Boolean Functions for Finite-Tree Dependencies*

Roberto Bagnara
Enea Zaffanella
Dept. of Mathematics
University of Parma, Italy
bagnara@cs.unipr.it
zaffanella@cs.unipr.it

Roberta Gori
Dept. of Computer Science
University of Pisa, Italy
gori@di.unipi.it

Patricia M. Hill
School of Computing
University of Leeds, U.K.
hill@comp.leeds.ac.uk

ABSTRACT

Several logic-based languages, such as Prolog II and its successors, SICStus Prolog and Oz, offer a computation domain including *rational trees*. Infinite rational trees allow for increased expressivity (cyclic terms can provide efficient representations of grammars and other useful objects) and for faster unification (due to the safe omission of the occurs-check). Unfortunately, the use of infinite rational trees has problems. For instance, many of the built-in and library predicates are ill-defined for such trees and need to be supplemented by run-time checks whose cost may be significant. In a companion paper [3] we have proposed a data-flow analysis aimed at the knowledge of those program variables (the *finite variables*) that will always be bound to finite terms. The analysis domain introduced in [3] correctly captures the creation and propagation of cyclic terms, but is not capable of propagating the guarantees of finiteness that come from built-in predicates and program annotations. Here we present a domain of Boolean functions that precisely captures how the finiteness of some variables influences the finiteness of other variables. This domain of *finite-tree dependencies* provides relational information that is important for the precision of the overall finiteness analysis. It also combines obvious similarities, interesting differences and somewhat unexpected connections with classical domains for *groundness dependencies*.

1. INTRODUCTION

The intended computation domain of most (concurrent and/or constraint and/or functional) logic-based languages includes the algebra of *finite trees*. Other logic-based languages, such as Prolog II and its successors [10, 12], SICStus Prolog [40], and Oz [38], refer to a computation domain of *rational trees*. A rational tree is a possibly infinite tree with a finite number of distinct subtrees and, as is the case for finite trees, where each node has a finite number of immediate descendants. These properties will ensure that rational trees, even though infinite in the sense that they admit paths of infinite length, can be finitely represented. One possible representation makes use of connected, rooted, directed and possibly cyclic graphs where nodes are labeled with variable and function symbols as is the case of finite trees.

Applications of rational trees in logic programming include graphics [21], parser generation and grammar manipulation [10, 24], and computing with finite-state automata [10]. Other applications are described in [23] and [25]. Going from Prolog to CLP, [35] combines constraints on rational trees and record structures, while the logic-based language *Oz* allows constraints over rational and feature trees [38]. The expressive power of rational trees is put to use, for instance, in several areas of natural language processing. Rational trees are used in implementations of the HPSG formalism (Head-driven Phrase Structure Grammar) [36], in the ALE system (Attribute Logic Engine) [8], and in the ProFIT system (Prolog with Features, Inheritance and Templates) [22].

While rational trees allow for increased expressivity, they also come equipped with a surprising number of problems. As we will see, some of these problems are so serious that rational trees must be used in a very controlled way, disallowing them in any context where they are "dangerous". This, in turn, causes a secondary problem: in order to disallow rational trees in selected contexts one must first detect them, an operation that may be expensive.

The first thing to be aware of is that almost any semantics-based program manipulation technique developed in the field of logic programming —whether it be an analysis, a transformation, or an optimization— assumes a computation domain of *finite trees*. Some of these techniques might work with the rational trees but their correctness has only been proved in the case of finite trees. Others are clearly inapplicable. Let us consider a very simple Prolog program:

```
list([]).
list([_|T]) :- list(T).
```

Most automatic and semi-automatic tools for proving program termination and for complexity analysis agree on the fact that `list/1` will terminate when invoked with a ground argument. Consider now the query

```
?- X = [a|X], list(X).
```

*The work of the first two authors has been partly supported by MURST project "Certificazione automatica di programmi mediante interpretazione astratta". The work of the second and fourth authors has been partly supported by EPSRC under grant M05645.

Quaderno 252, Dipartimento di Matematica, Università di Parma, Italy.

and note that, after the execution of the first rational unification, the variable `X` will be bound to a rational term containing no variables, i.e., the predicate `list/1` will be invoked with `X` ground. However, if such a query is given to, say, SICStus Prolog, then the only way to get the prompt back is by pressing `^C`. The problem stems from the fact that the analysis techniques employed by these tools are only sound for finite trees: as soon as they are applied to a system where the creation of cyclic terms is possible, their results are inapplicable. The situation can be improved by combining these termination and/or complexity analyses by a finiteness analysis providing the precondition for the applicability of the other techniques.

The implementation of built-in predicates is another problematic issue. Indeed, it is widely acknowledged that, for the implementation of a system that provides real support for the rational trees, the biggest effort concerns proper handling of built-ins. Of course, the meaning of 'proper' depends on the actual built-in. Built-ins such as `copy_term/2` and `==/2` maintain a clear semantics when passing from finite to rational trees. For others, like `sort/2`, the extension can be questionable:[1] both raising an exception and answering `Y = [a]` can be argued to be "the right reaction" to the query

```
?- X = [a|X], sort(X, Y).
```

Other built-ins do not tolerate infinite trees in some argument positions. A good implementation should check for finiteness of the corresponding arguments and make sure "the right thing" (i.e., failing or raising an appropriate exception) always happens. However, such behavior appears to be uncommon. A small experiment we conducted on six Prolog implementations with queries like

```
?- X = 1+X, Y is X.
?- X = [97|X], name(Y, X).
?- X = [X|X], Y =..  [f|X].
```

resulted in infinite loops, memory exhaustion and/or system thrashing, segmentation faults or other fatal errors. One of the implementations tested, SICStus Prolog, is a professional one and implements run-time checks to avoid most cases where built-ins can have catastrophic effects.[2] The remaining systems are a bit more than research prototypes, but will clearly have to do the same if they evolve to the stage of production tools. Again, a data-flow analysis aimed at the detection of those variables that are definitely bound to finite terms would allow to avoid a (possibly significant) fraction of the useless run-time checks. Note that what has been said for built-in predicates applies to libraries as well. Even though it may be argued that it is enough for programmers to know that they should not use a particular library predicate with infinite terms, it is clear that the use of a "safe" library, including automatic checks which ensure that such predicates are never called with an illegal argument, will result in more robust systems. With the appropriate data-flow analyses, safe libraries do not have to be inefficient libraries.

Another serious problem is the following: the ISO Prolog standard term ordering cannot be extended to rational trees [M. Carlsson, Personal communication, October 2000]. Consider the rational trees defined by `A = f(B, a)` and `B = f(A, b)`. Clearly, `A == B` does not hold. Since the standard term ordering is total, we must have either `A @< B` or `B @< A`. Assume `A @< B`. Then `f(A, b) @< f(B, a)`, since the ordering of terms having the same principal functor is inherited by the ordering of subterms considered in a left-to-right fashion. Thus `B @< A` must hold, which is a contradiction. A dual contradiction is obtained by assuming `B @< A`. As a consequence, applying one of the Prolog term-ordering predicates to one or two infinite terms may cause inconsistent results, giving rise to bugs that are exceptionally difficult to diagnose. For this reason, any system that extends ISO Prolog with rational trees ought to detect such situations and make sure they are not ignored (e.g., by throwing an exception or aborting execution with a meaningful message). However, predicates such as the term-ordering ones are likely to be called a significant number of times, since they are often used to maintain structures implementing ordered collections of terms. This is another instance of the efficiency issue mentioned above.

1.1 Detecting the Creation of Infinite Terms

In [3] we have introduced a composite abstract domain for finite-tree analysis, denoted by $H \times P$. The H domain, written with the initial of *Herbrand* and called the *finiteness* component, is the direct representation of the property of interest: a set of variables, usually denoted by h, that cannot be bound to infinite terms. Not surprisingly, H is too weak to attain any reasonable precision. One of the reasons is that it lacks any information about the *sharing* of program variables. Consider how the H domain could safely approximate a unification of the form $x = y$ even assuming that, before the unification, x and y can only be bound to finite terms, i.e., $x, y \in h$. In H there is no information that can exclude that x and y share a common variable, and there is no information to ensure that both x and y are unbound variables. Thus, the abstract unification operator of H cannot exclude that $x = y$ results in the creation of cyclic terms. What is worse, since nothing is known about the sharing of variables between x, y and the other variables of interest, no variable can safely remain in h after the binding.

For this reason, the parametric domain $H \times P$ of [3] combines H with a generic domain P (the *parameter* of the construction) providing sharing information. Here the term "sharing information" is to be understood in its broader meaning, which includes variable aliasing, groundness, linearity, freeness and any other kind of information that can improve the precision on these components, such as explicit structural information. Several domain combinations and abstract operators have been proposed in the literature to capture these properties, and are characterized by different precision/complexity trade-offs (see [6] for an account of some of them).

Sharing information is exploited in $H \times P$ for two purposes: detecting when new infinite terms are possibly created (this is done along the lines of [39]) and confining the propagation of those terms as much as possible. As shown in [3], by giving a generic specification for this parameter component in terms of the *abstract queries* it supports [15], it is possible to define and establish the correctness of the abstract operators on the finite-tree domain independently from any particular domain for sharing analysis.

[1] Even though `sort/2` is not required to be a built-in by the standard, it is offered as such by several implementations.
[2] SICStus 3.8.5 still loops on `?- X = [97|X], name(Y, X).`

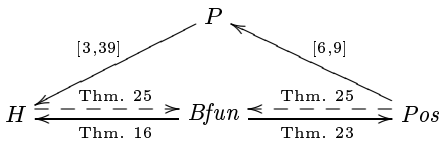

Figure 1: The domains discussed in this paper and the information flows between them.

1.2 Propagating Guarantees of Finiteness

The domain $H \times P$ captures the negative aspect of term-finiteness, that is, the circumstances under which finiteness can be lost. When a binding has the potential for creating one or more rational terms, the abstract unification operator of $H \times P$ removes from h all the variables that may be bound to non-finite terms.

However, term-finiteness has also a positive aspect: there are cases where a variable is granted to be bound to a finite term and this knowledge can be propagated to other variables. Guarantees of finiteness are provided by several built-ins like `unify_with_occurs_check/2`, `var/1`, `name/2`, all the arithmetic predicates, besides those explicitly provided to test for term-finiteness such as the `acyclic_term/1` predicate of SICStus Prolog.[3] The term-finiteness information encoded by H is *attribute independent* [18], which means that each variable is considered in isolation. What is missing is information concerning how finiteness of one variable affects the finiteness of other variables. This kind of information, usually termed *relational information*, is not captured at all by H and it is only partially captured by the composite domain $H \times P$ of [3].

Here we present a domain of Boolean functions that precisely captures how the finiteness of some variables influences the finiteness of other variables. This domain of *finite-tree dependencies* provides relational information that is important for the precision of the overall finiteness analysis. It also combines obvious similarities, interesting differences and somewhat unexpected connections with classical domains for *groundness dependencies*.

Finite-tree and groundness dependencies are similar in that they both track *covering* information (a term s covers t if all the variables in t also occur in s) and share several abstract operations. However, they are different because covering does not tell the whole story. Suppose x and y are free variables before either the unification $x = f(y)$ or the unification $x = f(x, y)$ are executed. In both cases, x will be ground if and only if y will be so. When $x = f(y)$ is the performed unification, this equivalence will also carry over to finiteness. In contrast, when the unification is $x = f(x, y)$, x will never be finite and will be totally independent, as far as finiteness is concerned, from y. Among the unexpected connections is the fact that finite-tree dependencies can improve the groundness information obtained by the usual approaches to groundness analysis.

Figure 1 summarizes the information flows between the domains discussed in this paper.

The paper is structured as follows: the required notations and preliminary concepts are given in Section 2; the concrete domain for the analysis is presented in Section 3; Section 4 introduces the use of Boolean functions for tracking finite-tree dependencies, whereas Section 5 illustrates the interaction between groundness and finite-tree dependencies. The paper is concluded in Section 6 with some final remarks and observations.

[3]On most implementations conforming to the ISO Prolog standard, the predicate `acyclic_term/1` can be defined by `acyclic_term(T) :- unify_with_occurs_check(T, _).`

2. PRELIMINARIES

2.1 Infinite Terms and Substitutions

For a set S, $\wp(S)$ is the powerset of S, whereas $\wp_{\mathrm{f}}(S)$ is the set of all the *finite* subsets of S. Let *Sig* denote a possibly infinite set of function symbols, ranked over the set of natural numbers. Let *Vars* denote a denumerable set of variable symbols, 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 symbol 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 not containing variables) and one with rank greater than 0 (so that there exist infinite terms).

A path $p \in \big(\mathbb{N} \setminus \{0\}\big)^\star$ is any finite sequence of (non-zero) natural numbers. The empty path is denoted by ϵ, whereas $i.p$ denotes the path obtained by concatenating the sequence formed by the natural number $i \neq 0$ with the sequence of the path p. 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.

If $t \in \mathit{Terms}$ then $\mathrm{vars}(t)$ denotes the set of variables occurring in t. If $\mathrm{vars}(t) = \varnothing$ then t is said to be *ground*; t is a *finite term* (or *Herbrand term*) if it contains a finite number of occurrences of function symbols. The sets of all ground and finite terms are denoted by *GTerms* and *HTerms*, respectively.

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

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

A *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, defined as

$$\mathrm{dom}(\sigma) \stackrel{\text{def}}{=} \big\{\, x \in \mathit{Vars} \;\big|\; \sigma(x) \neq x \,\big\},$$

is a finite set of variables.

Given a substitution $\sigma\colon \mathit{Vars} \to \mathit{HTerms}$, the symbol '$\sigma$' also denotes the function $\sigma\colon \mathit{HTerms} \to \mathit{HTerms}$ defined as

follows, for each term $t \in \mathit{HTerms}$:

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

If $t \in \mathit{HTerms}$, we write $t\sigma$ to denote $\sigma(t)$.

