Data-Flow Analysis for Constraint Logic-Based Languages

Roberto Bagnara

Dipartimento di Informatica
Università di Pisa
Corso Italia, 40
56125 Pisa

Dottorato di Ricerca in Informatica
Pisa – Genova – Udine

VIII ciclo

A Franco

I believe in things that are developed through hard work.
I always like people who have developed long and hard,
especially through introspection and a lot of dedication.
I think what they arrive at is usually a much deeper
and more beautiful thing
than the person who seems to have this ability and fluidity
from the beginning.
— BILL EVANS, from *Contemporary Keyboard* (1981)

Acknowledgments

Firstly I wish to thank my friend and advisor Giorgio Levi for his encouragement and guidance. Giorgio supported me in all possible ways and helped me to overcome the difficult periods.

Thanks (for various reasons) to my former colleagues at the Department of Physics of the University of Bologna: Antonio Bassi, Dante Bollini, Luigi degli Esposti, Pierluigi Frabetti, Mauro Lolli, Patrizia Nicoli, and Umberto Placci.

Thanks to all my former colleagues at CERN for their friendship and for providing an environment where I could learn so much: Tim Berners-Lee, Andre Bogaerts, Roberto Divià, Philippe Gavillet, Giorgio Heiman, Hans Müller, Chris Parkman, Antonio Pastore, Yves Perrin, Jørgen Petersen, Achille Petrilli, Louis Tremblet, Bruce Wessels, and Pietro Zanarini.

Thanks to all my colleagues and friends at the Dipartimento di Informatica of the University of Pisa. In particular, thanks to Bruno Bacci, Piero Bonatti, Alessio Bragadini, Marco Comini, Simone Contiero, Marco Danelutto, Pierpaolo Degano, Andrea Dell'Amico, Agostino Dovier, Moreno Falaschi, Paolo Ferragina, Tito Flagella, Luca Francesconi, Maurizio Gabbrielli, Roberto Giacobazzi, Francesca Levi, Paolo Mancarella, Andrea Masini, Dino Pedreschi, Susanna Pelagatti, Corrado Priami, Paola Quaglia, Alessandra Raffaetà, Francesca Rossi, Francesca Scozzari, Laura Semini, Alessandro Sperduti, Stefano Suin, Franco Turini, Paolo Volpe, and Enea Zaffanella. Special thanks are due to Enea for being a really nice person and a good friend of mine, for all the work and the discussions done together, and for helping me with some technical issues. Thanks to Maria Cristina Scudellari for the work we have done together on the material that is included in Chapter 7.

Thanks to Kim Marriott for inviting me to visit the Monash University of Melbourne. The month I have spent there was of great inspiration.

I have been fortunate to meet some exceptionally good teachers: Dino Pieri, Attilio Forino, Giorgio Levi, Fabrizio Luccio, Marco Vanneschi, and Simone Martini. To them goes my deep gratitude.

I wish to thank Maurizio Gabbrielli, Roberto Giacobazzi, Patricia M. Hill, Giorgio Levi, Catiuscia Palamidessi, Francesca Rossi, Vijay Saraswat, and Enea Zaffanella for reading draft versions of this thesis and for providing many useful comments. I am particularly grateful to the external reviewers of this thesis, Andrew M. King and William H. Winsborough, for their care and competence. Andy and Will gave me a lot of useful suggestions. Thanks also to Tony Cohn and Pat Hill for allowing me to finish the thesis with relative peace of mind.

Last but most important, I thank my parents for their support and encouragement. I am especially grateful to my brother Abramo with whom I have shared these eighteen years of computer science. In particular, the help he gave me in the development of CHINA is invaluable.

Thanks to Rossana for being with me, for her encouragement, and for putting up with me during these last months when I was working day and night on the thesis.

Ringraziamenti[1]

Devo molto a molte persone, come tutti. Qui desidero ringraziare tutte quelle persone che, in qualche modo, hanno contribuito a che questa tesi fosse scritta. Farò questo in modo approssimativamente cronologico.

Innanzitutto voglio ringraziare mio padre, di un'infinità di cose, naturalmente, ma in particolare per avermi trasmesso il senso della qualità. Ringrazio mia madre, per avermi sostenuto nei tanti momenti difficili.

Il mio rapporto con l'informatica è iniziato circa 18 anni fa ed è costellato di persone che mi hanno aiutato e che ricordo con affetto. Dunque si torna indietro all'ITIS di Cesena, anno 1979. Grazie a Pierluigi Mengolini che mi ha introdotto all'uso delle calcolatrici programmabili e al microprocessore Zilog Z80. Grazie a Roberto Giunchi che mi consentì di costruire il mio primo microcomputer invece di un televisore in bianco e nero come usava a quei tempi in quella scuola; grazie a mio padre che finanziò il progetto e grazie a mio fratello Abramo che partecipò con foga selvaggia.

Fuori dalla scuola, grazie a Franco Dapporto per avermi fatto usare la sua TI58, per le innumerevoli discussioni sull'informatica, e per le ore passate a "sbilinare" con un programma per gli scacchi scritto in Fortran. Grazie a Valeriano Ballardini per tutto quello che mi ha insegnato da quando ci siamo conosciuti.

