_1, x_2, x_3\}$, *so that*

$$\mathrm{occ}(\sigma_i, x_1) = \mathrm{occ}(\sigma_i, x_2) = \mathrm{occ}(\sigma_i, x_3) = \varnothing.$$

Moreover,

$$\begin{aligned}
\mathrm{occ}_0(\sigma_0, x_4) &= \{x_4\},\\
\mathrm{occ}_1(\sigma_0, x_4) &= \{x_2, x_4\},\\
\mathrm{occ}_2(\sigma_0, x_4) &= \{x_1, x_2, x_4\},\\
\mathrm{occ}_3(\sigma_0, x_4) &= \{x_1, x_2, x_3, x_4\} = \mathrm{occ}(\sigma_0, x_4).
\end{aligned}$$

Also, note that

$$\mathrm{occ}_1(\sigma_3, x_4) = \{x_1, x_2, x_3, x_4\} = \mathrm{occ}(\sigma_3, x_4).$$

The definition of abstraction is based on the occurrence operator.

Definition 3.32 (Abstraction.) *The concrete domain $\mathcal{D}^\flat$ is related to SS by means of the* abstraction function $\alpha\colon \mathcal{D}^\flat \to SS$. *For each $\Sigma \in \wp(RSubst)$ and $VI \in \wp_{\mathrm{f}}(Vars)$,*

$$\alpha\bigl((\Sigma, VI)\bigr) \stackrel{\mathrm{def}}{=} \mathrm{Alub}\bigl\{\, \alpha(\sigma, VI) \bigm| \sigma \in \Sigma \,\bigr\}$$

where $\alpha\colon RSubst \times \wp_{\mathrm{f}}(Vars) \to SS$ is defined, for each $\sigma \in RSubst$ and $VI \in \wp_{\mathrm{f}}(Vars)$, by

$$\alpha(\sigma, VI) \stackrel{\mathrm{def}}{=} \Bigl(\bigl\{\, \mathrm{occ}(\sigma, v) \cap VI \bigm| v \in Vars \,\bigr\} \setminus \{\varnothing\}, VI\Bigr).$$

Example 3.33 *Let us consider* Examples 3.14 *and* 3.31 *once more. Then, assuming $VI = \{x_1, x_2, x_3, x_4\}$,*

$$\alpha(\sigma_0, VI) = \Bigl(\bigl\{\mathrm{occ}(\sigma_0, x_4)\bigr\}, VI\Bigr) = \Bigl(\bigl\{\{x_1, x_2, x_3, x_4\}\bigr\}, VI\Bigr).$$

As a second example, consider the substitution

$$\sigma = \bigl\{x_1 \mapsto f(x_1), x_2 \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_2\bigr\}.$$

Then

$$\mathrm{occ}(\sigma, x_1) = \mathrm{occ}(\sigma, x_2) = \mathrm{occ}(\sigma, x_3) = \mathrm{occ}(\sigma, x_4) = \varnothing$$

so that, if we again assume $VI = \{x_1, x_2, x_3, x_4\}$,

$$\alpha(\sigma, VI) = (\varnothing, VI).$$

Any substitution in rational solved form is equivalent, with respect to any equality theory, to a variable-idempotent substitution having the same abstraction.

Theorem 3.34 *Let $\sigma \in RSubst$ be satisfiable in the equality theory T. Then, there exists $\sigma' \in VSubst$ such that* $\mathrm{vars}(\sigma) = \mathrm{vars}(\sigma')$, $T \vdash \forall(\sigma \leftrightarrow \sigma')$, $y \in \mathrm{dom}(\sigma') \cap \mathrm{range}(\sigma')$ *implies* $y \in \mathrm{vars}(y\sigma')$ *and* $\alpha(\sigma, VI) = \alpha(\sigma', VI)$, *for any* $VI \in \wp_{\mathrm{f}}(Vars)$.

Equivalent substitutions in rational solved form have the same abstraction. We note that this property is essential for the implementation of the *SS* domain.

Theorem 3.35 *Let $\sigma, \sigma' \in \mathit{RSubst}$ be satisfiable in the syntactic equality theory T and suppose $T \vdash \forall(\sigma \leftrightarrow \sigma')$. Then $\alpha(\sigma, \mathit{VI}) = \alpha(\sigma', \mathit{VI})$, for any $\mathit{VI} \in \wp_{\mathrm{f}}(\mathit{Vars})$.*

3.3.1 Proofs of the Results of Section 3.3

Proof of Lemma 3.25 on page 58. The proof is by induction on n. For the base case, when $n = 1$, if $\mathit{occ}_0(\sigma, v) \neq \varnothing$ then $v \notin \operatorname{dom}(\sigma)$ and $\mathit{occ}_0(\sigma, v) = \{v\}$. Thus, $v = v\sigma$ so that, by Definition 3.24, $v \in \operatorname{occ}_1(\sigma, v)$. Suppose $n > 1$. Then, if $y \in \operatorname{occ}_{n-1}(\sigma, v)$, we have, by Definition 3.24, $\operatorname{vars}(y\sigma) \cap \operatorname{occ}_{n-2}(\sigma, v) \neq \varnothing$. By the inductive hypothesis,

$$\operatorname{occ}_{n-2}(\sigma, v) \subseteq \operatorname{occ}_{n-1}(\sigma, v)$$

so that $\operatorname{vars}(y\sigma) \cap \operatorname{occ}_{n-1}(\sigma, v) \neq \varnothing$ and thus $y \in \operatorname{occ}_n(\sigma, v)$. $\square$

Proof of Lemma 3.27 on page 58. By Definition 3.24, $\operatorname{occ}_0(\sigma, v) = \varnothing$ and, for all $n > 0$, we have $\operatorname{occ}_n(\sigma, v) = \varnothing$ if $\operatorname{occ}_{n-1}(\sigma, v) = \varnothing$. Thus, $\operatorname{occ}_n(\sigma, v) = \varnothing$, for all $n \geq 0$, so that, by Definition 3.26, $\operatorname{occ}(\sigma, v) = \varnothing$. $\square$

Proof of Lemma 3.28 on page 58. Suppose first that $v \in \operatorname{dom}(\sigma)$. Then

$$\{\, y \in \mathit{Vars} \mid v \in \operatorname{vars}(y\sigma) \setminus \operatorname{dom}(\sigma) \,\} = \varnothing.$$

Also, by Lemma 3.27, $\operatorname{occ}_1(\sigma, v) = \operatorname{occ}(\sigma, v) = \varnothing$.

Suppose next that $v \notin \operatorname{dom}(\sigma)$. It follows from Definition 3.24, that

$$\begin{aligned}
\operatorname{occ}_0(\sigma, v) &= \{v\}, \\
\operatorname{occ}_1(\sigma, v) &= \{\, y \in \mathit{Vars} \mid \operatorname{vars}(y\sigma) \cap \{v\} \neq \varnothing \,\} \\
&= \{\, y \in \mathit{Vars} \mid v \in \operatorname{vars}(y\sigma) \,\}, \\
\operatorname{occ}_2(\sigma, v) &= \Bigl\{\, y \in \mathit{Vars} \Bigm| \operatorname{vars}(y\sigma) \cap \{\, y_1 \in \mathit{Vars} \mid v \in \operatorname{vars}(y_1\sigma) \,\} \neq \varnothing \,\Bigr\} \\
&= \{\, y \in \mathit{Vars} \mid v \in \operatorname{vars}(y\sigma^2) \,\}.
\end{aligned}$$

Since $\sigma \in \mathit{VSubst}$, $\operatorname{vars}(y\sigma) = \operatorname{vars}(y\sigma^2)$. Thus, $\operatorname{occ}_1(\sigma, v) = \operatorname{occ}_2(\sigma, v)$ and hence, by Definition 3.24, we have also $\operatorname{occ}_n(\sigma, v) = \operatorname{occ}_1(\sigma, v)$, for all $n \geq 1$. Therefore, by Definition 3.26, $\operatorname{occ}(\sigma, v) = \operatorname{occ}_1(\sigma, v) = \{\, y \in \mathit{Vars} \mid v \in \operatorname{vars}(y\sigma) \,\}$. $\square$

Proof of Lemma 3.29 on page 58. As $\sigma \in \mathit{ISubst}$ then, for all $y \in \mathit{Vars}$, we have

$\mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma) = \mathrm{vars}(y\sigma)$. Also, as $\sigma \in \mathit{VSubst}$, we can apply Lemma 3.28 so that

$$\begin{aligned}\mathrm{occ}(\sigma, v) &= \{\, y \in \mathit{Vars} \mid v \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma) \,\}\\ &= \{\, y \in \mathit{Vars} \mid v \in \mathrm{vars}(y\sigma) \,\}\\ &= \mathrm{sg}(\sigma, v).\end{aligned}$$

□

To prove Proposition 3.30 and Theorem 3.34, we need to show that the operator 'occ' is invariant with respect to $\mathcal{S}$-transformation.

Lemma 3.36 *Let $\sigma, \sigma' \in \mathit{RSubst}$ be such that $\sigma \overset{\mathcal{S}}{\longmapsto}{}^{*} \sigma'$. Then, for all $v \in \mathit{Vars}$, we have $\mathrm{occ}(\sigma, v) = \mathrm{occ}(\sigma', v)$.*

Proof. Suppose first that $\sigma \overset{\mathcal{S}}{\longmapsto} \sigma'$. Thus we assume that $(x \mapsto t) \in \sigma$ and $(y \mapsto s) \in \sigma$, where $x \neq y$, and that

$$\sigma' = \bigl(\sigma \setminus \{y \mapsto s\}\bigr) \cup \bigl\{y \mapsto s[x/t]\bigr\}. \tag{3.8}$$

If $x \notin \mathrm{vars}(s)$, then $\sigma' = \sigma$ and there is nothing to prove. Also, if $v \in \mathrm{dom}(\sigma)$ then, by Theorem 3.15, $v \in \mathrm{dom}(\sigma')$ so that, by Lemma 3.27, $\mathrm{occ}(\sigma, v) = \mathrm{occ}(\sigma', v) = \varnothing$.

We now assume that $x \in \mathrm{vars}(s)$ and $v = v\sigma = v\sigma'$. We first prove that, for each $m \geq 0$,

$$\mathrm{occ}_m(\sigma, v) \subseteq \mathrm{occ}(\sigma', v). \tag{3.9}$$

The proof is by induction on m. By Definition 3.24, we have that

$$\mathrm{occ}_0(\sigma, v) = \mathrm{occ}_0(\sigma', v) = \{v\},$$

so that (3.9) holds for $m = 0$. Suppose then that $m > 0$ and $v_m \in \mathrm{occ}_m(\sigma, v)$. Then, to prove (3.9), we must show that $v_m \in \mathrm{occ}(\sigma', v)$. By Definition 3.24, there exists

$$v_{m-1} \in \mathrm{vars}(v_m\sigma) \cap \mathrm{occ}_{m-1}(\sigma, v). \tag{3.10}$$

Hence, by the inductive hypothesis, $v_{m-1} \in \mathrm{occ}(\sigma', v)$. If $v_{m-1} \in \mathrm{vars}(v_m\sigma')$, then, by (3.7), $v_m \in \mathrm{occ}(\sigma', v)$. Suppose now that $v_{m-1} \notin \mathrm{vars}(v_m\sigma')$. Since, by (3.10), we have that $v_{m-1} \in \mathrm{vars}(v_m\sigma)$, it follows, using (3.8), that $v_m = y$ and $v_{m-1} = x$. However, by assumption, $v \notin \mathrm{dom}(\sigma)$, so that $x \neq v$ and $m > 1$. Thus, by Definition 3.24, there exists

$$v_{m-2} \in \mathrm{vars}(x\sigma) \cap \mathrm{occ}_{m-2}(\sigma, v). \tag{3.11}$$

However, $x\sigma = t$ and $x \in \mathrm{vars}(s)$ so that, by (3.11), $v_{m-2} \in \mathrm{vars}\bigl(s[x/t]\bigr)$. Since, by (3.8), $\bigl(y \mapsto s[x/t]\bigr) \in \sigma'$, we have also $v_{m-2} \in \mathrm{vars}(y\sigma')$. Moreover, by (3.11), $v_{m-2} \in \mathrm{occ}_{m-2}(\sigma, v)$ so that, by the inductive hypothesis, we have that $v_{m-2} \in \mathrm{occ}(\sigma', v)$. Thus, by Eq. (3.7), as $v_m = y$, $v_m \in \mathrm{occ}(\sigma', v)$.

Conversely, we now prove that, for all $m \geq 0$,

$$\mathrm{occ}_m(\sigma', v) \subseteq \mathrm{occ}(\sigma, v). \tag{3.12}$$

The proof is by induction on m. As before, $\mathrm{occ}_0(\sigma', v) = \mathrm{occ}_0(\sigma, v) = \{v\}$ so that (3.12) holds for $m = 0$. Suppose then that $m > 0$ and $v_m \in \mathrm{occ}_m(\sigma', v)$. Then, to prove (3.12), we must show that $v_m \in \mathrm{occ}(\sigma, v)$. By Definition 3.24, there exists

$$v_{m-1} \in \mathrm{vars}(v_m\sigma') \cap \mathrm{occ}_{m-1}(\sigma', v). \tag{3.13}$$

Hence, by the inductive hypothesis, $v_{m-1} \in \mathrm{occ}(\sigma, v)$. If $v_{m-1} \in \mathrm{vars}(v_m\sigma)$ then, by (3.7), we have $v_m \in \mathrm{occ}(\sigma, v)$. Suppose now that $v_{m-1} \notin \mathrm{vars}(v_m\sigma)$. Since, by (3.13), we have $v_{m-1} \in \mathrm{vars}(v_m\sigma')$, it follows, using (3.8), that $v_m = y$ and $v_{m-1} \in \mathrm{vars}(t) = \mathrm{vars}(x\sigma)$. Hence, since $v_{m-1} \in \mathrm{occ}(\sigma, v)$, by (3.7), also $x \in \mathrm{occ}(\sigma, v)$. Furthermore, $x \in \mathrm{vars}(y\sigma)$ so again, by (3.7), as $v_m = y$, $v_m \in \mathrm{occ}(\sigma, v)$.

Combining (3.9) and (3.12) we obtain the result that, if σ' is obtained from σ by a single $\mathcal{S}$-step, then $\mathrm{occ}(\sigma, v) = \mathrm{occ}(\sigma', v)$.

Suppose now that $\sigma = \sigma_1 \overset{\mathcal{S}}{\longmapsto} \cdots \overset{\mathcal{S}}{\longmapsto} \sigma_n = \sigma'$. If $n = 1$, then $\sigma = \sigma'$. If $n > 1$, we have by the first part of the proof that, for each $i = 2, \ldots, n$, $\mathrm{occ}(\sigma_{i-1}, v) = \mathrm{occ}(\sigma_i, v)$, and hence the required result. $\square$

In Lemma 3.36 the choice of the variable v is arbitrary: thus the following Corollary follows easily from Definition 3.32.

Corollary 3.37 *Let $\sigma, \sigma' \in RSubst$ be such that $\sigma \overset{\mathcal{S}}{\longmapsto}{}^* \sigma'$. Then, for all $VI \in \wp_{\mathrm{f}}(Vars)$, we have $\alpha(\sigma, VI) = \alpha(\sigma', VI)$.*

The next result shows that the operator 'occ' behaves as expected on all variable-idempotent substitutions.

Lemma 3.38 *Let $\sigma \in VSubst$ and $y, v \in Vars$. Then*

$$y \in \mathrm{occ}(\sigma, v) \qquad \Longleftrightarrow \qquad v \in \mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr).$$

Proof. By Lemma 3.28, $y \in \mathrm{occ}(\sigma, v)$ if and only if $v \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma)$.

To prove the first implication ($\Rightarrow$), let $v \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma)$. Then, for all $i > 0$, we have $v \in \mathrm{vars}(y\sigma^i) \setminus \mathrm{dom}(\sigma)$, so that $v \in \mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr)$.

To prove the other implication ($\Leftarrow$), assume that $v \in \mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr)$. We prove by contradiction that $v \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma)$. In fact, assume that $v \notin \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma)$. Then, since $\sigma \in VSubst$, by Definition 3.4 we obtain $v \notin \mathrm{vars}(y\sigma\sigma) \setminus \mathrm{dom}(\sigma)$ so that, for all $i > 0$, $v \notin \mathrm{vars}(y\sigma^i) \setminus \mathrm{dom}(\sigma)$. Thus, by definition of 'rt', $v \notin \mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr)$. $\square$

In order to prove Proposition 3.30 it is useful to provide an auxiliary result relating the function 'rt' and the concept of equivalence under a syntactic equality theory.

Lemma 3.39 *Let $\sigma \in RSubst$ be satisfiable in the syntactic equality theory T. Suppose $s, t \in HTerms$ are such that $T \vdash \forall\bigl(\sigma \rightarrow (s = t)\bigr)$. Then* $\mathrm{rt}(s,\sigma) = \mathrm{rt}(t,\sigma)$.

Proof. We suppose, toward a contradiction, that $\mathrm{rt}(s,\sigma) \neq \mathrm{rt}(t,\sigma)$. Then, there must exist a finite path p such that:

a. $x = \mathrm{rt}(s,\sigma)[p] \in Vars \setminus \mathrm{dom}(\sigma)$, $y = \mathrm{rt}(t,\sigma)[p] \in Vars \setminus \mathrm{dom}(\sigma)$ and $x \neq y$; or

b. $x = \mathrm{rt}(s,\sigma)[p] \in Vars \setminus \mathrm{dom}(\sigma)$ and $r = \mathrm{rt}(t,\sigma)[p] \notin Vars$ or, symmetrically, we have $r = \mathrm{rt}(s,\sigma)[p] \notin Vars$ and $x = \mathrm{rt}(t,\sigma)[p] \in Vars \setminus \mathrm{dom}(\sigma)$; or

c. $r_1 = \mathrm{rt}(s,\sigma)[p] \notin Vars$, $r_2 = \mathrm{rt}(t,\sigma)[p] \notin Vars$ and r_1 and r_2 have different principal functors.

Then, by definition of 'rt', there must exists an index $i \in \mathbb{N}$ such that one of these holds:

1. $x = s\sigma^i[p] \in Vars \setminus \mathrm{dom}(\sigma)$, $y = t\sigma^i[p] \in Vars \setminus \mathrm{dom}(\sigma)$ and $x \neq y$; or

2. $x = s\sigma^i[p] \in Vars \setminus \mathrm{dom}(\sigma)$ and $r = t\sigma^i[p] \notin Vars$ or, symmetrically, we have $r = s\sigma^i[p] \notin Vars$ and $x = t\sigma^i[p] \in Vars \setminus \mathrm{dom}(\sigma)$; or

3. $r_1 = s\sigma^i[p] \notin Vars$ and $r_2 = t\sigma^i[p] \notin Vars$ have different principal functors.

By Lemma 3.20, we have $T \vdash \forall\bigl(\sigma \rightarrow (s\sigma^i = t\sigma^i)\bigr)$; from this, by the identity axioms, we obtain that

$$T \vdash \forall\Bigl(\sigma \rightarrow \bigl(s\sigma^i[p] = t\sigma^i[p]\bigr)\Bigr). \tag{3.14}$$

We now prove that each case leads to a contradiction.

Consider case 1. Let $r_1, r_2 \in GTerms \cap HTerms$ be two ground and finite terms having different principal functors, so that $T \vdash \forall(r_1 \neq r_2)$. By Lemma 3.19, we have that $\sigma' = \sigma \cup \{x \mapsto r_1, y \mapsto r_2\} \in RSubst$ is satisfiable; moreover, $T \vdash \forall(\sigma' \rightarrow \sigma)$, $T \vdash \forall\bigl(\sigma' \rightarrow (x = r_1)\bigr)$ and $T \vdash \forall\bigl(\sigma' \rightarrow (y = r_2)\bigr)$. This is a contradiction, since, by (3.14), we have $T \vdash \forall\bigl(\sigma \rightarrow (x = y)\bigr)$.

Consider case 2. Without loss of generality, consider the first subcase, where $x = s\sigma^i$ and $r = t\sigma^i[p] \notin Vars$. Let $r' \in GTerms \cap HTerms$ be such that r and r' have different principal functors, so that $T \vdash \forall(r \neq r')$. By Lemma 3.19, $\sigma' = \sigma \cup \{x \mapsto r'\} \in RSubst$ is satisfiable; we also have $T \vdash \forall(\sigma' \rightarrow \sigma)$ and $T \vdash \forall\bigl(\sigma' \rightarrow (x = r')\bigr)$. This is a contradiction, since, by (3.14), $T \vdash \forall\bigl(\sigma \rightarrow (x = r)\bigr)$.

Finally, consider case 3. In this case $T \vdash \forall(r_1 \neq r_2)$. This immediately leads to a contradiction, since, by (3.14), $T \vdash \forall\bigl(\sigma \rightarrow (r_1 = r_2)\bigr)$. □

Proof of Proposition 3.30 on page 58. By Theorem 3.16, there exists $\tau \in VSubst$ such that $\sigma \overset{\mathcal{S}}{\longmapsto}{}^{*} \tau$. By Theorem 3.15, $\mathrm{dom}(\sigma) = \mathrm{dom}(\tau)$ and $T \vdash \forall(\sigma \leftrightarrow \tau)$. By Lemma 3.36, $\mathrm{occ}(\sigma, v) = \mathrm{occ}(\tau, v)$. Moreover, by Lemma 3.38, we have $y \in \mathrm{occ}(\tau, v)$ if and only if

$v \in \operatorname{vars}\bigl(\operatorname{rt}(y, \tau)\bigr)$. Thus, we have $y \in \operatorname{occ}(\sigma, v)$ if and only if $v \in \operatorname{vars}\bigl(\operatorname{rt}(y, \tau)\bigr)$ and, to complete the proof, it is sufficient to show that

$$v \in \operatorname{vars}\bigl(\operatorname{rt}(y, \sigma)\bigr) \qquad \Longleftrightarrow \qquad v \in \operatorname{vars}\bigl(\operatorname{rt}(y, \tau)\bigr).$$

We only prove the first implication, since the other one follows by symmetry.

Suppose $v \in \operatorname{vars}\bigl(\operatorname{rt}(y, \sigma)\bigr)$. Then there exists an index $i \geq 0$ such that $v \in \operatorname{vars}(y\sigma^i)$. Note that $v \notin \operatorname{dom}(\sigma) = \operatorname{dom}(\tau)$, so that $v \in \operatorname{vars}\bigl(\operatorname{rt}(y\sigma^i, \tau)\bigr)$. Since $T \vdash \forall(\tau \to \sigma)$, by Lemma 3.20 we have $T \vdash \forall\bigl(\tau \to (y = y\sigma^i)\bigr)$. By Lemma 3.39, $\operatorname{rt}(y, \tau) = \operatorname{rt}(y\sigma^i, \tau)$. Thus, $v \in \operatorname{vars}\bigl(\operatorname{rt}(y, \tau)\bigr)$. □

Proof of Theorem 3.34 on page 59. All but the last property follow by Corollary 3.17. The last property, $\alpha(\sigma, \mathit{VI}) = \alpha(\sigma', \mathit{VI})$, follows by Corollary 3.37. □

To prove Theorem 3.35, we need to show that the abstraction function α is invariant when we exchange equivalent variables to obtain an ordered substitution.

Lemma 3.40 *Let $\sigma \in \mathit{VSubst}$, $v, w \in \mathit{Vars}$ and $(v \mapsto w) \in \sigma$. Let $\rho = \{v \mapsto w, w \mapsto v\}$ and define $\sigma' = \rho \circ \sigma = \{\, x\rho \mapsto t\rho \mid x \mapsto t \in \sigma \,\}$. Then*

1. *$\sigma' \in \mathit{VSubst}$,*
2. *$\operatorname{vars}(\sigma) = \operatorname{vars}(\sigma')$,*
3. *$\alpha(\sigma, \mathit{VI}) = \alpha(\sigma', \mathit{VI})$, for all $\mathit{VI} \in \wp_{\mathrm{f}}(\mathit{Vars})$, and*
4. *$T \vdash \forall(\sigma \leftrightarrow \sigma')$, for any equality theory T.*

Proof. Since σ' is obtained from σ by renaming variables and $\sigma \in \mathit{VSubst}$, we have also that $\sigma' \in \mathit{VSubst}$. Also, $\operatorname{vars}(\sigma) \setminus \{v, w\} = \operatorname{vars}(\sigma') \setminus \{v, w\}$ so that, since $(v \mapsto w) \in \sigma$ and $(w \mapsto v) \in \sigma'$, we have $\operatorname{vars}(\sigma) = \operatorname{vars}(\sigma')$.

To prove property 3, we have to show that, if

$$\alpha(\sigma, \mathit{VI}) \stackrel{\text{def}}{=} (\mathit{sh}, \mathit{VI}) \qquad \text{and} \qquad \alpha(\sigma', \mathit{VI}) \stackrel{\text{def}}{=} (\mathit{sh}', \mathit{VI}),$$

then $\mathit{sh} = \mathit{sh}'$. By the hypothesis, for all $y \in \mathit{Vars}$ we have $x \in \operatorname{vars}(y\sigma)$ if and only if $x\rho \in \operatorname{vars}(y\sigma')$. As $\sigma, \sigma' \in \mathit{VSubst}$, we can use the alternative characterization of 'occ' given by Lemma 3.28 and conclude that, for each $x \in \mathit{Vars}$, $\operatorname{occ}(\sigma, x) = \operatorname{occ}(\sigma', x\rho)$. Therefore $\mathit{sh} \subseteq \mathit{sh}'$. The reverse inclusion follows by symmetry so that $\mathit{sh} = \mathit{sh}'$.

To prove property 4, we first show by induction on the size of $r \in \mathit{HTerms}$ that

$$T \vdash \forall\bigl((v = w) \to (r = r\rho)\bigr). \tag{3.15}$$

For the base case, $\operatorname{size}(r) = 1$. If r is a constant or a variable z such that $z \neq v$ and $z \neq w$, then we have $r = r\rho$. If $r = v$, then $r\rho = w$ and $T \vdash \forall\bigl((v = w) \to (v = w)\bigr)$. Finally, if $r = w$, then $r\rho = v$ and, using the congruence axioms, $T \vdash \forall\bigl((v = w) \to (w = v)\bigr)$.

For the inductive step, let $r = f(r_1, \dots, r_n)$. Then $r\rho = f(r_1\rho, \dots, r_n\rho)$. Thus, using the inductive hypothesis, for each $i = 1, \dots, n$, $T \vdash \forall\bigl((v = w) \rightarrow (r_i = r_i\rho)\bigr)$. Hence, by the congruence axioms, (3.15) holds.

Note that $(v \mapsto w) \in \sigma$. Therefore, it follows from (3.15) that, for each $(x \mapsto t) \in \sigma$, $T \vdash \forall\bigl(\sigma \rightarrow \{x = t, x = x\rho, t = t\rho\}\bigr)$ and, using the congruence axioms, we obtain $T \vdash \forall\bigl(\sigma \rightarrow \{x\rho = t\rho\}\bigr)$. Thus, $T \vdash \forall(\sigma \rightarrow \sigma')$. Since $(w \mapsto v) \in \sigma'$, the reverse implication follows by symmetry so that $T \vdash \forall(\sigma' \leftrightarrow \sigma)$. $\square$

Lemma 3.41 *Let $\sigma \in \mathit{VSubst}$. Then there exists an ordered substitution $\sigma' \in \mathit{VSubst}$ such that* $\mathrm{vars}(\sigma) = \mathrm{vars}(\sigma')$, $\alpha(\sigma, \mathit{VI}) = \alpha(\sigma', \mathit{VI})$, *for all $\mathit{VI} \in \wp_{\mathrm{f}}(\mathit{Vars})$, and $T \vdash \forall(\sigma \leftrightarrow \sigma')$, for any equality theory T.*

Proof. The proof is by induction on the number $b \geq 0$ of the bindings $(v \mapsto w) \in \sigma$ such that $w \in \mathrm{param}(\sigma)$ and $w > v$ (the number of *unordered* bindings). For the base case, when $b = 0$, σ is ordered and the result holds by taking $\sigma' = \sigma$.

For the inductive case, when $b > 0$, let $(v \mapsto w) \in \sigma$ be an unordered binding and define $\rho = \{v \mapsto w, w \mapsto v\}$. Then, by Lemma 3.40, we have $\rho \circ \sigma \in \mathit{VSubst}$, $\mathrm{vars}(\sigma) = \mathrm{vars}(\rho \circ \sigma)$, $\alpha(\sigma, \mathit{VI}) = \alpha(\rho \circ \sigma, \mathit{VI})$, for all $\mathit{VI} \in \wp_{\mathrm{f}}(\mathit{Vars})$, and, finally, $T \vdash \forall(\sigma \leftrightarrow \rho \circ \sigma)$, for any equality theory T. In order to apply the inductive hypothesis to $\rho \circ \sigma$, we must show that the number of unordered bindings in $\rho \circ \sigma$ is less than b. To this end, roughly speaking, we start showing that any ordered binding in σ is mapped by ρ into another ordered binding in $\rho \circ \sigma$, therefore proving that the number of unordered bindings is not increasing. There are three cases. First, any ordered binding $(y \mapsto t) \in \sigma$ such that $t \notin \mathit{Vars}$ is mapped by ρ into the binding $(y\rho \mapsto t\rho) \in (\rho \circ \sigma)$ which is clearly ordered, since $t\rho \notin \mathit{Vars}$. Second, consider any ordered binding $(y \mapsto z) \in \sigma$ such that $z \in \mathrm{dom}(\sigma)$. Since $w \in \mathrm{param}(\sigma)$, we have $z \neq w$. If also $z \neq v$ then we have $z\rho = z$ and $z \in \mathrm{dom}(\rho \circ \sigma)$; otherwise $z = v$ so that $z\rho = w$ and, as $(w \mapsto v) \in (\rho \circ \sigma)$, $z\rho \in \mathrm{dom}(\rho \circ \sigma)$. Thus, in either case, such a binding is mapped by ρ into the binding $(y\rho \mapsto z\rho) \in (\rho \circ \sigma)$ which is ordered since $z\rho \in \mathrm{dom}(\rho \circ \sigma)$. Third, consider any ordered binding $(y \mapsto z) \in \sigma$ such that $z \in \mathrm{param}(\sigma)$ and $z < y$. The ordering relation implies $y \neq v$ and we also have $y \neq w$, since $w \in \mathrm{param}(\sigma)$. Hence, we obtain $y\rho = y$. Now, as $z \in \mathrm{param}(\sigma)$, $z \neq v$. If $z \neq w$, then $z\rho = z$. On the other hand, if $z = w$, then $z\rho = v$ so that $z\rho < z$. Thus, in both cases, as $z < y$, $z\rho < y$ and hence, $(y\rho \mapsto z\rho) \in (\rho \circ \sigma)$ is ordered. Finally, to show that the number of unordered bindings is strictly decreasing, we note that the unordered binding $(v \mapsto w) \in \sigma$ is mapped by ρ into the binding $(w \mapsto v) \in (\rho \circ \sigma)$, which is ordered.

Therefore, by applying the inductive hypothesis, there exists a substitution σ' such that $\sigma' \in \mathit{VSubst}$ is ordered, $\mathrm{vars}(\rho \circ \sigma) = \mathrm{vars}(\sigma')$, $\alpha(\rho \circ \sigma, \mathit{VI}) = \alpha(\sigma', \mathit{VI})$, for all $\mathit{VI} \in \wp_{\mathrm{f}}(\mathit{Vars})$, and $T \vdash \forall(\rho \circ \sigma \leftrightarrow \sigma')$, for any equality theory T. Then the required result follows by transitivity. $\square$

Proof of Theorem 3.35 on page 60. By Theorem 3.34, we can assume $\sigma, \sigma' \in \mathit{VSubst}$, $T \vdash \forall(\sigma \leftrightarrow \sigma')$ and $\alpha(\sigma, \mathit{VI}) = \alpha(\sigma', \mathit{VI})$, for any $\mathit{VI} \in \wp_{\mathrm{f}}(\mathit{Vars})$. By Lemma 3.41, we can also assume that σ and σ' are ordered substitutions so that, by Lemma 3.11, $\mathrm{dom}(\sigma') = \mathrm{dom}(\sigma)$.

We prove the result by showing that, for all $v \in \mathit{Vars}$, both $\mathrm{occ}(\sigma, v) \subseteq \mathrm{occ}(\sigma', v)$ and $\mathrm{occ}(\sigma', v) \subseteq \mathrm{occ}(\sigma, v)$. We just prove the first of these as the other case is symmetric.

Suppose that $w \in \mathit{Vars}$ and that $v \in \mathrm{vars}(w\sigma) \setminus \mathrm{dom}(\sigma)$. Then, using the alternative characterization of 'occ' for variable-idempotent substitutions given by Lemma 3.28, we just have to show that $v \in \mathrm{vars}(w\sigma') \setminus \mathrm{dom}(\sigma')$.

By Lemma 3.12 (replacing τ by σ, σ by σ' and $s = t$ by $w = w$), we have that there exists $z \in \mathrm{vars}(w\sigma') \setminus \mathrm{dom}(\sigma')$ such that $v \in \mathrm{vars}(z\sigma)$. Therefore, as $\mathrm{dom}(\sigma') = \mathrm{dom}(\sigma)$, $z \notin \mathrm{dom}(\sigma)$, and hence, $v = z$ so that $v \in \mathrm{vars}(w\sigma') \setminus \mathrm{dom}(\sigma')$, as required. □

3.4 Abstract Unification

The operation of abstract unification and the results stating its correctness, idempotence and commutativity are presented here in three stages. In the first two stages, we consider substitutions containing just a single binding. For the first, it is assumed that the set of variables of interest is fixed so that the definition is based on the SH domain. Then, in the second, using the SS domain, the definition is extended to allow for the introduction of new variables in the binding. The final stage extends this definition further to deal with arbitrary substitutions.

3.4.1 Abstract Operations for Sharing Sets

The abstract unifier amgu abstracts the effect of a single binding on an element of the SH domain. For this we need some ancillary definitions.

Definition 3.42 (Auxiliary functions.) *The* closure under union *function (also called* star-union*),* $(\cdot)^\star \colon \mathit{SH} \to \mathit{SH}$*, is given, for each* $sh \in \mathit{SH}$*, by*

$$sh^\star \stackrel{\mathrm{def}}{=} \bigl\{\, S \in \mathit{SG} \bigm| \exists n \geq 1 \,.\, \exists S_1, \ldots, S_n \in sh \,.\, S = S_1 \cup \cdots \cup S_n \,\bigr\}.$$

For each $sh \in \mathit{SH}$ *and each* $V \in \wp_{\mathrm{f}}(\mathit{Vars})$*, the extraction of the* relevant component of sh with respect to V *is encoded by* $\mathrm{rel} \colon \wp_{\mathrm{f}}(\mathit{Vars}) \times \mathit{SH} \to \mathit{SH}$ *defined as*

$$\mathrm{rel}(V, sh) \stackrel{\mathrm{def}}{=} \{\, S \in sh \mid S \cap V \neq \varnothing \,\};$$

the irrelevant component of sh with respect to V *is thus defined as*

$$\overline{\mathrm{rel}}(V, sh) \stackrel{\mathrm{def}}{=} sh \setminus \mathrm{rel}(V, sh).$$

For each $sh_1, sh_2 \in \mathit{SH}$*, the* binary union *function* $\mathrm{bin} \colon \mathit{SH} \times \mathit{SH} \to \mathit{SH}$ *is given by*

$$\mathrm{bin}(sh_1, sh_2) \stackrel{\mathrm{def}}{=} \{\, S_1 \cup S_2 \mid S_1 \in sh_1, S_2 \in sh_2 \,\}.$$

Definition 3.43 (amgu.) *The function* amgu: $SH \times Bind \to SH$ *captures the effects of a binding on an SH element. Suppose* $x \in Vars$, $r \in HTerms$, *and* $sh \in SH$. *Let* $v_x \stackrel{\mathrm{def}}{=} \{x\}$, $v_r \stackrel{\mathrm{def}}{=} \mathrm{vars}(r)$ *and* $v_{xr} \stackrel{\mathrm{def}}{=} v_x \cup v_r$. *Then*

$$\mathrm{amgu}(sh, x \mapsto r) \stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}(v_{xr}, sh) \cup \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh)^\star, \mathrm{rel}(v_r, sh)^\star\bigr).$$

The following correctness result for amgu is proved in Section 3.4.4.

Theorem 3.44 *Let* $\sigma \in RSubst$ *and* $(x \mapsto r) \in Bind$ *be such that* $\mu \in \mathrm{mgs}\bigl(\{x = r\} \cup \sigma\bigr)$ *in the syntactic equality theory* T*; let also* $(sh, VI) \in SS$, *where* $\mathrm{vars}(x \mapsto r) \cup \mathrm{vars}(\sigma) \subseteq VI$. *Then*

$$\alpha(\sigma, VI) \preceq_{SS} (sh, VI) \implies \alpha(\mu, VI) \preceq_{SS} \bigl(\mathrm{amgu}(sh, x \mapsto r), VI\bigr).$$

The following theorems, proved in Section 3.4.4, show that amgu is idempotent and commutative.

Theorem 3.45 *Let* $sh \in SH$ *and* $(x \mapsto r) \in Bind$. *Then*

$$\mathrm{amgu}(sh, x \mapsto r) = \mathrm{amgu}\bigl(\mathrm{amgu}(sh, x \mapsto r), x \mapsto r\bigr).$$

Theorem 3.46 *Let* $sh \in SH$ *and* $(x \mapsto r), (y \mapsto t) \in Bind$. *Then*

$$\mathrm{amgu}\bigl(\mathrm{amgu}(sh, x \mapsto r), y \mapsto t\bigr) = \mathrm{amgu}\bigl(\mathrm{amgu}(sh, y \mapsto t), x \mapsto r\bigr).$$

3.4.2 Considering the Variables of Interest

The definitions and results of Section 3.4.1 can be lifted to apply to the proper set-sharing domain.

Definition 3.47 (Amgu.) *The operation* Amgu: $SS \times Bind \to SS$ *extends the SS description it takes as an argument to the set of variables occurring in the binding it is given as the second argument; then it applies* amgu. *Formally:*

$$U \stackrel{\mathrm{def}}{=} \mathrm{vars}(x \mapsto r) \setminus VI,$$
$$\mathrm{Amgu}\bigl((sh, VI), x \mapsto r\bigr) \stackrel{\mathrm{def}}{=} \Bigl(\mathrm{amgu}\Bigl(sh \cup \bigl\{\, \{u\} \bigm| u \in U \,\bigr\}, x \mapsto r\Bigr), VI \cup U\Bigr).$$

The results for amgu can easily be extended to apply to Amgu giving us the following corollaries.

Corollary 3.48 *Let* $\sigma \in RSubst$ *and* $(x \mapsto r) \in Bind$ *be such that* $\mu \in \mathrm{mgs}\bigl(\{x = r\} \cup \sigma\bigr)$ *in the syntactic equality theory* T*; let also* $(sh, VI) \in SS$, *where* $U = \mathrm{vars}(x \mapsto r)$ *and* $\mathrm{vars}(\sigma) \subseteq VI$. *Then*

$$\alpha(\sigma, VI) \preceq_{SS} (sh, VI) \implies \alpha\bigl(\mu, VI \cup U\bigr) \preceq_{SS} \mathrm{Amgu}\bigl((sh, VI), x \mapsto r\bigr).$$

Corollary 3.49 *Let $sh \in SH$ and $(x \mapsto r) \in Bind$. Then*

$$\mathrm{Amgu}\big((sh, VI), x \mapsto r\big) = \mathrm{Amgu}\Big(\mathrm{Amgu}\big((sh, VI), x \mapsto r\big), x \mapsto r\Big).$$

Corollary 3.50 *Let $sh \in SH$ and $(x \mapsto r), (y \mapsto t) \in Bind$. Then*

$$\mathrm{Amgu}\Big(\mathrm{Amgu}\big((sh, VI), x \mapsto r\big), y \mapsto t\Big) = \mathrm{Amgu}\Big(\mathrm{Amgu}\big((sh, VI), y \mapsto t\big), x \mapsto r\Big).$$

3.4.3 Abstract Unification for Set-Sharing

We now extend the above definitions and results for a single binding to any substitution.

Definition 3.51 (Aunify.**)** *The function* Aunify: $SS \times RSubst \to SS$ *generalizes* Amgu *to any substitution $\mu \in RSubst$ in the context of some syntactic equality theory T: for each $(sh, VI) \in SS$ and each μ satisfiable in T, where $(x \mapsto r) \in \mu$,*

$$\mathrm{Aunify}\big((sh, VI), \varnothing\big) \stackrel{\mathrm{def}}{=} (sh, VI);$$
$$\mathrm{Aunify}\big((sh, VI), \mu\big) \stackrel{\mathrm{def}}{=} \mathrm{Aunify}\Big(\big(\mathrm{Amgu}(sh, VI), x \mapsto r\big), \mu \setminus \{x \mapsto r\}\Big);$$

and, if μ is not satisfiable in T,

$$\mathrm{Aunify}\big((sh, VI), \mu\big) \stackrel{\mathrm{def}}{=} \bot.$$

For the distinguished elements $\bot$ and $\top$ of SS,

$$\mathrm{Aunify}(\bot, \mu) \stackrel{\mathrm{def}}{=} \bot,$$
$$\mathrm{Aunify}(\top, \mu) \stackrel{\mathrm{def}}{=} \top.$$

As a result of Corollary 3.50, Amgu and Aunify commute.

Lemma 3.52 *Let $(sh, VI) \in SS$, $\nu \in RSubst$ and $(y \mapsto t) \in Bind$. Then*

$$\mathrm{Aunify}\Big(\mathrm{Amgu}\big((sh, VI), y \mapsto t\big), \nu\Big) = \mathrm{Amgu}\Big(\mathrm{Aunify}\big((sh, VI), \nu\big), y \mapsto t\Big).$$

As a consequence of this and Corollaries 3.48, 3.49 and 3.50, we have the following correctness, idempotence and commutativity results required for Aunify to be sound and well-defined.

Theorem 3.53 *Let $(sh, VI) \in SS$ and $\sigma, \nu \in RSubst$, where* $\mathrm{vars}(\sigma) \subseteq VI$. *Suppose also that $\mu \in \mathrm{mgs}(\nu \cup \sigma)$ in the syntactic equality theory T. Then*

$$\alpha(\sigma, VI) \preceq_{SS} (sh, VI) \implies \alpha\big(\mu, VI \cup \mathrm{vars}(\nu)\big) \preceq_{SS} \mathrm{Aunify}\big((sh, VI), \mu\big).$$

This theorem shows also that it is safe for the analyzer to perform part or all of the concrete unification algorithm before computing Aunify.

Theorem 3.54 *Let* $(sh, VI) \in SS$ *and* $\nu \in RSubst$. *Then*

$$\mathrm{Aunify}\big((sh, VI), \nu\big) = \mathrm{Aunify}\Big(\mathrm{Aunify}\big((sh, VI), \nu\big), \nu\Big).$$

Theorem 3.55 *Let* $(sh, VI) \in SS$ *and* $\nu_1, \nu_2 \in RSubst$. *Then*

$$\mathrm{Aunify}\Big(\mathrm{Aunify}\big((sh, VI), \nu_1\big), \nu_2\Big) = \mathrm{Aunify}\Big(\mathrm{Aunify}\big((sh, VI), \nu_2\big), \nu_1\Big).$$

The proofs of all these results are in Section 3.4.5.

3.4.4 Proofs of the Results of Section 3.4.1

In the proofs we use the fact that $(\cdot)^\star$ and rel are monotonic so that, for all $sh_1, sh_2 \in SH$ and $V \in \wp_{\mathrm{f}}(Vars)$,

$$sh_1 \subseteq sh_2 \implies sh_1^\star \subseteq sh_2^\star, \tag{3.16}$$

$$sh_1 \subseteq sh_2 \implies \mathrm{rel}(V, sh_1) \subseteq \mathrm{rel}(V, sh_2). \tag{3.17}$$

We will also use the fact that $(\cdot)^\star$ is idempotent.

Let $t_1, \ldots, t_n$ be terms. For the sake of brevity we will use the notation $v_{t_1 \cdots t_n}$ to denote $\bigcup_{i=1}^{n} \mathrm{vars}(t_i)$. In particular, if x and y are variables, and r and t are terms, we will use the following definitions:

$$v_x \stackrel{\mathrm{def}}{=} \{x\}, \qquad v_r \stackrel{\mathrm{def}}{=} \mathrm{vars}(r), \qquad v_{xr} \stackrel{\mathrm{def}}{=} v_x \cup v_r,$$

$$v_y \stackrel{\mathrm{def}}{=} \{y\}, \qquad v_t \stackrel{\mathrm{def}}{=} \mathrm{vars}(t), \qquad v_{yt} \stackrel{\mathrm{def}}{=} v_y \cup v_t.$$

Proof of Theorem 3.44 on page 67. We first prove the result under the assumption that $\alpha(\sigma, VI) = (sh, VI)$. We do this in two parts. In the first, we partition σ into two substitutions one of which, called σ^-, is the same as σ when σ and μ are idempotent. We construct a new substitution ν which, in the case that σ and μ are idempotent, is a most general solution for $x\sigma = r\sigma$. Finally we compose ν with σ^- to define a substitution that has the same abstraction as μ but with a number of useful properties including that of variable-idempotence. In the second part, we use this composed substitution in place of μ to prove the result.

Part 1. By Theorem 3.34, we can assume that

$$\sigma \in VSubst, \tag{3.18}$$

$$y \in \mathrm{dom}(\sigma) \cap \mathrm{range}(\sigma) \implies y \in \mathrm{vars}(y\sigma). \tag{3.19}$$

Let $\sigma^\circ, \sigma^- \in RSubst$ be defined such that

$$\sigma^\circ = \big\{\, (y \mapsto t) \in \sigma \;\big|\; y \in \mathrm{vars}(x\sigma = r\sigma) \,\big\}, \tag{3.20}$$

$$\sigma^- = \sigma \setminus \sigma^\circ. \tag{3.21}$$

Then, it follows from (3.19) and Lemma 3.10 that

$$\sigma^{\circ} \in VSubst, \qquad \sigma^{-} \in VSubst. \tag{3.22}$$

Suppose $z \in \mathrm{vars}(\sigma^{\circ}) \setminus \mathrm{dom}(\sigma^{\circ})$. Then $z \in \mathrm{vars}(y\sigma^{\circ})$ for some $y \in \mathrm{dom}(\sigma^{\circ})$. By (3.20), $z \in \mathrm{vars}(x\sigma\sigma = r\sigma\sigma) \setminus \mathrm{dom}(\sigma^{\circ})$. By (3.18), $z \in \mathrm{vars}(x\sigma = r\sigma)$. Thus, as z was an arbitrary variable in $\mathrm{vars}(\sigma^{\circ}) \setminus \mathrm{dom}(\sigma^{\circ})$,

$$\mathrm{vars}(\sigma^{\circ}) \subseteq \mathrm{vars}(x\sigma = r\sigma). \tag{3.23}$$

Combining (3.20), (3.21), and (3.23), we have

$$\mathrm{dom}(\sigma^{-}) \cap \mathrm{vars}(\sigma^{\circ}) = \varnothing. \tag{3.24}$$

Let $\nu \in \mathrm{mgs}\bigl(\{x\sigma = r\sigma\} \cup \sigma^{\circ}\bigr)$ in T so that

$$T \vdash \forall\bigl(\nu \leftrightarrow \{x\sigma = r\sigma\} \cup \sigma^{\circ}\bigr), \tag{3.25}$$

$$\mathrm{vars}(\nu) \subseteq \mathrm{vars}(x\sigma = r\sigma) \cup \mathrm{vars}(\sigma^{\circ}). \tag{3.26}$$

By Theorem 3.34, we can assume that

$$\nu \in VSubst. \tag{3.27}$$

By (3.20), (3.24), and (3.26), we have

$$\mathrm{dom}(\sigma^{-}) \cap \mathrm{vars}(\nu) = \varnothing. \tag{3.28}$$

Therefore, as $\sigma^{-}, \nu \in VSubst$ (by (3.22) and (3.27)), we can use Lemma 3.18 to obtain the following properties for $\nu \circ \sigma^{-}$.

$$T \vdash \forall\bigl((\nu \circ \sigma^{-}) \leftrightarrow (\nu \cup \sigma^{-})\bigr), \tag{3.29}$$

$$\mathrm{dom}(\nu \circ \sigma^{-}) = \mathrm{dom}(\sigma^{-}) \cup \mathrm{dom}(\nu), \tag{3.30}$$

$$\nu \circ \sigma^{-} \in VSubst. \tag{3.31}$$

Now we have

$$
\begin{aligned}
&T \vdash \forall\bigl(\mu \leftrightarrow \{x = r\} \cup \sigma\bigr) && \text{[by hypothesis]}\\
&T \vdash \forall\bigl(\mu \leftrightarrow \{x\sigma = r\sigma\} \cup \sigma\bigr) && \text{[by Lemma 3.20]}\\
&T \vdash \forall\bigl(\mu \leftrightarrow \nu \cup \sigma^{-}\bigr) && \text{[by (3.21), (3.21) and (3.25)]}\\
&T \vdash \forall\bigl(\mu \leftrightarrow \nu \circ \sigma^{-}\bigr) && \text{[by (3.29)].}
\end{aligned} \tag{3.32}
$$

Therefore, by Theorem 3.35,

$$\alpha(\mu, VI) = \alpha(\nu \circ \sigma^{-}, VI). \tag{3.33}$$

Part 2. To prove the result under the assumption $\alpha(\sigma, VI) = (sh, VI)$, we define $sh' \in SH$ so that

$$\alpha(\mu, VI) = (sh', VI). \tag{3.34}$$

By (3.33), it holds $\alpha(\nu \circ \sigma^-, VI) = (sh', VI)$. We show that $sh' \subseteq \mathrm{amgu}(sh, x \mapsto r)$. If $sh' = \varnothing$, there is nothing to prove. Therefore, we assume that there exists $S \in sh'$ so that $S \neq \varnothing$ and, for some $v \in Vars$,

$$v \notin \mathrm{dom}(\nu \circ \sigma^-), \tag{3.35}$$

$$S \stackrel{\text{def}}{=} \mathrm{occ}(\nu \circ \sigma^-, v). \tag{3.36}$$

Note that (3.30) and (3.35) imply that

$$v \notin \mathrm{dom}(\nu), \qquad v \notin \mathrm{dom}(\sigma^-). \tag{3.37}$$

Let

$$S' \stackrel{\text{def}}{=} \bigcup \{\, \mathrm{occ}(\sigma, y) \mid y \in \mathrm{occ}(\nu, v) \,\}. \tag{3.38}$$

We show that

$$S = S'. \tag{3.39}$$

By (3.18), (3.27) and (3.31), $\sigma, \nu, \nu\circ\sigma^- \in VSubst$ and, by (3.35) and (3.37), $v \notin \mathrm{dom}(\nu\circ\sigma^-)$ and $v \notin \mathrm{dom}(\nu)$. Thus, it follows from Lemma 3.28 with (3.36) and (3.38), that it suffices to show that, for each $w \in Vars$, $v \in \mathrm{vars}(w\sigma^-\nu)$ if and only if there exists $z \in \mathrm{vars}(w\sigma) \setminus \mathrm{dom}(\sigma)$ such that $v \in \mathrm{vars}(z\nu)$.

First, we suppose that $v \in \mathrm{vars}(w\sigma^-\nu)$. Thus, there exists $y \in \mathrm{vars}(w\sigma^-)$ such that $v \in \mathrm{vars}(y\nu)$. Since $\sigma^\circ, \nu \in VSubst$, $T \vdash \forall(\nu \to \sigma^\circ)$ (by (3.25)), $v \notin \mathrm{dom}(\nu)$ (by (3.37)) and $T \vdash \forall\bigl(\nu \to (y\nu = y)\bigr)$ (using Lemma 3.20), we can apply Lemma 3.12 (replacing τ by ν, σ by σ° and $s = t$ by $y\nu = y$) so that there exists $z \in \mathrm{vars}(y\sigma^\circ) \setminus \mathrm{dom}(\sigma^\circ)$ such that $v \in \mathrm{vars}(z\nu)$. We want to show that $z \in \mathrm{vars}(w\sigma) \setminus \mathrm{dom}(\sigma)$. Now either $z \in \mathrm{dom}(\nu)$ or $z = v$ so that, by (3.28) (if $z \in \mathrm{dom}(\nu)$) or (3.37) (if $z = v$), $z \notin \mathrm{dom}(\sigma^-)$. However, $z \notin \mathrm{dom}(\sigma^\circ)$, so that $z \notin \mathrm{dom}(\sigma)$. Thus, it remains to prove that $z \in \mathrm{vars}(w\sigma)$. Now, as $y \in \mathrm{vars}(w\sigma^-)$ and $z \in \mathrm{vars}(y\sigma^\circ)$, we have $z \in \mathrm{vars}(w\sigma^-\sigma^\circ)$. So we must show that $\mathrm{vars}(w\sigma^-\sigma^\circ) \setminus \mathrm{dom}(\sigma) \subseteq \mathrm{vars}(w\sigma)$. To see this note that, if $w \notin \mathrm{dom}(\sigma^-)$, then $w\sigma^- = w$ and, by (3.20), $w\sigma^\circ = w\sigma$ so that $w\sigma^-\sigma^\circ = w\sigma$. On the other hand, if $w \in \mathrm{dom}(\sigma^-)$, then, by (3.21), $w\sigma^- = w\sigma$ so that $w\sigma^-\sigma^\circ = w\sigma\sigma^\circ$ Now, as $\sigma \in VSubst$ and $\sigma^\circ \subseteq \sigma$, we can apply Lemma 3.8 so that $\mathrm{vars}(w\sigma\sigma^\circ) \setminus \mathrm{dom}(\sigma) \subseteq \mathrm{vars}(w\sigma)$. Hence, $\mathrm{vars}(w\sigma^-\sigma^\circ) \setminus \mathrm{dom}(\sigma) \subseteq \mathrm{vars}(w\sigma)$.

Suppose now there exists $z \in \mathrm{vars}(w\sigma) \setminus \mathrm{dom}(\sigma)$ such that $v \in \mathrm{vars}(z\nu)$. Then we have $v \in \mathrm{vars}(w\sigma\nu)$. We need to show that $v \in \mathrm{vars}(w\sigma^-\nu)$. By (3.21), if $w \in \mathrm{dom}(\sigma^-)$ then $w\sigma\nu = w\sigma^-\nu$, so that $v \in \mathrm{vars}(w\sigma^-\nu)$. On the other hand, once again by (3.21), if $w \notin \mathrm{dom}(\sigma^-)$ then $v \in \mathrm{vars}(w\sigma^\circ\nu)$. Moreover, $w = w\sigma^-$ so that, by (3.25) and

Lemma 3.20 with the congruence axioms, $T \vdash \forall\big(\nu \to (w\sigma^{\circ}\nu = w\sigma^{-})\big)$. Hence, since $\nu \in \mathit{VSubst}$ and $v \notin \mathrm{dom}(\nu)$ (by (3.37)), we can apply Lemma 3.12 (replacing τ by ν, σ by the empty substitution and $s = t$ by $w\sigma^{\circ}\nu = w\sigma^{-}$) and obtain $v \in \mathrm{vars}(w\sigma^{-}\nu)$.

As a consequence of the previous two paragraphs, we have $S = S'$.

Let

$$S_x \stackrel{\mathrm{def}}{=} \bigcup\Big(\{\, \mathrm{occ}(\sigma, y) \mid y \in \mathrm{occ}(\nu, v) \,\} \cap \mathrm{rel}(v_x, sh)\Big), \tag{3.40}$$

$$S_r \stackrel{\mathrm{def}}{=} \bigcup\Big(\{\, \mathrm{occ}(\sigma, y) \mid y \in \mathrm{occ}(\nu, v) \,\} \cap \mathrm{rel}(v_r, sh)\Big), \tag{3.41}$$

$$S_0 \stackrel{\mathrm{def}}{=} \bigcup\Big(\{\, \mathrm{occ}(\sigma, y) \mid y \in \mathrm{occ}(\nu, v) \,\} \cap \overline{\mathrm{rel}}(v_{xr}, sh)\Big). \tag{3.42}$$

Note that by (3.38), (3.39) and the fact that

$$\overline{\mathrm{rel}}(v_{xr}, sh) = sh \setminus \big(\mathrm{rel}(v_x, sh) \cup \mathrm{rel}(v_r, sh)\big),$$

we have

$$S_0 = S \setminus (S_x \cup S_r). \tag{3.43}$$

We now consider the two cases $S_0 \neq \varnothing$ and $S_0 = \varnothing$ separately.

Consider first the case when $S_0 \neq \varnothing$. Then, by (3.42), for some $y \in \mathit{Vars}$,

$$y \in \mathrm{occ}(\nu, v), \tag{3.44}$$

$$\mathrm{occ}(\sigma, y) \in \overline{\mathrm{rel}}(v_{xr}, sh). \tag{3.45}$$

Thus, by Lemma 3.27, $y \notin \mathrm{dom}(\sigma)$ and hence, by (3.20), $y \notin \mathrm{dom}(\sigma^{\circ})$. Also, by (3.45), $\mathrm{occ}(\sigma, y) \cap v_{xr} = \varnothing$. Thus as $\sigma \in \mathit{VSubst}$ (by (3.18)) we can use Lemma 3.28 to see that, for each $w \in v_{xr}$, $y \notin \mathrm{vars}(w\sigma)$ and hence, $y \notin \mathrm{vars}(x\sigma = r\sigma)$. Therefore, by (3.23) and (3.26), $y \notin \mathrm{vars}(\nu)$. As $\nu \in \mathit{VSubst}$ (by (3.27)), we can apply Lemma 3.28 to both $\mathrm{occ}(\nu, y)$ and $\mathrm{occ}(\nu, v)$. Thus, as $y \notin \mathrm{vars}(\nu)$, $\mathrm{occ}(\nu, y) = \{y\}$ and also (using (3.44)) $v = y$ so that $\mathrm{occ}(\nu, v) = \{v\}$. It therefore follows from (3.38) and (3.39) that $S = \mathrm{occ}(\sigma, v)$ and hence from (3.45), that

$$S \in \overline{\mathrm{rel}}(v_{xr}, sh). \tag{3.46}$$

Now consider the case when $S_0 = \varnothing$. By (3.43), and the assumption that $S \neq \varnothing$,

$$S = S_x \cup S_r \neq \varnothing. \tag{3.47}$$

As a consequence of (3.40) and (3.41),

$$S_x \in \mathrm{rel}(v_x, sh)^{\star} \cup \varnothing, \tag{3.48}$$

$$S_r \in \mathrm{rel}(v_r, sh)^{\star} \cup \varnothing. \tag{3.49}$$

Now, by (3.47) $S_x \neq \varnothing$ or $S_r \neq \varnothing$. We will show that both $S_x \neq \varnothing$ and $S_r \neq \varnothing$. Suppose first that $S_x \neq \varnothing$. Then, by (3.48), $x \in S_x$. Hence, by (3.47), $x \in S$. By (3.36),

$x \in \mathrm{occ}(\nu \circ \sigma^-, v)$. However, $\nu \circ \sigma^- \in \mathit{VSubst}$ (by (3.31)) so that we can apply Lemma 3.28 to $\mathrm{occ}(\nu \circ \sigma^-, v)$ and obtain that $v \in \mathrm{vars}(x\sigma^-\nu)$. By the definition of μ in the hypothesis and (3.32), $T \vdash \forall\big(\nu \circ \sigma^- \to (x = r)\big)$ and hence, by Lemma 3.20 with the congruence axioms, $T \vdash \forall\big(\nu \circ \sigma^- \to (x\sigma^-\nu = r)\big)$. Thus, as $\nu \circ \sigma^- \in \mathit{VSubst}$ (by (3.31)) and $v \notin \mathrm{dom}(\nu \circ \sigma^-)$ (by (3.35)), by Lemma 3.12 (replacing τ by $\nu \circ \sigma^-$, σ by the empty substitution and $s = t$ by $x\sigma^-\nu = r$), we have $v \in \mathrm{vars}(r\sigma^-\nu)$. By re-applying Lemma 3.28 to $\mathrm{occ}(\nu \circ \sigma^-, v)$, it can be seen that, as $v \notin \mathrm{dom}(\nu)$ (by (3.35)), $v_r \cap \mathrm{occ}(\nu \circ \sigma^-, v) \neq \varnothing$. Hence, by (3.36), $S \cap v_r \neq \varnothing$. Thus, by (3.38) and (3.39), there exists a $y \in \mathrm{occ}(\nu, v)$ such that $\mathrm{occ}(\sigma, y) \cap v_r \neq \varnothing$. Therefore, by (3.41), $S_r \cap v_r \neq \varnothing$ and so $S_r \neq \varnothing$. By a similar argument, if $S_r \neq \varnothing$ then we have $S_x \neq \varnothing$. Hence $S_x \neq \varnothing$ and $S_r \neq \varnothing$ so that, by (3.48) and (3.49), $S_x \in \mathrm{rel}(v_x, sh)^\star$ and $S_r \in \mathrm{rel}(v_r, sh)^\star$. Therefore, we have, by (3.47),

$$S \in \mathrm{bin}\big(\mathrm{rel}(v_x, sh)^\star, \mathrm{rel}(v_r, sh)^\star\big). \tag{3.50}$$

Combining (3.46) when $S_0 \neq \varnothing$ and (3.50) when $S_0 = \varnothing$ we obtain

$$S \in \overline{\mathrm{rel}}(v_{xr}, sh) \cup \mathrm{bin}\big(\mathrm{rel}(v_x, sh)^\star, \mathrm{rel}(v_r, sh)^\star\big).$$

Therefore, by Definition 3.43, $S \in \mathrm{amgu}(sh, x \mapsto r)$.

As a consequence, since S was any set in sh', we have $sh' \subseteq \mathrm{amgu}(sh, x \mapsto r)$ and hence, by (3.34),

$$\alpha(\mu, \mathit{VI}) \preceq_{SS} \big(\mathrm{amgu}(sh, x \mapsto r), \mathit{VI}\big). \tag{3.51}$$

We now drop the assumption that $\alpha(\sigma, \mathit{VI}) = (sh, \mathit{VI})$ and just assume the hypothesis of the theorem that $\alpha(\sigma, \mathit{VI}) \preceq_{SS} (sh, \mathit{VI})$. Suppose $\alpha(\sigma, \mathit{VI}) = (sh_1, \mathit{VI})$. Then $sh_1 \subseteq sh$. It follows from Definition 3.43 that amgu is monotonic on its first argument so that

$$\mathrm{amgu}(sh_1, x \mapsto r) \subseteq \mathrm{amgu}(sh, x \mapsto r).$$

Thus, by (3.51) (replacing sh by sh_1), we obtain the required result

$$\alpha(\mu, \mathit{VI}) \preceq_{SS} \big(\mathrm{amgu}(sh, x \mapsto r), \mathit{VI}\big).$$

□

Lemma 3.56 *For each* $sh_1, sh_2 \in \mathit{SH}$, *we have* $\mathrm{bin}(sh_1, sh_2)^\star = \mathrm{bin}(sh_1^\star, sh_2^\star)$.

Proof. Suppose $S \in \mathit{SG}$. Then $S \in \mathrm{bin}(sh_1, sh_2)^\star$ means that, for some $n \in \mathbb{N}$, there exist sets $R_1, \ldots, R_n \in sh_1$ and $T_1, \ldots, T_n \in sh_2$ such that $S = (R_1 \cup T_1) \cup \cdots \cup (R_n \cup T_n)$. Thus $S = (R_1 \cup \cdots \cup R_n) \cup (T_1 \cup \cdots \cup T_n)$. However $R_1 \cup \cdots \cup R_n \in sh_1^\star$ and $T_1 \cup \cdots \cup T_n \in sh_2^\star$. Thus $S \in \mathrm{bin}(sh_1^\star, sh_2^\star)$.

On the other hand, $S \in \mathrm{bin}(sh_1^\star, sh_2^\star)$ means that $S = R \cup T$ where, for some $k, l \in \mathbb{N}$, some $R_1, \ldots, R_k \in sh_1$ and some $T_1, \ldots, T_l \in sh_2$, we have $R = R_1 \cup \cdots \cup R_k$ and $T = T_1 \cup \cdots \cup T_l$. Let n be the maximum of $\{k, l\}$ and suppose that, for each $i, j \in \mathbb{N}$

where $k+1 \leq i \leq n$ and $l+1 \leq j \leq n$, we define $R_i \stackrel{\text{def}}{=} R_k$ and $T_j \stackrel{\text{def}}{=} T_l$. Then, $S = (R_1 \cup T_1) \cup \cdots \cup (R_n \cup T_n)$. However, for $1 \leq i \leq n$, $R_i \cup T_i \in \mathrm{bin}(sh_1, sh_2)$. Thus $S \in \mathrm{bin}(sh_1, sh_2)^\star$. □

Proof of Theorem 3.45 on page 67. Let

$$\begin{aligned} sh_- &\stackrel{\text{def}}{=} \overline{\mathrm{rel}}(v_{xr}, sh),\\ sh_{xr} &\stackrel{\text{def}}{=} \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh)^\star, \mathrm{rel}(v_r, sh)^\star\bigr). \end{aligned}$$

Then, by Lemma 3.56, $sh_{xr}^\star = sh_{xr}$ and $\mathrm{bin}(sh_{xr}, sh_{xr}) = sh_{xr}$. Moreover,

$$\begin{aligned} \mathrm{rel}(v_x, sh_{xr}) &= sh_{xr}, & \mathrm{rel}(v_x, sh_-) &= \varnothing,\\ \mathrm{rel}(v_r, sh_{xr}) &= sh_{xr}, & \mathrm{rel}(v_r, sh_-) &= \varnothing,\\ \overline{\mathrm{rel}}(v_{xr}, sh_{xr}) &= \varnothing, & \overline{\mathrm{rel}}(v_{xr}, sh_-) &= sh_-. \end{aligned}$$

Hence, we have

$$\begin{aligned} \mathrm{rel}(v_x, sh_- \cup sh_{xr}) &= sh_{xr},\\ \mathrm{rel}(v_r, sh_- \cup sh_{xr}) &= sh_{xr},\\ \overline{\mathrm{rel}}(v_{xr}, sh_- \cup sh_{xr}) &= sh_-. \end{aligned}$$

Now, by Definition 3.43,

$$\begin{aligned} &\mathrm{amgu}\bigl(\mathrm{amgu}(sh, x \mapsto r), x \mapsto r\bigr)\\ &\quad= \overline{\mathrm{rel}}(v_{xr}, sh_- \cup sh_{xr}) \cup \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_- \cup sh_{xr})^\star, \mathrm{rel}(v_r, sh_- \cup sh_{xr})^\star\bigr)\\ &\quad= sh_- \cup sh_{xr}\\ &\quad= \mathrm{amgu}(sh, x \mapsto r). \end{aligned}$$

□

For the proof of commutativity, we require the following auxiliary results.

Lemma 3.57 *For each $V \in \wp_{\mathrm{f}}(\mathit{Vars})$ and $sh \in SH$ we have $\overline{\mathrm{rel}}(V, sh^\star) = \overline{\mathrm{rel}}(V, sh)^\star$.*

Proof. Let $S \in SG$. Then $S \in \overline{\mathrm{rel}}(V, sh^\star)$ means $S \in sh^\star$ and $S \cap V = \varnothing$. In other words, there exist $S_1, \ldots, S_n \in sh$ such that $S = \bigcup_{i=1}^n S_i$ and, for each $i = 1, \ldots, n$, we have $S_i \cap V = \varnothing$. This amounts to saying that there exist $S_1, \ldots, S_n \in \overline{\mathrm{rel}}(V, sh)$ such that $S = \bigcup_{i=1}^n S_i$, which is equivalent to $S \in \overline{\mathrm{rel}}(V, sh)^\star$. □

The auxiliary function 'rel' possesses a weaker property.

Lemma 3.58 *For each $V \in \wp_{\mathrm{f}}(\mathit{Vars})$ and $sh \in SH$ we have $\mathrm{rel}(V, sh^\star) \supseteq \mathrm{rel}(V, sh)^\star$.*

Proof. Let $S \in SG$. Then $S \in \mathrm{rel}(V, sh)^\star$ means that there exist $S_1, \ldots, S_n \in sh$ such that $S_i \cap V \neq \varnothing$, for each $i = 1, \ldots, n$, and $S = \bigcup_{i=1}^n S_i$. Thus $S \cap V \neq \varnothing$ and $S \in \mathrm{rel}(V, sh^\star)$. Hence, $\mathrm{rel}(V, sh^\star) \supseteq \mathrm{rel}(V, sh)^\star$. $\square$

Lemma 3.59 *For each $V \in \wp_\mathrm{f}(\mathit{Vars})$, $sh_1, sh_2 \in SH$ and $S \in \wp_\mathrm{f}(\mathit{Vars})$,*

$$\begin{aligned} &S \in \mathrm{rel}(V, sh_1 \cup sh_2)^\star \cup \{\varnothing\} \\ &\qquad\iff \exists S_1 \in \mathrm{rel}(V, sh_1)^\star \cup \{\varnothing\} \mathrel{.} \exists S_2 \in \mathrm{rel}(V, sh_2)^\star \cup \{\varnothing\} \mathrel{.} S = S_1 \cup S_2. \end{aligned}$$

Proof. If $S = \varnothing$ the statement is trivial.

Let $S \in \mathrm{rel}(V, sh_1 \cup sh_2)^\star$. For some $n \in \mathbb{N}$, there exists n sets $R_1, \ldots, R_n \in (sh_1 \cup sh_2)$ such that $R_i \cap V \neq \varnothing$ for each $i = 1, \ldots, n$, and $S = \bigcup_{i=1}^n R_i$. Suppose that, for $j = 1, 2$, we have $S_j = \bigcup\{\, R_i \in sh_j \mid 1 \leq i \leq n \,\}$. Thus $S_1 \in \mathrm{rel}(V, sh_1)^\star \cup \{\varnothing\}$, $S_2 \in \mathrm{rel}(V, sh_2)^\star \cup \{\varnothing\}$, and $S = S_1 \cup S_2$.

Suppose

$$\exists S_1 \in \mathrm{rel}(V, sh_1)^\star \cup \{\varnothing\} \mathrel{.} \exists S_2 \in \mathrm{rel}(V, sh_2)^\star \cup \{\varnothing\} \mathrel{.} S = S_1 \cup S_2,$$

where S_1 and S_2 are not both empty. Then, for some $m \geq 0$ and $n \geq 0$, there must exist $R_1, \ldots, R_m \in \mathrm{rel}(V, sh_1)$ and $T_1, \ldots, T_n \in \mathrm{rel}(V, sh_2)$ such that $S_1 = \bigcup_{i=1}^m R_i$ and $S_2 = \bigcup_{i=1}^n T_i$. Then $R_1, \ldots, R_m, T_1, \ldots, T_n \in \mathrm{rel}(V, sh_1 \cup sh_2)$ and

$$S = \Bigl(\bigcup_{i=1}^m R_i\Bigr) \cup \Bigl(\bigcup_{i=1}^n T_i\Bigr).$$

Thus $S \in \mathrm{rel}(V, sh_1 \cup sh_2)^\star$. $\square$

Lemma 3.60 *For each $V_1, V_2 \in \wp_\mathrm{f}(\mathit{Vars})$ and $sh \in SH$ we have*

$$\mathrm{rel}\bigl(V_1, \overline{\mathrm{rel}}(V_2, sh)\bigr) = \overline{\mathrm{rel}}\bigl(V_2, \mathrm{rel}(V_1, sh)\bigr).$$

Proof. Suppose $S \in SG$. Then $S \in \mathrm{rel}\bigl(V_1, \overline{\mathrm{rel}}(V_2, sh)\bigr)$ means $S \cap V_1 \neq \varnothing$ and $S \cap V_2 = \varnothing$. Similarly, $S \in \overline{\mathrm{rel}}\bigl(V_2, \mathrm{rel}(V_1, sh)\bigr)$ means that $S \cap V_2 = \varnothing$ and $S \cap V_1 \neq \varnothing$. $\square$

Proof of Theorem 3.46 on page 67. We let R, S, T, and U (possibly subscripted) denote elements of $sh^\star$. The subscripts reflect certain properties of the sets. In particular, subscripts x, r, xr, y, t, yt indicate sets of variables that definitely have a variable in common with the subscripted set. For example, R_x is a set in $sh^\star$ that has a common element with v_x and T_{xt} is a set in $sh^\star$ that has common elements with v_x and v_t. In contrast, the subscript '$-$' indicates that the subscripted set does not share with one of the sets v_{xr} or v_{yt}. Of course, in the proof, each set is formally defined as needed.

We will prove the implication

$$S \in \mathrm{amgu}\bigl(\mathrm{amgu}(sh, x \mapsto r), y \mapsto t\bigr) \implies S \in \mathrm{amgu}\bigl(\mathrm{amgu}(sh, y \mapsto t), x \mapsto r\bigr).$$

The converse will then holds by simply exchanging x and y, and r and t.

There are two cases due to the two components of the definition of amgu.

Case 1. Assume $S \in \overline{\mathrm{rel}}\bigl(v_{yt}, \mathrm{amgu}(sh, x \mapsto r)\bigr)$.
Then $S \in \mathrm{amgu}(sh, x \mapsto r)$ and $S \cap v_{yt} = \varnothing$. Again there are two possibilities.

Subcase 1a. Suppose first that $S \in \overline{\mathrm{rel}}(v_{xr}, sh)$. Thus $S \in sh$, and, since in this case we have $S \cap v_{yt} = \varnothing$, $S \in \overline{\mathrm{rel}}(v_{yt}, sh)$. Definition 3.43 implies $\overline{\mathrm{rel}}(v_{yt}, sh) \subseteq \mathrm{amgu}(sh, y \mapsto t)$ and thus we have also $S \in \mathrm{amgu}(sh, y \mapsto t)$. Now, since the hypothesis of this subcase implies $S \cap v_{xr} = \varnothing$, we obtain $S \in \overline{\mathrm{rel}}\bigl(v_{xr}, \mathrm{amgu}(sh, y \mapsto t)\bigr)$. Hence, again by Definition 3.43, we can conclude that $S \in \mathrm{amgu}\bigl(\mathrm{amgu}(sh, y \mapsto t), x \mapsto r\bigr)$.

Subcase 1b. Suppose now that $S \in \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh)^\star, \mathrm{rel}(v_r, sh)^\star\bigr)$.

Then, there exist $S_x, S_r \in SG$ such that $S = S_x \cup S_r$, where

$$S_x \in \mathrm{rel}(v_x, sh)^\star, \qquad S_r \in \mathrm{rel}(v_r, sh)^\star.$$

By the hypothesis for this case we have $S \cap v_{yt} = \varnothing$ and thus $S_x \cap v_{yt} = \varnothing$ and $S_r \cap v_{yt} = \varnothing$. This allows us to state that

$$S_x \in \overline{\mathrm{rel}}\bigl(v_{yt}, \mathrm{rel}(v_x, sh)^\star\bigr), \qquad S_r \in \overline{\mathrm{rel}}\bigl(v_{yt}, \mathrm{rel}(v_r, sh)^\star\bigr),$$

and hence, by Lemma 3.57,

$$S_x \in \overline{\mathrm{rel}}\bigl(v_{yt}, \mathrm{rel}(v_x, sh)\bigr)^\star, \qquad S_r \in \overline{\mathrm{rel}}\bigl(v_{yt}, \mathrm{rel}(v_r, sh)\bigr)^\star,$$

Thus, by Lemma 3.60,

$$S_x \in \mathrm{rel}\bigl(v_x, \overline{\mathrm{rel}}(v_{yt}, sh)\bigr)^\star, \qquad S_r \in \mathrm{rel}\bigl(v_r, \overline{\mathrm{rel}}(v_{yt}, sh)\bigr)^\star,$$

so that, by (3.16), (3.17), and Definition 3.43,

$$S_x \in \mathrm{rel}\bigl(v_x, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star, \qquad S_r \in \mathrm{rel}\bigl(v_r, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star.$$

Therefore, $S_x \cup S_r \in \mathrm{bin}\Bigl(\mathrm{rel}\bigl(v_x, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star, \mathrm{rel}\bigl(v_r, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star\Bigr)$, so that, as $S_x \cup S_r = S$, it follows from Definition 3.43 that

$$S \in \mathrm{amgu}\bigl(\mathrm{amgu}(sh, y \mapsto t), x \mapsto r\bigr).$$

Case 2. Assume $S \in \mathrm{bin}\Bigl(\mathrm{rel}\bigl(v_y, \mathrm{amgu}(sh, x \mapsto r)\bigr)^\star, \mathrm{rel}\bigl(v_t, \mathrm{amgu}(sh, x \mapsto r)\bigr)^\star\Bigr)$.

Then there exist $S_y, S_t \in SG$ such that

$$\begin{aligned} S &= S_y \cup S_t, \\ S_y &\in \operatorname{rel}\bigl(v_y, \operatorname{amgu}(sh, x \mapsto r)\bigr)^\star, \\ S_t &\in \operatorname{rel}\bigl(v_t, \operatorname{amgu}(sh, x \mapsto r)\bigr)^\star. \end{aligned} \tag{3.52}$$

Then, by Lemma 3.58,

$$S_y \cap v_y \neq \varnothing, \qquad S_t \cap v_t \neq \varnothing. \tag{3.53}$$

By Definition 3.43 and Lemma 3.59, there exist R_-, R_{xr}, T_-, and T_{xr} such that

$$S_y = R_- \cup R_{xr}, \qquad S_t = T_- \cup T_{xr} \tag{3.54}$$

where

$$\begin{aligned} R_- &\in \operatorname{rel}\bigl(v_y, \overline{\operatorname{rel}}(v_{xr}, sh)\bigr)^\star \cup \{\varnothing\}, \\ R_{xr} &\in \operatorname{rel}\Bigl(v_y, \operatorname{bin}\bigl(\operatorname{rel}(v_x, sh)^\star, \operatorname{rel}(v_r, sh)^\star\bigr)\Bigr)^\star \cup \{\varnothing\}, \\ T_- &\in \operatorname{rel}\bigl(v_t, \overline{\operatorname{rel}}(v_{xr}, sh)\bigr)^\star \cup \{\varnothing\}, \\ T_{xr} &\in \operatorname{rel}\Bigl(v_t, \operatorname{bin}\bigl(\operatorname{rel}(v_x, sh)^\star, \operatorname{rel}(v_r, sh)^\star\bigr)\Bigr)^\star \cup \{\varnothing\}. \end{aligned} \tag{3.55}$$

Then, by Lemmas 3.60 and 3.57,

$$\begin{aligned} R_- &\in \overline{\operatorname{rel}}\bigl(v_{xr}, \operatorname{rel}(v_y, sh)^\star\bigr) \cup \{\varnothing\}, \\ T_- &\in \overline{\operatorname{rel}}\bigl(v_{xr}, \operatorname{rel}(v_t, sh)^\star\bigr) \cup \{\varnothing\}. \end{aligned} \tag{3.56}$$

Also, using Lemmas 3.58, 3.56, and then the idempotence of $(\cdot)^\star$,

$$\begin{aligned} R_{xr} &\in \operatorname{rel}\Bigl(v_y, \operatorname{bin}\bigl(\operatorname{rel}(v_x, sh)^\star, \operatorname{rel}(v_r, sh)^\star\bigr)\Bigr) \cup \{\varnothing\}, \\ T_{xr} &\in \operatorname{rel}\Bigl(v_t, \operatorname{bin}\bigl(\operatorname{rel}(v_x, sh)^\star, \operatorname{rel}(v_r, sh)^\star\bigr)\Bigr) \cup \{\varnothing\}. \end{aligned} \tag{3.57}$$

Subcase 2a. Suppose $R_{xr} = T_{xr} = \varnothing$. Then, by (3.54),

$$S_y = R_-, \quad S_t = T_-. \tag{3.58}$$

By (3.53), $R_-, T_- \neq \varnothing$ and hence, using (3.56), $R_- \cup T_- \in \operatorname{bin}\bigl(\operatorname{rel}(v_y, sh)^\star, \operatorname{rel}(v_t, sh)^\star\bigr)$, so that, by Definition 3.43, $R_- \cup T_- \in \operatorname{amgu}(sh, y \mapsto t)$. Also, it follows from (3.56) that $R_- \cap v_{xr} = \varnothing$ and $T_- \cap v_{xr} = \varnothing$, so that $R_- \cup T_- \in \overline{\operatorname{rel}}\bigl(v_{xr}, \operatorname{amgu}(sh, y \mapsto t)\bigr)$. However, by (3.52) and (3.58), $S = R_- \cup T_-$ so that, by Definition 3.43,

$$S \in \operatorname{amgu}\bigl(\operatorname{amgu}(sh, y \mapsto t), x \mapsto r\bigr).$$

Subcase 2b. Suppose $R_{xr} \cup T_{xr} \neq \varnothing$. Then, by (3.57),

$$(R_{xr} \cup T_{xr}) \cap v_{yt} \neq \varnothing. \tag{3.59}$$

The proof of this subcase is in two parts. In the first part we divide R_{xr} and T_{xr} into a number of subsets. In the second part, these subsets will be reassembled so as to prove the required result.