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 conveniently denoted by the set of their bindings. Thus a substitution σ is identified with the (finite) set

$$\big\{\, x \mapsto \sigma(x) \;\big|\; x \in \mathrm{dom}(\sigma) \,\big\}.$$

We denote by $\mathrm{vars}(\sigma)$ the set of all variables occurring in the bindings of σ.

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* (RSF) if it has no circular subset. The set of all substitutions in rational solved form is denoted by *RSubst*.

The composition of substitutions is defined in the usual way. Thus $\tau \circ \sigma$ is the substitution such that, for all terms $t \in \mathit{HTerms}$,

$$(\tau \circ \sigma)(t) = \tau\big(\sigma(t)\big)$$

and has the formulation

$$\tau \circ \sigma = \big\{\, x \mapsto x\sigma\tau \;\big|\; x \in \mathrm{dom}(\sigma), x \neq x\sigma\tau \,\big\} \cup \big\{\, x \mapsto x\tau \;\big|\; x \in \mathrm{dom}(\tau) \setminus \mathrm{dom}(\sigma) \,\big\}.$$

As usual, σ^0 denotes the identity function (i.e., the empty substitution) and, when $i \in \mathbb{N}$ with $i > 0$, 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 by $\sigma^\infty(s)$ [27, 31]. Therefore, the function $\mathrm{rt}\colon \mathit{HTerms} \times \mathit{RSubst} \to \mathit{Terms}$ such that

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

is well defined. In general, this function is not a substitution: while having a finite domain, its "bindings" $x \mapsto t$ can map a domain variable x into a term $t \in \mathit{Terms} \setminus \mathit{HTerms}$.

Some of the proofs in the following sections rely on the following properties of the 'rt' function.

PROPOSITION 1. *Let $\sigma \in \mathit{RSubst}$ and $t \in \mathit{HTerms}$. Then*

$$\mathrm{vars}\big(\mathrm{rt}(t, \sigma)\big) \cap \mathrm{dom}(\sigma) = \varnothing, \tag{1a}$$

$$\mathrm{rt}(t, \sigma) \in \mathit{HTerms} \iff \exists i \in \mathbb{N} \,.\, \mathrm{rt}(t, \sigma) = t\sigma^i. \tag{1b}$$

PROOF.

(1a) Let $x \in \mathrm{dom}(\sigma)$ and, towards a contradiction, suppose $x \in \mathrm{vars}\big(\mathrm{rt}(s, \sigma)\big)$. Thus, there exists a finite path p such that $x = \mathrm{rt}(s, \sigma)[p]$. Thus, by definition of 'rt', there exists an index $i \in \mathbb{N}$ such that $x = \sigma^i(s)[p]$. Since $x \in \mathrm{dom}(\sigma)$, then $x \neq x\sigma$, so that $x \neq \sigma^{i+1}(s)[p]$. Also note that, being $\sigma \in \mathit{RSubst}$, σ contains no circular subsets, so that we have $x \neq \sigma^j(s)[p]$, for each index $j > i$. This implies $x \neq \mathrm{rt}(s, \sigma)[p]$, which is a contradiction. Since no such finite path p can exist, we can conclude $x \notin \mathrm{vars}\big(\mathrm{rt}(s, \sigma)\big)$.

(1b) Since substitutions map finite terms into finite terms, a finite number of applications cannot produce an infinite term, so that the left implication holds. Proving the right implication by contraposition, suppose that $\mathrm{rt}(t, \sigma) \neq t\sigma^i$, for all $i \in \mathbb{N}$. Then, by definition of 'rt', we have $t\sigma^i \neq t\sigma^{i+1}$, for all $i \in \mathbb{N}$. Letting $n \in \mathbb{N}$ be the number of bindings in $\sigma \in \mathit{RSubst}$, for all $i \in \mathbb{N}$ it holds $\mathrm{size}(t\sigma^i) < \mathrm{size}(t\sigma^{i+n})$, because σ has no circular subsets. Thus $\mathrm{rt}(t, \sigma) \notin \mathit{HTerms}$, because there is no finite upper bound to the number of function symbols occurring in $\mathrm{rt}(t, \sigma)$. □

2.2 Equations

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 a finite set of equations, that is, as the set $\{\, x = t \mid x \mapsto t \in \sigma \,\}$. We say that a set of equations e is in *rational solved form* if $\big\{\, s \mapsto t \;\big|\; (s = t) \in e \,\big\} \in \mathit{RSubst}$. 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).

Some logic-based languages, such as Prolog II, SICStus and Oz, are based on $\mathcal{RT}$, the theory of rational trees [10, 11]. This is a syntactic equality theory (i.e., a theory where the function symbols are uninterpreted), augmented with a *uniqueness axiom* for each substitution in rational solved form. Informally speaking these axioms state that, if a ground rational tree is assigned to each of the non-domain variables of a substitution, then this substitution uniquely defines a ground rational tree for each of its domain variables. Thus, any set of equations in rational solved form is, by definition, satisfiable in $\mathcal{RT}$.[4]

Given a set of equations $e \in \wp_{\mathrm{f}}(\mathit{Eqs})$ that is satisfiable in $\mathcal{RT}$, a substitution $\sigma \in \mathit{RSubst}$ is called a *solution for e in $\mathcal{RT}$* if $\mathcal{RT} \vdash \forall(\sigma \to e)$. If in addition $\mathrm{vars}(\sigma) \subseteq \mathrm{vars}(e)$, then σ is said to be a *relevant* solution for e. Finally, σ is a *most general solution for e in $\mathcal{RT}$* if $\mathcal{RT} \vdash \forall(\sigma \leftrightarrow e)$. In this paper, the set of all the relevant most general solution for e in $\mathcal{RT}$ will be denoted by $\mathrm{mgs}(e)$.

Definition 1. **($\downarrow(\cdot)\colon \mathit{RSubst} \to \wp(\mathit{RSubst})$.)** The function $\downarrow(\cdot)\colon \mathit{RSubst} \to \wp(\mathit{RSubst})$ is defined, for each $\sigma \in \mathit{RSubst}$, by

$$\downarrow\sigma \stackrel{\text{def}}{=} \big\{\, \tau \in \mathit{RSubst} \;\big|\; \exists \sigma' \in \mathit{RSubst} \,.\, \tau \in \mathrm{mgs}(\sigma \cup \sigma') \,\big\}.$$

The function $\downarrow(\cdot)$ corresponds to the closure by entailment in the equality theory $\mathcal{RT}$.

PROPOSITION 2. *Let $\sigma \in \mathit{RSubst}$. Then*

$$\downarrow\sigma = \big\{\, \tau \in \mathit{RSubst} \;\big|\; \mathcal{RT} \vdash \forall(\tau \to \sigma) \,\big\}.$$

PROOF. The result follows by the following chain of equivalences, where the left implication in the last step is obtained by taking $\sigma' = \tau$.

[4] 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 more "lazy" than one would usually expect. We refer the interested reader to [29, 30, 32] for details on the subject.

$$\begin{aligned}\tau \in \downarrow\sigma &\iff \exists\sigma' \in \mathit{RSubst}\,.\,\tau \in \mathrm{mgs}(\sigma\cup\sigma')\\ &\iff \exists\sigma' \in \mathit{RSubst}\,.\,\mathcal{RT}\vdash\forall\big(\tau\leftrightarrow(\sigma\cup\sigma')\big)\\ &\iff \exists\sigma' \in \mathit{RSubst}\,.\\ &\qquad \mathcal{RT}\vdash\Big(\forall(\tau\to\sigma)\wedge\forall(\tau\to\sigma')\\ &\qquad\qquad \wedge\forall\big((\sigma\cup\sigma')\to\tau\big)\Big)\\ &\iff \mathcal{RT}\vdash\forall(\tau\to\sigma).\end{aligned}$$

□

Notice that, since entailment is a transitive relation, we have, for all $\sigma,\tau,\upsilon \in \mathit{RSubst}$,

$$\upsilon\in\downarrow\tau \;\wedge\; \tau\in\downarrow\sigma \quad\Longrightarrow\quad \upsilon\in\downarrow\sigma.$$

The following results are needed in the sequel: Lemma 3 is a simple generalization of [26, Lemma 1]; Lemma 4 has been proved in [26]; Lemma 5 has been proved in [3]; finally, Lemma 6, which is new, relates the function 'rt' and the concept of equality under the theory $\mathcal{RT}$.

LEMMA 3. *Let $\sigma\in\mathit{RSubst}$ and $\{x\mapsto t\}\in\mathit{RSubst}$ be both satisfiable in the equality theory T, where $x\notin\mathrm{dom}(\sigma)$ and $\mathrm{vars}(t)\cap\mathrm{dom}(\sigma)=\varnothing$. Let also $\sigma'\stackrel{\mathrm{def}}{=}\sigma\cup\{x\mapsto t\}$. Then $\sigma'\in\mathit{RSubst}$ and σ' is satisfiable in T.*

PROOF. Note that σ' is a substitution, since $\sigma\in\mathit{RSubst}$ and $x\notin\mathrm{dom}(\sigma)$. Moreover, as $\mathrm{vars}(t)\cap\mathrm{dom}(\sigma)=\varnothing$, σ' cannot contain circular subsets. Hence, $\sigma'\in\mathit{RSubst}$.

Since both σ and $\{x\mapsto t\}$ are satisfiable in T, we have

$$\begin{aligned}&T\vdash\forall\,\mathit{Vars}\setminus\mathrm{dom}(\sigma):\exists\,\mathrm{dom}(\sigma)\,.\,\sigma,\\ &T\vdash\forall\,\mathit{Vars}\setminus\{x\}:\exists x\,.\,\{x=t\}.\end{aligned}$$

Hence, since $x\notin\mathrm{dom}(\sigma)$,

$$\begin{aligned}&T\vdash\forall\,\mathit{Vars}\setminus\big(\mathrm{dom}(\sigma)\cup\{x\}\big):\\ &\qquad\exists\big(\mathrm{dom}(\sigma)\cup\{x\}\big)\,.\,\sigma\cup\{x=t\}.\end{aligned}$$

Thus σ' is satisfiable in T. □

LEMMA 4. *Let T be an equality theory, $\sigma\in\mathit{RSubst}$ and $t\in\mathit{HTerms}$. Then*

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

LEMMA 5. *Let $s\in\mathit{GTerms}\cap\mathit{HTerms}$ and $t\in\mathit{HTerms}$, where $\mathrm{size}(t)>\mathrm{size}(s)$. Let also T be any syntactic equality theory. Then $T\vdash\forall(s\neq t)$.*

LEMMA 6. *Let $\sigma\in\mathit{RSubst}$ and $s,t\in\mathit{HTerms}$, where $\mathcal{RT}\vdash\forall\big(\sigma\to(s=t)\big)$. Then $\mathrm{rt}(s,\sigma)=\mathrm{rt}(t,\sigma)$.*

PROOF. Suppose, towards a contradiction, that it holds $\mathrm{rt}(s,\sigma)\neq\mathrm{rt}(t,\sigma)$. Then, there must exist a finite path p such that:

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

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

c. $r_1=\mathrm{rt}(s,\sigma)[p]\notin\mathit{Vars}$, $r_2=\mathrm{rt}(t,\sigma)[p]\notin\mathit{Vars}$ 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. $x=s\sigma^i[p]\in\mathit{Vars}\setminus\mathrm{dom}(\sigma)$, $y=t\sigma^i[p]\in\mathit{Vars}\setminus\mathrm{dom}(\sigma)$ and $x\neq y$; or
2. $x=s\sigma^i[p]\in\mathit{Vars}\setminus\mathrm{dom}(\sigma)$ and $r=t\sigma^i[p]\notin\mathit{Vars}$ or, symmetrically, $r=s\sigma^i[p]\notin\mathit{Vars}$ and $x=t\sigma^i[p]\in\mathit{Vars}\setminus\mathrm{dom}(\sigma)$; or
3. $r_1=s\sigma^i[p]\notin\mathit{Vars}$ and $r_2=t\sigma^i[p]\notin\mathit{Vars}$ have different principal functors.

By Lemma 4, we have $\mathcal{RT}\vdash\forall\big(\sigma\to(s\sigma^i=t\sigma^i)\big)$; from this, since $\mathcal{RT}$ is a syntactic equality theory, we obtain that

$$\mathcal{RT}\vdash\forall\big(\sigma\to(s\sigma^i[p]=t\sigma^i[p])\big). \tag{2}$$

We now prove that each case leads to a contradiction.

Consider case 1. Let $r_1,r_2\in\mathit{GTerms}\cap\mathit{HTerms}$ be terms having different principal functors, so that $\mathcal{RT}\vdash\forall(r_1\neq r_2)$. By Lemma 3, we have $\sigma'=\sigma\cup\{x\mapsto r_1,y\mapsto r_2\}\in\mathit{RSubst}$ is satisfiable and $\mathcal{RT}\vdash\forall(\sigma'\to\sigma)$, $\mathcal{RT}\vdash\forall\big(\sigma'\to(x=r_1)\big)$, $\mathcal{RT}\vdash\forall\big(\sigma'\to(y=r_2)\big)$. This is a contradiction, since, by (2), we have $\mathcal{RT}\vdash\forall\big(\sigma\to(x=y)\big)$.

Consider case 2. Without loss of generality, consider the first subcase, where $x=s\sigma^i$ and $r=t\sigma^i[p]\notin\mathit{Vars}$. Let $r'\in\mathit{GTerms}\cap\mathit{HTerms}$ be such that r and r' have different principal functors, so that $\mathcal{RT}\vdash\forall(r\neq r')$. By Lemma 3, $\sigma'=\sigma\cup\{x\mapsto r'\}\in\mathit{RSubst}$ is satisfiable; we also have $\mathcal{RT}\vdash\forall(\sigma'\to\sigma)$ and $\mathcal{RT}\vdash\forall\big(\sigma'\to(x=r')\big)$. This is a contradiction, since, by (2), $\mathcal{RT}\vdash\forall\big(\sigma\to(x=r)\big)$.