Dipartimento di fisica dell'università di Bologna, 1984–1987. Grazie a Dante Bollini per la libertà con cui mi consentì di svolgere il mio lavoro, il che mi permise di sperimentare nuove tecniche e di imparare moltissime cose oltre a quelle che mi insegnò lui stesso (ivi incluso il "*bootstrap* da pannello" dell'HP1000!). Grazie ai miei colleghi Antonio Bassi, Luigi degli Esposti, Mauro Lolli, Patrizia Nicoli e Umberto Placci, per tutte le discussioni istruttive che abbiamo avuto e, soprattutto, per la loro amicizia. Grazie ad Antonio per il suo entusiasmo e per la sua disponibilità. Sono particolarmente grato a Mauro per tutto quello che mi ha insegnato, per

[1] Questa sezione di ringraziamenti è insolitamente (per qualcuno, forse, intollerabilmente) lunga. Ho ritenuto che, dopo tanto lavoro, mi fosse consentito esprimere con una certa libertà la mia gratitudine alle persone che hanno avuto un influsso, talvolta determinante, sulla mia "storia informatica" e dunque su questa tesi. Con questa nota desidero rassicurare il lettore interessato ai soli contenuti scientifici sul fatto che egli può ignorare questa sezione senza alcun pregiudizio per la comprensione del seguito.

tutte le nostre discussioni sulla contraddizione tra qualità e "tempi di consegna", e per non aver mai rinunciato al confronto. Grazie anche a Pierluigi Frabetti per il suo incoraggiamento.

CERN, laboratorio europeo per la fisica delle particelle, 1985 e 1988–1989. Grazie a Philippe Gavillet, che nel 1985 mi introdusse al problema e alle tecniche di diagnosi remota e automatica dei guasti hardware e software. Grazie ai miei colleghi Tim Berners-Lee, Andre Bogaerts, Roberto Divià, Hans Müller, Chris Parkman, Antonio Pastore, Yves Perrin, Jørgen Petersen, Louis Tremblet e Bruce Wessels per la loro amicizia e per aver costituito un gruppo di lavoro che per me è stato una vera palestra. Grazie a Louis, con il quale ho svolto la prima parte del mio lavoro al CERN, per tutte le ore passate in discussioni interessanti ed istruttive. Grazie a Tim per tutto quello che mi ha insegnato (protocolli di rete, sistemi di *Remote Procedure Call*, compilatori, interfacce utente, eccetera) e per aver involontariamente determinato la svolta che mi ha riportato all'università: come studente di informatica, questa volta. Sempre al CERN, sebbene al di fuori del lavoro ufficiale, ho avuto la fortuna di conoscere Giorgio Heiman, Achille Petrilli e Pietro Zanarini: discutendo con loro ho avuto modo di apprendere molte cose utili. Pietro, in particolare, mi ha insegnato che buona parte dei problemi dell'informatica si risolve inventando un linguaggio al giusto livello di astrazione e scrivendo un compilatore o un interprete per detto linguaggio. Prima di arrivare al CERN passai qualche mese studiando il sistema MoniCa (MONItor CAmac), scritto in linguaggio macchina MC680x0 da Horst Von Heicken, allora al CERN. Sono enormemente grato a Horst per tutto quello che ho imparato in questo modo: mi è capitato raramente di vedere codice di così estrema pulizia e chiarezza.

Desidero ringraziare in modo particolare gli insegnanti (e vorrei dire *i maestri*, ciascuno a modo suo) che ho avuto la fortuna di incontrare nel mio cammino. In ordine di apparizione: Dino Pieri, Attilio Forino, Giorgio Levi, Fabrizio Luccio, Marco Vanneschi e Simone Martini. Essi sono responsabili in parte del grande desiderio che ho di insegnare, il che costituisce il principale motivo per il quale mi sono imbarcato in questa avventura.

Grazie a tutti gli amici del dipartimento di informatica dell'università di Pisa. In particolare, grazie (per vari motivi) a Bruno Bacci, Piero Bonatti, Alessio Bragadini, Marco Comini, Simone Contiero, Marco Danelutto, Pierpaolo Degano, Andrea Dell'Amico, Agostino Dovier, Moreno Falaschi, Paolo Ferragina, Tito Flagella, Luca Francesconi, Maurizio Gabbrielli, Roberto Giacobazzi, Francesca Levi, Paolo Mancarella, Andrea Masini, Dino Pedreschi, Susanna Pelagatti, Corrado Priami, Paola Quaglia, Alessandra Raffaetà, Francesca Rossi, Francesca Scozzari, Laura Semini, Alessandro Sperduti, Stefano Suin, Franco Turini, e Paolo Volpe. Un ringraziamento speciale va a Letizia Petrellese per l'ottimo lavoro che svolge al dipartimento, e soprattutto per quell'attitudine preziosissima che consiste nel farsi carico dei problemi in prima persona. Grazie anche a Rita Cantini per la cortesia e per la disponibilità.