First, by (3.57), there exist $R_x, R_r, T_x, T_r \in \wp_{\mathrm{f}}(\mathit{Vars})$ such that

$$R_{xr} = R_x \cup R_r, \qquad T_{xr} = T_x \cup T_r, \tag{3.60}$$

where either $R_x = R_r = \varnothing$ or

$$R_x \in \mathrm{rel}(v_x, sh)^\star, \qquad R_r \in \mathrm{rel}(v_r, sh)^\star,$$

and either $T_x = T_r = \varnothing$ or

$$T_x \in \mathrm{rel}(v_x, sh)^\star, \qquad T_r \in \mathrm{rel}(v_r, sh)^\star.$$

Thus, if either $R_x \cup T_x = \varnothing$ or $R_r \cup T_r = \varnothing$, it follows that

$$R_{xr} \cup T_{xr} = (R_x \cup R_r) \cup (T_x \cup T_r) = \varnothing.$$

However, by (3.59), $R_{xr} \cup T_{xr} \neq \varnothing$, so that we have

$$R_x \cup T_x \neq \varnothing, \qquad R_r \cup T_r \neq \varnothing. \tag{3.61}$$

We now subdivide the sets R_x, T_x, R_r, and T_r further. First note that

$$\begin{aligned}
sh &= \overline{\mathrm{rel}}(v_{yt}, sh) \cup \mathrm{rel}(v_y, sh) \cup \overline{\mathrm{rel}}\bigl(v_y, \mathrm{rel}(v_t, sh)\bigr), \\
sh &= \overline{\mathrm{rel}}(v_{yt}, sh) \cup \overline{\mathrm{rel}}\bigl(v_t, \mathrm{rel}(v_y, sh)\bigr) \cup \mathrm{rel}(v_t, sh).
\end{aligned}$$

Hence, by Lemma 3.59, sets R_{x-}, R_{xy}, R_{xt}, R_{r-}, R_{ry}, R_{rt}, T_{x-}, T_{xy}, T_{xt}, T_{r-}, T_{ry}, $T_{rt} \in \wp_{\mathrm{f}}(\mathit{Vars})$ exist such that

$$\begin{aligned}
R_x &= R_{x-} \cup R_{xy} \cup R_{xt}, \qquad & T_x &= T_{x-} \cup T_{xy} \cup T_{xt}, \\
R_r &= R_{r-} \cup R_{ry} \cup R_{rt}, \qquad & T_r &= T_{r-} \cup T_{ry} \cup T_{rt},
\end{aligned} \tag{3.62}$$

where

$$\begin{aligned}
R_{x-}, T_{x-} &\in \mathrm{rel}\bigl(v_x, \overline{\mathrm{rel}}(v_{yt}, sh)\bigr)^\star \cup \{\varnothing\}, \\
R_{r-}, T_{r-} &\in \mathrm{rel}\bigl(v_r, \overline{\mathrm{rel}}(v_{yt}, sh)\bigr)^\star \cup \{\varnothing\},
\end{aligned} \tag{3.63}$$

and

$$
\begin{aligned}
R_{xy}, T_{xy} &\in \operatorname{rel}\bigl(v_x, \operatorname{rel}(v_y, sh)\bigr)^\star \cup \{\varnothing\},\\
R_{ry}, T_{ry} &\in \operatorname{rel}\bigl(v_r, \operatorname{rel}(v_y, sh)\bigr)^\star \cup \{\varnothing\},\\
R_{xt}, T_{xt} &\in \operatorname{rel}\bigl(v_x, \operatorname{rel}(v_t, sh)\bigr)^\star \cup \{\varnothing\},\\
R_{rt}, T_{rt} &\in \operatorname{rel}\bigl(v_r, \operatorname{rel}(v_t, sh)\bigr)^\star \cup \{\varnothing\},
\end{aligned}
\tag{3.64}
$$

and also

$$
\begin{aligned}
(R_x \setminus R_{xy}) \cap v_y &= \varnothing, \qquad & (T_x \setminus T_{xt}) \cap v_t &= \varnothing,\\
(R_r \setminus R_{ry}) \cap v_y &= \varnothing, \qquad & (T_r \setminus T_{rt}) \cap v_t &= \varnothing.
\end{aligned}
\tag{3.65}
$$

We note a few simple but useful consequences of these definitions. First, it follows from (3.63) using (3.16), (3.17), and Definition 3.43, that

$$
\begin{aligned}
R_{x-}, T_{x-} &\in \operatorname{rel}\bigl(v_x, \operatorname{amgu}(sh, y \mapsto t)\bigr)^\star \cup \{\varnothing\},\\
R_{r-}, T_{r-} &\in \operatorname{rel}\bigl(v_r, \operatorname{amgu}(sh, y \mapsto t)\bigr)^\star \cup \{\varnothing\}.
\end{aligned}
\tag{3.66}
$$

Secondly, using (3.63) with Lemma 3.58, we have

$$
R_{x-}, T_{x-}, R_{r-}, T_{r-} \in \overline{\operatorname{rel}}(v_{yt}, sh)^\star \cup \{\varnothing\},
\tag{3.67}
$$

and then, using this with (3.59), (3.60), and (3.62), it follows that

$$
R_{xy} \cup T_{xy} \cup R_{ry} \cup T_{ry} \cup R_{xt} \cup T_{xt} \cup R_{rt} \cup T_{rt} \neq \varnothing.
\tag{3.68}
$$

In the second part of the proof for this subcase, the component subsets of S are reassembled in an order that proves the required result. First, let

$$
\begin{aligned}
U_y &\stackrel{\text{def}}{=} R_- \cup R_{xy} \cup R_{ry} \cup T_{xy} \cup T_{ry},\\
U_t &\stackrel{\text{def}}{=} T_- \cup R_{xt} \cup R_{rt} \cup T_{xt} \cup T_{rt},
\end{aligned}
\tag{3.69}
$$

$$
U \stackrel{\text{def}}{=} U_y \cup U_t.
\tag{3.70}
$$

By relations (3.55) and (3.64) (with Lemma 3.58), each component set in the definition of U_y is in $\operatorname{rel}(v_y, sh)^\star \cup \{\varnothing\}$; similarly, each component set in the definition of U_t is in $\operatorname{rel}(v_t, sh)^\star \cup \{\varnothing\}$. Thus, by the definition of $(\cdot)^\star$,

$$
\begin{aligned}
U_y &\in \operatorname{rel}(v_y, sh)^\star \cup \{\varnothing\},\\
U_t &\in \operatorname{rel}(v_t, sh)^\star \cup \{\varnothing\}.
\end{aligned}
$$

By (3.60) and (3.65) we have $\bigl(R_{xr} \setminus (R_{xy} \cup R_{ry})\bigr) \cap v_y = \varnothing$ and hence, by (3.54), we have also that $\bigl(S_y \setminus (R_{xy} \cup R_{ry} \cup R_-)\bigr) \cap v_y = \varnothing$. By (3.53), $S_y \cap v_y \neq \varnothing$. Thus, $R_{xy} \cup R_{ry} \cup R_- \neq \varnothing$

and, as a consequence of (3.69), $U_y \neq \varnothing$. For similar reasons, $U_t \neq \varnothing$. Hence, by (3.70),

$$U \in \mathrm{bin}\bigl(\mathrm{rel}(v_y, sh)^\star, \mathrm{rel}(v_t, sh)^\star\bigr),$$

and therefore, using Definition 3.43, it follows that

$$U \in \mathrm{amgu}(sh, y \mapsto t). \tag{3.71}$$

Now, by (3.68), at least one of the following two inequalities holds:

$$\begin{aligned} R_{xy} \cup T_{xy} \cup R_{xt} \cup T_{xt} &\neq \varnothing, \\ R_{ry} \cup T_{ry} \cup R_{rt} \cup T_{rt} &\neq \varnothing. \end{aligned} \tag{3.72}$$

Assume first that $R_{xy} \cup T_{xy} \cup R_{xt} \cup T_{xt} = \varnothing$ and $R_{ry} \cup T_{ry} \cup R_{rt} \cup T_{rt} \neq \varnothing$. Then, using (3.61) and (3.62) with the first of these, $R_{x-} \cup T_{x-} \neq \varnothing$. Also, using (3.64) with the second, we have $(R_{ry} \cup R_{rt} \cup T_{ry} \cup T_{rt}) \cap v_r \neq \varnothing$ and therefore it follows from (3.69) and (3.70), that $U \cap v_r \neq \varnothing$. Hence, by (3.66) and (3.71),

$$\begin{aligned} R_{x-} \cup T_{x-} &\in \mathrm{rel}\bigl(v_x, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star, \\ U \cup R_{r-} \cup T_{r-} &\in \mathrm{rel}\bigl(v_r, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star. \end{aligned} \tag{3.73}$$

Similarly, assuming $R_{xy} \cup T_{xy} \cup R_{xt} \cup T_{xt} \neq \varnothing$ and $R_{ry} \cup T_{ry} \cup R_{rt} \cup T_{rt} = \varnothing$ it follows that

$$\begin{aligned} R_{r-} \cup T_{r-} &\in \mathrm{rel}\bigl(v_r, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star, \\ R_{x-} \cup T_{x-} \cup U &\in \mathrm{rel}\bigl(v_x, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star. \end{aligned} \tag{3.74}$$

Finally, assuming $R_{xy} \cup T_{xy} \cup R_{xt} \cup T_{xt} \neq \varnothing$ and $R_{ry} \cup T_{ry} \cup R_{rt} \cup T_{rt} \neq \varnothing$ it follows from (3.64) that $U \cap v_x \neq \varnothing$ and $U \cap v_r \neq \varnothing$, and hence

$$\begin{aligned} R_{x-} \cup T_{x-} \cup U &\in \mathrm{rel}\bigl(v_x, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star, \\ U \cup R_{r-} \cup T_{r-} &\in \mathrm{rel}\bigl(v_r, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star. \end{aligned} \tag{3.75}$$

Thus, as one of the inequalities in (3.72) holds, one of (3.73), (3.74) or (3.75) holds so that

$$R_{x-} \cup T_{x-} \cup U \cup R_{r-} \cup T_{r-} \in \mathrm{bin}\Bigl(\mathrm{rel}\bigl(v_x, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star, \mathrm{rel}\bigl(v_r, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star\Bigr).$$

However, since $S = R_{x-} \cup T_{x-} \cup U \cup R_{r-} \cup T_{r-}$, we have

$$S \in \mathrm{bin}\Bigl(\mathrm{rel}\bigl(v_x, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star, \mathrm{rel}\bigl(v_r, \mathrm{amgu}(sh, y \mapsto t)\bigr)^\star\Bigr).$$

Hence, by Definition 3.43, $S \in \mathrm{amgu}\bigl(\mathrm{amgu}(sh, y \mapsto t), x \mapsto r\bigr)$. $\square$

3.4.5 Proofs of the Results of Section 3.4.3

We prove all the results in this section by induction on the cardinality of a substitution ν. All proofs are obvious if ν is empty or does not unify. Thus, in the following proofs we assume that ν unifies, is not empty and $(x \mapsto r) \in \nu$, with $\nu' \stackrel{\text{def}}{=} \nu \setminus \{x \mapsto r\}$.

Proof of Lemma 3.52 on page 68. We have

$$
\begin{aligned}
&\mathrm{Aunify}\Bigl(\mathrm{Amgu}\bigl((sh, \mathit{VI}), y \mapsto t\bigr), \nu\Bigr)\\
&\qquad [\text{by Definition 3.51}]\\
&\qquad = \mathrm{Aunify}\biggl(\mathrm{Amgu}\Bigl(\mathrm{Amgu}\bigl((sh, \mathit{VI}), y \mapsto t\bigr), x \mapsto r\Bigr), \nu'\biggr)\\
&\qquad [\text{by Corollary 3.50}]\\
&\qquad = \mathrm{Aunify}\biggl(\mathrm{Amgu}\Bigl(\mathrm{Amgu}\bigl((sh, \mathit{VI}), x \mapsto r\bigr), y \mapsto t\Bigr), \nu'\biggr)\\
&\qquad [\text{by the inductive hypothesis}]\\
&\qquad = \mathrm{Amgu}\biggl(\mathrm{Aunify}\Bigl(\mathrm{Amgu}\bigl((sh, \mathit{VI}), x \mapsto r\bigr), \nu'\Bigr), y \mapsto t\biggr)\\
&\qquad [\text{by Definition 3.51}]\\
&\qquad = \mathrm{Amgu}\Bigl(\mathrm{Aunify}\bigl((sh, \mathit{VI}), \nu\bigr), y \mapsto t\Bigr).
\end{aligned}
$$

□

Proof of Theorem 3.53 on page 68. Let $\mu' \in \mathrm{mgs}(\nu' \cup \sigma)$. Then

$$
\begin{aligned}
&\alpha(\sigma, \mathit{VI}) \preceq_{SS} (sh, \mathit{VI})\\
&\quad [\text{by the induction hypothesis}]\\
&\quad \implies \alpha\bigl(\mu', \mathit{VI} \cup \mathrm{vars}(\nu')\bigr) \preceq_{SS} \mathrm{Aunify}\bigl((sh, \mathit{VI}), \nu'\bigr)\\
&\quad [\text{by Corollary 3.48}]\\
&\quad \implies \alpha\bigl(\mu, \mathit{VI} \cup \mathrm{vars}(\nu)\bigr) \preceq_{SS} \mathrm{Amgu}\Bigl(\mathrm{Aunify}\bigl((sh, \mathit{VI}), \nu'\bigr), x \mapsto r\Bigr)\\
&\quad [\text{by Lemma 3.52}]\\
&\quad \implies \alpha\bigl(\mu, \mathit{VI} \cup \mathrm{vars}(\nu)\bigr) \preceq_{SS} \mathrm{Aunify}\Bigl(\mathrm{Amgu}\bigl((sh, \mathit{VI}), x \mapsto r\bigr), \nu'\Bigr)\\
&\quad [\text{by Definition 3.51}]\\
&\quad \implies \alpha\bigl(\mu, \mathit{VI} \cup \mathrm{vars}(\nu)\bigr) \preceq_{SS} \mathrm{Aunify}\bigl((sh, \mathit{VI}), \nu\bigr).
\end{aligned}
$$

□

Proof of Theorem 3.54 on page 69. We have

$$\mathrm{Aunify}\Bigl(\mathrm{Aunify}\bigl((sh, \mathit{VI}), \nu\bigr), \nu\Bigr)$$

[by Definition 3.51]

$$= \mathrm{Aunify}\Bigl(\mathrm{Amgu}\Bigl(\mathrm{Aunify}\bigl(\mathrm{Amgu}((sh,\, VI), x \mapsto r), \nu'\bigr), x \mapsto r\Bigr), \nu'\Bigr)$$

[by Lemma 3.52]

$$= \mathrm{Aunify}\Bigl(\mathrm{Aunify}\Bigl(\mathrm{Amgu}\bigl(\mathrm{Amgu}((sh,\, VI), x \mapsto r), x \mapsto r\bigr), \nu'\Bigr), \nu'\Bigr)$$

[by the inductive hypothesis]

$$= \mathrm{Aunify}\Bigl(\mathrm{Amgu}\bigl(\mathrm{Amgu}((sh,\, VI), x \mapsto r), x \mapsto r\bigr), \nu'\Bigr)$$

[by Corollary 3.49]

$$= \mathrm{Aunify}\Bigl(\mathrm{Amgu}\bigl((sh,\, VI), x \mapsto r\bigr), \nu'\Bigr)$$

[by Definition 3.51]

$$= \mathrm{Aunify}\bigl((sh,\, VI), \nu\bigr).$$

□

Proof of Theorem 3.55 on page 69. The induction is on the set of equations ν_1. The comments at the start of this section apply therefore to ν_1 instead of ν and thus we let $\nu_1' \stackrel{\mathrm{def}}{=} \nu_1 \setminus \{x \mapsto r\}$ so that we have

$$\mathrm{Aunify}\Bigl(\mathrm{Aunify}\bigl((sh,\, VI), \nu_1\bigr), \nu_2\Bigr)$$

[by Definition 3.51]

$$= \mathrm{Aunify}\Bigl(\mathrm{Aunify}\Bigl(\mathrm{Amgu}\bigl((sh,\, VI), x \mapsto r\bigr), \nu_1'\Bigr), \nu_2\Bigr)$$

[by the inductive hypothesis]

$$= \mathrm{Aunify}\Bigl(\mathrm{Aunify}\Bigl(\mathrm{Amgu}\bigl((sh,\, VI), x \mapsto r\bigr), \nu_2\Bigr), \nu_1'\Bigr)$$

[by Lemma 3.52]

$$= \mathrm{Aunify}\Bigl(\mathrm{Amgu}\Bigl(\mathrm{Aunify}\bigl((sh,\, VI), \nu_2\bigr), x \mapsto r\Bigr), \nu_1'\Bigr)$$

[by Definition 3.51]

$$= \mathrm{Aunify}\Bigl(\mathrm{Aunify}\bigl((sh,\, VI), \nu_2\bigr), \nu_1\Bigr).$$

□

3.5 Abstract Projection

The abstract unification function Aunify is the key operation that makes the abstract domain *SS* suitable for computing static approximations of the substitutions generated by the execution of logic programs. This operator is combined with simpler ones so as to provide a complete definition of the abstract semantics. As far as the correctness of the analysis is concerned, these auxiliary operators do not pose any serious problem.

For instance, the 'merge-over-all-paths' operator is implemented as the lub (Alub) of the domain SS: in this case, correctness follows from a standard result of abstract interpretation theory [CC77a]. Other operators, such as the consistent renaming of variables, are almost trivial.

We now define the abstract version of the concrete projection operator 'proj'. This operator is interesting in that it modifies the set of variables of interest.

Definition 3.61 (Abstract projection.) *For each $V \in \wp_{\mathrm{f}}(\mathit{Vars})$, the abstract function* aexists: $SH \times \wp_{\mathrm{f}}(\mathit{Vars}) \to SH$ *provides the* existential quantification *of an element of SH: for each $sh \in SH$,*

$$\mathrm{aexists}(sh, V) \stackrel{\mathrm{def}}{=} \bigl\{\, S \setminus V \bigm| S \in sh, S \setminus V \neq \varnothing \,\bigr\} \cup \bigl\{\, \{x\} \bigm| x \in V \,\bigr\}.$$

The function Aproj: $SS \times \wp_{\mathrm{f}}(\mathit{Vars}) \to SS$ projects *an element of SS onto a new set of variables of interest. For each $d \in SS$ and $W \in \wp_{\mathrm{f}}(\mathit{Vars})$,*

$$\mathrm{Aproj}(d, W) \stackrel{\mathrm{def}}{=} \begin{cases} \top, & \text{if } d = \top; \\ (sh', W) & \text{if } d = (sh, \mathit{VI}); \\ \bot, & \text{if } d = \bot; \end{cases}$$

where $sh' = \Bigl(\mathrm{aexists}(sh, \mathit{VI} \setminus W) \setminus \bigl\{\, \{x\} \bigm| x \in \mathit{VI} \setminus W \,\bigr\}\Bigr) \cup \bigl\{\, \{x\} \bigm| x \in W \setminus \mathit{VI} \,\bigr\}.$

The statement and proof of correctness for this abstract projection operator are omitted for space reasons.

3.6 Summary

The set-sharing domain SS, which was defined in [JL89, Lan90], is considered to be the principal abstract domain for sharing analysis of logic programs in both practical work and theoretical study. In this chapter we have defined a new abstraction function mapping a set of substitutions in rational solved form into their corresponding sharing abstraction. The new function is a generalization of the classical abstraction function of [JL89], which was defined for idempotent substitutions only. Using our new abstraction function, we have proved the correctness of the classical abstract unification operator Aunify. Other contributions of our work are the formal proofs of the commutativity and idempotence of the Aunify operator on the set-sharing domain. Even if commutativity was a known property, the corresponding proof in [Lan90] was not satisfactory. As far as idempotence is concerned, our result differs from that given in [Lan90], which was based on a composite abstract unification operator performing also the renaming of variables. It is our opinion that our main result, the correctness of the Aunify operator, is really valuable as it allows for the safe application of sharing analysis based on SS to any constraint logic language supporting syntactic term structures, based on either finite trees or rational trees. This happens because our result does not rely on the presence (or even the absence) of the occurs-check in the concrete unification procedure implemented by the analysed language. Furthermore, as the groundness domain Def is included in SS [CFW94, CFW98], our main correctness result also shows that Def is sound for non-idempotent substitutions.

From a technical point of view, we have introduced a new class of concrete substitutions based on the notion of *variable-idempotence*, generalizing the classical concept of idempotence. We have shown that any substitution is equivalent to a variable-idempotent one, providing a finite sequence of transformations for its construction. This result assumes an arbitrary equality theory and is therefore applicable to the study of any abstract property which is preserved by logical equivalence. Our application of this idea to the study of the correctness of abstract unification for SS has shown that it is particularly suitable for data-flow analyzers where the corresponding abstraction function only depends on the set of variables occurring in a term. However, as we will show in Chapter 6 when we combine the set-sharing domain with freeness and linearity, this concept can be usefully exploited in more general contexts.

Chapter 4

Set-Sharing is Redundant for Pair-Sharing

Although the usual goal of sharing analysis is to detect which pairs of variables may share, the standard choice for sharing analysis is a domain that characterizes set-sharing. In this chapter we show that the set-sharing domain is over-complex for pair-sharing analysis. By defining an equivalence relation over the domain SS we obtain a simpler domain which is as precise as SS as far as the computation of pair-sharing is concerned. We also prove that no further abstraction of this domain satisfies this property. A key feature of the simplified domain is that it allows a significant reduction in the theoretical complexity of the abstract unification procedure.

> **Note:** this chapter is mainly based on the results of [BHZ97]; an extended version will appear in [BHZ02].

4.1 Pair-Sharing or Set-Sharing?

Soon after the introduction of the set-sharing domain by Jacobs and Langen, a remarkable amount of research work has been dedicated to the comparison between this domain and its classic challenger, the pair-sharing domain ASub of Søndergaard.

From a practical point of view, the ASub domain looks more appealing, being characterized by a representation and abstract operators that only require polynomial space and time, respectively. In contrast, in the set-sharing domain, the representation and the abstract operators require an exponential amount of space and time.

In spite of this, today the set-sharing domain has become the preferred one in most theoretical work on the sharing analysis for logic languages. The adequacy of this domain is not normally questioned. Researchers appear to be more concerned as to which *add-ons* are best: linearity, freeness, depth-k abstract substitutions and so on [BC93, BCM94a, CDFB93, Kin94, KS94, MH91], rather than whether it is the optimal domain for the sharing information under investigation.

The reason for this standard choice lies in the accuracy of the domain: even if, from a formal point of view, neither of the two domains is uniformly more precise than the other, the dependencies encoded in the set-sharing domain make it more "interesting", in particular when integrating it with other kinds of information. Indeed, the SS domain is quite difficult to understand. Taking an abstract element and writing down its concretization (namely, the concrete substitutions that are approximated by it) is fairly simple, provided one strictly follows the well-established principles of abstract interpretation. However, it is quite common for researchers to get into trouble when trying to informally explain such a mathematical formula, in order to provide a reader-friendly interpretation.

So the question arises: is this complexity actually needed for an accurate sharing analysis? Before providing an answer, we must agree on what the purpose of sharing analysis is. The results in this chapter rely on the following

Assumption: *The goal of sharing analysis for logic programs is to detect which* pairs *of variables are definitely independent.*

We thus focus our attention on the pair-sharing property, investigating whether all of the information in the SS domain is really needed for detecting which pairs of variables can share. The answer turns out to be negative: there exists a domain that is simpler than SS and, at the same time, as precise as SS, as far as pair-sharing is concerned.

4.2 The Pair-Sharing Property

Let us define the pair-sharing property through a domain that captures it exactly. This domain is very similar to Søndergaard's ASub, which however also includes groundness and linearity information [Søn86].

Definition 4.1 (The *pair-sharing* domain.) *Let S be a set. Then*

$$\mathrm{pairs}(S) \stackrel{\mathrm{def}}{=} \bigl\{\, P \in \wp(S) \bigm| \# P = 2 \,\bigr\}.$$

The pair-sharing domain *is given by the complete lattice*

$$PS \stackrel{\mathrm{def}}{=} \left\{\, (ps, VI) \;\middle|\; VI \in \wp_{\mathrm{f}}(Vars), ps \in \wp\bigl(\mathrm{pairs}(VI)\bigr) \,\right\} \cup \{\bot_{PS}, \top_{PS}\}$$

ordered by $\preceq_{PS}$, which is defined, for each $d, (ps_1, VI_1), (ps_2, VI_2) \in PS$, by

$$\begin{aligned} \bot_{PS} &\preceq_{PS} d, \\ d &\preceq_{PS} \top_{PS}, \\ (ps_1, VI_1) \preceq_{PS} (ps_2, VI_2) &\quad\iff\quad (VI_1 = VI_2) \land (ps_1 \subseteq ps_2). \end{aligned}$$

An element of the pair-sharing domain is, roughly speaking, the "end-user image" of the result of the analysis. That is, the only interest of the end-user of our analysis (e.g., the optimizer module of the compiler) is knowing which *pairs* of variables possibly share. The PS domain will be used to measure the accuracy of the other domains in computing pair-sharing.

4.3 The Information Content of Sharing Sets

We now look at the information content of the elements of the SS domain. First consider the pair-sharing information.

Pair-sharing. Clearly, PS is a strict abstraction of SS through the abstraction function $\alpha_{PS} \colon SS \to PS$ given, for each $(sh, VI) \in SS$, by

$$\begin{aligned} \alpha_{PS}(\bot) &\stackrel{\text{def}}{=} \bot_{PS}, \\ \alpha_{PS}(\top) &\stackrel{\text{def}}{=} \top_{PS}, \\ \alpha_{PS}\bigl((sh, VI)\bigr) &\stackrel{\text{def}}{=} \bigl(\downarrow(sh) \cap \text{pairs}(VI), VI\bigr), \end{aligned}$$

where $\downarrow \colon SH \to SH$ is defined, for each $sh \in SH$, by

$$\downarrow(sh) \stackrel{\text{def}}{=} \bigl\{\, T \in SG \bigm| \exists S \in sh \mathrel{.} T \subseteq S \,\bigr\}.$$

As it has been observed by several authors, the SS lattice encodes several properties besides pair-sharing. We next give examples that show the relevance of these properties with respect to computing the pair-sharing information. In what follows, the set of variables of interest is fixed as $VI \stackrel{\text{def}}{=} \{x, y, z\}$ and will be omitted from elements of SS. Moreover, the elements of SH will be written in a simplified notation, omitting the inner braces. For example, the element

$$\bigl(\bigl\{\{x\}, \{x, y\}, \{x, z\}\bigr\}, \{x, y, z\}\bigr)$$

will be written simply as $\{x, xy, xz\}$. This notation will be adopted in all the examples in the thesis, provided no confusion can arise.

Groundness. Consider $sh_1 \stackrel{\text{def}}{=} \{xy\}$ and $sh_2 \stackrel{\text{def}}{=} \{xy, z\}$. They encode the same pair-sharing information, namely $\alpha_{PS}(sh_1) = \alpha_{PS}(sh_2) = \{xy\}$. Since z does not occur in any sharing group of sh_1, we know that the variable z is ground. In contrast, in concrete substitutions abstracted by sh_2, z is not necessarily ground. This knowledge is useful for pair-sharing detection:

$$\begin{aligned} \alpha_{PS}\bigl(\text{amgu}(sh_1, x \mapsto z)\bigr) &= \alpha_{PS}(\varnothing) = \varnothing, \\ \alpha_{PS}\bigl(\text{amgu}(sh_2, x \mapsto z)\bigr) &= \alpha_{PS}\bigl(\{xyz\}\bigr) = \{xy, xz, yz\}. \end{aligned}$$

Incidentally, this example constitutes a proof of the fact that any abstraction of SS which is as precise as SS on pair-sharing is also as precise as SS on groundness. The proof is by contraposition: lose only one ground variable and precision on pair-sharing is compromised.

Ground dependencies. Let $sh_1 \stackrel{\mathrm{def}}{=} \{xy, xyz\}$ and $sh_2 \stackrel{\mathrm{def}}{=} \{xy, xz, yz\}$. They still encode the same pair-sharing information. They also encode the same groundness information (no variable is ground). However, in contrast to sh_2, x occurs in all sharing groups in sh_1 that contain y. Thus, for sh_1, the groundness of y depends solely on the groundness of x. Let us bind variable x to the term $a \in \mathit{GTerms} \cap \mathit{HTerms}$ and see what happens:

$$\begin{aligned} \alpha_{PS}\bigl(\mathrm{amgu}(sh_1, x \mapsto a)\bigr) &= \alpha_{PS}(\varnothing) = \varnothing, \\ \alpha_{PS}\bigl(\mathrm{amgu}(sh_2, x \mapsto a)\bigr) &= \alpha_{PS}\bigl(\{yz\}\bigr) = \{yz\}. \end{aligned}$$

Therefore, a knowledge of ground dependencies is important for pair-sharing detection.

Pair-sharing dependencies. The following has been obtain by adapting a similar example in [CF93]. Let

$$\begin{aligned} sh_1 &\stackrel{\mathrm{def}}{=} \{x, y, z, xyz\}, \\ sh_2 &\stackrel{\mathrm{def}}{=} \{x, y, z, xy, xz, yz\}. \end{aligned}$$

They encode the same pair-sharing, groundness, and ground dependency information. Again, let us ground x and look at the results:

$$\begin{aligned} \alpha_{PS}\bigl(\mathrm{amgu}(sh_1, x \mapsto a)\bigr) &= \alpha_{PS}\bigl(\{y, z\}\bigr) = \varnothing, \\ \alpha_{PS}\bigl(\mathrm{amgu}(sh_2, x \mapsto a)\bigr) &= \alpha_{PS}\bigl(\{y, z, yz\}\bigr) = \{yz\}. \end{aligned}$$

In sh_1, x occurs in all the sharing groups that contain the pair yz. Thus in sh_1 the sharing between y and z depends on the (non-) groundness of x, while in sh_2 this is not the case.

Redundant information? Given these three examples, one gets the impression that different elements in SH do encode different information with respect to the computation of the pair-sharing property. However, this is not always the case. Consider

$$\begin{aligned} sh_1 &\stackrel{\mathrm{def}}{=} \{x, y, z, xy, xz, yz\}, \\ sh_2 &\stackrel{\mathrm{def}}{=} \{x, y, z, xy, xz, yz, xyz\}. \end{aligned}$$

These two different elements do encode the same pair-sharing, groundness, ground dependency, and sharing dependency information. Since the set of variables of interest is $\mathit{VI} = \{x, y, z\}$, we have $sh_2 = \wp(\mathit{VI}) \setminus \{\varnothing\}$. This means that any sharing is possible, and thus that sh_2 describes all the substitutions in rational solved form having VI as the set of variables of interest. In contrast, the only relevant information in sh_1 is that the sharing group xyz is not allowed: sh_1 represents all the substitutions $\sigma \in \mathit{RSubst}$ such that

$$\mathrm{vars}\bigl(\mathrm{rt}(x, \sigma)\bigr) \cap \mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr) \cap \mathrm{vars}\bigl(\mathrm{rt}(z, \sigma)\bigr) = \varnothing.$$

That is, the variables x, y, and z cannot share the *same* variable (but they still can share pairwise). As observed before, this difference is irrelevant from the end-user point of view. We will show that sh_1 and sh_2 are completely equivalent with respect to the pair-sharing property and that the sharing group xyz in sh_2 is redundant for pair-sharing.

4.4 Set-Sharing is Redundant for Pair-Sharing

In the previous example, we noted that the sharing group xyz was redundant for sh_2. We now formalize this notion of redundancy.

Definition 4.2 (Redundancy.) *Let $sh \in SH$ and $S \in SG$. S is* redundant for sh *if and only if $\# S > 2$ and*

$$\mathrm{pairs}(S) = \bigcup \bigl\{ \mathrm{pairs}(T) \bigm| T \in sh, T \subset S \bigr\}.$$

Read it this way: S is redundant for sh if and only if all its sharing pairs can be extracted from the elements of sh that are contained in S. As the name suggests, redundant sharing groups can be dropped. For the moment, as we are walking on theoretical ground, we *add* them so as to obtain a sort of *normal form*. A notable advantage is that we can still use subset inclusion for the ordering. We thus define an upper closure operator over SH that induces an equivalence relation over the elements of SH.

Definition 4.3 (A closure operator on SH.) *The function $\rho \colon SH \to SH$ is given, for each $sh \in SH$, by*

$$\rho(sh) \stackrel{\mathrm{def}}{=} sh \cup \{ S \in SG \mid S \textit{ is redundant for } sh \}.$$

Theorem 4.4 *The function $\rho \colon SH \to SH$ is an upper closure operator.*

In Definition 4.2, a sharing group S can be added to a sharing set sh without changing the pair-sharing information if and only if, *for each variable x in S*, every pair such as xy in S is in some sharing group in sh which is also a subset of S. This implies that, for each variable x in S, S must be the union of some of the sets in sh that contain x. This observation leads to the following alternative definition for ρ.

Theorem 4.5 *If $sh \in SH$ then*

$$\rho(sh) = \left\{ S \in SG \;\middle|\; \forall x \in S : S \in \mathrm{rel}\bigl(\{x\}, sh\bigr)^{\star} \right\}.$$

While the original definition refers directly to the pair-sharing concept, the alternative definition provided by Theorem 4.5 is very elegant and concise, and turns out to be useful for proving several results.

Abusing notation, we can easily define the overloading $\rho\colon SS \to SS$ such that

$$\begin{aligned}
\rho(\bot) &\stackrel{\text{def}}{=} \bot,\\
\rho(\top) &\stackrel{\text{def}}{=} \top,\\
\rho\bigl((sh, \mathit{VI})\bigr) &\stackrel{\text{def}}{=} \bigl(\rho(sh), \mathit{VI}\bigr).
\end{aligned}$$

We have thus implicitly defined a new domain that we will denote by *PSD* (the *pair-sharing dependency* domain). The domain *PSD* is the quotient of *SS* with respect to the equivalence relation induced by ρ: d_1 and d_2 are equivalent if and only if $\rho(d_1) = \rho(d_2)$. Clearly, *PSD* is a proper abstraction of *SS*.

It is straightforward to prove the following.

Theorem 4.6 *For each $d \in SS$ we have $\alpha_{PS}\bigl(\rho(d)\bigr) = \alpha_{PS}(d)$.*

Thus the addition of redundant sharing groups does not cause any precision loss, as far as pair-sharing is concerned. In other words, *PSD* is as good as *SS* for *representing* pair-sharing. We now show that ρ is a congruence with respect to the operations Aunify, Alub, and Aproj.

Theorem 4.7 *Let $d_1, d_2 \in SS$. If $\rho(d_1) = \rho(d_2)$ then, for each $\sigma \in \mathit{RSubst}$, each $d' \in SS$, and each $V \in \wp_{\mathrm{f}}(\mathit{Vars})$,*

1. $\rho\bigl(\mathrm{Aunify}(d_1, \sigma)\bigr) = \rho\bigl(\mathrm{Aunify}(d_2, \sigma)\bigr)$;
2. $\rho\bigl(\mathrm{Alub}(d', d_1)\bigr) = \rho\bigl(\mathrm{Alub}(d', d_2)\bigr)$; *and*
3. $\rho\bigl(\mathrm{Aproj}(d_1, V)\bigr) = \rho\bigl(\mathrm{Aproj}(d_2, V)\bigr)$.

As a corollary of the two results above we have that *PSD* is as good as *SS* for *propagating* pair-sharing through the analysis process. We also show that any proper abstraction of *PSD* is less precise than *PSD* on computing pair-sharing.

Theorem 4.8 *For each $d_1, d_2 \in SS$, $\rho(d_1) \neq \rho(d_2)$ implies*

$$\exists \sigma \in \mathit{RSubst} \;.\; \alpha_{PS}\bigl(\mathrm{Aunify}(d_1, \sigma)\bigr) \neq \alpha_{PS}\bigl(\mathrm{Aunify}(d_2, \sigma)\bigr).$$

To summarize, the equivalence relation induced by ρ identifies two elements if and only if their behavior in the analysis process is indistinguishable with respect to the pair-sharing property.

4.5 Star-Union is Not Needed

When moving from the theoretical to the practical ground, the first issue concerns the choice of an actual representation for the elements of *PSD*, which are the equivalence classes induced by ρ over *SS*. One possibility is to fix once and for all the representative of each equivalence class. In particular, this would allow for an implementation of the equivalence check as an identity check.

One obvious candidate representative for the class is the image under ρ of any element of the class. By a standard result of the theory of closure operators, this element is the *maximum* element in its class with respect to the lattice ordering. Of course, as far as efficiency is concerned, this would be a really unfortunate choice as, in general, $\# \rho(sh)$ is an exponential function of $\# sh$.

A much more interesting alternative is made possible by the following result, which shows that all the equivalence classes based on ρ are also endowed with a *minimum* element with respect to the lattice ordering.

Theorem 4.9 *For all $sh_1, sh_2 \in SH$,*

$$\rho(sh_1) = \rho(sh_2) \implies \rho(sh_1 \cap sh_2) = \rho(sh_2).$$

Not surprisingly, the minimum element is the only element of the equivalence class containing no redundant sharing groups.

Theorem 4.10 *For all $sh \in SH$,*

$$sh \setminus \{\, S \in SG \mid S \text{ is redundant for } sh \,\} = \bigcap \bigl\{\, sh' \in SH \bigm| \rho(sh') = \rho(sh) \,\bigr\}.$$

Using the minimum elements (from now on also called *reduced* elements) as representatives for the equivalence classes would seem the best thing to do: memory occupation and the computational cost would be kept at a minimum. However, it must not be forgotten that reducing a sharing set (i.e., removing all its redundant sharing groups) has a price. Moreover, the abstract operators on sharing sets may generate non-reduced elements even from reduced ones.

A different solution to the problem of deciding on the classes' representatives is to allow the implementation to select it dynamically. In this setting, the implementation is left free to choose *any* element of an equivalence class as the representative of the class. Moreover, the implementation can make different choices at different times during the analysis. These choices can be guided by several heuristics, with the objective of finding a good trade-off between the cost of reductions and the benefits of working with smaller, and possibly minimal, elements. One of the consequences of allowing this kind of freedom is that the equivalence check can no longer be implemented as an identity check. This does not constitute a serious drawback: as we will see the complexity of the equivalence check is bounded by the square of the number of sharing groups.

Since the computational complexity of all the abstract operators depends on the cardinality of the sharing sets involved, a general recipe for efficiency is avoiding, wherever possible, the generation of redundant sharing groups. For this purpose, another very interesting practical consequence of the theory developed in the previous section is that the star-union operator can be *safely* replaced by the binary-union operator.

Theorem 4.11 *For each $sh \in SH$ we have $sh^\star = \rho\bigl(\mathrm{bin}(sh, sh)\bigr)$.*

In words, $\mathrm{bin}(sh, sh)$ is granted to be in the same equivalence class of $sh^\star$ and it is quite likely to contain less redundant sharing groups. Moreover, in the worst-case, the complexity of the star-union operator is exponential in the number of sharing groups of the input, while for the binary-union operator the complexity is quadratic. For notational convenience, for each $sh \in SH$, we will write sh^2 to denote $\mathrm{bin}(sh, sh)$.

This method for computing (a representative of) the star-union can be safely applied in the computation for abstract unification. We prove here the result for the operator amgu. Since Aunify is defined in terms of amgu (via Amgu), the revised definition for amgu can be used in the computation of Aunify.

Theorem 4.12 *Let $sh \in SH$ and $(x \mapsto t) \in Bind$. Let also*

$$sh_- \stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh\bigr), \qquad sh_x \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\{x\}, sh\bigr), \qquad sh_t \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr).$$

Then

$$\rho\bigl(\mathrm{amgu}(sh, x \mapsto t)\bigr) = \rho\bigl(sh_- \cup \mathrm{bin}(sh_x^2, sh_t^2)\bigr).$$

4.6 The Quotient of Abstract Interpretations

All the results in this chapter have been obtained by following a precise methodology, that we now briefly review and reinterpret in the light of recent results in abstract interpretation theory. We would like to stress that all the conceptual devices we have resorted to are part of the classic inheritance of the semantics of programming languages: in this respect, we did not invent anything.

For the purpose of the present discussion, let us adopt the closure operator approach to abstract interpretation and let us call SS the "concrete domain" and PSD the "abstract domain". We have looked for an *upper closure operator* over SS that is

1. more concrete than the upper closure operator associated to PS,

2. such that the induced equivalence relation over SS is a congruence relation with respect to all the semantic operators of the domain.

These properties give rise to a *completeness* result of the abstract semantics (PSD) with respect to the concrete semantics (SS). In other words, every pair-sharing captured by SS is also captured by PSD, with their respective operations. Moreover, we established that ρ is the weakest upper closure operator satisfying the above requisites. This property constitutes a *minimality* result of the abstract semantics with respect to the property under investigation: if two elements are different in the abstract semantics then there exists a context (i.e., a program, in our case a single substitution) that shows the difference between the two elements in terms of pair-sharing.

In recent years, several researchers have investigated how different abstract domains can be composed or decomposed, enriched or simplified, in order to enjoy a given set of properties. In particular, after an initial attempt in [CFW92], Cortesi, Filé, and Winsborough defined, in [CFW94, CFW98], the notion of *quotient of an abstract interpretation with respect to a certain property.* This notion is intended to isolate, in a given abstract domain, those parts that are useful to compute the selected property.

Note that, in [CFW98], the word "quotient" is used in a context which is different from the usual one. When the authors talk about the quotient of a domain D "with respect to the property P", they mean "with respect to the equivalence relation $\equiv_{\alpha_P}$ induced by the property P". However, such an equivalence relation is defined by

$$d_1 \equiv_{\alpha_P} d_2 \quad \stackrel{\text{def}}{\iff} \quad \forall i \geq 0 : \forall \mu : \alpha_P\bigl(\mu^i(d_1)\bigr) = \alpha_P\bigl(\mu^i(d_2)\bigr), \tag{4.1}$$

where μ is an arbitrary "derived operator", that is, any expression built from operators and elements of the domain D and involving only one variable. Note how this definition significantly differs from the, by now standard, notion of equivalence relation induced by an abstraction function (see [CC77a]), which is formalized as follows:

$$d_1 \equiv_{\alpha_P} d_2 \quad \stackrel{\text{def}}{\iff} \quad \alpha_P(d_1) = \alpha_P(d_2).$$

Our work can thus be considered as an application of [CFW98] where we take set-sharing (SS) as the starting domain and pair-sharing (PS) as the property under investigation. Once put together, the completeness and minimality results mentioned above imply that the domain PSD we have found is exactly the quotient of SS with respect to PS (using the terminology of [CFW98]). It is our opinion that the formalization of the problem in terms of completeness and minimality is more intuitive than the formalization given by (4.1). As a matter of fact, when in [CFW98] the authors prove that Def is the quotient of Pos with respect to the property of groundness, they adopt the same approach.

It is interesting to note that, in the view of another recent result on abstract domain completeness [GR97], PSD is the *least fully-complete extension* (lfce) of PS with respect to SS. The lfce exactly formalizes the methodology requiring to couple a completeness and a minimality result. From a theoretical point of view, the quotient of an abstract interpretation with respect to a property of interest and the least fully-complete extension of an upper closure operator with respect to a reference concrete domain are not equivalent. It is known [CFW94] that the quotient may not exist, while the lfce is always defined (assuming the concrete semantics operators are continuous, as it is almost always the case). However, it is also known [GRS98b] that when the quotient exists it is exactly the same as the lfce, so that the latter has also been called *generalized quotient.* In particular, for the case considered here, these two approaches to the completeness problem in abstract interpretation are equivalent.

4.7 Proofs

Lemma 4.13 *Suppose $sh \in SH$. Then S is redundant for $\rho(sh)$ if and only if S is redundant for sh.*

Proof. Since $sh \subseteq \rho(sh)$, if S is redundant for sh, then S is redundant for $\rho(sh)$.

Suppose that S is redundant for $\rho(sh)$. Then

$$S = \bigcup \{\, \mathrm{pairs}(T) \mid T \in \rho(sh), T \subset S \,\}.$$

Thus, by definition of ρ,

$$S = \bigcup \{\, \mathrm{pairs}(T) \mid T \in sh, T \subset S \,\} \cup \bigcup \{\, \mathrm{pairs}(T) \mid T \text{ is redundant for } sh, T \subset S \,\}.$$

However, if T is redundant for sh, $T = \bigcup \{\, \mathrm{pairs}(U) \mid U \in sh, U \subset T \,\}$, and hence, if $T \subset S$, we have $T \subseteq \bigcup \{\, \mathrm{pairs}(U) \mid U \in sh, U \subset S \,\}$. It follows that,

$$\bigcup \{\, \mathrm{pairs}(T) \mid T \text{ is redundant for } sh, T \subset S \,\} \subseteq \bigcup \{\, \mathrm{pairs}(T) \mid T \in sh, T \subset S \,\}.$$

Thus, $S = \bigcup \{\, \mathrm{pairs}(T) \mid T \in sh, T \subset S \,\}$ and so S is redundant for sh. $\square$

Proof of Theorem 4.4 on page 89. Monotonicity and extensivity of ρ are direct consequences of the definition. For idempotence, suppose that $sh \in SH$. We show that $\rho\big(\rho(sh)\big) = \rho(sh)$. By definition,

$$\rho\big(\rho(sh)\big) = \rho(sh) \cup \{\, S \in SG \mid S \text{ is redundant for } \rho(sh) \,\}.$$

Therefore, by Lemma 4.13,

$$\begin{aligned} \rho\big(\rho(sh)\big) &= \rho(sh) \cup \{\, S \in SG \mid S \text{ is redundant for } sh \,\} \\ &= \rho(sh). \end{aligned}$$

$\square$

Proof of Theorem 4.5 on page 89. Let us define, for each $sh \in SH$,

$$\dot{\rho}(sh) \stackrel{\text{def}}{=} \Big\{\, S \in SG \;\Big|\; \forall x \in S : S \in \mathrm{rel}\big(\{x\}, sh\big)^{\star} \,\Big\}.$$

Let $sh \in SH$: we want to show that $\rho(sh) = \dot{\rho}(sh)$. First suppose $S \in \rho(sh)$. If $S \in sh$, then $S \in \dot{\rho}(sh)$. Suppose $S \notin sh$. Then as S is redundant for sh, we have $S = \{x, x_1, \ldots, x_n\}$ with $n \geq 2$, and, for each x_i there exists a T_i such that $T_i \in sh$, $T_i \subset S$, and $\{x, x_i\} \subseteq T_i$. Thus $S = T_1 \cup \cdots \cup T_n$. As $T_1, \ldots, T_n \in \mathrm{rel}\big(\{x\}, sh\big)$, we have $S \in \mathrm{rel}\big(\{x\}, sh\big)^{\star}$. Since the choice of $x \in S$ was arbitrary, $S \in \dot{\rho}(sh)$.

Secondly, suppose $S \in \dot\rho(sh)$. If $S \in sh$, then $S \in \rho(sh)$. Suppose that $S \notin sh$. Then we need to show that S is redundant for sh. That is, we need to show that $\# S > 2$ and

$$\mathrm{pairs}(S) = \bigcup \{\, \mathrm{pairs}(T) \mid T \in sh, T \subset S \,\}. \tag{4.2}$$

By definition of $\dot\rho(sh)$, for each $x \in S$,

$$S = \bigcup \{\, T \in sh \mid T \subseteq S, x \in T \,\}. \tag{4.3}$$

Since $S \notin sh$, the case $T = S$ can be ruled out in (4.3) obtaining

$$S = \bigcup \{\, T \in sh \mid T \subset S, x \in T \,\}, \tag{4.4}$$

and thus $\# S > 2$. Also, as (4.4) holds for all $x \in S$, $S = \bigcup \{\, T \in sh \mid T \subset S \,\}$. Thus,

$$\mathrm{pairs}(S) \supseteq \bigcup \{\, \mathrm{pairs}(T) \mid T \in sh, T \subset S \,\}. \tag{4.5}$$

Suppose $\{x, y\} \in \mathrm{pairs}(S)$ for some $x, y \in \mathit{Vars}$. Then, by (4.4), there is a $T \in sh$ such that $T \subset S$ and $x, y \in T$ and hence $\{x, y\} \in \mathrm{pairs}(T)$. Hence

$$\mathrm{pairs}(S) \subseteq \bigcup \{\, \mathrm{pairs}(T) \mid T \in sh, T \subset S \,\}. \tag{4.6}$$

Combining (4.5) and (4.6) gives (4.2) as required. □

Since both ρ (by Theorem 4.4) and $(\cdot)^\star$ are upper closure operators it follows that

$$sh_1 \subseteq \rho(sh_2) \iff \rho(sh_1) \subseteq \rho(sh_2), \tag{4.7}$$
$$sh_1 \subseteq sh_2^\star \iff sh_1^\star \subseteq sh_2^\star. \tag{4.8}$$

Lemma 4.14 *For each $sh \in SH$ and each $V \in \wp_{\mathrm{f}}(\mathit{Vars})$, $\overline{\mathrm{rel}}\bigl(V, \rho(sh)\bigr) = \rho\bigl(\overline{\mathrm{rel}}(V, sh)\bigr)$.*

Proof. By Theorem 4.5,

$$\begin{aligned} S \in \rho\bigl(\overline{\mathrm{rel}}(V, sh)\bigr) &\iff \forall x \in S : S = \bigcup \left\{ T \subseteq S \;\middle|\; \begin{array}{l} T \in \mathrm{rel}\bigl(\{x\}, sh\bigr) \\ T \notin \mathrm{rel}\bigl(V, sh\bigr) \end{array} \right\} \\ &\iff S \in \rho(sh) \land S \cap V = \varnothing \\ &\iff S \in \overline{\mathrm{rel}}\bigl(V, \rho(sh)\bigr). \end{aligned}$$

□

Lemma 4.15 *For each $sh_1, sh_2 \in SH$ and each $V \in \wp_{\mathrm{f}}(\mathit{Vars})$,*

$$sh_1 \subseteq \rho(sh_2) \implies \mathrm{rel}(V, sh_1)^\star \subseteq \mathrm{rel}(V, sh_2)^\star.$$

Proof. Suppose $S \in \mathrm{rel}(V, sh_1)$. Then, $S \in sh_1$ and $V \cap S \neq \varnothing$. By the hypothesis, $S \in \rho(sh_2)$. Suppose $x \in V \cap S$. Then, by Theorem 4.5, we have $S = T_1 \cup \cdots \cup T_k$ where, for each $i = 1, \ldots, k$, $x \in T_i$ and $T_i \in sh_2$. Hence, $T_i \in \mathrm{rel}(V, sh_2)$ for $i = 1, \ldots, k$. Thus $S \in \mathrm{rel}(V, sh_2)^\star$. The result then follows from (4.8). □

Lemma 4.16 *Suppose $sh_1, sh_2 \in SH$. Then, for each $(x \mapsto t) \in Bind$,*

$$\rho(sh_1) = \rho(sh_2) \implies \rho\bigl(\mathrm{amgu}(sh_1, x \mapsto t)\bigr) = \rho\bigl(\mathrm{amgu}(sh_2, x \mapsto t)\bigr).$$

Proof. We will show that

$$sh_1 \subseteq \rho(sh_2) \implies \mathrm{amgu}(sh_1, x \mapsto t) \subseteq \rho\bigl(\mathrm{amgu}(sh_2, x \mapsto t)\bigr).$$

The result then follows from (4.7).

Let $v_x \stackrel{\mathrm{def}}{=} \{x\}$, $v_t \stackrel{\mathrm{def}}{=} \mathrm{vars}(t)$. Suppose $S \in \mathrm{amgu}(sh_1, x \mapsto t)$. Then, by definition of amgu,

$$S \in \overline{\mathrm{rel}}(v_x \cup v_t, sh_1) \cup \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_1)^\star, \mathrm{rel}(v_t, sh_1)^\star\bigr).$$

There are two cases:

1. $S \in \overline{\mathrm{rel}}(v_x \cup v_t, sh_1)$. Then $S \in sh_1$ so that, by hypothesis, $S \in \rho(sh_2)$. Hence we have $S \in \overline{\mathrm{rel}}\bigl(v_x \cup v_t, \rho(sh_2)\bigr)$. Thus, by Lemma 4.14,

$$S \in \rho\bigl(\overline{\mathrm{rel}}(v_x \cup v_t, sh_2)\bigr).$$

2. $S \in \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_1)^\star, \mathrm{rel}(v_t, sh_1)^\star\bigr)$. Then, $S = T \cup R$, where $T \in \mathrm{rel}(v_x, sh_1)^\star$ and $R \in \mathrm{rel}(v_t, sh_1)^\star$. By Lemma 4.15, $T \in \mathrm{rel}(v_x, sh_2)^\star$ and $R \in \mathrm{rel}(v_t, sh_2)^\star$. Hence,

$$S \in \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_2)^\star, \mathrm{rel}(v_t, sh_2)^\star\bigr).$$

Combining cases 1 and 2 we obtain

$$S \in \rho\bigl(\overline{\mathrm{rel}}(v_x \cup v_t, sh_2)\bigr) \cup \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_2)^\star, \mathrm{rel}(v_t, sh_2)^\star\bigr).$$

Hence as ρ is extensive and monotonic

$$S \in \rho\Bigl(\overline{\mathrm{rel}}(v_x \cup v_t, sh_2) \cup \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_2)^\star, \mathrm{rel}(v_t, sh_2)^\star\bigr)\Bigr),$$

and hence $S \in \rho\bigl(\mathrm{amgu}(sh_2, x \mapsto t)\bigr)$. □

Corollary 4.17 *Suppose $d_1, d_2 \in SS$. Then, for each $(x \mapsto t) \in Bind$,*

$$\rho(d_1) = \rho(d_2) \implies \rho\bigl(\mathrm{Amgu}(d_1, x \mapsto t)\bigr) = \rho\bigl(\mathrm{Amgu}(d_2, x \mapsto t)\bigr).$$

Proof. There are three cases:

1. $d_1 = \bot$. By hypothesis, $d_2 = \bot$. Straightforward.

2. $d_1 = \top$. By hypothesis, $d_2 = \top$. Straightforward.

3. $d_1 = (sh_1, VI)$. By hypothesis, we have $d_2 = (sh_2, VI)$ and $\rho(sh_1) = \rho(sh_2)$. Let $V \stackrel{\mathrm{def}}{=} \mathrm{vars}(x \mapsto t) \setminus VI$ and $s_V \stackrel{\mathrm{def}}{=} \bigl\{\, \{y\} \bigm| y \in V \,\bigr\}$. By definition of Amgu and ρ, we have

$$\begin{aligned}\rho\Bigl(\mathrm{Amgu}\bigl((sh_1, VI), x \mapsto t\bigr)\Bigr) &= \rho\Bigl(\bigl(\mathrm{amgu}(sh_1 \cup s_V, x \mapsto t), VI \cup V\bigr)\Bigr)\\ &= \Bigl(\rho\bigl(\mathrm{amgu}(sh_1 \cup s_V, x \mapsto t)\bigr), VI \cup V\Bigr),\end{aligned}$$

and, similarly,

$$\rho\Bigl(\mathrm{Amgu}\bigl((sh_2, VI), x \mapsto t\bigr)\Bigr) = \Bigl(\rho\bigl(\mathrm{amgu}(sh_2 \cup s_V, x \mapsto t)\bigr), VI \cup V\Bigr).$$

Note that, since s_V contains singletons only, it holds

$$\rho(sh_1) = \rho(sh_2) \implies \rho(sh_1 \cup s_V) = \rho(sh_2 \cup s_V).$$

Thus, the proof is completed by applying Lemma 4.16.

□

Theorem 4.18 *Suppose $d_1, d_2 \in SS$. Then, for each $\sigma \in RSubst$,*

$$\rho(d_1) = \rho(d_2) \implies \rho\bigl(\mathrm{Aunify}(d_1, \sigma)\bigr) = \rho\bigl(\mathrm{Aunify}(d_2, \sigma)\bigr).$$

Proof. The proof is by induction on the number of bindings in σ.

The base case, when $\#\,\sigma = 0$ and thus $\sigma = \varnothing$, follows easily by the definition of Aunify.

For the inductive case, when $\#\,\sigma = n > 0$, take $(x \mapsto t) \in \sigma$ and let $\sigma' = \sigma \setminus \{x \mapsto t\}$. Since $\#\,\sigma' = n - 1$, by applying the inductive hypothesis we have that, for all $d_1', d_2' \in SS$,

$$\rho(d_1') = \rho(d_2') \implies \rho\bigl(\mathrm{Aunify}(d_1', \sigma')\bigr) = \rho\bigl(\mathrm{Aunify}(d_2', \sigma')\bigr). \tag{4.9}$$

Moreover, by definition of Aunify, we obtain

$$\begin{aligned}\rho\bigl(\mathrm{Aunify}(d_1, \sigma)\bigr) &= \rho\Bigl(\mathrm{Aunify}\bigl(\mathrm{Amgu}(d_1, x \mapsto t), \sigma'\bigr)\Bigr),\\ \rho\bigl(\mathrm{Aunify}(d_2, \sigma)\bigr) &= \rho\Bigl(\mathrm{Aunify}\bigl(\mathrm{Amgu}(d_2, x \mapsto t), \sigma'\bigr)\Bigr).\end{aligned}$$

Therefore, to complete the proof, it is sufficient to instantiate (4.9) by taking, for $i \in \{1, 2\}$, $d_i' = \mathrm{Amgu}(d_i, x \mapsto t)$ and prove that $\rho(d_1') = \rho(d_2')$. The latter is indeed a consequence of Corollary 4.17. □

Lemma 4.19 *Suppose $sh_1, sh_2 \in SH$. Then $\rho(sh_1 \cup sh_2) = \rho\bigl(\rho(sh_1) \cup \rho(sh_2)\bigr)$.*

Proof. This is a classical property of upper closure operators [GHK+80, War42]. We prove it here for completeness. By monotonicity of ρ, we have $\rho(sh_1) \cup \rho(sh_2) \subseteq \rho(sh_1 \cup sh_2)$. Hence, by monotonicity and idempotence of ρ, we obtain $\rho\bigl(\rho(sh_1) \cup \rho(sh_2)\bigr) \subseteq \rho(sh_1 \cup sh_2)$. By extensiveness of ρ, $sh_1 \cup sh_2 \subseteq \rho(sh_1) \cup \rho(sh_2)$, and hence, again by monotonicity of ρ, we conclude $\rho(sh_1 \cup sh_2) \subseteq \rho\bigl(\rho(sh_1) \cup \rho(sh_2)\bigr)$. $\square$

Theorem 4.20 *Suppose that $d_1, d_2 \in SS$. Then, for all $d' \in SS$,*

$$\rho(d_1) = \rho(d_2) \implies \rho\bigl(\mathrm{Alub}(d', d_1)\bigr) = \rho\bigl(\mathrm{Alub}(d', d_2)\bigr).$$

Proof. There are three cases:

1. $d_1 = \bot$. By hypothesis, $d_2 = \bot$. Straightforward.

2. $d_1 = \top$. By hypothesis, $d_2 = \top$. Straightforward.

3. $d_1 = (sh_1, VI)$. By hypothesis, $d_2 = (sh_2, VI)$ and $\rho(sh_1) = \rho(sh_2)$. Let $d' \in SS$. Again, if $d' \in \{\bot, \top\}$ the proof is straightforward. Thus, let $d' = (sh', VI')$; if $VI \neq VI'$ then the proof is straightforward, so we consider $VI = VI'$. By definition of Alub, for $i \in \{1, 2\}$, we have

$$\begin{aligned} \rho\Bigl(\mathrm{Alub}\bigl((sh', VI), (sh_i, VI)\bigr)\Bigr) &= \rho\bigl((sh' \cup sh_i, VI)\bigr) \\ &= \bigl(\rho(sh' \cup sh_i), VI\bigr). \end{aligned}$$

   We conclude the proof by applying Lemma 4.19.

$\square$

Lemma 4.21 *Suppose $sh_1, sh_2 \in SH$. Then, for each $V \in \wp_{\mathrm{f}}(Vars)$,*

$$\rho(sh_1) = \rho(sh_2) \implies \rho\bigl(\mathrm{aexists}(sh_1, V)\bigr) = \rho\bigl(\mathrm{aexists}(sh_2, V)\bigr).$$

Proof. We show that

$$sh_1 \subseteq \rho(sh_2) \implies \mathrm{aexists}(sh_1, V) \subseteq \rho\bigl(\mathrm{aexists}(sh_2, V)\bigr).$$

The result then follows from (4.7).

Suppose $sh_1 \subseteq \rho(sh_2)$ and $S \in \mathrm{aexists}(sh_1, V)$. Then, as aexists is monotonic, we have $S \in \mathrm{aexists}\bigl(\rho(sh_2), V\bigr)$. Note that, if $S = \{x\}$ and $x \in V$, the result is trivial. Otherwise, by definition of aexists, there exists $S' \in \rho(sh_2)$ such that $S = S' \cap V$. By Theorem 4.5,

$$\forall x \in S' : \exists T_1, \ldots, T_k \in \mathrm{rel}\bigl(\{x\}, sh_2\bigr) \;.\; S = \Bigl(\bigcup_{i=1}^{k} T_i\Bigr) \cap V,$$

hence

$$\forall x \in S : \exists T_1, \ldots, T_k \in \mathrm{rel}\bigl(\{x\}, sh_2\bigr) \;.\; S = \left(\bigcup_{i=1}^{k} T_i\right) \cap V,$$

and hence

$$\forall x \in S : \exists T_1, \ldots, T_k \in \mathrm{rel}\bigl(\{x\}, sh_2\bigr) \;.\; S = \bigcup_{i=1}^{k} (T_i \cap V).$$

However,

$$\forall x \in S : (T_1 \cap V), \ldots, (T_k \cap V) \in \mathrm{rel}\bigl(\{x\}, \mathrm{aexists}(sh_2, V)\bigr),$$

and thus $S \in \rho\bigl(\mathrm{aexists}(sh_2, V)\bigr)$. $\square$

Theorem 4.22 *Suppose that $d_1, d_2 \in SS$ and $V \in \wp_{\mathrm{f}}(Vars)$. Then*

$$\rho(d_1) = \rho(d_2) \implies \rho\bigl(\mathrm{Aproj}(d_1, V)\bigr) = \rho\bigl(\mathrm{Aproj}(d_2, V)\bigr).$$

Proof. There are three cases:

1. $d_1 = \bot$. By hypothesis, $d_2 = \bot$. Straightforward.

2. $d_1 = \top$. By hypothesis, $d_2 = \top$. Straightforward.

3. $d_1 = (sh_1, VI)$. By hypothesis, $d_2 = (sh_2, VI)$ and $\rho(sh_1) = \rho(sh_2)$. By definition of Aproj, for each $i \in \{1, 2\}$, we have

$$\rho\Bigl(\mathrm{Aproj}\bigl((sh_i, VI), V\bigr)\Bigr) = \bigl(\rho(sh_i'), V\bigr),$$

where

$$sh_i' = \Bigl(\mathrm{aexists}(sh_i, VI \setminus V) \setminus \bigl\{\, \{x\} \bigm| x \in VI \setminus V \,\bigr\}\Bigr) \cup \bigl\{\, \{x\} \bigm| x \in V \setminus VI \,\bigr\}.$$

Since singletons do not affect the operator ρ, we have

$$\rho(sh_i') = \Bigl(\rho\bigl(\mathrm{aexists}(sh_i, VI \setminus V)\bigr) \setminus \bigl\{\, \{x\} \bigm| x \in VI \setminus V \,\bigr\}\Bigr) \cup \bigl\{\, \{x\} \bigm| x \in V \setminus VI \,\bigr\}.$$

Thus, we can conclude the proof by applying Lemma 4.21.

$\square$

Proof of Theorem 4.7 on page 90. Statements 1, 2 and 3 follow, respectively, from Theorems 4.18, 4.20 and 4.22. $\square$

Lemma 4.23 *Let $\sigma \in RSubst$ be such that $t \in GTerms$, for all $(x \mapsto t) \in \sigma$. Then, for all $(sh, VI) \in SS$ we have*

$$\mathrm{Aunify}\bigl((sh, VI), \sigma\bigr) = \Bigl(\overline{\mathrm{rel}}\bigl(\mathrm{dom}(\sigma), sh\bigr), VI \cup \mathrm{dom}(\sigma)\Bigr).$$

Proof. By induction on the number of bindings of σ. The base case, when $\#\sigma = 0$, is obvious. For the inductive case, let $\#\sigma > 0$, $(x \mapsto t) \in \sigma$ and $\sigma' = \sigma \setminus \{x \mapsto t\}$. Note that $\mathrm{vars}(t) = \varnothing$, so that

$$\begin{aligned}\mathrm{Amgu}\bigl((sh, VI), x \mapsto t\bigr) &= \Bigl(\overline{\mathrm{rel}}\bigl(\{x\}, sh'\bigr) \cup \mathrm{bin}\Bigl(\mathrm{rel}\bigl(\{x\}, sh'\bigr)^{\star}, \mathrm{rel}\bigl(\varnothing, sh'\bigr)^{\star}\Bigr), VI \cup \{x\}\Bigr) \\ &= \Bigl(\overline{\mathrm{rel}}\bigl(\{x\}, sh\bigr), VI \cup \{x\}\Bigr),\end{aligned}$$

where $sh' = sh \cup \bigl\{\{x\}\bigr\}$, if $x \notin VI$, and $sh' = sh$, otherwise.

Thus, by applying the definition of Aunify, the above result and the inductive hypothesis, we have

$$\begin{aligned}\mathrm{Aunify}\bigl((sh, VI), \sigma\bigr) &= \mathrm{Aunify}\Bigl(\mathrm{Amgu}\bigl((sh, VI), x \mapsto t\bigr), \sigma'\Bigr) \\ &= \mathrm{Aunify}\Bigl(\Bigl(\overline{\mathrm{rel}}\bigl(\{x\}, sh\bigr), VI \cup \{x\}\Bigr), \sigma'\Bigr) \\ &= \Bigl(\overline{\mathrm{rel}}\Bigl(\mathrm{dom}(\sigma'), \overline{\mathrm{rel}}\bigl(\{x\}, sh\bigr)\Bigr), VI \cup \{x\} \cup \mathrm{dom}(\sigma')\Bigr) \\ &= \Bigl(\overline{\mathrm{rel}}\bigl(\mathrm{dom}(\sigma), sh\bigr), VI \cup \mathrm{dom}(\sigma)\Bigr).\end{aligned}$$

□

Theorem 4.24 *Let $d_1 \stackrel{\mathrm{def}}{=} (sh_1, VI)$ and $d_2 \stackrel{\mathrm{def}}{=} (sh_2, VI)$ be two elements of SS. Then $\rho(d_1) \neq \rho(d_2)$ implies*

$$\exists \sigma \in RSubst \mathrel{.} \alpha_{PS}\bigl(\mathrm{Aunify}(d_1, \sigma)\bigr) \neq \alpha_{PS}\bigl(\mathrm{Aunify}(d_2, \sigma)\bigr).$$

Proof. Suppose $\rho(d_1) \neq \rho(d_2)$. Then it follows that $\rho(sh_1) \neq \rho(sh_2)$. We assume that $S \in \rho(sh_1) \setminus \rho(sh_2)$. (If such an S does not exist we simply swap sh_1 and sh_2.)

Let $a \in GTerms \cap HTerms$ be a ground and finite term and let

$$\sigma_S \stackrel{\mathrm{def}}{=} \{\, x \mapsto a \mid x \in VI \setminus S \,\}.$$

For $i = 1, 2$, we define

$$sh_i^S \stackrel{\mathrm{def}}{=} \{\, T \mid T \subseteq S, T \in sh_i \,\},$$

so that, by Lemma 4.23, we have

$$(sh_i^S, VI) = \mathrm{Aunify}\bigl((sh_i, VI), \sigma_S\bigr).$$

Suppose first that $\# S \geq 2$. For this case, let $\sigma \stackrel{\text{def}}{=} \sigma_S$. Then,

$$(sh_i^S, VI) = \operatorname{Aunify}\bigl((sh_i, VI), \sigma\bigr).$$

Since $S \notin \rho(sh_2)$, there exists a pair $P \subseteq S$ such that

$$\forall T \in sh_2 : T \subseteq S \implies P \nsubseteq T.$$

However, by definition of sh_2^S, this is the same as saying $\forall T \in sh_2^S : P \nsubseteq T$ or, equivalently,

$$\bigl(\{P\}, VI\bigr) \npreceq_{PS} \alpha_{PS}\bigl((sh_2^S, VI)\bigr).$$

Also, since $S \in \rho(sh_1)$, there exists $T' \in sh_1$ such that $T' \subseteq S$ and $P \subseteq T'$. Moreover, $T' \in sh_1^S$, so that

$$\bigl(\{P\}, VI\bigr) \preceq_{PS} \alpha_{PS}\bigl((sh_1^S, VI)\bigr).$$

Thus

$$\alpha_{PS}\bigl((sh_1^S, VI)\bigr) \neq \alpha_{PS}\bigl((sh_2^S, VI)\bigr).$$

Second, suppose that $\# S = 1$. Let $\sigma = \sigma_S \cup \{x \mapsto z\}$, where $S = \{x\}$ and $z \in \mathit{Vars} \setminus VI$ is a new variable. By applying the definition of Aunify we obtain, for each $i \in \{1, 2\}$,

$$\begin{aligned}
\operatorname{Aunify}\bigl((sh_i, VI), \sigma\bigr) &= \operatorname{Aunify}\bigl((sh_i, VI), \sigma_S \cup \{x \mapsto z\}\bigr) \\
&= \operatorname{Aunify}\bigl((sh_i^S, VI), \{x \mapsto z\}\bigr) \\
&= \operatorname{Amgu}\bigl((sh_i^S, VI), x \mapsto z\bigr) \\
&= \Bigl(\operatorname{amgu}\Bigl(sh_i^S \cup \bigl\{\{z\}\bigr\}, x \mapsto z\Bigr), VI \cup \{z\}\Bigr).
\end{aligned}$$

Also, by definition of sh_1^S and sh_2^S, we have $sh_1^S = \bigl\{\{x\}\bigr\}$ and $sh_2^S = \varnothing$. Hence

$$\begin{aligned}
\operatorname{amgu}\Bigl(sh_1^S \cup \bigl\{\{z\}\bigr\}, x \mapsto z\Bigr) &= \bigl\{\{x, z\}\bigr\}, \\
\operatorname{amgu}\Bigl(sh_2^S \cup \bigl\{\{z\}\bigr\}, x \mapsto z\Bigr) &= \varnothing.
\end{aligned}$$

Thus,

$$\alpha_{PS}\Bigl(\operatorname{Aunify}\bigl((sh_1, VI), \sigma\bigr)\Bigr) = \Bigl(\bigl\{\{x, z\}\bigr\}, VI \cup \{z\}\Bigr)$$

is distinct from

$$\alpha_{PS}\Bigl(\operatorname{Aunify}\bigl((sh_2, VI), \sigma\bigr)\Bigr) = \bigl(\varnothing, VI \cup \{z\}\bigr).$$

□

Proof of Theorem 4.8 on page 90. We have four cases. If one of the following holds

1. $d_1 = \bot$ or $d_2 = \bot$,

2. $d_1 = \top$ or $d_2 = \top$,

3. $d_1 = (sh_1, VI_1)$, $d_2 = (sh_2, VI_2)$, and $VI_1 \neq VI_2$,

then we simply take $\sigma = \varnothing$. The last case,

4. $d_1 = (sh_1, VI)$ and $d_2 = (sh_2, VI)$,

has been proved as Theorem 4.24. □

Proof of Theorem 4.9 on page 91. From $sh_1 \cap sh_2 \subseteq sh_2$ and the monotonicity of ρ, we have that $\rho(sh_1 \cap sh_2) \subseteq \rho(sh_2)$.

To prove the reverse inclusion, we will show $\rho(sh_1 \cap sh_2) \supseteq sh_1 \cup sh_2$. Then, by ρ monotonicity and idempotence, we obtain

$$\begin{aligned}\rho(sh_1 \cap sh_2) &= \rho\bigl(\rho(sh_1 \cap sh_2)\bigr)\\ &\supseteq \rho(sh_1 \cup sh_2)\\ &\supseteq \rho(sh_2).\end{aligned}$$

Let $S \in sh_1 \cup sh_2$. We will prove $S \in \rho(sh_1 \cap sh_2)$ by induction on $\# S$.

Let $\# S \leq 2$. As $S \in \rho(sh_1) = \rho(sh_2)$ we have, by definition of ρ, both $S \in sh_1$ and $S \in sh_2$. Thus $S \in sh_1 \cap sh_2$ and the result follows by ρ extensivity.

Let now $\# S = k > 2$. There are three cases:

a) If $S \in sh_1 \cap sh_2$ then the result follows, as before, by ρ extensivity.

b) Let $S \in sh_1 \setminus sh_2$. As $S \in \rho(sh_1) = \rho(sh_2)$, we have $S \in \rho(sh_2) \setminus sh_2$. Thus, by the definition of ρ,
$$\mathrm{pairs}(S) = \bigcup \bigl\{ \mathrm{pairs}(T) \bigm| T \in sh_2, T \subset S \bigr\}.$$
Note that, for all such T, we have $\# T < k$ and thus, by the inductive hypothesis, $T \in \rho(sh_1 \cap sh_2)$. Hence, by the definition of ρ and ρ idempotence, we have
$$S \in \rho\bigl(\rho(sh_1 \cap sh_2)\bigr) = \rho(sh_1 \cap sh_2).$$

c) The case for $S \in sh_2 \setminus sh_1$ is symmetric to case b) above.

□

Lemma 4.25 *S is redundant for sh if and only if S is redundant for $sh \setminus \{S\}$.*

Proof. We have that S is redundant for sh if and only if $\# S > 2$ and

$$\mathrm{pairs}(S) = \bigcup \bigl\{ \mathrm{pairs}(T) \bigm| T \in sh, T \subset S \bigr\}$$

if and only if $\# S > 2$ and

$$\text{pairs}(S) = \bigcup \{ \text{pairs}(T) \bigm| T \in sh \setminus \{S\}, T \subset S \}$$

if and only if S is redundant for $sh \setminus \{S\}$. □

Corollary 4.26 *Let $S \in sh$. Then S is redundant for sh if and only if $\rho(sh) = \rho\big(sh \setminus \{S\}\big)$.*

Proof. By the monotonicity of ρ we have $\rho(sh) \supseteq \rho\big(sh \setminus \{S\}\big)$. Assume S is redundant for sh. Then by Lemma 4.25, S is redundant for $sh \setminus \{S\}$ and thus $S \in \rho\big(sh \setminus \{S\}\big)$. By the extensivity of ρ we have also $sh \setminus \{S\} \subseteq \rho\big(sh \setminus \{S\}\big)$, and thus $sh \subseteq \rho\big(sh \setminus \{S\}\big)$. By the monotonicity and idempotence of ρ we can conclude that $\rho(sh) \subseteq \rho\big(sh \setminus \{S\}\big)$.

Assume now $\rho(sh) = \rho\big(sh \setminus \{S\}\big)$. Since $S \in sh$, by extensivity $S \in \rho(sh) = \rho\big(sh \setminus \{S\}\big)$. Thus S is redundant for $sh \setminus \{S\}$. □

Proof of Theorem 4.10 on page 91. Let

$$sh_{\text{red}} \stackrel{\text{def}}{=} sh \setminus \{ S \in SG \mid S \text{ is redundant for } sh \}. \tag{4.10}$$

We first prove that $\rho(sh_{\text{red}}) = \rho(sh)$.

For each $S \in SG$ such that S is redundant for sh, let $sh_S \stackrel{\text{def}}{=} sh \setminus \{S\}$ and note that $sh_{\text{red}} = \bigcap \{ sh_S \mid S \text{ is redundant for } sh \}$. By Corollary 4.26, we have $\rho(sh_S) = \rho(sh)$. Thus we can apply Theorem 4.9 and obtain

$$\rho(sh_{\text{red}}) = \rho\Big(\bigcap \{ sh_S \mid S \text{ is redundant for } sh \}\Big) = \rho(sh).$$

Having proved $\rho(sh_{\text{red}}) = \rho(sh)$, we only need to prove

$$sh_{\text{red}} = \bigcap \{ sh' \in SH \bigm| \rho(sh') = \rho(sh_{\text{red}}) \}.$$

The inclusion $sh_{\text{red}} \supseteq \bigcap \{ sh' \in SH \bigm| \rho(sh') = \rho(sh_{\text{red}}) \}$ is obvious, since sh_{red} is one of the sets that are intersected in the right hand side. For the reverse inclusion, let $sh' \in SH$ be such that $\rho(sh') = \rho(sh_{\text{red}})$. We have:

$$\begin{aligned}
S \in sh_{\text{red}} &\iff S \in sh \setminus \{ T \in SG \mid T \text{ red. for } sh \} && \text{[eq. (4.10)]} \\
&\iff S \in \rho(sh) \setminus \{ T \in SG \mid T \text{ red. for } sh \} && \text{[Def. 4.3]} \\
&\iff S \in \rho(sh) \setminus \{ T \in SG \bigm| T \text{ red. for } \rho(sh) \} && \text{[Lemma 4.13]} \\
&\iff S \in \rho(sh') \setminus \{ T \in SG \bigm| T \text{ red. for } \rho(sh') \} && [\rho(sh) = \rho(sh')] \\
&\iff S \in sh' \setminus \{ T \in SG \bigm| T \text{ red. for } \rho(sh') \} && [\text{def. } \rho] \\
&\iff S \in sh' \setminus \{ T \in SG \mid T \text{ red. for } sh' \} && \text{[Lemma 4.13]} \\
&\implies S \in sh'.
\end{aligned}$$

Hence $sh_{\text{red}} \subseteq \bigcap \{ sh' \in SH \bigm| \rho(sh') = \rho(sh_{\text{red}}) \}$. □

Proof of Theorem 4.11 on page 92. The inclusion $sh^\star \supseteq \rho\bigl(\mathrm{bin}(sh, sh)\bigr)$ follows from Theorem 4.5 and the definition of the 'bin' operator. We now prove the other inclusion $sh^\star \subseteq \rho\bigl(\mathrm{bin}(sh, sh)\bigr)$. Let $S \in sh^\star$. Then

$$\exists T_1, \dots, T_n \in sh \;.\; S = \bigcup_{i=1}^{n} T_i, \qquad \text{with } n \geq 1.$$

If $S \in \mathrm{bin}(sh, sh)$, then, by definition, $S \in \rho\bigl(\mathrm{bin}(sh, sh)\bigr)$. Suppose now $S \notin \mathrm{bin}(sh, sh)$. Then $\# S > 1$ and there exists $\{x, y\} \in \mathrm{pairs}(S)$. Then there must exist $i, j \in \{1, \dots, n\}$ (note that i and j need not be distinct) such that $x \in T_i$ and $y \in T_j$. This implies $\{x, y\} \in \mathrm{pairs}(T_i \cup T_j)$. However, $T_i \cup T_j \in \mathrm{bin}(sh, sh)$. Hence, as $S \notin \mathrm{bin}(sh, sh)$, $T_i \cup T_j \subset S$. Since the choices of $\{x, y\} \in \mathrm{pairs}(S)$ and $i, j \in \{1, \dots, n\}$ such that $x \in T_i$ and $y \in T_j$ were arbitrary, S is redundant for $\mathrm{bin}(sh, sh)$. $\square$

Lemma 4.27 *For each $sh_1, sh_2 \in SH$,*

$$\rho\bigl(\mathrm{bin}(sh_1, sh_2)\bigr) = \rho\Bigl(\mathrm{bin}\bigl(\rho(sh_1), \rho(sh_2)\bigr)\Bigr).$$

Proof. By the monotonicity of 'bin' and ρ, we have

$$\rho\bigl(\mathrm{bin}(sh_1, sh_2)\bigr) \subseteq \rho\Bigl(\mathrm{bin}\bigl(\rho(sh_1), \rho(sh_2)\bigr)\Bigr).$$

Thus, we must show that

$$\rho\Bigl(\mathrm{bin}\bigl(\rho(sh_1), \rho(sh_2)\bigr)\Bigr) \subseteq \rho\bigl(\mathrm{bin}(sh_1, sh_2)\bigr).$$

Since ρ is monotonic and idempotent, we just need to show that

$$\mathrm{bin}\bigl(\rho(sh_1), \rho(sh_2)\bigr) \subseteq \rho\bigl(\mathrm{bin}(sh_1, sh_2)\bigr).$$

Using Theorem 4.5 we have:

$$\begin{aligned}
&S \in \mathrm{bin}\bigl(\rho(sh_1), \rho(sh_2)\bigr)\\
&\quad\iff S = S_1 \cup S_2 \text{ where } S_1 \in \rho(sh_1) \text{ and } S_2 \in \rho(sh_2)\\
&\quad\iff S = S_1 \cup S_2 \text{ where, for each } i = 1,\, 2,\\
&\qquad\qquad \forall x \in S_i : S_i = \bigcup \Bigl\{\, T_i \subseteq S \Bigm| T_i \in \mathrm{rel}\bigl(\{x\}, sh_i\bigr) \,\Bigr\}\\
&\quad\implies \forall x \in S : S = \bigcup \Bigl\{\, T \subseteq S \Bigm| T \in \mathrm{rel}\bigl(\{x\}, \mathrm{bin}(sh_1, sh_2)\bigr) \,\Bigr\}\\
&\quad\iff S \in \rho\bigl(\mathrm{bin}(sh_1, sh_2)\bigr).
\end{aligned}$$

$\square$

Proof of Theorem 4.12 on page 92. By Definition 3.43 for amgu,

$$\rho\bigl(\mathrm{amgu}(sh, x \mapsto t)\bigr) = \rho\bigl(sh_- \cup \mathrm{bin}(sh_x^\star, sh_t^\star)\bigr).$$

By Lemma 4.19, $\rho\bigl(\mathrm{amgu}(sh, x \mapsto t)\bigr) = \rho\Bigl(\rho(sh_-) \cup \rho\bigl(\mathrm{bin}(sh_x^\star, sh_t^\star)\bigr)\Bigr)$. However, by Theorem 4.11, $\mathrm{bin}(sh_x^\star, sh_t^\star) = \mathrm{bin}\bigl(\rho(sh_x^2), \rho(sh_t^2)\bigr)$. By Lemma 4.27 and the idempotence of ρ, $\rho\bigl(\mathrm{bin}(sh_x^\star, sh_t^\star)\bigr) = \rho\bigl(\mathrm{bin}(sh_x^2, sh_t^2)\bigr)$. Thus, by Lemma 4.19,

$$\rho\bigl(\mathrm{amgu}(sh, x \mapsto t)\bigr) = \rho\bigl(sh_- \cup \mathrm{bin}(sh_x^2, sh_t^2)\bigr).$$

□

4.8 Summary

We have questioned whether the set-sharing domain *SS* is the right choice for tracking pair-sharing between program variables. The answer turned out to be negative. We have presented a new domain *PSD* that is, at the same time, a strict abstraction of *SS* and as precise as *SS* on pair-sharing. We have also shown that no abstract domain weaker than *PSD* can enjoy this last property. This work has led us to an important theoretical result that is very likely to be valuable even from a practical perspective: the exponential star-union operation in the abstract unification procedure can be safely replaced by the binary union operation, which has quadratic complexity, therefore obtaining a new abstract unification procedure characterized by a polynomial complexity in the number of sharing groups of the abstract description.

Chapter 5

Set-Sharing and Complementation

Complementation, the inverse of the reduced product operation, is a technique for systematically finding minimal decompositions of abstract domains. Filé and Ranzato advanced the state of the art by introducing a simple method for computing a complement. As an application, they considered the extraction by complementation of the pair-sharing domain PS from the Jacobs and Langen's set-sharing domain SS. However, since the result of this operation was still SS, they concluded that PS was too abstract for this. Here, we show that the reason for this result lies not with PS but with SS and, more precisely, with the redundant information contained in SS with respect to ground-dependencies and pair-sharing. In fact, a proper decomposition is obtained if the non-redundant version of SS, PSD, is substituted for SS. To establish the results for PSD, we define a general schema for subdomains of SS that includes PSD and Def as special cases. This sheds new light on the structure of PSD and exposes a natural though unexpected connection between Def and PSD. Moreover, we substantiate the claim that complementation *alone* is not sufficient to obtain *truly minimal* decompositions of domains. The right solution to this problem is to *first* remove redundancies by computing the quotient of the domain with respect to the observable behavior, and only *then* decompose it by complementation.

> **Note:** this chapter is mainly based on the results of [ZHB99]; an extended version will appear in [ZHB02].

5.1 Sharing Domains as Test-Cases

Complementation [CFG+97], which is the inverse of the well-known reduced product operation [CC79], can systematically obtain minimal decompositions of complex abstract domains. It has been argued that these decompositions would be useful in finding space saving representations for domains and to simplify domain verification problems.

In [FR96], Filé and Ranzato presented a new method for computing the complement, which is simpler than the original proposal by Cortesi et al. [CFG+95, CFG+97] because it has the advantage that, in order to compute the complement, only a relatively small number of elements (namely the *meet-irreducible* elements of the reference domain) need be considered. As an application of this method, the authors considered the Jacobs and Langen's set-sharing domain [JL92] and minimally decomposed SS into three components; using the words of the authors [FR96, Section 1]:

> "[...] each representing one of the elementary properties that coexist in the elements of *Sharing*, and that are as follows: (i) the ground-dependency information; (ii) the pair-sharing information, or equivalently variable independence; (iii) the set-sharing information, without variable independence and ground-dependency."

However, this decomposition did not use the usual domain PS for pair-sharing. Filé and Ranzato observed that the complement of the pair-sharing domain PS with respect to SS is again SS and concluded that PS was too abstract to be extracted from SS by means of complementation. Thus, in order to obtain their non-trivial decomposition of SS, they used a different (and somewhat unnatural) definition for an alternative pair-sharing domain, called PS'. The nature of PS' and its connection with PS will be examined more carefully at the end of this chapter.

We noticed that the reason why Filé and Ranzato obtained this result was not to be found in the definition of PS, which accurately represents the property of pair-sharing, but in the use of the domain SS. As observed in Chapter 4, for groundness and pair-sharing the domain SS includes redundant elements. In this chapter we show that a proper decomposition, using the natural definition of the pair-sharing domain PS, can be obtained by applying the method given in [FR96] to the non-redundant PSD domain, instead of SS. Moreover, we show that PS is *exactly* one of the components obtained by complementation of PSD. Thus the problem exposed by Filé and Ranzato was, in fact, due to the "information preserving" property of complementation, as any factorization obtained in this way is such that the reduced product of the factors gives back the original domain. In particular, any factorization of SS has to encode the redundant information identified in Chapter 4. We will show that such a problem disappears when PSD is used as the reference domain.

To establish the results for PSD, we define a general schema for subdomains of SS that includes, as special cases, both PSD and the groundness dependency domain Def [AMSS98]. This sheds new light on the structure of the domain PSD, which is smaller but significantly more involved than SS. Moreover, we discover a natural connection between the abstract domains Def and PSD. The results confirm that PSD is, in fact, the "appropriate" abstraction of the set-sharing domain SS that has to be considered when groundness and pair-sharing are the properties of interest.