Finally, consider case 3. In this case $\mathcal{RT}\vdash\forall(r_1\neq r_2)$. This immediately leads to a contradiction, since, by (2), $\mathcal{RT}\vdash\forall\big(\sigma\to(r_1=r_2)\big)$. □

2.3 Boolean Functions

We now introduce Boolean functions based on the notion of Boolean valuation. An important class of Boolean functions that has been used for data-flow analysis of logic-based languages is *Pos* [1], introduced in [33] under the name *Prop* and further refined and studied in [13, 34].

Definition 2. **(Boolean valuations.)** Consider any finite set of variables $\mathit{VI}\in\wp_{\mathrm{f}}(\mathit{Vars})$ and let $\mathbb{B}\stackrel{\mathrm{def}}{=}\{0,1\}$. The set of *Boolean valuations over VI* is given by

$$\mathit{Bval}\stackrel{\mathrm{def}}{=}\mathit{VI}\to\mathbb{B}.$$

For each $a\in\mathit{Bval}$, each $x\in\mathit{VI}$, and each $c\in\mathbb{B}$ the valuation $a[c/x]\in\mathit{Bval}$ is given, for each $y\in\mathit{VI}$, by

$$a[c/x](y)\stackrel{\mathrm{def}}{=}\begin{cases}c, & \text{if } x=y;\\ a(y), & \text{otherwise.}\end{cases}$$

For $X=\{x_1,\ldots,x_k\}\subseteq\mathit{VI}$, we write $a[c/X]$ as a shorthand for $a[c/x_1]\cdots[c/x_k]$. The distinguished elements $\mathbf{0},\mathbf{1}\in\mathit{Bval}$ are given by

$$\begin{aligned}\mathbf{0}&\stackrel{\mathrm{def}}{=}\lambda x\in\mathit{VI}\,.\,0,\\ \mathbf{1}&\stackrel{\mathrm{def}}{=}\lambda x\in\mathit{VI}\,.\,1.\end{aligned}$$

Definition 3. **(Boolean functions.)** The set of *Boolean functions over VI* is

$$Bfun \stackrel{\text{def}}{=} Bval \to \mathbb{B}.$$

Bfun is partially ordered by the relation $\models$ where, for each $\phi, \psi \in Bfun$,

$$\phi \models \psi \iff \big(\forall a \in Bval : \phi(a) = 1 \implies \psi(a) = 1\big).$$

The distinguished elements $\top, \bot \in Bfun$ are the functions defined, respectively, by

$$\bot \stackrel{\text{def}}{=} \lambda a \in Bval \,.\, 0,$$
$$\top \stackrel{\text{def}}{=} \lambda a \in Bval \,.\, 1.$$

For $\phi \in Bfun$, $x \in VI$, and $c \in \mathbb{B}$, the Boolean function $\phi[c/x] \in Bfun$ is given, for each $a \in Bval$, by

$$\phi[c/x](a) \stackrel{\text{def}}{=} \phi\big(a[c/x]\big).$$

When $X \subseteq VI$, $\phi[c/X]$ is defined in the expected way. If $\phi \in Bfun$ and $x, y \in VI$ the function $\phi[y/x] \in Bfun$ is given, for each $a \in Bval$, by

$$\phi[y/x](a) \stackrel{\text{def}}{=} \phi\Big(a\big[a(y)/x\big]\Big).$$

Boolean functions are constructed from the elementary functions corresponding to variables and by means of the usual logical connectives. Thus x denotes the Boolean function ϕ such that, for each $a \in Bval$, $\phi(a) = 1$ if and only if $a(x) = 1$. For $\phi_1, \phi_2 \in Bfun$, we write $\phi_1 \wedge \phi_2$ to denote the function ϕ such that, for each $a \in Bval$, $\phi(a) = 1$ if and only if both $\phi_1(a) = 1$ and $\phi_2(a) = 1$. A variable is restricted away using Schröder's elimination principle [37]:

$$\exists x \,.\, \phi \stackrel{\text{def}}{=} \phi[1/x] \vee \phi[0/x].$$

Existential quantification is both monotonic and extensive on *Bfun*. The other Boolean connectives and quantifiers are handled similarly.

$Pos \subset Bfun$ consists precisely of those functions assuming the true value under the *everything-is-true* assignment, i.e.,

$$Pos \stackrel{\text{def}}{=} \big\{\, \phi \in Bfun \;\big|\; \phi(\mathbf{1}) = 1 \,\big\}.$$

In what follows we will need the notion of entailed and disentailed variables of a Boolean function.

Definition 4. **(True and false variables.)** For a function $\phi \in Bfun$, the set of *variables necessarily true for* ϕ and the set of *variables necessarily false for* ϕ are given, respectively, by

$$\mathrm{true}(\phi) \stackrel{\text{def}}{=} \big\{\, x \in VI \;\big|\; \forall a \in Bval : \phi(a) = 1 \implies a(x) = 1 \,\big\},$$
$$\mathrm{false}(\phi) \stackrel{\text{def}}{=} \big\{\, x \in VI \;\big|\; \forall a \in Bval : \phi(a) = 1 \implies a(x) = 0 \,\big\}.$$

3. THE CONCRETE DOMAIN

Throughout the paper, we assume a knowledge of the basic concepts of abstract interpretation theory [16, 19].

For the purpose of this paper, we assume a concrete domain consisting of pairs of the form (Σ, V), where V is a finite set of *variables of interest* and Σ is a (possibly infinite) set of substitutions in rational solved form.

Definition 5. **(The concrete domain.)** Let

$$\mathcal{D}^\flat \stackrel{\text{def}}{=} \wp(RSubst) \times \wp_{\mathrm{f}}(Vars).$$

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

$$\big\{\, \exists \Delta \,.\, \sigma \;\big|\; \sigma \in \Sigma,\, \Delta = \mathrm{vars}(\sigma) \setminus V \,\big\}$$

where σ is interpreted as the logical conjunction of the equations corresponding to its bindings. The operation of projecting $x \in Vars$ away from $(\Sigma, V) \in \mathcal{D}^\flat$ is defined as follows:

$$\exists x \,.\, (\Sigma, V) \stackrel{\text{def}}{=} \left\{ \sigma' \in RSubst \;\middle|\; \begin{array}{l} \sigma \in \Sigma, \\ \overline{V} = Vars \setminus V, \\ \mathcal{RT} \vdash \forall\big(\exists \overline{V} \,.\, (\sigma' \leftrightarrow \exists x \,.\, \sigma)\big) \end{array} \right\}.$$

Concrete domains for constraint 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 [4] for an example).

The concrete element

$$\big(\{\{x \mapsto f(y)\}\}, \{x, y\}\big)$$

expresses a dependency between x and y. In contrast,

$$\big(\{\{x \mapsto f(y)\}\}, \{x\}\big)$$

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. Thus, the set of variables of interest is crucial for defining the meaning of the concrete and abstract descriptions. Despite this, always specifying the set of variables of interest would significantly clutter the presentation. Moreover, most of the needed functions on concrete and abstract descriptions, preserve the set of variables of interest. For these reasons, we assume the existence of a set $VI \in \wp_{\mathrm{f}}(Vars)$ that contains, at each stage of the analysis, the current variables of interest.[5] As a consequence, when the context makes it clear that $\Sigma \in \wp(RSubst)$, we will write $\Sigma \in \mathcal{D}^\flat$ as a shorthand for $(\Sigma, VI) \in \mathcal{D}^\flat$.

3.1 Operators on Substitutions in RSF

There are cases when an analysis tries to capture properties of the particular substitutions computed by a specific rational unification algorithm. For instance, this is the case for analyses tracking structure sharing for the purpose of compile-time garbage collection, or providing upper bounds to the amount of memory needed to perform a given computation. More often the interest is on properties of the rational trees themselves. In these cases it is possible to define abstraction and concretization functions that are independent from the finite representations actually computed. Moreover, it is important that these functions precisely capture the properties under investigation, so as to avoid any unnecessary precision loss.

[5] This parallels what happens in the efficient implementation of data-flow analyzers. In fact, almost all the abstract domains currently in use do not need to represent explicitly the set of variables of interest. In contrast, this set is maintained externally and in a unique copy, typically by the fixpoint computation engine.

Pursuing this goal requires the ability to observe properties of (infinite) rational trees while just dealing with one of their finite representations. This is not always an easy task, since even simple properties can be "hidden" when using non-idempotent substitutions. For instance, when $\mathrm{rt}(x,\sigma) \in \mathit{GTerms} \setminus \mathit{HTerms}$ is an infinite and ground rational tree, all of its finite representations in *RSubst* will map the variable x into a finite term that is not ground.

These are the motivations behind the introduction of two computable operators on substitutions that will later be used to define the concretization functions for the considered abstract domains. First, the groundness operator 'gvars' captures the set of variables that are mapped to ground rational trees by the 'rt' function. We define it by means of the *occurrence operator* 'occ' introduced in [26].

Definition 6. **(Occurrence functions.)** For each $n \in \mathbb{N}$, the *occurrence function* $\mathrm{occ}_n \colon \mathit{RSubst} \times \mathit{Vars} \to \wp_{\mathrm{f}}(\mathit{Vars})$ is defined, for each $\sigma \in \mathit{RSubst}$ and each $v \in \mathit{Vars}$, by

$$\mathrm{occ}_0(\sigma, v) \stackrel{\mathrm{def}}{=} \{v\} \setminus \mathrm{dom}(\sigma)$$

and, for $n > 0$, by

$$\mathrm{occ}_n(\sigma, v) \stackrel{\mathrm{def}}{=} \big\{\, y \in \mathit{Vars} \;\big|\; \mathrm{vars}(y\sigma) \cap \mathrm{occ}_{n-1}(\sigma, v) \neq \varnothing \,\big\}.$$

For each $\sigma \in \mathit{RSubst}$, $v \in \mathit{Vars}$ and each $n \geq 0$, we have $\mathrm{occ}_n(\sigma, v) \subseteq \mathrm{vars}(\sigma) \cup \{v\}$, where $\mathrm{vars}(\sigma)$ is a finite set. Also, $\mathrm{occ}_n(\sigma, v) \subseteq \mathrm{occ}_{n+1}(\sigma, v)$. Thus the following operator is finitely computable.

Definition 7. **(Occurrence operator.)** The *occurrence operator* $\mathrm{occ} \colon \mathit{RSubst} \times \mathit{Vars} \to \wp_{\mathrm{f}}(\mathit{Vars})$ is given, for each substitution $\sigma \in \mathit{RSubst}$ and $v \in \mathit{Vars}$, by

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

where $\ell \in \mathbb{N}$ is such that $\mathrm{occ}_\ell(\sigma, v) = \mathrm{occ}_n(\sigma, v)$ for all $n \geq \ell$.

Definition 8. **(Groundness operator.)** The *groundness operator* $\mathrm{gvars} \colon \mathit{RSubst} \to \wp_{\mathrm{f}}(\mathit{Vars})$ is given, for each substitution $\sigma \in \mathit{RSubst}$, by

$$\mathrm{gvars}(\sigma) \stackrel{\mathrm{def}}{=} \big\{\, y \in \mathrm{dom}(\sigma) \;\big|\; \forall v \in \mathrm{vars}(\sigma) : y \notin \mathrm{occ}(\sigma, v) \,\big\}.$$

The finiteness operator 'hvars', introduced in [3], captures the set of variables that 'rt' maps to finite terms.

Definition 9. **(Finiteness functions.)** For each $n \in \mathbb{N}$, the *finiteness function* $\mathrm{hvars}_n \colon \mathit{RSubst} \to \wp(\mathit{Vars})$ is defined, for each $\sigma \in \mathit{RSubst}$, by

$$\mathrm{hvars}_0(\sigma) \stackrel{\mathrm{def}}{=} \mathit{Vars} \setminus \mathrm{dom}(\sigma)$$

and, for $n > 0$, by

$$\mathrm{hvars}_n(\sigma) \stackrel{\mathrm{def}}{=} \mathrm{hvars}_{n-1}(\sigma) \cup \big\{\, y \in \mathrm{dom}(\sigma) \;\big|\; \mathrm{vars}(y\sigma) \subseteq \mathrm{hvars}_{n-1}(\sigma) \,\big\}.$$

Observe that, for each $\sigma \in \mathit{RSubst}$ and each $n \in \mathbb{N}$ with $n > 0$, we have $\mathrm{hvars}_n(\sigma) \subseteq \mathrm{hvars}_{n+1}(\sigma)$. Note also that $\mathit{Vars} \setminus \mathrm{hvars}_n(\sigma) \subseteq \mathrm{dom}(\sigma)$ is a finite set. By these two properties, the following fixpoint computation is well defined and finitely computable.

Definition 10. **(Finiteness operator.)** The *finiteness operator* $\mathrm{hvars} \colon \mathit{RSubst} \to \wp(\mathit{Vars})$ is given, for each substitution $\sigma \in \mathit{RSubst}$, by

$$\mathrm{hvars}(\sigma) \stackrel{\mathrm{def}}{=} \mathrm{hvars}_\ell(\sigma)$$

where $\ell \stackrel{\mathrm{def}}{=} \ell(\sigma) \in \mathbb{N}$ is such that $\mathrm{hvars}_\ell(\sigma) = \mathrm{hvars}_n(\sigma)$ for each $n \geq \ell$.

The following proposition summarizes two results proved in [26] and [3], and shows that the functions 'gvars' and 'hvars' precisely capture the intended properties.