Il ringraziamento al mio supervisore, Giorgio Levi, merita un discorso a parte. Quando iniziai il dottorato di ricerca mi fu profetizzato che sarei stato oggetto di sfruttamento e soprattutto da parte del mio supervisore, chiunque fosse stato, per il motivo che così è, così è sempre stato e così sempre sarà. Al di là della profezia, le voci sullo sfruttamento più o meno selvaggio dei dottorandi sono una costante della vita universitaria. Ebbene, al termine di questi quattro anni desidero rendere testimonianza a Giorgio del fatto che egli non mi ha mai, mai, mai una sola volta sfruttato in alcuna maniera. Al contrario, egli mi ha sempre supportato, sostenuto ed aiutato in ogni modo possibile. Non mi ha mai chiesto di fare nulla che non fosse nel mio interesse ed è sempre stato, prima di tutto, un amico, sia nell'accordo che nel disaccordo.[2]

Grazie a Kim Marriott per avermi invitato a passare un mese, che è stato di grande ispirazione, in visita alla Monash University di Melbourne. Grazie a Enea Zaffanella per il lavoro e le discussioni fatte insieme, e per l'aiuto che mi ha dato in varie occasioni. Grazie a Maria Cristina Scudellari per il lavoro fatto insieme sul materiale qui riportato al capitolo 7. Grazie a tutti coloro che hanno letto parti di questa tesi per la loro attenzione e i loro suggerimenti: Maurizio Gabbrielli, Roberto Giacobazzi, Patricia M. Hill, Giorgio Levi, Catiuscia Palamidessi, Francesca Rossi, Vijay Saraswat ed Enea Zaffanella. Grazie a Tony Cohn e Pat Hill per avermi consentito di finire questo lavoro di tesi con tranquillità.

Grazie ai miei revisori esterni, Andrew M. King e William H. Winsborough, per la cura con cui hanno svolto il loro lavoro e per avermi letteralmente sommerso di commenti e consigli utili.

Un grazie particolare a mio fratello Abramo che mi è sempre stato vicino in questi 18 anni di informatica: con lui e da lui ho imparato tanto. In particolare, Abramo mi aiutato moltissimo nello sviluppo di CHINA.

Grazie a Rossana per essermi stata vicina, per avermi incoraggiato, e per aver sopportato, in questi ultimi mesi, i miei assurdi ritmi di lavoro ed il mio umore altalenante.

Le mie scuse, infine, a tutti coloro che ho trascurato durante la stesura di questa tesi.

Roberto Bagnara

[2]Dicendo questo non intendo assolutamente sostenere che lo sfruttamento dei dottorandi non esista, ma, semplicemente, che la mia esperienza personale è di segno totalmente contrario.

Suvereto, 26 febbraio 1997

Si sopravvaluta facilmente
l'importanza del proprio dire e fare,
rispetto a ciò che uno è diventato
solo grazie agli altri.
— DIETRICH BONHOEFFER, da *Resistenza e Resa: lettere e scritti dal carcere* (1943–1945)

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Contents

1.1 The Objective

This study is aimed at the *development* of *precise*, *practical*, and *theoretically well-founded data-flow analyzers* for *constraint logic-based languages*. The emphasis on the above words is important for a full understanding of what follows, since some of the author's personal views and expectations are hidden behind them. Thus, in order to make our aims clear from the very beginning, we should start explaining the exact meaning of each word. This can be better achieved by treating them in reverse order.

Constraint. Many problems can be conveniently expressed, reasoned about, and solved by means of constraints. These problems range from temporal, spatial, and physical reasoning, to image processing, automated diagnosis, plus all the problems that are typical of the fields of operation research and mathematical programming. As we will see, an important class of data-flow analyses for constraint logic-based languages can also be easily understood in terms of constraint manipulation.

The use of constraints as a description language and of techniques based on constraints has a long history in the field of Artificial Intelligence. Many constraint-based applications have been developed in the last 30 years in order to explore and solve specific problems. These systems, however, were often *ad hoc* systems with very little in common among them. Eventually, researchers realized that constraints could allow for programming machines in a novel way: *constraint programming.*[1]

Logic-Based Languages. Those programming paradigms that are "based on logic" are the best suited for constraint programming. Indeed, viewing basic constraints as atomic formulae in first-order predicate logic is a natural thing to do. Logic-based languages then provide logical and meta-logical ways of composing and dealing with constraints.

A significant part of the material contained in the present thesis deals with constraints and ways of approximating them by means of other constraints. Thus, several results are of interest independently from the considered programming paradigm. Nonetheless we focus our attention on the "constraint logic programming" paradigm for reasons that will soon be explained.

Constraint logic programming (CLP) is a generalization of the pure logic programming paradigm (LP), having very similar model-theoretic, fixpoint and operational semantics [JL87, JM94]. It embodies a quite natural view of "constraint computations" expressed by definite clauses. The CLP notion of constraint solving in a given algebraic structure encompasses the one of unification over some Herbrand universe. This gives CLP languages a great advantage over LP in terms of expressivity and flexibility.

We believe that CLP languages have a future, and this is the first reason for our particular interest. They provide elegant and simple linguistic support for constraint programming. This makes them suitable for a number of applications. Several CLP languages and systems exist to date, both in academic and industrial installations. Thus, CLP languages now have a significant user community. As a consequence we have a rough idea as to what a constraint logic program looks like: for a project that aims at some experimental validation of the theoretical ideas, this is an important factor.

Another motivation for studying data-flow analysis of CLP languages is that such languages can benefit from the analysis' results more than other paradigms. Both CLP compilers and programmers can do a much better job if they are given information about the run-time behavior of programs.

In some cases CLP programs can be naturally more efficient than the correspondent LP ones, because of the possibility of reasoning directly in the "domain of discourse" (integer arithmetic, for instance), without requiring complicated encodings of data objects as first-order terms. However, the basic operational step in CLP program execution, a test for solvability of constraints, is generally far more complex than unification. Ideally, a cor-

[1] An embryonic notion of constraint programming goes back to the SKETCHPAD system of Sutherland: an interactive system where complex graphic objects could be specified by imposing constraints on the various attributes of the objects themselves [Sut63]. One of the first examples of true constraint programming languages is CONSTRAINTS, by Sussman and Steele [SS80].

rect implementation of a CLP language needs a *complete* solver, that is a full decision procedure for the satisfiability of constraints in the language's domain(s).[2] The indiscriminate use of such complete solvers in their full generality can lead to severe inefficiencies. For these reasons, the optimized compilation of CLP programs can give rise to impressive performance improvements, even more impressive than the ones obtainable for the compilation of Prolog.