5.2 Lattice Theory: a Supplement

Besides the basic notions introduced in Chapter 2, the technical results presented in this chapter also relies on the following lattice theory concepts and results.

A complete lattice C is *meet-continuous* if for any chain $Y \subseteq C$ and each $x \in C$,

$$x \wedge \left(\bigvee Y\right) = \bigvee_{y \in Y} (x \wedge y).$$

Most domains for abstract interpretation [CFG+97] and, in particular, all the domains considered here are meet-continuous.

Assume that C is a meet-continuous lattice. Then the inverse of the reduced product operation, called *weak relative pseudo-complement*, is well defined and given as follows. Let $\rho, \rho_1 \in \mathrm{uco}(C)$ be such that $\rho \sqsubseteq \rho_1$. Then

$$\rho \sim \rho_1 \stackrel{\text{def}}{=} \bigsqcup \{ \rho_2 \in \mathrm{uco}(C) \mid \rho_1 \sqcap \rho_2 = \rho \}.$$

Given $\rho \in \mathrm{uco}(C)$, the *weak pseudo-complement* (or, by an abuse of terminology now customary in the field of abstract interpretation, simply *complement*) of ρ is denoted by $id_C \sim \rho$, where id_C is the identity over C. Let $D_i \stackrel{\text{def}}{=} \rho_{D_i}(C)$ with $\rho_{D_i} \in \mathrm{uco}(C)$ for $i = 1, \ldots, n$. Then $\{ D_i \mid 1 \leq i \leq n \}$ is a *decomposition for* C if $C = D_1 \sqcap \cdots \sqcap D_n$. The decomposition is also called *minimal* if, for each $k \in \mathbb{N}$ with $1 \leq k \leq n$ and each $E_k \in \mathrm{uco}(C)$, $D_k \sqsubset E_k$ implies

$$C \sqsubset D_1 \sqcap \cdots \sqcap D_{k-1} \sqcap E_k \sqcap D_{k+1} \sqcap \cdots \sqcap D_n.$$

Assume now that C is a complete lattice. If $X \subseteq C$, then $\mathrm{Moore}(X)$ denotes the *Moore completion of* X, namely,

$$\mathrm{Moore}(X) \stackrel{\text{def}}{=} \left\{ \bigwedge Y \mid Y \subseteq X \right\}.$$

We say that C *is meet-generated by* X if $C = \mathrm{Moore}(X)$. An element $x \in C$ is *meet-irreducible* if

$$\forall y, z \in C : \big((x = y \wedge z) \implies (x = y \text{ or } x = z)\big).$$

The set of meet-irreducible elements of a complete lattice C is denoted by $\mathrm{MI}(C)$. Note that $\top \in \mathrm{MI}(C)$. An element $x \in C$ is a *dual-atom* if $x \neq \top$ and, for each $y \in C$, $x \leq y < \top$ implies $x = y$. The set of dual-atoms is denoted by $\mathrm{dAtoms}(C)$. Note that $\mathrm{dAtoms}(C) \subset \mathrm{MI}(C)$. The domain C is *dual-atomistic* if $C = \mathrm{Moore}\big(\mathrm{dAtoms}(C)\big)$. Thus, if C is dual-atomistic, $\mathrm{MI}(C) = \{\top\} \cup \mathrm{dAtoms}(C)$. The following result holds [FR96, Theorem 4.1].

Theorem 5.1 *If C is meet-generated by $\mathrm{MI}(C)$ then $\mathrm{uco}(C)$ is pseudo-complemented and for any $\rho \in \mathrm{uco}(C)$*

$$id_C \sim \rho = \mathrm{Moore}\big(\mathrm{MI}(C) \setminus \rho(C)\big).$$

Another interesting result is the following [FR96, Corollary 4.5].

Theorem 5.2 *If C is dual-atomistic then $\mathrm{uco}(C)$ is pseudo-complemented and for any*

$\rho \in \mathrm{uco}(C)$

$$id_C \sim \rho = \mathrm{Moore}\bigl(\mathrm{dAtoms}(C) \setminus \rho(C)\bigr).$$

5.3 A Handful of Sharing Domains

In this section we consider a particular subset of the lattice of upper closure operators defined on the set-sharing domain. The elements of this subset will be partially ordered with respect to precision and it will be shown that a few of these abstractions indeed correspond to well-known domains for static analysis.

As discussed in Section 2.3, the knowledge of the set VI of variable of interest has a key role in the formal definition of the concrete meaning of an abstract element. Thus, in Chapters 3 and 4 we have adopted a domain representation that *explicitly* includes this set. However, when defining the abstract operators and stating their properties, we have also followed an incremental approach, taking care to provide results for both the implicit and the explicit representations. The ratio was that we wanted to make clear that, from a technical point of view, no problem can arise when extending the results proven for the implicit version of the abstract operators (e.g., amgu) so that they apply to the explicit one (e.g., Amgu). Typically, the latter are obtained as corollaries of the former. For these reasons, from now on we will only provide the definitions and results for the implicit case. We would like to emphasize that this is done for ease of presentation only: a generalization to the explicit case can be easily obtained by mimicking the structure of the proofs presented in Chapters 3 and 4. Thus, all the abstract domains defined are restricted to a fixed set of variables of interest VI of finite cardinality n, which is not included explicitly in the representation of the domain elements; also, when considering abstract semantics operators having some arguments in $RSubst$, such as amgu, the considered substitutions are assumed to have their variables only in VI.

The sets SG and SH, as defined in Chapter 3, are independent of any particular set of variables of interest, although the actual set-sharing lattice SS as well as the domains PS and PSD defined in Chapters 3 and 4 include VI explicitly. To avoid cluttering up the presentation with a handful of new symbols, we will overload names such as SG, SH, PS and PSD so that the variables that they contain are implicitly restricted to VI. To start with, we provide an overloaded definition for the set-sharing lattice.

Definition 5.3 *Let* $SG \stackrel{\mathrm{def}}{=} \wp(VI) \setminus \{\varnothing\}$ *be the set of* sharing groups. *The set-sharing lattice is defined by* $SH \stackrel{\mathrm{def}}{=} \wp(SG)$, *ordered by subset inclusion.*

Note that we have not considered the distinguished element $\bot$ for representing the semantics of those programs having no successful computations, so that the partial order on SH is simply given by subset inclusion. Since in this chapter we adopt the upper closure operator approach to abstract interpretation, all the domains we define are ordered by subset inclusion. As usual, this ordering provides a formalization of precision where the less precise domain elements are those occurring higher in the partial order (more precise elements contain less sharing groups). It is clear that one can define the overloading $SS \stackrel{\text{def}}{=} \{\bot\} \cup SH$, with the usual extension of the lattice ordering (since SG is the maximum element of SH, the introduction of the distinguished element $\top$ is not necessary).

5.3.1 The Lattice Structure of SH

Since SH is a power set, SH is dual-atomistic and

$$\mathrm{dAtoms}(SH) = \bigl\{\, SG \setminus \{S\} \bigm| S \in SG \,\bigr\}.$$

As in Section 4.3, all the examples in this chapter will represent sharing sets with the simplified notation that omits the inner braces.

Example 5.4 *Suppose $VI = \{x, y, z\}$. Then the seven dual-atoms of SH are:*

$$\begin{aligned}
&\left.\begin{aligned}
s_1 &= \{\; y, z, xy, xz, yz, xyz\},\\
s_2 &= \{x, \; z, xy, xz, yz, xyz\},\\
s_3 &= \{x, y, \; xy, xz, yz, xyz\},
\end{aligned}\right\} &&\textit{these lack a singleton;}\\
&\left.\begin{aligned}
s_4 &= \{x, y, z, \; xz, yz, xyz\},\\
s_5 &= \{x, y, z, xy, \; yz, xyz\},\\
s_6 &= \{x, y, z, xy, xz, \; xyz\},
\end{aligned}\right\} &&\textit{these lack a pair;}\\
&\;\; s_7 = \{x, y, z, xy, xz, yz \}, &&\textit{this lacks } VI.
\end{aligned}$$

The meet-irreducible elements of SH are $s_1, \ldots, s_7$, and the top element SG.

5.3.2 The Tuple-Sharing Domains

To provide a general characterization of domains such as the groundness and pair-sharing domains contained in SH, we first identify the sets of elements that have the same cardinality.

Definition 5.5 (The tuples of cardinality k.) *The functions* $\mathrm{tuples}_k \colon SG \to SH$ *and* $\mathrm{tuples}_k \colon SH \to SH$ *are defined, for each $k \in \mathbb{N}$ such that $1 \leq k \leq n$, as*

$$\begin{aligned}
\mathrm{tuples}_k(S) &\stackrel{\text{def}}{=} \bigl\{\, T \in \wp(S) \bigm| \# T = k \,\bigr\},\\
\mathrm{tuples}_k(sh) &\stackrel{\text{def}}{=} \bigcup \bigl\{\, \mathrm{tuples}_k(S') \bigm| S' \in sh \,\bigr\}.
\end{aligned}$$

In particular, if $S \in SG$ and $sh \in SH$, we have (again by overloading)

$$\begin{aligned}
\mathrm{pairs}(S) &= \mathrm{tuples}_2(S),\\
\mathrm{pairs}(sh) &\stackrel{\text{def}}{=} \mathrm{tuples}_2(sh).
\end{aligned}$$

The usual domains that represent groundness and pair-sharing information will be shown to be special cases of the following more general domain.

Definition 5.6 (The *tuple-sharing* domains TS_k.) *The function $\rho_{TS_k} \colon SH \to SH$ is defined, for each $k \in \mathbb{N}$ such that $1 \leq k \leq n$, as*

$$\rho_{TS_k}(sh) \stackrel{\text{def}}{=} \bigl\{\, S \in SG \bigm| \operatorname{tuples}_k(S) \subseteq \operatorname{tuples}_k(sh) \,\bigr\}.$$

Since $\rho_{TS_k} \in \operatorname{uco}(SH)$, it induces the lattice $TS_k \stackrel{\text{def}}{=} \rho_{TS_k}(SH)$.

Note that $\rho_{TS_k}\bigl(\operatorname{tuples}_k(sh)\bigr) = \rho_{TS_k}(sh)$ and that there is a one to one correspondence between TS_k and $\wp\bigl(\operatorname{tuples}_k(VI)\bigr)$. The isomorphism is given, for each $1 \leq k \leq n$, by the pair of functions $\operatorname{tuples}_k \colon TS_k \to \wp\bigl(\operatorname{tuples}_k(VI)\bigr)$ and $\rho_{TS_k} \colon \wp\bigl(\operatorname{tuples}_k(VI)\bigr) \to TS_k$. Thus the domain TS_k is the smallest domain that can represent properties characterized by sets of variables of cardinality k. We now consider the tuple-sharing domains for $k = 1$, 2, and n.

Definition 5.7 (The *groundness* domain Con.) *The operator $\rho_{Con} \colon SH \to SH$ and the corresponding domain Con are defined as*

$$\begin{aligned}
\rho_{Con} &\stackrel{\text{def}}{=} \rho_{TS_1}, \\
Con &\stackrel{\text{def}}{=} TS_1(SH) = \rho_{Con}(SH).
\end{aligned}$$

This domain, which represents groundness information, is isomorphic to a domain of conjunctions of Boolean variables. The isomorphism tuples_1 maps each element of Con to the set of variables that are possibly non-ground. From the domain $\operatorname{tuples}_1(Con)$, by set complementation, we obtain the classical domain G [JS87] for representing the set of variables that are definitely ground (so that we have $TS_1 \stackrel{\text{def}}{=} Con \equiv \mathsf{G}$).

We now define the upper closure operator corresponding to the pair-sharing domain of Definition 4.1. The overloading of the domain name PS is intentional: as already explained, the two definitions differ in that we now deal implicitly with the fixed set VI of variables of interest.

Definition 5.8 (The *pair-sharing* domain PS.) *The operator $\rho_{PS} \colon SH \to SH$ and the corresponding domain PS are defined as*

$$\begin{aligned}
\rho_{PS} &\stackrel{\text{def}}{=} \rho_{TS_2}, \\
PS &\stackrel{\text{def}}{=} TS_2(SH) = \rho_{PS}(SH).
\end{aligned}$$

The isomorphism tuples_2 maps each element of PS to the set of pairs of variables that may be bound to terms that share a common variable. The domain for representing variable independence can be obtained by set complementation.

Finally, in the case when $k = n$ we have a domain consisting of just two elements:

$$TS_n = \bigl\{SG, SG \setminus \{VI\}\bigr\}.$$

Note that the bottom of TS_n differs from the top element SG only in that it lacks the sharing group VI. There is no intuitive reading for the information encoded by this element: it describes all but those substitutions $\sigma \in RSubst$ such that

$$\bigcap \Bigl\{ \operatorname{vars}\bigl(\operatorname{rt}(x, \sigma)\bigr) \Bigm| x \in VI \Bigr\} \neq \varnothing$$

Just as for SH, the domain TS_k (where $1 \leq k \leq n$) is dual-atomistic and:

$$\operatorname{dAtoms}(TS_k) = \Bigl\{ \bigl(SG \setminus \{ U \in SG \mid T \subseteq U \}\bigr) \Bigm| T \in \operatorname{tuples}_k(VI) \Bigr\}.$$

Thus we have

$$\begin{aligned}
\operatorname{dAtoms}(Con) &= \Bigl\{ \bigl(SG \setminus \{ U \in SG \mid x \in U \}\bigr) \Bigm| x \in VI \Bigr\}, \\
\operatorname{dAtoms}(PS) &= \Bigl\{ \bigl(SG \setminus \{ U \in SG \mid x, y \in U \}\bigr) \Bigm| x, y \in VI, x \neq y \Bigr\}.
\end{aligned}$$

Example 5.9 *Consider Example 5.4. Then the dual-atoms of Con are*

$$\begin{aligned}
r_1 &= s_1 \cap s_4 \cap s_5 \cap s_7 = \{ \quad y, z, \qquad\quad yz\}, \\
r_2 &= s_2 \cap s_4 \cap s_6 \cap s_7 = \{x, \quad z, \quad\; xz \quad\; \}, \\
r_3 &= s_3 \cap s_5 \cap s_6 \cap s_7 = \{x, y, \quad xy \qquad\quad \};
\end{aligned}$$

the dual-atoms of PS are

$$\begin{aligned}
m_1 &= s_4 \cap s_7 = \{x, y, z, \qquad xz, yz\}, \\
m_2 &= s_5 \cap s_7 = \{x, y, z, xy, \qquad yz\}, \\
m_3 &= s_6 \cap s_7 = \{x, y, z, xy, xz \qquad\}.
\end{aligned}$$

It can be seen from the dual-atoms that, for each $j = 1, \ldots, n$, where $j \neq k$, the precision of the information encoded by domains TS_j and TS_k is not comparable. Also, we note that, if $j < k$, then $\rho_{TS_j}(TS_k) = \{SG\}$ and $\rho_{TS_k}(TS_j) = TS_j$.

5.3.3 The Tuple-Sharing Dependency Domains

We now need to define domains that capture the propagation of groundness and pair-sharing; in particular, the dependency of these properties on the further instantiation of the variables. In the same way as with TS_k for Con and PS, we first define a general subdomain TSD_k of SH. This must be safe and precise with respect to the tuple-sharing property represented by TS_k when performing the usual abstract operations. This was the motivation behind the introduction of the pair-sharing dependency domain PSD in Chapter 4. We now generalize this for tuple-sharing.

Definition 5.10 (The *tuple-sharing dependency* domain TSD_k.) *The function*

$\rho_{TSD_k} \colon SH \to SH$ *is defined, for each* $k \in \mathbb{N}$ *such that* $1 \leq k \leq n$, *as*

$$\rho_{TSD_k}(sh) \stackrel{\mathrm{def}}{=} \Bigl\{\, S \in SG \Bigm| \forall T \subseteq S : \# T < k \implies S = \bigcup \{\, U \in sh \mid T \subseteq U \subseteq S \,\} \,\Bigr\}.$$

Since $\rho_{TSD_k} \in \mathrm{uco}(SH)$, *it induces the lattice* $TSD_k \stackrel{\mathrm{def}}{=} \rho_{TSD_k}(SH)$.

It follows from the definitions that the domains TSD_k form a strict chain.

Proposition 5.11 *For* $j, k \in \mathbb{N}$ *with* $1 \leq j < k \leq n$, $TSD_j \subset TSD_k$.

Moreover, TSD_k is not less precise than TS_k.

Proposition 5.12 *For* $k \in \mathbb{N}$ *with* $1 \leq k \leq n$, *we have* $TS_k \subseteq TSD_k$. *Furthermore, if* $n > 1$ *then* $TS_k \subset TSD_k$.

As an immediate consequence of Propositions 5.11 and 5.12 we have that that TSD_k is not less precise than $TS_1 \sqcap \cdots \sqcap TS_k$.

Corollary 5.13 *For* $j, k \in \mathbb{N}$ *with* $1 \leq j \leq k \leq n$, *we have* $TS_j \subseteq TSD_k$.

In Chapter 4 we have seen how, when working on elements of the domain PSD, the star-union operator can be safely replaced by binary union. In order to generalize this result so that it applies to an arbitrary tuple-sharing dependency domain, we now introduce the j-self-union operator.

Definition 5.14 (j-self-union.) *The* j-self-union *function* $(\cdot)^j \colon SH \to SH$ *is defined, for each* $j \geq 1$ *and* $sh \in SH$, *as*

$$sh^j \stackrel{\mathrm{def}}{=} \Bigl\{\, S \in SG \Bigm| \exists sh' \subseteq sh \,.\, \bigl(\# sh' \leq j, S = \bigcup sh'\bigr) \,\Bigr\}.$$

Note that, letting $j = 1$, 2, and n, we have $sh^1 = sh$, $sh^2 = \mathrm{bin}(sh, sh)$, and, as $\# VI = n$, $sh^n = sh^\star$.

For the TSD_k domain, the star-union operator can be replaced by the k-self-union operator.

Proposition 5.15 *For* $1 \leq k \leq n$, *we have* $\rho_{TSD_k}\bigl(sh^k\bigr) = sh^\star$.

We now instantiate the tuple-sharing dependency domains for $k = 1$, 2, and n.

Definition 5.16 (The *ground dependency* domain Def.) *The domain* Def *is induced by the upper closure operator* $\rho_{Def} \colon SH \to SH$. *They are defined as*

$$\begin{aligned} \rho_{Def} &\stackrel{\mathrm{def}}{=} \rho_{TSD_1}, \\ Def &\stackrel{\mathrm{def}}{=} TSD_1 = \rho_{Def}(SH). \end{aligned}$$

By Proposition 5.15, we have, for all $sh \in SH$, $\rho_{TSD_1}(sh) = sh^\star$ so that TSD_1 is a representation of the domain Def used for capturing definite groundness [CFW94, CFW98]. It also provides evidence for the fact that the computation of the star-union is not needed for the elements in Def.

We now define the upper closure operator for the pair-sharing dependency domain PSD, corresponding to Definition 4.3. Note that the equivalence of the two definitions has already been stated in Theorem 4.5. Again, the overloading of the domain name PSD is intentional.

Definition 5.17 (The *pair-sharing dependency* domain PSD.) *The upper closure operator $\rho_{PSD} \colon SH \to SH$ and the corresponding domain PSD are defined as*

$$\begin{aligned} \rho_{PSD} &\stackrel{\text{def}}{=} \rho_{TSD_2}, \\ PSD &\stackrel{\text{def}}{=} TSD_2 = \rho_{PSD}(SH). \end{aligned}$$

By Proposition 5.15 we have, for all $sh \in SH$, that $\rho_{PSD}(sh^2) = sh^\star$, therefore confirming the result stated in Theorem 4.11: for elements in PSD the star-union operator $sh^\star$ can be replaced by the 2-self-union $sh^2 = \mathrm{bin}(sh, sh)$ without any loss of precision. Furthermore, Corollary 5.13 confirms the observation made in Chapter 4 that PSD also captures groundness.

Finally, letting $k = n$, we observe that $TSD_n = SH$. Figure 5.1 summarizes the relations between the tuple-sharing and the tuple-sharing dependency domains (domains occurring lower are the most precise ones).

Going on strengthening and generalizing the results of Chapter 4, we now show that, for each $k \in \{1, \dots, n\}$, TSD_k is the quotient of SH with respect to the reduced product $TS_1 \sqcap \cdots \sqcap TS_k$. To this end, we need to provide the "implicit VI" version of the Aunify operator.

Definition 5.18 (aunify.) *The function* aunify: $SH \times RSubst \to SH$ *generalizes* amgu *to any substitution in $RSubst$. If $sh \in SH$ and $\sigma \in RSubst$, then*

$$\mathrm{aunify}(sh, \sigma) \stackrel{\text{def}}{=} \begin{cases} sh, & \text{if } \sigma = \varnothing; \\ \mathrm{aunify}\bigl(\mathrm{amgu}(sh, x \mapsto r), \sigma \setminus \{x \mapsto r\}\bigr), & \text{if } (x \mapsto r) \in \sigma. \end{cases}$$

We are now ready to state the completeness and minimality results for the tuple-sharing dependency domains. Both results are proved at the end of this section.

Theorem 5.19 *Let $sh_1, sh_2 \in SH$ be such that $\rho_{TSD_k}(sh_1) = \rho_{TSD_k}(sh_2)$, where $1 \leq k \leq n$. Then, for each $\sigma \in RSubst$, $sh' \in SH$ and $V \in \wp(VI)$,*

$$\begin{aligned} \rho_{TSD_k}\bigl(\mathrm{aunify}(sh_1, \sigma)\bigr) &= \rho_{TSD_k}\bigl(\mathrm{aunify}(sh_2, \sigma)\bigr), \\ \rho_{TSD_k}(sh' \cup sh_1) &= \rho_{TSD_k}(sh' \cup sh_2), \\ \rho_{TSD_k}\bigl(\mathrm{aexists}(sh_1, V)\bigr) &= \rho_{TSD_k}\bigl(\mathrm{aexists}(sh_2, V)\bigr). \end{aligned}$$

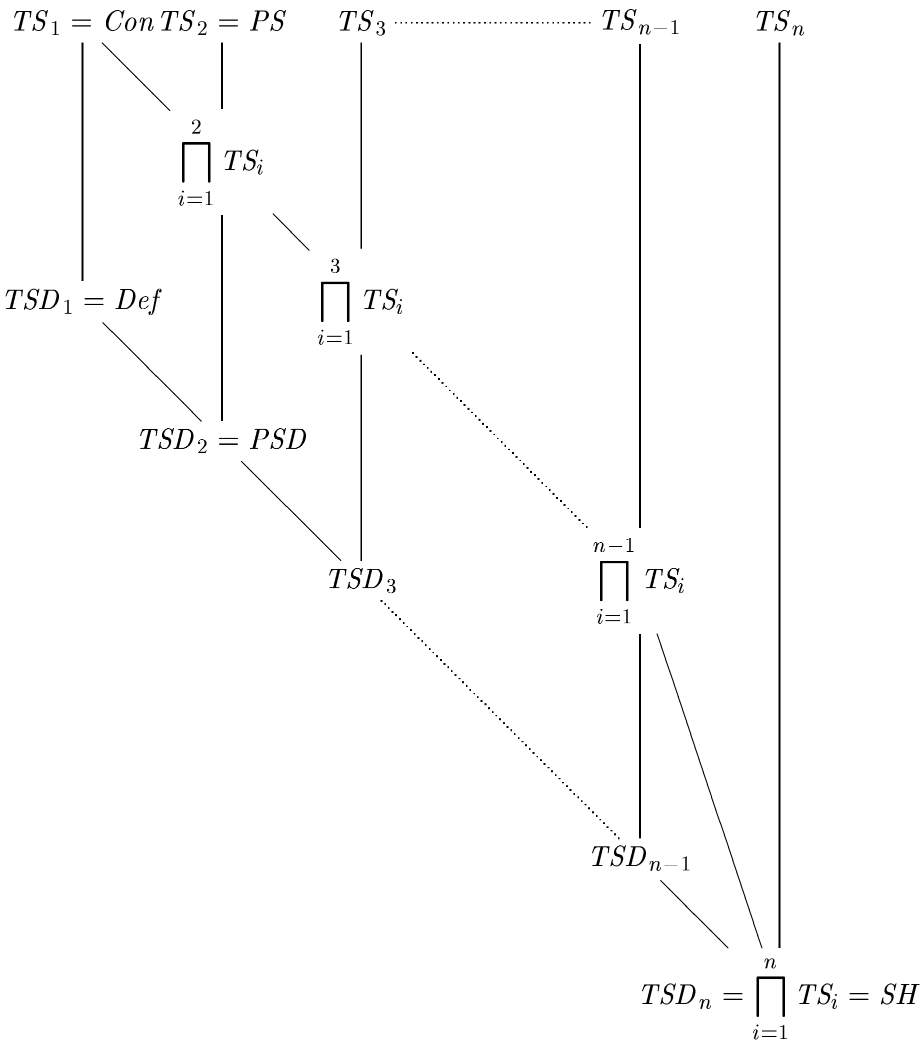

Figure 5.1: The set-sharing domain SH and some of its abstractions.

Theorem 5.20 *Let $sh_1, sh_2 \in SH$ be such that $\rho_{TSD_k}(sh_1) \neq \rho_{TSD_k}(sh_2)$, where $1 \leq k \leq n$. Then there exist $\sigma \in RSubst$ and $j \in \{1, \ldots, k\}$ such that*

$$\rho_{TS_j}\bigl(\mathrm{aunify}(sh_1, \sigma)\bigr) \neq \rho_{TS_j}\bigl(\mathrm{aunify}(sh_2, \sigma)\bigr).$$

5.3.4 Proofs of Theorems 5.19 and 5.20

In what follows we use the fact that ρ_{TSD_k} is an upper closure operator so that, for each $sh_1, sh_2 \in SH$,

$$sh_1 \subseteq \rho_{TSD_k}(sh_2) \iff \rho_{TSD_k}(sh_1) \subseteq \rho_{TSD_k}(sh_2). \tag{5.1}$$

Lemma 5.21 *For each $sh \in SH$ and $V \in \wp(VI)$, $\overline{\mathrm{rel}}\bigl(V, \rho_{TSD_k}(sh)\bigr) = \rho_{TSD_k}\bigl(\overline{\mathrm{rel}}(V, sh)\bigr)$.*

Proof. By Definition 5.10,

$$\begin{aligned}
&S \in \rho_{TSD_k}\bigl(\overline{\mathrm{rel}}(V, sh)\bigr)\\
&\quad\iff \forall T \subseteq S : \Bigl(\#T < k \implies S = \bigcup\bigl\{\, U \in \overline{\mathrm{rel}}(V, sh) \bigm| T \subseteq U \subseteq S \,\bigr\}\Bigr)\\
&\quad\iff \forall T \subseteq S : \Bigl(\#T < k \implies S = \bigcup\{\, U \in sh \mid T \subseteq U \subseteq S \,\}\Bigr)\\
&\qquad\qquad \land S \cap V = \varnothing\\
&\quad\iff S \in \rho_{TSD_k}(sh) \land S \cap V = \varnothing\\
&\quad\iff S \in \overline{\mathrm{rel}}\bigl(V, \rho_{TSD_k}(sh)\bigr).
\end{aligned}$$

□

Lemma 5.22 *For each $sh_1, sh_2 \in SH$, each $V \in \wp(VI)$ and each $k \in \mathbb{N}$ with $1 < k \leq n$,*

$$\rho_{TSD_k}(sh_1) \subseteq \rho_{TSD_k}(sh_2) \implies \mathrm{rel}(V, sh_1)^\star \subseteq \mathrm{rel}(V, sh_2)^\star.$$

Proof. We prove that

$$sh_1 \subseteq \rho_{TSD_k}(sh_2) \implies \mathrm{rel}(V, sh_1) \subseteq \mathrm{rel}(V, sh_2)^\star.$$

The result then follows from Eq. (5.1).

Suppose that $S \in \mathrm{rel}(V, sh_1)$. Then, $S \in sh_1$ and $S \cap V \neq \varnothing$. By the hypothesis, $S \in \rho_{TSD_k}(sh_2)$. Let $x \in S \cap V$. Then, by Definition 5.10, we have

$$\begin{aligned}
S &= \bigcup\bigl\{\, U \in sh_2 \bigm| \{x\} \subseteq U \subseteq S \,\bigr\}\\
&= \bigcup\bigl\{\, U \in \mathrm{rel}(V, sh_2) \bigm| \{x\} \subseteq U \subseteq S \,\bigr\}.
\end{aligned}$$

Thus $S \in \mathrm{rel}(V, sh_2)^\star$. □

Lemma 5.23 *Let $sh_1, sh_2 \in SH$ be such that $\rho_{TSD_k}(sh_1) = \rho_{TSD_k}(sh_2)$. Then, for each $(x \mapsto t) \in Bind$ and each $k \in \mathbb{N}$ with $1 \leq k \leq n$,*

$$\rho_{TSD_k}\bigl(\mathrm{amgu}(sh_1, x \mapsto t)\bigr) = \rho_{TSD_k}\bigl(\mathrm{amgu}(sh_2, x \mapsto t)\bigr).$$

Proof. We will show that

$$sh_1 \subseteq \rho_{TSD_k}(sh_2) \implies \mathrm{amgu}(sh_1, x \mapsto t) \subseteq \rho_{TSD_k}\bigl(\mathrm{amgu}(sh_2, x \mapsto t)\bigr).$$

The result then follows from Eq. (5.1).

Let $v_x \stackrel{\mathrm{def}}{=} \{x\}$, $v_t \stackrel{\mathrm{def}}{=} \mathrm{vars}(t)$, and $v_{xt} \stackrel{\mathrm{def}}{=} v_x \cup v_t$. Suppose $S \in \mathrm{amgu}(sh_1, x \mapsto t)$. Then, by definition of amgu,

$$S \in \overline{\mathrm{rel}}(v_{xt}, sh_1) \cup \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_1)^\star, \mathrm{rel}(v_t, sh_1)^\star\bigr).$$

There are two cases:

1. $S \in \overline{\mathrm{rel}}(v_{xt}, sh_1)$. In this case $S \in sh_1$, so that $S \in \rho_{TSD_k}(sh_2)$. It also holds $S \cap V = \varnothing$. Hence, we have $S \in \overline{\mathrm{rel}}\bigl(v_{xt}, \rho_{TSD_k}(sh_2)\bigr)$. Thus, by Lemma 5.21,

$$S \in \rho_{TSD_k}\bigl(\overline{\mathrm{rel}}(v_{xt}, sh_2)\bigr).$$

2. $S \in \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_1)^\star, \mathrm{rel}(v_t, sh_1)^\star\bigr)$. In this case $S = T \cup R$, where $T \in \mathrm{rel}(v_x, sh_1)^\star$ and $R \in \mathrm{rel}(v_t, sh_1)^\star$.

The proof here splits into two branches, 2a and 2b, depending on whether we have $k > 1$ or $k = 1$.

2a. Assume $k > 1$. Then, by Lemma 5.22, $T \in \mathrm{rel}(v_x, sh_2)^\star$ and $R \in \mathrm{rel}(v_t, sh_2)^\star$. Hence,

$$S \in \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_2)^\star, \mathrm{rel}(v_t, sh_2)^\star\bigr).$$

Combining case 1 and case 2a we obtain

$$S \in \rho_{TSD_k}\bigl(\overline{\mathrm{rel}}(v_{xt}, sh_2)\bigr) \cup \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_2)^\star, \mathrm{rel}(v_t, sh_2)^\star\bigr).$$

As ρ_{TSD_k} is extensive and monotonic, we obtain

$$S \in \rho_{TSD_k}\Bigl(\overline{\mathrm{rel}}(v_{xt}, sh_2) \cup \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_2)^\star, \mathrm{rel}(v_t, sh_2)^\star\bigr)\Bigr),$$

and hence, when $k > 1$, $S \in \rho_{TSD_k}\bigl(\mathrm{amgu}(sh_2, x \mapsto t)\bigr)$.

2b. Secondly suppose that $k = 1$. In this case, by Proposition 5.15:

$$\rho_{TSD_1}(sh_2) = sh_2^\star,$$
$$\rho_{TSD_1}\bigl(\mathrm{amgu}(sh_2, x \mapsto t)\bigr) = \mathrm{amgu}(sh_2, x \mapsto t)^\star.$$

Thus, by the hypothesis,

$$\begin{aligned} S &\in \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_2^\star)^\star, \mathrm{rel}(v_t, sh_2^\star)^\star\bigr) \\ &= \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_2^\star), \mathrm{rel}(v_t, sh_2^\star)\bigr). \end{aligned}$$

Therefore we can write $S = T_{\text{-}} \cup T_x \cup R_{\text{-}} \cup R_t$, where

$$
\begin{aligned}
T_{\text{-}} \cup T_x &\in \mathrm{rel}(v_x, sh_2^\star),\\
R_{\text{-}} \cup R_t &\in \mathrm{rel}(v_t, sh_2^\star),\\
T_{\text{-}}, R_{\text{-}} &\in \overline{\mathrm{rel}}(v_{xt}, sh_2)^\star,\\
T_x &\in \mathrm{rel}(v_x, sh_2)^\star \setminus \varnothing,\\
R_t &\in \mathrm{rel}(v_t, sh_2)^\star \setminus \varnothing.
\end{aligned}
$$

Thus $S \in \Bigl(\overline{\mathrm{rel}}(v_{xt}, sh_2) \cup \mathrm{bin}\bigl(\mathrm{rel}(v_x, sh_2)^\star, \mathrm{rel}(v_t, sh_2)^\star\bigr)\Bigr)^\star = \mathrm{amgu}(sh_2, x \mapsto t)^\star$.

Combining case 1 and case 2b for $k = 1$, the result follows immediately by the monotonicity and extensivity of $(\cdot)^\star$. □

Lemma 5.24 *Let $sh_1, sh_2 \in SH$ be such that $\rho_{TSD_k}(sh_1) = \rho_{TSD_k}(sh_2)$. Then, for each $\sigma \in RSubst$ and each $k \in \mathbb{N}$ with $1 \leq k \leq n$,*

$$\rho_{TSD_k}\bigl(\mathrm{aunify}(sh_1, \sigma)\bigr) = \rho_{TSD_k}\bigl(\mathrm{aunify}(sh_2, \sigma)\bigr).$$

Proof. The proof is by induction on the cardinality of σ. The base case, when $\# \sigma = 0$ and thus $\sigma = \varnothing$, is obvious from the definition of aunify. For the inductive case, when $\# \sigma = m > 0$, assume the result holds for all substitutions $\mu \in RSubst$ such that $\# \mu < m$. Let $(x \mapsto t) \in \sigma$ and let $\sigma' = \sigma \setminus \{x \mapsto t\}$. Note that $\# \sigma' = m - 1 < m$.

For $i \in \{1, 2\}$, by definition of aunify,

$$\mathrm{aunify}(sh_i, \sigma) = \mathrm{aunify}\bigl(\mathrm{amgu}(sh_i, x \mapsto t), \sigma'\bigr).$$

By Lemma 5.23, it also holds

$$\rho_{TSD_k}\bigl(\mathrm{amgu}(sh_1, x \mapsto t)\bigr) = \rho_{TSD_k}\bigl(\mathrm{amgu}(sh_2, x \mapsto t)\bigr). \tag{5.2}$$

However, this allows us to complete the proof, since Eq. (5.2) is exactly the condition needed in order to apply the inductive hypothesis to σ'. □

Lemma 5.25 *For each $sh_1, sh_2 \in SH$, $\rho_{TSD_k}(sh_1 \cup sh_2) = \rho_{TSD_k}\bigl(\rho_{TSD_k}(sh_1) \cup \rho_{TSD_k}(sh_2)\bigr)$.*

Proof. This is a classical property of upper closure operators. The proof is essentially the same as the one given for Lemma 4.19. □

Lemma 5.26 *Let $sh_1, sh_2 \in SH$ be such that $\rho_{TSD_k}(sh_1) = \rho_{TSD_k}(sh_2)$. Then, for each $V \subseteq VI$,*

$$\rho_{TSD_k}\bigl(\mathrm{aexists}(sh_1, V)\bigr) = \rho_{TSD_k}\bigl(\mathrm{aexists}(sh_2, V)\bigr).$$

Proof. We show that

$$sh_1 \subseteq \rho_{TSD_k}(sh_2) \implies \mathrm{aexists}(sh_1, V) \subseteq \rho_{TSD_k}\bigl(\mathrm{aexists}(sh_2, V)\bigr).$$

The result then follows from Eq. (5.1).

Suppose $sh_1 \subseteq \rho_{TSD_k}(sh_2)$ and $S \in \mathrm{aexists}(sh_1, V)$. Then, as aexists is monotonic, we have $S \in \mathrm{aexists}\bigl(\rho_{TSD_k}(sh_2), V\bigr)$. We distinguish two cases.

1. $\exists x \in V \,.\, S = \{x\}$. Then, by Definition 3.61, $S \in \mathrm{aexists}(sh_2, V)$ and hence, by Definition 5.10, $S \in \rho_{TSD_k}\bigl(\mathrm{aexists}(sh_2, V)\bigr)$.
2. Otherwise, by definition of aexists and Definition 5.10, there exists a sharing group $S' \in \rho_{TSD_k}(sh_2)$ such that $S = S' \cap V \neq \varnothing$ and

$$\forall T \subseteq S' : \Bigl(\# T < k \implies S = \bigcup \{\, U \in sh_2 \mid T \subseteq U \subseteq S' \,\} \cap V\Bigr).$$

Hence

$$\forall T \subseteq S : \Bigl(\# T < k \implies S = \bigcup \bigl\{\, U \in \mathrm{aexists}(sh_2, V) \bigm| T \subseteq U \subseteq S \,\bigr\}\Bigr),$$

and thus $S \in \rho_{TSD_k}\bigl(\mathrm{aexists}(sh_2, V)\bigr)$.

$\square$

Proof of Theorem 5.19 on page 115. The three stated congruence properties follow from Lemmas 5.24, 5.25 and 5.26, respectively. $\square$

Proof of Theorem 5.20 on page 117. Let $S \in \rho_{TSD_k}(sh_1) \setminus \rho_{TSD_k}(sh_2)$. (If such an S does not exist we simply swap sh_1 and sh_2.)

Let $t \in \mathit{GTerms} \cap \mathit{HTerms}$ be a ground and finite term and let

$$\sigma \stackrel{\mathrm{def}}{=} \{\, x \mapsto t \mid x \in \mathit{VI} \setminus S \,\}.$$

Then, by Lemma 4.23, for $i = 1$, 2, we define $sh_i^S \stackrel{\mathrm{def}}{=} \mathrm{aunify}(sh_i, \sigma)$, so that

$$\begin{aligned} sh_1^S &= \{\, T \subseteq S \mid T \in sh_1 \,\}, \\ sh_2^S &= \{\, T \subset S \mid T \in sh_2 \,\}. \end{aligned}$$

Now, if $\# S = j$ and $j \leq k$, then we have $S \in sh_1 \setminus sh_2$. Therefore $S \in sh_1^S \setminus sh_2^S$ and we can easily observe that $S \in \rho_{TS_j}(sh_1^S)$ but $S \notin \rho_{TS_j}(sh_2^S)$.

On the other hand, if $\# S = j$ and $j > k$, then by Definition 5.10 there exists T with $\# T < k$ such that

$$S = \bigcup \{\, U \in sh_1^S \mid T \subseteq U \,\}$$

but

$$S \supset \bigcup \{\, U \in sh_2^S \mid T \subseteq U \,\} \stackrel{\mathrm{def}}{=} S'.$$

Let $x \in S \setminus S'$. We have $h \stackrel{\text{def}}{=} \#(T \cup \{x\}) \leq k$ and thus we have $T \cup \{x\} \in \rho_{TS_h}(sh_1^S)$ but $T \cup \{x\} \notin \rho_{TS_h}(sh_2^S)$. $\square$

5.4 The Meet-Irreducible Elements

In Section 5.5, we will use the method of Filé and Ranzato [FR96] to decompose the dependency domains TSD_k. In preparation for this, in this section, we identify the meet-irreducible elements for the domains and state some general results.

We have already observed that TS_k and $TSD_n = SH$ are dual-atomistic. However, TSD_k, for $k < n$, is not dual-atomistic and we need to identify the meet-irreducible elements. In fact, the set of dual-atoms for TSD_k is

$$\mathrm{dAtoms}(TSD_k) = \left\{ SG \setminus \{S\} \;\middle|\; S \in SG, \# S \leq k \right\}.$$

Note that $\# \mathrm{dAtoms}(TSD_k) = \sum_{j=1}^{k} \binom{n}{j}$. Specializing this for $k = 1$ and $k = 2$, respectively, we have

$$\begin{aligned} \mathrm{dAtoms}(Def) &= \left\{ SG \setminus \{\{x\}\} \;\middle|\; x \in VI \right\}, \\ \mathrm{dAtoms}(PSD) &= \left\{ SG \setminus \{S\} \;\middle|\; S \in \mathrm{pairs}(VI) \right\} \cup \mathrm{dAtoms}(Def), \end{aligned}$$

and we have $\# \mathrm{dAtoms}(Def) = n$ and $\# \mathrm{dAtoms}(PSD) = n(n+1)/2$. We present as an example of this the dual-atoms for Def and PSD when $n = 3$.

Example 5.27 *Consider Example 5.4. Then the 3 dual-atoms for Def are s_1, s_2, s_3 and the 6 dual-atoms for PSD are $s_1, \ldots, s_6$. Note that these are not all the meet-irreducible elements since sets that do not contain the sharing group xyz such as $\{x\}$ and $\varnothing = \rho_{Def}(\varnothing)$ cannot be obtained by the meet (which is set intersection) of a set of dual-atoms. Thus, unlike Con and PS, neither Def nor PSD are dual-atomistic.*

Consider next the set M_k of the meet-irreducible elements of TSD_k that are neither the top element SG nor dual-atoms. M_k has an element for each sharing group $S \in SG$ such that $\# S > k$ and each tuple $T \subset S$ with $\# T = k$. Such an element is obtained from SG by removing all the sharing groups U such that $T \subseteq U \subseteq S$. Formally, for $1 \leq k \leq n$,

$$M_k \stackrel{\text{def}}{=} \left\{ SG \setminus \{ U \in SG \mid T \subseteq U \subseteq S \} \;\middle|\; T, S \in SG, T \subset S, \# T = k \right\}.$$

As there are $\binom{n}{k}$ possible choices for T and $2^{n-k} - 1$ possible choices for S, we have $\# M_k = \binom{n}{k}(2^{n-k} - 1)$ and $\# \mathrm{MI}(TSD_k) = \sum_{j=0}^{k-1} \binom{n}{j} + \binom{n}{k} 2^{n-k}$.

We now show that we have identified all the meet-irreducible elements of TSD_k.

Theorem 5.28 *If $k \in \mathbb{N}$ with $1 \leq k \leq n$, then*

$$\mathrm{MI}(TSD_k) = \{SG\} \cup \mathrm{dAtoms}(TSD_k) \cup M_k.$$

Proof. We prove the two inclusions separately.

1. $\mathrm{MI}(\mathit{TSD}_k) \supseteq \{SG\} \cup \mathrm{dAtoms}(\mathit{TSD}_k) \cup M_k$.

   Let m be in the right-hand side. If $m \in \{SG\} \cup \mathrm{dAtoms}(\mathit{TSD}_k)$ there is nothing to prove. Therefore we assume $m \in M_k$. We need to prove that

   $$\forall sh_1, sh_2 \in \mathit{TSD}_k : m = sh_1 \cap sh_2 \implies (m = sh_1 \vee m = sh_2).$$

   Suppose $m = sh_1 \cap sh_2$. Obviously, we have $m \subseteq sh_1$ and $m \subseteq sh_2$. Moreover, by definition of M_k, there exist $T, S \in SG$ where $\# T = k$ and $T \subset S$ such that

   $$m = SG \setminus \bigl\{ U \in SG \bigm| T \subseteq U \subseteq S \bigr\}.$$

   Since $S \notin m$, we have $S \notin sh_1$ or $S \notin sh_2$. Let us consider the first case (the other is symmetric). Then, applying the definition of TSD_k, there is a $T' \subset S$ with $\# T' < k$ such that

   $$\bigcup \{ U' \in sh_1 \mid T' \subseteq U' \subseteq S \} \neq S.$$

   Since $\# T' < \# T$, there exists x such that $x \in T \setminus T'$. Thus $T' \subset S \setminus \{x\}$ and $S \setminus \{x\} \in m$. Hence, as $m \subseteq sh_1$, we have $S \setminus \{x\} \in sh_1$. Consider an arbitrary $U \in SG$ where $T \subseteq U \subseteq S$. Then $x \in U$. Thus, since $S = \bigl(S \setminus \{x\}\bigr) \cup U$ and $S \notin sh_1$, $U \notin sh_1$. Thus, as this is true for all such U, $sh_1 \subseteq m$.

2. $\mathrm{MI}(\mathit{TSD}_k) \subseteq \{SG\} \cup \mathrm{dAtoms}(\mathit{TSD}_k) \cup M_k$.

   Let $sh \in \mathit{TSD}_k$. We need to show that sh is the meet of elements in the right-hand side. If $sh = SG$ then there is nothing to prove. Suppose $sh \neq SG$. For each $S \in SG$ such that $S \notin sh$, we will show there is an element m_S in the right-hand side such that $S \notin m_S$ and $sh \subseteq m_S$. Then $sh = \bigcap \{ m_S \mid S \notin sh \}$.

   There are two cases.

   2a. $\# S \leq k$; Let $m_S = SG \setminus \{S\}$. Then $m_S \in \mathrm{dAtoms}(\mathit{TSD}_k)$ and $sh \subseteq m_S$.

   2b. $\# S > k$; in this case, applying the definition of TSD_k, there must exist a set $T' \subset S$ with $\# T' < k$ such that

   $$\bigcup \{ U' \in sh \mid T' \subset U' \subseteq S \} \subset S.$$

   However, since $T' \subset S$, we have $S = \bigcup \bigl\{ T' \cup \{x\} \bigm| x \in S \setminus T' \bigr\}$. Thus, for some $x \in S \setminus T'$, if U is such that $T' \cup \{x\} \subseteq U \subseteq S$ then $U \notin sh$. Choose $T \in SG$ so that $T' \cup \{x\} \subseteq T$ and $\# T = k$ and let $m_S = SG \setminus \{ U \in SG \mid T \subseteq U \subseteq S \}$. Then $m_S \in M_k$, $S \notin m_S$, and $sh \subseteq m_S$.

□

We illustrate the above results for the case when $n = 3$.

Example 5.29 *Consider again Example 5.27. First, consider the domain Def. The meet-irreducible elements which are not dual-atoms, besides SG, are the following (see Figure 5.2, where the rightmost elements are the most precise ones):*

$$
\begin{array}{ll}
q_1 = \{\quad y, z, \quad\quad xz, yz, xyz\} \subset s_1, & \\
q_2 = \{\quad y, z, xy, \quad yz, xyz\} \subset s_1, & r_1 = \{\quad y, z, \quad\quad\quad yz\} \subset q_1 \cap q_2, \\
q_3 = \{x, \quad z, \quad\quad xz, yz, xyz\} \subset s_2, & \\
q_4 = \{x, \quad z, xy, xz, \quad xyz\} \subset s_2, & r_2 = \{x, \quad z, \quad\quad xz \quad\} \subset q_3 \cap q_4, \\
q_5 = \{x, y, \quad xy, \quad yz, xyz\} \subset s_3, & \\
q_6 = \{x, y, \quad xy, xz, \quad xyz\} \subset s_3, & r_3 = \{x, y, \quad xy \quad\quad\} \subset q_5 \cap q_6.
\end{array}
$$

Next, consider the domain PSD. The only meet-irreducible elements that are not dual-atoms, beside SG, are the following (see Figure 5.3):

$$
\begin{aligned}
m_1 &= \{x, y, z, \quad xz, yz \quad\} \subset s_4 \\
m_2 &= \{x, y, z, xy, \quad yz \quad\} \subset s_5 \\
m_3 &= \{x, y, z, xy, xz \quad\quad\} \subset s_6.
\end{aligned}
$$

Each of these lack a pair and none contains the sharing group xyz.

Looking at Examples 5.9 and 5.29, it can be seen that all the dual-atoms of the domains *Con* and *PS* are meet-irreducible elements of the domains *Def* and *PSD*, respectively. In fact, the following general result shows that the dual-atoms of the domain TS_k are meet-irreducible elements for the domain TSD_k.

Corollary 5.30 *Let* $k \in \mathbb{N}$ *with* $1 \leq k \leq n$. *Then*

$$\text{dAtoms}(TS_k) = \{\, sh \in \text{MI}(TSD_k) \mid VI \notin sh \,\}.$$

For the decomposition, we need to identify which meet-irreducible elements of TSD_k are in TS_j. Using Corollaries 5.13 and 5.30 we have the following result.

Corollary 5.31 *If* $j, k \in \mathbb{N}$ *with* $1 \leq j < k \leq n$, *then*

$$\text{MI}(TSD_k) \cap TS_j = \{SG\}.$$

By combining Proposition 5.11 with Theorem 5.28 we can identify the meet-irreducible elements of TSD_k that are in TSD_j, where $j < k$.

Corollary 5.32 *If* $j, k \in \mathbb{N}$ *with* $1 \leq j < k \leq n$, *then*

$$\text{MI}(TSD_k) \cap TSD_j = \text{dAtoms}(TSD_j).$$

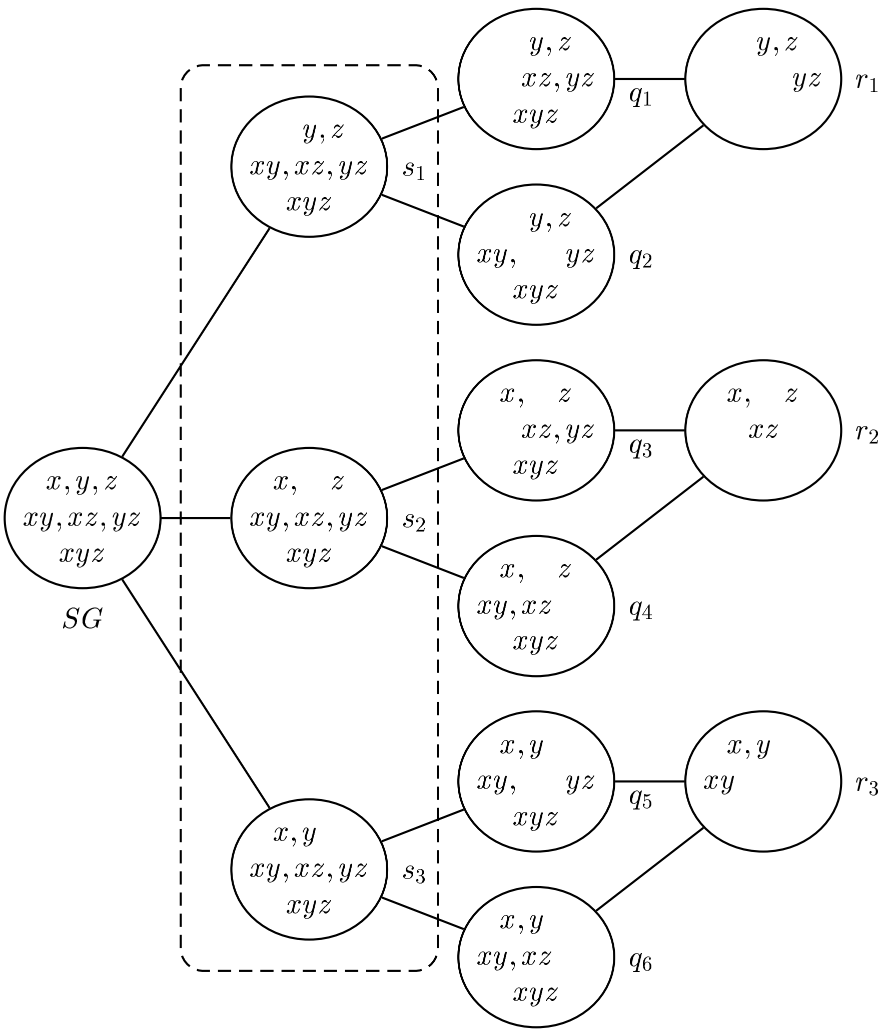

Figure 5.2: The meet-irreducible elements of *Def* for $n = 3$, with dual-atoms emphasized.

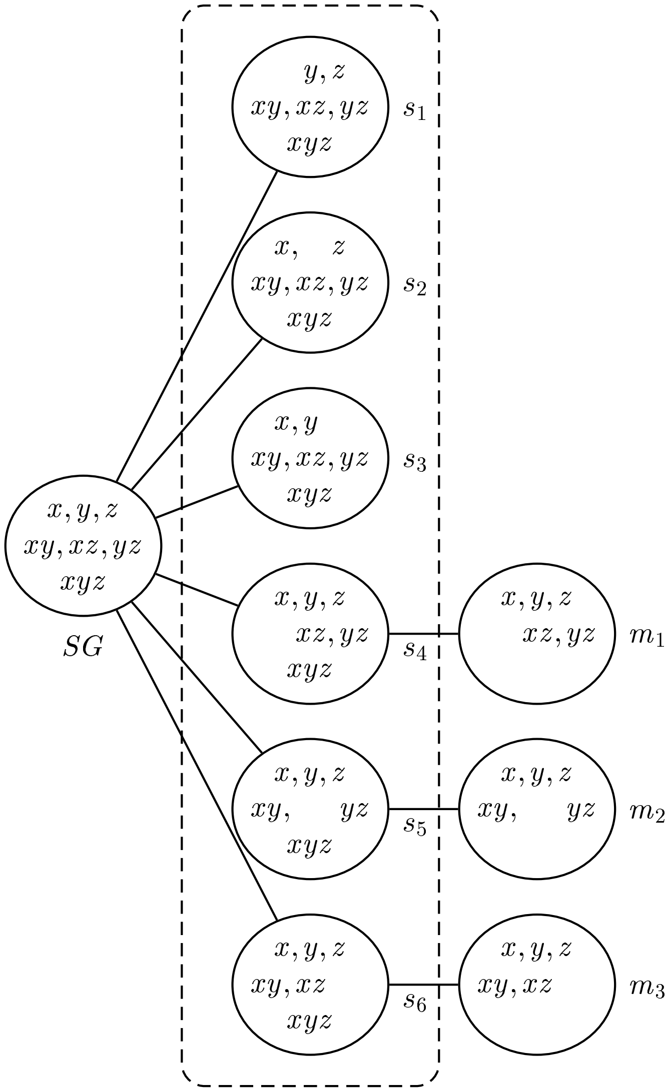

Figure 5.3: The meet-irreducible elements of PSD for $n = 3$, with dual-atoms emphasized.

5.5 The Decomposition of the Domains

5.5.1 Removing the Tuple-Sharing Domains

We first consider the decomposition of TSD_k with respect to TS_j. It follows from Theorem 5.1 and Corollaries 5.13 and 5.31 that, for $1 \leq j < k \leq n$, we have

$$\begin{aligned} TSD_k \sim TS_j &= \mathrm{Moore}\bigl(\mathrm{MI}(TSD_k) \setminus \rho_{TS_j}(TSD_k)\bigr) \\ &= \mathrm{Moore}\bigl(\mathrm{MI}(TSD_k) \setminus TS_j\bigr) \\ &= TSD_k. \end{aligned} \tag{5.3}$$

Since $SH = TSD_n$, we have, using Eq. (5.3) and setting $k = n$, that, if $j < n$,

$$SH \sim TS_j = SH. \tag{5.4}$$

Thus, in general, TS_j is too abstract to be removed from SH by means of complementation. (Note that here it is required $j < n$, because we have $SH \sim TS_n \neq SH$.) In particular, letting $j = 1$, 2 (assuming $n > 2$) in Eq. (5.4), we have

$$SH \sim PS = SH \sim Con = SH, \tag{5.5}$$

showing that Con and PS are too abstract to be removed from SH by means of complementation. Also, by Eq. (5.3), letting $j = 1$ and $k = 2$ it follows that the complement of Con in PSD is PSD.

Now consider decomposing TSD_k using TS_k. It follows from Theorem 5.1, Proposition 5.12 and Corollary 5.30 that, for $1 \leq k \leq n$, we have

$$\begin{aligned} TSD_k \sim TS_k &= \mathrm{Moore}\bigl(\mathrm{MI}(TSD_k) \setminus \rho_{TS_k}(TSD_k)\bigr) \\ &= \mathrm{Moore}\bigl(\mathrm{MI}(TSD_k) \setminus TS_k\bigr) \\ &= \{\, sh \in TSD_k \mid VI \in sh \,\}. \end{aligned} \tag{5.6}$$

Thus we have

$$TSD_k \sim (TSD_k \sim TS_k) = TS_k. \tag{5.7}$$

We have therefore extracted *all* the domain TS_k from TSD_k. So by letting $k = 1$, 2 in Eq. (5.6), we have found the complements of Con in Def and PS in PSD:

$$\begin{aligned} Def \sim Con &= \{\, sh \in Def \mid VI \in sh \,\}, \\ PSD \sim PS &= \{\, sh \in PSD \mid VI \in sh \,\}. \end{aligned}$$

Thus if we denote the domains induced by these complements as $Def^{\oplus}$ and $PSD^{\oplus}$, respectively, we have the following result.

Theorem 5.33

$$\begin{aligned}
Def \sim Con &= Def^{\oplus}, & Def \sim Def^{\oplus} &= Con, \\
PSD \sim PS &= PSD^{\oplus}, & PSD \sim PSD^{\oplus} &= PS.
\end{aligned}$$

Moreover, Con and $Def^{\oplus}$ form a minimal decomposition for Def and, similarly, PS and $PSD^{\oplus}$ form a minimal decomposition for PSD.

5.5.2 Removing the Dependency Domains

First we note that, by Theorem 5.28, Proposition 5.11, and Corollary 5.32, the complement of TSD_j in TSD_k, where $1 \leq j < k \leq n$, is given as follows:

$$\begin{aligned}
TSD_k \sim TSD_j &= \mathrm{Moore}\bigl(\mathrm{MI}(TSD_k) \setminus \rho_{TSD_j}(TSD_k)\bigr) \\
&= \mathrm{Moore}\bigl(\mathrm{MI}(TSD_k) \setminus TSD_j\bigr) \\
&= \bigl\{\, sh \in TSD_k \bigm| \forall S \in SG : \# S \leq j \implies S \in sh \,\bigr\}.
\end{aligned} \tag{5.8}$$

It therefore follows from Eq. (5.8) and setting $k = n$ that the complement of ρ_{TSD_j} in SH for $j < n$ is:

$$SH_j^+ \stackrel{\mathrm{def}}{=} SH \sim TSD_j = \bigl\{\, sh \in SH \bigm| \forall S \in SG : \# S \leq j \implies S \in sh \,\bigr\}. \tag{5.9}$$

In particular, in Eq. (5.9) when $j = 1$, we have the following result for Def, also proved in [FR96, Lemma 5.4]:

$$SH_{Def}^+ \stackrel{\mathrm{def}}{=} SH \sim Def = \bigl\{\, sh \in SH \bigm| \forall x \in VI : \{x\} \in sh \,\bigr\}.$$

Also, in Eq. (5.9) when $j = 2$, we have the following result for PSD:

$$SH_{PSD}^+ \stackrel{\mathrm{def}}{=} SH \sim PSD = \bigl\{\, sh \in SH \bigm| \forall S \in SG : \# S \leq 2 \implies S \in sh \,\bigr\}.$$

We next construct the complement of PSD with respect to Def. By Eq. (5.8),

$$PSD^+ \stackrel{\mathrm{def}}{=} PSD \sim Def = \bigl\{\, sh \in PSD \bigm| \forall x \in VI : \{x\} \in sh \,\bigr\}.$$

Then the complement factor $Def^- \stackrel{\mathrm{def}}{=} PSD \sim PSD^+$ is exactly $SH \sim SH_{Def}^+$ so that PSD and SH behave similarly for Def.

5.5.3 Completing the Decomposition

Just as for SH, the complement of SH_{Def}^+ using PS (or, more generally, using TS_j where $1 < j < n$) is SH_{Def}^+. By Corollary 5.30 and Theorem 5.1, as PS is dual-atomistic, the complement of PS in PSD^+ is given as follows.

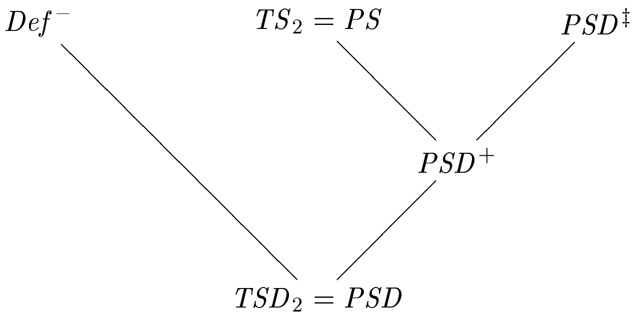

Figure 5.4: A non-trivial decomposition of *PSD*.

Theorem 5.34

$$\begin{aligned} PSD^\ddagger &\stackrel{\text{def}}{=} PSD^+ \sim PS = \{\, sh \in PSD \mid VI \in sh, \forall x \in VI : \{x\} \in sh \,\}, \\ PS &= PSD^+ \sim PSD^\ddagger. \end{aligned}$$

So, we have extracted *all* the domain *PS* from PSD^+ and we have the following result (see Figure 5.4).

Corollary 5.35 *Def^-, PS, and $PSD^\ddagger$ form a minimal decomposition for PSD.*

5.6 Discussion

By studying the set-sharing domain in a more general framework, we have been able to show that the domain *PSD* has a natural place in a scheme of domains based on *SH*. Since the well-known domain *Def* for groundness analysis is an instance of this scheme, we have been able to highlight the close relationship between *Def* and *PSD* and the many properties they share. In particular, it was somehow unexpected that these domains could both be obtained as instances of a single parametric construction. As another contribution, we have generalized and strengthened the results found in [CFW94, CFW98] and in Chapter 4, stating that

- *Def* is the quotient of *SH* with respect to the groundness domain $G \equiv Con$; and
- *PSD* is the quotient of *SH* with respect to $Con \sqcap PS$ (the reduced product of groundness and pair-sharing).

We have provided a minimal decomposition for *PSD* whose components include Def^- and *PS*. Moreover, we have shown that *Def* and *PSD* are *not* dual-atomistic and we have completely specified their meet-irreducible elements.

Our starting point was the work of Filé and Ranzato. In [FR96], they noted, as we have, that $SH^+_{Def} \sim PS = SH^+_{Def}$ so that nothing of the domain *PS* could be extracted from SH^+_{Def}. They observed that ρ_{PS} maps all dual-atoms that contain the sharing group *VI* to the top element SG and thus lose all pair-sharing information. To avoid this, they replaced the classical pair-sharing domain PS with the domain PS' where, for all $sh \in SH^{+}_{Def}$,

$$\rho_{PS'}(sh) = \rho_{PS}(sh) \setminus \bigl(\{VI\} \setminus sh\bigr),$$

and noted that $SH^{+}_{Def} \sim PS' = \{\, sh \in SH^{+}_{Def} \mid VI \in sh \,\}$. To understand the nature of this new domain PS', we first observe that,

$$PS' = PS \sqcap TS_n.$$

This is because $TS_n = \mathrm{MI}(TS_n) = \bigl\{SG \setminus \{VI\}, SG\bigr\}$. In addition,

$$SH^{+}_{Def} \sim TS_n = \{\, sh \in SH^{+}_{Def} \mid VI \in sh \,\},$$

which is precisely the same as $SH^{+}_{Def} \sim PS'$. As a consequence, since $SH^{+}_{Def} \sim PS = SH^{+}_{Def}$, it is not surprising that it is precisely the *added* component TS_n that is removed when we compute the complement for SH^{+}_{Def} with respect to PS'.

We would like to point out that, in our opinion, the problems outlined above are not the consequence of the particular domains considered. Rather, they are mainly related to the methodology for decomposing a domain. As shown here, complementation *alone* is not sufficient to obtain *truly minimal* decompositions of domains. The reason being that complementation only depends on the domain's data (that is, the domain elements and the partial order relation modeling their intrinsic precision), while it is completely independent from the domain operators that manipulate that data. In particular, if the concrete domain contains elements that are redundant with respect to its operators (because the observable behavior of these elements is exactly the same in all possible program contexts) then any factorization of the domain obtained by complementation will encode this redundancy. However, the theoretical solution to this problem is well-known [CFW94, CFW98, GR97, GRS98b] and it is straightforward to improve the methodology so as to obtain truly minimal decompositions: *first* remove all redundancies from the domain (this can be done by computing the quotient of the domain with respect to the observable behavior) and only *then* decompose it by complementation. This is precisely what is done here.

We conclude our discussion about complementation with a few remarks. It is our opinion that, from a theoretical point of view, complementation is an excellent concept to work with: by allowing the splitting of complex domains into simpler components, avoiding redundancies between them, it really enhances our understanding of the domains themselves.

However, as things stand at present, complementation has never been exploited from a practical point of view. This may be because it is easier to implement a single complex domain than to implement several simpler domains and integrate them together. Note that complementation requires the implementation of a full integration between components (i.e., the reduced product together with its corresponding best approximations of the concrete semantic operators), otherwise precision would be lost and the theoretical results would not apply.

Moreover, complementation appears to have little relevance when trying to design or evaluate better implementations of a known abstract domain. In particular, this reasoning applies to the use of complementation as a tool for obtaining space saving representations for domains. As a notable example, the GER representation for Pos [BS99] is a well-known domain decomposition that does enable significant memory and time savings with no precision loss. This is not (and could not be) based on complementation. Observe that the complement of G with respect to Pos is Pos itself. This is because of the isomorphisms $Pos \equiv SH$ [CS98] and $G \equiv Con \stackrel{\mathrm{def}}{=} TS_1$ so that, by Eq. (5.5), $Pos \sim G = Pos$. It is not difficult to observe that the same phenomenon happens if one considers the groundness equivalence component E, that is, $Pos \sim E = Pos$. Intuitively, each element of the domain E defines a partition of the variable of interest VI into groundness equivalence classes. In fact, it can be shown that two variables $x, y \in VI$ are ground-equivalent in the abstract element $sh \in SH \equiv Pos$ if and only if $\mathrm{rel}\bigl(\{x\}, sh\bigr) = \mathrm{rel}\bigl(\{y\}, sh\bigr)$. In particular, this implies both $\{x\} \notin sh$ and $\{y\} \notin sh$. Thus, it can be easily observed that in all the dual-atoms of Pos no variable is ground-equivalent to another variable (because each dual-atom lacks just a *single* sharing group).

Chapter 6

Freeness and Linearity

It is well-known that freeness and linearity positively interact with aliasing information, therefore allowing improvements to both the precision and the efficiency of the sharing analysis of logic programs. In this chapter we present a novel combination of set-sharing with freeness and linearity information which is characterized by an improved abstract unification operator. We provide a new abstraction function and prove the correctness of the analysis, both for the finite-tree and the rational-tree languages. Moreover, we show that the same notion of redundant information as identified in Chapter 4 also applies to this abstract domain combination.

6.1 Yet Another Domain Combination

Even though the set-sharing domain is, in a sense, remarkably precise, more precision is attainable by combining it with other domains. In particular, definite freeness and definite linearity information have received much attention by the literature on sharing analysis.

As argued informally by Søndergaard [Søn86], the mutual interaction between linearity and aliasing information can improve the accuracy of a sharing analysis. This observation has been formally applied in [CDY91] to the specification of the abstract mgu operator for the domain ASub. In his PhD thesis [Lan90], Langen proposed a similar integration with linearity, but for the set-sharing domain (he has also shown how the aliasing information allows for freeness to be computed with a good degree of accuracy, but freeness information was not exploited to improve aliasing). King [Kin94] has also shown how a more refined tracking of linearity allows for further precision improvements.

The synergy attainable from a bi-directional interaction between aliasing and freeness information was initially pointed out by Muthukumar and Hermenegildo [MH91, MH92]. Since then, several authors considered the integration of set-sharing with freeness, sometimes including additional explicit structural information [CDFB93, CDFB96, Fil94, KS94].

Building on the results obtained in [Søn86], [CDY91] and [MH91], but independently from [Lan90], Hans and Winkler [HW92] proposed a combined integration of freeness and linearity information with set-sharing. Similar combinations have been proposed in [BC93, BCM94a, BCM94b]. From a more pragmatic point of view, Codish et al. [CMB$^+$93, CMB$^+$95] integrate the information captured by the domains of [Søn86] and [MH91] by performing the analysis with both the domains at the same time, exchanging information between the two components at each step.

Most of the above proposals differ in the carrier of the underlying abstract domain. Even when considering the simplest domain combinations, where no explicit structural information is taken into account, there is no general consensus on the specification of the abstract unification procedure. From a theoretical point of view, once the abstract domain has been related to the concrete one by means of a Galois connection, it is always possible to specify the best correct approximation of each operator of the concrete semantics. However, as we will see in Chapter 8, empirical observations suggest that sub-optimal operators are likely to result in better complexity/precision trade-offs. As a consequence, it is almost impossible to identify "the right combination" of variable aliasing with freeness and linearity information, at least when practical issues, such as the complexity of the abstract unification procedure, are taken into account.

Given this state of affairs, we will now consider a domain combination whose carrier is essentially the same as specified by Langen [Lan90] and Hans and Winkler [HW92] (the same domain combination was also considered by Bruynooghe et al. [BCM94a, BCM94b], but with the addition of compoundness and explicit structural information). The novelty of our proposal lies in the specification of an improved abstract unification procedure, better exploiting the interaction between sharing and linearity. By extending the results of Chapter 3 to this combination, we provide a new abstraction function that can be applied to any logic language computing on domains of syntactic structures, with or without the occurs-check; by using this abstraction function, we also prove the correctness of the abstract unification procedure.

It is worth stressing that, though very precise, our abstract domain and operators are not meant to subsume all the similar approaches that can be found in the literature. Rather, the goal is to provide a strong basis, with formal results of correctness, where other proposals for improved precision can be plugged in. Some of them will be presented, discussed and experimentally evaluated in Chapter 8.

6.2 The Domain *SFL*

The abstract domain *SFL* is made up of three components: the first one is the set-sharing component, providing aliasing and groundness information; the other two components provide freeness and linearity information, each represented by simply recording those variables of interest that are known to enjoy the corresponding property. For the same reasons explained in Chapter 5, we consider a fixed set of variables of interest VI.

Definition 6.1 (The domain *SFL*.) *Let* $F \stackrel{\mathrm{def}}{=} \wp(VI)$ *and* $L \stackrel{\mathrm{def}}{=} \wp(VI)$ *be partially* *ordered by reverse subset inclusion. The abstract domain SFL is defined as*

$$\mathit{SFL} \stackrel{\mathrm{def}}{=} \bigl\{\, \langle sh, f, l \rangle \bigm| sh \in \mathit{SH}, f \in F, l \in L \,\bigr\}$$

and is ordered by $\leq_S$, the component-wise extension of the orderings defined on the sub-domains. With this ordering, SFL is a complete lattice whose least upper bound operation is denoted by alub_S. *The bottom element $(\varnothing, \mathit{VI}, \mathit{VI})$ will be denoted by $\bot_S$.*

The domain *SFL* contains many redundancies, i.e., different abstract elements representing the same set of concrete computation states. Reasoning informally,[1] all the elements where $f \not\subseteq \mathrm{vars}(sh)$, such as $\bot_S$, represent the semantics of those program fragments that have no successful computations: this is because any free variable necessarily shares (at least, with itself). Similarly, the element $d_1 = \langle sh, f, l \rangle$ has the same meaning as the element $d_2 = \langle sh, f, l' \rangle$, where $l' = \bigl(\mathit{VI} \setminus \mathrm{vars}(sh)\bigr) \cup f \cup l$: in this case, the reason is that any variable that is either ground or free is also necessarily linear.

All of these redundancies can be removed by taking, as abstract domain, the image of the concrete domain under the abstraction function. Apart from the simple cases shown above, it is somehow difficult to *explicitly* characterize such a set. For instance, as observed in [Fil94], the element

$$\bigl\langle \{xy, yz, xz\}, \{x, y, z\}, \{x, y, z\} \bigr\rangle \in \mathit{SFL} \tag{6.1}$$

does not correspond to the abstraction of any concrete computation state (and can be shown to be equivalent to $\bot_S$). Moreover, such an approach would be complicated by the necessity of proving that all the abstract operators are well-defined on the given abstract domain.

For the above reasons, we adopt the redundant domain *SFL*. It is worth stressing that these "spurious" elements do not cause problems to the analysis as far as its correctness is concerned. Rather, they might affect the precision of the analysis. However, the abstract operators of *SFL* are designed to avoid the simpler cases, while the more involved ones, such as (6.1), occur rarely in practice (we refer the interested reader to Section 8.8, where such a claim is confirmed by our experimental evaluation).

6.3 The Abstraction Function

When defining the abstraction function for the domain *SFL*, we need to solve the same problems faced in Chapter 3. Namely, we would like to specify an abstraction function that precisely captures the properties of interest. This function should be invariant under logical equivalence, so that different representations of the same set of rational trees will be abstracted uniformly. We proceed toward a solution by considering each domain component separately. For the set-sharing component, we use the occurrence operator 'occ' of Definition 3.26.

[1] A formal argument can only be obtained when the abstraction function has been specified.

6.3.1 The Freeness Operator

As for possible sharing, the definite freeness information is abstracted from a substitution in rational solved form by means of a fixpoint computation.

Definition 6.2 (Freeness functions.) *For each* $n \in \mathbb{N}$, $\mathrm{fvars}_n \colon \mathit{RSubst} \to \wp(\mathit{Vars})$ *is defined, for each* $\sigma \in \mathit{RSubst}$, *by*

$$\begin{aligned} \mathrm{fvars}_0(\sigma) &\stackrel{\mathrm{def}}{=} \mathit{Vars} \setminus \mathrm{dom}(\sigma), \\ \mathrm{fvars}_{n+1}(\sigma) &\stackrel{\mathrm{def}}{=} \mathrm{fvars}_n(\sigma) \cup \bigl\{\, y \in \mathrm{dom}(\sigma) \bigm| y\sigma \in \mathrm{fvars}_n(\sigma) \,\bigr\}. \end{aligned}$$

For each $n \in \mathbb{N}$, although $\mathrm{fvars}_n(\sigma)$ is an infinite set, its set complement $\mathit{Vars} \setminus \mathrm{fvars}_n(\sigma)$ is finite, so that each freeness function is finitely computable. Since $\mathrm{fvars}_n(\sigma) \subseteq \mathrm{fvars}_{n+1}(\sigma)$ and $\#\bigl(\mathit{Vars} \setminus \mathrm{fvars}_0(\sigma)\bigr) = \#\,\sigma$, there is an index $\ell \leq \#\,\sigma$ such that $\mathrm{fvars}_\ell(\sigma) = \mathrm{fvars}_n(\sigma)$ for all $n \geq \ell$.

Definition 6.3 (Freeness operator.) *For each* $\sigma \in \mathit{RSubst}$, *the* freeness operator $\mathrm{fvars} \colon \mathit{RSubst} \to \wp(\mathit{Vars})$ *is defined by*

$$\mathrm{fvars}(\sigma) \stackrel{\mathrm{def}}{=} \mathrm{fvars}_{\#\,\sigma}(\sigma).$$

Example 6.4 *Consider* $\sigma \in \mathit{RSubst}$, *where*

$$\sigma = \bigl\{x_1 \mapsto x_2, x_2 \mapsto f(x_3), x_3 \mapsto x_4, x_4 \mapsto x_5\bigr\}.$$

Then,

$$\begin{aligned} \mathrm{fvars}_0(\sigma) &= \mathit{Vars} \setminus \{x_1, x_2, x_3, x_4\}, \\ \mathrm{fvars}_1(\sigma) &= \mathit{Vars} \setminus \{x_1, x_2, x_3\}, \\ \mathrm{fvars}_2(\sigma) &= \mathit{Vars} \setminus \{x_1, x_2\} \\ &= \mathrm{fvars}(\sigma). \end{aligned}$$

Thus, $x_1 \notin \mathrm{fvars}(\sigma)$, *although* $x_1\sigma \in \mathit{Vars}$. *Also,* $x_3 \in \mathrm{fvars}(\sigma)$, *although* $x_3\sigma \in \mathrm{dom}(\sigma)$.

6.3.2 The Groundness Operator

As all ground trees are linear, a knowledge of the definite groundness information associated to a substitution can be useful for proving properties concerning the linearity abstraction. As we observed in Chapters 3 and 5, the set-sharing abstraction already captures groundness information. For both a simplified notation and a clearer intuitive reading, we now explicitly define the set of variables that are associated to ground trees by a substitution in *RSubst*. This set is expressed in terms of the 'occ' operator.

Definition 6.5 (Groundness operator.) *For each* $\sigma \in \mathit{RSubst}$, *the* groundness operator gvars: *RSubst* $\to \wp_{\mathrm{f}}(\mathit{Vars})$ *is defined by*

$$\mathrm{gvars}(\sigma) \stackrel{\mathrm{def}}{=} \bigl\{\, y \in \mathrm{dom}(\sigma) \bigm| \forall v \in \mathrm{vars}(\sigma) : y \notin \mathrm{occ}(\sigma, v) \,\bigr\}.$$

Example 6.6 *Consider* $\sigma \in \mathit{RSubst}$, *where*

$$\sigma = \bigl\{x_1 \mapsto x_2, x_2 \mapsto f(a), x_3 \mapsto x_4, x_4 \mapsto f(x_2, x_4)\bigr\}.$$

Then $\mathrm{gvars}(\sigma) = \{x_1, x_2, x_3, x_4\}$. *Observe that* $x_1 \in \mathrm{gvars}(\sigma)$, *although* $x_1\sigma \in \mathit{Vars}$. *Also,* $x_3 \in \mathrm{gvars}(\sigma)$, *although* $\mathrm{vars}(x_3\sigma^i) = \{x_2, x_4\} \neq \varnothing$ *for all* $i \geq 2$.

6.3.3 The Linearity Operator

Linearity information is more conveniently specified by characterizing the set of variables that are *not* linear by means of another fixpoint computation. The base for the inductive definition is specified by using both the 'occ' and the 'gvars' operators.

Definition 6.7 (Non-linearity functions.) *For each* $n \in \mathbb{N}$, *the* non-linearity function $\mathrm{nlvars}_n \colon \mathit{RSubst} \to \wp_{\mathrm{f}}(\mathit{Vars})$ *is defined, for each* $\sigma \in \mathit{RSubst}$, *by*

$$\begin{aligned}
\mathrm{nlvars}_0(\sigma) &\stackrel{\mathrm{def}}{=} \bigl\{\, y \in \mathrm{dom}(\sigma) \bigm| \exists v \in \mathrm{vars}(y\sigma) \setminus \mathrm{gvars}(\sigma) \mathrel{.} \neg\, \mathrm{occ_lin}(v, y\sigma) \,\bigr\} \\
&\qquad \cup \left\{\, y \in \mathrm{dom}(\sigma) \;\middle|\; \begin{array}{r} \exists w, z \in \mathrm{vars}(y\sigma) \mathrel{.} \exists v \in \mathrm{vars}(\sigma) \mathrel{.} \\ w \neq z \wedge \{w, z\} \subseteq \mathrm{occ}(\sigma, v) \end{array} \right\}, \\
\mathrm{nlvars}_{n+1}(\sigma) &\stackrel{\mathrm{def}}{=} \mathrm{nlvars}_n(\sigma) \cup \bigl\{\, y \in \mathrm{dom}(\sigma) \bigm| \mathrm{vars}(y\sigma) \cap \mathrm{nlvars}_n(\sigma) \neq \varnothing \,\bigr\}.
\end{aligned}$$

It is easy to observe that $\mathrm{nlvars}_n(\sigma) \subseteq \mathrm{nlvars}_{n+1}(\sigma)$ and $\#\,\mathrm{nlvars}_n(\sigma) \leq \#\,\sigma$, so that there is an index $\ell \leq \#\,\sigma$ such that $\mathrm{nlvars}_\ell(\sigma) = \mathrm{nlvars}_n(\sigma)$ for all $n \geq \ell$.

Definition 6.8 (Linearity operator.) *For each* $\sigma \in \mathit{RSubst}$, *the* linearity operator $\mathrm{lvars} \colon \mathit{RSubst} \to \wp(\mathit{Vars})$ *is defined by*

$$\mathrm{lvars}(\sigma) \stackrel{\mathrm{def}}{=} \mathit{Vars} \setminus \mathrm{nlvars}_{\#\,\sigma}(\sigma).$$

Example 6.9 *Consider* $\sigma \in \mathit{RSubst}$, *where*

$$\begin{aligned}
\sigma = \bigl\{&x_1 \mapsto f(x_2, x_2), x_2 \mapsto f(a, x_3), x_4 \mapsto g(x_2, x_5), \\
&x_5 \mapsto f(x_3), x_6 \mapsto g(a, x_1), x_7 \mapsto f(x_8, x_8), x_8 \mapsto f(x_8)\bigr\}.
\end{aligned}$$

Then,

$$\begin{aligned}
\mathrm{nlvars}_0(\sigma) &= \{x_1, x_4\}, \\
\mathrm{nlvars}_1(\sigma) &= \{x_1, x_4, x_6\} \\
&= \mathrm{nlvars}_2(\sigma).
\end{aligned}$$

Note that $x_1 \notin \mathrm{lvars}(\sigma)$ *due to the first case in the definition of* nlvars_0*, since* $x_2 \notin \mathrm{gvars}(\sigma)$*;* $x_4 \notin \mathrm{lvars}(\sigma)$ *due to the second case of the definition of* nlvars_0*, since* $\{x_2, x_5\} \subseteq \mathrm{occ}(\sigma, x_3)$*;* $x_6 \notin \mathrm{lvars}(\sigma)$ *since the non-linear variable* x_1 *occurs in* $x_6\sigma$*. Finally,* $x_7 \in \mathrm{lvars}(\sigma)$*, although, for all* $i > 0$*, the variable* x_8 *occurs more than once in the term* $x_7\sigma^i$*.*

The freeness, groundness and linearity operators precisely capture the intended properties over the domain of rational trees.

Proposition 6.10 *If* $\sigma \in RSubst$ *and* $y \in Vars$ *then*

$$y \in \mathrm{fvars}(\sigma) \iff \mathrm{rt}(y, \sigma) \in Vars, \tag{6.2}$$

$$y \in \mathrm{gvars}(\sigma) \iff \mathrm{rt}(y, \sigma) \in GTerms, \tag{6.3}$$

$$y \in \mathrm{lvars}(\sigma) \iff \mathrm{rt}(y, \sigma) \in LTerms. \tag{6.4}$$

6.3.4 The Abstraction Function for *SFL*

We are now in position to define the abstraction function mapping substitutions in rational solved form to elements of the domain *SFL*.

Definition 6.11 (The abstraction function for ***SFL*****.)** *For each* $\sigma \in RSubst$*, the function* $\alpha_S \colon RSubst \to SFL$ *is defined by*

$$\alpha_S(\sigma) \stackrel{\mathrm{def}}{=} \bigl\langle\, \mathrm{ssets}(\sigma), \mathrm{fvars}(\sigma) \cap VI, \mathrm{lvars}(\sigma) \cap VI \,\bigr\rangle,$$

where

$$\mathrm{ssets}(\sigma) \stackrel{\mathrm{def}}{=} \bigl\{\, \mathrm{occ}(\sigma, v) \cap VI \bigm| v \in Vars \,\bigr\} \setminus \{\varnothing\}.$$

The concrete domain $\mathcal{D}^\flat$ *is related to the abstract domain SFL by means of the* abstraction function $\alpha_S \colon \mathcal{D}^\flat \to SFL$ *such that, for each* $\Sigma \in \wp(RSubst)$*,*

$$\alpha_S(\Sigma) \stackrel{\mathrm{def}}{=} \mathrm{alub}_S\bigl\{\, \alpha_S(\sigma) \bigm| \sigma \in \Sigma \,\bigr\}.$$

Since the abstraction function α_S *is additive, it induces a Galois connection; the corresponding concretization function* γ_S *is provided by the adjoint* [CC77a]

$$\gamma_S\bigl(\langle sh, f, l\rangle\bigr) \stackrel{\mathrm{def}}{=} \bigl\{\, \sigma \in RSubst \bigm| \mathrm{ssets}(\sigma) \subseteq sh, \mathrm{fvars}(\sigma) \supseteq f, \mathrm{lvars}(\sigma) \supseteq l \,\bigr\}.$$

We introduced the new operator 'ssets' for notational convenience only: its role is essentially the same of the function $\alpha \colon RSubst \times \wp_{\mathrm{f}}(Vars) \to SS$ introduced in Definition 3.32, which, however, explicitly deals with the set of variables of interest.