PROPOSITION 7. *Let $\sigma \in \mathit{RSubst}$ and $x \in \mathit{Vars}$. Then*

$$y \in \mathrm{gvars}(\sigma) \iff \mathrm{rt}(y,\sigma) \in \mathit{GTerms}, \tag{7a}$$
$$y \in \mathrm{hvars}(\sigma) \iff \mathrm{rt}(y,\sigma) \in \mathit{HTerms}. \tag{7b}$$

COROLLARY 8. *Let $\sigma \in \mathit{RSubst}$ and $t \in \mathit{HTerms}$. Then*

$$\mathrm{vars}(t) \subseteq \mathrm{gvars}(\sigma) \iff \mathrm{rt}(t,\sigma) \in \mathit{GTerms}, \tag{8a}$$
$$\mathrm{vars}(t) \subseteq \mathrm{hvars}(\sigma) \iff \mathrm{rt}(t,\sigma) \in \mathit{HTerms}. \tag{8b}$$

Example 1. Let

$$\sigma = \big\{x \mapsto f(y,z), y \mapsto g(z,x), z \mapsto f(a)\big\},$$
$$\tau = \big\{v \mapsto g(z,w), x \mapsto f(y), y \mapsto g(w), z \mapsto f(v)\big\}.$$

Then

$$\mathrm{gvars}(\sigma) \cap \mathrm{vars}(\sigma) = \{x,y,z\},$$
$$\mathrm{hvars}(\tau) \cap \mathrm{vars}(\tau) = \{w,x,y\}.$$

The following proposition states how 'gvars' and 'hvars' behave with respect to the further instantiation of variables.

PROPOSITION 9. *Let $\sigma, \tau \in \mathit{RSubst}$, where $\tau \in {\downarrow}\sigma$. Then*

$$\mathrm{hvars}(\sigma) \supseteq \mathrm{hvars}(\tau), \tag{9a}$$
$$\mathrm{gvars}(\sigma) \cap \mathrm{hvars}(\sigma) \subseteq \mathrm{gvars}(\tau) \cap \mathrm{hvars}(\tau). \tag{9b}$$

PROOF.

(9a). Suppose $x \in \mathrm{hvars}(\tau) \setminus \mathrm{hvars}(\sigma)$. Then, by case (7b) of Proposition 7, $\mathrm{rt}(x,\tau) \in \mathit{HTerms}$. By Proposition 1, there exists $i \in \mathbb{N}$ such that $\mathrm{rt}(x,\tau) = x\tau^i$ and also $\mathrm{vars}(x\tau^i) \cap \mathrm{dom}(\tau) = \varnothing$. Let $t \in \mathit{GTerms} \cap \mathit{HTerms}$ and

$$v \stackrel{\mathrm{def}}{=} \big\{\, y \mapsto t \;\big|\; y \in \mathrm{vars}(x\tau^i) \,\big\}.$$

Then, by Lemma 3, $\tau' \stackrel{\mathrm{def}}{=} \tau \cup v \in \mathit{RSubst}$ is satisfiable. Moreover $x\tau^i\tau' \in \mathit{GTerms} \cap \mathit{HTerms}$. Let $n \stackrel{\mathrm{def}}{=} \mathrm{size}(x\tau^i\tau')$. Note that, since $\mathrm{rt}(x,\sigma) \notin \mathit{HTerms}$, there exists $j \in \mathbb{N}$ such that $\mathrm{size}(x\sigma^j) > n$. Therefore, by Lemma 5,

$$\mathcal{RT} \vdash \forall(x\tau^i\tau' \neq x\sigma^j). \tag{6}$$

Also, by Lemma 4, $\mathcal{RT} \vdash \forall\big(\sigma \to (x = x\sigma^j)\big)$ and $\mathcal{RT} \vdash \forall\big(\tau \to (x = x\tau^i)\big)$. By definition, $\tau' \in {\downarrow}\tau$ and, by hypothesis, $\tau \in {\downarrow}\sigma$, so that $\tau' \in {\downarrow}\sigma$. Thus, by Proposition 2 and transitivity, we have $\mathcal{RT} \vdash \forall\big(\tau' \to (x\tau^i = x\sigma^j)\big)$. Applying Lemma 4, we obtain $\mathcal{RT} \vdash \forall\big(\tau' \to (x\tau^i\tau' = x\sigma^j)\big)$, which contradicts (6).

(9b). Suppose $x \in \mathrm{hvars}(\sigma) \cap \mathrm{gvars}(\sigma)$. Then, by Proposition 7, $\mathrm{rt}(x,\sigma) \in \mathit{GTerms} \cap \mathit{HTerms}$. Thus, by case (1b) of Proposition 1, there exists $i \in \mathbb{N}$ such that $\mathrm{rt}(x,\sigma) = x\sigma^i$ and also $\mathrm{vars}(x\sigma^i) = \varnothing$. Thus $\mathrm{rt}(x\sigma^i,\tau) = x\sigma^i$. Since,

by hypothesis, $\tau \in \downarrow\sigma$, by Lemma 4 and transitivity we obtain $\mathcal{RT} \vdash \forall\big(\tau \to (x = x\sigma^i)\big)$. Thus, by Lemma 6, $\mathrm{rt}(x,\tau) = \mathrm{rt}(x\sigma^i,\tau) = x\sigma^i$. Therefore, by Proposition 7, $x \in \mathrm{gvars}(\tau) \cap \mathrm{hvars}(\tau)$. □

The next result shows how 'hvars' behaves with respect to projecting away some of its variables.

PROPOSITION 10. *Let $\sigma, \tau \in RSubst$ and let $W \subseteq Vars$, where $\mathcal{RT} \vdash \forall(\exists W\,.\,\sigma \leftrightarrow \exists W\,.\,\tau)$. Then*

$$\mathrm{hvars}(\sigma) \setminus W = \mathrm{hvars}(\tau) \setminus W.$$

PROOF. Consider a variable $z \in \mathrm{hvars}(\sigma)\setminus W$. We assume that $z \notin \mathrm{hvars}(\tau)$ to obtain a contradiction.

By case (7b) of Proposition 7, $\mathrm{rt}(z,\sigma) \in HTerms$. By Proposition 1, there exists $i \in \mathbb{N}$ such that $\mathrm{rt}(z,\sigma) = z\sigma^i$ and $\mathrm{vars}(z\sigma^i) \cap \mathrm{dom}(\sigma) = \varnothing$.

Take $t \in GTerms \cap HTerms$ and let

$$\upsilon \stackrel{\mathrm{def}}{=} \big\{\, y \mapsto t \;\big|\; y \in \mathrm{vars}(z\sigma^i) \,\big\}.$$

By Lemma 3, $\sigma' \stackrel{\mathrm{def}}{=} \sigma \cup \upsilon \in \downarrow\sigma$ is satisfiable. Thus, by Proposition 2, we have

$$\mathcal{RT} \vdash \forall(\sigma' \to \sigma). \tag{7}$$

By the definition of σ', $z\sigma^i\sigma' \in GTerms \cap HTerms$. As $z \notin \mathrm{hvars}(\tau)$, there exists $j \in \mathbb{N}$ such that $\mathrm{size}(z\tau^j) > \mathrm{size}(z\sigma^i\sigma')$. Thus, by Lemma 5,

$$\mathcal{RT} \vdash \forall(z\sigma^i\sigma' \neq z\tau^j). \tag{8}$$

By applying Lemma 4, we have $\mathcal{RT} \vdash \forall\big(\sigma \to (z = z\sigma^i)\big)$ and $\mathcal{RT} \vdash \forall\big(\sigma' \to (z\sigma^i = z\sigma^i\sigma')\big)$. Thus, by (7),

$$\mathcal{RT} \vdash \forall\big(\sigma' \to (z = z\sigma^i\sigma')\big). \tag{9}$$

Using (7), the hypothesis and the logically true statement $\forall(\sigma \to \exists W\,.\,\sigma)$, we obtain $\mathcal{RT} \vdash \forall(\sigma' \to \exists W\,.\,\tau)$. By Lemma 4, we have $\mathcal{RT} \vdash \forall\big(\tau \to (z = z\tau^j)\big)$; thus, as $\mathcal{RT}$ is a first-order theory, $\mathcal{RT} \vdash \forall\big(\exists W\,.\,\tau \to \exists W\,.\,(z = z\tau^j)\big)$. Therefore, by transitivity, we obtain

$$\mathcal{RT} \vdash \forall\big(\sigma' \to \exists W\,.\,(z = z\tau^j)\big). \tag{10}$$

Observe now that $\mathrm{vars}(z = z\sigma^i\sigma') = \{z\}$ and, as a consequence, we have $\mathrm{vars}(z = z\sigma^i\sigma') \cap W = \varnothing$. Therefore, by (9) and (10), we obtain

$$\begin{aligned}
&\mathcal{RT} \vdash \forall\big(\sigma' \to (z = z\sigma^i\sigma' \wedge \exists W\,.\,z = z\tau^j)\big)\\
&\quad\iff \mathcal{RT} \vdash \forall\big(\sigma' \to \exists W\,.\,(z = z\sigma^i\sigma' \wedge z = z\tau^j)\big)\\
&\quad\iff \mathcal{RT} \vdash \forall\big(\sigma' \to \exists W\,.\,(z\sigma^i\sigma' = z\tau^j)\big).
\end{aligned}$$

But this contradicts (8), so that the assumption was false and $z \in \mathrm{hvars}(\tau)$. As the choice of z was arbitrary, we have

$$\mathrm{hvars}(\sigma) \setminus W \subseteq \mathrm{hvars}(\tau) \setminus W.$$

The reverse inclusion follows by symmetry. □

4. FINITE-TREE DEPENDENCIES

Any finite-tree domain must keep track of those variables that are definitely bound to finite terms, since this is the final information delivered by the analysis. In [3] we have introduced the composite abstract domain $H \times P$, where the set of such variables is explicitly represented in the *finiteness component* H.

Definition 11. **(The finiteness component H.)** The set $H \stackrel{\mathrm{def}}{=} \wp(VI)$, partially ordered by reverse subset inclusion, is called *finiteness component*. The concretization function $\gamma_H : H \to \wp(RSubst)$ is given, for each $h \in H$, by

$$\gamma_H(h) \stackrel{\mathrm{def}}{=} \big\{\, \sigma \in RSubst \;\big|\; \mathrm{hvars}(\sigma) \supseteq h \,\big\}.$$

As proven in [3], equivalent substitutions in rational solved form have the same finiteness abstraction.

PROPOSITION 11. *Let $\sigma, \tau \in RSubst$, where $\sigma \in \gamma_H(h)$ and $\mathcal{RT} \vdash \forall(\sigma \leftrightarrow \tau)$. Then $\tau \in \gamma_H(h)$.*

As explained before, the precision of the finite-tree analysis of [3] is highly dependent on the precision of the generic component P. The information provided by P on groundness, freeness, linearity, and sharing of variables is exploited, in the combination $H \times P$, to circumscribe as much as possible the creation and propagation of cyclic terms. However, finite-tree analysis can also benefit from other kinds of relational information. In particular, we now show how *finite-tree dependencies* allow to obtain a positive propagation of finiteness information.

Let us consider the finite terms $t_1 = f(x)$, $t_2 = g(y)$, and $t_3 = h(x,y)$: it is clear that, for each assignment of rational terms to x and y, t_3 is finite if and only if t_1 and t_2 are so. We will capture this by the Boolean formula $t_3 \leftrightarrow (t_1 \wedge t_2)$.[6] The reasoning is based on the following facts:

1. t_1, t_2, and t_3 are finite terms, so that the finiteness of their instances depends only on the finiteness of the terms that take the place of x and y.
2. $\mathrm{vars}(t_3) \supseteq \mathrm{vars}(t_1) \cup \mathrm{vars}(t_2)$, that is, t_3 *covers* both t_1 and t_2; this means that, if an assignment to the variables of t_3 produces a finite instance of t_3, that very assignment will necessarily result in finite instances of t_1 and t_2. Conversely, an assignment producing non-finite instances of t_1 or t_2 will forcibly result in a non-finite instance of t_3.
3. Similarly, t_1 and t_2, taken together, cover t_3.

The important point to notice is that the indicated dependency will keep holding for any further simultaneous instantiation of t_1, t_2, and t_3. In other words, such dependencies are preserved by forward computations (which proceed by consistently instantiating program variables).

Consider the abstract binding $x \mapsto t$ where t is a finite term such that $\mathrm{vars}(t) = \{y_1, \ldots, y_n\}$. After this binding has been successfully performed, the destinies of x and t concerning term-finiteness are tied together: forever. This tie can be described by the dependency formula

$$x \leftrightarrow (y_1 \wedge \cdots \wedge y_n), \tag{11}$$

meaning that x will be bound to a finite term if and only if y_i is bound to a finite term, for each $i = 1, \ldots, n$. While the dependency expressed by (11) is a correct description of any computation state following the application of the binding $x \mapsto t$, it is not as precise as it could be. Suppose that x and y_k are indeed the same variable. Then (11) is logically equivalent to

$$x \to (y_1 \wedge \cdots \wedge y_{k-1} \wedge y_{k+1} \wedge \cdots \wedge y_n). \tag{12}$$

[6] The introduction of such Boolean formulas, called *dependency formulas*, is originally due to P. W. Dart [20].

Correct: whenever x is bound to a finite term, all the other variables will be bound to finite terms. The point is that x has just been bound to a non-finite term, irrevocably: no forward computation can change this. Thus, the implication (12) holds vacuously. The precise and correct description for the state of affairs caused by the cyclic binding is, instead, the negated atom $\neg x$, whose intuitive reading is "x is not (and never will be) finite."

We are building an abstract domain for finite-tree dependencies where we are making the deliberate choice of including only information that cannot be withdrawn by forward computations. The reason for this choice is that we want the concrete constraint accumulation process to be paralleled, at the abstract level, by another constraint accumulation process: logical conjunction of Boolean formulas. For this reason, it is important to distinguish between *permanent* and *contingent* information. Permanent information, once established for a program point p, maintains its validity in all program points that follow p in any forward computation. Contingent information, instead, does not carry its validity beyond the point where it is established.