The CLP paradigm inherits from LP the complete freedom that programmers have when writing their program. Usually there are no prescriptive declarations (*types*, for instance) and any program that is syntactically correct is a legal one. Even though this is a much debated point, absolute freedom makes CLP languages attractive for the so called *fast-prototyping* of applications. The back side of the coin is that there is not much the compiler can do in order to help programmers and maintainers during the various phases of the software life-cycle. This clearly poses serious software engineering problems and, as a matter of fact, many prototypes do not evolve into real applications. The good news is that (constraint) logic programming, due to its unique semantic features, has a unique potential for formal program development. This possibility can be turned into a reality by providing a set of *semantics-based tools* (i.e., based on program analysis) for writing, debugging, manipulating, and reasoning about programs.

In summary, data-flow analysis has a great potential for providing valuable information to both the compiler and the developer of CLP programs. It promises to play a fundamental role in the achievement of the last point in the following wish list for CLP: (1) retaining the declarativity and flexibility of logic programming; (2) augmenting expressivity by allowing direct programming in the intended computational domain; (3) gaining a competitive advantage, in terms of efficiency, productivity, and maintainability, over other programming paradigms.

Data-Flow Analyzers. A *data-flow analyzer* is a computer program: it takes a program P as its input and, after a reasonable amount of time, it delivers some partial information about the run-time behavior of P. As such, a data-flow analyzer is quite different from that which is usually meant by "a data-flow analysis": a collection of techniques by which one can, in principle, derive the desired information about the behavior of programs. While it is undoubtedly true that a data-flow analyzer implements some data-flow analyses, the standard connotations of the two terms allow for a huge gap in the level of detail that one has to provide.

[2]Strictly speaking, this is not true, as the hypothesis of completeness of the constraint solver can be relaxed, still obtaining useful systems [JM94, BCSZ96]. However, if the system does not employ a complete solver the user must be prepared to receive "answer constraints" that are inconsistent. Thus, a complete solver is required anyway, sooner or later.

Designing a data-flow analysis requires a significant effort. The theory of *abstract interpretation* [CC77, CC92b] is a blessing in this respect, as it provides a variety of frameworks for ensuring the correctness of the analysis, once you have defined a domain of approximate program properties. The theory also provides standard ways of composing existing domains, reasoning about them, and deriving approximate versions of the relevant operations of the language at hand. However, the theory does not tell you how to invent an *abstract domain* that represents a reasonable compromise between precision and efficiency for the class of programs being analyzed. This is what makes the design of data-flow analyses potentially hard, depending on the use one has in mind for them.

If the purpose of the analysis is to write a paper, a number of shortcuts are possible. Some of them are completely justified by the needs of the presentation: cluttering the description with unnecessary details is counter-productive. The author's definition for "unnecessary detail" is: something that can be easily filled in by the interested reader without breaking any result in the paper. Other shortcuts are more questionable. Here is an incomplete categorization of things that make life easier.

- *Pretend that the language to be analyzed is not what it is.* Assuming that, in CLP languages, program variables and constraints are *typed* is an example. If one takes CLP($\mathcal{R}$), for instance, a clause can contain a variable X and absolutely no indication of whether X is a numerical variable or not. There is no typing at all, indeed: in some execution paths X can be a numerical variable whereas in some other paths X is bound to an uninterpreted functor. The same thing happens for constraints of the form $X = Y$ and for implicit constraints that are determined by repeated occurrences of variables in the head of clauses, like $p(X, X)$: sometimes $X = Y$ denotes arithmetic equality, sometimes it is an unification constraint.

  Pretending that implemented logic-based languages perform unification without omitting the *occur-check* is another notable example of this kind of sin. In the literature on data-flow analysis this problem is systematically swept under the carpet, even in papers dealing with the analysis of Prolog (which, proverbially, omits the *occur-check*).

- *Assume that it can easily be implemented.* Excessive use of semantic notions, for instance, can be the source of difficulties. The reduced product, for example, is a nice conceptual device that is applicable in many cases. However, it is defined in terms of the concretization function. As a consequence, representing and computing some canonical form of the reduced elements may turn out to be impractical. There are indeed cases where one must content oneself with an approximation of the reduced product. Still on this subject, specifying an abstract domain as a space of (upper or lower) closure operators on some complete lattice usually allows for elegant formalizations. It is important not to forget that a closure operator is a Platonic object: sooner or later it will have to be represented on the machine, and this can pose serious problems.

- *Disregard efficiency.* This is not so bad, in general, as the situation can be redressed by resorting to widening operators. The problem of designing widening operators and of determining where and when to apply them must not be underestimated.

- *Compromise precision.* The opposite mistake. A worse one, indeed: too much anxiety about efficiency can lead to the selection of analysis frameworks that lose too much information from the very beginning. A state of things that has few chances of being fixed.

- *Pretend that the programs to be analyzed are reasonable.* For instance, that no program clause contains more than, say, 64 variables. Or that programs do not use "nasty constructs" such as `assert/1`. After all, if a program turns out to be *unreasonable* the analysis can answer "don't know": this is always a legitimate thing to do in data-flow analysis.