With the definitions given in this section, one of our goals has been achieved: substitutions in *RSubst* that are equivalent have the same abstraction.

Theorem 6.12 *Let $\sigma, \tau \in RSubst$ be satisfiable in the syntactic equality theory T and suppose $T \vdash \forall(\sigma \leftrightarrow \tau)$. Then $\alpha_S(\sigma) = \alpha_S(\tau)$.*

6.3.5 Proofs of the Results of Section 6.3.

In Chapter 3 we introduced variable-idempotent substitutions as a tool to reason about the aliasing of variables when we cannot assume idempotence. We will now show that variable-idempotence is also useful when reasoning about the freeness of a rational term as well as the multiplicity of the variables occurring in it, that is, its linearity.

For a substitution $\sigma \in VSubst$, when computing the operators 'fvars' and 'nlvars' the fixpoint is reached after a single iteration.

Lemma 6.13 *For each $\sigma \in VSubst$ we have* $\mathrm{fvars}(\sigma) = \mathrm{fvars}_1(\sigma)$.

Proof. We show that $\mathrm{fvars}_2(\sigma) \subseteq \mathrm{fvars}_1(\sigma)$. Let $y \in \mathrm{fvars}_2(\sigma)$. By Definition 6.2, we have two cases:

1. if $y \in \mathrm{fvars}_1(\sigma)$ then there is nothing to prove;
2. assume now $y \in \mathrm{dom}(\sigma)$ and $y\sigma \in \mathrm{fvars}_1(\sigma)$. Again by Definition 6.2, we have two subcases:
   (a) if $y\sigma \in Vars \setminus \mathrm{dom}(\sigma)$ then we easily obtain $y \in \mathrm{fvars}_1(\sigma)$;
   (b) otherwise, let $y\sigma \in \mathrm{dom}(\sigma)$ and $y\sigma\sigma \in Vars \setminus \mathrm{dom}(\sigma)$. Then $y \neq y\sigma \neq y\sigma\sigma$ and $\mathrm{vars}(y\sigma) \neq \mathrm{vars}(y\sigma\sigma)$, which is a contradiction since $\sigma \in VSubst$.

□

Lemma 6.14 *For each $\sigma \in VSubst$ we have* $\mathrm{nlvars}(\sigma) = \mathrm{nlvars}_1(\sigma)$.

Proof. We show that $\mathrm{nlvars}_2(\sigma) \subseteq \mathrm{nlvars}_1(\sigma)$. Let $y \in \mathrm{nlvars}_2(\sigma)$. By Definition 6.7, we have two cases:

1. if $y \in \mathrm{nlvars}_1(\sigma)$ then there is nothing to prove;
2. let $y \in \mathrm{dom}(\sigma)$ and $z \in \mathrm{vars}(y\sigma) \cap \mathrm{nlvars}_1(\sigma)$. By Definition 6.7 we have two subcases:
   (a) if $z \in \mathrm{nlvars}_0(\sigma)$ then, again by Definition 6.7, $y \in \mathrm{nlvars}_1(\sigma)$;
   (b) otherwise, let $z \in \mathrm{dom}(\sigma)$ and suppose $\mathrm{vars}(z\sigma) \cap \mathrm{nlvars}_0(\sigma) \neq \varnothing$. However, since $\sigma \in VSubst$, we have $\mathrm{vars}(z\sigma) \subseteq \mathrm{vars}(y\sigma)$. Thus $\mathrm{vars}(y\sigma) \cap \mathrm{nlvars}_0(\sigma) \neq \varnothing$ and $y \in \mathrm{nlvars}_1(\sigma)$.

□

When $\sigma \in VSubst$, the following simplified characterizations for the operators 'fvars', 'gvars' and 'lvars' can be used.

Proposition 6.15 *For each $\sigma \in VSubst$, we have*

$$\mathrm{fvars}(\sigma) = \bigl\{\, y \in \mathit{Vars} \bigm| y\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma) \,\bigr\}, \tag{6.5}$$
$$\mathrm{gvars}(\sigma) = \bigl\{\, y \in \mathit{Vars} \bigm| \mathrm{vars}(y\sigma) \subseteq \mathrm{dom}(\sigma) \,\bigr\}, \tag{6.6}$$
$$\mathrm{lvars}(\sigma) = \left\{\, y \in \mathit{Vars} \;\middle|\; \begin{array}{l} \forall z \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma) : \mathrm{occ\_lin}(z, y\sigma), \\ \forall z \in \mathrm{vars}(y\sigma) \cap \mathrm{dom}(\sigma) : z \in \mathrm{gvars}(\sigma) \end{array} \right\}. \tag{6.7}$$

The proof of Proposition 6.15 relies on the following simple result, which will also be needed later.

Lemma 6.16 *For all $\sigma \in RSubst$, $x \in \mathrm{gvars}(\sigma)$ if and only if $\mathrm{vars}(x\sigma) \subseteq \mathrm{gvars}(\sigma)$.*

Proof. We prove the result by showing that $x \notin \mathrm{gvars}(\sigma)$ if and only if there exists $w \in \mathrm{vars}(x\sigma) \setminus \mathrm{gvars}(\sigma)$.

First, let $x \notin \mathrm{gvars}(\sigma)$. By Definition 6.5, there exists $v \in \mathit{Vars}$ such that $x \in \mathrm{occ}(\sigma, v)$. By Definitions 3.24 and 3.26, there exists $w \in \mathrm{vars}(x\sigma)$ such that $w \in \mathrm{occ}(\sigma, v)$. Thus, by Definition 6.5, $w \notin \mathrm{gvars}(\sigma)$.

Assume now that there exists $w \in \mathrm{vars}(x\sigma)$ such that $w \notin \mathrm{gvars}(\sigma)$. By Definition 6.5, there exists $v \in \mathit{Vars}$ such that $w \in \mathrm{occ}(\sigma, v)$. By Definitions 3.24 and 3.26, we have $x \in \mathrm{occ}(\sigma, v)$. Thus, by Definition 6.5, $x \notin \mathrm{gvars}(\sigma)$. □

Proof of Proposition 6.15. Equation (6.5) is easily obtained by applying Lemma 6.13 and then unfolding Definition 6.2.

Consider equation (6.6). By Definition 6.5, $y \in \mathrm{gvars}(\sigma)$ if and only if, for all $v \in \mathit{Vars}$, we have $y \notin \mathrm{occ}(\sigma, v)$. By Lemma 3.28, this holds if and only if there does not exist $v \in \mathit{Vars}$ such that $v \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma)$, i.e., if and only if $\mathrm{vars}(y\sigma) \subseteq \mathrm{dom}(\sigma)$.

Consider now equation (6.7). Given Definition 6.8, it is simpler to reason about the set complement of the left-hand side of equation (6.7). Thus, by applying Lemma 6.14 and then unfolding Definition 6.7, we have that $y \notin \mathrm{lvars}(\sigma)$ if and only if $y \in \mathrm{nlvars}(\sigma)$ if and only if any of the following holds:

$$y \in \mathrm{dom}(\sigma) \land \exists v \in \mathrm{vars}(y\sigma) \setminus \mathrm{gvars}(\sigma) \,.\, \neg\, \mathrm{occ\_lin}(v, y\sigma), \tag{6.8}$$
$$\begin{aligned} &y \in \mathrm{dom}(\sigma) \land \exists w_1, w_2 \in \mathrm{vars}(y\sigma) \,. \\ &\qquad \exists v \in \mathrm{vars}(\sigma) \,.\, w_1 \neq w_2 \land \{w_1, w_2\} \subseteq \mathrm{occ}(\sigma, v), \end{aligned} \tag{6.9}$$
$$\begin{aligned} &y \in \mathrm{dom}(\sigma) \land \exists y' \in \mathrm{vars}(y\sigma) \cap \mathrm{dom}(\sigma) \,. \\ &\qquad \exists v \in \mathrm{vars}(y'\sigma) \setminus \mathrm{gvars}(\sigma) \,.\, \neg\, \mathrm{occ\_lin}(v, y'\sigma), \end{aligned} \tag{6.10}$$
$$\begin{aligned} &y \in \mathrm{dom}(\sigma) \land \exists y' \in \mathrm{vars}(y\sigma) \cap \mathrm{dom}(\sigma) \,.\, \exists w_1, w_2 \in \mathrm{vars}(y'\sigma) \,. \\ &\qquad \exists v \in \mathrm{vars}(\sigma) \,.\, w_1 \neq w_2 \land \{w_1, w_2\} \subseteq \mathrm{occ}(\sigma, v). \end{aligned} \tag{6.11}$$

On the other hand, variable y is not in the right-hand side of equation (6.7) if and only if any of the following holds:

$$y \in \operatorname{dom}(\sigma) \wedge \exists z \in \operatorname{vars}(y\sigma) \setminus \operatorname{dom}(\sigma) \,.\, \neg \operatorname{occ\_lin}(z, y\sigma), \tag{6.12}$$

$$y \in \operatorname{dom}(\sigma) \wedge \exists z \in \operatorname{vars}(y\sigma) \cap \operatorname{dom}(\sigma) \,.\, z \notin \operatorname{gvars}(\sigma). \tag{6.13}$$

Thus, in order to complete the proof, we need to show that

$$(6.8) \vee (6.9) \vee (6.10) \vee (6.11) \iff (6.12) \vee (6.13).$$

We first prove the implication $\Rightarrow$.

1. Suppose that (6.8) holds. If $v \notin \operatorname{dom}(\sigma)$, by taking $z = v$, we have that (6.12) holds. Otherwise, $v \in \operatorname{dom}(\sigma)$, by taking $z = v$, we have that (6.13) holds.

2. Suppose that (6.9) holds. Note that, by Definition 6.5, we have that $w_1 \notin \operatorname{gvars}(\sigma)$ and $w_2 \notin \operatorname{gvars}(\sigma)$. Since $w_1 \neq w_2$, there exists an index $j \in \{1, 2\}$ such that $w_j \neq v$. Since $w_j \in \operatorname{occ}(\sigma, v)$, we obtain $w_j \in \operatorname{dom}(\sigma)$. Hence, by taking $z = w_j$, (6.13) holds.

3. Suppose that (6.10) holds. Then, by Lemma 6.16, $y' \notin \operatorname{gvars}(\sigma)$. As a consequence, by taking $z = y'$, we have that (6.13) holds.

4. Suppose that (6.11) holds. By Definition 6.5, we have $w_1 \notin \operatorname{gvars}(\sigma)$. Thus, as in the previous case, by Lemma 6.16 we obtain $y' \notin \operatorname{gvars}(\sigma)$ and, by taking $z = y'$, we have that (6.13) holds.

We now prove the other implication ($\Leftarrow$).

1. Suppose that (6.12) holds. Note that $z \notin \operatorname{dom}(\sigma)$ implies $z \notin \operatorname{gvars}(\sigma)$. Thus, by taking $v = z$, we have that (6.8) holds.

2. Finally, suppose that (6.13) holds. By Definition 6.5, there exists $w \in \mathit{Vars}$ such that $z \in \operatorname{occ}(\sigma, w)$. Since $z \in \operatorname{vars}(y\sigma)$, by Definitions 3.24 and 3.26, we have $y \in \operatorname{occ}(\sigma, w)$. Since $\sigma \in \mathit{VSubst}$, by Lemma 3.28 we obtain $w \in \operatorname{vars}(y\sigma)$. Also note that $z \neq w$, because $z \in \operatorname{dom}(\sigma)$ whereas $w \notin \operatorname{dom}(\sigma)$. Thus, by taking $w_1 = z$, $w_2 = w$ and $v = w$, we have that (6.9) holds.

□

The following proposition shows that, for a substitution $\sigma \in \mathit{VSubst}$, the freeness, groundness and linearity operators precisely capture the intended properties.

Proposition 6.17 *Let $\sigma \in \mathit{VSubst}$ and $y \in \mathit{Vars}$. Then:*

$$y \in \operatorname{fvars}(\sigma) \iff \operatorname{rt}(y, \sigma) \in \mathit{Vars}, \tag{6.14}$$

$$y \in \operatorname{gvars}(\sigma) \iff \operatorname{rt}(y, \sigma) \in \mathit{GTerms}, \tag{6.15}$$

$$y \in \operatorname{lvars}(\sigma) \iff \operatorname{rt}(y, \sigma) \in \mathit{LTerms}. \tag{6.16}$$

Proof. We start by proving item (6.14). By Proposition 6.15, $y \in \mathrm{fvars}(\sigma)$ if and only if $y\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$. To prove the first implication ($\Rightarrow$), let $y\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$. Then, $\mathrm{rt}(y, \sigma) \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$ and, more generally, $\mathrm{rt}(y, \sigma) \in \mathit{Vars}$.

To prove the other implication ($\Leftarrow$), assume that $\mathrm{rt}(y, \sigma) \in \mathit{Vars}$. We prove by contradiction that $y\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$. In fact, assume that $y\sigma \notin \mathit{Vars} \setminus \mathrm{dom}(\sigma)$. We have two cases:

1. if $y\sigma \notin \mathit{Vars}$ then, by definition, $\mathrm{rt}(y, \sigma) \notin \mathit{Vars}$.

2. otherwise, let $y\sigma \in \mathrm{dom}(\sigma)$. Thus, we have $y \neq y\sigma \neq y\sigma\sigma$, so that $(y \mapsto y\sigma) \in \sigma$ and $(y\sigma \mapsto y\sigma\sigma) \in \sigma$. Since $\sigma \in \mathit{VSubst}$, we also have $\{y\sigma\} = \mathrm{vars}(y\sigma) = \mathrm{vars}(y\sigma\sigma)$. Therefore, $y\sigma\sigma \notin \mathit{Vars}$, so that there exists an $n > 0$ such that $y\sigma\sigma = f(t_1, \ldots, t_n)$, and $\mathrm{size}(y\sigma\sigma) > 1$. Since $\mathrm{rt}(y, \sigma) \in \mathit{Vars}$, this is a contradiction because we also have $\mathrm{size}(y\sigma^2) \leq \mathrm{size}\bigl(\mathrm{rt}(y, \sigma)\bigr) = 1$.

We now prove item (6.15). By Definition 6.5, we have $y \in \mathrm{gvars}(\sigma)$ if and only if $y \notin \mathrm{occ}(\sigma, v)$, for all $v \in \mathit{Vars}$. By Lemma 3.38, this is equivalent to $v \notin \mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr)$, for all $v \in \mathit{Vars}$. Thus, $\mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr) = \varnothing$ and $\mathrm{rt}(y, \sigma) \in \mathit{GTerms}$.

Finally, we prove item (6.16). In order to prove the first implication ($\Rightarrow$), assume $y \in \mathrm{lvars}(\sigma)$ so that, by Proposition 6.15, we have

$$\forall z \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma) : \mathrm{occ\_lin}(z, y\sigma), \tag{6.17}$$

$$\forall z \in \mathrm{vars}(y\sigma) \cap \mathrm{dom}(\sigma) : \mathrm{vars}(z\sigma) \subseteq \mathrm{dom}(\sigma). \tag{6.18}$$

We need to show that $\mathrm{rt}(y, \sigma) \in \mathit{LTerms}$ and we proceed by contradiction, negating the conclusion. Thus assume there exists $v \in \mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr)$ such that $\mathrm{occ\_lin}\bigl(v, \mathrm{rt}(y, \sigma)\bigr)$ does not hold. Note that $v \notin \mathrm{dom}(\sigma)$; also, since $\sigma \in \mathit{VSubst}$, $\mathrm{vars}(y\sigma) = \mathrm{vars}(y\sigma^i)$, for all $i > 0$, so that $v \in \mathrm{vars}(y\sigma)$. If $\mathrm{occ\_lin}(v, y\sigma)$ does not hold, then we obtain the negation of equation (6.17), hence a contradiction. So, assume that $\mathrm{occ\_lin}(v, y\sigma)$ hold. As a consequence, there exists an index $j > 1$ such that $\mathrm{occ\_lin}(v, y\sigma^{j-1})$ holds and $\mathrm{occ\_lin}(v, y\sigma^j)$ does not hold. Thus, there exists $w \in \mathrm{vars}(y\sigma^{j-1}) \cap \mathrm{dom}(\sigma)$ such that $v \in \mathrm{vars}(w\sigma) \setminus \mathrm{dom}(\sigma)$. Since $\sigma \in \mathit{VSubst}$, $w \in \mathrm{vars}(y\sigma^{j-1})$ if and only if $w \in \mathrm{vars}(y\sigma)$. Hence $j = 2$, $w \in \mathrm{vars}(y\sigma) \cap \mathrm{dom}(\sigma)$ and $\mathrm{vars}(w\sigma) \nsubseteq \mathrm{dom}(\sigma)$. Hence we have contradicted equation (6.18).

To prove the other implication ($\Leftarrow$), assume $\mathrm{rt}(y, \sigma) \in \mathit{LTerms}$, so that, by definition, we have $\mathrm{occ\_lin}\bigl(z, \mathrm{rt}(y, \sigma)\bigr)$, for all $z \in \mathrm{vars}\bigl(\mathrm{rt}(y, \sigma)\bigr)$. We need to show that equations (6.17) and (6.18) hold. We proceed by contradiction, negating the conclusion. There are two cases.

1. Assume that equation (6.17) does not hold, i.e., there exists $z \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma)$ such that $\mathrm{occ\_lin}(z, y\sigma)$ does not hold. Then, for all $i > 0$, $z \in \mathrm{vars}(y\sigma^i)$, but $\mathrm{occ\_lin}(z, y\sigma^i)$ does not hold. As a consequence, $\mathrm{occ\_lin}\bigl(z, \mathrm{rt}(y, \sigma)\bigr)$ does not hold and $\mathrm{rt}(y, \sigma) \notin \mathit{LTerms}$, obtaining a contradiction.

2. Assume that equation (6.18) does not hold, i.e., there exists $z \in \mathrm{vars}(y\sigma) \cap \mathrm{dom}(\sigma)$ such that $\mathrm{vars}(z\sigma) \not\subseteq \mathrm{dom}(\sigma)$. Thus, let $v \in \mathrm{vars}(z\sigma) \setminus \mathrm{dom}(\sigma)$. Since $\sigma \in \mathit{VSubst}$ and $v \in \mathrm{vars}(y\sigma\sigma)$, then $v \in \mathrm{vars}(y\sigma)$. Then, since $z \in \mathrm{vars}(y\sigma) \cap \mathrm{dom}(\sigma)$, $\mathrm{occ\_lin}(v, y\sigma\sigma)$ does not hold. Also, since $v \notin \mathrm{dom}(\sigma)$, for all $i \geq 2$, $\mathrm{occ\_lin}(v, y\sigma^i)$ does not hold. By definition, $\mathrm{occ\_lin}\bigl(v, \mathrm{rt}(y, \sigma)\bigr)$ does not hold and $\mathrm{rt}(y, \sigma) \notin \mathit{LTerms}$, obtaining the contradiction.

□

In order to prove Proposition 6.10, i.e., to show that the freeness and linearity operators precisely capture the intended properties even for arbitrary substitutions in *RSubst*, we now prove that these operators are invariant under the application of S-steps.

Lemma 6.18 *For each $m > 0$ we have*

$$\mathrm{fvars}_{m-1}(\sigma) \subseteq \mathrm{fvars}_m(\sigma),$$
$$\mathrm{nlvars}_{m-1}(\sigma) \subseteq \mathrm{nlvars}_m(\sigma).$$

Proof. Straightforward by Definitions 6.2 and 6.7. □

Lemma 6.19 *Let $\sigma, \sigma' \in \mathit{RSubst}$ and $\sigma \overset{S}{\longmapsto} \sigma'$. Then $\mathrm{fvars}(\sigma) = \mathrm{fvars}(\sigma')$.*

Proof. Let $(x \mapsto t), (y \mapsto s) \in \sigma$, where $x \neq y$, such that

$$\sigma' = \bigl(\sigma \setminus \{y \mapsto s\}\bigr) \cup \bigl\{y \mapsto s[x/t]\bigr\}. \tag{6.19}$$

If $x \notin \mathrm{vars}(s)$ then we have $\sigma = \sigma'$ and the result trivially holds. Thus, we assume $x \in \mathrm{vars}(s)$. We prove, by induction on $m \geq 0$, that we have

$$\mathrm{fvars}_m(\sigma) \subseteq \mathrm{fvars}(\sigma').$$

For the base case, when $m = 0$, by Theorem 3.15 we have

$$\begin{aligned} \mathrm{fvars}_0(\sigma) &= \mathit{Vars} \setminus \mathrm{dom}(\sigma) \\ &= \mathit{Vars} \setminus \mathrm{dom}(\sigma') \\ &= \mathrm{fvars}_0(\sigma') \\ &\subseteq \mathrm{fvars}(\sigma'). \end{aligned}$$

For the inductive step, when $m > 0$, assume $\mathrm{fvars}_{m-1}(\sigma) \subseteq \mathrm{fvars}(\sigma')$. Assume that $z \in \mathrm{fvars}_m(\sigma)$.

$$\mathrm{fvars}_m(\sigma) = \mathrm{fvars}_{m-1}(\sigma) \cup \bigl\{\, z \in \mathrm{dom}(\sigma) \bigm| z\sigma \in \mathrm{fvars}_{m-1}(\sigma) \,\bigr\}.$$

If $z \in \mathrm{fvars}_{m-1}(\sigma)$ then, by the inductive hypothesis, $z \in \mathrm{fvars}(\sigma')$. Assume now that

$z \in \{ z \in \mathrm{dom}(\sigma) \mid z\sigma \in \mathrm{fvars}_{m-1}(\sigma) \}$. If $z \neq y$ then $z\sigma = z\sigma'$ so that,

$$\{ z \in \mathrm{dom}(\sigma) \mid z\sigma \in \mathrm{fvars}_{m-1}(\sigma) \} \subseteq \mathrm{fvars}(\sigma') \cup \{ z \in \mathrm{dom}(\sigma') \mid z\sigma \in \mathrm{fvars}(\sigma') \}.$$

Thus, by Definitions 6.2 and 6.3, we have $z \in \mathrm{fvars}_{\ell+1}(\sigma') = \mathrm{fvars}(\sigma')$.

If otherwise $z = y$ then $s = z\sigma \in \mathrm{fvars}_{m-1}(\sigma)$; however, since $x \in \mathrm{vars}(s)$, we also have $x = s = z\sigma$ and $z\sigma \in \mathrm{fvars}_{m-1}(\sigma')$. Since $x \in \mathrm{dom}(\sigma)$, $x \notin \mathrm{fvars}_0(\sigma)$. By Definitions 6.2 and 6.3, we have that $x \in \mathrm{fvars}_{m-1}(\sigma)$ implies $t \in \mathrm{fvars}_{m-2}(\sigma)$. By Lemma 6.18 and the inductive hypothesis, $t \in \mathrm{fvars}(\sigma')$. Since $z\sigma' = t$ and $z \in \mathrm{dom}(\sigma')$ (by Theorem 3.15), then

$$z \in \{ z \in \mathrm{dom}(\sigma') \mid z\sigma' \in \mathrm{fvars}(\sigma') \} \cup \mathrm{fvars}(\sigma') = \mathrm{fvars}(\sigma').$$

Thus, it holds $\mathrm{fvars}(\sigma) \subseteq \mathrm{fvars}(\sigma')$. We now prove, for each $m > 0$,

$$\mathrm{fvars}_m(\sigma') \subseteq \mathrm{fvars}(\sigma). \tag{6.20}$$

The proof is by induction on m. The base case, when $m = 0$, is proved as above. For the inductive step, when $m > 0$, assume $\mathrm{fvars}_{m-1}(\sigma') \subseteq \mathrm{fvars}(\sigma)$. Assume that $z \in \mathrm{fvars}_m(\sigma')$.

$$\mathrm{fvars}_m(\sigma') = \mathrm{fvars}_{m-1}(\sigma') \cup \{ z \in \mathrm{dom}(\sigma) \mid z\sigma \in \mathrm{fvars}_{m-1}(\sigma') \}.$$

If $z \in \mathrm{fvars}_{m-1}(\sigma')$ then, by the inductive hypothesis, $z \in \mathrm{fvars}(\sigma)$. Assume now that $z \in \{ z \in \mathrm{dom}(\sigma') \mid z\sigma \in \mathrm{fvars}_{m-1}(\sigma') \}$. If $z \neq y$ then $z\sigma' = z\sigma$ so that,

$$\{ z \in \mathrm{dom}(\sigma') \mid z\sigma' \in \mathrm{fvars}_{m-1}(\sigma') \} \subseteq \mathrm{fvars}(\sigma) \cup \{ z \in \mathrm{dom}(\sigma) \mid z\sigma \in \mathrm{fvars}(\sigma) \}.$$

Hence, by Definitions 6.2 and 6.3, we have $z \in \mathrm{fvars}_{\ell+1}(\sigma) = \mathrm{fvars}(\sigma)$.

If otherwise $z = y$, then $s[x/t] = z\sigma'$; since $x \in \mathrm{vars}(s)$, we have $x[x/t] = s[x/t] = z\sigma'$ and $z\sigma' \in \mathrm{fvars}_{m-1}(\sigma')$. By Lemma 6.18 and by the inductive hypothesis, $t \in \mathrm{fvars}(\sigma)$. Note that $x \in \mathrm{dom}(\sigma)$, $x\sigma = t$ and $t \in \mathrm{fvars}(\sigma)$. Then, by Definitions 6.2 and 6.3, we have $x \in \mathrm{fvars}(\sigma)$. In the same way, since $y \in \mathrm{dom}(\sigma)$, $y\sigma = x$ and $x \in \mathrm{fvars}(\sigma)$, we can conclude that $y \in \mathrm{fvars}(\sigma)$. □

Lemma 6.20 *Let $\sigma, \sigma' \in RSubst$ and $\sigma \xmapsto{\mathcal{S}} \sigma'$. Then* $\mathrm{gvars}(\sigma) = \mathrm{gvars}(\sigma')$.

Proof. By applying Definition 6.5, Theorem 3.15, Lemma 3.36 and again Definition 6.5, we obtain

$$\begin{aligned}
\mathrm{gvars}(\sigma) &= \{ x \in \mathrm{dom}(\sigma) \mid \forall v \in \mathrm{vars}(\sigma) : x \notin \mathrm{occ}(\sigma, v) \} \\
&= \{ x \in \mathrm{dom}(\sigma') \mid \forall v \in \mathrm{vars}(\sigma') : x \notin \mathrm{occ}(\sigma, v) \} \\
&= \{ x \in \mathrm{dom}(\sigma') \mid \forall v \in \mathrm{vars}(\sigma') : x \notin \mathrm{occ}(\sigma', v) \} \\
&= \mathrm{gvars}(\sigma').
\end{aligned}$$

□

The following three simple results will be used in the proof of the invariance of 'lvars' under $\mathcal{S}$-transformation.

Lemma 6.21 *Let $\sigma \in RSubst$ and $(x \mapsto t) \in \sigma$. If $\{x, w\} \subseteq \mathrm{occ}(\sigma, v)$ then there exists $w' \in \mathrm{vars}(t)$ such that $\{w, w'\} \subseteq \mathrm{occ}(\sigma, v)$.*

Proof. Assume that $\{x, w\} \subseteq \mathrm{occ}(\sigma, v)$, for some v. Since $x \in \mathrm{dom}(\sigma)$, by Definitions 3.24 and 3.26, $x \in \mathrm{occ}(\sigma, v)$ if and only if $\mathrm{vars}(x\sigma) \cap \mathrm{occ}(\sigma, v) \neq \varnothing$. Then, since $x\sigma = t$, there must exist $w' \in \mathrm{vars}(t)$ such that $w' \in \mathrm{occ}(\sigma, v)$. Hence, $\{w, w'\} \subseteq \mathrm{occ}(\sigma, v)$. □

Lemma 6.22 *Let $\sigma \in RSubst$ and $(x \mapsto t) \in \sigma$. If $v \in \mathrm{vars}(t) \setminus \mathrm{gvars}(\sigma)$ then there exists $w \in \mathrm{vars}(\sigma)$ such that $\{x, v\} \subseteq \mathrm{occ}(\sigma, w)$.*

Proof. Assume $v \notin \mathrm{gvars}(\sigma)$. We have two cases:

1. $v \notin \mathrm{dom}(\sigma)$. In this case, by Definitions 3.24 and 3.26, $v \in \mathrm{occ}(\sigma, v)$. Since $x\sigma = t$, $v \in \mathrm{vars}(t)$ and $v \in \mathrm{occ}(\sigma, v)$, by Definitions 3.24 and 3.26, we can conclude that $\{x, v\} \in \mathrm{occ}(\sigma, v)$.

2. $v \in \mathrm{dom}(\sigma)$. Since $v \notin \mathrm{gvars}(\sigma)$ then $\exists w$ such that $v \in \mathrm{occ}(\sigma, w)$. Since $x\sigma = t$, $v \in \mathrm{vars}(t)$ and $v \in \mathrm{occ}(\sigma, w)$, by Definitions 3.24 and 3.26, we can conclude that $\{x, v\} \in \mathrm{occ}(\sigma, w)$.

□

Lemma 6.23 *Let $\sigma \in RSubst$ and $(x \mapsto t) \in \sigma$. If $\{w, z\} \subseteq \mathrm{occ}(\sigma, v)$ and $z \in \mathrm{vars}(t)$, then $\{w, x\} \subseteq \mathrm{occ}(\sigma, v)$.*

Proof. Assume that $\{w, z\} \subseteq \mathrm{occ}(\sigma, v)$. Since $z \in \mathrm{occ}(\sigma, v)$, $z \in \mathrm{vars}(t)$ and $x\sigma = t$, by Definitions 3.24 and 3.26, we can conclude that $\{x, w\} \in \mathrm{occ}(\sigma, v)$. □

Lemma 6.24 *Let $\sigma, \sigma' \in RSubst$ and $\sigma \overset{\mathcal{S}}{\longmapsto} \sigma'$. Then $\mathrm{lvars}(\sigma) = \mathrm{lvars}(\sigma')$.*

Proof. By Definition 6.8, the thesis is equivalent to $\mathrm{nlvars}(\sigma) = \mathrm{nlvars}(\sigma')$.

Let $(x \mapsto t), (y \mapsto s) \in \sigma$, where $x \neq y$, such that

$$\sigma' = \bigl(\sigma \setminus \{y \mapsto s\}\bigr) \cup \bigl\{y \mapsto s[x/t]\bigr\}.$$

If $x \notin \mathrm{vars}(s)$ then $\sigma = \sigma'$ and the result trivially holds. Thus, we assume $x \in \mathrm{vars}(s)$.

We first prove, by induction on $m \geq 0$, that we have

$$\mathrm{nlvars}_m(\sigma) \subseteq \mathrm{nlvars}_m(\sigma').$$

From this, by Lemma 6.18, we obtain the inclusion $\mathrm{nlvars}(\sigma) \subseteq \mathrm{nlvars}(\sigma')$.

For the base case, when $m = 0$, let $z \in \mathrm{nlvars}_0(\sigma)$. Then, by Definition 6.7, we have

$$z \in \{\, y \in \mathrm{dom}(\sigma) \mid \exists v \in \mathrm{vars}(y\sigma) \setminus \mathrm{gvars}(\sigma) \,.\, \neg\,\mathrm{occ\_lin}(v, y\sigma) \,\} \cup \left\{ y \in \mathrm{dom}(\sigma) \;\middle|\; \begin{array}{r} \exists w_1, w_2 \in \mathrm{vars}(y\sigma) \,.\, \exists v \in \mathrm{vars}(\sigma) \,. \\ w_1 \neq w_2 \wedge \{w_1, w_2\} \subseteq \mathrm{occ}(\sigma, v) \end{array} \right\}.$$

We have two cases.

1. Suppose $z \in \mathrm{dom}(\sigma)$ and there exists $v \in \mathrm{vars}(z\sigma) \setminus \mathrm{gvars}(\sigma)$ such that $\mathrm{occ\_lin}(v, z\sigma)$ does not hold. By Theorem 3.15 and Lemma 6.20, we have that $z \in \mathrm{dom}(\sigma')$ and $v \in \mathrm{vars}(z\sigma') \setminus \mathrm{gvars}(\sigma')$. If $z \neq y$, then $z\sigma' = z\sigma$. Hence $\mathrm{occ\_lin}(v, z\sigma')$ does not hold and, by Definition 6.7, we have $z \in \mathrm{nlvars}_0(\sigma')$. Otherwise, if $z = y$, then we have $z\sigma = y\sigma = s$ and $z\sigma' = y\sigma' = s[x/t]$. We have two subcases.

   (a) If $v \neq x$, then $v \in \mathrm{vars}(y\sigma')$ and $\neg\,\mathrm{occ\_lin}(v, y\sigma')$. Thus, by Definition 6.7, $z = y \in \mathrm{nlvars}_0(\sigma')$.

   (b) Otherwise, let $v = x$. As $x \notin \mathrm{gvars}(\sigma)$, by Lemma 6.16, there exists $w \in \mathrm{vars}(t)$ such that $w \notin \mathrm{gvars}(\sigma)$. Moreover, since $\mathrm{occ\_lin}(x, s)$ does not hold, we also have that $\mathrm{occ\_lin}(w, s[x/t])$ does not hold. Also, by Lemma 6.20, $w \notin \mathrm{gvars}(\sigma')$. Thus, by Definition 6.7, $z = y \in \mathrm{nlvars}_0(\sigma')$.

2. Suppose now $z \in \mathrm{dom}(\sigma)$ and there exist $\{w_1, w_2\} \subseteq \mathrm{vars}(z\sigma)$, with $w_1 \neq w_2$, and $v \in \mathrm{vars}(\sigma)$ such that $\{w_1, w_2\} \subseteq \mathrm{occ}(\sigma, v)$. By Theorem 3.15, $z \in \mathrm{dom}(\sigma')$. By Lemma 3.36, we have $\mathrm{occ}(\sigma, v) = \mathrm{occ}(\sigma', v)$. If $z \neq y$ then we have $z\sigma = z\sigma'$, so that $\{w_1, w_2\} \subseteq \mathrm{vars}(z\sigma')$. Hence, by Definition 6.7, $z \in \mathrm{nlvars}_0(\sigma')$. Otherwise, let $z = y$, so that $z\sigma = y\sigma = s$ and $z\sigma' = y\sigma' = s[x/t]$. We have two subcases.

   (a) If $x \neq w_1$ and $x \neq w_2$, then we have $\{w_1, w_2\} \subseteq \mathrm{vars}(y\sigma')$. Hence, by Definition 6.7, $z = y \in \mathrm{nlvars}_0(\sigma')$.

   (b) Suppose now there exists an index $i \in \{1, 2\}$ such that $x = w_i$. Without loss of generality, we can assume $i = 1$, i.e., $x = w_1$ and $x \neq w_2$. Since we have $\{x, w_2\} \subseteq \mathrm{occ}(\sigma, v) = \mathrm{occ}(\sigma', v)$ and $(x \mapsto t) \in \sigma$, by Lemma 6.21 there exists $w' \in \mathrm{vars}(t)$ such that $\{w', w_2\} \subseteq \mathrm{occ}(\sigma', v)$. If $w' \neq w_2$ then, by Definition 6.7, we have $z = y \in \mathrm{nlvars}_0(\sigma')$. Otherwise, assume $w' = w_2$. Hence we have $w' \in \mathrm{vars}(s)$ and $w' \in \mathrm{vars}(t)$, so that $\mathrm{occ\_lin}(w', y\sigma')$ does not hold. Since $w' \in \mathrm{occ}(\sigma, v)$, by Definition 6.5 and Lemma 6.20, $w' \notin \mathrm{gvars}(\sigma) = \mathrm{gvars}(\sigma')$. Therefore, by Definition 6.7, we obtain $z = y \in \mathrm{nlvars}_0(\sigma')$.

Consider now the inductive case, when $m > 0$. Thus, let $z \in \mathrm{nlvars}_m(\sigma)$ and assume that $\mathrm{nlvars}_{m-1}(\sigma) \subseteq \mathrm{nlvars}_{m-1}(\sigma')$ holds. By Definition 6.7 we have

$$z \in \mathrm{nlvars}_{m-1}(\sigma) \cup \{\, y \in \mathrm{dom}(\sigma) \mid \mathrm{vars}(y\sigma) \cap \mathrm{nlvars}_{m-1}(\sigma) \neq \varnothing \,\}.$$

If $z \in \mathrm{nlvars}_{m-1}(\sigma)$ then, by the inductive hypothesis, $z \in \mathrm{nlvars}_{m-1}(\sigma')$. Hence, by Lemma 6.18, $z \in \mathrm{nlvars}_m(\sigma')$. Otherwise, let $z \in \mathrm{dom}(\sigma)$ and $v \in \mathrm{vars}(z\sigma) \cap \mathrm{nlvars}_{m-1}(\sigma)$. By the inductive hypothesis, $v \in \mathrm{nlvars}_{m-1}(\sigma')$. If $z \neq y$, then we have $z\sigma = z\sigma'$, so that $v \in \mathrm{vars}(z\sigma')$ and, by Definition 6.7, $z \in \mathrm{nlvars}_m(\sigma')$. Suppose now $z = y$, so that $z\sigma = s$ and $z\sigma' = s[x/t]$. We have two cases.

1. If $v \neq x$, then $v \in \mathrm{vars}(y\sigma')$. By Definition 6.7, $z = y \in \mathrm{nlvars}_m(\sigma')$.
2. Otherwise, let $v = x$, so that $x \in \mathrm{nlvars}_{m-1}(\sigma)$. We have three subcases.
   (a) Suppose there exists $w \in \mathrm{vars}(t) \setminus \mathrm{gvars}(\sigma)$ such that $\mathrm{occ\_lin}(w, t)$ does not hold. Then, by Lemma 6.20, $w \in \mathrm{vars}\big(s[x/t]\big) \setminus \mathrm{gvars}(\sigma')$ and $\mathrm{occ\_lin}\big(w, s[x/t]\big)$ does not hold. As a consequence, by Definition 6.7 and Lemma 6.18, we obtain $z = y \in \mathrm{nlvars}_0(\sigma') \subseteq \mathrm{nlvars}_m(\sigma')$.
   (b) Suppose now there exist $\{w_1, w_2\} \subseteq \mathrm{vars}(t)$, with $w_1 \neq w_2$, and $v \in \mathrm{vars}(\sigma)$ such that $\{w_1, w_2\} \subseteq \mathrm{occ}(\sigma, v)$. By Lemma 3.36, we have $\mathrm{occ}(\sigma, v) = \mathrm{occ}(\sigma', v)$. Also, $x\sigma = x\sigma' = t$, so that $\{w_1, w_2\} \subseteq \mathrm{vars}\big(s[x/t]\big)$. By Definition 6.7 and Lemma 6.18, we have $z = y \in \mathrm{nlvars}_0(\sigma') \subseteq \mathrm{nlvars}_m(\sigma')$.
   (c) Finally, suppose $\exists w \in \mathrm{vars}(t)$ such that $w \in \mathrm{nlvars}_{m-2}(\sigma)$ (so that $m > 1$). By the inductive hypothesis, $w \in \mathrm{nlvars}_{m-2}(\sigma')$. Also, $w \in \mathrm{vars}\big(s[x/t]\big)$ so that, by Definition 6.7 and Lemma 6.18, we have $z = y \in \mathrm{nlvars}_{m-1}(\sigma') \subseteq \mathrm{nlvars}_m(\sigma')$.

We now prove, again by induction on $m \geq 0$, that we have

$$\mathrm{nlvars}_m(\sigma') \subseteq \mathrm{nlvars}_{m+1}(\sigma).$$

From this, by Lemma 6.18, we obtain the inclusion $\mathrm{nlvars}(\sigma') \subseteq \mathrm{nlvars}(\sigma)$, therefore completing the proof.

For the base case, when $m = 0$, let $z \in \mathrm{nlvars}_0(\sigma')$. Then, by Definition 6.7, we have

$$z \in \big\{\, y \in \mathrm{dom}(\sigma') \;\big|\; \exists v \in \mathrm{vars}(y\sigma') \setminus \mathrm{gvars}(\sigma') \,.\, \neg\, \mathrm{occ\_lin}(v, y\sigma') \,\big\} \\ \cup \left\{\, y \in \mathrm{dom}(\sigma') \;\middle|\; \begin{array}{r} \exists w_1, w_2 \in \mathrm{vars}(y\sigma') \,.\, \exists v \in \mathrm{vars}(\sigma') \,. \\ w_1 \neq w_2 \wedge \{w_1, w_2\} \subseteq \mathrm{occ}(\sigma', v) \end{array} \right\}.$$

We have two cases.

1. Let $z \in \mathrm{dom}(\sigma')$ and suppose there exists $v \in \mathrm{vars}(z\sigma') \setminus \mathrm{gvars}(\sigma')$ such that $\mathrm{occ\_lin}(v, z\sigma')$ does not hold. By Theorem 3.15 and Lemma 6.20, $z \in \mathrm{dom}(\sigma)$ and $v \notin \mathrm{gvars}(\sigma)$. If $z \neq y$, then $z\sigma = z\sigma'$. Hence $v \in \mathrm{vars}(z\sigma)$ and $\mathrm{occ\_lin}(v, z\sigma)$ does not hold, so that, by Definition 6.7 and Lemma 6.18, $z \in \mathrm{nlvars}_0(\sigma) \subseteq \mathrm{nlvars}_1(\sigma)$. Otherwise, suppose $z = y$. Then we have $z\sigma = y\sigma = s$, $z\sigma' = y\sigma' = s[x/t]$, $v \in \mathrm{vars}\big(y\sigma'\big) \setminus \mathrm{gvars}(\sigma)$ and $\neg\, \mathrm{occ\_lin}\big(v, y\sigma'\big)$. We have four subcases.
   (a) Suppose $v \notin \mathrm{vars}(s)$ and $\mathrm{occ\_lin}(v, t)$ does not hold. Then $v \in \mathrm{vars}(t) \setminus \mathrm{gvars}(\sigma)$ and, since $t = x\sigma$, $x \in \mathrm{nlvars}_0(\sigma)$. By Definition 6.7, $z = y \in \mathrm{nlvars}_1(\sigma)$.

(b) Suppose $v \notin \mathrm{vars}(s)$ and $\mathrm{occ\_lin}(v,t)$ holds. Then, it must be the case that $\mathrm{occ\_lin}(x,s)$ does not hold. Also, since $v \in \mathrm{vars}(x\sigma)$ and $v \notin \mathrm{gvars}(\sigma)$, by Lemma 6.16 we have $x \notin \mathrm{gvars}(\sigma)$. Thus we have $x \in \mathrm{vars}(y\sigma) \setminus \mathrm{gvars}(\sigma)$, so that, by Definition 6.7, we obtain $z = y \in \mathrm{nlvars}_0(\sigma) \subseteq \mathrm{nlvars}_1(\sigma)$.

(c) Suppose now $v \in \mathrm{vars}(s)$ and $\mathrm{occ\_lin}(v,s)$ does not hold. Then, by Definition 6.7 and Lemma 6.18, $z = y \in \mathrm{nlvars}_0(\sigma) \subseteq \mathrm{nlvars}_1(\sigma)$.

(d) Finally, suppose $v \in \mathrm{vars}(s)$ and $\mathrm{occ\_lin}(v,s)$ holds. Then, it must be the case that $v \in \mathrm{vars}(t)$. If $v \neq x$ then, by Lemma 6.22, there exists $w \in \mathrm{vars}(\sigma)$ such that $\{v,x\} \subseteq \mathrm{occ}(\sigma,w)$. Therefore, by Definition 6.7 and Lemma 6.18, we have $z = y \in \mathrm{nlvars}_0(\sigma) \subseteq \mathrm{nlvars}_1(\sigma)$. Otherwise, let $v = x$, so that $x \notin \mathrm{gvars}(\sigma)$. Since $\mathrm{occ\_lin}(x,s)$ holds but $\mathrm{occ\_lin}\big(x, s[x/t]\big)$ does not hold, we have $x \in \mathrm{vars}(t)$ and also $\neg\,\mathrm{occ\_lin}(x,t)$. By Definition 6.7, $x \in \mathrm{nlvars}_0(\sigma)$ and $z = y \in \mathrm{nlvars}_1(\sigma)$.

2. Suppose now $z \in \mathrm{dom}(\sigma')$ and there exist $\{w_1,w_2\} \subseteq \mathrm{vars}(z\sigma')$, with $w_1 \neq w_2$, and $v \in \mathrm{vars}(\sigma')$ such that $\{w_1,w_2\} \subseteq \mathrm{occ}(\sigma',v)$. By Theorem 3.15, $z \in \mathrm{dom}(\sigma)$. Also, by Lemma 3.36, we have $\mathrm{occ}(\sigma,v) = \mathrm{occ}(\sigma',v)$. Now, if $z \neq y$ then we have $z\sigma = z\sigma'$, so that $\{w_1,w_2\} \subseteq \mathrm{vars}(z\sigma)$. Thus, by Definition 6.7 and Lemma 6.18, we obtain $z \in \mathrm{nlvars}_0(\sigma) \subseteq \mathrm{nlvars}_1(\sigma)$. Otherwise, let $z = y$, so that $z\sigma = y\sigma = s$ and $z\sigma' = y\sigma' = s[x/t]$. We have three subcases.

   (a) Suppose $\{w_1,w_2\} \subseteq \mathrm{vars}(s)$. Then, since $y\sigma = s$, by Definition 6.7 and Lemma 6.18, $z = y \in \mathrm{nlvars}_0(\sigma) \subseteq \mathrm{nlvars}_1(\sigma)$.

   (b) Suppose $\{w_1,w_2\} \cap \mathrm{vars}(s) = \varnothing$. Then $\{w_1,w_2\} \subseteq \mathrm{vars}(t)$. Since $(x \mapsto t) \in \sigma$ we have $x \in \mathrm{nlvars}_0(\sigma)$. By Definition 6.7, $z = y \in \mathrm{nlvars}_1(\sigma)$.

   (c) Finally, without loss of generality, suppose that $w_1 \in \mathrm{vars}(s) \setminus \mathrm{vars}(t)$ and $w_2 \in \mathrm{vars}(t) \setminus \mathrm{vars}(s)$. Then, we have $\{w_1,x\} \subseteq \mathrm{vars}(s)$ and, by Lemma 6.23, $\{w_1,x\} \subseteq \mathrm{occ}(\sigma,v)$. Note that $w_1 \neq x$. In fact, if $w_1 = x$ then we would have $w_1 \in \mathrm{vars}\big(s[x/t]\big)$ if and only if $w_1 \in \mathrm{vars}(t)$, therefore contradicting the hypothesis of this subcase. Thus, since $w_1 \neq x$, by Definition 6.7 and Lemma 6.18, $z = y \in \mathrm{nlvars}_0(\sigma) \subseteq \mathrm{nlvars}_1(\sigma)$.

For the inductive step, when $m > 0$, assume $\mathrm{nlvars}_{m-1}(\sigma') \subseteq \mathrm{nlvars}_m(\sigma)$ and let $z \in \mathrm{nlvars}_m(\sigma')$. By Definition 6.7 we have

$$z \in \mathrm{nlvars}_{m-1}(\sigma') \cup \big\{\, y \in \mathrm{dom}(\sigma') \bigm| \mathrm{vars}(y\sigma') \cap \mathrm{nlvars}_{m-1}(\sigma') \neq \varnothing \,\big\}.$$

We have two cases.

1. If $z \in \mathrm{nlvars}_{m-1}(\sigma')$ then, by the inductive hypothesis, $z \in \mathrm{nlvars}_m(\sigma)$. Hence, by Lemma 6.18, $z \in \mathrm{nlvars}_{m+1}(\sigma)$.

2. Otherwise, assume $z \in \mathrm{dom}(\sigma')$ and $\exists v \in \mathrm{vars}(z\sigma') \cap \mathrm{nlvars}_{m-1}(\sigma')$. By Theorem 3.15, $z \in \mathrm{dom}(\sigma)$. By the inductive hypothesis, we have $v \in \mathrm{nlvars}_m(\sigma)$. If

$z \neq y$, then $z\sigma = z\sigma'$, so that $v \in \mathrm{vars}(z\sigma)$ and, by Definition 6.7, $z \in \mathrm{nlvars}_{m+1}(\sigma)$. Suppose now $z = y$, so that $z\sigma = y\sigma = s$ and $z\sigma' = y\sigma' = s[x/t]$. We have two subcases.

(a) Suppose that $v \in \mathrm{vars}(t)$. Then, by Definition 6.7, we have $x \in \mathrm{nlvars}_m(\sigma)$ and $z = y \in \mathrm{nlvars}_{m+1}(\sigma)$.

(b) Suppose that $v \in \mathrm{vars}(s)$. Then, by Definition 6.7 and Lemma 6.18, we have $z = y \in \mathrm{nlvars}_m(\sigma) \subseteq \mathrm{nlvars}_{m+1}(\sigma)$.

□

Lemma 6.25 *Let $\sigma, \sigma' \in RSubst$, where $\sigma \overset{\mathcal{S}}{\longmapsto}{}^* \sigma'$. Then:*

$$\mathrm{fvars}(\sigma) = \mathrm{fvars}(\sigma'); \tag{6.21}$$

$$\mathrm{lvars}(\sigma) = \mathrm{lvars}(\sigma'). \tag{6.22}$$

Proof. If the derivation has length 0 there is nothing to prove. Suppose that $\sigma \overset{\mathcal{S}}{\longmapsto} \sigma'$. Then the equalities (6.21) and (6.22) follow from Lemmas 6.19 and 6.24, respectively.

Suppose now that $\sigma = \sigma_1 \overset{\mathcal{S}}{\longmapsto} \cdots \overset{\mathcal{S}}{\longmapsto} \sigma_n = \sigma'$, where $n > 1$. By the first part of the proof we have that, for each $i = 2, \ldots, n$, the equalities hold between σ_{i-1} and σ_i, and hence the required result. □

Proof of Proposition 6.10 on page 136. Note that, by Definition 6.5, the equivalence (6.3) follows from Proposition 3.30.

We now prove the other two equivalences. By Theorem 3.16, there exists $\tau \in VSubst$ such that $\sigma \overset{\mathcal{S}}{\longmapsto}{}^* \tau$. By Theorem 3.15, $\mathrm{dom}(\sigma) = \mathrm{dom}(\tau)$ and $T \vdash \forall(\sigma \leftrightarrow \tau)$. By Lemma 6.25, we have $\mathrm{fvars}(\sigma) = \mathrm{fvars}(\tau)$ and $\mathrm{lvars}(\sigma) = \mathrm{lvars}(\tau)$. From all of the above, by Proposition 6.17, we obtain

$$\begin{aligned} y \in \mathrm{fvars}(\sigma) \quad &\Longleftrightarrow \quad \mathrm{rt}(y, \tau) \in Vars, \\ y \in \mathrm{lvars}(\sigma) \quad &\Longleftrightarrow \quad \mathrm{rt}(y, \tau) \in LTerms. \end{aligned}$$

Therefore, in order to prove equivalences (6.2) and (6.4) it is sufficient to show that

$$\mathrm{rt}(y, \sigma) \in Vars \quad \Longleftrightarrow \quad \mathrm{rt}(y, \tau) \in Vars, \tag{6.23}$$

$$\mathrm{rt}(y, \sigma) \in LTerms \quad \Longleftrightarrow \quad \mathrm{rt}(y, \tau) \in LTerms. \tag{6.24}$$

Consider (6.23). We will prove only one implication, since the other one will follow by symmetry. Suppose $\mathrm{rt}(y, \sigma) \in Vars$. Then there exists an index $i \geq 0$ such that $y\sigma^i \in Vars \setminus \mathrm{dom}(\sigma)$. Thus, as $y \notin \mathrm{dom}(\sigma) = \mathrm{dom}(\tau)$, we have $\mathrm{rt}(y\sigma^i, \tau) \in Vars$. Since $T \vdash \forall(\tau \to \sigma)$, by Lemma 3.20 we have $T \vdash \forall\bigl(\tau \to (y = y\sigma^i)\bigr)$. By Lemma 3.39, we obtain $\mathrm{rt}(y, \tau) = \mathrm{rt}(y\sigma^i, \tau)$, so that $\mathrm{rt}(y, \tau) \in Vars$.

Consider now (6.24). Again, we will prove only one implication, since the other one will follow by symmetry. Reasoning by contraposition, suppose that $\mathrm{rt}(y, \tau) \notin LTerms$, so that there exists $v \in \mathrm{vars}\big(\mathrm{rt}(y, \tau)\big)$ such that $\mathrm{occ\_lin}\big(v, \mathrm{rt}(y, \tau)\big)$ does not hold. By definition of 'rt', there exists an index $i \geq 0$ such that $v \in \mathrm{vars}(y\tau^i)$ and $\mathrm{occ\_lin}(v, y\tau^i)$ does not hold. Thus, as $v \notin \mathrm{dom}(\tau) = \mathrm{dom}(\sigma)$, we obtain that $v \in \mathrm{vars}\big(\mathrm{rt}(y\tau^i, \sigma)\big)$ and $\mathrm{occ\_lin}\big(v, \mathrm{rt}(y\tau^i, \sigma)\big)$ does not hold, so that $\mathrm{rt}(y\tau^i, \sigma) \notin \mathit{LTerms}$. Since $T \vdash \forall(\sigma \rightarrow \tau)$, by Lemma 3.20 we have $T \vdash \forall\big(\sigma \rightarrow (y = y\tau^i)\big)$. By Lemma 3.39, $\mathrm{rt}(y, \sigma) = \mathrm{rt}(y\tau^i, \sigma)$, so that $\mathrm{rt}(y, \sigma) \notin \mathit{LTerms}$. □

Lemma 6.26 *Let $\sigma, \sigma' \in \mathit{RSubst}$ where $\sigma \overset{\mathcal{S}}{\longmapsto}{}^{*} \sigma'$. Then $\alpha_S(\sigma) = \alpha_S(\sigma')$.*

Proof. By Definition 6.11, this is an easy consequence of Lemmas 3.36 and 6.25. □

Corollary 6.27 *Let $\sigma \in \mathit{RSubst}$ be satisfiable in the equality theory T. There exists $\sigma' \in \mathit{VSubst}$ such that* $\mathrm{vars}(\sigma) = \mathrm{vars}(\sigma')$, $T \vdash \forall(\sigma \leftrightarrow \sigma')$, $y \in \mathrm{dom}(\sigma') \cap \mathrm{range}(\sigma')$ *implies* $y \in \mathrm{vars}(y\sigma')$ *and* $\alpha_S(\sigma) = \alpha_S(\sigma')$.

Proof. All but the last property follow by Corollary 3.17. The property $\alpha_S(\sigma) = \alpha_S(\sigma')$ follows by Lemma 6.26. □

Lemma 6.28 *Let $\sigma, \tau \in \mathit{VSubst}$ be satisfiable in the syntactic equality theory T and suppose $T \vdash \forall(\sigma \leftrightarrow \tau)$. Then* $\mathrm{fvars}(\sigma) = \mathrm{fvars}(\tau)$.

Proof. We prove $\mathrm{fvars}(\sigma) \subseteq \mathrm{fvars}(\tau)$, since the other inclusion follows by symmetry.

By contraposition, suppose there exists $y \in \mathrm{fvars}(\sigma) \setminus \mathrm{fvars}(\tau)$. Then, by Proposition 6.17, $\mathrm{rt}(y, \sigma) \in \mathit{Vars}$ and $\mathrm{rt}(y, \tau) \notin \mathit{Vars}$. Thus, there exists an index $i > 0$ such that $y\tau^i \notin \mathit{Vars}$ and, by definition of 'rt', $\mathrm{rt}(y\tau^i, \sigma) \notin \mathit{Vars}$. Since $T \vdash \forall(\sigma \rightarrow \tau)$, by Lemma 3.20 we have $T \vdash \forall\big(\sigma \rightarrow (y = y\tau^i)\big)$. By Lemma 3.39, $\mathrm{rt}(y, \sigma) = \mathrm{rt}(y\tau^i, \sigma)$, so that $\mathrm{rt}(y, \sigma) \notin \mathit{Vars}$, contradicting our previous assumption. □

Lemma 6.29 *Let $\sigma, \tau \in \mathit{RSubst}$ be satisfiable in the syntactic equality theory T and suppose that $T \vdash \forall(\sigma \leftrightarrow \tau)$. Then* $\mathrm{gvars}(\sigma) = \mathrm{gvars}(\tau)$.

Proof. By Definition 6.5, this is a simple consequence of Theorem 3.35. □

Lemma 6.30 *Let $\sigma, \tau \in \mathit{VSubst}$ be satisfiable in the syntactic equality theory T and suppose that $T \vdash \forall(\sigma \leftrightarrow \tau)$. Then* $\mathrm{lvars}(\sigma) = \mathrm{lvars}(\tau)$.

Proof. We prove $\mathrm{lvars}(\sigma) \subseteq \mathrm{lvars}(\tau)$, since the other inclusion follows by symmetry.

By contraposition, suppose there exists $y \in \mathrm{lvars}(\sigma) \setminus \mathrm{lvars}(\tau)$. Then, by Proposition 6.17, $\mathrm{rt}(y, \sigma) \in \mathit{LTerms}$ and $\mathrm{rt}(y, \tau) \notin \mathit{LTerms}$. Thus, there exists an index $i > 0$ such that

$$\exists v \in \mathrm{vars}(y\tau^i) \setminus \mathrm{dom}(\tau) \;.\; \neg\, \mathrm{occ\_lin}(v, y\tau^i). \tag{6.25}$$

Note that $v \notin \mathrm{gvars}(\tau)$ so that, by Lemma 6.29, $v \notin \mathrm{gvars}(\sigma)$. Therefore, by Proposition 6.15, there exists $w \in \mathrm{vars}(v\sigma) \setminus \mathrm{dom}(\sigma)$. From this, by (6.25), we obtain

$$\exists w \in \mathrm{vars}(y\tau^i\sigma) \setminus \mathrm{dom}(\sigma) \;.\; \neg\, \mathrm{occ\_lin}(w, y\tau^i\sigma). \tag{6.26}$$

By definition of 'rt', we have that (6.26) implies $\mathrm{rt}(y\tau^i, \sigma) \notin \mathit{LTerms}$. Since $T \vdash \forall(\sigma \to \tau)$, by Lemma 3.20 we have $T \vdash \forall\bigl(\sigma \to (y = y\tau^i)\bigr)$. By Lemma 3.39, $\mathrm{rt}(y, \sigma) = \mathrm{rt}(y\tau^i, \sigma)$, so that $\mathrm{rt}(y, \sigma) \notin \mathit{LTerms}$, contradicting our previous assumption. □

Proof of Theorem 6.12 on page 137. The equality result for the sharing components of the two abstract descriptions is a consequence of Theorem 3.35, so that we only have to prove the equality of the freeness and linearity components.

By Corollary 6.27, we can assume there exist $\tau, \tau' \in \mathit{VSubst}$ such that $T \vdash \forall(\sigma \leftrightarrow \tau)$, $T \vdash \forall(\sigma' \leftrightarrow \tau')$ and $\alpha_S(\sigma) = \alpha_S(\tau)$, $\alpha_S(\sigma') = \alpha_S(\tau')$. Since we have $T \vdash \forall(\tau \leftrightarrow \tau')$, the result then follows by Lemmas 6.28 and 6.30. □

6.4 The Abstract Operators on *SFL*

The specification of the abstract unification operator on the domain *SFL* is rather complex, since it is based on a very detailed case analysis. In the next definition we introduce several auxiliary abstract operators.

Definition 6.31 (Auxiliary operators on *SFL*.) *Let $s, t \in \mathit{HTerms}$ be such that $\mathrm{vars}(s) \cup \mathrm{vars}(t) \subseteq \mathit{VI}$. For each $d = \langle sh, f, l \rangle \in \mathit{SFL}$ we define the following predicates: s and t are* independent *in d if and only if $\mathrm{ind}_d \colon \mathit{HTerms}^2 \to \mathit{Bool}$ holds for (s, t), where*

$$\mathrm{ind}_d(s, t) \stackrel{\mathrm{def}}{=} \Bigl(\mathrm{rel}\bigl(\mathrm{vars}(s), sh\bigr) \cap \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr) = \varnothing\Bigr);$$

t is ground *in d if and only if $\mathrm{ground}_d \colon \mathit{HTerms} \to \mathit{Bool}$ holds for t, where*

$$\mathrm{ground}_d(t) \stackrel{\mathrm{def}}{=} \bigl(\mathrm{vars}(t) \subseteq \mathit{VI} \setminus \mathrm{vars}(sh)\bigr);$$

$y \in \mathrm{vars}(t)$ occurs linearly *(in t) in d if and only if $\mathrm{occ\_lin}_d \colon \mathit{VI} \times \mathit{HTerms} \to \mathit{Bool}$ holds for (y, t), where*

$$\begin{aligned}\mathrm{occ\_lin}_d(y, t) \stackrel{\mathrm{def}}{=} \mathrm{ground}_d(y) \lor \Bigl(&\mathrm{occ\_lin}(y, t) \land (y \in l) \\ &\land \forall z \in \mathrm{vars}(t) : \bigl(y \neq z \implies \mathrm{ind}_d(y, z)\bigr)\Bigr);\end{aligned}$$

t is free *in d if and only if $\mathrm{free}_d \colon \mathit{HTerms} \to \mathit{Bool}$ holds for t, where*

$$\mathrm{free}_d(t) \stackrel{\mathrm{def}}{=} \exists y \in \mathit{VI} \,.\, (y = t) \land (y \in f);$$

t is linear *in d if and only if $\mathrm{lin}_d \colon \mathit{HTerms} \to \mathit{Bool}$ holds for t, where*

$$\mathrm{lin}_d(t) \stackrel{\mathrm{def}}{=} \forall y \in \mathrm{vars}(t) : \mathrm{occ\_lin}_d(y, t).$$

The function $\mathrm{share\_with}_d \colon \mathit{HTerms} \to \wp(\mathit{VI})$ *yields the set of variables of interest that may share with the given term. For each* $t \in \mathit{HTerms}$,

$$\mathrm{share\_with}_d(t) \stackrel{\mathrm{def}}{=} \mathrm{vars}\Bigl(\mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr)\Bigr).$$

The function $\mathrm{cyclic}_x^t \colon \mathit{SH} \to \mathit{SH}$ *strengthens the sharing set* sh *by forcing the coupling of* x *with* t*. For each* $sh \in \mathit{SH}$ *and each* $(x \mapsto t) \in \mathit{Bind}$,

$$\mathrm{cyclic}_x^t(sh) \stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh\bigr) \cup \mathrm{rel}\bigl(\mathrm{vars}(t) \setminus \{x\}, sh\bigr).$$

These auxiliary operators correctly approximate the intended properties.

Theorem 6.32 *Let* $d \in \mathit{SFL}$ *and* $\sigma \in \gamma_S(d)$*. Let also* $y \in \mathit{VI}$ *and* $s, t \in \mathit{HTerms}$ *be such that* $\mathrm{vars}(s) \cup \mathrm{vars}(t) \subseteq \mathit{VI}$*. Then*

$$\mathrm{ind}_d(s,t) \implies \mathrm{vars}\bigl(\mathrm{rt}(s,\sigma)\bigr) \cap \mathrm{vars}\bigl(\mathrm{rt}(t,\sigma)\bigr) = \varnothing; \tag{6.27}$$

$$\mathrm{ind}_d(y,t) \iff y \notin \mathrm{share\_with}_d(t); \tag{6.28}$$

$$\mathrm{free}_d(t) \implies \mathrm{rt}(t,\sigma) \in \mathit{Vars}; \tag{6.29}$$

$$\mathrm{ground}_d(t) \implies \mathrm{rt}(t,\sigma) \in \mathit{GTerms}; \tag{6.30}$$

$$\mathrm{lin}_d(t) \implies \mathrm{rt}(t,\sigma) \in \mathit{LTerms}. \tag{6.31}$$

We now introduce the abstract mgu operator, specifying how a single binding affects each component of the domain *SFL* in the context of a syntactic equality theory T.

Definition 6.33 **(amgu_S.)** *The function* $\mathrm{amgu}_S \colon \mathit{SFL} \times \mathit{Bind} \to \mathit{SFL}$ *is defined as follows. Let* $d = \langle sh, f, l \rangle \in \mathit{SFL}$ *and* $(x \mapsto t) \in \mathit{Bind}$*, where* $\{x\} \cup \mathrm{vars}(t) \subseteq \mathit{VI}$*. Let also*

$$sh' \stackrel{\mathrm{def}}{=} \mathrm{cyclic}_x^t(sh_- \cup sh''),$$

where

$$sh_x \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\{x\}, sh\bigr), \qquad sh_- \stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh\bigr),$$

$$sh_t \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr), \qquad sh_{xt} \stackrel{\mathrm{def}}{=} sh_x \cap sh_t,$$

$$sh'' \stackrel{\mathrm{def}}{=} \begin{cases} \mathrm{bin}(sh_x, sh_t), & \text{if } \mathrm{free}_d(x) \lor \mathrm{free}_d(t); \\ \mathrm{bin}\bigl(sh_x \cup \mathrm{bin}(sh_x, sh_{xt}^\star), sh_t \cup \mathrm{bin}(sh_t, sh_{xt}^\star)\bigr), & \text{if } \mathrm{lin}_d(x) \land \mathrm{lin}_d(t); \\ \mathrm{bin}(sh_x^\star, sh_t), & \text{if } \mathrm{lin}_d(x); \\ \mathrm{bin}(sh_x, sh_t^\star), & \text{if } \mathrm{lin}_d(t); \\ \mathrm{bin}(sh_x^\star, sh_t^\star), & \text{otherwise.} \end{cases}$$

Letting $S_x \stackrel{\mathrm{def}}{=} \mathrm{share\_with}_d(x)$ *and* $S_t \stackrel{\mathrm{def}}{=} \mathrm{share\_with}_d(t)$, *we also define*

$$
\begin{aligned}
f' &\stackrel{\mathrm{def}}{=} \begin{cases} f, & \text{if } \mathrm{free}_d(x) \wedge \mathrm{free}_d(t); \\ f \setminus S_x, & \text{if } \mathrm{free}_d(x); \\ f \setminus S_t, & \text{if } \mathrm{free}_d(t); \\ f \setminus (S_x \cup S_t), & \text{otherwise;} \end{cases} \\
l' &\stackrel{\mathrm{def}}{=} \bigl(VI \setminus \mathrm{vars}(sh')\bigr) \cup f' \cup l'',
\end{aligned}
$$

where

$$
l'' \stackrel{\mathrm{def}}{=} \begin{cases} l \setminus (S_x \cap S_t), & \text{if } \mathrm{lin}_d(x) \wedge \mathrm{lin}_d(t); \\ l \setminus S_x, & \text{if } \mathrm{lin}_d(x); \\ l \setminus S_t, & \text{if } \mathrm{lin}_d(t); \\ l \setminus (S_x \cup S_t), & \text{otherwise.} \end{cases}
$$

Then

$$
\mathrm{amgu}_S\bigl(d, x \mapsto t\bigr) \stackrel{\mathrm{def}}{=} \begin{cases} \bot_S, & \text{if } d = \bot_S \vee \bigl(T = \mathcal{FT} \wedge x \in \mathrm{vars}(t)\bigr); \\ \langle sh', f', l' \rangle & \text{otherwise.} \end{cases}
$$

It is now time to highlight the similarities and differences of the operator amgu_S with respect to the corresponding ones defined in the "classical" proposals for an integration of set-sharing with freeness and linearity, such as [BCM94a, HW92, Lan90]. Note that, in order to obtain a comparison focused on the properties encoded by *SFL*, we will remove from the abstract operator specified in [BCM94a] all the enhancements depending on properties that cannot be represented on *SFL* (i.e., compoundness and explicit structural information).

- In the computation of the set-sharing component, the main difference can be observed in the second, third and fourth cases of the definition of sh'': here we omit one of the star-unions even when the terms x and t possibly share. In contrast, in [BCM94a, HW92, Lan90] the corresponding star-union is avoided only when $\mathrm{ind}_d(x,t)$ holds. Note that when $\mathrm{ind}_d(x,t)$ holds in the second case of sh'', then we have $sh_{xt} = \varnothing$; thus, the whole computation for this case reduces to $sh'' = \mathrm{bin}(sh_x, sh_t)$, as was the case in the previous proposals.

- Another improvement on the set-sharing component can be observed in the definition of sh': the cyclic_x^t operator allows the set-sharing description to be further enhanced when dealing with *definitely cyclic bindings*, i.e., when $x \in \mathrm{vars}(t)$. This is the rewording of a similar enhancement proposed in [Bag97a] for the domain *Pos* in the context of groundness analysis. Its net effect is to recover some groundness and sharing dependencies that would have been unnecessarily lost when using the standard operators. When $x \notin \mathrm{vars}(t)$, we have $\mathrm{cyclic}_x^t(sh_- \cup sh'') = sh_- \cup sh''$.

- The computation of the freeness component f' is the same as specified in [BCM94a, HW92], and is more precise than the one defined in [Lan90].
- The computation of the linearity component l' is the same as specified in [BCM94a], and is more precise than those defined in [HW92, Lan90].

In the following examples we show that the improvements in the abstract computation of the sharing component allow, in particular cases, the derivation of information that is more precise than that obtainable when using the classical abstract unification operators.

Example 6.34 *Let $VI = \{x, x_1, x_2, y, y_1, y_2, z\}$ and $\sigma \in RSubst$ be such that*

$$\sigma \stackrel{\mathrm{def}}{=} \bigl\{x \mapsto f(x_1, x_2, z), y \mapsto f(y_1, z, y_2)\bigr\}.$$

By Definition 6.11, we have $d \stackrel{\mathrm{def}}{=} \alpha_S\bigl(\{\sigma\}\bigr) = \langle sh, f, l\rangle$, where

$$\begin{aligned} sh &= \{xx_1, xx_2, xyz, yy_1, yy_2\}, \\ f &= VI \setminus \{x, y\}, \\ l &= VI. \end{aligned}$$

Consider the binding $(x \mapsto y) \in Bind$. On the concrete domain, we compute (a substitution equivalent to) $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = y\}\bigr)$, where

$$\tau = \bigl\{x \mapsto f(y_1, y_2, y_2), y \mapsto f(y_1, y_2, y_2), x_1 \mapsto y_1, x_2 \mapsto y_2, z \mapsto y_2\bigr\}.$$

Note that $\alpha_S\bigl(\{\tau\}\bigr) = \langle sh_\tau, f_\tau, l_\tau\rangle$, where $sh_\tau = \{xx_1yy_1, xx_2yy_2z\}$, so that the pairs of variables $P_x = \{x_1, x_2\}$ and $P_y = \{y_1, y_2\}$ keep their independence.

When abstractly evaluating the binding, both $\mathrm{lin}_d(x)$ and $\mathrm{lin}_d(y)$ hold so that we apply the second case of the definition of sh''. By using the notation of Definition 6.33, we have

$$\begin{aligned} sh_x &= \{xx_1, xx_2, xyz\}, & sh_- &= \varnothing, \\ sh_t &= \{yy_1, yy_2, xyz\}, & sh_{xt} &= \{xyz\}. \end{aligned}$$

Since we compute the star-closure of sh_{xt} only, we obtain the set-sharing component

$$sh' = \{xx_1yy_1, xx_1yy_2, xx_1yz, xx_2yy_1, xx_2yy_2, xx_2yz, xyy_1z, xyy_2z, xyz\}.$$

Thus, we precisely capture the fact that pairs P_x and P_y keep their independence.

In contrast, since $\mathrm{ind}_d(x, y)$ does not hold, all of the classical definitions of abstract unification would have required the star-closure of both sh_x and sh_t, resulting in an abstract element including, among others, the sharing group $S = \{x, x_1, x_2, y, y_1, y_2\}$. Since both $P_x \subset S$ and $P_y \subset S$, this independence information would have been unnecessarily lost.

Similar examples can be devised for the third and fourth cases of the definition of sh'', where only one side of the binding is known to be linear.

Example 6.34 has another interesting, unexpected consequence. By repeating the above abstract computation on the domain ASub (e.g., using the abstract semantics operators specified in [Kin00]), we discover that even this simpler domain precisely captures the independence of pairs P_x and P_y. Therefore, the example provides a formal proof that all the classical approaches based on set-sharing are not *uniformly* more precise than the pair-sharing domain ASub. Such a property is enjoyed by our combination *SFL* with the improved abstract unification operator.

The next example shows the precision improvements arising from the use of the cyclic_x^t operator.

Example 6.35 *Let* $VI = \{x, x_1, x_2, y\}$ *and* $\sigma \stackrel{\mathrm{def}}{=} \bigl\{x \mapsto f(x_1, x_2)\bigr\}$. *By Definition 6.11, we have* $d \stackrel{\mathrm{def}}{=} \alpha_S\bigl(\{\sigma\}\bigr) = \langle sh, f, l\rangle$, *where*

$$\begin{aligned} sh &= \{xx_1, xx_2, y\},\\ f &= VI \setminus \{x\},\\ l &= VI. \end{aligned}$$

Let $t = f(x, y)$ *and consider the cyclic binding* $(x \mapsto t) \in Bind$. *In the concrete, we compute (a substitution equivalent to)* $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$, *where*

$$\tau = \bigl\{x \mapsto f(x_1, x_2), x_1 \mapsto f(x_1, x_2), y \mapsto x_2, \bigr\}.$$

Note that if we further instantiate τ *by grounding* y, *then variables* x, x_1 *and* x_2 *would become ground too. Formally,* $\alpha_S\bigl(\{\tau\}\bigr) = \langle sh_\tau, f_\tau, l_\tau\rangle$, *where* $sh_\tau = \{xyx_1x_2\}$. *Thus, as observed above,* y *covers* x, x_1 *and* x_2.

When abstractly evaluating the binding, we compute

$$\begin{aligned} sh_x &= \{xx_1, xx_2\},\\ sh_t &= \{xx_1, xx_2, y\},\\ sh_{xt} &= sh_x,\\ sh_- \cup sh'' &= \{xx_1, xx_1x_2, xx_1x_2y, xx_1y, xx_2, xx_2y\},\\ sh' &= \mathrm{cyclic}_x^t(sh_- \cup sh'')\\ &= \{xx_1x_2y, xx_1y, xx_2y\}. \end{aligned}$$

Note that, in the element $sh_- \cup sh''$ *(which is the abstract element that would have been computed when not exploiting the* cyclic_x^t *operator) variable* y *covers none of variables* x, x_1 *and* x_2. *Thus, by applying the* cyclic_x^t *operator, this covering information is restored.*

The next result states that the abstract mgu operator is a correct approximation of the concrete one. As already discussed, this result applies to any syntactic equality theory, including both the finite-tree theory $\mathcal{FT}$ and the rational-tree theory $\mathcal{RT}$. In contrast, all the published proofs of correctness for a combination of set-sharing with freeness and linearity information assume the occurs-check is performed.

Theorem 6.36 *Let $d \in \mathit{SFL}$ and $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \operatorname{vars}(t) \subseteq \mathit{VI}$. Then, for all $\sigma \in \gamma_S(d)$ and $\tau \in \operatorname{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ in the syntactic equality theory T, we have*

$$\tau \in \gamma_S\bigl(\operatorname{amgu}_S(d, x \mapsto t)\bigr).$$

The full abstract unification operator aunify_S, capturing the effect of a sequence of bindings on an abstract element, can now be specified by a straightforward inductive definition using the operator amgu_S.

Definition 6.37 (aunify_S.) *The operator $\operatorname{aunify}_S \colon \mathit{SFL} \times \mathit{Bind}^\star \to \mathit{SFL}$ is defined, for each $d \in \mathit{SFL}$ and each sequence of bindings $bs \in \mathit{Bind}^\star$, by*

$$\operatorname{aunify}_S(d, bs) \stackrel{\mathrm{def}}{=} \begin{cases} d, & \text{if } bs = \epsilon; \\ \operatorname{aunify}_S\bigl(\operatorname{amgu}_S(d, x \mapsto t), bs'\bigr), & \text{if } bs = (x \mapsto t) \mathbin{.} bs'. \end{cases}$$

It is worth stressing that the second argument of aunify_S is a *sequence* of bindings and not a substitution (which is a *set* of bindings). This difference comes from the fact that amgu_S is neither commutative nor idempotent, so that the multiplicity and the actual order of application of the bindings can influence the overall result of the abstract computation.

Note that the correctness of the aunify_S operator is simply inherited from the correctness of the underlying amgu_S operator. In particular, an arbitrary reordering of the sequence bs of bindings still result in a correct implementation of aunify_S. It is known since [Lan90, pp. 66-67] that more precise results are obtained if aunify_S is implemented so that the *grounding bindings* are considered before the others. A binding $(x \mapsto t) \in \mathit{Bind}$ is grounding in the context of an abstract description $d \in \mathit{SFL}$ if and only if one side of the binding is known to be definitely ground; namely, we have $\operatorname{share\_with}_d(x) = \varnothing$ or $\operatorname{share\_with}_d(t) = \varnothing$. In our implementation of aunify_S, we always reorder the sequences of bindings so that the grounding ones come first.

The 'merge-over-all-path' operator on the domain SFL is provided by alub_S and is correct by definition. Finally, we define the abstract existential quantification operator.

Definition 6.38 ($\operatorname{aexists}_S$.) *The function $\operatorname{aexists}_S \colon \mathit{SFL} \times \wp_{\mathrm{f}}(\mathit{VI}) \to \mathit{SFL}$ provides the* existential quantification *of an abstract element with respect to a subset of the variables of interest. For each $d \stackrel{\mathrm{def}}{=} \langle sh, f, l \rangle \in \mathit{SFL}$ and $V \subseteq \mathit{VI}$,*

$$\operatorname{aexists}_S\bigl(\langle sh, f, l\rangle, V\bigr) \stackrel{\mathrm{def}}{=} \bigl\langle \operatorname{aexists}(sh, V), f \cup V, l \cup V \bigr\rangle.$$

The abstract projection operator for SFL, which actually modifies the set VI of variables of interest, can be easily defined using $\operatorname{aexists}_S$, as done in Definition 3.61 for the plain set-sharing domain SS. The statement and proof of correctness for this operator are omitted for space reasons.

6.5 Proof of the Correctness of amgu_S

In this section we prove that the amgu_S operator is a correct approximation of the concrete unification procedure. We start by proving Theorem 6.32.

Proof of Theorem 6.32 on page 150. Let $d = \langle sh, f, l \rangle$, $V_s = \mathrm{vars}(s)$ and $V_t = \mathrm{vars}(t)$. By Definition 6.11, we have $\forall v \in \mathit{Vars} : \mathrm{occ}(\sigma, v) \cap \mathit{VI} \in sh \cup \{\varnothing\}$, $f \subseteq \mathrm{fvars}(\sigma)$, and $l \subseteq \mathrm{lvars}(\sigma)$.

Consider the implication (6.27). By Definition 6.31, the hypothesis and Proposition 3.30, we have

$$
\begin{aligned}
\mathrm{ind}_d(s,t) &\iff \mathrm{rel}(V_s, sh) \cap \mathrm{rel}(V_t, sh) = \varnothing \\
&\iff \forall S \in sh, w_1 \in V_s, w_2 \in V_t : \{w_1, w_2\} \nsubseteq S \\
&\implies \forall v \in \mathit{Vars}, w_1 \in V_s, w_2 \in V_t : \{w_1, w_2\} \nsubseteq \mathrm{occ}(\sigma, v) \\
&\iff \forall w_1 \in V_s, w_2 \in V_t : \mathrm{vars}\bigl(\mathrm{rt}(w_1, \sigma)\bigr) \cap \mathrm{vars}\bigl(\mathrm{rt}(w_1, \sigma)\bigr) = \varnothing \\
&\iff \mathrm{vars}\bigl(\mathrm{rt}(s, \sigma)\bigr) \cap \mathrm{vars}\bigl(\mathrm{rt}(t, \sigma)\bigr) = \varnothing.
\end{aligned}
$$

Consider the equivalence (6.28). By Definition 6.31, we have

$$
\begin{aligned}
\mathrm{ind}_d(y,t) &\iff \mathrm{rel}\bigl(\{y\}, sh\bigr) \cap \mathrm{rel}(V_t, sh) = \varnothing \\
&\iff \forall S \in \mathrm{rel}(V_t, sh) : y \notin S \\
&\iff y \notin \mathrm{share\_with}_d(t).
\end{aligned}
$$

Consider now the implication (6.29). By Definition 6.31, the hypothesis and Proposition 6.10, we have

$$
\begin{aligned}
\mathrm{free}_d(t) &\iff t \in f \\
&\implies t \in \mathrm{fvars}(\sigma) \\
&\iff \mathrm{rt}(t, \sigma) \in \mathit{Vars}.
\end{aligned}
$$

Consider now the implication (6.30). By Definition 6.31, the hypothesis and Proposition 6.10, we have

$$
\begin{aligned}
\mathrm{ground}_d(t) &\iff \mathrm{vars}(t) \subseteq \mathit{VI} \setminus \mathrm{vars}(sh) \\
&\implies \forall w \in V_t : w \in \mathrm{gvars}(\sigma) \\
&\iff \forall w \in V_t : \mathrm{vars}\bigl(\mathrm{rt}(w, \sigma)\bigr) = \varnothing \\
&\iff \mathrm{rt}(t, \sigma) \in \mathit{GTerms}.
\end{aligned}
$$

Finally consider the implication (6.31). By Definition 6.31, the hypothesis, the above results and Proposition 6.10, we have

$$\begin{aligned}
\mathrm{lin}_d(t) &\iff \forall y, z \in \mathrm{vars}(t) : \mathrm{ground}_d(y) \\
&\qquad \lor \Bigl((y \in l) \land \mathrm{occ_lin}(y, t) \land \bigl(y \neq z \implies \mathrm{ind}_d(y, z)\bigr)\Bigr) \\
&\implies \forall y, z \in \mathrm{vars}(t) : \mathrm{ground}_d(y) \\
&\qquad \lor \Bigl(\bigl(y \in \mathrm{lvars}(\sigma)\bigr) \land \mathrm{occ_lin}(y, t) \land \bigl(y \neq z \implies \mathrm{ind}_d(y, z)\bigr)\Bigr) \\
&\implies \forall y, z \in \mathrm{vars}(t) : \mathrm{rt}(y, \sigma) \in \mathit{GTerms} \\
&\qquad \lor \Bigl(\bigl(\mathrm{rt}(y, \sigma) \in \mathit{LTerms}\bigr) \land \mathrm{occ_lin}(y, t) \\
&\qquad\qquad \land \bigl(y \neq z \implies \mathrm{vars}(\mathrm{rt}(y, \sigma)) \cap \mathrm{vars}(\mathrm{rt}(z, \sigma)) = \varnothing\bigr)\Bigr) \\
&\iff \mathrm{rt}(t, \sigma) \in \mathit{LTerms}.
\end{aligned}$$

□

The following lemma will be systematically used in the following correctness proofs.