An example of contingent information is given by the h component of $H \times P$: having $x \in h$, in the description of some program point, means that x is definitely bound to a finite term *at that program point*; nothing is claimed about the finiteness of x at other program points and, in fact, unless x is ground, x can still be bound to a non-finite term. However, if at some program point variable x is both finite and ground, then x is permanently finite. In this case we will make sure our Boolean dependency formula entails the positive atom x.

At this stage, we already know something about the abstract domain we are designing. In particular, we have positive and negated atoms, the requirement of describing program predicates of any arity implies that arbitrary conjunctions of these atomic formulas must be allowed and, finally, it is not difficult to observe that the merge-over-all-paths operations [16] will be logical disjunction, so that the domain will have to be closed under this operation. This means that the carrier of our domain must be able to express any Boolean function: *Bfun* is the carrier.

Definition 12. ($\gamma_F\colon$ *Bfun* $\to \wp($*RSubst*$)$.) The function hval: *RSubst* $\to$ *Bval* is defined, for each $\sigma \in$ *RSubst* and each $x \in$ *VI*, by

$$\mathrm{hval}(\sigma)(x) = 1 \iff^{\mathrm{def}} x \in \mathrm{hvars}(\sigma).$$

The concretization function $\gamma_F\colon$ *Bfun* $\to \wp($*RSubst*$)$ is given, for each $\phi \in$ *Bfun*, by

$$\gamma_F(\phi) \stackrel{\mathrm{def}}{=} \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : \phi\big(\mathrm{hval}(\tau)\big) = 1 \,\big\}.$$

Finite-tree dependencies only capture information that cannot be withdrawn by any forward computation.

Proposition 12. *Let $\sigma, \tau \in$ RSubst and $\phi \in$ Bfun, where $\sigma \in \gamma_F(\phi)$ and $\tau \in {\downarrow}\sigma$. Then $\tau \in \gamma_F(\phi)$.*

Proof. By the hypothesis, $\tau \in {\downarrow}\sigma$, so that, for each $\upsilon \in {\downarrow}\tau$, $\upsilon \in {\downarrow}\sigma$. Therefore, as $\sigma \in \gamma_F(\phi)$, it follows from Definition 12 that, for all $\upsilon \in {\downarrow}\tau$, $\phi\big(\mathrm{hval}(\upsilon)\big) = 1$ and hence $\tau \in \gamma_F(\phi)$. □

As a consequence, every element of the codomain of γ_F contains equivalence classes with respect to the $\mathcal{RT}$ theory.

Corollary 13. *Let $\sigma, \tau \in$ RSubst, where $\sigma \in \gamma_F(\phi)$ and $\mathcal{RT} \vdash \forall(\sigma \leftrightarrow \tau)$. Then $\tau \in \gamma_F(\phi)$.*

By the next lemma, the γ_F function is *meet-preserving* and hence monotonic. This result also implies that the image of *Bfun* under γ_F is Moore-closed. Hence [17], γ_F has an adjoint (abstraction) function that is given by

$$\alpha_F(\Sigma) \stackrel{\mathrm{def}}{=} \bigwedge \big\{\, \phi \in \mathit{Bfun} \;\big|\; \Sigma \subseteq \gamma_F(\phi) \,\big\}.$$

Lemma 14. *Let $\phi_1, \phi_2 \in$ Bfun. Then*

$$\gamma_F(\phi_1 \wedge \phi_2) = \gamma_F(\phi_1) \cap \gamma_F(\phi_2).$$

Proof.

$$\begin{aligned}
&\gamma_F(\phi_1 \wedge \phi_2)\\
&= \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : (\phi_1 \wedge \phi_2)\big(\mathrm{hval}(\tau)\big) = 1 \,\big\}\\
&= \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : \forall i \in \{1,2\} : \phi_i\big(\mathrm{hval}(\tau)\big) = 1 \,\big\}\\
&= \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : \phi_1\big(\mathrm{hval}(\tau)\big) = 1 \,\big\}\\
&\quad \cap \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : \phi_2\big(\mathrm{hval}(\tau)\big) = 1 \,\big\}\\
&= \gamma_F(\phi_1) \cap \gamma_F(\phi_2).
\end{aligned}$$

□

The next theorem considers most of the operators needed to compute the concrete semantics of a logic program, showing how they can be correctly approximated on the abstract domain *Bfun*.

Theorem 15. *Let $\Sigma, \Sigma_1, \Sigma_2 \in \wp($RSubst$)$ and $\phi, \phi_1, \phi_2 \in$ Bfun, where $\gamma_F(\phi) \supseteq \Sigma$, $\gamma_F(\phi_1) \supseteq \Sigma_1$, and $\gamma_F(\phi_2) \supseteq \Sigma_2$. Let also $(x \mapsto t) \in$ Bind, where $\{x\} \cup \mathrm{vars}(t) \subseteq$ VI. Then the following hold:*

$$\gamma_F\Big(x \leftrightarrow \bigwedge \mathrm{vars}(t)\Big) \supseteq \big\{\{x \mapsto t\}\big\}; \tag{15a}$$

$$\gamma_F(\neg x) \supseteq \big\{\{x \mapsto t\}\big\}, \text{ if } x \in \mathrm{vars}(t); \tag{15b}$$

$$\gamma_F(x) \supseteq \left\{ \sigma \in \mathit{RSubst} \;\middle|\; \begin{array}{r} x \in \mathrm{gvars}(\sigma) \\ \cap\, \mathrm{hvars}(\sigma) \end{array} \right\}; \tag{15c}$$

$$\gamma_F(\phi_1 \wedge \phi_2) \supseteq \big\{\, \mathrm{mgs}(\sigma_1 \cup \sigma_2) \;\big|\; \sigma_1 \in \Sigma_1, \sigma_2 \in \Sigma_2 \,\big\}; \tag{15d}$$

$$\gamma_F(\phi_1 \vee \phi_2) \supseteq \Sigma_1 \cup \Sigma_2; \tag{15e}$$

$$\gamma_F(\exists x \,.\, \phi) \supseteq \exists\, x \,.\, \Sigma. \tag{15f}$$

Proof. Assuming the hypothesis of the theorem, we will prove each relation separately.

(15a). Let $\sigma = \{x \mapsto t\}$. Suppose $\tau \in {\downarrow}\sigma$. Then, by Proposition 2, $\mathcal{RT} \vdash \forall\big(\tau \to \sigma\big)$. It follows from Lemma 6 that $\mathrm{rt}(x, \tau) = \mathrm{rt}(t, \tau)$ and thus, by case (8b) of Corollary 8, $x \in \mathrm{hvars}(\tau)$ if and only if $\mathrm{vars}(t) \subseteq \mathrm{hvars}(\tau)$. This is equivalent to $\big(x \leftrightarrow \bigwedge \mathrm{vars}(t)\big)\big(\mathbf{0}\big[1/\mathrm{hvars}(\tau)\big]\big) = 1$ and, by Definition 12, to $\big(x \leftrightarrow \bigwedge \mathrm{vars}(t)\big)\big(\mathrm{hval}(\tau)\big) = 1$. As this holds for all $\tau \in {\downarrow}\sigma$, by Definition 12, $\sigma \in \gamma_F\big(x \leftrightarrow \bigwedge \mathrm{vars}(t)\big)$.

(15b). Let $\sigma = \{x \mapsto t\}$, where $x \in \mathrm{vars}(t)$. By Definition 10, we have $x \notin \mathrm{hvars}(\sigma)$. By case (9a) of Proposition 9, for all $\tau \in {\downarrow}\sigma$, we have $\mathrm{hvars}(\tau) \subseteq \mathrm{hvars}(\sigma)$. Thus $x \notin \mathrm{hvars}(\tau)$ and $(\neg x)\big(\mathrm{hval}(\tau)\big) = 1$. Therefore, by Definition 12, $\sigma \in \gamma_F(\neg x)$.

(15c). Let $\sigma \in \mathit{RSubst}$ such that $x \in \mathrm{gvars}(\sigma) \cap \mathrm{hvars}(\sigma)$. By case (9b) of Proposition 9, we have $x \in \mathrm{hvars}(\tau)$ for all $\tau \in {\downarrow}\sigma$. So $(x)\big(\mathrm{hval}(\tau)\big) = 1$. Therefore, by Definition 12, $\sigma \in \gamma_F(x)$.

(15d). Let $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$. Then, by hypothesis $\sigma_1 \in \gamma_F(\phi_1)$ and $\sigma_2 \in \gamma_F(\phi_2)$. Let $\tau \in \mathrm{mgs}(\sigma_1 \cup \sigma_2)$. By definition of mgs, $\mathcal{RT} \vdash \forall(\tau \to \sigma_1)$ and $\mathcal{RT} \vdash \forall(\tau \to \sigma_2)$. Thus, by Proposition 2, we have $\tau \in {\downarrow}\sigma_1 \cap {\downarrow}\sigma_2$. Therefore, by Proposition 12, $\tau \in \gamma_F(\phi_1) \cap \gamma_F(\phi_2)$. The result then follows by Lemma 14.

(15e).

$$
\begin{aligned}
&\gamma_F(\phi_1 \vee \phi_2)\\
&= \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : (\phi_1 \vee \phi_2)\big(\mathrm{hval}(\tau)\big) = 1 \,\big\}\\
&= \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : \exists i \in \{1,2\} \,.\, \phi_i\big(\mathrm{hval}(\tau)\big) = 1 \,\big\}\\
&\supseteq \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : \phi_1\big(\mathrm{hval}(\tau)\big) = 1 \,\big\}\\
&\qquad \cup \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : \phi_2\big(\mathrm{hval}(\tau)\big) = 1 \,\big\}\\
&= \gamma_F(\phi_1) \cup \gamma_F(\phi_2)\\
&\supseteq \Sigma_1 \cup \Sigma_2.
\end{aligned}
$$

(15f). Let $\sigma \in \Sigma$ and let $\sigma' \in \exists\, x \,.\, \{\sigma\}$. We will show that $\sigma' \in \gamma_F(\exists x \,.\, \phi)$.

Let $\tau' \in {\downarrow}\sigma'$. Then there exists $\sigma'_1 \in \mathit{RSubst}$ such that $\mathcal{RT} \vdash \forall\big(\tau' \leftrightarrow (\sigma' \cup \sigma'_1)\big)$. Let $\sigma_1 \in \exists\, x \,.\, \{\sigma'_1\}$ and let $W \stackrel{\mathrm{def}}{=} (\mathit{Vars} \setminus \mathit{VI}) \cup \{x\}$. Then, by Definition 5, it follows $\mathcal{RT} \vdash \forall\big(\exists W \,.\, (\sigma' \leftrightarrow \sigma)\big)$ and $\mathcal{RT} \vdash \forall\big(\exists W \,.\, (\sigma'_1 \leftrightarrow \sigma_1)\big)$. As a consequence

$$\mathcal{RT} \vdash \forall\big(\exists W \,.\, (\sigma' \cup \sigma'_1) \leftrightarrow \exists W \,.\, (\sigma \cup \sigma_1)\big).$$

Therefore $\sigma \cup \sigma_1$ is satisfiable so that, for some $\tau \in \mathit{RSubst}$, $\mathcal{RT} \vdash \forall\big(\tau \leftrightarrow (\sigma \cup \sigma_1)\big)$. Thus $\mathcal{RT} \vdash \forall(\exists W \,.\, \tau \leftrightarrow \exists W \,.\, \tau')$. By Proposition 10, $\mathrm{hvars}(\tau') \setminus W = \mathrm{hvars}(\tau) \setminus W$ so that

$$(\mathrm{hvars}(\tau') \cap \mathit{VI}) \cup \{x\} = (\mathrm{hvars}(\tau) \cap \mathit{VI}) \cup \{x\}. \qquad (14)$$

Let $c \stackrel{\mathrm{def}}{=} \mathrm{hval}(\tau)(x)$. Then, since $\tau \in {\downarrow}\sigma$ and, by hypothesis, $\sigma \in \gamma_F(\phi)$, we have the following chain of implications:

$$
\begin{aligned}
\phi\big(\mathrm{hval}(\tau)\big) &= 1 && \text{[by Defn. 12]}\\
\phi\big(\mathrm{hval}(\tau)[c/x]\big) &= 1 && \text{[by Defn. 2]}\\
\phi\big(\mathbf{0}\big[1/\mathrm{hvars}(\tau) \cap \mathit{VI}\big][c/x]\big) &= 1 && \text{[by Defn. 12]}\\
\phi\big(\mathbf{0}\big[1/\big(\mathrm{hvars}(\tau) \cap \mathit{VI}\big) \cup \{x\}\big][c/x]\big) &= 1 && \text{[by Defn. 2]}\\
\phi\big(\mathbf{0}\big[1/\big(\mathrm{hvars}(\tau') \cap \mathit{VI}\big) \cup \{x\}\big][c/x]\big) &= 1 && \text{[by (14)]}\\
\phi\big(\mathbf{0}\big[1/\mathrm{hvars}(\tau') \cap \mathit{VI}\big][c/x]\big) &= 1 && \text{[by Defn. 2]}\\
\phi\big(\mathrm{hval}(\tau')[c/x]\big) &= 1 && \text{[by Defn. 12]}\\
\phi[c/x]\big(\mathrm{hval}(\tau')\big) &= 1. && \text{[by Defn. 3]}
\end{aligned}
$$

From this last relation, since $\phi[c/x] \models \exists x \,.\, \phi$, it follows that $(\exists x \,.\, \phi)\big(\mathrm{hval}(\tau')\big) = 1$. As this holds for all $\tau' \in {\downarrow}\sigma'$, by Definition 12, $\sigma' \in \gamma_F(\exists x \,.\, \phi)$. □

Cases (15a), (15b), and (15d) of Theorem 15 ensure that the following definition of amgu_F provides a correct approximation on *Bfun* of the concrete unification of rational trees.