In the author's experience, the researcher who engages in the task of designing a data-flow analysis with reasonable chances of being successfully implementable must be prepared to have a bad time. The mistakes that have been outlined above can trap him at any stage. Furthermore, during the pencil-and-paper development many questions arise naturally, and most of them cannot be answered. One typical form of such questions is the following: "am I sure that *this* is reasonably implementable?" Sometimes the answer comes later in the actual implementation stage. If the answer is negative some other solution has to be found. The implementor will have to conduct his own research, and an analysis distinct from the one proposed by the designer will have to be implemented.

Another class of questions is: "how much am I going to gain if I do that, for the *average* program?" For relatively new programming paradigms such as CLP the notion of *average program* makes little sense, which, again, means that there can be no answer. Thus, the risk of spending much time refining an analysis so as to make it implementable, and later discovering that relatively few programs may benefit from the results, is high.

In an attempt to circumvent these difficulties, we started our work with *analyzers*, not just *analyses*, in mind. In other words, our interest is in "implementable data-flow analyses", or, stated differently, "data-flow analyses that are actually implemented by analyzers". There is no free lunch, of course: experience has shown that dealing with analyzers, while perhaps saving you from overlooking important aspects of the problem, forebode a plethora of other troubles. This, however, is another story. By the way, the name of the author's pet analyzer is CHINA[3]. CHINA is a data-flow analyzer for CLP languages based on the *abstract compilation* approach. It performs bottom-up analysis deriving information on both call- and success-patterns by means of program transformations and optimized fixpoint computation techniques. It currently supports *combined* domains for: (parametric) structural information, simple types, definiteness, aliasing and freeness, numerical bounds and relations of and among numerical leaves, and polymorphic types. CHINA consists of about 40,000 lines of C++ code for the abstract interpreter, and about 5,000 lines of Prolog code for the abstract compiler.

Theoretically Well-Founded. The approach of putting oneself in the implementor's shoes, of course, must not be to the detriment of a careful study of the correctness issues. This requires, besides firm semantic bases, formal methodologies to ensure the correctness of the analysis. Abstract interpretation is *the* framework of choice for this purpose.

It is the author's impression that too often the importance ascribed to a result is directly proportional to the strength of its conclusion. The ratio between the strength of the conclusion and the strength of the premises should, instead, be considered as the right measure. A weak conclusion that can be established starting from weak premises can be very valuable. This is especially the case when one deals with real implementations where, in order to tackle the various complexity precision trade-offs, one needs as much freedom as possible. Often this implies that the strong premises cannot be ensured, while weak conclusions would be enough. *Scalable theories* are particularly precious in this respect. The author's favorite reference for abstract interpretation is *Abstract Interpretation Frameworks*, by Cousot and Cousot [CC92b]. There, the authors start from very weak hypotheses that only guarantee correctness and finiteness of the analysis. Only later Galois connections and insertions are introduced in a blaze of glory, with all their nice properties.

Practical. A "practical analyzer" is one that has a chance to be turned into a useful tool. On one side this means that compromising assumptions about the languages and the programs must be avoided as much as possible. On the other side, potential efficiency must be taken into consideration. However, striving too much for efficiency should be avoided. In the literature there are papers reporting on the experimental evaluation of data-flow analyzers. In some of them one can find analysis' times well under the second for non-trivial, lengthy programs. It is true that some of these papers have, among their objectives, the praiseworthy one of convincing the reader of the practicality of data-flow analysis. Nonetheless the author finds these

[3]CLP(HERBRAND INTEGRATED-WITH NUMERIC-DOMAINS) ANALYZER.

results a bit disconcerting. What can one conclude from the fact that a program of some hundreds lines can be analyzed in a couple of seconds?

Hypothesis 1: That the problem of data-flow analysis for logic programs has been *brilliantly* solved once and for all?

Hypothesis 2: That special assumptions have been exploited so that the results cannot be generalized?

Hypothesis 3: That much more precision is attainable?

The first hypothesis is hardly credible. In the case of Prolog, the effectiveness of analysis has been demonstrated experimentally by Parma [Tay90] and Aquarius Prolog [VD92]. Impressive speedups have been reported. And today's analyzers are far more precise and efficient than those employed in Parma and Aquarius Prolog. Despite this fact, to the author's knowledge no known Prolog system to date, whether existing or being developed, includes a global data-flow analysis stage to drive the optimization phase. Is this an indication that at least somebody does not believe in Hypothesis 1? Probably yes, and probably he is right.[4]

Hypothesis 2 has already been discussed. Whenever it applies we are faced with the research problem of removing any limiting assumption.

The author favors the last hypothesis. A couple of seconds is a ridiculous time for, say, optimized compilation. If one takes into account that

1. only *production versions* deserve to be compiled with the optimization pass turned on, and

2. a production version is compiled once and used thousands, perhaps millions of times,

then it is clear that analysis times remain totally acceptable even if multiplied by a factor of 50 or 100.[5] Then the research problem is: how can we make profitable use of the extra-time that users are willing to pay for?

[4]Since there are serious indications that the logic programming community is not fond of itself, the answer cannot be: *yes, definitely.* Some explanations are in order. On the vast majority of Prolog implementations (here included those who claim to have been *industrialized*) the user has to wait for a run-time exception just to know that he has misspelled a (non-dynamic) predicate name. A run-time error (or, if lucky, a mild warning) is all one can hope for in cases of built-in misuse like `p(ArgNo, Term, Arg) :- arg(Argno, Term, Arg).` By the way, normally run-time errors come without any indication about which clause was executed when the erroneous situation occurred. Sometimes we really wonder if this does not unveil some masochism on the part of the logic programming community. The fact that we are *Programming in Logic* is no excuse for having less tools at our disposal than those *Programming in Assembly*!