Lemma 6.39 *Let T be an equality theory and $\sigma \in \mathit{RSubst}$. Then, for each $s, t \in \mathit{HTerms}$,*

$$\mathrm{mgs}\bigl(\sigma \cup \{s = t\}\bigr) = \mathrm{mgs}\bigl(\sigma \cup \{s = t\sigma\}\bigr).$$

Proof. By congruence axioms (2.6) and (2.7), for any terms $p, q, r \in \mathit{HTerms}$,

$$T \vdash \forall (p = q \land q = r) \leftrightarrow \forall (p = r \land q = r).$$

Also, by Lemma 3.20, for all $\tau \in \mathit{RSubst}$ and $r \in \mathit{HTerms}$, $T \vdash \forall\bigl(\tau \to (r = r\tau)\bigr)$, so that

$$T \vdash \forall\bigl(\tau \leftrightarrow \tau \cup \{r = r\tau\}\bigr).$$

Using these results,

$$\begin{aligned}
&T \vdash \forall\bigl(\sigma \cup \{s = t\} \leftrightarrow \sigma \cup \{s = t, t = t\sigma\}\bigr), \\
&T \vdash \forall\bigl(\sigma \cup \{s = t\} \leftrightarrow \sigma \cup \{s = t\sigma, t = t\sigma\}\bigr), \\
&T \vdash \forall\bigl(\sigma \cup \{s = t\} \leftrightarrow \sigma \cup \{s = t\sigma\}\bigr).
\end{aligned}$$

The thesis follows by the definition of mgs. □

The following Lemma, which will be used several times in the following proofs without an explicit reference to it, states the well-known result that groundness is closed by entailment.

Lemma 6.40 *Let $\sigma, \tau \in \mathit{RSubst}$ be satisfiable in the syntactic equality theory T and such that $T \vdash \forall(\tau \to \sigma)$. Then* $\mathrm{gvars}(\sigma) \subseteq \mathrm{gvars}(\tau)$.

Proof. We prove the result by showing that $x \notin \mathrm{gvars}(\tau)$ implies $x \notin \mathrm{gvars}(\sigma)$.

By Corollary 6.27, we can assume there exist $\sigma', \tau' \in \mathit{VSubst}$ such that $T \vdash \forall(\sigma \leftrightarrow \sigma')$ and $T \vdash \forall(\tau \leftrightarrow \tau')$, so that $T \vdash \forall(\tau' \rightarrow \sigma')$. Also, by Lemma 6.29, $\mathrm{gvars}(\sigma) = \mathrm{gvars}(\sigma')$ and $\mathrm{gvars}(\tau) = \mathrm{gvars}(\tau')$. Therefore, it is sufficient to prove that $x \notin \mathrm{gvars}(\tau')$ implies $x \notin \mathrm{gvars}(\sigma')$.

Assume $x \notin \mathrm{gvars}(\tau')$. By Definition 6.5, there exists $v \in \mathit{Vars}$ such that $x \in \mathrm{occ}(\tau', v)$. By Lemma 3.28, $v \in \mathrm{vars}(x\tau') \setminus \mathrm{dom}(\tau')$. Also, by Lemma 3.20, $T \vdash \forall(\tau' \rightarrow x\tau' = x)$. Thus, by Lemma 3.12 (taking $s = x\tau'$ and $t = x$) there exists $z \in \mathrm{vars}(x\sigma') \setminus \mathrm{dom}(\sigma')$ such that $v \in \mathrm{vars}(z\tau')$. By Definition 3.26, we have $x \in \mathrm{occ}(\sigma', z)$ so that, by Definition 6.5, $x \notin \mathrm{gvars}(\sigma')$. $\square$

Another useful result is the following consequence of Theorem 6.12.

Lemma 6.41 *Let $e \subseteq \mathit{Eqs}$ be satisfiable and $\sigma, \tau \in \mathrm{mgs}(e)$, in the context of the syntactic equality theory T. Then* $\mathrm{ssets}(\sigma) = \mathrm{ssets}(\tau)$, $\mathrm{fvars}(\sigma) = \mathrm{fvars}(\tau)$, $\mathrm{gvars}(\sigma) = \mathrm{gvars}(\tau)$ *and* $\mathrm{lvars}(\sigma) = \mathrm{lvars}(\tau)$.

Proof. By definition of mgs, we have $\sigma, \tau \in \mathit{RSubst}$ and $\sigma \iff e \iff \tau$. Thus, the result $\mathrm{gvars}(\sigma) = \mathrm{gvars}(\tau)$ follows by applying twice Lemma 6.40. The other equivalences follows by Theorem 6.12. $\square$

We now introduce a bit of terminology that will be helpful in order to simplify the notation in the following proofs.

Given $V \subseteq \mathit{Vars}$, we say that $t \in \mathit{HTerms}$ is V-linear if $\mathrm{occ\_lin}(v, t)$ holds for all variables $v \in \mathrm{vars}(t) \cap V$. Note that if a term is V-linear, then it is also W-linear, for all $W \subseteq V$. This terminology also applies to n-tuples of terms, by simply regarding the n-tuple construction as a term functor of arity n. Moreover, if $\bar{s}, \bar{t} \in \mathit{HTerms}^n$ are such that $\mathrm{mgs}(\bar{s} = \bar{t}) \neq \varnothing$, then we write $\mathrm{gvars}(\bar{s} = \bar{t})$ to denote the set $\mathrm{gvars}(\mu)$, where $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$. Note that, by Lemma 6.41, this notation is not ambiguous.

Lemma 6.42 *Let $\bar{s}, \bar{t} \in \mathit{HTerms}^n$ be such that $\mathrm{mgs}(\bar{s} = \bar{t}) \neq \varnothing$ and $G \stackrel{\mathrm{def}}{=} \mathrm{gvars}(\bar{s} = \bar{t})$. If $\bar{s}$ is $(\mathit{Vars} \setminus G)$-linear, then there exists $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$ such that, for each $z \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s})$,*

1. *$z\mu$ is $(\mathit{Vars} \setminus G)$-linear;*

2. *$\mathrm{vars}(z\mu) \cap \mathrm{dom}(\mu) \subseteq G$;*

3. *$\forall z' \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s}) : z \neq z' \implies \mathrm{vars}(z\mu) \cap \mathrm{vars}(z'\mu) \subseteq G$.*

Proof. We assume that the congruence and identity axioms hold.

Let $\bar{s} = (s_1, \ldots, s_n)$, and $\bar{t} = (t_1, \ldots, t_n)$, $W = \mathrm{vars}(\bar{s}) \cup \mathrm{vars}(\bar{t})$ and $V = \mathit{Vars} \setminus G$. We assume that $\bar{s}$ is V-linear and prove that the result holds by induction on the number of variables in W.

Suppose first that, for some $i = 1, \ldots, n$, $s_i = f(r_1, \ldots, r_m)$ and $t_i = f(u_1, \ldots, u_m)$, where $m \geq 0$. Let

$$\bar{s}_i \stackrel{\text{def}}{=} (s_1, \ldots, s_{i-1}, r_1, \ldots, r_m, s_{i+1}, \ldots, s_n),$$
$$\bar{t}_i \stackrel{\text{def}}{=} (t_1, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n).$$

Then $\mathrm{mvars}(\bar{s}_i) = \mathrm{mvars}(\bar{s})$ and $\mathrm{mvars}(\bar{t}_i) = \mathrm{mvars}(\bar{t})$ so that, since $\bar{s}$ is V-linear, $\bar{s}_i$ is V-linear. Moreover, by the congruence axiom (2.8), we have $\mathrm{mgs}(\bar{s}_i = \bar{t}_i) = \mathrm{mgs}(\bar{s} = \bar{t})$. Note that in the case that s_i and t_i are identical constants, the equation $s_i = t_i$ is just removed. Thus, as $\bar{s}$ and $\bar{t}$ are finite sequences of finite terms, we can assume that $s_i \in \mathit{Vars}$ or $t_i \in \mathit{Vars}$, for all $i = 1, \ldots, n$.

Secondly, suppose that for some $i = 1, \ldots, n$, $s_i = t_i$. By the previous paragraph, we can assume that $s_i \in \mathit{Vars}$. Let

$$\bar{s}_i \stackrel{\text{def}}{=} (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n),$$
$$\bar{t}_i \stackrel{\text{def}}{=} (t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n).$$

Then $\mathrm{mvars}(\bar{s}_i) \cup \{s_i\} = \mathrm{mvars}(\bar{s})$ and $\mathrm{mvars}(\bar{t}_i) \cup \{s_i\} = \mathrm{mvars}(\bar{t})$ so that, as $\bar{s}$ is V-linear, $\bar{s}_i$ is V-linear. Furthermore, by the congruence axiom (2.5), $\mathrm{mgs}(\bar{s}_i = \bar{t}_i) = \mathrm{mgs}(\bar{s} = \bar{t})$. As $\bar{s}$ and $\bar{t}$ are sequences of finite length n, we can assume that $s_i \neq t_i$, for all $i = 1, \ldots, n$.

Therefore, for the rest of the proof, we will assume that $s_i \neq t_i$ and $s_i \in \mathit{Vars}$ or $t_i \in \mathit{Vars}$, for all $i = 1, \ldots, n$.

The base case is when $W = \varnothing$, so that we have $\mathrm{vars}(\mu) = \varnothing$ for all $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$. Thus all three properties hold trivially.

To prove the inductive step, we assume that $W \neq \varnothing$, so that $n > 0$. Note that, in the case that $\mathrm{vars}(\bar{t}) \subseteq \mathrm{vars}(\bar{s})$, then all three properties hold trivially. This is because for all $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$ and for all $z \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s})$, we have $z \notin \mathrm{vars}(\mu)$. Similarly, the three properties hold trivially whenever $\mathrm{vars}(\bar{t}) \subseteq G$. This is because, if $z \in \mathrm{dom}(\mu)$, then $\mathrm{vars}(z\mu) \subseteq G$. Therefore we can also assume that $\mathrm{vars}(t_i) \setminus \bigl(\mathrm{vars}(\bar{s}) \cup G\bigr) \neq \varnothing$, for some $i = 1, \ldots, n$. As the order of equations is irrelevant, without loss of generality we assume that this property holds when $i = 1$, so that $\mathrm{vars}(t_1) \setminus \bigl(\mathrm{vars}(\bar{s}) \cup G\bigr) \neq \varnothing$. This can be rewritten as

$$\mathrm{vars}(t_1) \cap \bigl(V \setminus \mathrm{vars}(\bar{s})\bigr) \neq \varnothing. \tag{6.32}$$

Note that this implies that $t_1 \neq s_1$. By Proposition 6.10, another consequence of the above assumption is that, for all $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$, we have $\mathrm{rt}(t_1, \mu) \notin \mathit{GTerms}$. Since $\mu \implies \{s_1 = t_1\}$, by Lemma 3.39, we obtain $\mathrm{rt}(s_1, \mu) \notin \mathit{GTerms}$, so that again by Proposition 6.10, $\mathrm{vars}(s_1) \setminus G \neq \varnothing$. This in turn can be rewritten as

$$\mathrm{vars}(s_1) \cap V \neq \varnothing. \tag{6.33}$$

By exploiting (6.32) and (6.33), we can identify three different cases:

a. for all $i = 1, \ldots, n$, $V \cap \mathrm{vars}(s_i) \cap \mathrm{vars}(t_i) \neq \varnothing$;

b. $s_1 \in V \setminus \mathrm{vars}(t_1)$;

c. $t_1 \in V \setminus \mathrm{vars}(\bar{s})$ and $s_1 \notin \mathit{Vars}$;

Case a. For all $i = 1, \ldots, n$, $V \cap \mathrm{vars}(s_i) \cap \mathrm{vars}(t_i) \neq \varnothing$.

For each $i = 1, \ldots, n$, we are assuming that $s_i \in V$ or $t_i \in V$. Therefore, for each $i = 1, \ldots, n$, $s_i \in \mathrm{vars}(t_i)$ or $t_i \in \mathrm{vars}(s_i)$ so that, without loss of generality, we can assume, for some k where $0 \leq k \leq n$, $s_i \in V$ if $1 \leq i \leq k$ and $t_i \in V$ if $k+1 \leq i \leq n$.

Let $\mu \subseteq \mathit{Eqs}$ be defined as

$$\mu \stackrel{\text{def}}{=} \{s_1 = t_1, \ldots, s_k = t_k\} \cup \{t_{k+1} = s_{k+1}, \ldots, t_n = s_n\}.$$

We show that $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$. First we must show that $\mu \in \mathit{RSubst}$. As $\bar{s}$ is V-linear, $(s_1, \ldots, s_k)$ is linear; $(t_{k+1}, \ldots, t_n)$ is also linear, because $\bar{s}$ is V-linear and $t_i \in V \cap \mathrm{vars}(s_i)$ if $k+1 \leq i \leq n$; moreover, for the same reasons, $\{s_1, \ldots, s_k\} \cap \{t_{k+1}, \ldots, t_n\} = \varnothing$. As we are assuming that, for all $i = 1, \ldots, n$, $s_i \neq t_i$ and $V \cap \mathrm{vars}(s_i) \cap \mathrm{vars}(t_i) \neq \varnothing$, it follows that $t_i \notin \mathit{Vars}$ when $1 \leq i \leq k$ and $s_i \notin \mathit{Vars}$ when $k+1 \leq i \leq n$, so that each equation in μ is a binding and μ has no circular subsets. Thus $\mu \in \mathit{RSubst}$ and hence, by the congruence axiom (2.6), $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$.

As $\{t_{k+1}, \ldots, t_n\} \subseteq \mathrm{vars}\bigl((s_{k+1}, \ldots, s_n)\bigr)$, we have $\mathrm{dom}(\mu) \setminus \mathrm{vars}(\bar{s}) = \varnothing$ so that the required result holds trivially.

Case b. Suppose $s_1 \in V \setminus \mathrm{vars}(t_1)$ Let

$$\begin{aligned} \bar{s}_1 &\stackrel{\text{def}}{=} (s_2, \ldots, s_n), \\ \bar{t}_1 &\stackrel{\text{def}}{=} \bigl(t_2[s_1/t_1], \ldots, t_n[s_1/t_1]\bigr). \end{aligned}$$

As $\bar{s}$ is V-linear, $\bar{s}_1$ is V-linear and $s_1 \notin \mathrm{vars}(\bar{s}_1)$. Also, all occurrences of s_1 in $\bar{t}$ are replaced in $\bar{t}_1$ by t_1 so that, as $s_1 \notin \mathrm{vars}(t_1)$ (by the assumption for this case), $s_1 \notin \mathrm{vars}(\bar{t}_1)$. Thus,

$$s_1 \notin W_1 \stackrel{\text{def}}{=} \mathrm{vars}(\bar{s}_1) \cup \mathrm{vars}(\bar{t}_1), \tag{6.34}$$

so that $W_1 \subset W$. Let $G_1 \stackrel{\text{def}}{=} \mathrm{gvars}(\bar{s}_1 = \bar{t}_1)$ and $V_1 \stackrel{\text{def}}{=} \mathit{Vars} \setminus G_1$. Note that $G_1 \subseteq G$ and, by the assumption for this case, $s_1 \in V$, so that $s_1 \notin G$. As a consequence, $G_1 = G$, $V_1 = V$ and $\bar{s}_1$ is V_1-linear, so that the inductive hypothesis applies to $\bar{s}_1$ and $\bar{t}_1$. Thus, there exists $\mu_1 \in \mathrm{mgs}(\bar{s}_1 = \bar{t}_1)$ such that, for each $z \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s}_1)$, the three inductive properties hold.

Let $\mu \subseteq \mathit{Eqs}$ be defined as

$$\mu \stackrel{\text{def}}{=} \{s_1 = t_1\mu_1\} \cup \mu_1.$$

We show that $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$. By (6.34), we have $s_1 \notin \mathrm{vars}(\mu_1)$ so that $s_1 \notin \mathrm{dom}(\mu_1)$. Also, since $\mu_1 \in \mathit{RSubst}$, μ has no identities or circular subsets. Thus we have $\mu \in \mathit{RSubst}$. By Lemma 6.39, $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$.

Suppose that $z \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s})$. Then, as $\mathrm{vars}(\bar{s}) = \mathrm{vars}(\bar{s}_1) \cup \{s_1\}$, $z \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s}_1)$. Thus, the inductive properties 1, 2 and 3 using μ_1 and $\bar{s}_1$ can be applied to $z\mu_1$. Knowing this, we now show that the same properties using μ and $\bar{s}$ can be applied to $z\mu$. Since $\mathrm{dom}(\mu) = \mathrm{dom}(\mu_1) \cup \{s_1\}$ and $z \neq s_1$, we have $z\mu_1 = z\mu$ and $s_1 \notin \mathrm{vars}(z\mu)$. Each property is proved separately.

1. By the inductive property 1, we have that $z\mu_1$ is V_1-linear. As $z\mu = z\mu_1$ and $V = V_1$, $z\mu$ is V-linear.

2. By the inductive property 2, we have that $\mathrm{vars}(z\mu_1) \cap \mathrm{dom}(\mu_1) \subseteq G_1$. Since $z\mu = z\mu_1$, $G = G_1$ and we have $\mathrm{dom}(\mu) = \mathrm{dom}(\mu_1) \cup \{s_1\}$, where $s_1 \notin \mathrm{vars}(z\mu)$, we obtain $\mathrm{vars}(z\mu) \cap \mathrm{dom}(\mu) \subseteq G$.

3. Let $z' \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s})$ be such that $z \neq z'$. Since $z' \notin \mathrm{vars}(\bar{s})$, we have $z' \notin \mathrm{vars}(\bar{s}_1)$ and $z'\mu = z'\mu_1$. By applying inductive property 3, $\mathrm{vars}(z\mu_1) \cap \mathrm{vars}(z'\mu_1) \subseteq G_1$. As $z\mu = z\mu_1$, $z'\mu = z'\mu_1$ and $G = G_1$, we obtain $\mathrm{vars}(z\mu) \cap \mathrm{vars}(z'\mu) \subseteq G$.

Case c. Assume that $t_1 \in V \setminus \mathrm{vars}(\bar{s})$ and $s_1 \notin \mathit{Vars}$. Let

$$\begin{aligned} \bar{s}_1 &\stackrel{\mathrm{def}}{=} (s_2, \ldots, s_n), \\ \bar{t}_1 &\stackrel{\mathrm{def}}{=} \bigl(t_2[t_1/s_1], \ldots, t_n[t_1/s_1]\bigr). \end{aligned}$$

As $\bar{s}$ is V-linear, $\bar{s}_1$ is V-linear. Also, since by the assumption for this case $t_1 \notin \mathrm{vars}(\bar{s})$, we have $t_1 \notin \mathrm{vars}(\bar{s}_1)$. Moreover, all occurrences of t_1 in $\bar{t}$ are replaced in $\bar{t}_1$ by s_1 so that $t_1 \notin \mathrm{vars}(\bar{t}_1)$. Thus

$$t_1 \notin W_1 \stackrel{\mathrm{def}}{=} \mathrm{vars}(\bar{s}_1) \cup \mathrm{vars}(\bar{t}_1), \tag{6.35}$$

so that $W_1 \subset W$. Let $G_1 \stackrel{\mathrm{def}}{=} \mathrm{gvars}(\bar{s}_1 = \bar{t}_1)$ and $V_1 \stackrel{\mathrm{def}}{=} \mathit{Vars} \setminus G_1$. Note that $G_1 \subseteq G$ and, by the assumption for this case, $t_1 \in V$, so that $t_1 \notin G$. As a consequence, $G_1 = G$, $V_1 = V$ and $\bar{s}_1$ is V_1-linear, so that the inductive hypothesis applies to $\bar{s}_1$ and $\bar{t}_1$. Thus, there exists $\mu_1 \in \mathrm{mgs}(\bar{s}_1 = \bar{t}_1)$ such that, for each $z \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s_1})$, the three inductive properties hold.

Let $\mu \subseteq \mathit{Eqs}$ be defined as

$$\mu \stackrel{\mathrm{def}}{=} \{t_1 = s_1\mu_1\} \cup \mu_1.$$

Note that, by (6.35), $t_1 \notin \mathrm{vars}(\mu_1)$ and, in particular, $t_1 \notin \mathrm{dom}(\mu_1)$. Since $\mu_1 \in \mathit{RSubst}$, μ has no identities or circular subsets so that $\mu \in \mathit{RSubst}$. By Lemma 6.39, $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$.

Suppose that $z \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s})$. Then either $z \neq t_1$, so that $z\mu = z\mu_1$, or $z = t_1$, so that $z\mu = s_1\mu_1$. We show in each case that $z\mu$ satisfies the three required properties.

1. Suppose $z \neq t_1$. By the inductive property 1, $z\mu_1$ is V_1-linear. As $z\mu = z\mu_1$ and $V = V_1$, $z\mu$ is V-linear.

   Otherwise, let $z = t_1$, so that $z\mu = s_1\mu_1$. Consider an arbitrary variable $u \in \mathrm{vars}(s_1)$. Then $u \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s}_1)$ and the inductive properties using μ_1 and $\bar{s}_1$ can be applied to $u\mu_1$. Therefore, by property 1, $u\mu_1$ is V_1-linear. Moreover, by property 3, we have

$$\forall u' \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s}_1) : u \neq u' \implies \mathrm{vars}(u\mu_1) \cap \mathrm{vars}(u'\mu_1) \subseteq G_1.$$

In particular, this holds for all $u' \in \mathrm{vars}(s_1)$ such that $u \neq u'$. As a consequence, $z\mu = s_1\mu_1$ is V_1-linear. As $V = V_1$, $z\mu$ is V-linear.

2. Suppose $z \neq t_1$. By the inductive property 2, we have $\mathrm{vars}(z\mu_1) \cap \mathrm{dom}(\mu_1) \subseteq G_1$. Since $z\mu = z\mu_1$, $G = G_1$, $\mathrm{dom}(\mu) = \mathrm{dom}(\mu_1) \cup \{t_1\}$ and $t_1 \notin \mathrm{vars}(z\mu)$, we obtain $\mathrm{vars}(z\mu) \cap \mathrm{dom}(\mu) \subseteq G$.

   Otherwise, let $z = t_1$ so that $z\mu = s_1\mu_1$. Considering $u \in \mathrm{vars}(s_1)$, we have that $u \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s}_1)$, so that the inductive properties using μ_1 and $\bar{s}_1$ can be applied to $u\mu_1$. By property 2, $\mathrm{vars}(u\mu_1) \cap \mathrm{dom}(\mu_1) \subseteq G_1$. As this holds for all $u \in \mathrm{vars}(s_1)$, we have $\mathrm{vars}(s_1\mu_1) \cap \mathrm{dom}(\mu_1) \subseteq G_1$. As $z\mu = s_1\mu_1$, $G = G_1$, $\mathrm{dom}(\mu) = \mathrm{dom}(\mu_1) \cup \{t_1\}$ and $t_1 \notin \mathrm{vars}(z\mu)$, we obtain $\mathrm{vars}(z\mu) \cap \mathrm{dom}(\mu) \subseteq G$.

3. Suppose $z \neq t_1$ and let $z' \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s})$ be such that $z \neq z'$. Then, by inductive property 3, we have $\mathrm{vars}(z\mu_1) \cap \mathrm{vars}(z'\mu_1) \subseteq G_1$. Since $z\mu = z\mu_1$ and $G = G_1$, if also $z' \neq t_1$ (so that $z'\mu = z'\mu_1$) we obtain $\mathrm{vars}(z\mu) \cap \mathrm{vars}(z'\mu) \subseteq G$. Otherwise, let $z' = t_1$ (so that $z'\mu = s_1\mu_1$). We will show that

   $$\forall u \in \mathrm{vars}(s_1) : \mathrm{vars}(z\mu_1) \cap \mathrm{vars}(u\mu_1) \subseteq G_1. \tag{6.36}$$

   In fact, in the case that $u \in G_1$ then $\mathrm{vars}(u\mu_1) \subseteq G_1$. On the other hand, if $u \in \mathrm{vars}(s_1) \setminus G_1$, then we have $u \neq z$. As $\bar{s}$ is V-linear, $u \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s}_1)$ so that the property holds by inductive property 3 (taking $z' = u$). As (6.36) holds, we have $\mathrm{vars}(z\mu_1) \cap \mathrm{vars}(s_1\mu_1) \subseteq G_1$. Thus, by observing that $z\mu = z\mu_1$, $z'\mu = s_1\mu_1$ and $G = G_1$, we can conclude $\mathrm{vars}(z\mu) \cap \mathrm{vars}(z'\mu) \subseteq G$.

   Otherwise, let $z = t_1$ so that $z\mu = s_1\mu_1$. Let $z' \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s})$ be such that $z \neq z'$ (note that this implies $z' \neq t_1$, so that $z'\mu = z'\mu_1$). We will prove that, for all $u \in \mathrm{vars}(s_1)$,

   $$\mathrm{vars}(u\mu_1) \cap \mathrm{vars}(z'\mu_1) \subseteq G_1. \tag{6.37}$$

   In fact, if $u \in G_1$ then $\mathrm{vars}(u\mu_1) \subseteq G_1$. Suppose now $u \in \mathrm{vars}(s_1) \setminus G_1$. As $\bar{s}$ is V-linear, $u \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s}_1)$ so that the inductive property 3 can be applied to $u\mu_1$. Thus, for all $u' \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s}_1)$, if $u \neq u'$ we have $\mathrm{vars}(u\mu_1) \cap \mathrm{vars}(u'\mu_1) \subseteq G_1$. In particular, since $u \in \mathrm{vars}(s_1)$ and $z' \notin \mathrm{vars}(\bar{s})$, we have $u \neq z'$ so that, by taking $u' = z'$, we obtain (6.37). As the choice of $u \in \mathrm{vars}(s_1)$ is arbitrary, we have $\mathrm{vars}(s_1\mu_1) \cap \mathrm{vars}(z'\mu_1) \subseteq G_1$. By observing that $z\mu = s_1\mu_1$, $z'\mu = z'\mu_1$ and $G = G_1$ we obtain $\mathrm{vars}(z\mu) \cap \mathrm{vars}(z'\mu) \subseteq G$.

□

Corollary 6.43 *There exists $\mu' \in \mathit{VSubst}$ that (under the same hypotheses) satisfies all the properties stated for $\mu \in \mathit{RSubst}$ in Lemma 6.42.*

Proof. We start by proving that the properties stated for $\mu \in \mathit{RSubst}$ in Lemma 6.42 are invariant under the application of an $\mathcal{S}$-step.

Suppose that $\mu \in \mathit{RSubst}$ satisfies the properties stated in Lemma 6.42 and $\mu \overset{\mathcal{S}}{\longmapsto} \mu'$. First note that, by Theorem 3.15, we have $\mu' \in \mathrm{mgs}(\bar{s} = \bar{t})$ and $\mathrm{dom}(\mu') = \mathrm{dom}(\mu)$. Also, by Lemma 6.20, $\mathrm{gvars}(\mu') = \mathrm{gvars}(\mu) = G$. By definition of $\mathcal{S}$-step, there exist $\{x \mapsto t, y \mapsto s\} \subseteq \mu$ such that $x \neq y$ and

$$\mu' \stackrel{\mathrm{def}}{=} \bigl(\mu \setminus \{y \mapsto s\}\bigr) \cup \{y \mapsto s[x/t]\}.$$

Let $z \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s})$ and consider the term $z\mu'$. If $z \neq y$ or $x \notin \mathrm{vars}(s)$ then we have $z\mu' = z\mu$ and there is nothing to prove. Therefore, assume $z = y$, so that $z\mu = s$, and $x \in \mathrm{vars}(z\mu)$, so that $z\mu' = z\mu[x/t]$. Note that $x \in \mathrm{vars}(z\mu) \cap \mathrm{dom}(\mu)$ so that, by property 2, we have $x \in G$. As a consequence, $\mathrm{vars}(t) \subseteq G$ and we obtain

$$\mathrm{vars}(z\mu) \setminus G = \mathrm{vars}(z\mu') \setminus G.$$

From this, it is easy to conclude that properties 1, 2 and 3 hold for μ'.

By a simple induction, the above result generalizes to any finite sequence of $\mathcal{S}$-steps. Then, by Theorem 3.16 it follows than we can construct such a $\mu' \in \mathit{VSubst}$. □

In the following three sections, we prove the correctness of the abstract unification operator on each component of the *SFL* domain. A further section will join all of these results to establish the whole correctness of amgu_S.

6.5.1 The Correctness for Set-Sharing

Proposition 6.44 *Let $d = \langle sh, f, l\rangle \in \mathit{SFL}$, $\sigma \in \gamma_S(d) \cap \mathit{VSubst}$ and $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \mathrm{vars}(t) \subseteq \mathit{VI}$ and $y \in \mathrm{dom}(\sigma) \cap \mathrm{range}(\sigma)$ implies $y \in \mathrm{vars}(y\sigma)$. Suppose that $\{r, r'\} = \{x, t\}$ and $\mathrm{free}_d(r)$ holds. For all $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ in the syntactic equality theory T, letting*

$$\begin{aligned}
sh_- &= \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh\bigr),\\
sh_r &= \mathrm{rel}\bigl(\mathrm{vars}(r), sh\bigr),\\
sh_{r'} &= \mathrm{rel}\bigl(\mathrm{vars}(r'), sh\bigr),
\end{aligned}$$

we have

$$sh_- \cup \mathrm{bin}(sh_r, sh_{r'}) \supseteq \mathrm{ssets}(\tau). \tag{6.38}$$

Proof. We assume that the congruence and identity axioms hold. Note that if $\sigma \cup \{x = t\}$ is not satisfiable, then the result is trivial. We therefore assume, for the rest of the proof, that $\sigma \cup \{x = t\}$ is satisfiable in T. It follows from Lemma 6.41 that we just have to show that there exists $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ such that (6.38) holds.

Since $\mathrm{free}_d(r)$ holds, by Theorem 6.32, $\mathrm{rt}(r,\sigma) \in \mathit{Vars}$ and hence, by Proposition 6.15,

$$r\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma). \tag{6.39}$$

Let

$$\begin{aligned} R_{-} &= \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), \mathrm{ssets}(\sigma)\bigr), \\ R_r &= \mathrm{rel}\bigl(\mathrm{vars}(r), \mathrm{ssets}(\sigma)\bigr), \\ R_{r'} &= \mathrm{rel}\bigl(\mathrm{vars}(r'), \mathrm{ssets}(\sigma)\bigr). \end{aligned}$$

Since $\sigma \in \gamma_S(d)$, we have $sh \supseteq \mathrm{ssets}(\sigma)$ so that, using the monotonicity of $\overline{\mathrm{rel}}$, rel and bin, we obtain

$$sh_{-} \cup \mathrm{bin}(sh_r, sh_{r'}) \supseteq R_{-} \cup \mathrm{bin}(R_r, R_{r'}).$$

Thus, in order to prove (6.38) it is sufficient to show that

$$R_{-} \cup \mathrm{bin}(R_r, R_{r'}) \supseteq \mathrm{ssets}(\tau). \tag{6.40}$$

Note that, by Definition 3.26 and (6.39), we obtain

$$R_r = \bigl\{\mathrm{occ}(\sigma, r\sigma)\bigr\}. \tag{6.41}$$

Suppose first $x\sigma = t\sigma$. Then we have $\sigma \in \mathrm{mgs}\bigl(\sigma \cup \{x\sigma = t\sigma\}\bigr)$, so that by Lemma 6.39, $\sigma \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$. Thus, take $\tau \stackrel{\mathrm{def}}{=} \sigma$. Moreover, by (6.41), $R_{r'} = \bigl\{\mathrm{occ}(\sigma, r\sigma)\bigr\} = R_r$, so that $R_r = \mathrm{bin}(R_r, R_{r'})$. As a consequence,

$$\begin{aligned} R_{-} \cup \mathrm{bin}(R_r, R_{r'}) &= \bigl(\mathrm{ssets}(\sigma) \setminus R_r\bigr) \cup R_r \\ &= \mathrm{ssets}(\sigma) \\ &= \mathrm{ssets}(\tau). \end{aligned}$$

Otherwise, let $x\sigma \neq t\sigma$ and let $\nu, \mu, \tau \in \mathit{RSubst}$ be defined as

$$\begin{aligned} \nu &\stackrel{\mathrm{def}}{=} \bigl\{\, (y \mapsto s) \in \sigma \bigm| y \notin \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma) \,\bigr\}, \\ \mu &\stackrel{\mathrm{def}}{=} \{r\sigma \mapsto r'\sigma\}, \\ \tau &\stackrel{\mathrm{def}}{=} \nu \circ \mu. \end{aligned}$$

As $\nu \subseteq \sigma \in \mathit{VSubst}$, by Lemma 3.10 we have $\nu \in \mathit{VSubst}$; also, $\mu \in \mathit{VSubst}$ because it has a single binding; moreover, by construction, $\mathrm{dom}(\nu) \cap \mathrm{vars}(\mu) = \varnothing$; thus, by Lemma 3.18 we obtain $\tau \in \mathit{VSubst}$. By Lemma 6.39, we also have $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$.

Suppose $S \in \mathrm{ssets}(\tau)$. By Definition 3.43 and Theorem 3.44, we have

$$S \in R_{-} \cup \mathrm{bin}(R_r^{\star}, R_{r'}^{\star}).$$

If $S \in R_-$, then (6.40) holds trivially. Therefore suppose $S \in \operatorname{bin}(R_r^\star, R_{r'}^\star)$, so that there exist $S_r \in R_r^\star$ and $S_{r'} \in R_{r'}^\star$ such that $S = S_r \cup S_{r'}$. Note that, by (6.41), $R_r^\star = R_r$. Thus, to prove (6.40) holds, it is sufficient to show that $S_{r'} \in R_{r'}$.

As $S \in \operatorname{ssets}(\tau)$, by Lemma 3.28, there exists a variable $v \in \mathit{Vars} \setminus \operatorname{dom}(\tau)$ such that $S = \{\, y \in \mathit{VI} \mid v \in \operatorname{vars}(y\tau) \,\}$. As $v \notin \operatorname{dom}(\tau)$, $v \neq r\sigma$.

Let $y \in S$. We show that $y \in \operatorname{occ}(\sigma, r\sigma) \cup \operatorname{occ}(\sigma, v)$. Using Lemma 3.20, we have $T \vdash \forall(\tau \to y\tau = y)$. Therefore, since $T \vdash \forall(\tau \to \sigma)$, by Lemma 3.12 (replacing $s = t$ by $y\tau = y$), there exists $z \in \operatorname{vars}(y\sigma) \setminus \operatorname{dom}(\sigma)$ such that $v \in \operatorname{vars}(z\tau)$. If $z = r\sigma$, then $y \in S_r$. If $z \neq r\sigma$, then $z \notin \operatorname{dom}(\tau)$ and $z\tau = z$. Therefore $v = z$ and $y \in \operatorname{occ}(\sigma, v)$. □

Proposition 6.45 *Let $d = \langle sh, f, l\rangle \in \mathit{SFL}$, $\sigma \in \gamma_S(d) \cap \mathit{VSubst}$ and $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \operatorname{vars}(t) \subseteq \mathit{VI}$ and $y \in \operatorname{dom}(\sigma) \cap \operatorname{range}(\sigma)$ implies $y \in \operatorname{vars}(y\sigma)$. Suppose that $\operatorname{lin}_d(x)$ and $\operatorname{lin}_d(t)$ hold. For all $\tau \in \operatorname{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ in the syntactic equality theory T, letting*

$$sh_x = \operatorname{rel}\bigl(\{x\}, sh\bigr), \qquad sh_- = \overline{\operatorname{rel}}\bigl(\{x\} \cup \operatorname{vars}(t), sh\bigr),$$
$$sh_t = \operatorname{rel}\bigl(\operatorname{vars}(t), sh\bigr), \qquad sh_{xt} = sh_x \cap sh_t,$$

we have

$$sh_- \cup \operatorname{bin}\bigl(sh_x \cup \operatorname{bin}(sh_x, sh_{xt}^\star), sh_t \cup \operatorname{bin}(sh_t, sh_{xt}^\star)\bigr) \supseteq \operatorname{ssets}(\tau). \tag{6.42}$$

Proof. We assume that the congruence and identity axioms hold. Note that if $\sigma \cup \{x = t\}$ is not satisfiable, then the result is trivial. We therefore assume, for the rest of the proof, that $\sigma \cup \{x = t\}$ is satisfiable in T. It follows from Lemma 6.41 that we just have to show that there exists $\tau \in \operatorname{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ such that (6.42) holds.

Since, by hypothesis, $\operatorname{lin}_d(r)$ holds for each $r \in \{x, t\}$, by Theorem 6.32 we have that $\operatorname{rt}(r, \sigma) \in \mathit{LTerms}$ and hence, by Proposition 6.10, $\operatorname{vars}(r) \subseteq \operatorname{lvars}(\sigma)$. Thus, by Proposition 6.15,

$$\forall v \in \operatorname{vars}(r\sigma) \setminus \operatorname{dom}(\sigma) : \operatorname{occ\_lin}(v, r\sigma),$$
$$\operatorname{vars}(r\sigma) \cap \operatorname{dom}(\sigma) \subseteq \operatorname{gvars}(\sigma). \tag{6.43}$$

By defining $V_\sigma \stackrel{\text{def}}{=} \mathit{Vars} \setminus \operatorname{gvars}(\sigma)$, we obtain that both terms $x\sigma$ and $t\sigma$ are V_σ-linear. Let

$$\{u_1, \ldots, u_k\} \stackrel{\text{def}}{=} \operatorname{dom}(\sigma) \cap \bigl(\operatorname{vars}(x\sigma) \cup \operatorname{vars}(t\sigma)\bigr),$$
$$\bar{s} \stackrel{\text{def}}{=} (u_1, \ldots, u_k, x\sigma),$$
$$\bar{t} \stackrel{\text{def}}{=} (u_1\sigma, \ldots, u_k\sigma, t\sigma).$$

Since $x\sigma$ is V_σ-linear, it follows from (6.43) (letting $r = x$) that $\bar{s}$ is V_σ-linear. It also follows from (6.43) (applied twice, once with $r = x$ and once with $r = t$) that, for each $i = 1, \ldots, k$ we have $u_i \in \operatorname{gvars}(\sigma)$, so that $\operatorname{vars}(u_i\sigma) \subseteq \operatorname{gvars}(\sigma)$. Therefore, since $t\sigma$ is V_σ-linear, $\bar{t}$ is also V_σ-linear. By applying Lemma 6.39 and the congruence axioms,

$\sigma \cup \{x = t\} \implies \bar{s} = \bar{t}$. Thus, as $\sigma \cup \{x = t\}$ is satisfiable, there exists $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$. Let $V_\mu = \mathit{Vars} \setminus \mathrm{gvars}(\mu)$; since $\mathrm{gvars}(\sigma) \subseteq \mathrm{gvars}(\mu)$, then $V_\mu \subseteq V_\sigma$ and $\bar{s}$, $\bar{t}$ are also V_μ-linear. Therefore, we can apply Lemma 6.42 and Corollary 6.43 so that, by case (3), there exists $\mu \in \mathrm{mgs}(\bar{s} = \bar{t}) \cap \mathit{VSubst}$ such that, for all $w, w' \in \mathit{Vars}$ where $w \neq w'$ and either $\{w, w'\} \cap \mathrm{vars}(\bar{s}) = \varnothing$ or $\{w, w'\} \cap \mathrm{vars}(\bar{t}) = \varnothing$,

$$\mathrm{vars}(w\mu) \cap \mathrm{vars}(w'\mu) \subseteq \mathrm{gvars}(\mu). \tag{6.44}$$

Since $\sigma \in \mathit{VSubst}$, we have $\mathrm{vars}(u_i\sigma) \subseteq \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)$ for each $i = 1, \ldots, k$. Therefore

$$\mathrm{vars}(\mu) \subseteq \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma). \tag{6.45}$$

Let $\nu, \tau \in \mathit{RSubst}$ be defined as

$$\begin{aligned} \nu &\stackrel{\mathrm{def}}{=} \bigl\{ (y \mapsto s) \in \sigma \bigm| y \notin \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma) \bigr\}, \\ \tau &\stackrel{\mathrm{def}}{=} \nu \circ \mu. \end{aligned}$$

As $\nu \subseteq \sigma \in \mathit{VSubst}$, by Lemma 3.10 we have $\nu \in \mathit{VSubst}$; moreover, by (6.45), we have $\mathrm{dom}(\nu) \cap \mathrm{vars}(\mu) = \varnothing$; thus we can apply Lemma 3.18 to obtain $\tau \in \mathit{VSubst}$. By applying Lemma 6.39, we also have $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$.

Let

$$\begin{aligned} R_x &= \mathrm{rel}\bigl(\{x\}, \mathrm{ssets}(\sigma)\bigr), & R_- &= \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), \mathrm{ssets}(\sigma)\bigr), \\ R_t &= \mathrm{rel}\bigl(\mathrm{vars}(t), \mathrm{ssets}(\sigma)\bigr), & R_{xt} &= R_x \cap R_t. \end{aligned}$$

Since $\sigma \in \gamma_S(d)$, we have $sh \supseteq \mathrm{ssets}(\sigma)$ so that, using the monotonicity of $\overline{\mathrm{rel}}$, rel, $(\cdot)^\star$ and bin, we obtain

$$\begin{aligned} &sh_- \cup \mathrm{bin}\bigl(sh_x \cup \mathrm{bin}(sh_x, sh_{xt}^\star), sh_t \cup \mathrm{bin}(sh_t, sh_{xt}^\star)\bigr) \\ &\qquad \supseteq R_- \cup \mathrm{bin}\bigl(R_x \cup \mathrm{bin}(R_x, R_{xt}^\star), R_t \cup \mathrm{bin}(R_t, R_{xt}^\star)\bigr). \end{aligned}$$

It follows that, in order to prove (6.42), it is sufficient to show

$$R_- \cup \mathrm{bin}\bigl(R_x \cup \mathrm{bin}(R_x, R_{xt}^\star), R_t \cup \mathrm{bin}(R_t, R_{xt}^\star)\bigr) \supseteq \mathrm{ssets}(\tau). \tag{6.46}$$

Let S be an arbitrary sharing set in $\mathrm{ssets}(\tau)$. By Definition 3.43 and Theorem 3.44,

$$S \in R_- \cup \mathrm{bin}(R_x^\star, R_t^\star).$$

If $S \in R_-$, then (6.46) holds trivially. Therefore suppose $S \in \mathrm{bin}(R_x^\star, R_t^\star)$, so that there exist $S_x \in R_x^\star$ and $S_t \in R_t^\star$ such that $S = S_x \cup S_t$. We prove that (6.46) holds by showing that $S_x \in R_x \cup \mathrm{bin}(R_x, R_{xt}^\star)$ and $S_t \in R_t \cup \mathrm{bin}(R_t, R_{xt}^\star)$.

We first show that $S_t \in R_t \cup \mathrm{bin}(R_t, R_{xt}^\star)$. As $S_t \in R_t^\star$, we have $S_t = S_1 \cup S_2$ where

$S_1 \in (R_t \setminus R_{xt})^\star \cup \{\varnothing\}$ and $S_2 \in R_{xt}^\star \cup \{\varnothing\}$. Note that as $S_t \neq \varnothing$, we cannot have $S_1 = S_2 = \varnothing$. Suppose first that $S_1 = \varnothing$ so that $S_t = S_2 \neq \varnothing$. Then $S_t \in R_{xt}^\star$. However, since $R_{xt} \subseteq R_t$, $R_{xt}^\star \subseteq \operatorname{bin}(R_t, R_{xt}^\star)$. Thus $S_t \in \operatorname{bin}(R_t, R_{xt}^\star)$. Suppose next that $S_1 \neq \varnothing$. As $R_t \setminus R_{xt} = R_t \setminus R_x$, we have $S_1 = \bigcup\{\operatorname{occ}(\sigma, w) \mid w \in S_1 \setminus \operatorname{vars}(x\sigma)\}$. However, as $\operatorname{occ}(\sigma, w) = \varnothing$ for all $w \in \operatorname{dom}(\sigma)$ and $\operatorname{vars}(\bar{s}) \setminus \operatorname{vars}(x\sigma) \subseteq \operatorname{dom}(\sigma)$,

$$S_1 = \bigcup\Big\{\operatorname{occ}(\sigma, w) \;\Big|\; w \in S_1 \setminus \big(\operatorname{dom}(\sigma) \cup \operatorname{vars}(\bar{s})\big)\Big\}$$

Let $w_1, w_2 \in S_1 \setminus \big(\operatorname{dom}(\sigma) \cup \operatorname{vars}(\bar{s})\big)$. Then, as $S_1 \subseteq S$, $S \in \operatorname{ssets}(\tau)$ and $\tau \in \mathit{VSubst}$, by Lemma 3.28, there exists $v \in \mathit{Vars} \setminus \operatorname{dom}(\tau)$ such that $v \in \operatorname{vars}(w_1\tau) \cap \operatorname{vars}(w_2\tau)$. However, since $w_i \notin \operatorname{dom}(\sigma)$, we have $w_i\tau = w_i\mu$, for $i \in \{1, 2\}$. Thus, noting that $v \notin \operatorname{dom}(\tau)$ implies $v \notin \operatorname{gvars}(\tau)$, we can apply (6.44) to conclude that $w_1 = w_2$. As the choice of w_1 and w_2 was arbitrary, there exists a unique variable $w \in S_1 \setminus \big(\operatorname{dom}(\sigma) \cup \operatorname{vars}(\bar{s})\big)$ such that $S_1 = \operatorname{occ}(\sigma, w)$. Thus $S_1 \in R_t$. If $S_2 = \varnothing$ then $S_t = S_1 \in R_t$. If $S_2 \neq \varnothing$, then $S_t = S_1 \cup S_2 \in \operatorname{bin}(R_t, R_{xt}^\star)$. Hence, in both cases, $S_t \in R_t \cup \operatorname{bin}(R_t, R_{xt}^\star)$.

By the same reasoning, (replacing x by t, t by x and $\bar{s}$ by $\bar{t}$ in the previous paragraph) we obtain $S_x \in R_x \cup \operatorname{bin}(R_x, R_{xt}^\star)$, thus completing the proof. □

Corollary 6.46 *Let $d = \langle sh, f, l\rangle \in \mathit{SFL}$, $\sigma \in \gamma_S(d) \cap \mathit{VSubst}$ and $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \operatorname{vars}(t) \subseteq \mathit{VI}$ and $y \in \operatorname{dom}(\sigma) \cap \operatorname{range}(\sigma)$ implies $y \in \operatorname{vars}(y\sigma)$. Suppose that $\operatorname{lin}_d(x)$, $\operatorname{lin}_d(t)$ and $\operatorname{ind}_d(x, t)$ hold. For all $\tau \in \operatorname{mgs}\big(\sigma \cup \{x = t\}\big)$ in the syntactic equality theory T, letting*

$$\begin{aligned} sh_- &= \overline{\operatorname{rel}}\big(\{x\} \cup \operatorname{vars}(t), sh\big), \\ sh_x &= \operatorname{rel}\big(\{x\}, sh\big), \\ sh_t &= \operatorname{rel}\big(\operatorname{vars}(t), sh\big) \end{aligned}$$

we have

$$sh_- \cup \operatorname{bin}(sh_x, sh_t) \supseteq \operatorname{ssets}(\tau).$$

Proof. Since $\operatorname{ind}_d(x, t)$ holds, by Definition 6.31 we have $sh_x \cap sh_t = \varnothing$. The result then follows from Proposition 6.45. □

Proposition 6.47 *Let $d = \langle sh, f, l\rangle \in \mathit{SFL}$, $\sigma \in \gamma_S(d) \cap \mathit{VSubst}$ and $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \operatorname{vars}(t) \subseteq \mathit{VI}$ and $y \in \operatorname{dom}(\sigma) \cap \operatorname{range}(\sigma)$ implies $y \in \operatorname{vars}(y\sigma)$. Suppose that $\{r, r'\} = \{x, t\}$ and $\operatorname{lin}_d(r)$ holds. For all $\tau \in \operatorname{mgs}\big(\sigma \cup \{x = t\}\big)$ in the syntactic equality theory T, letting*

$$\begin{aligned} sh_- &= \overline{\operatorname{rel}}\big(\{x\} \cup \operatorname{vars}(t), sh\big), \\ sh_r &= \operatorname{rel}\big(\operatorname{vars}(r), sh\big), \\ sh_{r'} &= \operatorname{rel}\big(\operatorname{vars}(r'), sh\big), \end{aligned}$$

we have

$$sh_- \cup \operatorname{bin}(sh_r^\star, sh_{r'}) \supseteq \operatorname{ssets}(\tau). \tag{6.47}$$

Proof. We assume that the congruence and identity axioms hold. Note that if $\sigma \cup \{x = t\}$ is not satisfiable, then the result is trivial. We therefore assume, for the rest of the proof, that $\sigma \cup \{x = t\}$ is satisfiable in T. It follows from Lemma 6.41 that we just have to show that there exists $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ such that (6.47) holds.

Since $\mathrm{lin}_d(r)$ holds, by Theorem 6.32, $\mathrm{rt}(r, \sigma) \in \mathit{LTerms}$ and hence, by Proposition 6.10, $\mathrm{vars}(r) \subseteq \mathrm{lvars}(\sigma)$. Thus, by Proposition 6.15,

$$\begin{aligned} &\forall v \in \mathrm{vars}(r\sigma) \setminus \mathrm{dom}(\sigma) : \mathrm{occ\_lin}(v, r\sigma), \\ &\mathrm{vars}(r\sigma) \cap \mathrm{dom}(\sigma) \subseteq \mathrm{gvars}(\sigma). \end{aligned} \tag{6.48}$$

Therefore, by defining $V_\sigma \stackrel{\mathrm{def}}{=} \mathit{Vars} \setminus \mathrm{gvars}(\sigma)$, we obtain that the term $r\sigma$ is V_σ-linear. Let

$$\begin{aligned} \{u_1, \ldots, u_k\} &\stackrel{\mathrm{def}}{=} \mathrm{dom}(\sigma) \cap \bigl(\mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)\bigr), \\ \bar{s} &\stackrel{\mathrm{def}}{=} (u_1, \ldots, u_k, r\sigma), \\ \bar{t} &\stackrel{\mathrm{def}}{=} (u_1\sigma, \ldots, u_k\sigma, r'\sigma). \end{aligned}$$

Since $r\sigma$ is V_σ-linear it follows from (6.48) that $\bar{s}$ is V_σ-linear. By Lemma 6.39 and the congruence axioms, $\sigma \cup \{x = t\} \implies \bar{s} = \bar{t}$. Thus, as $\sigma \cup \{x = t\}$ is satisfiable, there exists $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$. Let $V_\mu = \mathit{Vars} \setminus \mathrm{gvars}(\mu)$; since $\mathrm{gvars}(\sigma) \subseteq \mathrm{gvars}(\mu)$, then $V_\mu \subseteq V_\sigma$ and $\bar{s}$ is also V_μ-linear. Therefore, we can apply Lemma 6.42 and Corollary 6.43 so that, by case (3), there exists $\mu \in \mathrm{mgs}(\bar{s} = \bar{t}) \cap \mathit{VSubst}$ such that, for all $w, w' \in \mathit{Vars} \setminus \mathrm{vars}(\bar{s})$ where $w \neq w'$,

$$\mathrm{vars}(w\mu) \cap \mathrm{vars}(w'\mu) \subseteq \mathrm{gvars}(\mu). \tag{6.49}$$

Since $\sigma \in \mathit{VSubst}$, we have $\mathrm{vars}(u_i\sigma) \subseteq \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)$ for each $i = 1, \ldots, k$. Therefore

$$\mathrm{vars}(\mu) \subseteq \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma). \tag{6.50}$$

Let $\nu, \tau \in \mathit{RSubst}$ be defined as

$$\begin{aligned} \nu &\stackrel{\mathrm{def}}{=} \bigl\{\, (y \mapsto s) \in \sigma \bigm| y \notin \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma) \,\bigr\}, \\ \tau &\stackrel{\mathrm{def}}{=} \nu \circ \mu. \end{aligned}$$

As $\nu \subseteq \sigma \in \mathit{VSubst}$, by Lemma 3.10 we have $\nu \in \mathit{VSubst}$; moreover, by (6.50), we have $\mathrm{dom}(\nu) \cap \mathrm{vars}(\mu) = \varnothing$; thus we can apply Lemma 3.18 to obtain $\tau \in \mathit{VSubst}$. By applying Lemma 6.39, we also have $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$.

Let

$$\begin{aligned} R_- &= \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), \mathrm{ssets}(\sigma)\bigr), \\ R_r &= \mathrm{rel}\bigl(\mathrm{vars}(r), \mathrm{ssets}(\sigma)\bigr), \\ R_{r'} &= \mathrm{rel}\bigl(\mathrm{vars}(r'), \mathrm{ssets}(\sigma)\bigr). \end{aligned}$$

Since $\sigma \in \gamma_S(d)$, we have $sh \supseteq \mathrm{ssets}(\sigma)$ so that, using the monotonicity of $\overline{\mathrm{rel}}$, rel, $(\cdot)^\star$ and bin, we obtain

$$sh_- \cup \mathrm{bin}(sh_r^\star, sh_{r'}) \supseteq R_- \cup \mathrm{bin}(R_r^\star, R_{r'}).$$

Thus, in order to prove (6.47) it is sufficient to show that

$$R_- \cup \mathrm{bin}(R_r^\star, R_{r'}) \supseteq \mathrm{ssets}(\tau). \tag{6.51}$$

Suppose $S \in \mathrm{ssets}(\tau)$. By Definition 3.43 and Theorem 3.44, we have

$$S \in R_- \cup \mathrm{bin}(R_r^\star, R_{r'}^\star).$$

If $S \in R_-$, then (6.51) holds trivially. Therefore suppose $S \in \mathrm{bin}(R_r^\star, R_{r'}^\star)$, so that there exist $S_r \in R_r^\star$ and $S_{r'} \in R_{r'}^\star$ such that $S = S_r \cup S_{r'}$. First note that, since $S_{r'} \in R_{r'}^\star$, there exists $S' \subseteq S_{r'}$ such that $S' \in R_{r'}$. Moreover, if $S_{r'} \in R_r^\star$, then we have $S \in R_r^\star$, so that $S = S \cup S' \in \mathrm{bin}(R_r^\star, R_{r'})$, proving (6.51). Thus, we now assume that $S_{r'} \notin R_r^\star$.

As $S_{r'} \in R_{r'}^\star$, we have $S_{r'} = \bigcup\bigl\{\, \mathrm{occ}(\sigma, w) \bigm| w \in S_{r'} \setminus \mathrm{dom}(\sigma) \,\bigr\}$. From this, as $S_{r'} \notin R_r^\star$, $\mathrm{vars}(\bar{s}) \subseteq \mathrm{dom}(\sigma) \cup \mathrm{vars}(r\sigma)$ and $\mathrm{vars}(r\sigma) \subseteq \mathrm{vars}(\bar{s})$, we obtain

$$S_{r'} = \bigcup\Bigl\{\, \mathrm{occ}(\sigma, w) \Bigm| w \in S_{r'} \setminus \bigl(\mathrm{dom}(\sigma) \cup \mathrm{vars}(\bar{s})\bigr) \,\Bigr\}.$$

Let $w_1, w_2 \in S_{r'} \setminus \bigl(\mathrm{dom}(\sigma) \cup \mathrm{vars}(\bar{s})\bigr)$. Note that, as $S_{r'} \subseteq S$, $S \in \mathrm{ssets}(\tau)$ and $\tau \in \mathit{VSubst}$, by Lemma 3.28 there exists $v \in \mathit{Vars} \setminus \mathrm{dom}(\tau)$ such that $v \in \mathrm{vars}(w_1\tau) \cap \mathrm{vars}(w_2\tau)$. However, since $w_i \notin \mathrm{dom}(\sigma)$, we have $w_i\tau = w_i\mu$, for $i \in \{1, 2\}$. Thus, noting that $v \notin \mathrm{dom}(\tau)$ implies $v \notin \mathrm{gvars}(\tau)$, we can apply (6.49) to conclude that $w_1 = w_2$. As the choice of w_1 and w_2 was arbitrary, there exists a unique variable $w \in S_{r'} \setminus \bigl(\mathrm{dom}(\sigma) \cup \mathrm{vars}(\bar{s})\bigr)$ such that $S_{r'} = \mathrm{occ}(\sigma, w)$. Thus $S_{r'} \in R_{r'}$ and (6.51) holds. □

Lemma 6.48 *Let $sh \in SH$, $(x \mapsto t) \in \mathit{Bind}$ and $V, W \subseteq VI$, where $\{x\} \cup \mathrm{vars}(t) \subseteq VI$ and $\mathrm{rel}(V, sh) \subseteq \mathrm{rel}(W, sh)$. If $sh' \stackrel{\mathrm{def}}{=} \mathrm{amgu}(sh, x \mapsto t)$, then $\mathrm{rel}(V, sh') \subseteq \mathrm{rel}(W, sh')$.*

Proof. Suppose that $S \in \mathrm{rel}(V, sh')$. By Definition 3.43, we have two cases.

1. Suppose that $S \in \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh\bigr)$. Then, we have $S \in sh$ and, in particular, $S \in \mathrm{rel}(V, sh)$. Thus, by hypothesis, $S \in \mathrm{rel}(W, sh)$ and we conclude $S \in \mathrm{rel}(W, sh')$.
2. Otherwise, let $S \in \mathrm{bin}\Bigl(\mathrm{rel}\bigl(\{x\}, sh\bigr)^\star, \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr)^\star\Bigr)$. Then, $S = S_0 \cup \cdots \cup S_n$ where $n \in \mathbb{N}$ and $S_i \in sh$, for each $0 \le i \le n$. Moreover, since $S \in \mathrm{rel}(V, sh')$, there exists an index $j \in \{0, \ldots, n\}$ such that $S_j \in \mathrm{rel}(V, sh)$. Hence, by the hypothesis, $S_j \in \mathrm{rel}(W, sh)$ and, since $S_j \subseteq S$, we can conclude $S \in \mathrm{rel}(W, sh')$.

□

Proposition 6.49 *Let $sh \in SH$, $(x \mapsto t) \in \mathit{Bind}$, where $x \in \mathrm{vars}(t) \subseteq VI$. Let $\tau \in \mathit{RSubst}$ be satisfiable in the syntactic equality theory T and suppose that $T \vdash \forall(\tau \to x = t)$ and $\mathrm{ssets}(\tau) \subseteq sh$. Then $\mathrm{ssets}(\tau) \subseteq \mathrm{cyclic}_x^t(sh)$.*

Proof. Take $V_x = \{x\}$ and $W_t = \mathrm{vars}(t) \setminus \{x\}$. Let $\tau = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where $n = \# \tau$. Define

$$\tau_0 \stackrel{\mathrm{def}}{=} \{x \mapsto t\}, \qquad sh_0 \stackrel{\mathrm{def}}{=} \mathrm{ssets}\bigl(\{x \mapsto t\}\bigr),$$

and, for each $i = 1, \ldots, n$,

$$\tau_i \in \mathrm{mgs}\bigl(\{x_1 = t_1, \ldots, x_i = t_i\} \cup \{x = t\}\bigr), \qquad sh_i \stackrel{\mathrm{def}}{=} \mathrm{amgu}(sh_{i-1}, x_i \mapsto t_i).$$

We show by induction on $i = 0, \ldots, n$ that $\mathrm{ssets}(\tau_i) \subseteq sh_i$ and

$$\mathrm{rel}(V_x, sh_i) \subseteq \mathrm{rel}(W_t, sh_i). \tag{6.52}$$

The base case, when $i = 0$, follows directly from Definition 3.26; note that (6.52) holds because $x \in \mathrm{dom}(\tau_0)$, so that $\mathrm{occ}(\tau_0, x) = \varnothing$.

In the inductive case, when $0 < i \leq n$, we have $\mathrm{ssets}(\tau_{i-1}) \subseteq sh_{i-1}$ so that, by Theorem 3.44, $\mathrm{ssets}(\tau_i) \subseteq sh_i$. By the inductive hypothesis, (6.52) holds for sh_{i-1}. Thus, by Lemma 6.48 (taking $V = V_x$ and $W = W_t$), we obtain that (6.52) also holds for sh_i.

By taking $i = n$, we obtain $\mathrm{rel}(V_x, sh_n) \subseteq \mathrm{rel}(W_t, sh_n)$. Note that, by hypothesis, we have $\tau \in \mathrm{mgs}\bigl(\tau \cup \{x = t\}\bigr) = \mathrm{mgs}(\tau_n)$, so that $T \vdash \forall(\tau \leftrightarrow \tau_n)$. By Lemma 6.41, we have $\mathrm{ssets}(\tau) = \mathrm{ssets}(\tau_n)$, so that $\mathrm{ssets}(\tau) \subseteq sh_n$. As a consequence, $\mathrm{ssets}(\tau) \subseteq sh \cap sh_n$. Thus, by Definition 6.31, we obtain $\mathrm{ssets}(\tau) \subseteq \mathrm{cyclic}_x^t(sh)$. □

6.5.2 The Correctness for Freeness

Proposition 6.50 *Let $\sigma \in VSubst$ and $(x \mapsto y) \in Bind$, where $\{x, y\} \subseteq VI$. Suppose also that $\{x, y\} \subseteq \mathrm{fvars}(\sigma)$. Then, for all $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = y\}\bigr)$ in the syntactic equality theory T, we have*

$$\mathrm{fvars}(\sigma) \subseteq \mathrm{fvars}(\tau). \tag{6.53}$$

Proof. We assume that the congruence and identity axioms hold. Note that if $\sigma \cup \{x = y\}$ is not satisfiable in T, then the result is trivial. We therefore assume, for the rest of the proof, that $\sigma \cup \{x = y\}$ is satisfiable in T. It follows from Lemma 6.41 that we just have to show that there exists $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = y\}\bigr)$ such that (6.53) holds.

As $\{x, y\} \subseteq \mathrm{fvars}(\sigma)$ we have, using Proposition 6.15,

$$\{x\sigma, y\sigma\} \subseteq Vars \setminus \mathrm{dom}(\sigma). \tag{6.54}$$

Suppose first $x\sigma = y\sigma$. Then we have $\sigma \in \mathrm{mgs}\bigl(\sigma \cup \{x\sigma = y\sigma\}\bigr)$, so that by Lemma 6.39, $\sigma \in \mathrm{mgs}\bigl(\sigma \cup \{x = y\}\bigr)$. Thus, by taking $\tau \stackrel{\mathrm{def}}{=} \sigma$, we trivially obtain $\mathrm{fvars}(\sigma) = \mathrm{fvars}(\tau)$.

Otherwise, let $x\sigma \neq y\sigma$ and take $\tau \stackrel{\mathrm{def}}{=} \sigma \cup \{x\sigma = y\sigma\}$. Then, since $\sigma \in RSubst$, it follows from (6.54) that $\tau \in Eqs$ has no identities or circular subsets so that $\tau \in RSubst$. Note that $\mathrm{dom}(\tau) = \mathrm{dom}(\sigma) \cup \{x\sigma\}$. By Lemma 6.39, $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$. Suppose

$z \in \mathrm{fvars}(\sigma)$, so that by Proposition 6.15, $z\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma)$. Then we show that $z \in \mathrm{fvars}(\tau)$. If $z\sigma = x\sigma$ then $z\tau = y\sigma$. On the other hand, if $z\sigma \neq x\sigma$, we have $z\tau = z\sigma$. In both cases, $z\tau \in \mathit{Vars} \setminus \mathrm{dom}(\tau)$, which implies $z \in \mathrm{fvars}(\tau)$. $\square$

Lemma 6.51 *Let $d \in \mathit{SFL}$ and $\sigma \in \gamma_S(d) \cap \mathit{VSubst}$. Let also $y \in \mathit{VI}$ and $t \in \mathit{HTerms}$ be such that* $\mathrm{vars}(t) \subseteq \mathit{VI}$ *and* $y \notin \mathrm{share\_with}_d(t)$. *Then* $\mathrm{vars}(y\sigma) \cap \mathrm{vars}(t\sigma) \subseteq \mathrm{dom}(\sigma)$.

Proof. Let $d = \langle sh, f, l \rangle$ and $V_t = \mathrm{vars}(t)$ so that, by Definition 6.31, $y \notin \mathrm{vars}\bigl(\mathrm{rel}(V_t, sh)\bigr)$. Thus

$$\forall w \in V_t, S \in sh : \{y, w\} \nsubseteq S.$$

By Definition 6.11, since $\sigma \in \gamma_S(d)$, this implies

$$\forall v \in \mathit{Vars}, w \in V_t : \{y, w\} \nsubseteq \mathrm{occ}(\sigma, v).$$

Thus, since $\sigma \in \mathit{VSubst}$, by Lemma 3.28 we obtain

$$\forall w \in V_t : \mathrm{vars}(y\sigma) \cap \mathrm{vars}(w\sigma) \subseteq \mathrm{dom}(\sigma),$$

which is equivalent to the thesis. $\square$

Proposition 6.52 *Let $d \in \mathit{SFL}$ and $\sigma \in \gamma_S(d) \cap \mathit{VSubst}$. Let also $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \mathrm{vars}(t) \subseteq \mathit{VI}$ and suppose that $x \in \mathrm{fvars}(\sigma)$. Then, for all $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ in the syntactic equality theory T, we have*

$$\mathrm{fvars}(\sigma) \setminus \mathrm{share\_with}_d(x) \subseteq \mathrm{fvars}(\tau). \tag{6.55}$$

Proof. We assume that the congruence and identity axioms hold. Note that if $\sigma \cup \{x = t\}$ is not satisfiable, then the result is trivial. We therefore assume, for the rest of the proof, that $\sigma \cup \{x = t\}$ is satisfiable in T. It follows from Lemma 6.41 that we just have to show that there exists $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ such that (6.55) holds.

As $x \in \mathrm{fvars}(\sigma)$ we have, using Proposition 6.15,

$$x\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma). \tag{6.56}$$

Suppose first $x\sigma = t\sigma$. Then we have $\sigma \in \mathrm{mgs}\bigl(\sigma \cup \{x\sigma = t\sigma\}\bigr)$, so that by Lemma 6.39, $\sigma \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$. Thus, by taking $\tau \stackrel{\mathrm{def}}{=} \sigma$, we trivially obtain $\mathrm{fvars}(\sigma) = \mathrm{fvars}(\tau)$, which implies the thesis.

Otherwise, let $x\sigma \neq t\sigma$ and take $\tau \stackrel{\mathrm{def}}{=} \sigma \cup \{x\sigma = t\sigma\}$. Then, since $\sigma \in \mathit{RSubst}$, it follows from (6.56) that $\tau \in \mathit{Eqs}$ has no identities or circular subsets so that $\tau \in \mathit{RSubst}$. By Lemma 6.39, $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$.

Suppose $y \in \mathrm{fvars}(\sigma) \setminus \mathrm{share\_with}_d(x)$. We show that $y \in \mathrm{fvars}(\tau)$. Since $y \in \mathrm{fvars}(\sigma)$, it follows by Proposition 6.15 that

$$y\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma). \tag{6.57}$$

Since $y \notin \mathrm{share\_with}_d(x)$ and $\sigma \in \mathit{VSubst}$, by Lemma 6.51, $\mathrm{vars}(y\sigma) \cap \mathrm{vars}(x\sigma) \subseteq \mathrm{dom}(\sigma)$. From this, by using (6.56) and (6.57), we obtain $y\sigma \neq x\sigma$; from this, again by (6.56), we derive $y \neq x\sigma$. Thus $y\tau = y\sigma$, so that $y \in \mathrm{fvars}(\tau)$. □

Proposition 6.53 *Let $d \in \mathit{SFL}$ and $\sigma \in \gamma_S(d) \cap \mathit{VSubst}$. Let also $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \mathrm{vars}(t) \subseteq \mathit{VI}$. Then, for all $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ in a syntactic equality theory T, we have*

$$\mathrm{fvars}(\sigma) \setminus \bigl(\mathrm{share\_with}_d(x) \cup \mathrm{share\_with}_d(t)\bigr) \subseteq \mathrm{fvars}(\tau). \tag{6.58}$$

Proof. We assume that the congruence and identity axioms hold. Note that if $\sigma \cup \{x = t\}$ is not satisfiable, then the result is trivial. We therefore assume, for the rest of the proof, that $\sigma \cup \{x = t\}$ is satisfiable in T. It follows from Lemma 6.41 that we just have to show that there exists $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ such that (6.58) holds.

Let

$$\begin{aligned}
\{u_1, \ldots, u_k\} &\stackrel{\mathrm{def}}{=} \mathrm{dom}(\sigma) \cap \bigl(\mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)\bigr),\\
\bar{s} &\stackrel{\mathrm{def}}{=} (u_1, \ldots, u_k, x\sigma),\\
\bar{t} &\stackrel{\mathrm{def}}{=} (u_1\sigma, \ldots, u_k\sigma, t\sigma).
\end{aligned}$$

Since $\sigma \in \mathit{VSubst}$, for each $i = 1, \ldots, k$, $\mathrm{vars}(u_i\sigma) \subseteq \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)$. Therefore, for any $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$, we have

$$\mathrm{vars}(\mu) \subseteq \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma). \tag{6.59}$$

Let

$$\begin{aligned}
\nu &\stackrel{\mathrm{def}}{=} \Bigl\{\, z = z\sigma\mu \Bigm| z \in \mathrm{dom}(\sigma) \setminus \bigl(\mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)\bigr) \Bigr\},\\
\tau &\stackrel{\mathrm{def}}{=} \nu \cup \mu.
\end{aligned}$$

Then, as $\sigma, \mu \in \mathit{RSubst}$, it follows from (6.59) that $\nu, \tau \in \mathit{Eqs}$ have no identities or circular subsets so that $\nu, \tau \in \mathit{RSubst}$. Thus, using Lemma 6.39 and the assumption that $\sigma \cup \{x = t\}$ is satisfiable, $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$.

Suppose $y \in \mathrm{fvars}(\sigma) \setminus \bigl(\mathrm{share\_with}_d(x) \cup \mathrm{share\_with}_d(t)\bigr)$. We show that $y \in \mathrm{fvars}(\tau)$. As $y \in \mathrm{fvars}(\sigma)$, by Proposition 6.15,

$$y\sigma \in \mathit{Vars} \setminus \mathrm{dom}(\sigma). \tag{6.60}$$

As $y \notin \mathrm{share\_with}_d(x) \cup \mathrm{share\_with}_d(t)$, it follows from Lemma 6.51 that

$$\mathrm{vars}(y\sigma) \cap \bigl(\mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)\bigr) \subseteq \mathrm{dom}(\sigma).$$

From this, by using (6.60), we obtain

$$y\sigma \notin \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma). \tag{6.61}$$

By using either (6.61), if $y \notin \mathrm{dom}(\sigma)$, or the fact that $\sigma \in \mathit{VSubst}$, if $y \in \mathrm{dom}(\sigma)$, it follows that

$$y \notin \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma). \tag{6.62}$$

Thus, by (6.62), $y\tau = y\nu$ and, by (6.61), $y\nu = y\sigma$, so that $y\tau = y\sigma$ and $y \in \mathrm{fvars}(\tau)$. $\square$

6.5.3 The Correctness for Linearity

Lemma 6.54 *Let $\sigma \in \mathit{VSubst}$ and $y \in \mathrm{dom}(\sigma)$ be such that $y \in \mathrm{lvars}(\sigma) \setminus \mathrm{gvars}(\sigma)$. Then $y \notin \mathrm{vars}(y\sigma)$.*

Proof. Suppose, by contraposition, that $y \in \mathrm{vars}(y\sigma)$. Since $\sigma \in \mathit{VSubst}$ and $y \in \mathrm{lvars}(\sigma)$, by Proposition 6.15 we have $\mathrm{vars}(y\sigma) \cap \mathrm{dom}(\sigma) \subseteq \mathrm{gvars}(\sigma)$. Therefore $y \in \mathrm{gvars}(\sigma)$, contradicting the assumption. $\square$

The following simple consequence of Proposition 6.10 will be used in the sequel.

Lemma 6.55 *If $\sigma \in \mathit{RSubst}$ then* $\mathrm{fvars}(\sigma) \cup \mathrm{gvars}(\sigma) \subseteq \mathrm{lvars}(\sigma)$.

Proof. Let $y \in \mathrm{fvars}(\sigma) \cup \mathrm{gvars}(\sigma)$. By Proposition 6.10, $\mathrm{rt}(y, \sigma) \in \mathit{Vars} \cup \mathit{GTerms}$. However, $\mathit{Vars} \cup \mathit{GTerms} \subset \mathit{LTerms}$ so that, again by Proposition 6.10, $y \in \mathrm{lvars}(\sigma)$. $\square$

Proposition 6.56 *Let $d \in \mathit{SFL}$ and $\sigma \in \gamma_S(d) \cap \mathit{VSubst}$, where $y \in \mathrm{dom}(\sigma) \cap \mathrm{range}(\sigma)$ implies $y \in \mathrm{vars}(y\sigma)$. Let also $(x \mapsto t) \in \mathit{Bind}$, where $\{x\} \cup \mathrm{vars}(t) \subseteq \mathit{VI}$, and suppose that $\{r, r'\} = \{x, t\}$ and $\mathrm{lin}_d(r)$ holds. For all $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ in the syntactic equality theory T, we have*

$$\mathrm{lvars}(\sigma) \setminus \mathrm{share\_with}_d(r) \subseteq \mathrm{lvars}(\tau). \tag{6.63}$$

Proof. We assume that the congruence and identity axioms hold. Note that if $\sigma \cup \{x = t\}$ is not satisfiable, then the result is trivial. We therefore assume, for the rest of the proof, that $\sigma \cup \{x = t\}$ is satisfiable in T. It follows from Lemma 6.41 that we just have to show that there exists $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ such that (6.63) holds.

Since $\mathrm{lin}_d(r)$ holds, then $\mathrm{vars}(r) \subseteq \mathrm{lvars}(\sigma)$. Thus, by Proposition 6.15,

$$\forall v \in \mathrm{vars}(r\sigma) \setminus \mathrm{dom}(\sigma) : \mathrm{occ\_lin}(v, r\sigma), \tag{6.64}$$

$$\mathrm{vars}(r\sigma) \cap \mathrm{dom}(\sigma) \subseteq \mathrm{gvars}(\sigma). \tag{6.65}$$

Therefore, by defining $V_\sigma \stackrel{\mathrm{def}}{=} \mathit{Vars} \setminus \mathrm{gvars}(\sigma)$, we obtain that the term $r\sigma$ is V_σ-linear. Let

$$\begin{aligned}
\{u_1, \ldots, u_k\} &\stackrel{\mathrm{def}}{=} \mathrm{dom}(\sigma) \cap \bigl(\mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)\bigr), \\
\bar{s} &\stackrel{\mathrm{def}}{=} (u_1, \ldots, u_k, r\sigma), \\
\bar{t} &\stackrel{\mathrm{def}}{=} (u_1\sigma, \ldots, u_k\sigma, r'\sigma).
\end{aligned}$$

Since $r\sigma$ is V_σ-linear it follows from (6.65) that $\bar{s}$ is V_σ-linear. By Lemma 6.39 and the congruence axioms, $\sigma \cup \{x = t\} \implies \bar{s} = \bar{t}$. Thus, as $\sigma \cup \{x = t\}$ is satisfiable, there exists

$\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$. Let $V_\mu = \mathit{Vars} \setminus \mathrm{gvars}(\mu)$; since $\mathrm{gvars}(\sigma) \subseteq \mathrm{gvars}(\mu)$, then $V_\mu \subseteq V_\sigma$ and $\bar{s}$ is also V_μ-linear. Therefore, we can apply Lemma 6.42 so that there exists $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$ such that, for all $w \in W \stackrel{\mathrm{def}}{=} \mathit{Vars} \setminus \mathrm{vars}(\bar{s})$,

$$w\mu \text{ is } V_\mu\text{-linear;} \tag{6.66}$$

$$\mathrm{vars}(w\mu) \cap \mathrm{dom}(\mu) \subseteq \mathrm{gvars}(\mu); \tag{6.67}$$

$$\forall w' \in W : w \neq w' \implies \mathrm{vars}(w\mu) \cap \mathrm{vars}(w'\mu) \subseteq \mathrm{gvars}(\mu). \tag{6.68}$$

Since $\sigma \in \mathit{VSubst}$, we have $\mathrm{vars}(u_i\sigma) \subseteq \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)$ for each $i = 1, \ldots, k$. Therefore

$$\mathrm{vars}(\mu) \subseteq \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma). \tag{6.69}$$

Let $\nu, \tau \subseteq \mathit{Eqs}$ be defined as

$$\nu \stackrel{\mathrm{def}}{=} \Bigl\{\, z = z\sigma\mu \Bigm| z \in \mathrm{dom}(\sigma) \setminus \bigl(\mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)\bigr) \,\Bigr\},$$
$$\tau \stackrel{\mathrm{def}}{=} \nu \cup \mu.$$

Then, as $\sigma, \mu \in \mathit{RSubst}$, it follows from (6.69) that ν and τ have no identities or circular subsets so that $\nu, \tau \in \mathit{RSubst}$. By applying Lemma 6.39, we obtain $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$.

Suppose $y \in \mathrm{lvars}(\sigma) \setminus \mathrm{share\_with}_d(r)$. Then we show that $y \in \mathrm{lvars}(\tau)$.

If $y \in \mathrm{gvars}(\sigma)$ then $y \in \mathrm{gvars}(\tau)$. Thus, by Lemma 6.55, $y \in \mathrm{lvars}(\tau)$. Therefore, for the rest of the proof, we assume $y \in \mathrm{lvars}(\sigma) \setminus \mathrm{gvars}(\sigma)$. Thus, by Lemma 6.54, we have $y \in \mathrm{dom}(\sigma)$ implies $y \notin \mathrm{vars}(y\sigma)$ so that, by the hypothesis,

$$y \notin \mathrm{dom}(\sigma) \cap \mathrm{range}(\sigma). \tag{6.70}$$

As $y \in \mathrm{lvars}(\sigma)$, by Proposition 6.15,

$$\forall v \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma) : \mathrm{occ\_lin}(v, y\sigma), \tag{6.71}$$

$$\mathrm{vars}(y\sigma) \cap \mathrm{dom}(\sigma) \subseteq \mathrm{gvars}(\sigma). \tag{6.72}$$

Since $y \notin \mathrm{share\_with}_d(r)$, it follows from Definition 6.31 and Theorem 6.32 that $\mathrm{vars}(y\sigma) \cap \mathrm{vars}(r\sigma) \subseteq \mathrm{gvars}(\sigma)$, so that, by (6.72), we obtain

$$\mathrm{vars}(y\sigma) \cap \mathrm{vars}(\bar{s}) \subseteq \mathrm{gvars}(\sigma). \tag{6.73}$$

We now prove that $y \in \mathrm{lvars}(\tau)$, by showing that

$$\forall v \in \mathrm{vars}(y\tau) \setminus \mathrm{dom}(\tau) : \mathrm{occ\_lin}(v, y\tau), \tag{6.74}$$

$$\mathrm{vars}(y\tau) \cap \mathrm{dom}(\tau) \subseteq \mathrm{gvars}(\tau). \tag{6.75}$$

Since $\mathrm{dom}(\tau) = \mathrm{dom}(\nu) \cup \mathrm{dom}(\mu)$, we have three cases.

1. Suppose first that $y \notin \mathrm{dom}(\tau)$. Then (6.74) holds because $\mathrm{occ\_lin}(y, y)$ is always

true; similarly, (6.75) is true, because $\mathrm{vars}(y\tau) = \{y\}$.

2. Next, suppose that $y \in \mathrm{dom}(\nu)$, so that $y\tau = y\nu = y\sigma\mu$.

   To prove (6.74) we have to prove, for all $w \in \mathrm{vars}(y\sigma)$,

$$\forall v \in \mathrm{vars}(w\mu) \setminus \mathrm{gvars}(\tau) : \mathrm{occ\_lin}(v, w\mu), \tag{6.76}$$
$$\forall w' \in \mathrm{vars}(y\sigma) \setminus \{w\} : \mathrm{vars}(w\mu) \cap \mathrm{vars}(w'\mu) \subseteq \mathrm{gvars}(\tau). \tag{6.77}$$

   We consider two subcases. If $w \in \mathrm{vars}(\bar{s})$ then, by (6.73), $w \in \mathrm{gvars}(\sigma)$. This implies $w \in \mathrm{gvars}(\tau)$ and, as a consequence, $\mathrm{vars}(w\mu) \subseteq \mathrm{gvars}(\tau)$. Thus, both (6.76) and (6.77) hold. Otherwise, if $w \notin \mathrm{vars}(\bar{s})$, then we have $w \in W$ and both (6.66) and (6.68) hold. Therefore, since $\mathrm{gvars}(\mu) \subseteq \mathrm{gvars}(\tau)$, (6.76) follows from (6.66). As for (6.77), this follows either from (6.68), when $w' \notin \mathrm{vars}(\bar{s})$, or from (6.73), when $w' \in \mathrm{vars}(\bar{s})$.

   In order to prove (6.67), note that

$$\mathrm{vars}(y\sigma\mu) \cap \mathrm{dom}(\tau) = \bigcup \bigl\{\, \mathrm{vars}(w\mu) \cap \mathrm{dom}(\tau) \bigm| w \in \mathrm{vars}(y\sigma) \,\bigr\}.$$

   We will prove that, for all $w \in \mathrm{vars}(y\sigma)$, $\mathrm{vars}(w\mu) \cap \mathrm{dom}(\tau) \subseteq \mathrm{gvars}(\tau)$. Let $w \in \mathrm{vars}(y\sigma)$. Suppose first that $w \in \mathrm{gvars}(\sigma)$. As a consequence, $w \in \mathrm{gvars}(\tau)$, which implies $\mathrm{vars}(w\tau) \subseteq \mathrm{gvars}(\tau)$. In particular, $\mathrm{vars}(w\tau) \cap \mathrm{dom}(\tau) \subseteq \mathrm{gvars}(\tau)$. Otherwise, let $w \notin \mathrm{gvars}(\sigma)$ so that, by (6.72), $w \notin \mathrm{dom}(\sigma)$. If also $w \notin \mathrm{dom}(\mu)$, then $w = w\mu \notin \mathrm{dom}(\tau)$ and there is nothing to prove. If $w \in \mathrm{dom}(\mu)$, $\mathrm{vars}(w\mu) \subseteq \mathrm{vars}(\mu)$ so that, by (6.69) and the definition of ν, $\mathrm{vars}(w\mu) \cap \mathrm{dom}(\nu) = \varnothing$. Moreover, by (6.73), we have $w \notin \mathrm{vars}(\bar{s})$. Thus (6.67) applies so that, as $\mathrm{gvars}(\mu) \subseteq \mathrm{gvars}(\tau)$, $\mathrm{vars}(w\mu) \cap \mathrm{dom}(\tau) \subseteq \mathrm{gvars}(\tau)$.

3. Finally, suppose $y \in \mathrm{dom}(\mu)$, so that $y\tau = y\mu$. First, we prove that $y \notin \mathrm{vars}(\bar{s})$. In fact, by definition of $\bar{s}$, if $y \in \mathrm{vars}(\bar{s})$ then $y \in \mathrm{vars}(r\sigma) \cup \mathrm{vars}(r'\sigma)$. Now, if $y \in \mathrm{dom}(\sigma)$, then $y \in \mathrm{range}(\sigma)$, therefore contradicting (6.70). Otherwise, if $y \notin \mathrm{dom}(\sigma)$, then $y \in \mathrm{vars}(y\sigma)$ and, by (6.73), $y \in \mathrm{gvars}(\sigma)$, contradicting our previous assumption.

   Thus, we have $y \notin \mathrm{vars}(\bar{s})$, so that $y \in W$ and (6.66) holds. Note that (6.74) follows because $\mathrm{gvars}(\mu) \subseteq \mathrm{gvars}(\tau) \subseteq \mathrm{dom}(\tau)$, so that $\mathrm{vars}(y\tau) \setminus \mathrm{dom}(\tau) \subseteq V_\mu$.

   Similarly, (6.67) holds and we obtain (6.75) by observing that $\mathrm{gvars}(\mu) \subseteq \mathrm{gvars}(\tau)$.

□

Proposition 6.57 *Let $d \in SFL$ and $\sigma \in \gamma_S(d) \cap VSubst$, where $y \in \mathrm{dom}(\sigma) \cap \mathrm{range}(\sigma)$ implies $y \in \mathrm{vars}(y\sigma)$. Let also $(x \mapsto t) \in Bind$, where $\{x\} \cup \mathrm{vars}(t) \subseteq VI$. For all $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ in the syntactic equality theory T, we have*

$$\mathrm{lvars}(\sigma) \setminus \bigl(\mathrm{share\_with}_d(x) \cup \mathrm{share\_with}_d(t)\bigr) \subseteq \mathrm{lvars}(\tau). \tag{6.78}$$

Proof. We assume that the congruence and identity axioms hold. Note that if $\sigma \cup \{x = t\}$ is not satisfiable, then the result is trivial. We therefore assume, for the rest of the proof, that $\sigma \cup \{x = t\}$ is satisfiable in T. It follows from Lemma 6.41 that we just have to show that there exists $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ such that (6.78) holds.