Definition 13. The function $\mathrm{amgu}_F \colon \mathit{Bfun} \times \mathit{Bind} \to \mathit{Bfun}$ captures the effects of a binding on a finite-tree dependency component. Let $\phi \in \mathit{Bfun}$ and $(x \mapsto t) \in \mathit{Bind}$. Then

$$\mathrm{amgu}_F(\phi, x \mapsto t) \stackrel{\mathrm{def}}{=} \begin{cases} \phi \wedge \big(x \leftrightarrow \bigwedge \mathrm{vars}(t)\big), & \text{if } x \notin \mathrm{vars}(t);\\ \phi \wedge \neg x, & \text{otherwise.} \end{cases}$$

Other semantic operators, such as the consistent renaming of variables, are very simple and, as usual, their approximation does not pose any problem.

The next theorem suggests how finite-tree dependencies can be exploited in order to improve the finiteness information encoded in the h component of the domain $H \times P$.

THEOREM 16. *Let $h \in H$ and $\phi \in \mathit{Bfun}$. Let also*

$$h' \stackrel{\mathrm{def}}{=} \mathrm{true}\Big(\phi \wedge \bigwedge h\Big).$$

Then

$$\gamma_H(h) \cap \gamma_F(\phi) = \gamma_H(h') \cap \gamma_F(\phi).$$

PROOF. Since $h \subseteq h'$, by the monotonicity of γ_H we have $\gamma_H(h) \supseteq \gamma_H(h')$, so that we obtain one of the inclusions: $\gamma_H(h) \cap \gamma_F(\phi) \supseteq \gamma_H(h') \cap \gamma_F(\phi)$.

In order to establish the other inclusion, we now prove that $\sigma \in \gamma_H(h')$ assuming $\sigma \in \gamma_H(h) \cap \gamma_F(\phi)$. To this end, by Definition 11, it is sufficient to prove that $h' \subseteq \mathrm{hvars}(\sigma)$.

Let $z \in h'$ and let $\psi = \big(\phi \wedge \bigwedge h\big)$, so that, by hypothesis, $h' = \mathrm{true}(\psi)$. Therefore, we have $\psi \models z$. Consider now $\psi' = \big(\phi \wedge \bigwedge \mathrm{hvars}(\sigma)\big)$. Since $\sigma \in \gamma_H(h)$, by Definition 11 we have $h \subseteq \mathrm{hvars}(\sigma)$, so that $\psi' \models \psi$ and thus $\psi' \models z$.

Since $\sigma \in \gamma_F(\phi)$, by Definition 12 we have $\phi\big(\mathrm{hval}(\sigma)\big) = 1$. Also note that $\big(\bigwedge \mathrm{hvars}(\sigma)\big)\big(\mathrm{hval}(\sigma)\big) = 1$. From these, by the definition of conjunction for Boolean formulas, we obtain $\psi'\big(\mathrm{hval}(\sigma)\big) = 1$. Thus we can observe that

$$
\begin{aligned}
\psi'\big(\mathrm{hval}(\sigma)\big) = 1 &\iff (\psi' \wedge z)\big(\mathrm{hval}(\sigma)\big) = 1\\
&\implies z \in \mathrm{hvars}(\sigma).
\end{aligned}
$$

□

Example 2. Consider the following program, where it is assumed that the only "external" query is `?- r(X, Y).`

```
p(X, Y) :- X = f(Y, _).
q(X, Y) :- X = f(_, Y).
r(X, Y) :- p(X, Y), q(X, Y), acyclic_term(X).
```

Consider the call at the predicate `p/2` in the body of the clause defining `r/2`. The predicate `p/2` is called with variables `X` and `Y` both unbound. When computing on the abstract domain $H \times P$, we obtain a finiteness description $h_p \in H$ such that $h_p = \{x, y\}$, expressing the fact that both `X` and `Y` are bound to finite terms. When computing finite-tree dependencies on *Bfun*, the abstract semantics of `p/2` is expressed by the Boolean formula $\phi_p = x \to y$ (`Y` is finite if `X` is so).

Considering now the call to the predicate `q/2`, we note that, since variable `X` is already bound to a non-variable term sharing with `Y`, all the finiteness information encoded by H will be lost (i.e., $h_q = \varnothing$). So, both `X` and `Y` are detected as possibly cyclic. However, the finite-tree dependency information is preserved, because $\phi_q = (x \to y) \wedge (x \to y) = x \to y$.

Finally, consider the effect, on the semantics of `r/2`, of the abstract evaluation of the built-in `acyclic_term(X)`. On the $H \times P$ domain we can only infer that variable `X` cannot be bound to an infinite term, while `Y` will be still considered as possibly cyclic, so that $h_r = \{x\}$. On the domain *Bfun* we can just confirm that the finite-tree dependency computed so far still holds, so that $\phi_r = x \to y$ (no stronger finite-tree

dependency can be inferred, since the finiteness of `X` is only contingent). Thus, by applying the result of Theorem 16, we can recover the finiteness of variable y:

$$h'_r = \mathrm{true}\Big(\phi_r \wedge \bigwedge h_r\Big) = \mathrm{true}\big((x \to y) \wedge x\big) = \{x, y\}.$$

A safe strategy to exploit finite-tree dependencies in the analysis is to apply the reduction step of Theorem 16 immediately after any application of any abstract operator. For instance, this is how predicates like `acyclic_term/1` are handled: the variables of the argument are added to the H component and this is followed by reduction. However, the approach of always performing reduction is unnecessarily inefficient since it is possible to identify cases when Theorem 16 cannot lead to a precision improvement.

THEOREM 17. *Let $x \in VI$, $h, h' \in H$ and $\phi, \phi' \in Bfun$, where $h \supseteq \mathrm{true}\Big(\phi \wedge \bigwedge h\Big)$ and $h' \supseteq \mathrm{true}\Big(\phi' \wedge \bigwedge h'\Big)$. Let also*

$$h_1 \stackrel{\text{def}}{=} h \cap h', \qquad h_2 \stackrel{\text{def}}{=} h \cup \{x\},$$
$$\phi_1 \stackrel{\text{def}}{=} \phi \vee \phi', \qquad \phi_2 \stackrel{\text{def}}{=} \exists x \,.\, \phi.$$

Then, for $i = 1, 2$,

$$h_i \supseteq \mathrm{true}\Big(\phi_i \wedge \bigwedge h_i\Big).$$

PROOF. We assume the hypotheses and prove each statement in turn. For the case where $i = 1$ we have:

$$\begin{aligned}
h_1 &\stackrel{\text{def}}{=} h \cap h' \\
&\supseteq \mathrm{true}\Big(\phi \wedge \bigwedge h\Big) \cap \mathrm{true}\Big(\phi' \wedge \bigwedge h'\Big) \\
&\supseteq \mathrm{true}\Big(\phi \wedge \bigwedge (h \cap h')\Big) \cap \mathrm{true}\Big(\phi' \wedge \bigwedge (h \cap h')\Big) \\
&= \mathrm{true}\Big(\phi \wedge \bigwedge (h \cap h') \vee \phi' \wedge \bigwedge (h \cap h')\Big) \\
&= \mathrm{true}\Big((\phi \vee \phi') \wedge \bigwedge (h \cap h')\Big) \\
&= \mathrm{true}\Big(\phi_1 \wedge \bigwedge h_1\Big).
\end{aligned}$$

For the case where $i = 2$ we have:

$$\begin{aligned}
h_2 &\stackrel{\text{def}}{=} h \cup \{x\} \\
&\supseteq \mathrm{true}\Big(\phi \wedge \bigwedge h\Big) \cup \{x\} \\
&\supseteq \mathrm{true}\Big((\exists x \,.\, \phi) \wedge \bigwedge h\Big) \cup \{x\} \\
&= \mathrm{true}\Big((\exists x \,.\, \phi) \wedge \bigwedge \big(h \cup \{x\}\big)\Big) \\
&= \mathrm{true}\Big(\phi_2 \wedge \bigwedge h_2\Big).
\end{aligned}$$

□

Information encoded in $H \times P$ and *Bfun* is not completely orthogonal and the following result provides a kind of consistency check.

THEOREM 18. *Let $h \in H$ and $\phi \in Bfun$. Then*

$$\gamma_H(h) \cap \gamma_F(\phi) \neq \varnothing \quad \Longrightarrow \quad h \cap \mathrm{false}(\phi) = \varnothing.$$

PROOF. Suppose that there exists $\sigma \in \gamma_H(h) \cap \gamma_F(\phi)$. By Definition 12, since $\sigma \in {\downarrow}\sigma$, we have $\phi\big(\mathrm{hval}(\sigma)\big) = 1$; therefore, we also have

$$\mathrm{hvars}(\sigma) \cap \mathrm{false}(\phi) = \varnothing;$$

by Definition 11, we have $h \subseteq \mathrm{hvars}(\sigma)$, so that we can conclude $h \cap \mathrm{false}(\phi) = \varnothing$. □

Note however that, provided the abstract operators computing on the two components are correct, the computed descriptions will always be mutually consistent, unless $\phi = \bot$.

5. GROUNDNESS DEPENDENCIES

Since information about the groundness of variables is crucial for many applications, it is very natural to consider a static analysis domain including both a finite-tree component and a groundness component. As a matter of fact, any reasonably precise implementation of the parameter component P of the abstract domain specified in [3] will include some kind of groundness information.[7] In this section we will highlight similarities, differences and connections relating the domain *Bfun* for finite-tree dependencies to the well-known abstract domain *Pos* for groundness dependencies. Note however that many of the established results hold also when considering a combination of *Bfun* with the groundness domain *Def* [1].

Definition 14. ($\gamma_G \colon Pos \to \wp(RSubst)$.) The auxiliary function gval: $RSubst \to Bval$ is defined as follows, for each $\sigma \in RSubst$ and each $x \in VI$:

$$\mathrm{gval}(\sigma)(x) = 1 \iff^{\text{def}} x \in \mathrm{gvars}(\sigma).$$

The concretization function $\gamma_G : Pos \to \wp(RSubst)$ is given, for each $\psi \in Pos$, by

$$\gamma_G(\psi) \stackrel{\text{def}}{=} \big\{\, \sigma \in RSubst \;\big|\; \forall \tau \in {\downarrow}\sigma : \psi\big(\mathrm{gval}(\tau)\big) = 1 \,\big\}.$$

As was the case for finite-tree dependencies, groundness dependencies only capture permanent information, therefore preserving the equivalence relation induced by $\mathcal{RT}$. Moreover, the γ_G function is *meet-preserving*.

PROPOSITION 19. *Let $\sigma, \tau \in RSubst$ and $\psi \in Pos$, where $\sigma \in \gamma_G(\psi)$ and $\tau \in {\downarrow}\sigma$. Then $\tau \in \gamma_G(\psi)$.*

PROOF. By the hypothesis, $\tau \in {\downarrow}\sigma$, so that, for each $\upsilon \in {\downarrow}\tau$, $\upsilon \in {\downarrow}\sigma$. Therefore, as $\sigma \in \gamma_G(\phi)$, it follows from Definition 14 that, for all $\upsilon \in {\downarrow}\tau$, $\psi\big(\mathrm{gval}(\upsilon)\big) = 1$ and hence $\tau \in \gamma_G(\phi)$. □

COROLLARY 20. *Let $\sigma, \tau \in RSubst$ and $\psi \in Pos$, where $\sigma \in \gamma_G(\psi)$ and $\mathcal{RT} \vdash \forall(\sigma \leftrightarrow \tau)$. Then $\tau \in \gamma_G(\psi)$.*

LEMMA 21. *Let $\psi_1, \psi_2 \in Pos$. Then*

$$\gamma_G(\psi_1 \wedge \psi_2) = \gamma_G(\psi_1) \cap \gamma_G(\psi_2).$$

[7] One could define P so as it explicitly contains the abstract domain *Pos*. Even when this is not the case, it should be noted that, as soon as the parameter P includes the set-sharing domain of Jacobs and Langen [28], then it will subsume the groundness information captured by the domain *Def* [9, 14].

PROOF.

$$
\begin{aligned}
&\gamma_G(\psi_1 \wedge \psi_2)\\
&= \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : (\psi_1 \wedge \psi_2)\big(\mathrm{gval}(\tau)\big) = 1 \,\big\}\\
&= \left\{ \sigma \in \mathit{RSubst} \;\middle|\; \begin{array}{c} \forall \tau \in {\downarrow}\sigma : \forall i \in \{1,2\} : \\ \psi_i\big(\mathrm{gval}(\tau)\big) = 1 \end{array} \right\}\\
&= \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : \psi_1\big(\mathrm{gval}(\tau)\big) = 1 \,\big\}\\
&\quad \cap \big\{\, \sigma \in \mathit{RSubst} \;\big|\; \forall \tau \in {\downarrow}\sigma : \psi_2\big(\mathrm{gval}(\tau)\big) = 1 \,\big\}\\
&= \gamma_G(\psi_1) \cap \gamma_G(\psi_2).
\end{aligned}
$$

□

The following is a simple variant of the standard abstract unification operator defined for groundness analysis over finite-tree domains: the only difference concerns the case of cyclic bindings [2].

Definition 15. The function $\mathrm{amgu}_G\colon \mathit{Pos} \times \mathit{Bind} \to \mathit{Pos}$ captures the effects of a binding on a groundness dependency component. Let $\psi \in \mathit{Pos}$ and $(x \mapsto t) \in \mathit{Bind}$. Then

$$\mathrm{amgu}_G(\psi, x \mapsto t) \stackrel{\text{def}}{=} \psi \wedge \Big(x \leftrightarrow \bigwedge\big(\mathrm{vars}(t) \setminus \{x\}\big)\Big).$$

Since non-ground terms can be made cyclic by instantiating their variables, those terms detected as definitely finite on *Bfun* are also definitely ground.