[5]Some people may disagree with this assertion. The author's opinion is motivated by the following (admittedly subjective) facts: the China system takes 30 minutes to compile without optimization; with full optimization it takes around 3 hours; China has been compiled in optimized mode perhaps 10 times during the last year, always overnight;

Performing other analyses is one answer, increasing the precision of existing analyses is another possibility.

Precise. The author does not share the view that only fast analyses with immediate, consistent payoffs in optimized compilation deserve to be studied. There are other applications of data-flow analysis, such as semantics-based programming environments, where one probably needs very precise information about programs' behavior in order, say, to assist the programmer during the development. Ironically, in this field the efficiency concerns are much more stringent, if one aims at interactive development tools.

So we should go for more precision. The problem is how to increase precision yet avoiding the concrete effects of exponential complexity. Consider groundness analysis, for instance. The cruder domains do not pose any efficiency problem. In contrast, the more refined domains for groundness, such as *Pos*, work perfectly until you bump into a program clause with more than, say, fifty variables. At that time, *Pos* will blow-up your computer's memory. One would like to have a more *linear*, or *stable* behavior. The right solution, as indicated by Cousot and Cousot [CC92c], is not to revert to the simpler domains. We should use complex domains instead, together with sophisticated widening/narrowing operators. With this technique we can try to limit precision losses to those cases where we cannot afford the complexity implied by the refined domains.

Ideally, it should be possible to endow data-flow analyzers with a *knob*. The user could then "rotate the knob" in order to control the complexity/precision ratio of the system. The widening/narrowing approach can make this possibility a reality. Unfortunately, the design of widening operators tends somewhat to escape the realm of theoretical analysis, and thus, in the author's opinion, it has not be studied enough. Indeed, the development of successful widening operators requires, perhaps more than other things, extensive experimentation.

Development. This word is emphasized just to warn the reader that the above description is about the ideas and aspirations that drove our work, not about what we have already achieved. There, is where we want to go; and it is far away from where we are now.

the author would be delighted to have a C++ compiler delivering a 10% speedup on the execution of CHINA in exchange for 12 hours (the entire night) of compilation time. Last but not least, software houses can afford machines that are 20 times faster than the author's PC. And the duration of the night is the same for them, too!

1.2 The Contents

In this thesis you can find an account of the author's research from March 1993 to October 1996. The work included here is all centered around the development of the China analyzer. By this we mean:

1. that almost any theoretical study we have undertaken was suggested by difficulties in the design or implementation of China;
2. that China has been used to experimentally validate, as much as possible, several ideas that appeared to be nice in theory.

The author believes that the thesis should be of interest to all those involved in data-flow analysis and semantics-based program manipulation of constraint programming languages. A considerable effort has been made in making the presentation as readable as possible: the reader must judge to what extent we succeeded.

1.2.1 How It All Started

Some history is required so as to put the subsequent material into context. Also, since some chapters in this thesis, or their leading ideas, are based on papers that have already been published, we take the occasion to report here the relevant bibliographic sources.

We tackled the problem of data-flow analysis of CLP languages over finite numeric domains back in 1992. In [Bag92] we proposed an abstract interpretation deriving *spatial approximations* of the success set of program predicates. The concrete interpretation of a constraint, that is, a *shape* in k-dimensional space, was abstracted by an enclosing, though not necessarily minimal, *bounding box*. Bounding boxes are just rectangular regions with sides parallel to the axes. Thus, a bounding box is univocally identified by its projections (i.e., intervals) over the axis associated to the variables. Of course, bounding boxes are very coarse approximations of general shapes. Finer spatial approximations exist and are well-known, such as enclosing convex polyhedra, grid and Z-order approximations [HMO91] and so on. However, the coarseness of bounding boxes is well repaid by the relative facility with which they can be derived, manipulated and used to deduce information relevant to static analysis. There are several techniques for deriving bounding boxes from a given set of arithmetic constraints (especially if one confines attention to linear constraints): variants of Fourier-Motzkin variable elimination [DE73, Pug92], the *sup-inf* method [Ble74, Sho77, Sho81] etc. In [Bag92] we considered a variant of the constraint propagation technique called *label inference with interval labels* [Dav87]. More precisely we used the Waltz algorithm [Wal75, Mac77, MF85] applying *label refinement*, in which the bounding box corresponding to a set of constraints is gradually restricted.

Later, we turned our attention to the analysis and compilation issues of languages such as CLP($\mathcal{R}$). In particular, we studied a notion of *future redundant constraint* less restrictive than the one introduced by Jørgensen *et al.* [JMM91]. An analysis for the detection of implicit numeric constraints, future redundant ones in particular, was sketched by Giacobazzi, Levi and the author in [BGL92], and the constraint propagation techniques used, collectively known as *constraint inference* [Dav87], were presented in [BGL93]. Meanwhile, the development of CHINA's first prototype was started. The applications of implicit and redundant constraint analysis were first described in [Bag94], together with some experimental results obtained with the prototype analyzer.