Let

$$\begin{aligned}\{u_1, \ldots, u_k\} &\stackrel{\mathrm{def}}{=} \mathrm{dom}(\sigma) \cap \bigl(\mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)\bigr),\\ \bar{s} &\stackrel{\mathrm{def}}{=} (u_1, \ldots, u_k, x\sigma),\\ \bar{t} &\stackrel{\mathrm{def}}{=} (u_1\sigma, \ldots, u_k\sigma, t\sigma).\end{aligned}$$

By Lemma 6.39 and the congruence axioms, $\sigma \cup \{x = t\} \implies \bar{s} = \bar{t}$. Thus, as $\sigma \cup \{x = t\}$ is satisfiable, there exists $\mu \in \mathrm{mgs}(\bar{s} = \bar{t})$. Since $\sigma \in \mathit{VSubst}$, $\mathrm{vars}(u_i\sigma) \subseteq \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)$ for each $i = 1, \ldots, k$. Therefore

$$\mathrm{vars}(\mu) \subseteq \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma). \tag{6.79}$$

Let $\nu, \tau \subseteq \mathit{Eqs}$ be defined as

$$\begin{aligned}\nu &\stackrel{\mathrm{def}}{=} \Bigl\{\, z = z\sigma\mu \Bigm| z \in \mathrm{dom}(\sigma) \setminus \bigl(\mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)\bigr) \,\Bigr\},\\ \tau &\stackrel{\mathrm{def}}{=} \nu \cup \mu.\end{aligned}$$

Then, as $\sigma, \mu \in \mathit{RSubst}$, it follows from (6.79) that ν and τ have no identities or circular subsets so that $\nu, \tau \in \mathit{RSubst}$. By applying Lemma 6.39, we obtain $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$.

Supposing $y \in \mathrm{lvars}(\sigma) \setminus \bigl(\mathrm{share\_with}_d(x) \cup \mathrm{share\_with}_d(t)\bigr)$, we show that $y \in \mathrm{lvars}(\tau)$.

If $y \in \mathrm{gvars}(\sigma)$ then $y \in \mathrm{gvars}(\tau)$. Thus, by Lemma 6.55, $y \in \mathrm{lvars}(\tau)$. Therefore, for the rest of the proof, we assume $y \in \mathrm{lvars}(\sigma) \setminus \mathrm{gvars}(\sigma)$. Thus, by Lemma 6.54, we have $y \in \mathrm{dom}(\sigma)$ implies $y \notin \mathrm{vars}(y\sigma)$ so that, by the hypothesis,

$$y \notin \mathrm{dom}(\sigma) \cap \mathrm{range}(\sigma). \tag{6.80}$$

As $y \in \mathrm{lvars}(\sigma)$, by Proposition 6.15,

$$\forall v \in \mathrm{vars}(y\sigma) \setminus \mathrm{dom}(\sigma) : \mathrm{occ\_lin}(v, y\sigma), \tag{6.81}$$

$$\mathrm{vars}(y\sigma) \cap \mathrm{dom}(\sigma) \subseteq \mathrm{gvars}(\sigma). \tag{6.82}$$

As $y \notin \mathrm{share\_with}_d(x) \cup \mathrm{share\_with}_d(t)$, by Definition 6.31 and Theorem 6.32, we have

$$\mathrm{vars}(y\sigma) \cap \bigl(\mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)\bigr) \subseteq \mathrm{gvars}(\sigma). \tag{6.83}$$

Moreover, by (6.79), this implies

$$\mathrm{vars}(y\sigma) \cap \mathrm{vars}(\mu) \subseteq \mathrm{gvars}(\sigma). \tag{6.84}$$

We now prove that $y \in \mathrm{lvars}(\tau)$, by showing that

$$\forall v \in \mathrm{vars}(y\tau) \setminus \mathrm{dom}(\tau) : \mathrm{occ\_lin}(v, y\tau), \tag{6.85}$$

$$\mathrm{vars}(y\tau) \cap \mathrm{dom}(\tau) \subseteq \mathrm{gvars}(\tau). \tag{6.86}$$

Since $\mathrm{dom}(\tau) = \mathrm{dom}(\nu) \cup \mathrm{dom}(\mu)$, we have three cases.

1. Suppose first that $y \notin \mathrm{dom}(\tau)$. Then (6.85) holds because $\mathrm{occ\_lin}(y, y)$ is always true; similarly, (6.86) is true, because $\mathrm{vars}(y\tau) = \{y\}$.

2. Next, suppose that $y \in \mathrm{dom}(\nu)$, so that $y\tau = y\nu = y\sigma\mu$.

   To prove (6.85), consider $v \in \mathrm{vars}(y\sigma\mu) \setminus \mathrm{dom}(\tau)$. If $v \in \mathrm{vars}(\mu)$ then there exists a variable $w \in \mathrm{vars}(y\sigma) \cap \mathrm{vars}(\mu)$ such that $v \in \mathrm{vars}(w\mu)$. By (6.84), we have $w \in \mathrm{gvars}(\sigma)$, which implies $w \in \mathrm{gvars}(\tau)$. Thus, $\mathrm{vars}(w\mu) \subseteq \mathrm{gvars}(\tau) \subseteq \mathrm{dom}(\tau)$, therefore contradicting our assumption that $v \notin \mathrm{dom}(\tau)$. Therefore, $v \notin \mathrm{vars}(\mu)$, so that $v \in \mathrm{vars}(y\sigma)$. Since $\mathrm{dom}(\sigma) \subseteq \mathrm{dom}(\tau)$, by (6.81), we have $\mathrm{occ\_lin}(v, y\sigma)$. Moreover, for all $w \in \mathrm{vars}(y\sigma) \cap \mathrm{dom}(\mu)$, we have $v \notin \mathrm{vars}(w\mu)$, because $v \notin \mathrm{vars}(\mu)$. Thus, we obtain $\mathrm{occ\_lin}(v, y\sigma\mu)$.

   In order to prove (6.86), note that

   $$\mathrm{vars}(y\sigma\mu) \cap \mathrm{dom}(\tau) = \bigcup \bigl\{\, \mathrm{vars}(w\mu) \cap \mathrm{dom}(\tau) \bigm| w \in \mathrm{vars}(y\sigma) \,\bigr\}.$$

   We will prove that, for all $w \in \mathrm{vars}(y\sigma)$, $\mathrm{vars}(w\mu) \cap \mathrm{dom}(\tau) \subseteq \mathrm{gvars}(\tau)$. Thus, let $w \in \mathrm{vars}(y\sigma)$. If $w \in \mathrm{vars}(\mu)$ then, by (6.84), $w \in \mathrm{gvars}(\sigma)$, which implies $w \in \mathrm{gvars}(\tau)$. Thus, we obtain $\mathrm{vars}(w\mu) \subseteq \mathrm{gvars}(\tau)$. Otherwise, let $w \notin \mathrm{vars}(\mu)$, so that $w = w\mu$. Thus, $w \notin \mathrm{dom}(\mu)$. If also $w \notin \mathrm{dom}(\sigma)$, then $w \notin \mathrm{dom}(\tau)$ and there is nothing to prove. If $w \in \mathrm{dom}(\sigma)$, by (6.82), $w \in \mathrm{gvars}(\sigma)$, which implies $w \in \mathrm{gvars}(\tau)$. Thus, $\mathrm{vars}(w\mu) \subseteq \mathrm{gvars}(\tau)$.

3. Finally, suppose $y \in \mathrm{dom}(\mu)$, so that $y\tau = y\mu$. Toward a contradiction, assume that $y \in \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)$. Now, if $y \in \mathrm{dom}(\sigma)$, then $y \in \mathrm{range}(\sigma)$, thus contradicting (6.80). Otherwise, if $y \notin \mathrm{dom}(\sigma)$, then $y \in \mathrm{vars}(y\sigma)$ and, by (6.83), $y \in \mathrm{gvars}(\sigma)$, contradicting our previous assumption.

   Thus, we have $y \notin \mathrm{vars}(x\sigma) \cup \mathrm{vars}(t\sigma)$, so that, by (6.79), we obtain $y\mu = y$. As a consequence, (6.85) holds trivially. As for (6.86), this follows from (6.82) if $y \in \mathrm{dom}(\sigma)$, while being trivial if $y \notin \mathrm{dom}(\sigma)$.

□

6.5.4 Putting Results Together

By exploiting the correctness results regarding each of the three components of the domain *SFL*, we now prove the correctness of the amgu_S operator. We start by proving a restricted result that only applies to variable-idempotent substitutions.

Lemma 6.58 *Let $d \in SFL$ and $\sigma \in \gamma_S(d) \cap VSubst$, where $y \in \mathrm{dom}(\sigma) \cap \mathrm{range}(\sigma)$ implies $y \in \mathrm{vars}(y\sigma)$. Let also $(x \mapsto t) \in Bind$, where $\{x\} \cup \mathrm{vars}(t) \subseteq VI$, and suppose there exists $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$ in the syntactic equality theory T. Then $\tau \in \gamma_S\bigl(\mathrm{amgu}_S(d, x \mapsto t)\bigr)$.*

Proof. Let $d = \langle sh, f, l \rangle$ and $d' = \langle sh', f', l' \rangle \stackrel{\mathrm{def}}{=} \mathrm{amgu}_S(d, x \mapsto t)$. Note that, by the existence of σ and τ as specified in the hypotheses, we have both $d \neq \bot_S$ and $d' \neq \bot_S$. Since $\sigma \in \gamma_S(d)$, it follows from Definition 6.11 that $\mathrm{ssets}(\sigma) \subseteq sh$, $\mathrm{fvars}(\sigma) \supseteq f$ and $\mathrm{lvars}(\sigma) \supseteq l$. Therefore, to prove $\tau \in \gamma_S(d')$, we have to show that

$$\mathrm{ssets}(\tau) \subseteq sh', \tag{6.87}$$

$$\mathrm{fvars}(\tau) \supseteq f', \tag{6.88}$$

$$\mathrm{lvars}(\tau) \supseteq l'. \tag{6.89}$$

We prove each inclusion separately.

(6.87). By Definition 6.33, we have $sh' = \mathrm{cyclic}_x^t(sh_- \cup sh'')$. We will show that

$$\mathrm{ssets}(\tau) \subseteq sh_- \cup sh''. \tag{6.90}$$

From this, the thesis will follow by Proposition 6.49. To prove (6.90) we need to consider five cases.

1. $\mathrm{free}_d(x) \vee \mathrm{free}_d(t)$ holds.

   We can apply Proposition 6.44, taking $r = x$ when $\mathrm{free}_d(x)$ holds and $r = t$ when $\mathrm{free}_d(x)$ does not hold, to conclude that $\mathrm{ssets}(\tau) \subseteq sh'$.

2. $\mathrm{lin}_d(x) \wedge \mathrm{lin}_d(t)$ holds.

   We can apply Proposition 6.45, obtaining

   $$\mathrm{ssets}(\tau) \subseteq sh_- \cup \mathrm{bin}\bigl(sh_x \cup \mathrm{bin}(sh_x, sh_{xt}^\star), sh_t \cup \mathrm{bin}(sh_t, sh_{xt}^\star)\bigr) \stackrel{\mathrm{def}}{=} sh'.$$

3. $\mathrm{lin}_d(x)$ holds.

   We can apply Proposition 6.47, taking $r = x$ to conclude that

   $$\mathrm{ssets}(\tau) \subseteq sh_- \cup \mathrm{bin}(sh_x^\star, sh_t) \stackrel{\mathrm{def}}{=} sh'.$$

4. $\mathrm{lin}_d(t)$ holds. This case is symmetric to the previous one.

5. When nothing is known about x and t, we can apply Theorem 3.44:

   $$\mathrm{ssets}(\tau) \subseteq sh_- \cup \mathrm{bin}(sh_x^\star, sh_t^\star) \stackrel{\mathrm{def}}{=} sh'.$$

(6.88). In order to show that $\mathrm{fvars}(\tau) \supseteq f'$, according to Definition 6.33, we consider four cases.

1. $\text{free}_d(x) \wedge \text{free}_d(t)$ holds.

   By Definition 6.31, $\{x, t\} \subseteq f \subseteq \text{fvars}(\sigma)$. Therefore we can apply Proposition 6.50 (where y is replaced by $t \in VI$) to conclude that

$$f' \stackrel{\text{def}}{=} f \subseteq \text{fvars}(\sigma) \subseteq \text{fvars}(\tau).$$

2. $\text{free}_d(x)$ holds.

   By Definition 6.31, $x \in f \subseteq \text{fvars}(\sigma)$. Therefore we can apply Proposition 6.52 to conclude that

$$\begin{aligned} f' &\stackrel{\text{def}}{=} f \setminus \text{share\_with}_d(x) \\ &\subseteq \text{fvars}(\sigma) \setminus \text{share\_with}_d(x) \\ &\subseteq \text{fvars}(\tau). \end{aligned}$$

3. $\text{free}_d(t)$ holds.

   This case is symmetric to the previous one.

4. When nothing is known about x and t, we can apply Proposition 6.53:

$$\begin{aligned} f' &\stackrel{\text{def}}{=} f \setminus \bigl(\text{share\_with}_d(x) \cup \text{share\_with}_d(t)\bigr) \\ &\subseteq \text{fvars}(\sigma) \setminus \bigl(\text{share\_with}_d(x) \cup \text{share\_with}_d(t)\bigr) \\ &\subseteq \text{fvars}(\tau). \end{aligned}$$

(6.89). In order to show that $\text{lvars}(\tau) \supseteq l'$, according to Definition 6.33, we start by proving $\text{lvars}(\tau) \supseteq l''$. There are four cases that have to be considered.

1. $\text{lin}_d(x) \wedge \text{lin}_d(t)$ holds.

   We can apply Proposition 6.56 twice, the first time taking $r = x$ and the second time taking $r = t$, to conclude that

$$\begin{aligned} l'' &\stackrel{\text{def}}{=} l \setminus \bigl(\text{share\_with}_d(x) \cap \text{share\_with}_d(t)\bigr) \\ &= \bigl(l \setminus \text{share\_with}_d(x)\bigr) \cup \bigl(l \setminus \text{share\_with}_d(t)\bigr) \\ &\subseteq \bigl(\text{lvars}(\sigma) \setminus \text{share\_with}_d(x)\bigr) \cup \bigl(\text{lvars}(\sigma) \setminus \text{share\_with}_d(t)\bigr) \\ &\subseteq \text{lvars}(\tau). \end{aligned}$$

2. $\text{lin}_d(x)$ holds.

   We can apply Proposition 6.56 (where we take $r = x$) to conclude that

$$\begin{aligned} l'' &\stackrel{\text{def}}{=} l \setminus \text{share\_with}_d(x) \\ &\subseteq \text{lvars}(\sigma) \setminus \text{share\_with}_d(x) \\ &\subseteq \text{lvars}(\tau). \end{aligned}$$

3. $\mathrm{lin}_d(t)$ holds.

   This case is symmetric to the previous one.

4. When nothing is known about x and t, we can apply Proposition 6.57:

$$\begin{aligned} l'' &\stackrel{\text{def}}{=} l \setminus \bigl(\text{share\_with}_d(x) \cup \text{share\_with}_d(t)\bigr) \\ &\subseteq \mathrm{lvars}(\sigma) \setminus \bigl(\text{share\_with}_d(x) \cup \text{share\_with}_d(t)\bigr) \\ &\subseteq \mathrm{lvars}(\tau). \end{aligned}$$

Therefore, we have $\mathrm{lvars}(\tau) \supseteq l''$. By (6.87) proved above, $\mathrm{ssets}(\tau) \subseteq sh'$. By Definition 6.5, $\mathit{VI} \setminus \mathrm{vars}(sh') \subseteq \mathrm{gvars}(\tau)$. Moreover, by (6.88) proved above, $\mathrm{fvars}(\tau) \supseteq f'$. Thus, by applying Lemma 6.55 we obtain the thesis:

$$\begin{aligned} \mathrm{lvars}(\tau) &\supseteq \mathrm{gvars}(\tau) \cup \mathrm{fvars}(\tau) \cup \mathrm{lvars}(\tau) \\ &\supseteq \bigl(\mathit{VI} \setminus \mathrm{vars}(sh')\bigr) \cup f' \cup l'' \\ &\stackrel{\text{def}}{=} l'. \end{aligned}$$

□

Finally, by exploiting the results proved in Section 6.3, we drop the assumption about variable-idempotent substitutions, therefore completing the proof of correctness of amgu_S.

Proof of Theorem 6.36 on page 154. Let $d' = \mathrm{amgu}_S(d, x \mapsto t)$.

If $d = \bot_S$ then we have $d' = \bot_S$ and the result holds trivially, since $\gamma_S(d) = \varnothing$. Similarly, if $T = \mathcal{FT}$ is the theory of finite trees and $x \in \mathrm{vars}(t)$, then $d' = \bot_S$. Again, the result holds trivially, since the equation $\{x = t\}$ is not satisfiable in $\mathcal{FT}$, so that $\mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr) = \varnothing$.

Therefore suppose there exists $\sigma \in \gamma_S(d)$ and $\tau \in \mathrm{mgs}\bigl(\sigma \cup \{x = t\}\bigr)$. By Corollary 3.17, there exists $\sigma' \in \mathit{VSubst}$ such that $T \vdash \forall(\sigma \leftrightarrow \sigma')$ and $y \in \mathrm{dom}(\sigma') \cap \mathrm{range}(\sigma')$ implies $y \in \mathrm{vars}(y\sigma')$. By Theorem 6.12 and Definition 6.11, we have $\sigma \in \gamma_S(d)$ if and only if $\sigma' \in \gamma_S(d)$. Therefore, the result follows by application of Lemma 6.58. □

6.6 Eliminating Redundancies in *SFL*

As done in Chapter 4 for the plain set-sharing domain, even when considering the richer domain *SFL* it is natural to question whether it contains redundancies with respect to the computation of the observable properties of the analysis.

It is worth stressing that the results presented in Chapters 4 and 5 cannot be simply inherited by the new domain. The concept of "redundancy" depends on both the starting domain and the given observables: in the *SFL* domain both of these have changed. First of all, as can be seen by looking at the definition of amgu_S, freeness and linearity positively interact in the computation of sharing information: *a priori* it is an open issue whether or not the "redundant" sharing groups can play a role in such an interaction. Secondly, since freeness and linearity information can be themselves usefully exploited in a number of applications of static analysis (e.g., in the optimized implementation of concrete unification or in occurs-check reduction), these properties have to be included in the observables.

This stated, we will now show that the domain *SFL* can be simplified by applying the same notion of redundancy as identified in Chapter 4. Namely, in the definition of *SFL* it is possible to replace the set-sharing component *SH* by *PSD* without affecting the precision on groundness, pair-sharing, freeness and linearity. In order to prove such a claim, we now formalize the new observable properties.

Definition 6.59 (The observables of *SFL***.)** *For each* $\langle sh, f, l\rangle \in \mathit{SFL}$*, the (overloaded)* groundness *and* pair-sharing *observables* $\rho_{Con}, \rho_{PS} \in \mathrm{uco}(\mathit{SFL})$ *are defined by*

$$\rho_{Con}\bigl(\langle sh, f, l\rangle\bigr) \stackrel{\mathrm{def}}{=} \bigl\langle \rho_{Con}(sh), \varnothing, \varnothing \bigr\rangle,$$
$$\rho_{PS}\bigl(\langle sh, f, l\rangle\bigr) \stackrel{\mathrm{def}}{=} \bigl\langle \rho_{PS}(sh), \varnothing, \varnothing \bigr\rangle;$$

the freeness *and* linearity *observables* $\rho_F, \rho_L \in \mathrm{uco}(\mathit{SFL})$ *are defined by*

$$\rho_F\bigl(\langle sh, f, l\rangle\bigr) \stackrel{\mathrm{def}}{=} \langle SG, f, \varnothing\rangle,$$
$$\rho_L\bigl(\langle sh, f, l\rangle\bigr) \stackrel{\mathrm{def}}{=} \langle SG, \varnothing, l\rangle.$$

The overloading of ρ_{PSD} working on the domain *SFL* leaves the freeness and linearity components untouched.

Definition 6.60 (Non-redundant *SFL***.)** *The operator* $\rho_{PSD} \colon \mathit{SFL} \to \mathit{SFL}$ *is defined, for each* $\langle sh, f, l\rangle \in \mathit{SFL}$*, by*

$$\rho_{PSD}\bigl(\langle sh, f, l\rangle\bigr) \stackrel{\mathrm{def}}{=} \bigl\langle \rho_{PSD}(sh), f, l \bigr\rangle.$$

For notational convenience, remembering that $\rho_{PSD} = \rho_{TSD_2}$, we will write SFL_2 to denote the induced lattice $\rho_{PSD}(\mathit{SFL})$.

By Corollary 5.13, we have that $\rho_{PSD} \sqsubseteq (\rho_{Con} \sqcap \rho_{PS})$; by the above definitions, it is also straightforward to observe that $\rho_{PSD} \sqsubseteq (\rho_F \sqcap \rho_L)$; thus, ρ_{PSD} is more precise than the reduced product $(\rho_{Con} \sqcap \rho_{PS} \sqcap \rho_F \sqcap \rho_L)$. Informally, this means that the domain SFL_2 is able to *represent* all of our observable properties without precision losses.

The next theorem shows that ρ_{PSD} is a congruence with respect to the aunify_S, alub_S and $\mathrm{aexists}_S$ operators. This means that the domain SFL_2 is able to *propagate* the information on the observables as precisely as *SFL*, therefore providing a completeness result.

Theorem 6.61 *Let* $d_1, d_2 \in \mathit{SFL}$ *be such that* $\rho_{PSD}(d_1) = \rho_{PSD}(d_2)$*. Then, for each se-*

quence of bindings $bs \in Bind^\star$, for each $d' \in SFL$ and $V \in \wp(VI)$,

$$\begin{aligned}
\rho_{PSD}\bigl(\mathrm{aunify}_S(d_1, bs)\bigr) &= \rho_{PSD}\bigl(\mathrm{aunify}_S(d_2, bs)\bigr),\\
\rho_{PSD}\bigl(\mathrm{alub}_S(d', d_1)\bigr) &= \rho_{PSD}\bigl(\mathrm{alub}_S(d', d_2)\bigr),\\
\rho_{PSD}\bigl(\mathrm{aexists}_S(d_1, V)\bigr) &= \rho_{PSD}\bigl(\mathrm{aexists}_S(d_2, V)\bigr).
\end{aligned}$$

Finally, by providing the minimality result, we show that the domain SFL_2 is indeed the generalized quotient of SFL with respect to the reduced product $(\rho_{Con} \sqcap \rho_{PS} \sqcap \rho_F \sqcap \rho_L)$.

Theorem 6.62 *For each $i \in \{1, 2\}$, let $d_i = \langle sh_i, f_i, l_i\rangle \in SFL$. Moreover, suppose that $\rho_{PSD}(d_1) \neq \rho_{PSD}(d_2)$. Then there exist a sequence of bindings $bs \in Bind^\star$ and an observable property $\rho \in \{\rho_{Con}, \rho_{PS}, \rho_F, \rho_L\}$ such that*

$$\rho\bigl(\mathrm{aunify}_S(d_1, bs)\bigr) \neq \rho\bigl(\mathrm{aunify}_S(d_2, bs)\bigr).$$

As far as the implementation is concerned, the results proved in Section 4.5 for the domain PSD can also be applied to SFL_2. In particular, in the definition of amgu_S all the calls to the star-union operator can be safely replaced by calls to the 2-self-union operator.

The following result provides another optimization that can be applied when both terms x and t are definitely linear, but none of them is definitely free (i.e., when we compute sh'' by the second case stated in Definition 6.33; for notational convenience, we denote this sharing-set by $sh^\diamond$).

Theorem 6.63 *Let $sh \in SH$ and $(x \mapsto t) \in Bind$, where $\{x\} \cup \mathrm{vars}(t) \subseteq VI$. Let*

$$\begin{aligned}
sh_x &\stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\{x\}, sh\bigr), & sh_- &\stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh\bigr),\\
sh_t &\stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr), & sh_{xt} &\stackrel{\mathrm{def}}{=} sh_x \cap sh_t,\\
W &\stackrel{\mathrm{def}}{=} \mathrm{vars}(t) \setminus \{x\}, & sh_W &\stackrel{\mathrm{def}}{=} \mathrm{rel}(W, sh).
\end{aligned}$$

Then it holds

$$\rho_{PSD}\bigl(\mathrm{cyclic}_x^t(sh^\diamond)\bigr) = \begin{cases} \rho_{PSD}\bigl(sh_- \cup \mathrm{bin}(sh_x, sh_t)\bigr), & \text{if } x \notin \mathrm{vars}(t);\\ \rho_{PSD}\bigl(sh_- \cup \mathrm{bin}(sh_x^2, sh_W)\bigr), & \text{otherwise;}\end{cases}$$

where $sh^\diamond \stackrel{\mathrm{def}}{=} sh_- \cup \mathrm{bin}\bigl(sh_x \cup \mathrm{bin}(sh_x, sh_{xt}^\star), sh_t \cup \mathrm{bin}(sh_t, sh_{xt}^\star)\bigr)$.

Therefore, even when terms x and t possibly share (i.e., when $sh_{xt} \neq \varnothing$), by using SFL_2 we can avoid the expensive computation of at least one of the two inner binary unions in the expression for $sh^\diamond$.

6.6.1 Proofs of the Results of Section 6.6

The next three Lemmas show that the precision of the abstract evaluation of the operators specified in Definition 6.31 is not affected by ρ_{PSD}.

Lemma 6.64 *For each $V \subseteq VI$ and $sh \in SH$ it holds*

$$\mathrm{vars}\bigl(\mathrm{rel}(V, sh)\bigr) = \mathrm{vars}\Bigl(\mathrm{rel}\bigl(V, \rho_{PSD}(sh)\bigr)\Bigr).$$

Proof. If $V = \varnothing$, the result is trivial. Thus, assume $V \neq \varnothing$.

The first inclusion ($\subseteq$) follows from the extensivity of ρ_{PSD} and the monotonicity of the operators rel and vars. To prove the other inclusion, let $S \in \mathrm{rel}\bigl(V, \rho_{PSD}(sh)\bigr)$. By instantiating Definition 5.10 for $k = 2$, we obtain

$$\forall x \in S : S = \bigcup \bigl\{\, T \in sh \bigm| \{x\} \subseteq T \subseteq S \,\bigr\}.$$

In particular, for all $x \in S \cap V$, it holds

$$\begin{aligned} S &= \bigcup \bigl\{\, T \in sh \bigm| \{x\} \subseteq T \subseteq S \,\bigr\} \\ &= \bigcup \bigl\{\, T \in \mathrm{rel}(V, sh) \bigm| \{x\} \subseteq T \subseteq S \,\bigr\} \\ &\subseteq \bigcup \mathrm{rel}(V, sh) \\ &= \mathrm{vars}\bigl(\mathrm{rel}(V, sh)\bigr). \end{aligned}$$

Since the choice of S was arbitrary, we obtain the desired inclusion

$$\mathrm{vars}\bigl(\mathrm{rel}(V, sh)\bigr) \supseteq \mathrm{vars}\Bigl(\mathrm{rel}\bigl(V, \rho_{PSD}(sh)\bigr)\Bigr).$$

□

Lemma 6.65 *For each $V, W \subseteq VI$ and $sh \in SH$ it holds*

$$\bigl(\mathrm{rel}(V, sh) \cap \mathrm{rel}(W, sh) = \varnothing\bigr) \iff \Bigl(\mathrm{rel}\bigl(V, \rho_{PSD}(sh)\bigr) \cap \mathrm{rel}\bigl(W, \rho_{PSD}(sh)\bigr) = \varnothing\Bigr).$$

Proof. To prove the first implication ($\Rightarrow$), we reason by contraposition and suppose

$$\mathrm{rel}\bigl(V, \rho_{PSD}(sh)\bigr) \cap \mathrm{rel}\bigl(W, \rho_{PSD}(sh)\bigr) \neq \varnothing.$$

Thus, there exists $S \in \rho_{PSD}(sh)$ such that $S \cap V \neq \varnothing$ and $S \cap W \neq \varnothing$. Consider $x \in S \cap V$ and $y \in S \cap W$, so that we have $\{x, y\} \subseteq S$.

By instantiating Definition 5.10 for $k = 2$, we obtain

$$\forall v \in S : S = \bigcup \bigl\{\, T \in sh \bigm| \{v\} \subseteq T \subseteq S \,\bigr\}.$$

In particular, by taking $v = x$, there exists $T \in sh$ such that $\{x, y\} \subseteq T$, so that

$$T \in \mathrm{rel}(V, sh) \cap \mathrm{rel}(W, sh) \neq \varnothing.$$

The other implication ($\Leftarrow$) follows by the extensivity of ρ_{PSD} and the monotonicity of rel. □

Lemma 6.66 *For each $i \in \{1,2\}$, let $d_i = \langle sh_i, f, l \rangle \in SFL$. If $\rho_{PSD}(sh_1) = \rho_{PSD}(sh_2)$ then, for all $s, t \in HTerms$ and $y \in VI$,*

$$\begin{aligned}
\mathrm{ind}_{d_1}(s,t) &\iff \mathrm{ind}_{d_2}(s,t); && (6.91)\\
\mathrm{free}_{d_1}(t) &\iff \mathrm{free}_{d_2}(t); && (6.92)\\
\mathrm{ground}_{d_1}(t) &\iff \mathrm{ground}_{d_2}(t); && (6.93)\\
\mathrm{occ_lin}_{d_1}(y,t) &\iff \mathrm{occ_lin}_{d_2}(y,t); && (6.94)\\
\mathrm{lin}_{d_1}(t) &\iff \mathrm{lin}_{d_2}(t); && (6.95)\\
\mathrm{share_with}_{d_1}(t) &= \mathrm{share_with}_{d_2}(t). && (6.96)
\end{aligned}$$

Proof. Consider equivalence (6.91) and let $V = \mathrm{vars}(s)$, $W = \mathrm{vars}(t)$. By Definition 6.31, Lemma 6.65 and the hypothesis, we obtain

$$\begin{aligned}
\mathrm{ind}_{d_1}(s,t) &\iff \mathrm{rel}(V, sh_1) \cap \mathrm{rel}(W, sh_1) \neq \varnothing\\
&\iff \mathrm{rel}\bigl(V, \rho_{PSD}(sh_1)\bigr) \cap \mathrm{rel}\bigl(W, \rho_{PSD}(sh_1)\bigr) \neq \varnothing\\
&\iff \mathrm{rel}\bigl(V, \rho_{PSD}(sh_2)\bigr) \cap \mathrm{rel}\bigl(W, \rho_{PSD}(sh_2)\bigr) \neq \varnothing\\
&\iff \mathrm{rel}(V, sh_2) \cap \mathrm{rel}(W, sh_2) \neq \varnothing\\
&\iff \mathrm{ind}_{d_2}(s,t).
\end{aligned}$$

The proof of (6.92) follows easily from Definition 6.31, since the predicate $\mathrm{free}_{d_i}(t)$ does not depend on the sharing component sh_i of d_i.

Consider now (6.93). By Definition 6.31, Lemma 6.64 and the hypothesis, we obtain

$$\begin{aligned}
\mathrm{ground}_{d_1}(t) &\iff \mathrm{vars}(t) \subseteq VI \setminus \mathrm{vars}(sh_1)\\
&\iff \mathrm{vars}(t) \subseteq VI \setminus \mathrm{vars}\bigl(\mathrm{rel}(VI, sh_1)\bigr)\\
&\iff \mathrm{vars}(t) \subseteq VI \setminus \mathrm{vars}\Bigl(\mathrm{rel}\bigl(VI, \rho_{PSD}(sh_1)\bigr)\Bigr)\\
&\iff \mathrm{vars}(t) \subseteq VI \setminus \mathrm{vars}\Bigl(\mathrm{rel}\bigl(VI, \rho_{PSD}(sh_2)\bigr)\Bigr)\\
&\iff \mathrm{vars}(t) \subseteq VI \setminus \mathrm{vars}\bigl(\mathrm{rel}(VI, sh_2)\bigr)\\
&\iff \mathrm{vars}(t) \subseteq VI \setminus \mathrm{vars}(sh_2)\\
&\iff \mathrm{ground}_{d_2}(t).
\end{aligned}$$

The proof of (6.94) follows from Definition 6.31, by applying the equivalences (6.91) and (6.93). Similarly, the proof of (6.95) follows from Definition 6.31 and (6.94). Finally, equation (6.96) follows from Definition 6.31 and Lemma 6.64. □

Lemma 6.67 *Let $sh_1, sh_2 \in SH$ be such that $sh_1 \subseteq \rho_{PSD}(sh_2)$. For each $V, W \subseteq VI$ and each $i \in \{1,2\}$, let also*

$$sh_{-,i} \stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}(V \cup W, sh_i), \qquad sh_{x,i} \stackrel{\mathrm{def}}{=} \mathrm{rel}(V, sh_i), \qquad sh_{t,i} \stackrel{\mathrm{def}}{=} \mathrm{rel}(W, sh_i).$$

Then, we have

$$\mathrm{bin}(sh_{x,1}, sh_{t,1}) \subseteq \rho_{PSD}\big(sh_{-,2} \cup \mathrm{bin}(sh_{x,2}, sh_{t,2})\big); \tag{6.97}$$
$$\mathrm{bin}(sh_{x,1}, sh^{\star}_{t,1}) \subseteq \rho_{PSD}\big(sh_{-,2} \cup \mathrm{bin}(sh_{x,2}, sh^{\star}_{t,2})\big). \tag{6.98}$$

Proof. Let $S \stackrel{\text{def}}{=} S_x \cup S_t \in \mathrm{bin}(sh_{x,1}, sh_{t,1})$ where $S_x \in sh_{x,1}$ and $S_t \in sh_{t,1}$. Consider an arbitrary variable $y \in S$.

Suppose first that $y \in S_x$ and let $w \in W \cap S_t$. Since $S_x, S_t \in sh_1$, by hypothesis $S_x, S_t \in \rho_{PSD}(sh_2)$. By Definition 5.10, $S = \bigcup(A \cup B)$, where

$$A \stackrel{\text{def}}{=} \big\{\, S' \in sh_2 \bigm| \{y\} \subseteq S' \subseteq S_x \,\big\}, \qquad B \stackrel{\text{def}}{=} \big\{\, S' \in sh_2 \bigm| \{w\} \subseteq S' \subseteq S_t \,\big\}.$$

In particular, we can write $A \cup B = sh'_{-} \cup sh'_x \cup sh'_t$, where

$$sh'_{-} \stackrel{\text{def}}{=} \overline{\mathrm{rel}}(V \cup W, A), \qquad sh'_x \stackrel{\text{def}}{=} \mathrm{rel}(V, A), \qquad sh'_t \stackrel{\text{def}}{=} \mathrm{rel}(W, A \cup B).$$

As $V \cap S_x \neq \varnothing$ and $v \in W \cap S_t$, $sh'_x \neq \varnothing$ and $sh'_t \neq \varnothing$. Thus $\bigcup(sh'_x \cup sh'_t) = \bigcup \mathrm{bin}(sh'_x, sh'_t)$, so that

$$S = \bigcup\big(sh'_{-} \cup \mathrm{bin}(sh'_x, sh'_t)\big).$$

By construction, we have $\{y\} \subseteq S' \subseteq S$ for all $S' \in A$. Thus, it also holds $\{y\} \subseteq S' \subseteq S$ for all $S' \in sh'_{-} \cup \mathrm{bin}(sh'_x, sh'_t)$, so that

$$S = \bigcup\big\{\, S' \in sh'_{-} \cup \mathrm{bin}(sh'_x, sh'_t) \bigm| \{y\} \subseteq S' \subseteq S \,\big\}.$$

By a symmetric argument, the same conclusion can be obtained when $y \in S_t$. As the choice of y was arbitrary, by Definition 5.10,

$$S \in \rho_{PSD}\big(sh'_{-} \cup \mathrm{bin}(sh'_x, sh'_t)\big). \tag{6.99}$$

Note that $sh'_{-} \subseteq sh_{-,2}$, $sh'_x \subseteq sh_{x,2}$ and $sh'_t \subseteq sh_{t,2}$, so that it holds

$$sh'_{-} \cup \mathrm{bin}(sh'_x, sh'_t) \subseteq sh_{-,2} \cup \mathrm{bin}(sh_{x,2}, sh_{t,2}).$$

Then, (6.97) follows from (6.99) by the monotonicity of ρ_{PSD}.

To prove (6.98), let $S \stackrel{\text{def}}{=} S_x \cup T_t \in \mathrm{bin}(sh_{x,1}, sh^{\star}_{t,1})$, where $S_x \in sh_{x,1}$ and $T_t \in sh^{\star}_{t,1}$. Consider an arbitrary variable $y \in S$.

Suppose that $y \in S_x$. Since $S_x \in sh_1$, by hypothesis $S_x \in \rho_{PSD}(sh_2)$, so that by Definition 5.10 we have $S = \bigcup\big(A \cup \{T_t\}\big)$, where

$$A \stackrel{\text{def}}{=} \big\{\, S' \in sh_2 \bigm| \{y\} \subseteq S' \subseteq S_x \,\big\}.$$

In particular, we can write $S = \bigcup(sh'_{-} \cup sh'_x \cup sh'_t)$, where

$$sh'_{-} \stackrel{\text{def}}{=} \overline{\text{rel}}(V \cup W, A), \qquad sh'_x \stackrel{\text{def}}{=} \text{rel}(V, A), \qquad sh'_t \stackrel{\text{def}}{=} \text{rel}(W, A) \cup \{T_t\}.$$

As $V \cap S_x \neq \varnothing$, $sh'_x \neq \varnothing$. As it also holds $sh'_t \neq \varnothing$, we have $\bigcup\bigl(sh'_x \cup sh'_t\bigr) = \bigcup \text{bin}\bigl(sh'_x, sh'_t\bigr)$, so that

$$S = \bigcup\bigl(sh'_{-} \cup \text{bin}(sh'_x, sh'_t)\bigr).$$

By construction, we have $\{y\} \subseteq S' \subseteq S$ for all $S' \in A$. Thus, it also holds $\{y\} \subseteq S' \subseteq S$ for all $S' \in sh'_{-} \cup \text{bin}(sh'_x, sh'_t)$, so that

$$S = \bigcup\bigl\{\, S' \in sh'_{-} \cup \text{bin}(sh'_x, sh'_t) \bigm| \{y\} \subseteq S' \subseteq S \,\bigr\}. \tag{6.100}$$

Suppose now that $y \in T_t$ and let $v \in V \cap S_x$. Since $S_x \in sh_1 \subseteq \rho_{PSD}(sh_2)$ we have, by Definition 5.10, $S = \bigcup\bigl(B \cup \{T_t\}\bigr)$, where

$$B \stackrel{\text{def}}{=} \bigl\{\, S' \in sh_2 \bigm| \{v\} \subseteq S' \subseteq S_x \,\bigr\}.$$

In particular, we can write $S = \bigcup(sh'_{-} \cup sh'_x \cup sh'_t)$, where

$$sh'_{-} \stackrel{\text{def}}{=} \varnothing, \qquad sh'_x \stackrel{\text{def}}{=} \text{rel}(V, B) = B, \qquad sh'_t \stackrel{\text{def}}{=} \{T_t\}.$$

As $v \in V \cap S_x$, $sh'_x \neq \varnothing$; as it also holds $sh'_t \neq \varnothing$, we have $\bigcup(sh'_x \cup sh'_t) = \bigcup\bigl(\text{bin}(sh'_x, sh'_t)\bigr)$. Since $y \in T_t$ and $sh'_{-} = \varnothing$, we have $\{y\} \subseteq S' \subseteq S$ for all $S' \in sh'_{-} \cup \text{bin}(sh'_x, sh'_t)$, so that (6.100) holds even in this case.

As the choice of y was arbitrary, by (6.100) and Definition 5.10,

$$S \in \rho_{PSD}\bigl(sh'_{-} \cup \text{bin}(sh'_x, sh'_t)\bigr). \tag{6.101}$$

Clearly, $sh'_{-} \subseteq sh_{-,2}$ and $sh'_x \subseteq sh_{x,2}$; also, by Lemma 4.15, $T_t \in sh^{\star}_{t,2}$, so that $sh'_t \subseteq sh^{\star}_{t,2}$. Thus, $sh'_{-} \cup \text{bin}(sh'_x, sh'_t) \subseteq sh_{-,2} \cup \text{bin}(sh_{x,2}, sh^{\star}_{t,2})$. The thesis (6.98) follows from (6.101) by the monotonicity of ρ_{PSD}. $\square$

Lemma 6.68 *Let $sh \in SH$ and $V, W \subseteq VI$. Let also*

$$sh_x \stackrel{\text{def}}{=} \text{rel}(V, sh), \qquad sh_{xt} \stackrel{\text{def}}{=} sh_x \cap sh_t,$$
$$sh_t \stackrel{\text{def}}{=} \text{rel}(W, sh), \qquad sh^{\diamond} \stackrel{\text{def}}{=} \text{bin}\bigl(sh_x \cup \text{bin}(sh_x, sh^{\star}_{xt}), sh_t \cup \text{bin}(sh_t, sh^{\star}_{xt})\bigr).$$

Then, $\rho_{PSD}(sh^{\diamond}) = \rho_{PSD}\bigl(\text{bin}(sh_x, sh_t)\bigr)$.

Proof. Observe that, since $sh_{xt} \subseteq sh_x$ and $sh_{xt} \subseteq sh_t$,

$$\text{bin}\bigl(\text{bin}(sh_x, sh^{\star}_{xt}), \text{bin}(sh_t, sh^{\star}_{xt})\bigr) = \text{bin}\bigl(\text{bin}(sh_x, sh_t), sh^{\star}_{xt}\bigr).$$

Therefore

$$sh^{\diamond} = \mathrm{bin}(sh_x, sh_t) \cup \mathrm{bin}\bigl(\mathrm{bin}(sh_x, sh_t), sh_{xt}^{\star}\bigr). \tag{6.102}$$

Thus, the inclusion $\rho_{PSD}(sh^{\diamond}) \supseteq \rho_{PSD}\bigl(\mathrm{bin}(sh_x, sh_t)\bigr)$ follows by the monotonicity of ρ_{PSD}. We now prove the other inclusion

$$\rho_{PSD}(sh^{\diamond}) \subseteq \rho_{PSD}\bigl(\mathrm{bin}(sh_x, sh_t)\bigr). \tag{6.103}$$

Let $S \in sh^{\diamond}$. Then, by (6.102), $S = S_x \cup S_t \cup T_{xt}$, where $S_x \in sh_x$, $S_t \in sh_t$, $T_{xt} \in sh_{xt}^{\star} \cup \varnothing$. Thus, for some $k \geq 0$, $T_{xt} = T_1 \cup \cdots \cup T_k$, where $T_i \in sh_{xt}$ for each $i = 1, \ldots, k$.

Consider an arbitrary variable $y \in S$. We will show that

$$S = \bigcup \bigl\{\, S' \in \mathrm{bin}(sh_x, sh_t) \bigm| \{y\} \subseteq S' \subseteq S \,\bigr\}. \tag{6.104}$$

Suppose first that $y \in S_x$. Then, since

$$S = (S_x \cup S_t) \cup (S_x \cup T_1) \cup \cdots \cup (S_x \cup T_k),$$

it follows that (6.104) holds. Similarly, (6.104) holds also when $y \in S_t$, by taking

$$S = (S_x \cup S_t) \cup (S_t \cup T_1) \cup \cdots \cup (S_t \cup T_k).$$

On the other hand, suppose now $y \notin S_x \cup S_t$, so that $k > 0$ and there exists $j \in \{1, \ldots, k\}$ such that $y \in T_j$. In this case, since $T_j \in sh_{xt}$, (6.104) holds by taking

$$S = (S_x \cup T_j) \cup (T_j \cup S_t) \cup (T_1 \cup T_j) \cup \cdots \cup (T_k \cup T_j),$$

As the choice of $y \in S$ is arbitrary, by (6.104) we have $S \in \rho_{PSD}\bigl(\mathrm{bin}(sh_x, sh_t)\bigr)$. Thus $sh^{\diamond} \subseteq \rho_{PSD}\bigl(\mathrm{bin}(sh_x, sh_t)\bigr)$ and (6.103) follows by the monotonicity and idempotence of ρ_{PSD}. □

Lemma 6.69 *Let $sh \in SH$ and $(x \mapsto t) \in Bind$, where $\{x\} \cup \mathrm{vars}(t) \subseteq VI$. Consider $W = \mathrm{vars}(t) \setminus \{x\}$ and let*

$$sh_x \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\{x\}, sh\bigr), \qquad sh_t \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr), \qquad sh_W \stackrel{\mathrm{def}}{=} \mathrm{rel}(W, sh).$$

Then

$$\mathrm{bin}(sh_x, sh_W) = \mathrm{cyclic}_x^t\bigl(\mathrm{bin}(sh_x, sh_t)\bigr); \tag{6.105}$$
$$\mathrm{bin}(sh_x^{\star}, sh_W) = \mathrm{cyclic}_x^t\bigl(\mathrm{bin}(sh_x^{\star}, sh_t)\bigr); \tag{6.106}$$
$$\mathrm{bin}(sh_x^{\star}, sh_W^{\star}) = \mathrm{cyclic}_x^t\bigl(\mathrm{bin}(sh_x^{\star}, sh_t^{\star})\bigr). \tag{6.107}$$

Proof. We start by proving equations (6.105) and (6.106) at the same time. Therefore, let $sh_x' \in \{sh_x, sh_x^{\star}\}$.

To prove the first two inclusions ($\subseteq$), we assume $S \in \mathrm{bin}(sh'_x, sh_W)$ and show that $S \in \mathrm{cyclic}^t_x\bigl(\mathrm{bin}(sh'_x, sh_t)\bigr)$. Since $sh_W \subseteq sh_t$, we have $S \in \mathrm{bin}(sh'_x, sh_t)$. Moreover, we have $S = S_x \cup S_W$, where $x \in S_x$ and $W \cap S_W \neq \varnothing$. Thus $W \cap S \neq \varnothing$ and hence, by Definition 6.31, $S \in \mathrm{cyclic}^t_x\bigl(\mathrm{bin}(sh'_x, sh_t)\bigr)$.

To prove the opposite inclusions ($\supseteq$), let $S \in \mathrm{cyclic}^t_x\bigl(\mathrm{bin}(sh'_x, sh_t)\bigr)$. Then, we have $S \in \mathrm{bin}(sh'_x, sh_t)$, so that $S = S_x \cup S_t$, where $S_x \in sh'_x$ and $S_t \in sh_t$. Thus $x \in S$ and, by Definition 6.31, $S \in \mathrm{rel}(W, sh)$. If $\mathrm{vars}(t) \cap S_t \neq \{x\}$, then $S_t \in sh_W$ and $S \in \mathrm{bin}(sh'_x, sh_W)$, so that both inclusions hold. Otherwise, let $\mathrm{vars}(t) \cap S_t = \{x\}$, so that $S_t \in sh'_x$ and $S_t \notin sh_W$. Since we know $S \in \mathrm{rel}(W, sh)$, there exists $w \in W \cap S_x$, so that $S_x \in sh_W$. First, consider the case when $sh'_x = sh_x$. Then $S_t \in sh_x$ and we have $S \in \mathrm{bin}(sh_x, sh_W)$, proving the second inclusion for (6.105). Secondly, consider the case when $sh'_x = sh^\star_x$. Then we can write $S_x = S_W \cup S_x$, where $S_W \in sh_W$. Thus $S = (S_x \cup S_t) \cup S_W \in \mathrm{bin}(sh^\star_x, sh_W)$, proving the second inclusion for (6.106).

Finally, we prove equation (6.107).

To prove the first inclusion ($\subseteq$), we now assume $S \in \mathrm{bin}(sh^\star_x, sh^\star_W)$ and show that $S \in \mathrm{cyclic}^t_x\bigl(\mathrm{bin}(sh^\star_x, sh^\star_t)\bigr)$. As $sh_W \subseteq sh_t$, we have $S \in \mathrm{bin}(sh^\star_x, sh^\star_t)$. Moreover, we have $S = S_x \cup S_W$, where $x \in S_x$ and $W \cap S_W \neq \varnothing$. Thus $W \cap S \neq \varnothing$ and hence, by Definition 6.31, $S \in \mathrm{cyclic}^t_x\bigl(\mathrm{bin}(sh^\star_x, sh^\star_t)\bigr)$.

To prove the opposite inclusion ($\supseteq$), let $S \in \mathrm{cyclic}^t_x\bigl(\mathrm{bin}(sh^\star_x, sh^\star_t)\bigr)$. Then, we have $S \in \mathrm{bin}(sh^\star_x, sh^\star_t)$, so that $S = S_x \cup S_t$, where $S_x \in sh^\star_x$ and $S_t \in sh^\star_t$. Thus $x \in S$ and, by Definition 6.31, $S \in \mathrm{rel}(W, sh)$. Suppose first that $\mathrm{vars}(t) \cap S_t \neq \{x\}$. Then $S_t = S_W \cup S_{xt}$, where $S_W \in sh^\star_W$ and $S_{xt} \in sh^\star_x \cup \{\varnothing\}$. Thus $S = (S_x \cup S_{xt}) \cup S_W \in \mathrm{bin}(sh^\star_x, sh^\star_W)$. Suppose next that $\mathrm{vars}(t) \cap S_t = \{x\}$, so that $S_t \in sh^\star_x$ $W \cap S_x \neq \varnothing$. Then $S_x = S_W \cup S_x$, where $S_W \in sh^\star_W$. Thus $S = (S_x \cup S_t) \cup S_W \in \mathrm{bin}(sh^\star_x, sh^\star_W)$, completing the proof. $\square$

Theorem 6.70 *Let $d_1, d_2 \in$ SFL and $(x \mapsto t) \in$ Bind, where $\rho_{PSD}(d_1) = \rho_{PSD}(d_2)$. Then*

$$\rho_{PSD}\bigl(\mathrm{amgu}_S(d_1, x \mapsto t)\bigr) = \rho_{PSD}\bigl(\mathrm{amgu}_S(d_2, x \mapsto t)\bigr).$$

Proof. Let $d_1 = \langle sh_1, f, l\rangle$. Then, by definition of ρ_{PSD} on *SFL*, it holds $d_2 = \langle sh_2, f, l\rangle$, where $\rho_{PSD}(sh_1) = \rho_{PSD}(sh_2)$.

For each $i \in \{1, 2\}$, let $\langle sh'_i, f'_i, l'_i\rangle = \mathrm{amgu}_S\bigl(d_i, x \mapsto t\bigr)$. We will prove the following:

$$\rho_{PSD}(sh'_1) = \rho_{PSD}(sh'_2), \tag{6.108}$$
$$f'_1 = f'_2, \tag{6.109}$$
$$l'_1 = l'_2. \tag{6.110}$$

Equation (6.108). We follow the same reasoning of Lemma 4.16, proving the result

$$sh_1 \subseteq \rho_{PSD}(sh_2) \implies sh'_1 \subseteq \rho_{PSD}(sh'_2). \tag{6.111}$$

Then, by using (4.8), we obtain

$$\rho_{PSD}(sh_1) \subseteq \rho_{PSD}(sh_2) \implies \rho_{PSD}(sh'_1) \subseteq \rho_{PSD}(sh'_2),$$

from which the thesis follows by symmetry.

Let $W = \operatorname{vars}(t) \setminus \{x\}$ and, for each $i \in \{1, 2\}$,

$$\begin{aligned} sh_{-,i} &= \overline{\operatorname{rel}}\bigl(\{x\} \cup \operatorname{vars}(t), sh_i\bigr), & sh_{W,i} &= \operatorname{rel}(W, sh_i), \\ sh_{x,i} &= \operatorname{rel}\bigl(\{x\}, sh_i\bigr), & sh_{xt,i} &= sh_{x,i} \cap sh_{t,i}, \\ sh_{t,i} &= \operatorname{rel}\bigl(\operatorname{vars}(t), sh_i\bigr). \end{aligned}$$

To prove (6.111), assume that $sh_1 \subseteq \rho_{PSD}(sh_2)$. By Definitions 6.33 and 6.31, for each $i = 1$, 2 we have

$$sh'_i = \operatorname{cyclic}_x^t(sh_{-,i} \cup sh''_i) = sh_{-,i} \cup \operatorname{cyclic}_x^t(sh''_i).$$

We first show that $sh_{-,1} \subseteq \rho_{PSD}\bigl(sh_{-,2} \cup \operatorname{cyclic}_x^t(sh''_2)\bigr)$. By the definition of $sh_{-,1}$, the assumption and the monotonicity of $\overline{\operatorname{rel}}$, we have

$$sh_{-,1} \subseteq \overline{\operatorname{rel}}\bigl(\{x\} \cup \operatorname{vars}(t), \rho_{PSD}(sh_2)\bigr).$$

Thus, by Lemma 4.14, $sh_{-,1} \subseteq \rho_{PSD}(sh_{-,2})$, from which the required result follows by monotonicity of ρ_{PSD}.

We next show that $\operatorname{cyclic}_x^t(sh''_1) \subseteq \rho_{PSD}\bigl(sh_{-,2} \cup \operatorname{cyclic}_x^t(sh''_2)\bigr)$. By applying cases (6.92) and (6.95) of Lemma 6.66, it can be seen that sh''_1 and sh''_2 are each computed by selecting the same alternative branch of Definition 6.33. We have five cases.

1. In the first case, for each $i = 1$, 2, we have $sh''_i = \operatorname{bin}(sh_{x,i}, sh_{t,i})$. By case (6.105) of Lemma 6.69, $\operatorname{cyclic}_x^t(sh''_i) = \operatorname{bin}(sh_{x,i}, sh_{W,i})$, for each $i = 1$, 2. Thus, by case (6.97) of Lemma 6.67, where we take $V = \{x\}$,

$$\operatorname{bin}(sh_{x,1}, sh_{W,1}) \subseteq \rho_{PSD}\bigl(sh_{-,2} \cup \operatorname{bin}(sh_{x,2}, sh_{W,2})\bigr),$$

   from which the thesis follows.

2. In the second case we have, for each $i = 1$, 2,

$$sh''_i = \operatorname{bin}\bigl(sh_{x,i} \cup \operatorname{bin}(sh_{x,i}, sh^\star_{xt,i}), sh_{t,i} \cup \operatorname{bin}(sh_{t,i}, sh^\star_{xt,i})\bigr).$$

   There are two cases.

   First suppose that $x \notin \operatorname{vars}(t)$, so that $\operatorname{cyclic}_x^t(sh''_i) = sh''_i$. Then, by Lemma 6.68, for each $i = 1$, 2, we have $sh''_i \subseteq \rho_{PSD}\bigl(\operatorname{bin}(sh_{x,i}, sh_{t,i})\bigr)$. Therefore, by case (6.97) of Lemma 6.67 and the monotonicity of ρ_{PSD}, we obtain

$$sh''_1 \subseteq \rho_{PSD}\bigl(sh_{-,2} \cup \operatorname{bin}(sh_{x,2}, sh_{t,2})\bigr),$$

so that the thesis holds.

Secondly, suppose that $x \in \operatorname{vars}(t)$. Thus, for each $i = 1, 2$, we have $sh_{xt,i} = sh_{x,i}$, so that $sh''_i = \operatorname{bin}(sh^\star_{x,i}, sh_{t,i})$. This case is therefore equivalent to the third case, proven below.

3. In the third case, for each $i = 1, 2$, we have $sh''_i = \operatorname{bin}(sh^\star_{x,i}, sh_{t,i})$. By case (6.106) of Lemma 6.69, $\operatorname{cyclic}^t_x(sh''_i) = \operatorname{bin}(sh^\star_{x,i}, sh_{W,i})$. Thus, by case (6.98) of Lemma 6.67, where we take $V = \{x\}$, we obtain

$$sh''_1 \subseteq \rho_{PSD}\bigl(sh_{-,2} \cup \operatorname{bin}(sh^\star_{x,2}, sh_{W,2})\bigr),$$

so that the thesis holds.

4. In the fourth case, for each $i = 1, 2$, we have $sh''_i = \operatorname{bin}(sh_{x,i}, sh^\star_{t,i})$. Moreover, as $\operatorname{lin}_d(t)$ holds and $\operatorname{lin}_d(x)$ does not hold, we can assume that $x \notin \operatorname{vars}(t)$, so that $\operatorname{cyclic}^t_x(sh''_i) = sh''_i$. Thus, by case (6.98) of Lemma 6.67, where we exchange the usual roles of V and W, we obtain

$$sh''_1 \subseteq \rho_{PSD}\bigl(sh_{-,2} \cup \operatorname{bin}(sh_{x,2}, sh^\star_{t,2})\bigr),$$

so that the thesis holds.

5. In the fifth case we have, for $i = 1, 2$, $sh''_i = \operatorname{bin}(sh^\star_{x,i}, sh^\star_{t,i})$. By case (6.107) of Lemma 6.69, $\operatorname{cyclic}^t_x(sh''_i) = \operatorname{bin}(sh^\star_{x,i}, sh^\star_{W,i})$. The thesis follows from Lemma 4.16, by replacing the term t by an arbitrary term $t' \in \mathit{HTerms}$ such that $\operatorname{vars}(t') = W$.

Equation (6.109). Consider the computation of f'_i as specified in Definition 6.33. By applying case (6.92) of Lemma 6.66, it can be seen that f'_1 and f'_2 are computed by selecting the same alternative branch. The thesis $f'_1 = f'_2$ thus follows from case (6.96) of Lemma 6.66.

Equation (6.110). Consider the computation of l'_i as specified in Definition 6.33: for each $i \in \{1, 2\}$ we have

$$l'_i = \bigl(\mathit{VI} \setminus \operatorname{vars}(sh'_i)\bigr) \cup f'_i \cup l''_i.$$

Let $r \in \mathit{HTerms}$, where $\operatorname{vars}(r) = \mathit{VI}$. For each $i \in \{1, 2\}$, $\operatorname{vars}(sh'_i) = \operatorname{share\_with}_{d'_i}(r)$; also, by equation (6.108), we know that $\rho_{PSD}(sh'_1) = \rho_{PSD}(sh'_2)$; thus, by case (6.96) of Lemma 6.66, we obtain $\operatorname{vars}(sh'_1) = \operatorname{vars}(sh'_2)$. By equation (6.109), we also know that $f'_1 = f'_2$. Therefore, to complete the proof, we only need to prove that $l''_1 = l''_2$. Consider the computation of l''_i as specified in Definition 6.33. By case (6.95) of Lemma 6.66, it can be seen that, in the computations of l''_1 and l''_2, the same alternative branch is selected. Hence, the thesis is obtained by applying case (6.96) of Lemma 6.66. □

Theorem 6.71 *Let $d_1, d_2 \in \mathit{SFL}$ and $bs \in \mathit{Bind}^\star$, where $\rho_{PSD}(d_1) = \rho_{PSD}(d_2)$. Then,*

$$\rho_{PSD}\bigl(\operatorname{aunify}_S(d_1, bs)\bigr) = \rho_{PSD}\bigl(\operatorname{aunify}_S(d_2, bs)\bigr).$$

Proof. The proof is by induction on the length of the sequence of bindings bs. The base case, when $|bs| = 0$ and thus $bs = \epsilon$, is obvious from the definition of aunify_S. For the inductive case, when $|bs| = m > 0$, let $bs = (x \mapsto t) \,.\, bs'$. By the hypothesis and Theorem 6.70, we have

$$\rho_{PSD}\bigl(\mathrm{amgu}_S(d_1, x \mapsto t)\bigr) = \rho_{PSD}\bigl(\mathrm{amgu}_S(d_2, x \mapsto t)\bigr). \tag{6.112}$$

Moreover, for each $i \in \{1, 2\}$, by definition of aunify_S we have

$$\mathrm{aunify}_S(d_i, bs) = \mathrm{aunify}_S\bigl(\mathrm{amgu}_S(d_i, x \mapsto t), bs'\bigr).$$

Thus, by (6.112), we can apply the inductive hypothesis and conclude the proof, since $|bs'| = m - 1 < m$. □

Theorem 6.72 *Let $d_1, d_2 \in SH$ and $V \subseteq VI$, where $\rho_{PSD}(d_1) = \rho_{PSD}(d_2)$. Then,*

$$\rho_{PSD}\bigl(\mathrm{aexists}_S(d_1, V)\bigr) = \rho_{PSD}\bigl(\mathrm{aexists}_S(d_2, V)\bigr).$$

Proof. Let $d_i = \langle sh_i, f_i, l_i \rangle$, for each $i = 1, 2$. By applying Definitions 6.38 and 6.60, for each $i = 1, 2$, we have

$$\begin{aligned}
\rho_{PSD}\bigl(\mathrm{aexists}_S(d_i, V)\bigr) &= \rho_{PSD}\Bigl(\bigl\langle \mathrm{aexists}(sh_i), f_i \cup V, l_i \cup V \bigr\rangle\Bigr) \\
&= \Bigl\langle \rho_{PSD}\bigl(\mathrm{aexists}(sh_i)\bigr), f_i \cup V, l_i \cup V \Bigr\rangle.
\end{aligned}$$

By the hypothesis and Definition 6.60, we also have $\rho_{PSD}(sh_1) = \rho_{PSD}(sh_2)$, $f_1 = f_2$ and $l_1 = l_2$. Thus, to complete the proof, we only need to show that

$$\rho_{PSD}\bigl(\mathrm{aexists}(sh_1)\bigr) = \rho_{PSD}\bigl(\mathrm{aexists}(sh_2)\bigr).$$

This follows from Lemma 5.26, taking $k = 2$. □

Proof of Theorem 6.61 on page 180. The congruence properties for aunify_S and $\mathrm{aexists}_S$ follow from Theorems 6.71 and 6.72, respectively. The congruence property for alub_S holds, as usual, because ρ_{PSD} is an upper closure operator. □

Proof of Theorem 6.62 on page 181. Suppose $\rho_{PSD}(d_1) \neq \rho_{PSD}(d_2)$. By Definition 6.60, we have three cases:

1. Suppose $\rho_{PSD}(sh_1) \neq \rho_{PSD}(sh_2)$. The proof for this case is the same as that given for Theorem 5.20, where we take $k = 2$ and we regard σ, the substitution constructed in that proof, as the sequence bs of its bindings. Since σ binds all of its domain variables to terms that are ground and finite, then no binary union and/or star-union needs to be computed. As a consequence, the behavior of amgu_S on the sharing component is the same as the behavior of 'amgu'.

2. Suppose now $f_1 \neq f_2$. In this case, by taking $\rho = \rho_F$ and $bs = \epsilon$, we obtain

$$\begin{aligned}
\rho_F\bigl(\mathrm{aunify}_S(d_1, \epsilon)\bigr) &= \rho_F(d_1) \\
&= \langle SG, f_1, \varnothing \rangle \\
&\neq \langle SG, f_2, \varnothing \rangle \\
&= \rho_F(d_2) \\
&= \rho_F\bigl(\mathrm{aunify}_S(d_2, \epsilon)\bigr).
\end{aligned}$$

3. Finally, suppose $l_1 \neq l_2$. Similarly to the previous case, by taking $\rho = \rho_L$ and $bs = \epsilon$, we obtain

$$\begin{aligned}
\rho_L\bigl(\mathrm{aunify}_S(d_1, \epsilon)\bigr) &= \rho_L(d_1) \\
&= \langle SG, \varnothing, l_1 \rangle \\
&\neq \langle SG, \varnothing, l_2 \rangle \\
&= \rho_L(d_2) \\
&= \rho_L\bigl(\mathrm{aunify}_S(d_2, \epsilon)\bigr).
\end{aligned}$$

□

Proof of Theorem 6.63 on page 181. Suppose first that $x \notin \mathrm{vars}(t)$. Then it holds

$$\mathrm{cyclic}_x^t(sh_- \cup sh^\diamond) = sh_- \cup sh^\diamond,$$

so that the thesis is a corollary of Lemma 6.68, where $V = \{x\}$ and $W = \mathrm{vars}(t)$.

Suppose now $x \in \mathrm{vars}(t)$. Then we have $sh_x = sh_{xt}$, so that $sh^\diamond = \mathrm{bin}(sh_x^\star, sh_t)$. In this case the thesis is a corollary of Theorem 4.12. □

6.7 Summary

In this chapter we have introduced the abstract domain *SFL*, combining the set-sharing domain *SH* with freeness and linearity information. While the carrier of *SFL* can be considered standard, we have provided the specification of a new abstract unification operator, showing examples where this achieves more precision than the operators used in the classical proposals. However, from both the theoretical and the practical points of view, the main contributions are the following:

- we have defined a precise abstraction function, mapping arbitrary substitutions in rational solved form into their most precise approximation on *SFL*;
- using this abstraction function, we have provided the mandatory proof of correctness for the new abstract unification operator, for both finite-tree and rational-tree languages;

- we have shown that, in the definition of *SFL*, we can replace the set-sharing domain *SH* by its non-redundant version *PSD*. As a consequence, it is possible to implement an algorithm for abstract unification running in polynomial time and still obtain the same precision on all the considered observables, that is groundness, independence, freeness and linearity.

Chapter 7

Experimental Evaluation

In this chapter we describe the experimental work we have conducted in order to validate, from a practical point of view, the theoretical results achieved in the previous chapters. After a brief description of our implementation of the *SFL* domain and of the corresponding non-redundant version SFL_2, we present the results obtained by comparing the two domains. These confirm that the efficiency of the analysis is greatly improved, while the precision, as predicted by the theory, is not affected at all.

7.1 The China Analyzer

The ideas presented in this thesis have been experimentally validated in the context of the development of the China analyzer [Bag97a]. China is a data-flow analyzer for CLP($\mathcal{H}_\mathcal{N}$) languages (i.e., Prolog, CLP($\mathcal{R}$), `clp(FD)` and so forth), $\mathcal{H}_\mathcal{N}$ being an extended Herbrand system where the values of a numeric domain $\mathcal{N}$ can occur as leaves of the terms. China, which is written in C++, performs bottom-up analysis deriving information on both call-patterns and success-patterns by means of program transformations and optimized fixpoint computation techniques.

Thanks to a careful modular design, the China analyzer is a generic data-flow analysis tool for constraint logic programs: besides groundness, aliasing, freeness and linearity, there are analyses capturing compoundness, term sizes, term finiteness, polymorphic types, numerical bounds and relations. In most cases, several abstract domains have been implemented for the analysis of the same combination of observables, therefore allowing for an *homogeneous* comparison of the different proposals, on both precision and performance. This is an important point, since a fair comparison between analyses performed on different data-flow systems is difficult, if not impossible, to achieve. First of all, there is no standard way of measuring the precision of the analysis. Moreover, in many cases the prototype analyzers are based on compromising assumptions on the programs being analysed, in particular when dealing with the built-ins of the language. Sometimes the logic program under investigation has to undergo a source-to-source translation phase before the real analysis: it is often the case that these pre-processing phases do not preserve the semantics of the original program. Things are even worse for performance comparisons, since in this case too many factors (hardware, operating system, compiler, ...) have a direct influence on the result of the comparison.

The CHINA analyzer keeps evolving, both by the addition of new features to the domain-independent framework and to the currently implemented abstract domains, as well as by the implementation and integration of new abstract domains. At the time of writing, the complete analyzer consists of more than 2.5 MB of source code (100 K lines): of these, 27% is pertinent to the domain-independent framework, while the modules strictly related to the implementation of the sharing analysis domains amount to another 13%, including all the variants that will be presented in Chapters 8 and 9.

A major point of the experimental evaluation is given by the test-suite, which is probably the largest one ever reported in the literature on data-flow analysis of (constraint) logic programs. The suite comprises all the programs we have access to, i.e., everything we could find by systematically dredging the Internet: more than 370 programs which, *once blank lines and comments have been stripped out*, amounts to more than 23 MB of code (700 K lines), distributed over 276 K clauses. Besides classical benchmarks, several real programs of respectable size are present: using code size as a measure, 38 programs are above 100 KB, with four of them above 1 MB; using the number of clauses, 45 programs are above 1 K clauses, with three of them above 10 K. The suite also includes a few synthetic benchmarks, which are small, artificial programs explicitly constructed to stress the capabilities of the analyzer and of its abstract domains with respect to precision and/or efficiency. On average, a benchmark is made up of 63 KB of code, distributed over 742 clauses. However, due to the huge differences among the programs in the suite, averages are not so meaningful. More insight can be obtained by computing *median* values: thus, one half of the programs in our benchmark suite have a size greater than 6 KB (resp., have more than 74 clauses).

7.2 The Implementation of Set-Sharing

At the implementation level, each variable is identified with a non-negative integer. Thus variables inherit from the integers the usual total ordering relation. Finite sets of variables are represented by dynamically resizing bit-vectors. Sharing sets, that is sets of sets of variables, are implemented by means of the `set` associative container provided by standard C++. The total ordering relation employed for this purpose, $< \subseteq \wp_{\mathrm{f}}(\mathbb{N}_0) \times \wp_{\mathrm{f}}(\mathbb{N}_0)$, is an extension of the $\subset$ partial ordering. In other words, for each $S_1, S_2 \in \wp_{\mathrm{f}}(\mathbb{N}_0)$, if $S_1 \subset S_2$ then $S_1 < S_2$. This ordering is exploited in several places in the implementation and proved to be a very effective device.

It is important to remark that the implementation of *SH* we use for comparison against *PSD* is a rather refined one. In particular, care was taken in the implementation of star-union. The algorithm we used is given in Figure 7.1. The optimization implemented by lines 2, 4, and 5 avoids the computation of redundant unions, and can give rise to efficiency gains of an order of magnitude and more. Suppose $sh = \{S_1, \ldots, S_n\}$, $i \in \{1, \ldots, n\}$,

Require: the sharing set $sh \stackrel{\text{def}}{=} \{S_1, \ldots, S_n\}$ such that $S_1 < \cdots < S_n$.
Ensure: on exit $sh_{\text{star}} = sh^\star$.

1: $sh_{\text{star}} := \varnothing$
2: $sh_{\text{done}} := \varnothing$
3: **for** $i := 1$ to n **do**
4:   **if** $S_i \neq \bigcup\{\, T \in sh_{\text{done}} \mid T \subset S_i \,\}$ **then**
5:     $sh_{\text{done}} := sh_{\text{done}} \cup \{S_i\}$
6:     $sh_{\text{star}} := sh_{\text{star}} \cup \{S_i\} \cup \{\, S_i \cup U \mid U \in sh_{\text{star}} \,\}$
7:   **end if**
8: **end for**

Figure 7.1: An optimized algorithm for computing star-union.

$J \subset \{1, \ldots, n\}$ with $i \notin J$, and $S_i = \bigcup_{j \in J} S_j$. Then $sh^\star = \bigl(sh \setminus \{S_i\}\bigr)^\star$. Observe how the total ordering used for representing sharing sets simplifies the task of checking the applicability condition for this optimization (line 4 of the algorithm in Figure 7.1). In fact, in order to have $S_i = \bigcup_{j \in J} S_j$ and $i \notin J$ we must have $\forall j \in J : S_j \subset S_i$ and thus $\forall j \in J : S_j < S_i$.

Theorem 7.1 *On exit from the algorithm of* Figure 7.1, $sh_{\text{star}} = sh^\star$.

The proof of the above result relies on the following two lemmas.

Lemma 7.2 *Let* $sh = sh_1 \cup \{S\}$. *Then* $sh^\star = sh_1^\star \cup \{S\} \cup \{\, S \cup T \mid T \in sh_1^\star \,\}$.

Proof. We start by proving that $sh^\star \supseteq sh_1^\star \cup \{S\} \cup \{\, S \cup T \mid T \in sh_1^\star \,\}$. By monotonicity of $(\cdot)^\star$ we have $sh_1^\star \subseteq sh^\star$, whereas, by extensivity of $(\cdot)^\star$, we have $\{S\} \subseteq sh^\star$. Let $T \in sh_1^\star$. Then, by definition of $(\cdot)^\star$, there exist $S_1, \ldots, S_n \in sh_1$ such that $T = S_1 \cup \cdots \cup S_n$. As a consequence, we obtain $S \cup T = S \cup S_1 \cup \cdots \cup S_n \in sh^\star$.

We now prove that $sh^\star \subseteq sh_1^\star \cup \{S\} \cup \{\, S \cup T \mid T \in sh_1^\star \,\}$. Let $S' \in sh^\star$. Then there exist $S_1, \ldots, S_n \in sh$ such that $S' = S_1 \cup \cdots \cup S_n$.

Suppose first $n = 1$ and $S_1 = S$. Then $S' = S$.

Suppose now $n > 1$ and there exists $i \in \{1, \ldots, n\}$ such that $S_i = S$. Then, if we let

$$T = \bigcup_{\substack{j=1 \\ j \neq i}}^{n} S_j,$$

we have $S' = S \cup T$ and $T \in sh_1^\star$, thus $S' \in \{\, S \cup T \mid T \in sh_1^\star \,\}$.

Finally, if $\nexists i \in \{1, \ldots, n\} \;.\; S_i = S$, then $S' \in sh_1^\star$. $\square$

Lemma 7.3 *Let* $sh_k = \{S_1, \ldots, S_k\}$ *for* $k = 1, \ldots, n$. *Let also* $sh_{\text{star},k}$ *and* $sh_{\text{done},k}$ *denote the values of the variables* sh_{star} *and* sh_{done}, *respectively, just after the* k*-th evaluation of the* **if** *statement (i.e., after line 7) in the algorithm of Figure 7.1. Then* $sh_k^\star \subseteq sh_{\text{star},k}$ *and* $sh_{\text{done},k} \subseteq sh_k$.

Proof. We reason by induction on k. For the case where $k = 1$ we have

$$sh_k = \{S_1\} = sh_k^\star = sh_{\mathrm{star},k} = sh_{\mathrm{done},k}.$$

Next assume $1 < k \leq n$. Then $sh_k = sh_{k-1} \cup \{S_k\}$.

Suppose $S_k \neq \bigcup\{\, S_j \in sh_{k-1} \mid S_j \subset S_k \,\}$. By the inductive hypothesis, since $sh_{\mathrm{done},k-1} \subseteq sh_{k-1}$, this implies that the **if** condition in line 4 evaluates to *true* and lines 5 and 6 are executed. Clearly, after the execution of line 5, $sh_{\mathrm{done},k} \subseteq sh_{k-1} \cup \{S_k\} = sh_k$. Moreover, after execution of line 6, $sh_{\mathrm{star},k} = sh_{\mathrm{star},k-1} \cup \{S_k\} \cup \{\, S_k \cup T \mid T \in sh_{\mathrm{star},k-1} \,\}$. By the inductive hypothesis, this implies

$$sh_{\mathrm{star},k} \supseteq sh_{k-1}^\star \cup \{S_k\} \cup \{\, S_k \cup T \mid T \in sh_{k-1}^\star \,\}.$$

Thus, by Lemma 7.2, $sh_k^\star \subseteq sh_{\mathrm{star},k}$.

Suppose now $S_k = \bigcup\{\, S_j \in sh_{k-1} \mid S_j \subset S_k \,\}$. Then, we have $sh_{\mathrm{done},k} = sh_{\mathrm{done},k-1}$. By the inductive hypothesis, $sh_{\mathrm{done},k-1} \subseteq sh_{k-1}$. Hence $sh_{\mathrm{done},k} \subseteq sh_k$. We now show that $sh_k^\star = sh_{k-1}^\star$. Clearly, by monotonicity of $(\cdot)^\star$ we have $sh_k^\star \supseteq sh_{k-1}^\star$. Assume now $T \in sh_k^\star$, i.e., $T = T_1 \cup \cdots \cup T_m$ where $T_i \in sh_k$ for $i = 1, \ldots, m$. If for no $j \in \{1, \ldots, m\}$ we have $T_j = S_k$, then $T \in sh_{k-1}^\star$. On the other hand, if there exists $j \in \{1, \ldots, m\}$ such that $T_j = S_k$, then

$$\begin{aligned} T &= T_1 \cup \cdots \cup T_{j-1} \cup T_j \cup T_{j+1} \cup \cdots \cup T_m \\ &= T_1 \cup \cdots \cup T_{j-1} \cup \bigcup\{\, S_j \in sh_{k-1} \mid S_j \subset S_k \,\} \cup T_{j+1} \cup \cdots \cup T_m \\ &\in sh_{k-1}^\star. \end{aligned}$$

The inductive hypothesis implies that $sh_{\mathrm{star},k-1} \supseteq sh_{k-1}^\star = sh_k^\star$. This concludes the proof, as the algorithm never removes elements from sh_{star}, so that $sh_{\mathrm{star},k} \supseteq sh_{\mathrm{star},k-1}$. □

Proof of Theorem 7.1 on the page before. An invariant of the algorithm is that if $S \in sh_{\mathrm{star}}$ then $S = \bigcup_{i \in I} S_i$ for some $I \subseteq \{1, \ldots, n\}$ such that $I \neq \varnothing$. The invariant is established in line 1 and preserved by any step. In particular, line 6, which is the only one to change sh_{star}, maintains the invariant. Thus, at the end of the algorithm, if S is an element of sh_{star} then $S \in sh^\star$, hence $sh_{star} \subseteq sh^\star$. The reverse inclusion is trivially satisfied for $sh = \varnothing$, while it is proven by Lemma 7.3 otherwise. □

As far as the implementation of *PSD* is concerned, the code for all the abstract operations can be reused, once star-union has been replaced by 2-self-union. This does not mean that it is not possible to produce sharing sets with less redundant sharing groups. For instance, consider the amgu operation applied to a sharing set sh and a binding $x \mapsto t$.[1]

[1] We consider the operator $\mathrm{amgu}\colon SH \times Bind \to SH$ for presentation purposes only: the same reasoning applies, as is, to the evaluation of the set-sharing component of $\mathrm{amgu}_S\colon SFL \times Bind \to SFL$, when the 2-self-union operations have to be computed.

Then, Theorem 4.12 tells us that $\mathrm{amgu}(sh, x \mapsto t)$ is equivalent to

$$sh_{-} \cup \mathrm{bin}(sh_x^2, sh_t^2), \tag{7.1}$$

where sh_{-}, sh_x and sh_t are defined as usual. We will show that expression (7.1), even though, in general, produces less redundant sharing groups than $\mathrm{amgu}(sh, x \mapsto t)$, it is not optimal in this respect. Let us define

$$sh_{x-} \stackrel{\mathrm{def}}{=} sh_x \setminus sh_t, \qquad sh_{t-} \stackrel{\mathrm{def}}{=} sh_t \setminus sh_x, \qquad sh_{xt} \stackrel{\mathrm{def}}{=} sh_x \cap sh_t.$$

Thus $sh_x = sh_{x-} \cup sh_{xt}$ and $sh_t = sh_{t-} \cup sh_{xt}$. For $sh_1, sh_2 \in SH$, we have

$$(sh_1 \cup sh_2)^2 = sh_1^2 \cup sh_2^2 \cup \mathrm{bin}(sh_1, sh_2).$$

So, the expansion of expression (7.1) looks like

$$\begin{aligned}
\cdots \cup \mathrm{bin}\bigl(sh_x^2, sh_t^2\bigr) &= \cdots \cup \mathrm{bin}\bigl((sh_{x-} \cup sh_{xt})^2, (sh_{t-} \cup sh_{xt})^2\bigr) \\
&= \cdots \cup \mathrm{bin}(\cdots \cup sh_{xt}^2, \cdots \cup sh_{xt}^2) \\
&= \cdots \cup \cdots \cup sh_{xt}^4.
\end{aligned}$$

However, it is straightforward to show that, for each $n \geq 2$, $\rho_{PSD}(sh^n) = \rho_{PSD}(sh^2)$ and $sh^n \supseteq sh^2$. It is thus clear that expression (7.1), by computing sh_{xt}^4, may introduce (and, in fact, often introduces) redundant sharing groups. It can be proved that a better way to compute a sharing set in the same equivalence class of $\mathrm{amgu}(sh, x \mapsto t)$ is given by the following expression:

$$sh_{-} \cup \bigl(\mathrm{bin}(sh_x, sh_{t-}) \cup \mathrm{bin}(sh_{x-}, sh_{xt})\bigr)^2 \cup sh_{xt}^2. \tag{7.2}$$

Theoretically speaking, expression (7.2) is undoubtedly better than expression (7.1). However, we were unable to obtain a competitive implementation of the amgu operation based on expression (7.2): the implementation relying on expression (7.1) followed by the elimination of redundant sharing groups was more efficient. Thus, the question whether even more efficient abstract operations can be obtained remains open.

For the choice of the elements representing the equivalence classes based on ρ_{PSD}, we have adopted the dynamic approach discussed in Section 4.5. In particular, reduction is performed after each binary union operation, at the end of each clause evaluation, and during the equivalence check.

The algorithm for removing redundant sharing groups is fairly easy. The only important observation is that redundant sharing groups can be removed in any order. In fact, suppose two sharing groups $S, T \in sh$ are redundant for sh and let $sh_1 \stackrel{\mathrm{def}}{=} sh \setminus \{S\}$ and $sh_2 \stackrel{\mathrm{def}}{=} sh \setminus \{T\}$. Then $\rho_{PSD}(sh_1) = \rho_{PSD}(sh_2) = \rho_{PSD}(sh)$ and, by applying Theorem 4.9, we obtain $\rho_{PSD}(sh_1 \cap sh_2) = \rho_{PSD}\bigl(sh \setminus \{S, T\}\bigr) = \rho_{PSD}(sh)$.

It turns out that reduction (i.e., the elimination of redundant sharing groups) and equivalence check fit together very well. Thus we present an algorithm for achieving both at the same time in Figure 7.2.

Require: the sharing sets $sh_{\text{new}} \stackrel{\text{def}}{=} \{S_1, \dots, S_n\}$ and $sh_{\text{old}} \stackrel{\text{def}}{=} \{T_1, \dots, T_m\}$ obtained at the current and previous iteration, respectively. The set sh_{old} is guaranteed not to contain redundant sharing groups. Moreover, $S_1 < \cdots < S_n$ and $T_1 < \cdots < T_m$.
Ensure: on exit the sharing set sh is such that $\rho_{PSD}(sh) = \rho_{PSD}(sh_{\text{new}})$ and is guaranteed not to contain redundant sharing groups. Moreover, the Boolean variable *changed* is set to **true** if and only if $sh \neq sh_{\text{old}}$ and hence $\rho_{PSD}(sh_{\text{new}}) \neq \rho_{PSD}(sh_{\text{old}})$.

```
sh := sh_new
changed := false
i := 1
j := 1
while ¬changed and i ≤ n and j ≤ m do
   if S_i = T_j then
      i := i + 1
      j := j + 1
   else if redundant(S_i, sh) then
      sh := sh \ {S_i}
      i := i + 1
   else
      changed := true
   end if
end while
while i ≤ n do
   if redundant(S_i, sh) then
      sh := sh \ {S_i}
   else
      changed := true
   end if
   i := i + 1
end while
```

Figure 7.2: An optimized algorithm for applying redundancy elimination and checking whether a fixpoint has been reached at the same time.

The function *redundant*, when given a sharing set $sh \in SH$ and a sharing group $S_i \in sh$, returns the Boolean value **true** if and only if S_i is redundant in sh. Its implementation is straightforward and matches the condition given in Definition 4.2. Notice that the implementation of *redundant* benefits from the total ordering used to represent sharing sets as ordered sequences of sharing groups.