LEMMA 22. *Let $x \in \mathit{VI}$. Then $\gamma_F(x) \subseteq \gamma_G(x)$.*

PROOF. Suppose that $\sigma \in \gamma_F(x)$. Then, by Definition 12, $(x)\big(\mathrm{hval}(\tau)\big) = 1$ for all $\tau \in {\downarrow}\sigma$, so that $x \in \mathrm{hvars}(\tau)$; in particular, $x \in \mathrm{hvars}(\sigma)$. We prove $x \in \mathrm{gvars}(\sigma)$ by contradiction. That is, we show that if $x \in \mathrm{hvars}(\sigma) \setminus \mathrm{gvars}(\sigma)$, then there exists $\tau \in {\downarrow}\sigma$ for which $x \notin \mathrm{hvars}(\tau)$.

Suppose therefore that $x \in \mathrm{hvars}(\sigma) \setminus \mathrm{gvars}(\sigma)$. Then, by Proposition 7, $\mathrm{rt}(x, \sigma) \in \mathit{HTerms} \setminus \mathit{GTerms}$. Hence, by Proposition 1, there exists $i \in \mathbb{N}$ such that $\mathrm{rt}(x, \sigma) = x\sigma^i$ and there exists $y \in \mathrm{vars}(x\sigma^i) \setminus \mathrm{dom}(\sigma)$. As we assumed that *Sig* contains a function symbol of non-zero arity, there exists $t \in \mathit{HTerms} \setminus \{y\}$ for which $\{y\} = \mathrm{vars}(t)$. It follows that $\sigma' = \{y \mapsto t\} \in \mathit{RSubst}$ and, by Definition 10, $y \notin \mathrm{hvars}(\sigma')$. Since $y \notin \mathrm{dom}(\sigma)$, by Lemma 3, $\tau = \sigma \cup \sigma' \in \mathit{RSubst}$. Since $\tau \in {\downarrow}\sigma'$ then, by case (9a) of Proposition 9, we have $y \notin \mathrm{hvars}(\tau)$.

By Lemma 4, $\mathcal{RT} \vdash \forall\big(\sigma \to (x = x\sigma^i)\big)$. Thus, since we also have $\tau \in {\downarrow}\sigma$, we obtain $\mathcal{RT} \vdash \forall\big(\tau \to (x = x\sigma^i)\big)$. By applying Lemma 6, we have $\mathrm{rt}(x, \tau) = \mathrm{rt}(x\sigma^i, \tau)$ so that, by case (8b) of Corollary 8, we obtain $x \in \mathrm{hvars}(\tau)$ if and only if $\mathrm{vars}(x\sigma^i) \subseteq \mathrm{hvars}(\tau)$. However, as observed before, $y \in \mathrm{vars}(x\sigma^i) \setminus \mathrm{hvars}(\tau)$, so that we also have $x \notin \mathrm{hvars}(\tau)$.

Therefore $x \in \mathrm{gvars}(\sigma) \cap \mathrm{hvars}(\sigma)$ and, by case (9b) of Proposition 9, for all $\tau \in {\downarrow}\sigma$, $x \in \mathrm{gvars}(\tau) \cap \mathrm{hvars}(\tau)$. As a consequence, for all $\tau \in {\downarrow}\sigma$, $(x)\big(\mathrm{gval}(\tau)\big) = 1$, so that, by Definition 14, we can conclude that $\sigma \in \gamma_G(x)$. □

The following theorem specifies a reduction process that can improve the precision of the groundness dependencies (*Pos*) component by exploiting the information available in the finite-tree dependencies (*Bfun*) component.

THEOREM 23. *Let $\phi \in \mathit{Bfun}$ and $\psi \in \mathit{Pos}$. Let $\nu \in \mathit{Pos}$ be defined as $\nu = \bigwedge \mathrm{true}(\phi)$. Then*

$$\gamma_F(\phi) \cap \gamma_G(\psi) = \gamma_F(\phi) \cap \gamma_G(\psi \wedge \nu).$$

PROOF. Since $\psi \wedge \nu \models \psi$, the inclusion

$$\gamma_F(\phi) \cap \gamma_G(\psi) \supseteq \gamma_F(\phi) \cap \gamma_G(\psi \wedge \nu)$$

follows by the monotonicity of γ_G. To prove the inclusion

$$\gamma_F(\phi) \cap \gamma_G(\psi) \subseteq \gamma_F(\phi) \cap \gamma_G(\psi \wedge \nu)$$

we will show that $\gamma_F(\phi) \subseteq \gamma_G(\nu)$. The thesis will follow as, by Lemma 21, $\gamma_G(\psi \wedge \nu) = \gamma_G(\psi) \cap \gamma_G(\nu)$. We have

$$
\begin{aligned}
\gamma_F(\phi) &\subseteq \gamma_F(\nu) && [\text{since } \phi \models \nu]\\
&= \bigcap \big\{\, \gamma_F(x) \;\big|\; x \in \mathrm{true}(\phi) \,\big\} && [\text{by Lemma 14}]\\
&\subseteq \bigcap \big\{\, \gamma_G(x) \;\big|\; x \in \mathrm{true}(\phi) \,\big\} && [\text{by Lemma 22}]\\
&= \gamma_G(\nu). && [\text{by Lemma 21}]
\end{aligned}
$$

□

The following two examples shows that, when computing on rational trees, finite-tree dependencies may provide groundness information that is not captured by the usual approaches.

Example 3. Consider the following program:

```
p(a, a).
p(X, Y) :- X = f(X, _).
q(X, Y) :- p(X, Y), X = a.
```

Consider first the predicate `p/2`. Concerning finite-tree dependencies, the abstract semantics of `p/2` is expressed by the Boolean formula $\phi_p = x \to y$ (y is finite if x is so). In contrast, the *Pos*-groundness abstract semantics of `p/2` is a plain "don't know": the Boolean formula $\psi_p = \top$. In fact, the groundness of `X` and `Y` can be completely decided by the call-pattern of `p/2`.

Consider now the predicate `q/2`. The finiteness semantics of `q/2` is $\phi_q = (x \to y) \wedge x = x \wedge y$, whereas the *Pos* formula expressing groundness dependencies is $\psi_q = \top \wedge x = x$. By applying the reduction process of Theorem 23, we obtain

$$\psi'_q = \psi_q \wedge \bigwedge \mathrm{true}(\phi_q) = x \wedge y,$$

therefore recovering the groundness of variable y.

Example 4. Consider the program:

```
p(a, Y).
p(X, a).
q(X, Y) :- p(X, Y), X = f(X, Z).
```

The abstract semantics of predicate `p/2`, for both finite-tree and groundness dependencies, is $\phi_p = \psi_p = x \vee y$.

Consider now the predicate `q/2`. Concerning finite-tree dependencies, the abstract semantics of `q/2` is expressed by the Boolean formula $\phi_q = (x \vee y) \wedge \neg x = \neg x \wedge y$ (x is definitely cyclic and y is definitely finite). On the domain *Pos*, by applying Definition 15, we compute

$$
\begin{aligned}
\psi_q &= \exists z \,.\, \big((x \vee y) \wedge (x \leftrightarrow z)\big)\\
&= x \vee y.
\end{aligned}
$$

Thus we can apply Theorem 23 and improve the groundness dependencies description by computing

$$\psi'_q = \psi_q \wedge \bigwedge \mathrm{true}(\phi_q) = y.$$

Since groundness information, besides being useful in itself, also has the potential of improving the precision of many other analyses, such as sharing information [6, 9], by applying the reduction step of Theorem 23 we can indeed trigger some reduction processes affecting the precision of other components. Theorem 23 can also be exploited in the application of a widening operator on the groundness dependencies component. However, as was the case for Theorem 16, there are cases when Theorem 23 cannot provide a precision gain.

THEOREM 24. *Let $\phi, \phi' \in$ Bfun and $\psi, \psi' \in$ Pos, where $\psi \models \bigwedge \mathrm{true}(\phi)$ and $\psi' \models \bigwedge \mathrm{true}(\phi')$. Let also*

$$\phi_1 \stackrel{\mathrm{def}}{=} \phi \vee \phi', \qquad \phi_2 \stackrel{\mathrm{def}}{=} \exists x \,.\, \phi,$$
$$\psi_1 \stackrel{\mathrm{def}}{=} \psi \vee \psi', \qquad \psi_2 \stackrel{\mathrm{def}}{=} \exists x \,.\, \psi.$$

Then, for $i = 1, 2$,

$$\psi_i \models \bigwedge \mathrm{true}(\phi_i).$$

PROOF. Let us assume the hypotheses hold and prove each statement in turn. For the case where $i = 1$ we have:

$$\begin{aligned}\psi_1 &\stackrel{\mathrm{def}}{=} \psi \vee \psi' \\ &\models \bigwedge \mathrm{true}(\phi) \vee \bigwedge \mathrm{true}(\phi') \\ &\models \bigwedge \mathrm{true}(\phi \vee \phi') \\ &\stackrel{\mathrm{def}}{=} \bigwedge \mathrm{true}(\phi_1).\end{aligned}$$

Since by hypothesis we have that $\psi \models \bigwedge \mathrm{true}(\phi)$ and existential quantification is a monotonic operation, for the case where $i = 2$ we have:

$$\begin{aligned}\psi_2 &\stackrel{\mathrm{def}}{=} \exists x \,.\, \psi \\ &\models \exists x \,.\, \bigwedge \mathrm{true}(\phi) \\ &= \bigwedge \big(\mathrm{true}(\phi) \setminus \{x\}\big) \\ &= \bigwedge \mathrm{true}(\exists x \,.\, \phi) \\ &\stackrel{\mathrm{def}}{=} \bigwedge \mathrm{true}(\phi_2).\end{aligned}$$

□

Another interesting question is whether or not we can define a reduction process working the other way round, i.e., trying to improve the finite-tree dependencies component based on the information encoded in the groundness dependencies component. The following theorem shows that this is theoretically possible, provided the information of the finiteness component H is also available.

THEOREM 25. *Let $h \in H$, $\phi \in$ Bfun and $\psi \in$ Pos. Let $\nu \in$ Bfun be defined as $\nu = (\exists\, VI \setminus h \,.\, \psi)$. Then*

$$\gamma_H(h) \cap \gamma_F(\phi) \cap \gamma_G(\psi) = \gamma_H(h) \cap \gamma_F(\phi \wedge \nu) \cap \gamma_G(\psi).$$

PROOF. Since $\phi \wedge \nu \models \phi$, the inclusion

$$\gamma_H(h) \cap \gamma_F(\phi) \cap \gamma_G(\psi) \supseteq \gamma_H(h) \cap \gamma_F(\phi \wedge \nu) \cap \gamma_G(\psi)$$

follows by the monotonicity of γ_G.

We now prove the reverse inclusion. Let us suppose that $\sigma \in \gamma_H(h) \cap \gamma_F(\phi) \cap \gamma_G(\psi)$. By Lemma 14 we have that $\gamma_F(\phi \wedge \nu) = \gamma_F(\phi) \cap \gamma_F(\nu)$. Therefore it is enough to show that $\sigma \in \gamma_F(\nu)$. By hypothesis, $\nu = \exists\, VI \setminus h \,.\, \psi$. Moreover, by Definition 11, $h \subseteq \mathrm{hvars}(\sigma)$. Thus, to prove the result, we will show, by contradiction, that $\sigma \in \gamma_F\big(\exists\, VI \setminus \mathrm{hvars}(\sigma) \,.\, \psi\big)$.

Suppose therefore that $\sigma \notin \gamma_F\big(\exists\, VI \setminus \mathrm{hvars}(\sigma) \,.\, \psi\big)$. Then there exists $\tau \in {\downarrow}\sigma$ such that

$$\big(\exists\, VI \setminus \mathrm{hvars}(\sigma) \,.\, \psi\big)\big(\mathrm{hval}(\tau)\big) = 0. \tag{15}$$

Take $t \in \mathit{GTerms} \cap \mathit{HTerms}$ and let

$$\upsilon \stackrel{\mathrm{def}}{=} \left\{ y \mapsto t \;\middle|\; y \in \mathrm{vars}(\sigma) \cap \big(\mathrm{hvars}(\tau) \setminus \mathrm{dom}(\sigma)\big) \right\}. \tag{16}$$

By Lemma 3, $\tau' \stackrel{\mathrm{def}}{=} \sigma \cup \upsilon \in \mathit{RSubst}$ is satisfiable in $\mathcal{RT}$.

Let z be any variable in $\mathrm{hvars}(\sigma)$. By case (7b) of Proposition 7, $\mathrm{rt}(z, \sigma) \in \mathit{HTerms}$. By Proposition 1, there exists $i \in \mathbb{N}$ such that $\mathrm{rt}(z, \sigma) = z\sigma^i$ and $\mathrm{vars}(z\sigma^i) \cap \mathrm{dom}(\sigma) = \varnothing$. Therefore, by Definition 10, $\mathrm{vars}(z\sigma^i) \subseteq \mathrm{hvars}(\sigma)$. Thus, we have

$$\mathrm{vars}(z\sigma^i) \subseteq \mathrm{hvars}(\sigma) \setminus \mathrm{dom}(\sigma). \tag{17}$$

By Lemma 4, as $\tau \in {\downarrow}\sigma$, $\mathcal{RT} \vdash \forall\big(\tau \to (z = z\sigma^i)\big)$. By Lemma 6, we have $\mathrm{rt}(z, \tau) = \mathrm{rt}(z\sigma^i, \tau)$ so that, by case (8b) of Corollary 8,

$$z \in \mathrm{hvars}(\tau) \iff \mathrm{vars}(z\sigma^i) \subseteq \mathrm{hvars}(\tau). \tag{18}$$

We now show that

$$\mathrm{hvars}(\tau) = \mathrm{hvars}(\sigma) \cap \mathrm{gvars}(\tau'). \tag{19}$$

Since $\tau \in {\downarrow}\sigma$, it follows from case (9a) of Proposition 9 that $\mathrm{hvars}(\tau) \subseteq \mathrm{hvars}(\sigma)$. Thus, as $z \in \mathrm{hvars}(\sigma)$, either $z \in \mathrm{hvars}(\tau)$ or $z \in \mathrm{hvars}(\sigma) \setminus \mathrm{hvars}(\tau)$. We consider these cases separately.