The formalization of the (concrete and abstract) domains and of the analysis was unsatisfactory. What characterizes a numeric constraint? A possible answer is "the set of its solutions". Some authors have formalized concrete domains that way [GDL92]. But this does not work in general as, in order to constructively define an *abstraction function* you must know the solutions. Unfortunately this is unfeasible or even impossible, in general: solving a set of equations of the form $(x_i - x_j)^2 + (y_i - y_j)^2 = c_{ij}$ is NP-hard [Yem79], whereas it is undecidable whether a transcendental expression is equal to zero [Ric68]. As a final example, solving a set of linear integer constraints with at most two variables per inequality is NP-complete [Lag85].

For these reasons we chose to define a constraint as "the set of its consequences under some *consequence relation*". This is a much more reasonable definition as it allows to capture both the *concrete domains* really implemented by CLP systems and the class of approximations we were considering. We were thus lead to the study of *partial information systems* [Sco82] and *constraint systems* [SRP91]. Another theoretical difficulty was due to the fact that our numerical domain was a combination of two other domains: one for real intervals and the other for *ordinal relationships*. The combination was something different from the reduced product: sometimes less precise, sometimes more. More importantly, we wanted the combination to be *programmable* in order to meet different balances between precision and efficiency. As a solution we gave the dignity of constraints to a restricted class of cc agents [SRP91, Sar93]. Some pieces of the theory were missing[6] and we became deeply involved in the subject. The results appeared first in [Bag95a] and refined in [Bag97].

Meanwhile, the development of the second prototype of CHINA was initiated. At some point we faced the problem of analyzing non-linear constraints for CLP($\mathcal{R}$). As in CLP($\mathcal{R}$) non-linears are delayed until they become linear, the analyzer is not allowed to derive anything from them unless the analysis

[6]For instance: what is the join of two finite cc agents? Is it representable by means of a finite cc agents? If not (as it is the case), how can we design suitable approximations of the join operation?

has inferred that they have indeed become linear. Having solved the theoretical difficulties about the combination of domains, we implemented the *Pos* groundness domain in the standard way, and we combined it with the numerical component. The interaction could be implemented in different ways: the numerical component could *ask* the definiteness component, unsolicited definiteness information could be sent to the numerical component, or a mixture of the two. In addition, some groundness information that is synthesized in the numerical component should be notified to the definiteness component. However, no one had studied the problem of efficiently detecting ground variables in the context of the standard implementation of *Pos*. This motivated our interest in groundness analysis, a topic that we believed was almost closed. The outcome of our studies on the subject was briefly sketched in [Bag96c] and later presented in [Bag96b].

The China system is still evolving, and has never previously been described, although a very short description of the first prototype of China is given in [Bag94].

Recently, we have been working on the precise detection of *call-patterns* by means of program transformation. In a joint work with Maria Cristina Scudellari (a former student at the University of Pisa) we devised a solution. However, this has not undergone experimental evaluation yet. It is presented here for the first time in Chapter 7.

1.2.2 Plan of the Thesis

Chapter 2 introduces some of the necessary mathematical background. It is mostly intended to define the terminology and notation that will be used throughout the thesis.

Chapter 3 is motivated by the fact that many interesting analyses for constraint logic-based languages are aimed at the detection of *monotonic* properties, that is to say, properties which are preserved as the computation progresses. Our basic claim is that most, if not all, of these analyses can be described within a unified notion of constraint domains. We present a class of constraint systems which allows for a smooth integration within an appropriate framework for the definition of non-standard semantics of constraint logic-based languages. Such a framework is also presented and motivated. We then show how such domains can be built, as well as construction techniques which induce a hierarchy of domains with interesting properties. In particular, we propose a general methodology for domain combination with asynchronous interaction (i.e., the interaction is not necessarily synchronized with the domains' operations). Following this methodology, interesting combinations of domains can be expressed with all the semantic elegance of concurrent constraint programming languages.

Chapter 4 presents the rational construction of a generic domain for structural analysis of CLP languages: Pattern($\mathcal{D}^\sharp$), where the parameter $\mathcal{D}^\sharp$ is an abstract domain satisfying certain properties. Our domain builds on the parameterized domain for the analysis of Prolog programs `Pat(`$\Re$`)`, which is due to Cortesi *et al.* [CLV93, CLV94]. However, the formalization of our CLP abstract domain is independent from specific implementation techniques: `Pat(`$\Re$`)` (slightly extended and corrected) is one of the possible implementations. Reasoning at a higher level of abstraction we are able to appeal to familiar notions of unification theory. One advantage is that we can identify an important parameter (a common anti-instance function, missing in [CLV93]) that gives some control over the precision and computational cost of the resulting generic structural domain. Moreover, we deal —apparently for the first time— with the *occur-check problem*: while, to our knowledge, all the analysis' domains that have been proposed in the literature (`Pat(`$\Re$`)` included) assume that the analyzed language implements complete unification, most real languages do omit the occur-check. This fact has serious implications both in terms of correctness and precision of the analysis.

Chapter 5 addresses the problem of compile-time detection of implicit and redundant numeric constraints in CLP programs. We discuss how this kind of constraints has several important applications in the general field of semantics based program manipulation, and, specifically, optimized compilation. In particular, we propose a novel optimization based on *call-graph simplification*. We then describe a sequence of approximations for characterizing, by means of ranges and relations, the values of the numerical leaves that appear in Herbrand terms. This, among other things, brings to light the need of information about how many numerical leaves a term has. For each approximation needed for the analysis we offer different alternatives that allow for dealing with the complexity/precision tradeoff in several ways. The overall numeric domain (or, better, a family of them) is obtained (among other things) by means of the product and the ask-and-tell constructions of Chapter 3. The ask-and-tell constraint system constitutes a very convenient formalism for expressing both (1) efficient reasoning techniques originating from the world of artificial intelligence, where approximate deduction holds the spotlight since the origins; and (2) all the abstract operations needed for the analysis. In practice, we define a family of concurrent languages that serve as target-languages in an abstract compilation approach.