If we assume (as we do in our implementation) that the result of the abstract evaluation of each clause is reduced, then the algorithm of Figure 7.2 allows part of the reduction work done at iteration k to be reused at iteration $k+1$, both for simplifying the reduction and for checking equivalence (i.e., the test required to detect whether a local fixpoint has been reached). Here, once again, the total ordering among sharing groups proves very useful. The algorithm proceeds as follows: the sharing groups in the sharing sets obtained at iterations k and $k+1$ (sh_{old} and sh_{new}, respectively) are considered in ascending order (recall that $S_i \subset S_j$ implies $S_i < S_j$ and thus, by contraposition, $S_i \geq S_j$ implies $S_i \not\subset S_j$). Since a sharing group can be "made redundant" only by its proper subsets, and since sh_{old} is reduced, as long as no difference is observed between the sharing groups in sh_{old} and sh_{new}, we know that the sharing groups seen so far are not redundant. Notice that, as a consequence of the analysis process, we have $sh_{\mathrm{old}} \subseteq sh_{\mathrm{new}}$, since sh_{new} is always obtained as $sh_{\mathrm{old}} \cup sh'$ for some $sh' \in \mathit{SH}$ (set theoretic union is the lub on SH). When two different sharing groups are observed there are two possibilities: either the sharing group $S_i \in sh_{\mathrm{new}}$ is redundant, in which case it is eliminated and the algorithm proceeds with the first loop, or S_i is not redundant, in which case we know that the fixpoint has not been reached ($\mathit{changed} := \mathbf{true}$) and the algorithm continues with simple reduction at lines 16–23. The algorithm for reduction only can be obtained by just considering lines 1, 3, 16–18, and 21–23.

7.3 The Results of the Comparison

We describe here the results obtained when comparing the abstract domain *SFL* with respect to its non-redundant version SFL_2. For these tests we have switched off all the other domains currently supported by China. The reader interested in the comparison between plain *SH* and *PSD* is referred to [BHZ97], where all the possible combinations (with freeness only, with linearity only, with both, and with none of them) are taken into account. As noted by several authors, from a practical point of view, sharing analysis without freeness or linearity does not make sense. Both these properties allow, in a significant proportion of cases, costly operations (such as star-union or 2-self-union) to be dispensed with, thereby improving both the precision and the efficiency of the analysis. Of course, in absence of freeness and/or linearity the analysis based on *SH* has to perform more star-unions. Since star-union is much more computationally complex than 2-self-union, the lack of freeness and/or linearity is more penalizing for *SH* than it is for *PSD*.

The comparison was performed on 372 programs for goal independent analysis and 256 programs for goal dependent analysis. This difference in the number of benchmarks considered comes from the fact that many programs either do not have a set of entry goals or use constructs such as `call(G)` where `G` is a term whose principal functor is not known. In these cases, the analyzer recognizes that goal dependent analysis is pointless, since no call-patterns can be excluded. Given the high number of benchmarks, the experimental results can only be summarized here. More information (including a description of the constantly growing benchmark suite and detailed results for each benchmark) can be found at the URI `http://www.cs.unipr.it/China`.

For each benchmark, the *precision* of each analysis is measured by counting the number of independent pairs as well as the numbers of definitely ground, free and linear variables detected by the corresponding abstract domain. As predicted by our theoretical results, the two considered domains achieve the same precision on all of the observed quantities, so that no precision comparison has to be reported.

The *efficiency* of each analysis is evaluated by recording, for each benchmark, the analysis fixpoint computation time: in particular, we do not consider the time needed to read and parse the analysed program and the time to write the final results of the analysis. The CHINA analyzer is endowed with a domain-independent mechanism imposing a time limit on the analysis of a program: for all the experiments presented in this thesis, the limit has been fixed to 1800 seconds. When this limit is reached CHINA forces a "don't know" result for those call-patterns and success-patterns whose analysis is still incomplete. All the experiments have been conducted on a GNU/Linux PC system equipped with an AMD Athlon clocked at 700 MHz and 128 MB of RAM; the timings are in seconds of user time, as provided by the `getrusage` system call.

In the first summary in Table 7.1 we provide the sums, averages, standard deviations and median values computed over the whole benchmark suite, (denoted Sum, Avg, StDev and Median, respectively). The computation of these values is performed twice. In the first half of the summary, we consider all programs (note that each timed-out analysis is given the value of the time-out threshold, i.e. 1800 seconds). In the second half, we only consider those programs whose analysis has terminated on *both* domains: as a matter of fact, all analyses terminating on SFL also terminates on SFL_2.

The values reported are self-explanatory. For instance, the average time for the goal dependent analysis using SFL_2 is less than half of the average time computed when using SFL; moreover, if we disregard the time-outs, this ratio falls further, going below one tenth. Median values register a small, almost negligible increase: this is due to the overhead of the process of removing redundant elements. This tiny efficiency loss, that basically affects the smaller benchmarks only, is more than rewarded by the many consistent time improvements registered on the larger benchmarks.

In the second and third summaries in Table 7.1 we provide more detailed views, where the timings are summarized by partitioning the suite into equivalence classes and reporting the cardinality of each class using percentages. In the second summary, we consider the distribution of the *absolute time differences*, that is we measure the absolute slow-down or speed-up obtained by replacing SFL with SFL_2. Note that the equivalence class called 'same time' actually comprises the benchmarks having a time difference below a given threshold, which is fixed at 0.1 seconds: besides the usual considerations on the potential inaccuracy of small time measures, the ratio behind this choice is that, probably, the user will not notice such a difference. The equivalence class called 'both timed out', as the name suggests, contains those benchmarks whose analysis, for both domains, could not be completed in the fixed time limit. In the third summary, for each analysis we show the distribution of the *total fixpoint computation times*. Here the class 'timed out' provides evidence for the many programs whose analyses timed out when using the SFL domain, while terminating in the given time limit when using SFL_2.

By combining the different views of the same experimental data, more insight can be gained. For instance, in the second summary of Table 7.1, the value 10.2 that can be found at the intersection of the row labeled 'improvement > 1' with the column labeled

Programs	Measure	Goal Independent		Goal Dependent	
		SFL	SFL_2	*SFL*	SFL_2
All Benchs	Sum	72071.40	19107.90	39175.90	19254.70
	Avg	193.74	51.37	153.03	75.21
	StDev	584.59	261.97	508.80	348.12
	Median	0.02	0.09	0.02	0.09
Terminated	Sum	1871.37	234.68	4975.90	407.07
	Avg	5.62	0.70	21.00	1.72
	StDev	48.05	4.76	141.02	9.07
	Median	0.02	0.09	0.02	0.09

Time difference class	% benchmarks	
	Goal Ind.	Goal Dep.
both timed out	1.6	3.5
degradation > 1	1.1	1.2
$0.5 <$ degradation ≤ 1	0.5	0.4
$0.2 <$ degradation ≤ 0.5	1.6	2.3
$0.1 <$ degradation ≤ 0.2	4.3	5.5
same time	73.4	75.8
$0.1 <$ improvement ≤ 0.2	0.3	—
$0.2 <$ improvement ≤ 0.5	1.1	0.8
$0.5 <$ improvement ≤ 1	0.5	0.4
improvement > 1	15.6	10.2

Total time class	Goal Ind.		Goal Dep.	
	SFL	SFL_2	*SFL*	SFL_2
timed out	10.5	1.6	7.4	3.5
$t > 10$	3.8	9.4	5.1	7.4
$5 < t \leq 10$	0.8	1.3	1.2	1.2
$1 < t \leq 5$	4.3	5.6	2.7	3.5
$0.5 < t \leq 1$	1.1	3.5	1.2	2.0
$0.2 < t \leq 0.5$	2.7	5.6	4.3	9.8
$t \leq 0.2$	76.9	72.8	78.1	72.7

Table 7.1: *SFL* vs SFL_2: efficiency comparison.

Goal Independent			Goal Dependent		
Program	*SFL*	SFL_2	Program	*SFL*	SFL_2
chat80	180.89	13.36	LeanTaP	511.10	5.43
diagnoser	14.36	3.46	action	88.99	12.97
linger_old	28.69	9.44	chasen-all	124.32	1.21
mfoil	200.40	2.38	circan	305.50	35.76
pappiall	29.54	2.02	ezan	453.76	24.42
protein	106.78	7.89	horgen	804.64	99.73
semigroup	223.66	81.49	puzzle	21.43	5.55
sprftp	781.36	3.77	reducer	663.19	12.77
synth	34.19	2.33	simple_analyzer	51.68	4.33
trs	140.61	2.96	strips	114.23	11.77
			synth	1738.94	60.21

Table 7.2: *SFL* vs SFL_2: time improvements greater than 10 secs.

'Goal Dep.' is to be read as follows: *"for 10.2% of the benchmarks evaluated in the goal dependent analysis (i.e., for 26 programs), the improvement in the fixpoint computation time when replacing SFL by SFL_2 was greater than 1 second."* Looking now at the third summary, we can see (by summing up percentages) that for the goal dependent analysis using the domain *SFL* only 16.4% of the programs required more than one second to be analysed. Thus, we can conclude that about 2/3 of these programs do benefit of significant efficiency improvements.

Note that our choice of considering absolute time differences is deliberate, since we believe that these provide a better picture than relative time differences. For instance, consider a program whose analysis takes 0.1 seconds and suppose that, by using a different abstract domain, we obtain the same precision results in 0.05 seconds (resp., 0.2 seconds). By using relative time differences, we will conclude that the new domain halves (resp., doubles) the fixpoint computation time, which could be regarded as a big win (resp., loss). However, as already noted above, the real point in such a case is that the user will not notice the difference. In contrast, if the new abstract domain is able to improve the total analysis time, say, from 1 minute to 50 seconds, then the user will be likely to appreciate such a 10 seconds speed-up, even if it is much smaller than the previous one in relative terms. Note however that this reasoning applies *in our context*: in particular, we do not have to generalize our experimental results to the so-called real world programs, since our benchmark suite already includes many of them. Relative time differences can be a better choice when using a benchmark suite made up of small or medium sized programs.

In order to provide some hints on the raw data, we present another couple of tables with detailed timing information. In Table 7.2 we report the set of benchmarks whose analysis terminated for both domains and such that the time improvement obtained was above 10 seconds. Note that these cannot be considered the *best cases*, since we also have as many as 43 benchmarks whose analysis completed on SFL_2 only. Finally, in Table 7.3 we report the *worst cases*, that is the few benchmarks on which, by using the domain

Goal Independent			Goal Dependent		
Program	*SFL*	SFL_2	Program	*SFL*	SFL_2
XRayVer33	0.48	1.31	botworld	0.82	1.86
chartgraph	0.37	1.67	music	38.42	48.08
horgen	0.42	5.29	semigroup	0.97	1.96
map	1.10	3.58	vhdl97_parser	11.25	32.92
music	10.50	12.37			
vhdl97_parser	0.35	0.92			

Table 7.3: *SFL* vs SFL_2: time degradations greater than 0.5 secs.

SFL_2, we obtained a time degradation above 0.5 seconds.

In summary, experimentation shows clearly that the SFL_2 domain is a good idea. By replacing *SFL* with SFL_2 we have a very few (noticeable) performance degradations while obtaining remarkable speed-ups in a significant number of cases, with the average case being definitely in favor of SFL_2. Moreover, the speed-ups occur when they are most needed, that is for the analysis of programs where *SFL* behaves badly. In other words, SFL_2 has a much more *stable* behavior: this is no surprise, since, among other things, we have replaced an operator with exponential complexity (star-union) with a quadratic one (2-self-union). This stability is highly desirable for practical data-flow analyzers. Of course, analyses based on SFL_2 always require less (often much less) memory than those based on *SFL*. The slow-downs happen when reduction is repeatedly attempted on sharing sets that have few or no redundant sharing groups. As pointed out in [BHZ97], the slow-downs can be almost eliminated by not applying reduction after each binary union and 2-self-union. With this choice, SFL_2 would be always more efficient than *SFL* but the maximal speed-ups obtained would not be as high as reported here.

The last lesson to be learned from the above tables is that, even though ρ_{PSD} is an important step toward a practical sharing analysis for logic programs, it is not the last one. As a matter of fact, there still are programs whose analysis requires too much time and/or memory space. Finding a solution to this problem will be the subject of Chapter 9.

Chapter 8

Even More Precision

Besides the integration of set-sharing with freeness and linearity information, a number of other proposals of refined domain combinations for sharing analysis have been circulating for years. One feature that is common to these proposals is that they do not seem to have undergone a thorough experimental evaluation even with respect to the expected precision gains. In this chapter we experimentally evaluate: helping SFL_2 with the definitely ground variables found using Pos, the domain of positive Boolean formulas; the incorporation of explicit structural information; the issue of reordering the bindings in the computation of the abstract unification operator aunify_S; a full implementation of the reduced product of SH and Pos; an original proposal for the addition of a new mode recording the set of variables that are deemed to be ground or free; a refined way of using linearity to improve the analysis; the recovery of hidden information in the combination of set-sharing with freeness information. Finally, we discuss the issue of whether tracking compoundness allows the computation of more sharing information.

> **Note:** this chapter contains an extended and improved version of the results that appeared in [BZH00].

8.1 Looking for Precision Improvements

There have been a number of proposals for more refined combinations which have the potential for improving the precision of the sharing analysis over and above that obtainable using the classical combinations of set-sharing with freeness and linearity. These include the implementation of more powerful abstract semantic operators (since it is well-known that the commonly used ones are sub-optimal) and/or the integration with other domains. Not one of these proposals seem to have undergone a thorough experimental evaluation, even with respect to the expected precision gains. The goal of this chapter is to systematically study these enhancements and provide a uniform theoretical presentation together with an extensive experimental evaluation that will give a strong indication as to their impact on the accuracy of the sharing information. Our investigation is primarily from the point of view of precision. Reasonable efficiency is also clearly of interest but this has to be secondary to the question as to whether precision is significantly improved: only if this is established, should better implementations be researched.

Clearly, the implementation of the various domain combinations was a major part of the work presented in this chapter. However, so as to adapt these assorted proposals into a uniform framework and provide a fair comparison of their results, a large amount of underlying conceptual work was also required. For instance, almost all of the proposed enhancements were designed for systems that perform the occurs-check and some of them were developed for rather different abstract domains: besides changing the representation of the domain elements, such a situation usually requires a reconsideration of the specification of the abstract operators.

Note that, while discussing the different enhancements, we will usually refer to the domain combination SFL, including the plain set-sharing component SH and requiring the evaluation of star-unions in the abstract unification computation. However, this is done for notational convenience only. Unless otherwise stated, the experimental results presented in this chapter have been conducted using the combination SFL_2, where SH and star-unions are replaced by PSD and 2-self-unions, respectively.

We compute the results of 36 different variations of the static analysis, which are then used to perform 32 comparisons. For each benchmark, precision is measured by counting the number of independent pairs (the corresponding columns are labeled 'I' in the tables) as well as the numbers of definitely ground (labeled 'G'), free ('F') and linear ('L') variables detected by each abstract domain. The results obtained for different analyses are compared by computing the *relative precision improvements* (or degradations) on each of these quantities and expressing them using percentages. The benchmark suite is then partitioned into several precision equivalence classes: the cardinalities of these classes are expressed again using percentages. For example, consider the first summary in Table 8.1 on page 208; then, the value 2.7 that can be found, for goal dependent analysis, at the intersection of the row labeled '$0 < p \leq 2$' with the column labeled 'I' is to be read as follows: *"for 2.7 percent of the benchmarks there has been an increase in the number of detected independent pairs of variables which is less than or equal to 2 percent."* The precision class labeled 'unknown' identifies those benchmarks for which a precision comparison was not possible, because one or both of the analyses timed-out. In summary, a precision table gives an approximation of the distribution of the programs in the benchmark suite with respect to the obtained precision gains.

For a rough estimate of the efficiency of the different analyses, for each comparison we also provide two of the views related to the fixpoint computation time that we introduced in Chapter 7. The reader is warned that, in this case, these by no means provide a faithful account of the intrinsic computational cost of the tested domain combinations. The reason is that, for ease of implementation, having targeted at precision we traded efficiency whenever possible. Therefore, these tables provide, so to speak, upper-bounds: refined implementations can be expected to perform at least as well as those reported in the tables. In the summaries reporting the distribution of total analysis time, the columns labeled 'B' and 'E' corresponds to the analyses using the base and the enhanced abstract domain, respectively.

Note that small discrepancies can be observed when comparing the precision results reported here with respect to those in [BZH00]. This has several reasons: first of all, the new amgu_S operator introduced in Chapter 6 is uniformly more precise than the operator used in [BZH00]; second, we now fully exploit the anticipation of the grounding bindings, whereas in the previous *SFL* implementation only the *syntactically* grounding bindings were anticipated: a binding $x \mapsto t$ is syntactically grounding if $\mathrm{vars}(t) = \varnothing$; third, we are now considering more benchmarks.

8.2 A Simple Combination with Pos

In the first enhanced combination, we improve the sharing information of *SFL* by exploiting the groundness information computed by *Pos*, the domain of positive Boolean functions [AMSS98]. As noted in Chapter 5, the set-sharing domain *SH*, and thus also *SFL*, already keeps track of ground dependencies, since it contains *Def*, the domain of definite Boolean functions. However, there are several good reasons to couple *SFL* with *Pos*:

1. *Pos* is strictly more expressive than *Def*, as it can represent (positive) disjunctive groundness dependencies that arise in the analysis of logic programs [AMSS98]. The ability to deal with disjunctive dependencies is also needed for the precise approximation of the primitive constraints of some CLP languages.

2. The increased precision on groundness propagates to the *SFL* component. It can be exploited to remove redundant sharing groups and to identify more linear variables, therefore having a positive impact on the computation of the amgu_S operator of the *SFL* domain. Moreover, the added groundness information allows the detection of more grounding bindings: in the computation of aunify_S, their approximation can be anticipated, yielding better results.

3. Besides being useful for improving precision on other properties, disjunctive dependencies also have a few direct applications, such as occurs-check reduction. As observed in [CKS96], if the groundness formula $x \vee y$ holds, the unification $x = y$ is occurs-check free, even when neither x nor y are definitely linear.

4. Detecting the set of definitely ground variables through *Pos* and exploiting it to simplify the operations on *SFL* can improve the efficiency of the analysis. In particular this is true if the set of ground variables is readily available, as is the case with the GER implementation of *Pos* [BS99].

5. The combination with *Pos* is essential for the application of a powerful widening technique on *SFL* [ZBH99b], which will be the topic of Chapter 9. This is very important, since analysis based on *SFL* without widenings does not scale well.

Prec. class	Goal Independent				Goal Dependent			
	I	G	F	L	I	G	F	L
$5 < p \leq 10$	—	0.3	—	—	—	0.4	—	—
$2 < p \leq 5$	—	0.5	—	0.3	0.4	—	—	—
$0 < p \leq 2$	0.8	0.5	—	0.5	3.1	2.0	—	2.3
same precision	97.6	97.0	98.4	97.6	93.0	94.1	96.5	94.1
unknown	1.6	1.6	1.6	1.6	3.5	3.5	3.5	3.5

Time difference class	% benchmarks	
	Goal Ind.	Goal Dep.
both timed out	1.6	3.5
degradation > 1	2.4	6.6
$0.5 <$ degradation ≤ 1	1.1	—
$0.2 <$ degradation ≤ 0.5	1.1	1.2
$0.1 <$ degradation ≤ 0.2	4.3	2.7
same time	84.7	84.4
$0.1 <$ improvement ≤ 0.2	0.5	0.8
$0.2 <$ improvement ≤ 0.5	0.5	0.4
$0.5 <$ improvement ≤ 1	1.1	—
improvement > 1	2.7	0.4

Total time class	Goal Ind.		Goal Dep.	
	B	E	B	E
timed out	1.6	1.6	3.5	3.5
$t > 10$	9.4	9.7	7.4	7.4
$5 < t \leq 10$	1.3	1.3	1.2	2.0
$1 < t \leq 5$	5.6	6.5	3.5	3.1
$0.5 < t \leq 1$	3.5	3.0	2.0	3.5
$0.2 < t \leq 0.5$	5.6	6.5	9.8	9.8
$t \leq 0.2$	72.8	71.5	72.7	70.7

Table 8.1: SFL_2 vs $(Pos \times SFL_2)$.

Thus, as a first technique to enhance the precision of sharing analysis, we consider the simple propagation of the set of definitely ground variables from the *Pos* component to the *SFL* component (a more precise combination will be considered in Section 8.5). We denote this domain by $(Pos \times SFL)$. As noted above, the GER implementation of [BS99], besides being the fastest implementation of *Pos* known to date, is the natural candidate for this combination, since it provides constant-time access to the set G of the definitely ground variables. In the *SFL* component, the set G is used: (a) to reorder the sequence of bindings in the abstract unification so as to handle the grounding ones first; (b) to eliminate the sharing groups containing at least one ground variable; and (c) to recover from previous linearity losses.

The experimental results for the domain $(Pos \times SFL_2)$ are compared with those obtained for the domain SFL_2 considered in isolation and reported in Table 8.1. It can be observed that a precision improvement is observed in all of the measured quantities but freeness, affecting up to 3.5% of the programs. As for the timings, the figures in the tables confirm that, in a significant number of cases, the costs associated to the inclusion of the *Pos* domain are balanced by the the efficiency improvements described in point 4 above.

Because of the reasons detailed above, we believe *Pos* should be part of the global domain employed by any "production analyzer" for (constraint) logic languages. That is why for the remaining comparisons, unless otherwise stated, this simple combination with the *Pos* domain is always included.

8.3 Tracking Explicit Structural Information

A way of increasing the precision of almost any analysis domain is by enhancing it with explicit structural information, that is by recording in the abstract domain some part of the concrete term structures computed by the program. This technique was proposed by A. Cortesi et al. in [CLV94], where the generic structural domain $\mathtt{Pat}(\Re)$ was introduced. A similar proposal, tailored to sharing analysis, is due to [BCM94a], where *abstract equation systems* are considered.

In our experimental evaluation we adopt the Pattern$(\cdot)$ operator [Bag97a, Bag97b, BHZ00], which is similar to $\mathtt{Pat}(\Re)$ and correctly supports the analysis of languages omitting the occurs-check in the unification procedure as well as those that do not. The construction Pattern$(\cdot)$ upgrades an abstract domain $\mathcal{D}$ (which must support a certain set of basic operations) with structural information. The resulting domain, where structural information is retained to some extent, is usually much more precise than $\mathcal{D}$ alone. There are many occasions where these precision gains give rise to consistent speed-ups. The reason for this is twofold. First, structural information has the potential of pruning some computation paths on the grounds that they cannot be followed by the program being analyzed. Second, maintaining a tuple of terms with many variables, each with its own description, can be cheaper than computing a description for the whole tuple [BHZ00]. Of course, there is also a price to be paid: when using Pattern$(\mathcal{D})$, the elements of $\mathcal{D}$ that are to be manipulated are often bigger (i.e., there are more variables of interest) than those that arise in analyses that are simply based on $\mathcal{D}$.

Of course, structural information is very valuable in itself. When exploited for optimized compilation it allows for enhanced clause indexing and simplified unification. Moreover, several program verification techniques are highly dependent on this kind of information. However, the value of this extra precision can only be measured from the point of view of the target application of the analysis.

When comparing the precision results, the difference in the number of variables tracked by the two analyses poses a non-trivial problem. How can we provide a *fair* measure of the precision gain? There is no easy answer to such a question. The approach chosen is simple though unsatisfactory: at the end of the analysis, first throw away all the structural information computed and then compute the cardinality of the usual sets. In other words, we only measure how the explicit structural information in Pattern($\mathcal{D}$) improves the precision on $\mathcal{D}$ itself, which is only a tiny part of the real gain in accuracy. As shown by the following example, this solution greatly underestimates the precision improvement coming from the integration of structural information.

Consider a simple but not trivial Prolog program: `mastermind`.[1] Consider also the only direct query for which it has been written, '`?- play.`', and focus the attention on the procedure `extend_code/1`. A standard goal dependent analysis of the program with the ($Pos \times SFL$) domain cannot say anything on the successes of `extend_code/1`. If the analysis is performed with Pattern($Pos \times SFL$) the situation changes radically. Here is what such a domain allows CHINA to derive:[2]

```
extend_code([([A|B],C,D)|E]) :-
  list(B), list(E),
  (functor(C,_,1);integer(C)),
  (functor(D,_,1);integer(D)),
  ground([C,D]), may_share([[A,B,E]]).
```

This has the following meaning: "during any execution of the program, whenever a call to the predicate `extend_code/1` succeeds, it will have its argument bound to a term of the form `[([A|B],C,D)|E]`, where `B` and `E` are bound to list cells (i.e., to terms whose principal functor is either `'.'/2` or `[]/0`); `C` and `D` are ground and bound to a functor of arity 1 or to an integer; and pair-sharing may only occur among `A`, `B`, and `E`". Once structural information has been discarded, we can only conclude that `extend_code/1` may succeed. Thus, according to our approach to the precision comparison, explicit structural information gives no improvements in the analysis of `extend_code/1` (which is far from being a fair conclusion).

Table 8.2 reports the results obtained by comparing the domain ($Pos \times SFL_2$) with the domain Pattern($Pos \times SFL_2$). When performing a goal independent analysis we obtain

[1] A program implementing the game "Mastermind", rewritten by H. Koenig and T. Hoppe after code by M. H. van Emden. Available at `http://www.cs.unipr.it/China/Benchmarks/Prolog/mastermind.pl`.

[2] Some extra groundness information obtained by the analysis has been omitted for simplicity: this says that, if `A` and `B` turn out to be ground, then `E` will also be ground.

Prec. class	Goal Independent				Goal Dependent			
	I	G	F	L	I	G	F	L
$p > 20$	3.0	4.3	1.9	3.0	1.2	3.5	1.6	3.1
$10 < p \leq 20$	1.9	2.2	—	2.4	2.0	1.6	—	2.7
$5 < p \leq 10$	1.6	3.2	2.2	2.4	0.8	2.0	0.8	1.6
$2 < p \leq 5$	5.9	3.5	3.0	5.4	2.7	1.2	1.6	2.0
$0 < p \leq 2$	8.9	6.7	6.7	12.4	4.3	1.6	2.0	4.3
same precision	72.8	74.2	80.4	68.5	79.3	80.5	84.4	76.6
unknown	5.9	5.9	5.9	5.9	9.8	9.8	9.8	9.8

Time diff. class	% benchmarks	
	Goal Ind.	Goal Dep.
both timed out	1.6	3.5
degradation > 1	12.1	16.4
$0.5 <$ degradation ≤ 1	2.2	1.2
$0.2 <$ degradation ≤ 0.5	1.9	1.6
$0.1 <$ degradation ≤ 0.2	2.7	3.9
same time	71.5	69.9
$0.1 <$ improvement ≤ 0.2	0.8	—
$0.2 <$ improvement ≤ 0.5	1.9	0.4
$0.5 <$ improvement ≤ 1	1.1	0.8
improvement > 1	4.3	2.3

Total time class	Goal Ind.		Goal Dep.	
	B	E	B	E
timed out	1.6	5.9	3.5	9.8
$t > 10$	9.7	8.6	7.4	7.4
$5 < t \leq 10$	1.3	1.9	2.0	2.0
$1 < t \leq 5$	6.5	4.0	3.1	4.7
$0.5 < t \leq 1$	3.0	4.6	3.5	2.7
$0.2 < t \leq 0.5$	6.5	8.9	9.8	9.8
$t \leq 0.2$	71.5	66.1	70.7	63.7

Table 8.2: $(Pos \times SFL_2)$ vs Pattern$(Pos \times SFL_2)$.

a precision improvement *in at least one of the measured quantities* for about one third of the benchmarks; the same happens for one sixth of the benchmarks in the case of a goal dependent analysis. Moreover, the increases in precision can be considerable, as witnessed by the percentages of benchmarks falling in the higher precision classes.

One issue that should be resolved is whether the improvements provided by explicit structural information subsume those previously obtained for the simple combination with *Pos*. Intuitively, it would seem that this cannot happen, since these two enhancements are based on different kinds of information: while the Pattern($\cdot$) construction encodes some *definite* structural information, the precision gain due to using *Pos* rather than just *Def* only stems from *disjunctive* groundness dependencies. However, the impact of these techniques on the overall analysis is really intricate and some overlapping cannot be excluded *a priori*: for instance, both techniques affect the ordering of bindings in the computation of abstract unification on *SFL*. In order to provide some experimental evidence for this qualitative reasoning, we also computed a comparison between the simpler domain Pattern(SFL_2) and the domain Pattern($\mathit{Pos} \times \mathit{SFL}_2$). Since the few differences between Tables 8.1 and 8.3 can be explained by discrepancies in the numbers of programs that timed-out, these results confirm our expectations that these two enhancements are effectively orthogonal.

Similar experimental evaluations, based on the abstract equation systems of [BCM94a], were reported by Mulkers et al. in [MSJB94, MSJB95]. Here a depth-k abstraction (replacing all subterms occurring at a depth greater than or equal to k with fresh abstract variables) is conducted on a small benchmark suite (19 programs) for values of k between 0 and 3. The domain they employed was not suitable for the analysis of real programs and, in fact, even the analysis of a modest-sized program like `ann` could only be carried out with depth-0 abstraction (i.e., without any structural information). Such a problem in finding practical analyzers that incorporated structural information with sharing analysis was not unique to this work: there was at least one other previous attempt to evaluate the impact of structural information on sharing analysis which failed because of combinatorial explosion [A. Cortesi, personal communication, 1996]. What makes the more realistic experimentation now possible is the adoption of the non-redundant domain SFL_2, where the exponential star-union operation is replaced by the quadratic 2-self-union. Note that, even if biased by the absence of widenings, the timings reported in Table 8.2 show that the Pattern($\cdot$) construction is computationally feasible. Indeed, as demonstrated by the results reported in [BHZ00], an analyzer that incorporates a carefully designed structural information component, besides being more precise, can also be very efficient.

The results obtained in this section demonstrate that there is a relevant amount of sharing information that is not detected when using the classical set-sharing domains. Therefore, in order to provide an experimental evaluation that is as systematic as possible, in all of the remaining experiments the comparison is performed both with and without explicit structural information.

Prec. class	Goal Independent				Goal Dependent			
	I	G	F	L	I	G	F	L
$5 < p \leq 10$	—	0.3	—	—	—	0.4	—	—
$2 < p \leq 5$	—	0.5	—	0.3	0.4	—	—	—
$0 < p \leq 2$	0.3	—	—	—	2.7	2.3	—	2.3
same precision	93.8	93.3	94.1	93.8	87.1	87.5	90.2	87.9
unknown	5.9	5.9	5.9	5.9	9.8	9.8	9.8	9.8

Time diff. class	% benchmarks	
	Goal Ind.	Goal Dep.
both timed out	5.9	9.8
degradation > 1	5.1	7.4
$0.5 <$ degradation ≤ 1	0.5	0.4
$0.2 <$ degradation ≤ 0.5	2.7	0.8
$0.1 <$ degradation ≤ 0.2	4.8	4.7
same time	79.8	76.6
$0.1 <$ improvement ≤ 0.2	0.3	—
$0.2 <$ improvement ≤ 0.5	—	—
$0.5 <$ improvement ≤ 1	0.3	—
improvement > 1	0.5	0.4

Total time class	Goal Ind.		Goal Dep.	
	B	E	B	E
timed out	5.9	5.9	9.8	9.8
$t > 10$	8.1	8.6	7.0	7.4
$5 < t \leq 10$	1.6	1.9	2.0	2.0
$1 < t \leq 5$	4.0	4.0	4.7	4.7
$0.5 < t \leq 1$	4.3	4.6	2.0	2.7
$0.2 < t \leq 0.5$	6.7	8.9	8.6	9.8
$t \leq 0.2$	69.4	66.1	66.0	63.7

Table 8.3: Pattern(SFL_2) vs Pattern($Pos \times SFL_2$).

8.4 Reordering the Non-Grounding Bindings

As already explained in Section 8.2, the result of abstract unification on *SFL* may depend on the order in which the bindings are considered and will be improved if the grounding bindings are considered first. This heuristic, which has been used for all the experiments in this thesis, is well-known [Lan90].

In the literature, the examples that illustrate the non-commutativity of the abstract mgu on domain combinations such as *SFL* all use a grounding binding. However, the next example shows that the problem is more general than that.

Example 8.1 *Let* $VI = \{u, v, w, x, y, z\}$ *be the set of relevant variables, and consider the SFL element*

$$d \stackrel{\mathrm{def}}{=} \bigl\langle \{vx, vz, w, x, y\}, \{v\}, \{v\} \bigr\rangle,$$

with the bindings $v \mapsto w$ *and* $x \mapsto y$*. Then, applying* aunify_S *to these bindings in the given ordering, we have:*

$$\begin{aligned} d_1 &\stackrel{\mathrm{def}}{=} \mathrm{aunify}_S\bigl(d, \langle x \mapsto y, v \mapsto w\rangle\bigr) \\ &= \mathrm{amgu}_S\bigl(\mathrm{amgu}_S(d, x \mapsto y), v \mapsto w\bigr) \\ &= \mathrm{amgu}_S\Bigl(\bigl\langle \{vxy, vz, w, xy\}, \varnothing, \varnothing \bigr\rangle, v \mapsto w\Bigr) \\ &= \bigl\langle \{vwxy, vwxyz, vwz, xy\}, \varnothing, \varnothing \bigr\rangle. \end{aligned}$$

Using the reverse ordering, we have:

$$\begin{aligned} d_2 &\stackrel{\mathrm{def}}{=} \mathrm{aunify}_S\bigl(d, \langle v \mapsto w, x \mapsto y\rangle\bigr) \\ &= \mathrm{amgu}_S\bigl(\mathrm{amgu}_S(d, v \mapsto w), x \mapsto y\bigr) \\ &= \mathrm{amgu}_S\Bigl(\bigl\langle \{vwx, vwz, x, y\}, \varnothing, \varnothing \bigr\rangle, x \mapsto y\Bigr) \\ &= \bigl\langle \{vwxy, vwz, xy\}, \varnothing, \varnothing \bigr\rangle. \end{aligned}$$

Variables y *and* z *possibly share in* d_1*, while being detected as definitely independent in* d_2*.*

In principle, optimality can be obtained by adopting the *brute-force* approach: trying all the possible orderings of the non-grounding bindings. However, this is clearly not feasible. While lacking a better alternative, it is reasonable to look for heuristics that can be applied in the context of a *local search* paradigm: at each step, the next binding for the amgu_S procedure is chosen by evaluating the effect of its abstract execution, considered in isolation, on the precision of the analysis.

Suppose the number of independent pairs is taken as a measure of precision. Then, at each step, for each of the bindings under consideration, the new component sh', as given by Definition 6.33, must be computed. However, because the computation of sh' is the most costly operation to be performed in the computation of the amgu_S operator, a direct application of this heuristic does not appear to be feasible. As an alternative, consider a heuristic based on the number of star-unions that have to be computed. Star-unions are likely to cause large losses in the number of independent pairs that are found. As only non-grounding bindings are considered, any binding requiring the computation of a star-union will need the star-union even if it is delayed, although a binding that does not require the star-union may require it if its computation is postponed: its variables may lose their freeness, linearity or independence as a result of evaluating the other bindings. It follows that one potential heuristic is:

H1: *"select first the binding requiring less star-unions"*.

Note that, by applying this heuristic to Example 8.1, we would have chosen the better ordering since the second binding does not require star-unions, while the first one requires both of them. However, if considered in isolation, this heuristic is not a general solution and can actually lead to precision losses. The problem is that, if a binding that needs a star-union is delayed, then, when the star-union is computed, it may be done on a larger sharing set, forcing more (independent) pairs of variables into the same sharing group.

Example 8.2 *Let VI be as in the previous example and consider the application of the bindings $u \mapsto x$ and $v \mapsto w$ to the abstract description*

$$d \stackrel{\text{def}}{=} \bigl\langle \{u, uw, v, w, xy, xz\}, \{u, x\}, \{u, x\} \bigr\rangle.$$

Since x and u are both free variables, no star-union is needed in the computation of $\mathrm{amgu}_S(d, u \mapsto x)$*, while two star-unions are needed when computing* $\mathrm{amgu}_S(d, v \mapsto w)$*. Thus, according to the heuristic* H1*, we have:*

$$\begin{aligned} d_1 &\stackrel{\text{def}}{=} \mathrm{aunify}_S\bigl(d, \langle u \mapsto x, v \mapsto w\rangle\bigr) \\ &= \mathrm{amgu}_S\bigl(\mathrm{amgu}_S(d, u \mapsto x), v \mapsto w\bigr) \\ &= \mathrm{amgu}_S\Bigl(\bigl\langle \{uwxy, uwxz, uxy, uxz, v, w\}, \{u, x\}, \{u, x\} \bigr\rangle, v \mapsto w\Bigr) \\ &= \bigl\langle \{uvwxy, uvwxyz, uvwxz, uxy, uxz, vw\}, \varnothing, \varnothing \bigr\rangle. \end{aligned}$$

Using the other ordering, we have:

$$\begin{aligned} d_2 &\stackrel{\text{def}}{=} \mathrm{aunify}_S\bigl(d, \langle v \mapsto w, u \mapsto x\rangle\bigr) \\ &= \mathrm{amgu}_S\bigl(\mathrm{amgu}_S(d, v \mapsto w), u \mapsto x\bigr) \\ &= \mathrm{amgu}_S\Bigl(\bigl\langle \{u, uvw, vw, xy, xz\}, \{x\}, \{x\} \bigr\rangle, u \mapsto x\Bigr) \\ &= \bigl\langle \{uvwxy, uvwxz, uxy, uxz, vw\}, \varnothing, \varnothing \bigr\rangle. \end{aligned}$$

Variables y and z possibly share in d_1, while being detected as definitely independent in d_2. Thus, in this case, the adoption of the heuristic H1 *decreases the number of known independent pairs.*

Another possibility is to consider a heuristic that uses the numbers of free and linear variables as a measure of precision for local optimization. That is, we could adopt the following rule:

H2: *"select first the binding maximizing freeness and linearity"*.

However, Example 8.2 is evidence that even such a proposal may cause a precision loss: the binding $u \mapsto x$ would be chosen first as it preserves the freeness of variable u.

In order to evaluate the effects of these two heuristics on real programs, we implemented them on top of the domain combination $(Pos \times SFL_2)$ and we compared the corresponding precision results with respect to those obtained when using the "straight" abstract computation, which considers the non-grounding bindings in the textual order, from left to right. The results reported in Tables 8.4 and 8.5 can be summarized as follows:

1. the precision on the groundness and freeness components is never affected;

2. the precision on the independence and linearity components is rarely affected, in particular when considering goal dependent analyses;

3. even for real programs, as was the case for the artificial examples given above, the precision can be increased as well as decreased.

Tables 8.4 and 8.5 show that the heuristic H2, based on freeness and linearity information, is slightly better than the use of the straight order, which, in its turn, is slightly better than the heuristic H1, based on the number of star-unions; for the latter, no precision improvement is observed. As these results cannot be generalized to the other orderings, our investigation is not conclusive. Besides designing "smarter" heuristics, it is interesting to provide a kind of *responsiveness test* for the underlying domain with respect to the order of approximation of the non-grounding bindings: a simple test consists in measuring how much the precision can be affected, in either way, by the application of an almost arbitrary order. This is the motivation for the comparison reported in Table 8.6, where we applied the rule:

H3: *"select first the right-most binding"*.

That is, we blindly reverse the usual order of the non-grounding bindings. As for the results given in Tables 8.4 and 8.5, the number of changes to the precision observed in Table 8.6 is small and all the observations made above still hold. Surprisingly, this blind heuristic provides marginally better precision results than those obtained using the others.

In contrast, the different orderings of the non-grounding bindings have a big, somehow unpredictable impact on efficiency. A few analyses that did not complete on the base domain now terminate in the given time limit, and the vice versa also holds. Even in this case, the best results are obtained with heuristic H3.

Goal Independent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
same precision	97.6	98.4	98.4	98.4	92.2	93.8	93.8	93.8
unknown	1.6	1.6	1.6	1.6	6.2	6.2	6.2	6.2
$-2 \leq p < 0$	0.8	—	—	—	1.6	—	—	—

Goal Dependent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
same precision	96.5	96.5	96.5	96.5	90.2	90.2	90.2	90.2
unknown	3.5	3.5	3.5	3.5	9.8	9.8	9.8	9.8

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
both timed out	1.6	5.9	3.5	9.4
degradation > 1	5.4	2.2	7.4	6.3
$0.5 <$ degradation ≤ 1	1.3	0.3	—	0.8
$0.2 <$ degradation ≤ 0.5	2.2	1.3	0.8	1.6
$0.1 <$ degradation ≤ 0.2	1.1	0.5	1.2	2.0
same time	82.5	80.9	84.8	76.6
$0.1 <$ improvement ≤ 0.2	0.8	1.9	0.8	0.4
$0.2 <$ improvement ≤ 0.5	0.8	0.8	0.4	—
$0.5 <$ improvement ≤ 1	0.3	1.1	—	—
improvement > 1	4.0	5.1	1.2	3.1

Total time class	Goal Independent				Goal Dependent			
	without SI		with SI		without SI		with SI	
	B	E	B	E	B	E	B	E
timed out	1.6	1.6	5.9	6.2	3.5	3.5	9.8	9.4
$t > 10$	9.7	9.7	8.6	8.1	7.4	7.4	7.4	7.8
$5 < t \leq 10$	1.3	0.8	1.9	1.9	2.0	2.0	2.0	2.3
$1 < t \leq 5$	6.5	7.3	4.0	4.3	3.1	2.7	4.7	3.9
$0.5 < t \leq 1$	3.0	2.2	4.6	4.3	3.5	3.9	2.7	3.5
$0.2 < t \leq 0.5$	6.5	7.0	8.9	8.9	9.8	10.9	9.8	10.2
$t \leq 0.2$	71.5	71.5	66.1	66.4	70.7	69.5	63.7	62.9

Table 8.4: The heuristic H1, based on the number of star-unions.

Goal Independent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
$0 < p \leq 2$	—	—	—	—	2.4	—	—	0.3
same precision	97.3	97.8	97.8	97.8	90.9	94.1	94.1	93.8
unknown	2.2	2.2	2.2	2.2	5.9	5.9	5.9	5.9
$-2 \leq p < 0$	0.5	—	—	—	0.8	—	—	—

Goal Dependent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
$0 < p \leq 2$	—	—	—	0.4	—	—	—	—
same precision	95.3	95.3	95.3	94.9	89.5	89.5	89.5	89.5
unknown	4.7	4.7	4.7	4.7	10.5	10.5	10.5	10.5

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
both timed out	1.6	5.9	3.5	9.8
degradation > 1	7.8	3.8	7.8	6.6
$0.5 <$ degradation ≤ 1	1.9	0.5	1.6	—
$0.2 <$ degradation ≤ 0.5	2.2	3.0	1.2	2.7
$0.1 <$ degradation ≤ 0.2	1.9	2.7	3.1	2.3
same time	78.5	74.7	79.3	74.2
$0.1 <$ improvement ≤ 0.2	0.5	0.5	—	—
$0.2 <$ improvement ≤ 0.5	0.3	1.3	0.4	0.4
$0.5 <$ improvement ≤ 1	—	1.1	0.8	—
improvement > 1	5.4	6.5	2.3	3.9

Total time class	Goal Independent				Goal Dependent			
	without SI		with SI		without SI		with SI	
	B	E	B	E	B	E	B	E
timed out	1.6	2.2	5.9	5.9	3.5	4.7	9.8	10.5
$t > 10$	9.7	9.9	8.6	7.3	7.4	6.6	7.4	6.3
$5 < t \leq 10$	1.3	1.1	1.9	2.4	2.0	1.6	2.0	2.3
$1 < t \leq 5$	6.5	6.7	4.0	4.8	3.1	3.9	4.7	5.1
$0.5 < t \leq 1$	3.0	2.2	4.6	4.8	3.5	3.9	2.7	3.5
$0.2 < t \leq 0.5$	6.5	8.3	8.9	10.8	9.8	10.5	9.8	9.0
$t \leq 0.2$	71.5	69.6	66.1	64.0	70.7	68.8	63.7	63.3

Table 8.5: The heuristic H2, based on freeness and linearity.

Goal Independent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
$0 < p \leq 2$	0.3	—	—	—	2.7	—	—	—
same precision	97.3	98.1	98.1	98.1	90.3	94.1	94.1	94.1
unknown	1.9	1.9	1.9	1.9	5.9	5.9	5.9	5.9
$-2 \leq p < 0$	0.5	—	—	—	1.1	—	—	—

Goal Dependent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
$0 < p \leq 2$	—	—	—	0.4	—	—	—	—
same precision	96.1	96.1	96.1	95.7	89.8	89.8	89.8	89.8
unknown	3.9	3.9	3.9	3.9	10.2	10.2	10.2	10.2

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
both timed out	1.3	5.6	3.5	9.4
degradation > 1	4.6	4.8	5.5	6.3
0.5 < degradation ≤ 1	0.8	0.5	—	—
0.2 < degradation ≤ 0.5	0.8	1.1	0.8	0.8
0.1 < degradation ≤ 0.2	2.2	2.2	1.2	0.4
same time	79.0	77.2	82.8	75.8
0.1 < improvement ≤ 0.2	1.6	0.5	2.0	0.8
0.2 < improvement ≤ 0.5	2.2	1.3	0.8	1.2
0.5 < improvement ≤ 1	0.5	0.3	0.4	—
improvement > 1	7.0	6.5	3.1	5.5

Total time class	Goal Independent				Goal Dependent			
	without SI		with SI		without SI		with SI	
	B	E	B	E	B	E	B	E
timed out	1.6	1.6	5.9	5.6	3.5	3.9	9.8	9.8
$t > 10$	9.7	9.1	8.6	8.3	7.4	7.0	7.4	6.6
$5 < t \leq 10$	1.3	1.9	1.9	1.1	2.0	1.6	2.0	2.0
$1 < t \leq 5$	6.5	6.2	4.0	5.4	3.1	2.7	4.7	5.9
$0.5 < t \leq 1$	3.0	2.7	4.6	3.8	3.5	3.5	2.7	2.0
$0.2 < t \leq 0.5$	6.5	7.8	8.9	9.4	9.8	10.9	9.8	9.4
$t \leq 0.2$	71.5	70.7	66.1	66.4	70.7	70.3	63.7	64.5

Table 8.6: The heuristic H3, reversing the ordering.

8.5 Pos and Sharing: the Reduced Product

The overlap between the information provided by *Pos* and the information provided by *SH* mentioned in Section 8.2 means that the Cartesian product $Pos \times SH$ contains redundancy, that is, there is more than one element that can characterize the same set of concrete computational states. In [BZH00], two techniques that allowed the removal of *some* of these redundancies were experimentally evaluated. In contrast, we now consider the full integration of these two domains.

The reduced product between *Pos* and *SH*, here denoted by $Pos \sqcap SH$, has been elegantly characterized in [CSS99], where set-sharing *à la* Jacobs and Langen is expressed in terms of elements of the *Pos* domain itself. The isomorphism maps each set-sharing element $sh \in SH$ into the Boolean formula $\phi \in Pos$ such that

$$[\phi]_{VI} = \{\, VI \setminus S \mid S \in sh \,\} \cup \{\, VI \}.$$

The sharing information encoded by an element $(\phi_g, \phi_{sh}) \in Pos \times Pos$ can be improved by replacing the second component (that is, the Boolean formula describing set-sharing information) with the conjunction $\phi_g \wedge \phi_{sh}$. The reader is referred to [CSS99] for a complete account of this composition and a justification of its correctness.

This specification of the reduced product can be reformulated, using the standard set-sharing representation for the second component, to define a reduction procedure $\mathrm{reduce}_{Pos} \colon SH \times Pos \rightarrow SH$ such that, for all $sh \in SH$, $\phi_g \in Pos$,

$$\mathrm{reduce}_{Pos}(sh, \phi_g) = \bigl\{\, S \in sh \bigm| (VI \setminus S) \in [\phi_g]_{VI} \,\bigr\}.$$

When using the domain *PSD* in place of *SH*, the 'reduce_{Pos}' operator specified above can interact in subtle ways with an implementation removing the ρ_{PSD}-redundant sharing groups from the elements of *PSD*. The following is an example where such an interaction provides results that are not correct.

Example 8.3 *Let* $VI = \{x, y, z\}$ *and* $sh = \{xy, xz, yz, xyz\} \in PSD$ *be the current set-sharing description. Suppose that the implementation internally represents sh by using the* ρ_{PSD}*-reduced element* $sh_{\mathrm{red}} = \{xy, xz, yz\}$*, so that* $sh = \rho_{PSD}(sh_{\mathrm{red}})$*. Suppose also that the groundness description computed on the domain Pos is* $\phi_g = (x \leftrightarrow y \leftrightarrow z)$*. Note that we have* $[\phi_g]_{VI} = \bigl\{\varnothing, \{x, y, z\}\bigr\}$*. Then we have*

$$sh' = \mathrm{reduce}_{Pos}(sh, \phi_g) = \{xyz\};$$
$$sh'_{\mathrm{red}} = \mathrm{reduce}_{Pos}(sh_{\mathrm{red}}, \phi_g) = \varnothing.$$

The two Pos-reduced elements sh' *and* sh'_{red} *are not equivalent, even modulo* ρ_{PSD}*.*

Note that the above example does not mean that the reduced product $Pos \sqcap PSD$ yields results that are not correct; neither does it mean that it is less precise than $Pos \sqcap SH$ for the computation of the observables. More simply, the optimizations used in our

Goal Independent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
$5 < p \leq 10$	—	—	—	—	0.3	—	—	—
$2 < p \leq 5$	0.3	—	—	—	—	—	—	—
$0 < p \leq 2$	2.7	—	—	0.5	3.5	—	—	0.5
same precision	86.8	89.8	89.8	89.2	80.6	84.4	84.4	83.9
unknown	10.2	10.2	10.2	10.2	15.6	15.6	15.6	15.6

Goal Dependent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
$p > 20$	0.4	—	—	—	—	—	—	—
$10 < p \leq 20$	—	—	—	—	0.4	—	—	—
$5 < p \leq 10$	—	—	—	—	0.8	—	—	—
$0 < p \leq 2$	3.1	—	—	—	2.7	—	—	—
same precision	89.1	92.6	92.6	92.6	79.3	83.2	83.2	83.2
unknown	7.4	7.4	7.4	7.4	16.8	16.8	16.8	16.8

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
both timed out	1.6	5.9	3.5	9.8
degradation > 1	15.9	18.5	10.2	15.2
$0.5 <$ degradation ≤ 1	0.3	0.3	0.8	0.8
$0.2 <$ degradation ≤ 0.5	0.8	1.9	0.8	0.8
$0.1 <$ degradation ≤ 0.2	—	0.5	—	1.6
same time	76.3	72.6	75.8	71.1
$0.1 <$ improvement ≤ 0.2	1.9	0.3	6.6	0.4
$0.2 <$ improvement ≤ 0.5	1.9	—	0.8	—
$0.5 <$ improvement ≤ 1	0.5	—	0.8	—
improvement > 1	0.8	—	0.8	0.4

Total time class	Goal Independent				Goal Dependent			
	without SI		with SI		without SI		with SI	
	B	E	B	E	B	E	B	E
timed out	1.6	10.2	5.9	15.6	3.5	7.4	9.8	16.8
$t > 10$	9.7	4.3	8.6	5.9	7.4	5.1	7.4	6.6
$5 < t \leq 10$	1.3	1.3	1.9	0.8	2.0	1.2	2.0	—
$1 < t \leq 5$	6.5	3.2	4.0	3.5	3.1	3.5	4.7	2.7
$0.5 < t \leq 1$	3.0	2.7	4.6	2.4	3.5	1.2	2.7	2.7
$0.2 < t \leq 0.5$	6.5	3.8	8.9	4.0	9.8	5.9	9.8	6.3
$t \leq 0.2$	71.5	74.5	66.1	67.7	70.7	75.8	63.7	64.8

Table 8.7: $(Pos \times SFL_2)$ vs $(Pos \otimes SFL)$.

current implementation of *PSD* are not compatible with the above reduction process. A correct, but very inefficient, implementation could be obtained by computing the closure $\rho_{PSD}(sh)$ before each application of the 'reduce$_{Pos}$' operator. As we are mainly interested in precision, we simply by-pass this problem by performing our comparisons using the plain set-sharing domain SH, with no ρ_{PSD}-redundancy elimination at all.

We thus implemented the above enhancement, denoted by $Pos \otimes SFL$. From a formal point of view, this is *not* the reduced product between *Pos* and *SFL*. While there is a complete reduction between *Pos* and *SH*, the same does not necessarily hold for the combination with freeness and linearity information. The results for the precision comparison, reported in Table 8.7, show that the combination $(Pos \otimes SFL)$ can provide precision improvements, in particular to the number of independent pairs. However, the efficiency results are biased by the lack of ρ_{PSD}-redundancy elimination, so that a significant number of analyses did not terminate in the given time limit.

8.6 Ground-or-free Variables

Most of the ideas investigated in the present work are based on earlier work by other authors. In this section, we describe one originally proposed in [BZH00]. Consider the analysis of the binding $x \mapsto t$ and suppose that, on a subset of all the computation paths, this binding is evaluated with x ground while, on the remaining computation paths, the binding is evaluated with x free. When using the usual combination $(Pos \times SFL)$, before the evaluation of the binding, x will be detected as definitely linear. This information is valuable: the relevant component for t does not have to be star-closed. However, the information that is lost, that is, x being either ground or free, is equally valuable, since this would allow the avoidance of *all* star-unions, even when x and t may share. This loss has the disadvantages that CPU time is wasted by performing unnecessary but costly operations and that the precision is potentially degraded. It is therefore useful to enrich our sharing analysis domain by adding 'ground-or-free' information.

We extend the domain *SFL* with the component $GF \stackrel{\mathrm{def}}{=} \wp(VI)$, consisting of the set of variables that are known to be either ground or free. As for freeness and linearity, the approximation ordering on GF is given by reverse subset inclusion. When computing the abstract mgu on the new domain

$$SGFL \stackrel{\mathrm{def}}{=} SH \times F \times GF \times L,$$

the property of being ground-or-free is used and propagated in almost the same way as freeness information.

Definition 8.4 **(amgu_{GF}.)** *Let $d = \langle sh, f, gf, l \rangle \in SGFL$. We define the predicate $\mathrm{gfree}_d \colon HTerms \to Bool$ such that, for each term $t \in HTerms$, where $\mathrm{vars}(t) \subseteq VI$,*

$$\mathrm{gfree}_d(t) \stackrel{\mathrm{def}}{=} \Bigl(\mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr) = \varnothing\Bigr) \vee \Bigl(\exists x \in VI \mathrel{.} x = t \wedge x \in gf\Bigr).$$

Consider the abstract operations over SFL given in Definitions 6.31 and 6.33 (as well as the notation introduced there). Then, the operator $\mathrm{amgu}_{GF} \colon SGFL \times Bind \to SGFL$ *is defined as*

$$\mathrm{amgu}_{GF}\bigl(d, x \mapsto t\bigr) \stackrel{\mathrm{def}}{=} \langle sh', f', gf', l' \rangle,$$

where sh' *is as specified in Definition 6.33, but replacing all the tests on predicate* free_d *by tests on predicate* gfree_d*;* f' *and* l'' *are as specified in Definition 6.33; finally, we have*

$$gf' = \bigl(VI \setminus \mathrm{vars}(sh')\bigr) \cup gf'';$$

$$gf'' = \begin{cases} gf, & \text{if } \mathrm{gfree}_d(x) \wedge \mathrm{gfree}_d(t); \\ gf \setminus S_x, & \text{if } \mathrm{gfree}_d(x); \\ gf \setminus S_t, & \text{if } \mathrm{gfree}_d(t); \\ gf \setminus (S_x \cup S_t), & \text{otherwise;} \end{cases}$$

$$l' = gf' \cup l''.$$

The computation of the set gf'' is very similar to the computation of the set f' as given in Definition 6.33. The new ground-or-free component gf' is obtained by adding to gf'' the set of all the ground variables: in other words, if a variable "loses freeness" then it also loses its ground-or-free status, unless it is known to be definitely ground.

To summarize, the incorporation of the set of ground-or-free variables is cheap, both in terms of computational complexity and in terms of code to be written. Thus we implemented the domain $(Pos \times SGFL_2)$ and we compared it with respect to $(Pos \times SFL_2)$: as usual, $SGFL_2$ denotes the domain $SGFL$ after having replaced SH by PSD.

In the first two summaries of Table 8.8, the new columns labeled 'GF' report the precision improvements measured on the ground-or-free property itself.[3] Disregarding the many improvements in these columns, few changes can be observed, and almost all of these concern just the linearity information.[4] As far as the timings are concerned, even though efficiency improvements are rare, raw data are better than it appears by looking at the summaries in Table 8.8: the overall slow-down, computed considering all benchmarks and all four variations of the analysis, is less than 53 seconds of CPU time.

The results show that tracking ground-or-free variables, while being potentially useful for improving the precision of a sharing analysis, rarely reaches such a goal. In contrast, the precision gains on the ground-or-free property itself are remarkable, affecting from 36% to 73% of the programs in the benchmark suite. It is possible to foresee several *direct* applications for this information that, together with the just mentioned negligible computational cost, fully justify the inclusion of this enhancement in a static analyzer. In particular, there are at least two ways in which a knowledge of ground-or-free variables

[3] For this comparison, in the analysis using $(Pos \times SFL_2)$, the number of ground-or-free variables is computed by summing the number of ground variables with the number of free variables.

[4] In fact the sole improvement to the number of independent pairs is due to a synthetic benchmark, named `gof`, that was explicitly written to show that variable independence can be affected.

Goal Independent	without Struct Info					with Struct Info				
Prec. class	I	G	F	GF	L	I	G	F	GF	L
$p > 20$	0.3	—	—	53.8	—	0.3	—	—	48.7	—
$10 < p \leq 20$	—	—	—	12.1	—	—	—	—	15.1	—
$5 < p \leq 10$	—	—	—	5.1	—	—	—	—	7.5	—
$2 < p \leq 5$	—	—	—	2.2	—	—	—	—	1.9	—
$0 < p \leq 2$	—	—	—	0.3	1.9	—	—	—	0.5	0.5
same precision	98.1	98.4	98.4	25.0	96.5	93.8	94.1	94.1	20.4	93.5
unknown	1.6	1.6	1.6	1.6	1.6	5.9	5.9	5.9	5.9	5.9

Goal Dependent	without Struct Info					with Struct Info				
Prec. class	I	G	F	GF	L	I	G	F	GF	L
$p > 20$	—	—	—	5.9	—	—	—	—	3.5	—
$10 < p \leq 20$	—	—	—	3.9	—	—	—	—	5.9	—
$5 < p \leq 10$	0.4	—	—	7.8	—	0.4	—	—	4.3	—
$2 < p \leq 5$	—	—	—	12.5	—	—	—	—	11.3	—
$0 < p \leq 2$	—	—	—	8.6	0.4	—	—	—	11.7	—
same precision	96.1	96.5	96.5	57.8	96.1	89.8	90.2	90.2	53.5	90.2
unknown	3.5	3.5	3.5	3.5	3.5	9.8	9.8	9.8	9.8	9.8

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
both timed out	1.6	5.9	3.5	9.8
degradation > 1	0.8	1.1	2.0	0.8
$0.5 <$ degradation ≤ 1	—	1.1	1.2	1.6
$0.2 <$ degradation ≤ 0.5	0.5	0.8	2.0	2.0
$0.1 <$ degradation ≤ 0.2	1.3	2.2	1.2	2.0
same time	92.2	88.4	90.2	83.2
$0.1 <$ improvement ≤ 0.2	0.8	—	—	—
$0.2 <$ improvement ≤ 0.5	1.1	0.3	—	—
$0.5 <$ improvement ≤ 1	0.3	—	—	—
improvement > 1	1.3	0.3	—	0.8

Total time class	Goal Independent				Goal Dependent			
	without SI		with SI		without SI		with SI	
	B	E	B	E	B	E	B	E
timed out	1.6	1.6	5.9	5.9	3.5	3.5	9.8	9.8
$t > 10$	9.7	9.7	8.6	8.6	7.4	7.4	7.4	7.4
$5 < t \leq 10$	1.3	1.3	1.9	1.9	2.0	2.0	2.0	2.0
$1 < t \leq 5$	6.5	6.5	4.0	4.3	3.1	3.1	4.7	4.7
$0.5 < t \leq 1$	3.0	3.0	4.6	4.3	3.5	3.9	2.7	2.7
$0.2 < t \leq 0.5$	6.5	7.3	8.9	8.9	9.8	9.8	9.8	9.4
$t \leq 0.2$	71.5	70.7	66.1	66.1	70.7	70.3	63.7	64.1

Table 8.8: $(Pos \times SFL_2)$ vs $(Pos \times SGFL_2)$.

could improve the concrete unification procedure.

The first case applies in the context of occurs-check reduction [CKS96, Søn86]. Ground-or-freeness can be of help for this application, since a unification between two ground-or-free variables is occurs-check free. Note that this is an improvement with respect to the technique used in [CKS96], since it is not required that the two considered variables are independent. As a second application, ground-or-freeness can be useful to replace the full concrete unification procedure by a simplified version. Since a ground-or-free term is either ground or free, a *single* run-time test for freeness will discriminate between the two cases: if this test succeeds, unification can be implemented by a single assignment; if the test fails, any specialized code for unification with a ground term can be safely invoked. In particular, when unifying two ground-or-free variables that are not free at run-time, the full unification procedure can be replaced by a simpler test for equivalence.

8.7 More Precise Exploitation of Linearity

In [Kin94], A. King proposes a domain for sharing analysis that performs a quite precise tracking of linearity. Roughly speaking, each sharing group in a sharing set carries its own linearity information. In contrast, in the approach of [Lan90], which is the one usually followed, a set of definitely linear variables is recorded along with each sharing set. The proposal in [Kin94] gives rise to a domain that is quite different from the ones presented here. Since [Kin94] does not provide an experimental evaluation and we are unaware of any subsequent work on the subject, the question whether this more precise tracking of linearity is actually worthwhile (both in terms of precision and efficiency) seems open. What interests us here is that some part of the theoretical work presented in [Kin94] may be usefully applied even in the more classical treatments of linearity, such as the one being used in the domain combination *SFL*. As far as we can tell, this fact was first noted in [BZH00].

Informally, [Kin94, Lemma 5, point 3] states the following: suppose that we are going to unify the linear term s with the (possibly non-linear) term t, where $\mathrm{vars}(s) \cap \mathrm{vars}(t) = \varnothing$; then, the unification will only produce a very restricted form of sharing; namely, two different variables occurring in s will not share the same variable with a variable occurring only once in t.

In order to show how this result can be exploited even on the domain *SFL*, we need to be more formal. Given the abstract element $d = \langle sh, f, l \rangle$, let $x \in (l \setminus f)$ be a non-free but linear variable and let t be such that $\mathrm{ind}_d(x, t)$ holds but $\mathrm{lin}_d(t)$ does not hold. Let also

$$R_{-} = \overline{\mathrm{rel}}\bigl(\{x\} \cup \mathrm{vars}(t), sh\bigr), \qquad R_x = \mathrm{rel}\bigl(\{x\}, sh\bigr), \qquad R_t = \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr).$$

In such a situation, when abstractly evaluating the binding $x \mapsto t$, the standard amgu_S operator gives the set-sharing component

$$sh' = R_{-} \cup \mathrm{bin}(R_x^{\star}, R_t).$$

	Goal Independent				Goal Dependent			
Prec. class	I	G	F	L	I	G	F	L
$p > 20$	0.3	—	—	—	—	—	—	—
$2 < p \leq 5$	—	—	—	—	0.4	—	—	—
same precision	93.8	94.1	94.1	94.1	89.8	90.2	90.2	90.2
unknown	5.9	5.9	5.9	5.9	9.8	9.8	9.8	9.8

Time difference class	% benchmarks	
	Goal Ind.	Goal Dep.
both timed out	5.9	9.8
degradation > 1	3.0	3.1
$0.5 <$ degradation ≤ 1	0.3	1.6
$0.2 <$ degradation ≤ 0.5	2.2	1.2
$0.1 <$ degradation ≤ 0.2	1.9	3.1
same time	86.0	80.1
$0.1 <$ improvement ≤ 0.2	—	—
$0.2 <$ improvement ≤ 0.5	0.3	0.4
$0.5 <$ improvement ≤ 1	—	0.4
improvement > 1	0.5	0.4

Total time class	Goal Ind.		Goal Dep.	
	B	E	B	E
timed out	5.9	5.9	9.8	9.8
$t > 10$	8.6	8.6	7.4	7.4
$5 < t \leq 10$	1.9	1.9	2.0	2.0
$1 < t \leq 5$	4.0	4.6	4.7	4.7
$0.5 < t \leq 1$	4.6	4.0	2.7	2.7
$0.2 < t \leq 0.5$	8.9	8.3	9.8	9.4
$t \leq 0.2$	66.1	66.7	63.7	64.1

Table 8.9: The effect of enhanced linearity (with structural info).

Suppose the set $\mathrm{vars}(t)$ is partitioned into the two components V_t^{l} and V_t^{nl}, where V_t^{l} is the set of the "good" variables, that is, those variables that can occur only once in t. Formally, let

$$
\begin{aligned}
V_t^{\mathrm{l}} &\stackrel{\mathrm{def}}{=} \left\{ y \in \mathrm{vars}(t) \;\middle|\; \begin{array}{l} y \in l \\ y \Subset \mathrm{mvars}(t) \implies y \notin \mathrm{vars}(sh) \\ \forall z \in \mathrm{vars}(t) : \bigl(y = z \lor \mathrm{ind}_d(y,z)\bigr) \end{array} \right\}; \\
V_t^{\mathrm{nl}} &\stackrel{\mathrm{def}}{=} \mathrm{vars}(t) \setminus V_t^{\mathrm{l}}; \\
R_t^{\mathrm{l}} &\stackrel{\mathrm{def}}{=} \mathrm{rel}(V_t^{\mathrm{l}}, sh); \\
R_t^{\mathrm{nl}} &\stackrel{\mathrm{def}}{=} \mathrm{rel}(V_t^{\mathrm{nl}}, sh).
\end{aligned}
$$

Note that $R_t^{\mathrm{nl}} \neq \varnothing$, otherwise we would have that $\mathrm{lin}_d(t)$ holds. If also $R_t^{\mathrm{l}} \neq \varnothing$ then the standard amgu_S can be replaced by an improved version (that we denote by amgu_K) computing the following set-sharing component:

$$sh'_K = R_- \cup \mathrm{bin}(R_x, R_t^{\mathrm{l}}) \cup \mathrm{bin}(R_x^\star, R_t^{\mathrm{nl}}).$$

As a consequence of King's result, only R_t^{nl} has to be combined with $R_x^\star$, while R_t^{l} (that is, the relevant component of sh with respect to the "good" variables V_t^{l}) can be combined with just R_x, without the star-union.

Example 8.5 *Suppose $VI = \{v, w, x, y, z\}$ is the set of variables of interest and consider the SFL element*

$$d \stackrel{\mathrm{def}}{=} \bigl\langle \{vx, wx, y, z\}, \{v, w, y\}, \{v, w, x, y\} \bigr\rangle$$

with the binding $x \mapsto f(y,z)$. Note that all the applicability conditions previously specified are met: in particular, $t = f(y,z)$ is possibly non-linear because $z \notin l$. As $R_x = \{vx, wx\}$ and $R_t = \{y, z\}$, a standard analysis would compute

$$
\begin{aligned}
d' &= \mathrm{amgu}_S\bigl(d, x \mapsto f(y,z)\bigr) \\
&= \bigl\langle \{vwxy, vwxz, vxy, vxz, wxy, wxz\}, \varnothing, \{y\} \bigr\rangle.
\end{aligned}
$$

On the other hand, since $V_t^{\mathrm{l}} = \{y\}$ and $V_t^{\mathrm{nl}} = \{z\}$, the enhanced analysis would compute

$$
\begin{aligned}
d'_K &= \mathrm{amgu}_K\bigl(d, x \mapsto f(y,z)\bigr) \\
&= \bigl\langle \{vwxz, vxy, vxz, wxy, wxz\}, \varnothing, \{y\} \bigr\rangle.
\end{aligned}
$$

Note that d'_K does not include the sharing group $vwxy$. This means that, if in the sequel of the computation variable z is bound to a ground term, then variables v and w will be detected as definitely independent. This independence is not captured when using the standard amgu_S*, since d' includes the sharing group $vwxy$ and therefore the variables v*

and w will potentially share even after grounding z.

The experimental evaluation for this enhancement is reported in Table 8.9. Note that we only report the comparison for the analyses performed with structural information turned on. This is required in order to check the applicability conditions triggering the enhanced abstract unification operator.

As far as the efficiency of the analysis is concerned, the situation is similar to the one described in the previous section for the ground-or-free enhancement: the overall slow-down, computed on the two variants of the analysis, is less than 78 seconds. Precision improvements are observed for only one program, which is a synthetic benchmark such as the above example. Despite its limited practical relevance, this result demonstrates that the operator aunify_S is *not* optimal, even when all the possible orderings of the non-grounding bindings are tried. Note that, in our opinion, the result stated in [Kin94, Lemma 5] can be made even stronger. In particular, we are currently investigating whether or not a similar enhancement of the abstract unification operator can be obtained even when the terms being unified possibly share.

8.8 Set-Sharing and Freeness

As noted by several authors, the standard combination of SH and freeness is not optimal. G. Filé [Fil94] formally identified the reduced product of these domains and proposed an improved abstract unification operator. This new operator exploits two properties that hold for the most precise abstract description of a *single* concrete substitution:

1. each free variable occurs in exactly one sharing group;
2. two free variables occur in the same sharing group if and only if they are aliases (i.e., they have become the same variable).

When considering the general case, where sets of concrete substitutions come into play, property 1 can be used to (partially) recover disjunctive information. In particular, it is possible to decompose the overall abstract description into a set of descriptions that necessarily come from different computation paths, each one satisfying property 1. The abstract unification procedure can thus be computed separately on each component, and the results of each subcomputation are then joined to give the final description. As such components are more precise than the original description (they possibly contain more ground variables and less sharing pairs), precision gains can be obtained.

Furthermore, by exploiting property 2 on each component, it is possible to correctly infer that for some of them the computation will fail due to a functor clash (or even due to the occurs-check, when considering a system working on finite trees). Note that a similar improvement is possible even without decomposing the abstract description. As an example, consider an abstract element such as the following:

$$d = \bigl\langle \{xy, u, v\}, \{x, y\}, \{x, y\} \bigr\rangle,$$

and suppose that the computation of the abstract semantics has to approximate the sequence of bindings $\langle x \mapsto f(u), y \mapsto g(v)\rangle$. Since the sharing group xy is the only one where the free variables x and y occur, property 2 states that x and y are indeed the same variable in all the concrete computation states described by $d \in \mathit{SFL}$. Thus, it can be safely concluded that all the corresponding concrete computations will fail due to the functor clash. In the same situation, when considering a binding such as $x \mapsto f(y)$, all concrete computations will fail due to the occurs-check, provided this is performed by the concrete unification procedure.

As was the case for the reduced product between Pos and SH, the interaction between the enhanced abstract unification operator and the elimination of ρ_{PSD}-redundant elements can lead to results that are not correct.

Example 8.6 *Let $\mathit{VI} = \{w, x, y, z\}$ and consider the concrete element $\Sigma \in \mathcal{D}^\flat$ such that $\Sigma = \wp(\sigma)$, where $\sigma = \{x \mapsto v, y \mapsto v, z \mapsto v\}$ (note that $v \notin \mathit{VI}$). By Definition 6.11, letting $d = \alpha_S(\Sigma) = \langle sh, f, l\rangle \in \mathit{SFL}$, we obtain that $sh = \{w, x, xy, xyz, xz, y, yz, z\}$ and $f = l = \mathit{VI}$. Suppose that the implementation represents d by using the reduced element $d_{\mathrm{red}} = \langle sh_{\mathrm{red}}, f, l\rangle$, where $sh_{\mathrm{red}} = sh \setminus \{xyz\}$, so that $sh = \rho_{\mathit{PSD}}(sh_{\mathrm{red}})$.*

According to the specification of the enhanced operator, the abstract element d_{red} can be decomposed into the following four components:

$$c_1 = \big\langle \{w, x, y, z\}, f, l\big\rangle, \qquad c_3 = \big\langle \{w, xz, y\}, f, l\big\rangle,$$
$$c_2 = \big\langle \{w, x, yz\}, f, l\big\rangle, \qquad c_4 = \big\langle \{w, xy, z\}, f, l\big\rangle.$$

Let $\big(x \mapsto f(y, w)\big) \in \mathit{Bind}$ and, for each $i \in \{1, \ldots, 4\}$, consider the computation of $c'_i = \langle sh'_i, f'_i, l'_i\rangle = \mathrm{amgu}_S\big(c_i, x \mapsto f(y, w)\big)$. Since both $\mathrm{lin}_{c_i}(x)$ and $\mathrm{lin}_{c_i}\big(f(y, w)\big)$ hold, the new linearity component will be computed as follows:

$$l'_i = l \setminus \big(\mathrm{share\_with}_{c_i}(x) \cap \mathrm{share\_with}_{c_i}\big(f(y, w)\big)\big).$$

In all four cases, we have $z \in l'_i$, so that variable z keeps its linearity even after merging the results of the four subcomputations in a single abstract description.

In contrast, when performing the same computation with the original abstract description d, in the decomposition phase we also obtain a fifth component,

$$c_5 = \big\langle \{w, xyz\}, f, l\big\rangle.$$

It is easy to observe that, in $c'_5 = \langle sh'_5, f'_5, l'_5\rangle = \mathrm{amgu}_S\big(c_5, x \mapsto f(y, w)\big)$, we have $z \notin l'_5$, so that z loses its linearity when merging the five results in a single abstract description.

Note that, in one of the concrete computation paths, we would have computed

$$\sigma' = \big\{x \mapsto f(x, w), y \mapsto f(y, w), z \mapsto f(z, w)\big\} \in \mathrm{mgs}\big(\sigma, x \mapsto f(y, w)\big).$$

Since $z \notin \mathrm{lvars}(\sigma')$, the result obtained using the abstract description d_{red} is not correct.

Goal Independent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
$p > 20$	0.3	—	—	—	—	—	—	—
$10 < p \leq 20$	—	—	—	0.3	—	—	—	0.3
$0 < p \leq 2$	0.3	—	—	0.5	3.0	—	—	0.5
same precision	96.8	97.3	97.3	96.5	87.6	90.6	90.6	89.8
unknown	2.7	2.7	2.7	2.7	9.4	9.4	9.4	9.4

Goal Dependent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
$5 < p \leq 10$	—	—	—	0.4	—	—	—	0.4
same precision	96.5	96.5	96.5	96.1	89.5	89.5	89.5	89.1
unknown	3.5	3.5	3.5	3.5	10.5	10.5	10.5	10.5

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
both timed out	1.6	5.9	3.5	9.8
degradation > 1	9.9	12.1	2.0	1.6
$0.5 <$ degradation ≤ 1	0.8	1.6	1.2	1.2
$0.2 <$ degradation ≤ 0.5	2.4	2.4	2.7	1.2
$0.1 <$ degradation ≤ 0.2	1.3	1.3	1.2	1.6
same time	83.3	76.1	89.5	84.0
$0.1 <$ improvement ≤ 0.2	—	—	—	—
$0.2 <$ improvement ≤ 0.5	—	—	—	—
$0.5 <$ improvement ≤ 1	0.3	—	—	—
improvement > 1	0.3	0.5	—	0.8

Total time class	Goal Independent				Goal Dependent			
	without SI		with SI		without SI		with SI	
	B	E	B	E	B	E	B	E
timed out	1.6	2.7	5.9	9.4	3.5	3.5	9.8	10.5
$t > 10$	9.7	9.9	8.6	7.3	7.4	7.4	7.4	6.6
$5 < t \leq 10$	1.3	1.6	1.9	1.6	2.0	2.0	2.0	2.0
$1 < t \leq 5$	6.5	5.6	4.0	5.4	3.1	3.1	4.7	4.7
$0.5 < t \leq 1$	3.0	2.7	4.6	3.5	3.5	4.3	2.7	2.7
$0.2 < t \leq 0.5$	6.5	7.3	8.9	8.3	9.8	9.4	9.8	9.0
$t \leq 0.2$	71.5	70.2	66.1	64.5	70.7	70.3	63.7	64.5

Table 8.10: The enhanced combination with freeness.

As already observed in Section 8.5, the above correctness problem lies not in the SFL_2 domain itself, but rather in the optimized implementation which removes the redundant elements from the set-sharing description. A correct implementation can still be obtained by computing the closure $\rho_{PSD}(d_{\mathrm{red}})$ before the decomposition phase.

By refining Example 8.6, we show that the domain *SFL* is strictly more precise than SFL_2 when using the enhanced operator.

Example 8.7 *Reconsider Example 8.6 and let $\Sigma' \in \mathcal{D}^\flat$ be such that $\Sigma' = \Sigma \setminus \{\sigma\}$. By Definition 6.11, we have $d_{\mathrm{red}} = \alpha_S(\Sigma')$. Thus, as observed in Example 8.6, an analysis using the domain SFL with the enhanced abstract semantics operator will conclude that z is definitely linear even after the evaluation of the binding $x \mapsto f(y, w)$. Note that this is correct, since $\sigma \notin \Sigma'$, so that the substitution σ' of Example 8.6 will not be computed.*

In contrast, a correct implementation of the enhanced operator on the domain SFL_2 will compute the closure $\rho_{PSD}(d_{\mathrm{red}}) = d$, before the decomposition; therefore, it will also produce the component c_5 and unnecessarily lose the linearity of variable z. Basically, the domain SFL_2 cannot distinguish between the concrete elements Σ and Σ'.

We implemented the first of the two ideas by Filé on the usual base domain ($Pos \times SFL_2$). As discussed above, our optimized implementation of the enhanced operator may yield results that are not correct. However, a non-optimized implementation or even one based on the domain ($Pos \times SFL$) would probably result in an unbearable number of time-outs, therefore making the overall precision comparison meaningless. Our precision results can thus be considered an over-estimation of the actual improvements that could be obtained by a correct implementation.

The results of the comparison, reported in Table 8.10, show that precision improvements are observed on both variable independence and linearity, in particular for goal independent analyses. As a matter of fact, for goal dependent analyses, the only precision improvements are obtained when analysing the synthetic benchmark `hvars`, which substantially corresponds to Example 8.7. When looking at the time comparisons, it should be observed that, even though we use the domain SFL_2 where we should have used the inefficient *SFL*, still the analysis of many programs had to be stopped because of the combinatorial explosion in the number of possible decompositions of the abstract description.

In principle, such an approach to the recovery of disjunctive information can be pursued beyond the integration of sharing with freeness. In fact, by exploiting the ground-or-free information as in Section 8.6, it is possible to obtain decompositions where each component contains *at most one* occurrence (in contrast with the *exactly one* occurrence of Filé's idea) of each ground-or-free variable. In each component, the ground-or-free variable could then be "promoted" as either a ground variable (if it does not occur in the sharing groups of that component) or as a free variable (if it occurs in exactly one sharing group).

It would also be interesting to experiment with the second idea of Filé. However, such a goal would require a big implementation effort, since at present there is no easy way to incorporate this enhancement into the modular design of the CHINA analyzer.

8.9 Tracking Compoundness

In [BCM94a, BCM94b], M. Bruynooghe et al. consider the combination of the standard set-sharing, freeness and linearity domains with compoundness information. A variable is said to be *definitely compound* if it is always bound to a non-variable term. Note that, according to this definition, a variable bound to a functor constant is compound. As for freeness and linearity, compoundness is represented by the set of variables that definitely enjoy this property.

As discussed in [BCM94a, BCM94b], compoundness information is useful in its own right for clause indexing. Here though, the focus is on improving sharing information, so that the question to be answered is: can the tracking of compoundness improve the sharing analysis itself? This question is also considered in [BCM94a, BCM94b] where a technique is proposed that exploits the combination of sharing, freeness and compoundness. This technique relies on the presence of the occurs-check.

Informally, consider the binding $x \mapsto t$ together with an abstract description where x is a free variable, t is a compound term and x *definitely* shares with t. Since x is free, x is aliased to one of the variables occurring in t. As a consequence, the execution of the binding $x \mapsto t$ will fail due to the occurs-check. In a more general case, when only *possible* sharing information is available, the precision of the abstract description can be safely improved by removing, just before computing the abstract binding, all the sharing groups containing both x and a variable in t. In addition, if this reduction step removes all the sharing groups containing a free variable, then it can be safely concluded that the computation will fail.

To see how this works in practice, consider the binding $x \mapsto f(y, z)$ and the description $d_1 \stackrel{\text{def}}{=} \langle sh_1, f_1, l_1 \rangle \in \mathit{SFL}$ such that

$$\begin{aligned} sh_1 &\stackrel{\text{def}}{=} \{wx, xy, xz, y, z\}, \\ f_1 &\stackrel{\text{def}}{=} \{x\}, \\ l_1 &\stackrel{\text{def}}{=} \{w, x, y, z\}. \end{aligned}$$

Since x is free and $f(y, z)$ is compound, the sharing groups xy and xz can be removed so that the amgu_S computation will give the set-sharing and linearity components

$$\begin{aligned} sh_1' &\stackrel{\text{def}}{=} \{wxy, wxz\}, \\ l_1' &\stackrel{\text{def}}{=} \{w, x, y, z\} \end{aligned}$$

instead of the less precise

$$\begin{aligned} sh_1' &\stackrel{\text{def}}{=} \{wxy, wxz, xy, xyz, xz\}, \\ l_1' &\stackrel{\text{def}}{=} \{w\}. \end{aligned}$$

Note that the precision improvement of this particular example could also be obtained by the technique introduced by Filé and discussed in the previous section. The reason is that x is unified with the term $f(y,z)$, which is "explicitly" compound. However, if the term was "implicitly" compound (i.e., if it was an abstract variable known to represent compound terms) then the technique by Filé would not be applicable. For example, consider the binding $x \mapsto y$ and the description $d_2 \stackrel{\mathrm{def}}{=} \langle sh_2, f_2, l_2 \rangle \in \mathit{SFL}$ such that

$$\begin{aligned} sh_2 &\stackrel{\mathrm{def}}{=} \{wx, xyz, y\}, \\ f_2 &\stackrel{\mathrm{def}}{=} \{x\}, \\ l_2 &\stackrel{\mathrm{def}}{=} \{w, x, y, z\} \end{aligned}$$

supplemented by a compoundness component ensuring that y is compound. Then the sharing group xyz can be removed so that the amgu_S will compute

$$\begin{aligned} sh_2' &\stackrel{\mathrm{def}}{=} \{wxy\}, \\ l_2' &\stackrel{\mathrm{def}}{=} \{w, x, y, z\} \end{aligned}$$

instead of

$$\begin{aligned} sh_2' &\stackrel{\mathrm{def}}{=} \{wxy, wxyz, xyz\}, \\ l_2' &\stackrel{\mathrm{def}}{=} \{w\}. \end{aligned}$$

To see how a knowledge of the compoundness can be used to identify definite failure, consider the binding $x \mapsto f(y,z)$ and the description $d_3 \stackrel{\mathrm{def}}{=} \langle sh_3, f_3, l_3 \rangle \in \mathit{SFL}$ such that

$$\begin{aligned} sh_3 &\stackrel{\mathrm{def}}{=} \{wxy, wxz, x, y, z\}, \\ f_3 &\stackrel{\mathrm{def}}{=} \{w, x\}, \\ l_3 &\stackrel{\mathrm{def}}{=} \{w, x, y, z\}. \end{aligned}$$

As in the examples above, variable x is free and term $t \stackrel{\mathrm{def}}{=} f(y,z)$ is compound so that, by applying the reduction step, we can remove the sharing groups wxy and wxz. However, this has removed all the sharing groups containing the free variable w, resulting in an inconsistent computation state.

We did not implement this technique, since it is only correct for the analysis of systems performing the occurs-check, whereas we are targeting at the analysis of systems possibly omitting it. Nonetheless, an experimental evaluation would be interesting for assessing how much this precision improvement can affect the accuracy of applications such as occurs-check reduction.

8.10 Summary

In this chapter we have investigated eight enhanced sharing analysis techniques that, at least in principle, have the potential for improving the precision of the sharing information over and above that obtainable using the domain *SFL*. These techniques either make a better use of the already available sharing information, by defining more powerful abstract semantic operators, or combine this sharing information with that captured by other domains. Our work has been systematic since, to the best of our knowledge, we have considered all the proposals that have appeared in the literature: that is, better exploitation of groundness, freeness, linearity, compoundness, and structural information.

Using the CHINA analyzer, seven of the eight enhancements have been experimentally evaluated. Because of the availability of a very large benchmark suite, including several programs of respectable size, the precision results are as conclusive as possible and provide an almost complete account of what is to be expected when analyzing any real program using these domains.