First, assume that $z \in \mathrm{hvars}(\tau)$. Then, by (18), we have $\mathrm{vars}(z\sigma^i) \subseteq \mathrm{hvars}(\tau)$. Also, by case (9a) of Proposition 9, we have $z \in \mathrm{hvars}(\sigma)$, so that we can apply (17) to derive $\mathrm{vars}(z\sigma^i) \cap \mathrm{dom}(\sigma) = \varnothing$. Therefore, $\mathrm{vars}(z\sigma^i) \subseteq \mathrm{dom}(\upsilon)$ and, by Definitions 8 and 10, $\mathrm{vars}(z\sigma^i) \subseteq \mathrm{gvars}(\upsilon) \cap \mathrm{hvars}(\upsilon)$. Since $\tau' \in {\downarrow}\upsilon$, by case (9b) of Proposition 9, we have $\mathrm{vars}(z\sigma^i) \subseteq \mathrm{gvars}(\tau') \cap \mathrm{hvars}(\tau')$. Thus, by Corollary 8, $\mathrm{rt}(z\sigma^i, \tau') \in \mathit{GTerms} \cap \mathit{HTerms}$. Now $\tau' \in {\downarrow}\sigma$ so that, by Lemma 4, $\mathcal{RT} \vdash \forall\big(\tau' \to (z = z\sigma^i)\big)$. Thus, by Lemma 6, $\mathrm{rt}(z\sigma^i, \tau') = \mathrm{rt}(z, \tau') \in \mathit{GTerms} \cap \mathit{HTerms}$ so that, by Proposition 7, $z \in \mathrm{hvars}(\tau') \cap \mathrm{gvars}(\tau')$. Hence, by case (9a) of Proposition 9, we can conclude $z \in \mathrm{hvars}(\sigma) \cap \mathrm{gvars}(\tau')$. Thus $\mathrm{hvars}(\tau) \subseteq \mathrm{hvars}(\sigma) \cap \mathrm{gvars}(\tau')$.

Secondly, we assume that $z \in \mathrm{hvars}(\sigma) \setminus \mathrm{hvars}(\tau)$. Since $z \notin \mathrm{hvars}(\tau)$, by (18), there exists $y \in \mathrm{vars}(z\sigma^i) \setminus \mathrm{hvars}(\tau)$. Also, since $z \in \mathrm{hvars}(\sigma)$, by (17), $y \in \mathrm{hvars}(\sigma) \setminus \mathrm{dom}(\sigma)$ so that, by Definition 8, we have $y \notin \mathrm{gvars}(\sigma)$. By (16), since $y \notin \mathrm{dom}(\sigma) \cup \mathrm{hvars}(\tau)$, we have $y \notin \mathrm{dom}(\upsilon)$ so that $y \notin \mathrm{gvars}(\tau')$. Thus, by case (8a) of Corollary 8, we have $\mathrm{rt}(z\sigma^i, \tau') \notin \mathit{GTerms}$. Also, as $\mathcal{RT} \vdash \forall\big(\tau' \to (z = z\sigma^i)\big)$, by Lemma 6 we have $\mathrm{rt}(z\sigma^i, \tau') = \mathrm{rt}(z, \tau') \notin \mathit{GTerms}$ and therefore, by case (7a) of Proposition 7, $z \notin \mathrm{gvars}(\tau')$. Thus $\mathrm{hvars}(\tau) \supseteq \mathrm{hvars}(\sigma) \cap \mathrm{gvars}(\tau')$.

It follows from (15) and (19) that,

$$\big(\exists\, VI \setminus \mathrm{hvars}(\sigma) \,.\, \psi\big)\big(\mathrm{gval}(\tau')\big) = 0. \tag{20}$$

It also holds by hypothesis that $\sigma \in \gamma_G(\psi)$, so that, since $\tau' \in \downarrow\sigma$, by Definition 14 we have $\psi\big(\mathrm{gval}(\tau')\big) = 1$. Therefore, as $\psi \models \exists\, VI \setminus \mathrm{hvars}(\sigma)\ .\ \psi$,

$$\big(\exists\, VI \setminus \mathrm{hvars}(\sigma)\ .\ \psi\big)\big(\mathrm{gval}(\tau')\big) = 1,$$

which contradicts (20). □

As was the case for Theorems 16 and 23, when abstractly evaluating the existential quantification and the merge-over-all-paths operations, the application of the reduction process of Theorem 25 is unnecessary.

THEOREM 26. *Let $h, h' \in H$, $\phi, \phi' \in Bfun$, $\psi, \psi' \in Pos$, where $\phi \models (\exists\, VI \setminus h\ .\ \psi)$ and $\phi' \models (\exists\, VI \setminus h'\ .\ \psi')$. Let also*

$$h_1 \stackrel{\text{def}}{=} h \cap h', \qquad h_2 \stackrel{\text{def}}{=} h \cup \{x\},$$
$$\phi_1 \stackrel{\text{def}}{=} \phi \vee \phi', \qquad \phi_2 \stackrel{\text{def}}{=} \exists x\ .\ \phi,$$
$$\psi_1 \stackrel{\text{def}}{=} \psi \vee \psi', \qquad \psi_2 \stackrel{\text{def}}{=} \exists x\ .\ \psi.$$

Then, for $i = 1, 2$,

$$\phi_i \models (\exists\, VI \setminus h_i\ .\ \psi_i).$$

PROOF. Let us assume the hypotheses. For the case where $i = 1$ we have:

$$\begin{aligned}
\phi_1 &\stackrel{\text{def}}{=} \phi \vee \phi' \\
&\models (\exists\, VI \setminus h\ .\ \psi) \vee (\exists\, VI \setminus h'\ .\ \psi') \\
&\models \big(\exists\, VI \setminus (h \cap h')\ .\ \psi\big) \vee \big(\exists\, VI \setminus (h \cap h')\ .\ \psi'\big) \\
&= \exists\, VI \setminus (h \cap h')\ .\ \psi \vee \psi' \\
&\stackrel{\text{def}}{=} \exists\, VI \setminus h_1\ .\ \psi_1.
\end{aligned}$$

Since by hypothesis we have that $\phi \models \exists\, VI \setminus h\ .\ \psi$ and existential quantification is a monotonic operation, for the case where $i = 2$ we have:

$$\begin{aligned}
\phi_2 &\stackrel{\text{def}}{=} \exists x\ .\ \phi \\
&\models \exists x\ .\ \exists\, VI \setminus h\ .\ \psi \\
&= \exists\, VI \setminus h\ .\ \exists x\ .\ \psi \\
&= \exists\, VI \setminus \big(h \cup \{x\}\big)\ .\ \exists x\ .\ \psi \\
&\stackrel{\text{def}}{=} \exists\, VI \setminus h_2\ .\ \psi_2.
\end{aligned}$$

□

We conjecture that Theorem 26 can be strengthened so that it also applies to the case of abstract unification. However, when precision is lost due to the use of a widening operator on the *Bfun* component, some of this could be recovered by means of the reduction process given by Theorem 25.

6. CONCLUSION

Several modern logic-based languages offer a computation domain based on rational trees. On the one hand, the use of such trees is encouraged by the possibility of using efficient and correct unification algorithms and by an increase in expressivity. On the other hand, these gains are countered by the extra problems rational trees bring with themselves and that can be summarized as follows: several built-ins, library predicates, program analysis and manipulation techniques are only well-defined for program fragments working with finite trees. For these reasons, applications exploiting rational trees tend to do so in a very controlled way, that is, most program variables can only be bound to finite terms. If we are able to detect with enough precision the program variables that may be bound to infinite terms, then we can take advantage of rational trees yet minimizing the impact of their disadvantages.

In a companion paper [3] we have proposed an initial solution to the problem, where the composite abstract domain $H \times P$ allows to track, with significant precision, the creation and propagation of infinite terms. Even though this information is crucial to any finite-tree analysis, propagating the guarantees of finiteness that come from several built-ins (including those that are explicitly provided to test term-finiteness) is also important. This is what motivates the work described in this paper, where we have introduced a domain of Boolean functions that is meant to be coupled to $H \times P$, supplementing its expressive power.

In this paper we have studied all the theoretical issues concerning this Boolean domain for finite-tree dependencies. Since this domain has many similarities with the domain *Pos* used for groundness analysis, we have investigated how these two domains relate to each other and what are the possibilities when both of them are included in the "global" domain of analysis. The work described here and in [3] allowed to complete the preparation of the theoretical stage that is required before proceeding to the implementation of the overall finite-tree analysis domain. The implementation of the domains is now in progress. Concerning the *Bfun* component things are quite easy, since all the techniques described in [5] (and almost all the code, including the widenings) can be reused unchanged, obtaining, we believe, comparable efficiency results.

7. REFERENCES

[1] 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.

[2] 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.

[3] R. Bagnara, R. Gori, P. M. Hill, and E. Zaffanella. Finite-tree analysis for constraint logic-based languages. Quaderno 251, Dipartimento di Matematica, Università di Parma, 2001. Available at `http://www.cs.unipr.it/~bagnara/`.

[4] R. Bagnara, P. M. Hill, and E. Zaffanella. Efficient structural information analysis for real CLP languages. In M. Parigot and A. Voronkov, editors, *Proc. of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 189–206, Reunion Island, France, 2000. Springer-Verlag, Berlin.

[5] R. Bagnara and P. Schachte. Factorizing equivalent variable pairs in ROBDD-based implementations of *Pos*. In A. M. Haeberer, editor, *Proc. 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.

[6] R. Bagnara, E. Zaffanella, and P. M. Hill. Enhanced sharing analysis techniques: A comprehensive evaluation. In M. Gabbrielli and F. Pfenning, editors, *Proc. of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 103–114, Montreal, Canada, 2000. Association for Computing Machinery.

[7] J. A. Campbell, editor. *Implementations of Prolog*. Ellis Horwood/Halsted Press/Wiley, 1984.

[8] B. Carpenter. *The Logic of Typed Feature Structures with Applications to Unification-based Grammars, Logic Programming and Constraint Resolution*, volume 32 of *Cambridge Tracts in Theoretical Computer Science*. CUP, New York, 1992.

[9] 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.

[10] 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. Ac. Press, New York, 1982.

[11] A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proc. of the International Conference on Fifth Generation Computer Systems (FGCS'84)*, pages 85–99, Tokyo, Japan, 1984. ICOT.

[12] A. Colmerauer. An introduction to Prolog-III. *Communications of the ACM*, 33(7):69–90, 1990.

[13] A. Cortesi, G. Filé, and W. Winsborough. *Prop* revisited: Propositional formula as abstract domain for groundness analysis. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 322–327, Amsterdam, The Netherlands, 1991. IEEE Computer Society Press.

[14] 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.

[15] 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), 2000.

[16] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[17] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, 1979.

[18] P. Cousot and R. Cousot. Abstract interpretation and applications to logic programs. *Journal of Logic Programming*, 13(2&3):103–179, 1992.

[19] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[20] P. W. Dart. On derived dependencies and connected databases. *Journal of Logic Programming*, 11(1&2):163–188, 1991.

[21] P. R. Eggert and K. P. Chow. Logic programming, graphics and infinite terms. Technical Report UCSB DoCS TR 83-02, Department of Computer Science, University of California at Santa Barbara, 1983.

[22] G. Erbach. ProFIT: Prolog with Features, Inheritance and Templates. In *Proc. of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, pages 180–187, Dublin, Ireland, 1995.

[23] M. Filgueiras. A Prolog interpreter working with infinite terms. In Campbell [7], pages 250–258.

[24] F. Giannesini and J. Cohen. Parser generation and grammar manipulation using Prolog's infinite trees. *Journal of Logic Programming*, 3:253–265, 1984.

[25] S. Haridi and D. Sahlin. Efficient implementation of unification of cyclic structures. In Campbell [7], pages 234–249.

[26] P. M. Hill, R. Bagnara, and E. Zaffanella. Soundness, idempotence and commutativity of set-sharing. *Theory and Practice of Logic Programming*, 2001. To appear. Available at `http://arXiv.org/abs/cs.PL/0102030`.

[27] B. Intrigila and M. Venturini Zilli. A remark on infinite matching vs infinite unification. *Journal of Symbolic Computation*, 21(3):2289–2292, 1996.

[28] D. Jacobs and A. Langen. Static analysis of logic programs for independent AND parallelism. *Journal of Logic Programming*, 13(2&3):291–314, 1992.

[29] 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.

[30] T. Keisu. *Tree Constraints*. PhD thesis, The Royal Institute of Technology, Stockholm, Sweden, May 1994. SICS Dissertation Series: SICS/D–16–SE.

[31] A. King. Pair-sharing over rational trees. *Journal of Logic Programming*, 46(1–2):139–155, 2000.

[32] 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.

[33] K. Marriott and H. Søndergaard. Notes for a tutorial on abstract interpretation of logic programs. North American Conference on Logic Programming, Cleveland, Ohio, USA, 1989.

[34] K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM Letters on Programming Languages and Systems*, 2(1–4):181–196, 1993.

[35] K. Mukai. *Constraint Logic Programming and the Unification of Information*. PhD thesis, Department of Computer Science, Faculty of Engineering, Tokio Institute of Technology, 1991.

[36] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, 1994.

[37] E. Schröder. *Der Operationskreis des Logikkalkuls*. B. G. Teubner, Leibzig, 1877.

[38] Gert Smolka and Ralf Treinen. Records for logic programming. *Journal of Logic Programming*, 18(3):229–258, 1994.

[39] H. Søndergaard. An application of abstract interpretation of logic programs: Occur check reduction. In B. Robinet and R. Wilhelm, editors, *Proc. of the 1986 European Symposium on Programming*, volume 213 of *Lecture Notes in Computer Science*, pages 327–338. Springer-Verlag, Berlin, 1986.

[40] Swedish Institute of Computer Science, Programming Systems Group. *SICStus Prolog User's Manual*, release 3 #0 edition, 1995.