The results demonstrate that good precision improvements can be obtained with the inclusion of explicit structural information. For the groundness domain *Pos*, several good reasons have been given as to why it should be combined with set-sharing. As for the remaining proposals, it is hard to justify them as far as the precision of the analysis is concerned. Regarding the efficiency of the analysis, it has been explained why the reported time comparisons can be considered as upper bounds to the additional cost required by the inclusion of each technique. Moreover, it has been argued that, from this point of view, the addition of a 'ground-or-free' mode and the more precise exploitation of linearity are both interesting.

No further positive indications can be derived from the precision and time comparisons of the remaining techniques. In particular, it has not been possible to identify a good heuristic for the reordering of the non-grounding bindings. The experimentation suggests that sensible precision improvements cannot be expected from this technique. When considering these negative results, the reader should be aware that the precision gains are measured with respect to an analysis using the domain $(Pos \times SFL_2)$ which, to our knowledge, is the most accurate sharing analysis tool ever implemented.

The experimentation reported in this chapter resulted in both positive and negative indications. We believe that all of these will provide the right focus in the design and development of useful tools for sharing analysis.

Chapter 9

Widenings for Set-Sharing

The experimental evaluation reported in Chapter 7 has shown that, when considering the bigger or more involved programs in our benchmark suite, both the analysis using the domain *SFL* and the one using its non-redundant abstraction *PSD*, incur significant efficiency problems. In this chapter, we study the problem of a scalable *and* precise sharing analysis for logic programs. We define a simple domain schema for sharing analysis that supports the implementation of several widening techniques. Using this schema, we transform the idea underlying the sharing domain of C. Fecht [Fec96a, Fec96b] into a widening operator on the set-sharing domain. Experimentation shows that the new analysis scales really well: efficiency problems are solved, while the rare precision losses registered are usually small.

Note: this chapter contains an extract of the results in [ZBH99a, ZBH99b].

9.1 The Scalability of the Analysis

By moving from the classical set-sharing domain *SH* to its non-redundant version *PSD*, we have achieved a significant improvement in the efficiency of the analysis. This applies both from a theoretical point of view (the worst-case complexity of the amgu operator on *PSD* is polynomial in the number of sharing groups of the input description) and from a practical point of view (as noted in Chapter 7, speed-ups of two orders of magnitude have been observed). However, we have not provided a fully scalable set-sharing analyzer: the analysis of some programs still requires too much time and/or memory space.

When facing such a situation, two possible solutions can be adopted. The first one is to revert to a simpler abstract domain, characterized by more efficient abstract semantics operators. While this approach is very likely to remove all efficiency problems, it is also possible that such a result will be obtained at the expense of precision. In particular, this solution can cause precision losses even when analysing those programs for which the more refined abstract domain had no efficiency problem at all. This is the key observation leading to the second solution: one only needs to avoid the negative effects of exponential complexity when they come into play. Such a behavior can be achieved, as indicated by Cousot and Cousot [CC92b], by using the refined domains together with widening/narrowing operators. With this technique we can try to limit precision losses to those cases where we cannot afford the complexity implied by the refined domains.

Unfortunately, the design of widening operators tends to escape the realm of theoretical analysis, and thus, in our opinion, it has not been studied enough. In fact, the development of successful widening operators requires, perhaps more than other things, extensive experimentation.

9.1.1 Fecht's Domain

C. Fecht [Fec96a, Fec96b] proposed a new domain for sharing analysis, denoted $\downarrow SH$, based on an abstraction of the usual Jacobs and Langen domain. The carrier of $\downarrow SH$ is the same as SH, but the concretization of a set of variables in $\downarrow SH$ is equivalent to the concretization of its power set in SH. Since the resulting approximation is rather crude, the domain $\downarrow SH$ is combined with the groundness information encoded by Pos and with the linearity information L (a set of linear variables, as in the case of the domain SFL). As a matter of fact, the above combination is isomorphic to the combination of Pos with the pair-sharing domain ASub [CKS96, Søn86]. The advantage of $\downarrow SH$ with respect to SH is twofold: first, an element of the domain can be normalized by removing all but the maximal sets, thereby reducing its size; second, the abstract operations are more efficient (but less precise) than those used for SH and its non-redundant version PSD.

In [ZBH99b] we compared our implementation of $Pos + \downarrow SH + F + L$ (also including freeness information) with respect to the domain $Pos + SFL_2$: while confirming the dramatic speed-up observed by Fecht, we found a precision loss on as many as 20% of the benchmarks considered; moreover, the biggest loss was as high as 40% of the quantity measured.

We note that Fecht did not present the domain $\downarrow SH$ as a widening for the set-sharing domain and neither did he discuss how a widening based on his proposal might be achieved. Indeed the approach of Fecht falls under the category "use a simpler domain" which, as clearly explained in [CC92b], is both contrary and inferior to the approach "use a complex domain with widening" that we are advocating.

9.2 A New Representation for Set-Sharing

We now introduce a new representation for set-sharing. It is made up of two components: one represents all possible subsets of each of its elements, as was the case for $\downarrow SH$, while the other records the set-sharing as before.

9.2.1 Clique Groups and Clique Sets

A sharing group $C \in SG$ that occurs in the first component cl of an abstract element represents all possible sharing groups between its variables: as a consequence we will call it a clique group or, more simply, a *clique* and call cl, itself, a *clique set*.

Definition 9.1 (Clique set.) *A* clique set cl *is an element of* $CL \stackrel{\text{def}}{=} SH$.

For each clique set, the corresponding meaning in terms of sharing groups can be computed by downward closure.

Definition 9.2 (Downward closure of clique sets.) *The functions* $\downarrow\colon SG \to SH$ *and* $\downarrow\colon CL \to SH$ *are defined, for each* $C \in SG$ *and each* $cl \in CL$*, by*

$$\downarrow C \stackrel{\text{def}}{=} \wp(C) \setminus \{\varnothing\}, \qquad\qquad \downarrow cl \stackrel{\text{def}}{=} \bigcup_{C \in cl} \downarrow C.$$

Observe that $\downarrow \in \mathrm{uco}(SH)$. If $cl \in CL$ and $C \in SG$ then we say that C is *down-redundant* in cl if there exists $C' \in cl$ such that $C \subset C'$. The addition to or removal from a clique set of down-redundant elements makes no difference to the sharing groups that it represents. In the implementation, as we prefer to keep the clique sets as small as possible, down-redundant cliques are removed via a *normalization* function.

Definition 9.3 (Normalization.) *The normalization function* $\|\cdot\|\colon CL \to CL$ *is defined, for each* $cl \in CL$*, as*

$$\|cl\| \stackrel{\text{def}}{=} cl \setminus \{\, C \in cl \mid \exists C' \in cl \,.\, C \subset C' \,\}.$$

9.2.2 Combining Clique Sets with Sharing Sets

The elements of our new sharing domain have a clique set and a sharing set.

Definition 9.4 (The SH^{W} representation.) *The set* SH^{W} *is given by*

$$SH^{\mathsf{W}} \stackrel{\text{def}}{=} \bigl\{\, (cl, sh) \bigm| cl \in CL, sh \in SH \,\bigr\}$$

ordered by $\sqsubseteq_W$ *defined as follows, for each* $(cl_1, sh_1), (cl_2, sh_2) \in SH^{\mathsf{W}}$*:*

$$(cl_1, sh_1) \sqsubseteq_W (cl_2, sh_2) \iff (cl_1 \subseteq cl_2) \land (sh_1 \subseteq sh_2).$$

With this ordering, SH^{W} *is a complete lattice.*

It is possible to generalize to the elements of SH^{W} the concept of down-redundancy introduced for cliques. Thus, we define an overloading of the normalization function removing these redundancies from a description in SH^{W}.

Definition 9.5 (Normalization and downward closure on SH^{W}.) *The normalization function* $\|\cdot\|\colon SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ *is defined, for each* $(cl, sh) \in SH^{\mathsf{W}}$*, as*

$$\|(cl, sh)\| \stackrel{\text{def}}{=} \bigl(\|cl\|, sh \setminus \downarrow cl\bigr).$$

The downward closure function $\downarrow(\cdot)\colon SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ *is defined, for each* $(cl, sh) \in SH^{\mathsf{W}}$*, as*

$$\downarrow(cl, sh) \stackrel{\text{def}}{=} (\downarrow cl, \downarrow cl \cup sh).$$

Note that $\downarrow(\cdot) \in \mathrm{uco}(SH^{\mathsf{W}})$ is the dual operation with respect to normalization. This operator implicitly defines the following partial order: for any $shw_1, shw_2 \in SH^{\mathsf{W}}$, $shw_1 \preceq_W shw_2$ if and only if $\downarrow(shw_1) \sqsubseteq_W \downarrow(shw_2)$.

The information content, in terms of sharing groups, of an element of SH^{W} is defined as the sharing set component of its downward closure.

Definition 9.6 (Meaning of SH^{W}.) *The function $\mathcal{I}(\cdot) \colon SH^{\mathsf{W}} \to SH$ is defined, for each $(cl, sh) \in SH^{\mathsf{W}}$, as*

$$\mathcal{I}\big((cl, sh)\big) \stackrel{\mathrm{def}}{=} \downarrow cl \cup sh.$$

9.3 The Abstract Operators

Many of the auxiliary abstract operators on the set-sharing domain SH can be easily generalized to work on elements of SH^{W}. Using these as building blocks, it is then possible to systematically derive the specification of the abstract unification operator for the new domain SH^{W}.

Definition 9.7 (Operators over SH^{W}.) *Let $(cl, sh), (cl_i, sh_i) \in SH^{\mathsf{W}}$, where $i = 1, 2$, and $V \in \wp(VI)$. The operators $\mathrm{rel}^{\mathsf{W}}, \overline{\mathrm{rel}}^{\mathsf{W}} \colon \wp(VI) \times SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ are defined as*

$$\begin{aligned}
\mathrm{rel}^{\mathsf{W}}\big(V, (cl, sh)\big) &\stackrel{\mathrm{def}}{=} \big(\mathrm{rel}(V, cl), \mathrm{rel}(V, sh)\big),\\
\overline{\mathrm{rel}}^{\mathsf{W}}\big(V, (cl, sh)\big) &\stackrel{\mathrm{def}}{=} \big(\overline{\mathrm{rel}}^{\mathsf{CL}}(V, cl), \overline{\mathrm{rel}}(V, sh)\big),
\end{aligned}$$

where $\overline{\mathrm{rel}}^{\mathsf{CL}} \colon \wp(VI) \times CL \to CL$ is defined as

$$\overline{\mathrm{rel}}^{\mathsf{CL}}(V, cl) \stackrel{\mathrm{def}}{=} \{\, C \setminus V \mid C \in cl \,\} \setminus \{\varnothing\}.$$

The union and binary union operators $\cup^{\mathsf{W}}, \mathrm{bin}^{\mathsf{W}} \colon SH^{\mathsf{W}} \times SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ are defined as

$$\begin{aligned}
(cl_1, sh_1) \cup^{\mathsf{W}} (cl_2, sh_2) &\stackrel{\mathrm{def}}{=} \big(cl_1 \cup cl_2, sh_1 \cup sh_2\big),\\
\mathrm{bin}^{\mathsf{W}}\big((cl_1, sh_1), (cl_2, sh_2)\big) &\stackrel{\mathrm{def}}{=} \big(\mathrm{bin}(cl_1, cl_2) \cup \mathrm{bin}(cl_1, sh_2) \cup \mathrm{bin}(sh_1, cl_2), \mathrm{bin}(sh_1, sh_2)\big).
\end{aligned}$$

The star-closure operator $(\cdot)^\star \colon SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ is defined as

$$(cl, sh)^\star \stackrel{\mathrm{def}}{=} \begin{cases} (\varnothing, sh^\star), & \text{if } cl = \varnothing;\\ \Big(\big\{\mathrm{vars}(cl) \cup \mathrm{vars}(sh)\big\}, \varnothing\Big), & \text{otherwise.} \end{cases}$$

The following theorem states that each operator on SH^{W} correctly approximates the corresponding operator on SH.

Theorem 9.8 *If $V \in \wp(VI)$ and $shw, shw_1, shw_2 \in SH^{\mathsf{W}}$, then*

$$\mathrm{rel}\bigl(V, \mathcal{I}(shw)\bigr) \subseteq \mathcal{I}\bigl(\mathrm{rel}^{\mathsf{W}}(V, shw)\bigr), \tag{9.1}$$
$$\overline{\mathrm{rel}}\bigl(V, \mathcal{I}(shw)\bigr) = \mathcal{I}\bigl(\overline{\mathrm{rel}}^{\mathsf{W}}(V, shw)\bigr), \tag{9.2}$$
$$\mathcal{I}(shw_1) \cup \mathcal{I}(shw_2) = \mathcal{I}\bigl(shw_1 \cup^{\mathsf{W}} shw_2\bigr), \tag{9.3}$$
$$\mathrm{bin}\bigl(\mathcal{I}(shw_1), \mathcal{I}(shw_2)\bigr) \subseteq \mathcal{I}\bigl(\mathrm{bin}^{\mathsf{W}}(shw_1, shw_2)\bigr), \tag{9.4}$$
$$\bigl(\mathcal{I}(shw)\bigr)^{\star} \subseteq \mathcal{I}\bigl(shw^{\star}\bigr). \tag{9.5}$$

By looking at the definition of the abstract operators and the corresponding correctness result, it can be seen that clique sets and sharing sets are treated differently: when working on the clique set component, sometimes we trade precision for efficiency; in contrast, when working on the sharing set component, efforts are made to preserve precision. In particular, when the clique set component is empty, the abstract operators of Definition 9.7 are equivalent to those introduced in Definition 3.42. This corresponds to the intuition behind the definition of the domain SH^{W}: we will move in the clique set component that part of the abstract description on which we allow for bigger precision losses, while keeping the other part in the sharing set component.

It is worth stressing that, thanks to the above correctness result, the specification of a correct abstract unification procedure working on the new set-sharing representation is straightforward. For instance, by considering the bare domain SH^{W} (i.e., without the addition of freeness and linearity information), we can define $\mathrm{amgu}^{\mathsf{W}} \colon SH^{\mathsf{W}} \times Bind \to SH^{\mathsf{W}}$ as follows:

$$\begin{aligned} &\mathrm{amgu}^{\mathsf{W}}(shw, x \mapsto t) \\ &\qquad \stackrel{\mathrm{def}}{=} \overline{\mathrm{rel}}^{\mathsf{W}}\bigl(\{x\} \cup \mathrm{vars}(t), shw\bigr) \cup^{\mathsf{W}} \mathrm{bin}^{\mathsf{W}}\Bigl(\mathrm{rel}^{\mathsf{W}}\bigl(\{x\}, shw\bigr)^{\star}, \mathrm{rel}^{\mathsf{W}}\bigl(\mathrm{vars}(t), shw\bigr)^{\star}\Bigr). \end{aligned}$$

It should also be clear that the above construction can be easily extended to domain combinations including freeness and linearity information, such as the domain *SFL*. After having defined $SFL^{\mathsf{W}} \stackrel{\mathrm{def}}{=} SH^{\mathsf{W}} \times F \times L$, all we need to do is to provide the approximations on SFL^{W} of the operators introduced in Definitions 6.31 and 6.33. This is an easy task for all but the predicate $\mathrm{ind}_d \colon HTerms^2 \to Bool$, because this one is defined in terms of the set intersection of sharing sets and we have not defined a set intersection operator on SH^{W}. However, the problem is easily solved by noting that $\mathrm{ind}_d(s, t)$ holds if and only if

$$\mathrm{rel}\Bigl(\mathrm{vars}(s), \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr)\Bigr) = \varnothing,$$

which can be easily approximated using the operator $\mathrm{rel}^{\mathsf{W}}$.

9.3.1 Proofs of Correctness

The following auxiliary lemma states the correctness of the abstract operators when they are restricted to work on clique sets.

Lemma 9.9 *Suppose $V \in \wp(VI)$ and $cl \in CL$. Then*

$$\operatorname{rel}(V, \downarrow cl) \subseteq \downarrow \operatorname{rel}(V, cl), \tag{9.6}$$

$$\overline{\operatorname{rel}}(V, \downarrow cl) = \downarrow \overline{\operatorname{rel}}^{\mathsf{CL}}(V, cl), \tag{9.7}$$

$$\downarrow cl_1 \cup \downarrow cl_2 = \downarrow(cl_1 \cup cl_2), \tag{9.8}$$

$$\operatorname{bin}(\downarrow cl_1, cl_2) \subseteq \downarrow \operatorname{bin}(cl_1, cl_2), \tag{9.9}$$

$$\operatorname{bin}(\downarrow cl_1, \downarrow cl_2) \subseteq \downarrow \operatorname{bin}(cl_1, cl_2), \tag{9.10}$$

$$(\downarrow cl)^\star = \downarrow(cl^\star). \tag{9.11}$$

Proof. Proof of (9.6): suppose that $S \in \operatorname{rel}(V, \downarrow cl)$. Then $V \cap S \neq \varnothing$ and there exists $C \in cl$ such that $S \subseteq C$. Thus $V \cap C \neq \varnothing$, so that $C \in \operatorname{rel}(V, cl)$. Hence $S \in \downarrow \operatorname{rel}(V, cl)$.

Proof of (9.7): we have $S \in \overline{\operatorname{rel}}(V, \downarrow cl)$ if and only if $S \neq \varnothing$, $V \cap S = \varnothing$ and there exists $C \in cl$ such that $S \subseteq C$. This holds if and only if $S \neq \varnothing$ and there exists $C \in cl$ such that $S \in \downarrow C \setminus V$, which is equivalent to $S \in \downarrow \overline{\operatorname{rel}}^{\mathsf{CL}}(V, cl)$.

Proof of (9.8): we have $S \in \downarrow cl_1 \cup \downarrow cl_2$ if and only if $S \neq \varnothing$ and there exists $C \in cl_1 \cup cl_2$ such that $S \subseteq C$. This holds if and only if $S \in \downarrow(cl_1 \cup cl_2)$.

Proof of (9.9): suppose that $S \in \operatorname{bin}(\downarrow cl_1, cl_2)$. Then $S = S_1 \cup C_2$, where $S_1 \neq \varnothing$, there exists $C_1 \in cl_1$ such that $S_1 \subseteq C_1$, and $C_2 \in cl_2$. Thus $S \subseteq C_1 \cup C_2 \in \operatorname{bin}(cl_1, cl_2)$. Hence $S \in \downarrow \operatorname{bin}(cl_1, cl_2)$.

As 'bin' is symmetric on its arguments, (9.10) is a corollary of (9.9).

Proof of (9.11): suppose $S \in (\downarrow cl)^\star$. Then there exist $n \geq 1$ and $\{S_1, \ldots, S_n\} \subseteq \downarrow cl$ such that $S = S_1 \cup \ldots \cup S_n$. Thus, for each $i \in \{1, \ldots, n\}$, there exists $C_i \in cl$ such that $S_i \subseteq C_i$, so that $S \subseteq C_1 \cup \ldots \cup C_n \stackrel{\mathrm{def}}{=} C$. Since $C \in cl^\star$, we obtain $S \in \downarrow(cl^\star)$. To prove the other inclusion, let now $S \in \downarrow(cl^\star)$. Thus $S \neq \varnothing$ and there exists $C \in cl^\star$ such that $S \subseteq C$. Then there exist $n \geq 1$ and $\{C_1, \ldots, C_n\} \subseteq cl$ such that $C = C_1 \cup \ldots \cup C_n$. Let $I = \{\, i \in \mathbb{N} \mid 1 \leq i \leq n, C_i \cap S \neq \varnothing \,\}$. Note that, as $S \neq \varnothing$, we have $I \neq \varnothing$. Also, for all $i \in I$, we have $C_i \cap S \in \downarrow cl$. Since $S = \cup_{i \in I}(C_i \cap S)$, we obtain $S \in (\downarrow cl)^\star$. □

Proof of Theorem 9.8 on page 238. For the proof, let us define $shw \stackrel{\mathrm{def}}{=} (cl, sh)$, $shw_1 \stackrel{\mathrm{def}}{=} (cl_1, sh_1)$, and $shw_2 \stackrel{\mathrm{def}}{=} (cl_2, sh_2)$.

Proof of (9.1): by applying case (9.6) of Lemma 9.9, we obtain

$$\begin{aligned}
\operatorname{rel}\bigl(V, \mathcal{I}(shw)\bigr) &= \operatorname{rel}(V, \downarrow cl \cup sh) \\
&= \operatorname{rel}(V, \downarrow cl) \cup \operatorname{rel}(V, sh) \\
&\subseteq \downarrow \operatorname{rel}(V, cl) \cup \operatorname{rel}(V, sh) \\
&= \mathcal{I}\Bigl(\bigl(\operatorname{rel}(V, cl), \operatorname{rel}(V, sh)\bigr)\Bigr) \\
&= \mathcal{I}\bigl(\operatorname{rel}^{\mathsf{W}}(V, shw)\bigr).
\end{aligned}$$

Proof of (9.2): by applying case (9.7) of Lemma 9.9, we obtain

$$\begin{aligned}
\overline{\mathrm{rel}}\bigl(V, \mathcal{I}(shw)\bigr) &= \overline{\mathrm{rel}}(V, \downarrow cl \cup sh) \\
&= \overline{\mathrm{rel}}(V, \downarrow cl) \cup \overline{\mathrm{rel}}(V, sh) \\
&= \downarrow \overline{\mathrm{rel}}^{\mathsf{CL}}(V, cl) \cup \overline{\mathrm{rel}}(V, sh) \\
&= \mathcal{I}\Bigl(\bigl(\overline{\mathrm{rel}}^{\mathsf{CL}}(V, cl), \overline{\mathrm{rel}}(V, sh)\bigr)\Bigr) \\
&= \mathcal{I}\bigl(\overline{\mathrm{rel}}^{\mathsf{W}}(V, shw)\bigr).
\end{aligned}$$

Proof of (9.3): by applying case (9.8) of Lemma 9.9, we obtain

$$\begin{aligned}
\mathcal{I}(shw_1) \cup \mathcal{I}(shw_2) &= (\downarrow cl_1 \cup sh_1) \cup (\downarrow cl_2 \cup sh_2) \\
&= \downarrow(cl_1 \cup cl_2) \cup (sh_1 \cup sh_2) \\
&= \mathcal{I}\bigl((cl_1 \cup cl_2, sh_1 \cup sh_2)\bigr) \\
&= \mathcal{I}(shw_1 \cup^{\mathsf{W}} shw_2).
\end{aligned}$$

Proof of (9.4): by applying cases (9.9) and (9.10) of Lemma 9.9, we obtain

$$\begin{aligned}
&\mathrm{bin}\bigl(\mathcal{I}(shw_1), \mathcal{I}(shw_2)\bigr) \\
&\quad = \mathrm{bin}\bigl(\downarrow cl_1 \cup sh_1, \downarrow cl_2 \cup sh_2\bigr) \\
&\quad = \mathrm{bin}(\downarrow cl_1, \downarrow cl_2) \cup \mathrm{bin}(\downarrow cl_1, sh_2) \cup \mathrm{bin}(sh_1, \downarrow cl_2) \cup \mathrm{bin}(sh_1, sh_2) \\
&\quad \subseteq \downarrow \mathrm{bin}(cl_1, cl_2) \cup \downarrow \mathrm{bin}(cl_1, sh_2) \cup \downarrow \mathrm{bin}(sh_1, cl_2) \cup \mathrm{bin}(sh_1, sh_2) \\
&\quad = \downarrow\bigl(\mathrm{bin}(cl_1, cl_2) \cup \mathrm{bin}(cl_1, sh_2) \cup \mathrm{bin}(sh_1, cl_2)\bigr) \cup \mathrm{bin}(sh_1, sh_2) \\
&\quad = \mathcal{I}\Bigl(\bigl(\mathrm{bin}(cl_1, cl_2) \cup \mathrm{bin}(cl_1, sh_2) \cup \mathrm{bin}(sh_1, cl_2), \mathrm{bin}(sh_1, sh_2)\bigr)\Bigr) \\
&\quad = \mathcal{I}\bigl(\mathrm{bin}^{\mathsf{W}}(shw_1, shw_2)\bigr).
\end{aligned}$$

Proof of (9.5): if $cl = \varnothing$ then the two expression are easily seen to be equal to $sh^\star$. Otherwise, if $cl \neq \varnothing$, by applying case (9.11) of Lemma 9.9, we obtain

$$\begin{aligned}
\bigl(\mathcal{I}(shw)\bigr)^\star &= (\downarrow cl \cup sh)^\star \\
&\subseteq \bigl(\downarrow(cl \cup sh)\bigr)^\star \\
&= \downarrow\bigl((cl \cup sh)^\star\bigr) \\
&= \downarrow\Bigl(\bigl\{\mathrm{vars}(cl) \cup \mathrm{vars}(sh)\bigr\}\Bigr) \\
&= \mathcal{I}(shw^\star).
\end{aligned}$$

□

9.4 Widening Set-Sharing

The partial order $\preceq_W$ can be interpreted as a way to compare the precision *potential* of descriptions. To clarify the meaning of such a sentence suppose that, for $i = 1, 2$, $shw_i = (cl_i, sh_i) \in SH^{\mathsf{W}}$ are such that $shw_1 \prec_W shw_2$ (namely, $shw_1 \preceq_W shw_2$, but $shw_2 \not\preceq_W shw_1$). According to the definition of $\preceq_W$, we have two cases:

1. if $\mathcal{I}(shw_1) \subset \mathcal{I}(shw_2)$ then shw_1 is more precise than shw_2;
2. otherwise, we have $\mathcal{I}(shw_1) = \mathcal{I}(shw_2)$ but $\downarrow(cl_1) \subset \downarrow(cl_2)$. In this case, while *now* being as precise as shw_1, the element shw_2 will probably lose more precision as the analysis goes on, because it has more cliques.

Definition 9.10 (Widening for SH^{W}.) *The function $\nabla \colon SH^{\mathsf{W}} \to SH^{\mathsf{W}}$ is a* widening *for SH^{W} if, for each $shw \in SH^{\mathsf{W}}$, we have $shw \preceq_W \nabla\, shw$.*

By definition, for any $shw \in SH^{\mathsf{W}}$, $\mathcal{I}(shw) \subseteq \mathcal{I}(\nabla\, shw)$ holds. The obvious corollary is that any analysis using these widenings, possibly a different widening at each step of the analysis, is correct. Also note that it is safe to widen and normalize descriptions within the actual computation of the whole abstract unification operator, before and/or after the execution of the operators of Definition 9.7.

Our widenings, being unary operators, do not depend on the iterates seen so far during the fixpoint computation; moreover, even the identity function on SH^{W} is a widening. Thus, Definition 9.10 looks quite different from the standard definition of widening operators, as introduced in [CC79]. Nonetheless, it is sufficient for our purposes, because SH^{W} is a finite domain and all of its ascending chains finitely converge.

Of course, really useful widenings are *guarded* by some applicability conditions. The simplest conditions are those based on the cardinality of the sets in the SH^{W} description. For example, for each widening ∇ one can define

$$\nabla_{f,n}(cl, sh) \stackrel{\mathrm{def}}{=} \begin{cases} \nabla(cl, sh), & \text{if } f(\#\, cl, \#\, sh) > n, \\ (cl, sh), & \text{otherwise,} \end{cases}$$

for suitable choices of $f \colon \mathbb{N}^2 \to \mathbb{N}$ and $n \in \mathbb{N}$.

Note that the size of the abstract domain elements is just one of the many possibilities for dynamically adjusting the precision/efficiency trade-off of the analysis. For instance, by adding a time parameter to the definition of widening, one could consider the elapsed analysis time, so that cruder approximations come into play when this quantity goes beyond a given threshold. In CHINA, a few widenings of this kind have been implemented but, in the current development phase, they are all disabled: being *time-dependent*, they make the precision results of the analysis non-deterministic, so that debugging becomes almost impossible.

Widenings can be ordered in the obvious way: $\nabla_1 \preceq_W \nabla_2$ if, for all $shw \in SH^{\mathsf{W}}$, $\nabla_1(shw) \preceq_W \nabla_2(shw)$. Near the top of this ordering, we have two *panic widenings*. These are defined by

$$\nabla^{p}(cl, sh) \stackrel{\mathrm{def}}{=} \Bigl(cl \cup \bigl\{\mathrm{vars}(sh)\bigr\}, \varnothing\Bigr),$$
$$\nabla^{P}(cl, sh) \stackrel{\mathrm{def}}{=} \Bigl(\bigl\{\mathrm{vars}(cl) \cup \mathrm{vars}(sh)\bigr\}, \varnothing\Bigr).$$

The panic widenings are present in the CHINA implementation, with very strict guards, only to ensure that the analyzer will never crash, however big (or involved) the analysed logic program is.

At the other extreme we can define very soft widenings. While referring the interested reader to [ZBH99b] where several widenings with different properties are specified and experimentally evaluated, we now introduce the widening based on Fecht's idea, which we will call *Fecht's widening.* This is simply given by

$$\nabla^{F}(cl, sh) \stackrel{\mathrm{def}}{=} (cl \cup sh, \varnothing).$$

Note that this widening does not introduce new sharing pairs. However, as it can introduce new singletons, it may destroy ground dependencies. Thus, as was the case for the domain $\downarrow SH$, better precision is obtained when the sharing domain using this widening is combined with Pos.

9.5 Experimental Evaluation

For the experimental evaluation of the Fecht's widening ∇^{F}, we consider the base domain $Pos \times SFL_2$, introduced in Section 8.2. The widening is guarded by a size threshold of 100 on the sharing set component. In other words, immediately before each abstract mgu operation the analyzer operates both normalization (to remove those cliques and sharing groups that are redundant with respect to the downward closure of the clique set) as well as redundancy elimination (to remove those sharing groups that are redundant with respect to ρ_{PSD}). If after this phase the operand (cl, sh) is such that $\# sh > 100$, then (cl, sh) is substituted by $\nabla^{F}(cl, sh)$. We call this guarded widening ∇^{F}_{100}.

We consider goal dependent and goal independent analyses, both with and without the structural information provided by the Pattern($\cdot$) construct. The reader is warned that the experimental results obtained for the widened domain *with* structural information are to be considered biased, because there is no widening at all on the Pattern component. Since all the time-dependent widenings are disabled, when using structural information the analysis time can still increase beyond the given time threshold. This is the only reason why we still obtain one time-out, for the goal dependent analysis (with structural information) of the program `symbolic1`.

In Tables 9.1 and 9.2 we provide the same kind of precision and efficiency summaries used in Chapter 7. By only looking at the average analysis times reported in Table 9.1, we see immediate evidence of the efficiency improvement achieved: once again, the ratios are under one tenth. Moreover, we have to stress that this has been obtained with respect

Goal Independent		w/o Struct Info		with Struct Info	
Programs	Measure	Base	∇^F_{100}	Base	∇^F_{100}
All Benchs	Sum	19733.80	94.99	45912.30	867.07
	Avg	53.05	0.26	123.42	2.33
	StDev	267.12	1.19	452.67	34.44
	Median	0.10	0.03	0.11	0.04
Both Completed	Sum	8933.80	74.21	6312.27	100.99
	Avg	24.41	0.20	18.04	0.29
	StDev	139.31	1.02	118.88	1.57
	Median	0.10	0.03	0.10	0.03

Goal Dependent		w/o Struct Info		with Struct Info	
Programs	Measure	Base	∇^F_{100}	Base	∇^F_{100}
All Benchs	Sum	19531.40	239.37	47985.40	4322.37
	Avg	76.29	0.94	187.44	16.88
	StDev	348.58	6.81	566.95	132.70
	Median	0.10	0.03	0.12	0.05
Both Completed	Sum	3331.36	55.95	2985.39	58.79
	Avg	13.49	0.23	12.92	0.25
	StDev	88.78	0.75	74.66	0.92
	Median	0.10	0.03	0.11	0.04

Table 9.1: $(\mathit{Pos} \times \mathit{SFL}_2)$ with ∇^F_{100}: time sum, average, deviation and median.

to the domain $\mathit{Pos} \times \mathit{SFL}_2$, which had already improved the analysis times by a similar ratio thanks to the use of the non-redundant set-sharing domain PSD.

When looking at the precision of the analysis, Table 9.2 provides additional good news: precision losses are present in a limited number of cases. In Table 9.3 we provide a detailed view of all of them, where rows are labeled by the name of the benchmark and columns are labeled by the measured quantity (I, G, F and L, interpreted as usual). The meaning of table cells, which are of the form n/m, is that the analysis using $(\mathit{Pos} \times \mathit{SFL}_2)$ detected n units (variables or pairs of variables) for the given observable, while m units were detected when using the same domain with ∇^F_{100}. For the reader's convenience, the differences are highlighted by using a boldface font.

Finally, it is worth stressing that there are other differences besides those reported in Table 9.3: namely, for the domain $(\mathit{Pos} \times \mathit{SFL}_2)$ without widenings we had 62 time-outs (spread over the four variations of the analysis). All of these analyses, but the single one mentioned before, now complete in the given time limit, therefore providing useful results which can be fairly regarded as precision improvements.

9.6 Summary

We have studied a new representation for set-sharing that allows for the incorporation of a variety of widening operators. Our experimental evaluation shows that, by basing one

Goal Independent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
same precision	97.6	98.1	98.4	98.4	92.2	94.1	93.5	92.5
unknown	1.6	1.6	1.6	1.6	5.9	5.9	5.9	5.9
$-2 \leq p < 0$	0.5	0.3	—	—	0.8	—	0.3	0.8
$-5 \leq p < -2$	—	—	—	—	0.5	—	—	—
$-10 \leq p < -5$	0.3	—	—	—	0.3	—	—	0.5
$-20 \leq p < -10$	—	—	—	—	—	—	—	0.3
$p < -20$	—	—	—	—	0.3	—	0.3	—

Goal Dependent	without Struct Info				with Struct Info			
Prec. class	I	G	F	L	I	G	F	L
same precision	95.3	96.5	96.5	96.5	89.1	90.2	90.2	89.8
unknown	3.5	3.5	3.5	3.5	9.8	9.8	9.8	9.8
$-2 \leq p < 0$	1.2	—	—	—	0.8	—	—	—
$-20 \leq p < -10$	—	—	—	—	0.4	—	—	0.4

Time diff. class	Goal Ind.		Goal Dep.	
	w/o SI	with SI	w/o SI	with SI
both timed out	—	—	—	0.4
degradation > 1	—	—	—	—
$0.5 <$ degradation ≤ 1	—	—	—	—
$0.2 <$ degradation ≤ 0.5	—	—	0.8	—
$0.1 <$ degradation ≤ 0.2	0.3	—	—	—
same time	71.0	70.2	73.4	69.5
$0.1 <$ improvement ≤ 0.2	5.1	4.6	5.9	3.1
$0.2 <$ improvement ≤ 0.5	4.0	2.7	5.1	3.9
$0.5 <$ improvement ≤ 1	2.4	3.8	—	0.8
improvement > 1	17.2	18.8	14.8	22.3

Total time class	Goal Independent				Goal Dependent			
	without SI		with SI		without SI		with SI	
	%1	%2	%1	%2	%1	%2	%1	%2
timed out	1.6	—	5.9	—	3.5	—	9.8	0.4
$t > 10$	9.7	0.5	8.6	1.6	7.4	1.6	7.4	3.9
$5 < t \leq 10$	1.3	0.3	1.9	1.6	2.0	1.2	2.0	1.6
$1 < t \leq 5$	6.5	3.2	4.0	3.5	3.1	4.3	4.7	7.0
$0.5 < t \leq 1$	3.0	4.0	4.6	5.6	3.5	3.9	2.7	3.5
$0.2 < t \leq 0.5$	6.5	7.5	8.9	9.7	9.8	9.0	9.8	12.1
$t \leq 0.2$	71.5	84.4	66.1	78.0	70.7	80.1	63.7	71.5

Table 9.2: $(Pos \times SFL_2)$ with ∇^F_{100}: efficiency and precision summaries.

Benchmark	I	G	F	L
Goal Independent, without Struct Info				
back52	**8752**/8748	838/838	1653/1653	5582/5582
caslog	**7126**/6649	569/569	390/390	2115/2115
reform_compiler	**28357**/28345	**1110**/1108	2311/2311	7751/7751
Goal Independent, with Struct Info				
chat_parser	**1888**/1709	149/149	**166**/127	**604**/565
chess	**512**/508	95/95	**78**/77	**233**/232
cselcomp	**5203**/5177	235/235	668/668	**2013**/2012
puzzle	**169**/114	0/0	55/55	**75**/62
motel	**4921**/4705	181/181	1216/1216	**1985**/1953
petsan	**4024**/4010	298/298	332/332	1099/1099
shopper_shop	**128**/123	22/22	30/30	**128**/118
Goal Dependent, without Struct Info				
ezan	**2011**/2009	453/453	277/277	751/751
quotan	**646**/639	149/149	98/98	286/286
synth	**1994**/1986	547/547	276/276	851/851
Goal Dependent, with Struct Info				
quotan	**709**/701	149/149	106/106	285/285
shopper_shop	**3518**/2932	584/584	42/42	**804**/668
synth	**2678**/2654	643/643	279/279	952/952

Table 9.3: $(Pos \times SFL_2)$ with ∇^F_{100}: detail of the precision losses.

of these widenings on an idea of C. Fecht, we obtain a data-flow analysis for groundness, independence, freeness and linearity, with unprecedented levels of precision *and* efficiency.

We thus believe we have made a significant further step toward the solution of the problem of practical, precise, and efficient sharing analysis of logic programs. By implementing the widenings defined in this chapter (with the addition of a time-dependent widening in the case that the abstract domain is enhanced by structural information), the CHINA analyzer is able to honor one of its most important design goals: never crash (e.g., by exhausting all the available memory), always terminate with a correct result and in reasonable time.

The problem of the graceful degradation of analyses based on set-sharing was already addressed in [Lan90]: there a new representation for sharing sets (using symbolic expressions built upon a *worst-case* constructor *wc*) is intended to abstract all but the definite ground dependency information on given subsets of the relevant variables. To the best of our knowledge, this domain has never been implemented. We believe that such an approach, by requiring the syntactic manipulation of complex symbolic expressions, can still suffer from efficiency problems when dealing with big programs.

Chapter 10

Related Work

It is almost impossible to provide a complete and fair account of all the literature relevant to sharing analysis. As we have seen throughout the thesis, in the broad area of abstract interpretation of logic programs, sharing analysis has been a very active research topic. In the recent past, many researchers contributed to the field by proposing new domains and abstract operators, by formally or experimentally comparing previous proposals or even by using sharing analysis domains as test-cases for the application of domain-independent abstract interpretation concepts and techniques. In many cases, minor variations of the same concepts and results have been proposed independently by different people, so that even the attribution of a particular enhancement in the field is sometimes problematic.

Many of these proposals have been discussed in the previous chapters, even though some of them *en passant* only. In the following, we will review other recent publications on sharing analysis, trying to highlight the connections with our work.

10.1 Alternative Domains for Sharing Analysis

In this section we review the approaches to the sharing analysis of logic programs that are, more or less, alternative with respect to the standard pair-sharing and set-sharing domains presented in Chapter 1. All the following proposals assume a concrete domain of idempotent substitutions, so that the corresponding correctness results apply to the equational theory $\mathcal{FT}$ only. Moreover, they enable the use of *abstract compilation* [CF92, GDL95, HWD92], where the data-flow analysis is compiled directly to a generalized abstract machine, computing the standard semantics of the program on the domain of abstract descriptions.

Bruynooghe et al. [BDB$^+$96] introduce a new domain for sharing and freeness analysis based on the concept of *pre-interpretation* [BBD94]. The elements of the abstract domain are sets of *domain relations*, where each domain relation is a set of assignments of values from the pre-interpretation to the tuple of variables of interest. The considered pre-interpretation has three elements, g, i and f, corresponding to ground, partially instantiated and free terms, respectively. In [BDB$^+$96] it is shown that a single domain relation can encode, besides freeness, a set of sharing groups: therefore, it can be argued that the carrier of this abstract domain has the same expressive power of the disjunctive completion [CC92a] of $SH \times F$; in particular, it subsumes the groundness domain Pos. However, the semantic operators defined in [BDB+96] are responsible, in certain cases, for some precision loss, thus making the analysis based on this domains not comparable with respect to an analysis using $SH \times F$. Moreover, in order to obtain an efficient implementation, the prototype analyzer uses a simplified version of the abstract operators, so that the precision of the analysis is further degraded. As a consequence, the experimental evaluation reported a precision loss on all but the smaller benchmarks considered.

Codish et al. [CLB97, CLB00] describe an algebraic approach to the sharing analysis of logic programs that is based on *set logic programs*. A set logic program is a logic program in which the terms are sets of variables and standard unification is replaced by a suitable unification for sets, called *ACI1-unification* (unification in the presence of an associative, commutative, and idempotent equality theory with a unit element). In [CLB97, CLB00], it is shown that the domain of *set-substitutions*, with a few modifications, can be used as an abstract domain for sharing analysis. The standard operations of composition, application and projection already defined for set-substitutions can be used for the abstract semantics construction. However, the similarities do not carry through to all the abstract operators. In particular, the standard ACI1-unification operator defined on set-substitutions, computing the most general ACI1-unifier with respect to the standard instantiation ordering, is *not* a correct approximation of concrete unification. Thus, the authors define a new preorder on set-substitutions, denoted $\preceq_{ir}$, and show that abstract unification corresponds to finding the most general ACI1-unifier with respect to $\preceq_{ir}$. The abstract domain is defined as the quotient of the domain of set-substitutions with respect to the equivalence relation induced by this preorder. Such a quotient is shown to be isomorphic to the set-sharing domain SH. Note that, as a consequence, as far as groundness and independence are concerned, this domain includes all of the redundant elements identified in Chapter 4.

Levi and Spoto [LS00] study the systematic construction of a new domain for sharing analysis by means of the linear refinement operator [GS97]. In particular, the authors show how a powerful abstract domain can be obtained as the refinement of the simple reduced product $PS \sqcap F$. By encoding the relational dependencies between pair-sharing and freeness, the new domain is able to represent both linearity and, to a limited extent, also structural information. However, the precision potential of this new abstract domain is not fully exploited by the actual analysis: for instance, the given executable specification is only an approximation of the actual abstraction function. Thus, from a formal point of view, this analysis is not uniformly more precise than the one based on the domain SFL. In [AS01], the new abstract domain has been implemented and experimentally evaluated on nine benchmarks. The authors compared their goal independent analysis with respect to a goal dependent analysis using the domain combination $SH \times F$. The comparison has shown that, from a practical point of view, their prototype implementation is almost as precise as $SH \times F$, even though being less efficient.

10.2 Optimal Abstract Unification Operators

Cortesi and Filé [CF99] proved that the abstract unification operator 'amgu' on the set-sharing domain SH, besides being correct, is also optimal in the sense defined by equation (2.3) of Chapter 2. This means that the only way to (uniformly) increase the precision on set-sharing is by considering a domain stronger than SH.

A similar result is obtained independently in [CLB00] for their alternative representation of SH based on set-substitutions. This highlights that the abstract operators defined on the two representations are themselves "isomorphic", meaning that each operator on SH is isomorphic to a particular composition of the operators defined on set-substitutions, and vice versa.

These optimality results only hold for the analysis of logic languages *computing on the domain of finite trees.* Note that what we are saying goes beyond the trivial observation that the operators are only defined for idempotent substitutions. What we really want to stress is that their "natural generalization" to the domain of substitutions in rational solved form (that is, just removing the occurs-check condition $x \notin t$) is correct but not optimal. A formal argument in this sense can be obtained by combining Proposition 6.49, that is the correctness result for the cyclic_x^t operator, with Example 6.35, showing that the cyclic_x^t operator can improve the precision of the analysis. Note that both [CF99] and [CLB00] clearly state that they only consider idempotent substitutions.

In [CLB00] the approach using set logic programs is generalized to include linearity information, by suitably annotating the set-substitutions. The abstract unification operator enhanced to exploit this information, denoted $\mathit{lin}\text{-}\mathit{mgu}_{ACI1}$, is very similar to the classical combinations of set-sharing with linearity [BCM94a]. In particular, the precision improvements arising from this enhancement are only exploited when the two terms being unified are definitely independent. As we have seen in Chapter 6, this choice results in a sub-optimal abstract unification operator. However, in [CLB00], the authors claim the optimality of the abstract operator $\mathit{lin}\text{-}\mathit{mgu}_{ACI1}$. This claim is formally stated, together with a proof, as Lemma A.10 in the Appendix of [CLB00]. Such an optimality result seems to be in direct contraposition with Propositions 6.45 and 6.47, stating that our improved operator is correct, and Example 6.34, showing that there exists a case where our operator provides more precision. By looking at the proof of Lemma A.10, it can be seen that the case when the two terms possibly share a variable is dealt with by simply referring to an example:[1] this one is supposed to show that all the possible sharing groups can be generated. However, even our improved operator correctly characterizes the given example, so that the proof of Lemma A.10 is wrong and the optimality of the abstract operator $\mathit{lin}\text{-}\mathit{mgu}_{ACI1}$ is not proved. In addition, *assuming* that the proofs of our propositions are correct, it follows that Lemma A.10 cannot hold.

[1]The proof refers to Example 8, which however has nothing to do with the possibility that the two terms share; we believe that Example 2 was intended.

10.3 Pair-Sharing over Rational Trees

As we have seen, most of the literature on sharing analysis assumes a domain of finite trees. For the domain of rational trees, prior to [HBZ98, HBZ02] (where the results of Chapter 3 were first presented), none of the many sharing domains proposed were proved correct. To our knowledge, King [Kin00] provided the first proof of correctness for a sharing domain enhanced with linearity information that applies to rational-tree languages.

In [Kin00], the abstract domain considered is the combination of the pair-sharing and linearity components of the Søndergaard's domain ASub. Using our notation, the domain is thus isomorphic to $(\rho_{PS} \sqcap \rho_L)(\mathit{SFL})$. Groundness information, in the form of a set of definitely ground variables (i.e., the domain Con), is used to improve the results of the analysis. However, the groundness domain is a parameter of the domain construction and the choice of the actual domain used to compute the groundness information is left unspecified. In particular, it is *assumed* without proof that the corresponding analysis of groundness is correct for rational-tree languages.

To formalize the abstraction and concretization function, [Kin00] borrows from [IZ96] the concept of limit for a sequence of substitutions. This limit operator is useful when reasoning about possibly infinite rational trees and ensures that our 'rt' operator is well-defined. However, in general, the limit is not finitely computable and the same holds for the abstraction function as specified in [Kin00]. In contrast, the abstraction functions defined in this thesis are based on the finitely computable 'occ', 'fvars' and 'lvars' operators.

Finally, the proofs of correctness in [Kin00] exploit the concept of *alternating paths* [Søn86], therefore having a completely different structure with respect to the corresponding proofs presented here. It can be fairly argued that the proofs in [Kin00] are simpler: one of the reasons is that they apply to a much weaker domain. It would be interesting to know whether or not the alternating paths concept (or a small variation of it) could be exploited to obtain simpler correctness proofs for analyses based on the set-sharing domain.

10.4 Generalized Quotient or Optimal Semantics?

A new domain for pair-sharing analysis has been defined in [Sco00] as

$$\mathsf{Sh}^{\mathsf{PSh}} = \mathit{PSD}^{+} \sqcap A,$$

where the PSD^{+} component is one of the factors obtained from PSD by complementation (see Section 5.5.2 in Chapter 5), while the A component is a strict abstraction of the groundness domain Pos. It can be seen from the definition that $\mathsf{Sh}^{\mathsf{PSh}}$ is a close relative of PSD. This new domain is obtained, just as in the case for PSD, by a construction that starts from the set-sharing domain $\mathit{SH} \equiv \mathsf{Sh}$ and aims at deriving the pair-sharing information encoded by $\mathit{PS} \equiv \mathsf{PSh}$. However, instead of applying the generalized quotient operator used to define PSD, the domain $\mathsf{Sh}^{\mathsf{PSh}}$ is obtained by applying a new domain-theoretic operator that is based on the concept of *optimal semantics* [GRS98a].

When comparing $\mathsf{Sh}^{\mathsf{PSh}}$ and *PSD*, the key point to note is that $\mathsf{Sh}^{\mathsf{PSh}}$ is neither an abstraction nor a concretization of the starting domain *SH*. On the one hand $\mathsf{Sh}^{\mathsf{PSh}}$ is strictly more precise for computing pair-sharing, since it contains formulas of *Pos* that are not in the domain *SH*. On the other hand *SH* and *PSD* are strictly more precise for computing groundness, since $\mathsf{Sh}^{\mathsf{PSh}}$ does not contain all of *Def*: in particular, it does not contain any of the elements in *Con*.

While these differences are correctly stated in [Sco00], the informal discussion goes further. For instance, it is argued in [Sco00, Section 6.1] that

> "in [BHZ02][2] the domain *PSD* is compared to its proper abstractions only, which is a rather restrictive hypothesis..."

This hypothesis was not made arbitrarily: rather, it is a distinctive feature of the generalized quotient approach itself. Moreover, such an observation is not really appropriate because, when devising the *PSD* domain in Chapter 4, the goal was to *simplify* the starting domain *SH* without losing precision on the observable *PS*. This is the goal of the generalized quotient operator and, in such a context, the "rather restrictive hypothesis" is not restrictive at all.

The choice of the generalized quotient can also provide several advantages that have been fully exploited throughout this thesis. Since an implementation for *SH* was already available, the application of this operator resulted in an *executable* specification of the simpler domain *PSD*. By optimizing this executable specification it was possible to arrive at a much more efficient implementation: exponential time and space savings have been achieved by removing the redundant sharing groups from the computed elements and by replacing the star-union operator with the 2-self-union operator. Moreover, the executable specification inherited all the correctness results readily available for that implementation of *SH*, so that the only new result that had to be proved was the correctness of the optimizations.

These advantages do not hold for the domain $\mathsf{Sh}^{\mathsf{PSh}}$. In fact, the definition of a feasible representation for its elements and, *a fortiori*, the definition of an executable specification of the corresponding abstract operators seem to be open issues.[3] Most importantly, the required correctness results cannot be inherited from those of *SH*. All the above reasons indicate that the generalized quotient was a sensible choice when looking for a domain simpler than *SH* while preserving precision on *PS*.

Things are different if the goal is *to improve the precision of a given analysis* with respect to the observable, as was the case in [Sco00]. In this context the generalized quotient is the wrong choice, since by definition it cannot help, whereas the operator defined in [Sco00] could be useful.

[2]As already noted, Chapters 4 and 7 are based on the results presented in [BHZ02].

[3]In [Sco00], the only representation given for the elements of $\mathsf{Sh}^{\mathsf{PSh}}$ is constituted by infinite sets of substitutions.

10.5 Finite-Tree Analysis

Sharing information has been shown to be important for finite-tree analysis [BGHZ01, BZGH01]. This aims at identifying those program variables that, at a particular program point, cannot be bound to an infinite rational tree (in other words, they are necessarily bound to acyclic terms). This novel analysis is irrelevant for those logic languages computing over a domain of finite trees, while having several applications for those (constraint) logic languages that are explicitly designed to compute over a domain including rational trees, such as Prolog II and its successors [Col82, Col90], SICStus Prolog [SIC95], and Oz [ST94].

The analysis specified in [BGHZ01] is based on a parametric abstract domain $H \times P$, where the H component (the Herbrand component) is a set of variables that are known to be bound to finite terms, while the parametric component P can be any domain capturing aliasing, groundness, freeness and linearity information that is useful to compute finite-tree information. An obvious choice for such a parameter is the domain SFL and we have developed two implementations, by instantiating P to both SFL and its non-redundant version SFL_2. It is worth noting that, in [BGHZ01], the correctness of the finite-tree analysis is proved by *assuming* the correctness of the underlying analysis on the parameter P. Thus, thanks to the results proved in Chapter 6, such a proof can now be considered complete.

While referring the reader to [BGHZ01, BZGH01] for a deeper introduction to finite-tree analysis and its applications, we now explain how, for such an application, the combination $H \times \mathit{SFL}$ has better precision than the combination $H \times \mathit{SFL}_2$.

In the computation of the finite-tree component, the information encoded in SFL is exploited in several ways. In particular, the set-sharing component SH is more precise than PSD when computing the following new abstract operator, which is needed for the definition of amgu_H, the abstract unification operator on the H component (note that we are intentionally skipping the definition of all the other abstract operators).

Definition 10.1 *For each $s, t \in \mathit{HTerms}$, $d \stackrel{\mathrm{def}}{=} \langle sh, f, l\rangle \in \mathit{SFL}$, let $sh_s = \mathrm{rel}\bigl(\mathrm{vars}(s), sh\bigr)$ and $sh_t = \mathrm{rel}\bigl(\mathrm{vars}(t), sh\bigr)$. Then*

$$\mathrm{share\_same\_var}_d(s,t) \stackrel{\mathrm{def}}{=} \mathrm{vars}(sh_s \cap sh_t).$$

Informally speaking, $\mathrm{share\_same\_var}_d(s,t)$ returns the set of all the variables that may share *the same variable* with both terms s and t. The next example witnesses that the domain SFL_2 is less precise than the domain SFL when computing with such an operator.

Example 10.2 *Consider the abstract element $d \stackrel{\mathrm{def}}{=} \langle sh, f, l\rangle \in \mathit{SFL}$, where*

$$sh \stackrel{\mathrm{def}}{=} \{xy, xz, yz\}.$$

Then we have

$$\begin{aligned} sh_x &= \operatorname{rel}\bigl(\{x\}, sh\bigr) = \{xy, xz\}, \\ sh_y &= \operatorname{rel}\bigl(\{y\}, sh\bigr) = \{xy, yz\}, \\ sh_x \cap sh_y &= \{xy\}. \end{aligned}$$

Thus, by Definition 10.1, $z \notin \mathrm{share\_same\_var}_d(x, y)$.

In contrast, when computing on the abstract domain SFL_2*, the element* d *is equivalent to* $d' = \rho_{\mathit{PSD}}(d) = \langle sh', f, l\rangle$*, where*

$$sh' \stackrel{\text{def}}{=} \rho_{\mathit{PSD}}(sh) = sh \cup \{xyz\}.$$

As a consequence, we have

$$\begin{aligned} sh'_x &= \operatorname{rel}\bigl(\{x\}, sh'\bigr) = \{xy, xz, xyz\}, \\ sh'_y &= \operatorname{rel}\bigl(\{y\}, sh'\bigr) = \{xy, yz, xyz\}, \\ sh'_x \cap sh'_y &= \{xy, xyz\}, \end{aligned}$$

so that $z \in \mathrm{share\_same\_var}_{d'}(x, y)$.

Thus, in $d \in \mathit{SFL}$ *the information provided by the lack of the sharing group* xyz *is not redundant for the computation of the function* $\mathrm{share\_same\_var}_d$.

Having showed that SFL_2 is less precise than SFL, it is time to question whether all of the information encoded in SFL is useful for finite-tree analysis. It turns out that many of the elements of the domain SFL are redundant even for finite-tree analysis. By looking at Example 10.2 we see that, when deciding if it is the case that $z \in \mathrm{share\_same\_var}_d(s, t)$, we need to consider *three* objects at a time: the variable z as well as the terms s and t. Thus, the domain PSD is not enough, since it precisely captures *pair*-sharing dependencies only. The above observation already contains the solution to our problem: *triple-sharing dependencies.*

Let $\mathit{SFL}_3 \stackrel{\text{def}}{=} \mathit{TSD}_3 \times F \times L$, where $\mathit{TSD}_3 = \rho_{\mathit{TSD}_3}(\mathit{SH})$ is the domain defined in Chapter 5 precisely capturing the dependencies on 3-tuples. We have shown that SFL_3 is complete with respect to SFL for finite-tree analysis. Namely, the finite-tree analyses using the domain $H \times \mathit{SFL}_3$ achieves the same precision results (on the finite-tree component H) of the domain $H \times \mathit{SFL}$.

As we have seen in Chapter 5, on the domain TSD_3 the star-closure operator can be safely replaced by the 3-self-union operator, whose complexity is cubic in the number of sharing groups of the given abstract element. Thus, even in this case the amgu_S operator can be implemented in polynomial time without losing precision on the observables, which now are groundness, independence, freeness, linearity and term-finiteness.

However, it is questionable whether using SFL_3 instead of SFL_2 would improve the precision from a practical point of view. Experimentation has shown that, in practice, finite-tree analysis using SFL_2 always achieves the same precision as that of the corresponding analysis using SFL (to be more precise, the same results are obtained on all but the single synthetic benchmark, named `hvars`, that we wrote in order to show that SFL *can* be more precise than SFL_2).

Chapter 11

Conclusions

In this thesis we have identified and provided a solution to some of the problems related to the specification and implementation of a practical sharing analysis tool for logic languages. The contributions of this work relate to the correctness, the precision and the efficiency of sharing analyses.

The starting point was the set-sharing domain of Jacobs and Langen. We have shown that the previous correctness results, even for this domain, were inadequate, since they assumed a computation domain of finite trees, while almost all the currently implemented logic languages, by omitting the occurs-check in the unification procedure, actually compute on a domain of possibly infinite rational trees. Thus we have generalized the standard definitions of the abstraction function and proved the correctness of the analysis, first for the basic set-sharing domain of Jacobs and Langen and then for its combination with freeness and linearity information.

Regarding the precision of the analysis, we have provided the specification of a new abstract unification operator for the combination of set-sharing with freeness and linearity. This operator is more precise than the classical ones in that it precisely captures the integration of aliasing with linearity as originally proposed, in a pair-sharing context, by Søndergaard. We have provided an implementation of many variations of this domain, most of them based on proposals that appeared in recent literature but unsupported by an adequate experimental evaluation. The comparisons of the precision results obtained for the different domains have provided insight into the practical impact of these proposals.

The issues pertaining to the efficiency of the static analysis have been pursued, as with precision, from both theoretical and practical points of view. Prior to this work, the pair-sharing domain ASub had been rightfully considered a better trade-off between complexity and precision with respect to the set-sharing domain. A first reason was that the time needed for the abstract unification algorithm may be exponential for the set-sharing domain, while enjoying a polynomial bound for ASub. We have solved this problem by identifying the weakest abstraction of the set-sharing domain that maintains the same precision on the properties of interest. For this new domain, we have defined a polynomial abstract unification operator. A second reason for preferring ASub was that the representation of a set-sharing element may require an exponential amount of space, while elements of ASub can be stored in polynomial space. We have also solved this problem by defining a new representation for set-sharing that supports the specification of many widening operators. With these operations, time and space requirements can be dynamically adjusted during the analysis so that all scalability problems are resolved. These theoretical results have been validated by an extensive experimental evaluation: this has shown that we often achieve significant efficiency improvements, while the precision losses due to the widenings are extremely rare.

In our opinion, the above correctness, precision and efficiency results, are all significant although correctness is regarded as the most important one, filling one of the most noticeable gaps between the theory and the practice of sharing analysis for logic languages. There are, though, other problems that need to be addressed if we aim at the development of a static analyzer for real logic languages: one is the handling of all the built-ins in the language; while another is the definition of a modular analysis, where each module of the source program can be analyzed in isolation, possibly by exploiting the previously computed results of the analysis of the other modules. In both cases, the main difficulty is not in the approximation step, where all the well-known techniques can be used; rather, the problem is in agreeing on the definition of the concrete operators corresponding to the considered language constructs. For instance, when dealing with rational-tree languages, a few of the language built-ins are not provided with a well-established concrete semantics, so that implementations can happen to behave incoherently [BGHZ01]. Similarly, each implementation typically adopts its own module system, characterized by small but tedious differences with respect to the others, so that a uniform solution is not available. These issues can be the target of further research.

Bibliography

[AH87] S. Abramsky and C. Hankin, editors. *Abstract Interpretation of Declarative Languages.* Ellis Horwood Ltd, West Sussex, England, 1987.

[AMSS98] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of Boolean functions for dependency analysis. *Science of Computer Programming*, 31(1):3–45, 1998.

[Apt90] K. R. Apt. Introduction to logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 495–574. Elsevier Science B.V., Amsterdam, 1990.

[AS01] G. Amato and F. Spoto. Abstract compilation for sharing analysis. In H. Kuchen and K. Ueda, editors, *Proceedings of the Fifth International Symposium on Functional and Logic Programming*, volume 2024 of *Lecture Notes in Computer Science*, pages 311–325, Tokyo, Japan, 2001. Springer-Verlag, Berlin.

[Bag97a] R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages.* PhD thesis, Dipartimento di Informatica, Università di Pisa, Pisa, Italy, 1997. Printed as Report TD-1/97.

[Bag97b] R. Bagnara. Structural information analysis for CLP languages. In M. Falaschi, M. Navarro, and A. Policriti, editors, *Proceedings of the "1997 Joint Conference on Declarative Programming (APPIA-GULP-PRODE'97)"*, pages 81–92, Grado, Italy, 1997.

[BBD94] D. Boulanger, M. Bruynooghe, and M. Denecker. Abstracting S-semantics using a model-theoretic approach. In M. Hermenegildo and J. Penjam, editors, *Proceedings of the 6th International Symposium on Programming Language Implementation and Logic Programming*, volume 844 of *Lecture Notes in Computer Science*, pages 432–446, Madrid, Spain, 1994. Springer-Verlag, Berlin.

[BC93] M. Bruynooghe and M. Codish. Freeness, sharing, linearity and correctness — All at once. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Static Analysis, Proceedings of the Third International Workshop*, volume 724 of *Lecture Notes in Computer Science*, pages 153–164, Padova, Italy, 1993. Springer-Verlag, Berlin. An extended version is available as Technical Report CW 179, Department of Computer Science, K.U. Leuven, September 1993.

[BCM94a] M. Bruynooghe, M. Codish, and A. Mulkers. Abstract unification for a composite domain deriving sharing and freeness properties of program variables. In F. S. de Boer and M. Gabbrielli, editors, *Verification and Analysis of Logic Languages, Proceedings of the W2 Post-Conference Workshop, International Conference on Logic Programming*, pages 213–230, Santa Margherita Ligure, Italy, 1994.

[BCM94b] M. Bruynooghe, M. Codish, and A. Mulkers. A composite domain for freeness, sharing, and compoundness analysis of logic programs. Technical Report CW 196, Department of Computer Science, K.U. Leuven, Belgium, July 1994.

[BDB+96] M. Bruynooghe, B. Demoen, D. Boulanger, M. Denecker, and A. Mulkers. A freeness and sharing analysis of logic programs based on a pre-interpretation. In Cousot and Schmidt [CS96], pages 128–142.

[BDJ+00] H. Blockeel, B. Demoen, G. Janssens, H. Vandencasteele, and W. Van Laer. Two advanced transformations for improving the efficiency of an ILP system. In J. Cussens and A. Frisch, editors, *Work-in-Progress Reports, Tenth International Conference on Inductive Logic Programming*, pages 43–59, London, UK, 2000.

[BdlBH94] F. Bueno, M. Garcìa de la Banda, and M. Hermenegildo. Effectiveness of global analysis in strict independence-based automatic program parallelization. In M. Bruynooghe, editor, *Logic Programming: Proceedings of the 1994 International Symposium*, MIT Press Series in Logic Programming, pages 253–268, Ithaca, NY, USA, 1994. The MIT Press.

[BGHZ01] R. Bagnara, R. Gori, P. M. Hill, and E. Zaffanella. Finite-tree analysis for constraint logic-based languages. In P. Cousot, editor, *Static Analysis: 8th International Symposium, SAS 2001*, volume 2126 of *Lecture Notes in Computer Science*, pages 165–184, Paris, France, 2001. Springer-Verlag, Berlin.

[BGL93] R. Barbuti, R. Giacobazzi, and G. Levi. A general framework for semantics-based bottom-up abstract interpretation of logic programs. *ACM Transactions on Programming Languages and Systems*, 15(1):133–181, 1993.

[BHZ97] R. Bagnara, P. M. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. In Van Hentenryck [Van97], pages 53–67.

[BHZ00] R. Bagnara, P. M. Hill, and E. Zaffanella. Efficient structural information analysis for real CLP languages. In M. Parigot and A. Voronkov, editors, *Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Artificial Intelligence*, pages 189–206, Réunion Island, France, 2000. Springer-Verlag, Berlin.

[BHZ02] R. Bagnara, P. M. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. *Theoretical Computer Science*, 2002. To appear.

[Bru91] M. Bruynooghe. A practical framework for the abstract interpretations of logic programs. *Journal of Logic Programming*, 10:91–124, 1991.

[BS99] R. Bagnara and P. Schachte. Factorizing equivalent variable pairs in ROBDD-based implementations of *Pos*. In A. M. Haeberer, editor, *Proceedings of the "Seventh International Conference on Algebraic Methodology and Software Technology (AMAST'98)"*, volume 1548 of *Lecture Notes in Computer Science*, pages 471–485, Amazonia, Brazil, 1999. Springer-Verlag, Berlin.

[BZGH01] R. Bagnara, E. Zaffanella, R. Gori, and P. M. Hill. Boolean functions for finite-tree dependencies. Quaderno 252, Dipartimento di Matematica, Università di Parma, 2001. Available at `http://www.cs.unipr.it/~bagnara/`.

[BZH00] R. Bagnara, E. Zaffanella, and P. M. Hill. Enhanced sharing analysis techniques: A comprehensive evaluation. In M. Gabbrielli and F. Pfenning, editors, *Proceedings of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 103–114, Montreal, Canada, 2000. Association for Computing Machinery.

[CC77a] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[CC77b] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In E. J. Neuhold, editor, *IFIP Conference on Formal Description of Programming Concepts*, pages 237–277. North-Holland, 1977.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, 1979.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[CC92b] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and

M. Wirsing, editors, *Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295, Leuven, Belgium, 1992. Springer-Verlag, Berlin.

[CDD85] J.-H. Chang, A. M. Despain, and D. DeGroot. AND-parallelism of logic programs based on a static data dependency analysis. In *Digest of Papers of COMPCON Spring'85*, pages 218–225. IEEE Computer Society Press, 1985.

[CDFB93] M. Codish, D. Dams, G. Filé, and M. Bruynooghe. Freeness analysis for logic programs — and correctness? In D. S. Warren, editor, *Logic Programming: Proceedings of the Tenth International Conference on Logic Programming*, MIT Press Series in Logic Programming, pages 116–131, Budapest, Hungary, 1993. The MIT Press. An extended version is available as Technical Report CW 161, Department of Computer Science, K.U. Leuven, December 1992.

[CDFB96] M. Codish, D. Dams, G. Filé, and M. Bruynooghe. On the design of a correct freeness analysis for logic programs. *Journal of Logic Programming*, 28(3):181–206, 1996.

[CDY91] M. Codish, D. Dams, and E. Yardeni. Derivation and safety of an abstract unification algorithm for groundness and aliasing analysis. In Furukawa [Fur91], pages 79–93.

[CF92] P. Codognet and G. Filé. Computations, abstractions and constraints. In *Proceedings of the Fourth IEEE International Conference on Computer Languages*. IEEE Computer Society Press, 1992.

[CF93] A. Cortesi and G. Filé. Comparison and design of abstract domains for sharing analysis. In D. Saccà, editor, *Proceedings of the "Eighth Italian Conference on Logic Programming (GULP'93)"*, pages 251–265, Gizzeria, Italy, 1993. Mediterranean Press.

[CF99] A. Cortesi and G. Filé. Sharing is optimal. *Journal of Logic Programming*, 38(3):371–386, 1999.

[CFG+95] A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. In A. Mycroft, editor, *Static Analysis: Proceedings of the 2nd International Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 100–117, Glasgow, UK, 1995. Springer-Verlag, Berlin.

[CFG+97] A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. *ACM Transactions on Programming Languages and Systems*, 19(1):7–47, 1997.

[CFW92] A. Cortesi, G. Filé, and W. Winsborough. Comparison of abstract interpretations. In M. Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, volume 623 of *Lecture Notes in Computer Science*, pages 521–532, Wien, Austria, 1992. Springer-Verlag, Berlin.

[CFW94] A. Cortesi, G. Filé, and W. Winsborough. The quotient of an abstract interpretation for comparing static analyses. In M. Alpuente, R. Barbuti, and I. Ramos, editors, *Proceedings of the "1994 Joint Conference on Declarative Programming (GULP-PRODE'94)"*, pages 372–397, Peñíscola, Spain, 1994. An extended version has been published as [CFW98].

[CFW96] A. Cortesi, G. Filé, and W. Winsborough. Optimal groundness analysis using propositional logic. *Journal of Logic Programming*, 27(2):137–167, 1996.

[CFW98] A. Cortesi, G. Filé, and W. Winsborough. The quotient of an abstract interpretation for comparing static analyses. *Theoretical Computer Science*, 202(1&2):163–192, 1998.

[CH94] D. Cabeza and M. Hermenegildo. Extracting non-strict independent and-parallelism using sharing and freeness information. In B. Le Charlier, editor, *Static Analysis: Proceedings of the 1st International Symposium*, volume 864 of *Lecture Notes in Computer Science*, pages 297–313, Namur, Belgium, 1994. Springer-Verlag, Berlin.

[CKS96] L. Crnogorac, A. D. Kelly, and H. Søndergaard. A comparison of three occur-check analysers. In Cousot and Schmidt [CS96], pages 159–173.

[Cla78] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322, Toulouse, France, 1978. Plenum Press.

[CLB97] M. Codish, V. Lagoon, and F. Bueno. An algebraic approach to sharing analysis of logic programs. In Van Hentenryck [Van97], pages 68–82.

[CLB00] M. Codish, V. Lagoon, and F. Bueno. An algebraic approach to sharing analysis of logic programs. *Journal of Logic Programming*, 42(2):111–149, 2000.

[CLV94] A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Combinations of abstract domains for logic programming. In *Conference Record of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 227–239, Portland, Oregon, 1994.

[CLV00] A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Combinations of abstract domains for logic programming: Open product and generic pattern construction. *Science of Computer Programming*, 38(1–3):27–71, 2000.

[CMB+93] M. Codish, A. Mulkers, M. Bruynooghe, M. Garcìa de la Banda, and M. Hermenegildo. Improving abstract interpretations by combining domains. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 194–205, Copenhagen, Denmark, 1993. ACM Press. Also available as Technical Report CW 162, Department of Computer Science, K.U. Leuven, December 1992.

[CMB+95] M. Codish, A. Mulkers, M. Bruynooghe, M. Garcìa de la Banda, and M. Hermenegildo. Improving abstract interpretations by combining domains. *ACM Transactions on Programming Languages and Systems*, 17(1):28–44, January 1995.

[Col82] A. Colmerauer. Prolog and infinite trees. In K. L. Clark and S. Å. Tärnlund, editors, *Logic Programming, APIC Studies in Data Processing*, volume 16, pages 231–251. Academic Press, New York, 1982.

[Col84] A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'84)*, pages 85–99, Tokyo, Japan, 1984. ICOT.

[Col90] A. Colmerauer. An introduction to Prolog-III. *Communications of the ACM*, 33(7):69–90, 1990.

[CS96] R. Cousot and D. A. Schmidt, editors. *Static Analysis: Proceedings of the 3rd International Symposium*, volume 1145 of *Lecture Notes in Computer Science*, Aachen, Germany, 1996. Springer-Verlag, Berlin.

[CS98] M. Codish and H. Søndergaard. The Boolean logic of set sharing analysis. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *Principles of Declarative Programming*, volume 1490 of *Lecture Notes in Computer Science*, pages 89–100, Pisa, Italy, 1998. Springer-Verlag, Berlin.

[CSS99] M. Codish, H. Søndergaard, and P. J. Stuckey. Sharing and groundness dependencies in logic programs. *ACM Transactions on Programming Languages and Systems*, 21(5):948–976, 1999.

[DR94] S. K. Debray and R. Ramakrishnan. Abstract interpretation of logic programs using magic transformations. *Journal of Logic Programming*, 18(2):149–176, 1994.

[Fec96a] C. Fecht. An efficient and precise sharing domain for logic programs. In H. Kuchen and S. D. Swierstra, editors, *Programming Languages: Implementations, Logics and Programs, Proceedings of the Eighth International Symposium*, volume 1140 of *Lecture Notes in Computer Science*, pages 469–470, Aachen, Germany, 1996. Springer-Verlag, Berlin. Poster.

[Fec96b] C. Fecht. Efficient and precise sharing domains for logic programs. Technical Report A/04/96, Universität des Saarlandes, Fachbereich 14 Informatik, Saarbrücken, Germany, 1996.

[Fil94] G. Filé. Share $\times$ Free: Simple and correct. Technical Report 15, Dipartimento di Matematica, Università di Padova, December 1994.

[FLMP89] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative modeling of the operational behavior of logic languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[FLMP93] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A model-theoretic reconstruction of the operational semantics of logic programs. *Information and Computation*, 103(1):86–113, 1993.

[FR96] G. Filé and F. Ranzato. Complementation of abstract domains made easy. In M. Maher, editor, *Logic Programming: Proceedings of the Joint International Conference and Symposium on Logic Programming*, MIT Press Series in Logic Programming, pages 348–362, Bonn, Germany, 1996. The MIT Press.

[Fur91] K. Furukawa, editor. *Logic Programming: Proceedings of the Eighth International Conference on Logic Programming*, MIT Press Series in Logic Programming, Paris, France, 1991. The MIT Press.

[GDL95] R. Giacobazzi, S. K. Debray, and G. Levi. Generalized semantics and abstract interpretation for constraint logic programs. *Journal of Logic Programming*, 25(3):191–247, 1995.

[GHK+80] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer-Verlag, Berlin, 1980.

[GR96] R. Giacobazzi and F. Ranzato. Compositional optimization of disjunctive abstract interpretations. In H. R. Nielson, editor, *Proceedings of the 1996 European Symposium on Programming*, volume 1058 of *Lecture Notes in Computer Science*, pages 141–155. Springer-Verlag, Berlin, 1996.

[GR97] R. Giacobazzi and F. Ranzato. Completeness in abstract interpretation: a domain perspective. In M. Johnson, editor, *Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology (AMAST'97)*, volume 1349 of *Lecture Notes in Computer Science*, pages 231–245, Sydney, Australia, 1997. Springer-Verlag, Berlin.

[GRS98a] R. Giacobazzi, F. Ranzato, and F. Scozzari. Building complete abstract interpretations in a linear logic-based setting. In Levi [Lev98], pages 215–229.

[GRS98b] R. Giacobazzi, F. Ranzato, and F. Scozzari. Complete abstract interpretations made constructive. In J. Gruska and J. Zlatuska, editors, *Proceedings of 23rd*

International Symposium on Mathematical Foundations of Computer Science (MFCS'98), volume 1450 of *Lecture Notes in Computer Science*, pages 366–377. Springer-Verlag, Berlin, 1998.

[GS97] R. Giacobazzi and F. Scozzari. Intuitionistic implication in abstract interpretation. In H. Glaser, P. Hartel, and H. Kuchen, editors, *Programming Languages: Implementations, Logics, and Programs*, volume 1292 of *Lecture Notes in Computer Science*, pages 175–189, Southampton, U.K., 1997. Springer-Verlag, Berlin.

[HBZ98] P. M. Hill, R. Bagnara, and E. Zaffanella. The correctness of set-sharing. In Levi [Lev98], pages 99–114.

[HBZ02] P. M. Hill, R. Bagnara, and E. Zaffanella. Soundness, idempotence and commutativity of set-sharing. *Theory and Practice of Logic Programming*, 2(2):155–201, 2002. To appear. Available at `http://arXiv.org/abs/cs.PL/0102030`.

[HG90] M. Hermenegildo and K. J. Greene. &-Prolog and its performance: Exploiting independent And-Parallelism. In D. H. D. Warren and P. Szeredi, editors, *Logic Programming: Proceedings of the Seventh International Conference on Logic Programming*, MIT Press Series in Logic Programming, pages 253–268, Jerusalem, Israel, 1990. The MIT Press.

[HR95] M. Hermenegildo and F. Rossi. Strict and non-strict independent and-parallelism in logic programs: Correctness, efficiency, and compile-time conditions. *Journal of Logic Programming*, 22(1):1–45, 1995.

[HW92] W. Hans and S. Winkler. Aliasing and groundness analysis of logic programs through abstract interpretation and its safety. Technical Report 92–27, Technical University of Aachen (RWTH Aachen), 1992.

[HWD92] M. Hermenegildo, R. Warren, and S. K. Debray. Global flow analysis as a practical compilation tool. *Journal of Logic Programming*, 13(4):349–366, 1992.

[ISO95] ISO/IEC. *ISO/IEC 13211-1: 1995 Information technology — Programming languages — Prolog — Part 1: General core.* International Standard Organization, 1995.

[IZ96] B. Intrigila and M. Venturini Zilli. A remark on infinite matching vs infinite unification. *Journal of Symbolic Computation*, 21(3):2289–2292, 1996.

[JL89] D. Jacobs and A. Langen. Accurate and efficient approximation of variable aliasing in logic programs. In E. L. Lusk and R. A. Overbeek, editors, *Logic Programming: Proceedings of the North American Conference*, MIT Press Series in Logic Programming, pages 154–165, Cleveland, Ohio, USA, 1989. The MIT Press.

[JL92] D. Jacobs and A. Langen. Static analysis of logic programs for independent AND parallelism. *Journal of Logic Programming*, 13(2&3):291–314, 1992.

[JLM87] J. Jaffar, J-L. Lassez, and M. J. Maher. Prolog-II as an instance of the logic programming scheme. In M. Wirsing, editor, *Formal Descriptions of Programming Concepts III*, pages 275–299. North-Holland, 1987.

[JM94] J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19&20:503–582, 1994.

[JS87] N. D. Jones and H. Søndergaard. A semantics-based framework for the abstract interpretation of Prolog. In Abramsky and Hankin [AH87], chapter 6, pages 123–142.

[Kei94] T. Keisu. *Tree Constraints.* PhD thesis, The Royal Institute of Technology, Stockholm, Sweden, May 1994. Also available in the SICS Dissertation Series: SICS/D–16–SE.

[Kin93] A. King. A new twist on linearity. Technical Report CSTR 93-13, Department of Electronics and Computer Science, Southampton University, Southampton, UK, 1993.

[Kin94] A. King. A synergistic analysis for sharing and groundness which traces linearity. In D. Sannella, editor, *Proceedings of the Fifth European Symposium on Programming*, volume 788 of *Lecture Notes in Computer Science*, pages 363–378, Edinburgh, UK, 1994. Springer-Verlag, Berlin.

[Kin00] A. King. Pair-sharing over rational trees. *Journal of Logic Programming*, 46(1–2):139–155, 2000.

[KS94] A. King and P. Soper. Depth-k sharing and freeness. In P. Van Hentenryck, editor, *Logic Programming: Proceedings of the Eleventh International Conference on Logic Programming*, MIT Press Series in Logic Programming, pages 553–568, Santa Margherita Ligure, Italy, 1994. The MIT Press.

[Lan90] A. Langen. *Advanced Techniques for Approximating Variable Aliasing in Logic Programs.* PhD thesis, Computer Science Department, University of Southern California, 1990. Printed as Report TR 91-05.

[Lev98] G. Levi, editor. *Static Analysis: Proceedings of the 5th International Symposium*, volume 1503 of *Lecture Notes in Computer Science*, Pisa, Italy, 1998. Springer-Verlag, Berlin.

[Llo87] L. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, second edition, 1987.

[LMM88] J.-L. Lassez, M. J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988.

[LS00] G. Levi and F. Spoto. Non pair-sharing and freeness analysis through linear refinement. In *Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation*, pages 52–61, Boston, MA, USA, 2000. ACM Press.

[Mah88] M. J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings, Third Annual Symposium on Logic in Computer Science*, pages 348–357, Edinburgh, Scotland, 1988. IEEE Computer Society.

[Mel87] C. S. Mellish. Abstract interpretation of Prolog programs. In Abramsky and Hankin [AH87], chapter 8, pages 181–198.

[MH91] K. Muthukumar and M. Hermenegildo. Combined determination of sharing and freeness of program variables through abstract interpretation. In Furukawa [Fur91], pages 49–63. An extended version appeared in [MH92].

[MH92] K. Muthukumar and M. Hermenegildo. Compile-time derivation of variable dependency using abstract interpretation. *Journal of Logic Programming*, 13(2&3):315–347, 1992.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[MS90] K. Marriott and H. Søndergaard. Analysis of constraint logic programs. In S. K. Debray and M. Hermenegildo, editors, *Logic Programming: Proceedings of the North American Conference*, MIT Press Series in Logic Programming, pages 531–547, Austin, Texas, USA, 1990. The MIT Press.

[MSJ94] K. Marriott, H. Søndergaard, and N. D. Jones. Denotational abstract interpretation of logic programs. *ACM Transactions on Programming Languages and Systems*, 16(3):607–648, 1994.

[MSJB94] A. Mulkers, W. Simoens, G. Janssens, and M. Bruynooghe. On the practicality of abstract equation systems. Report CW 198, Department of Computer Science, K. U. Leuven, Leuven, Belgium, 1994.

[MSJB95] A. Mulkers, W. Simoens, G. Janssens, and M. Bruynooghe. On the practicality of abstract equation systems. In L. Sterling, editor, *Logic Programming: Proceedings of the Twelfth International Conference on Logic Programming*, MIT Press Series in Logic Programming, pages 781–795, Kanagawa, Japan, 1995. The MIT Press.

[Pla84] D. A. Plaisted. The occur-check problem in Prolog. *New Generation Computing*, 2(4):309–322, 1984.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Sco00] F. Scozzari. Abstract domains for sharing analysis by optimal semantics. In J. Palsberg, editor, *Static Analysis: 7th International Symposium, SAS 2000*, volume 1824 of *Lecture Notes in Computer Science*, pages 397–412, Santa Barbara, CA, USA, 2000. Springer-Verlag, Berlin.

[Sco02] F. Scozzari. Logical optimality of groundness analysis. *Theoretical Computer Science*, 2002. To appear.

[SIC95] Swedish Institute of Computer Science, Programming Systems Group. *SICStus Prolog User's Manual*, release 3 #0 edition, 1995.

[Søn86] H. Søndergaard. An application of abstract interpretation of logic programs: Occur check reduction. In B. Robinet and R. Wilhelm, editors, *Proceedings of the 1986 European Symposium on Programming*, volume 213 of *Lecture Notes in Computer Science*, pages 327–338. Springer-Verlag, Berlin, 1986.

[ST94] Gert Smolka and Ralf Treinen. Records for logic programming. *Journal of Logic Programming*, 18(3):229–258, 1994.

[Tay91] A. Taylor. *High Performance Prolog Implementation.* PhD thesis, Basser Department of Computer Science, University of Sydney, Sydney, Australia, June 1991.

[Van97] P. Van Hentenryck, editor. *Static Analysis: Proceedings of the 4th International Symposium*, volume 1302 of *Lecture Notes in Computer Science*, Paris, France, 1997. Springer-Verlag, Berlin.

[VD92] P. VanRoy and A. M. Despain. High-performance logic programming with the Aquarius Prolog compiler. *IEEE Computer*, 25(1):54, 1992.

[War42] M. Ward. The closure operators of a lattice. *Ann. Math.*, 43(2):191–196, 1942.

[ZBH99a] E. Zaffanella, R. Bagnara, and P. M. Hill. Widening set-sharing. Quaderno 188, Dipartimento di Matematica, Università di Parma, 1999.

[ZBH99b] E. Zaffanella, R. Bagnara, and P. M. Hill. Widening Sharing. In G. Nadathur, editor, *Principles and Practice of Declarative Programming*, volume 1702 of *Lecture Notes in Computer Science*, pages 414–431, Paris, France, 1999. Springer-Verlag, Berlin.

[ZHB99] E. Zaffanella, P. M. Hill, and R. Bagnara. Decomposing non-redundant sharing by complementation. In A. Cortesi and G. Filé, editors, *Static Analysis:*

Proceedings of the 6th International Symposium, volume 1694 of *Lecture Notes in Computer Science*, pages 69–84, Venice, Italy, 1999. Springer-Verlag, Berlin.

[ZHB02] E. Zaffanella, P. M. Hill, and R. Bagnara. Decomposing non-redundant sharing by complementation. *Theory and Practice of Logic Programming*, 2(2):233–261, 2002. To appear. Available at `http://arXiv.org/abs/cs.PL/0101025`.