Chapter 6 deals with groundness analysis for (constraint) logic programs. This subject has been widely studied, and interesting domains have been proposed. *Pos* (a domain of Boolean functions) has been recognized as the most suitable domain for capturing the kind of dependencies arising in groundness analysis. Its standard implementation is based on *reduced ordered binary-decision diagrams* (ROBDDs), a well-known symbolic representation for Boolean functions. Even though several authors have reported positive experiences using ROBDDs for groundness analysis, in the literature there is no reference to the problem of the efficient detection of those variables that are deemed to be ground in the context of a ROBDD. This is not surprising, since most currently implemented analyzers need to derive this information only *at the end* of the analysis and only for presentation purposes. Things are much different when this information is required *during* the analysis. This need arises when dealing with languages which employ some sort of *delay mechanism*, which are typically based on groundness conditions. In these cases, the *naïf* approaches are too inefficient, since the abstract interpreter must quickly (and often) decide whether a constraint is delayed or not. Fast access to ground variables is also necessary when aliasing analysis is performed using a domain that does not keep track of ground dependencies. We introduce and study the problem, proposing two possible solutions. The second one, besides making possible the quick detection of ground variables, has also the effect of keeping the ROBDDs as small as possible, improving the efficiency of groundness analysis in itself.

Chapter 7 presents some recent work about the analysis of *call-patterns* for constraint logic programs. In principle, call-patterns can be reconstructed, to a limited extent, from the success-patterns. This, however, often implies significant precision losses. As the precision of call-patterns is very important for many applications, their direct computation is desirable. Top-down analysis methods are usually advocated for this purpose, since the standard execution strategy of (constraint) logic programs is top-down. Alternatively, bottom-up analysis methods based on the *Magic Sets* or similar transformations can be used [Kan93, CDY94, Nil91]. This approach, however, can result in a loss of precision because the connection between call- and success-patterns is not preserved. In a recent work, Debray and Ramakrishnan [DR94] introduced a bottom-up analysis technique for logic programs based on program transformation. They showed, among other things, that their bottom-up analysis is at least as precise (on both call and success-patterns) as any top-down abstract interpretation using the same abstract domain and abstract operators. The basic idea behind [DR94] is to employ a variation of the *Magic Templates* algorithm [Ram88]. Moreover, the (possibly approximated or abstract) “meaning” of a program clause (or predicate) is a partial function mapping descriptions of tuples of terms (the arguments at the moment of the invocation: call-patterns) into sets of such descriptions (describing the possible arguments at the return point of that invocation: success-patterns). The solution of [DR94], however, is not generalizable to the entire class of CLP languages, since they exploit the peculiar properties of the Herbrand constraint system, where a tuple of terms is a strong normal form for constraints. This way, and by duplicating the arguments, they are able to describe the partial functions which represent the connection between call and success-patterns *in the clauses heads*. Of course, this cannot be done for generic CLP languages. Our aim is thus to generalize the overall result of [DR94] to the general case of constraint logic programs: "*the abstract interpretation of languages with a top-down execution strategy need not itself be top-down*".

Chapter 8 draws some final conclusion about what we have done and learned. More importantly, it traces some lines for further research describing what remains to be done in order to approach the objectives that have been delineated in this introduction.

1.3 The Approach

As the reader will have already noticed, our approach is based on a mixture of theory and experimentation. In the field of computer science these two aspects of research are too often confined in distinct, separate worlds. About this subject, the "Committee on Academic Careers for Experimental Computer Scientists" of the U.S.A. National Research Council has words that deserve a long quotation [NRC94].

> [...] the crux of the problem is a critical difference in the way the theoretical and experimental research methodologies approach research questions. The problem derives from the enormous complexity that is fundamental to computational problems [...] This complexity is confronted in the theoretical and experimental research in different ways, as the following oversimplified formulation exhibits.
>
> When presented with a computational problem, a theoretician tries to simplify it to a clean, core question that can be defined with mathematical rigor and analyzed completely. In the simplification, significant parts of the problem may be removed to expose the core question, and simplifying assumptions may be introduced. The goal is to reduce the complexity to a point where it is analytically tractable. As anyone who has tried it knows, theoretical analysis can be extremely difficult, even for apparently straightforward questions.
>
> When presented with a computational problem, an experimentalist tries to decompose it into subproblems, so that each can be solved separately and reassembled for an overall solution. In the decomposition, careful attention is paid to the partitioning

> so that clean interfaces with controlled interactions remain. The goal is to contain the complexity, and limit the number and variety of mechanisms needed to solve the problem. As anyone who has tried it knows, experimentation can be extremely difficult to get right, requiring science, engineering, and occasionally, good judgment and taste.
>
> The distinction between these two methodologies naturally fosters a point of view that looks with disdain the research of the other. When experimentalists consider a problem that has been attacked theoretically and study the related theorems that have been produced, they may see the work as irrelevant. After all, the aspects that were abstracted away embodied critical complicating features of the original problem, and these have not been addressed. The theoretician knows no analysis would have been possible had they been retained, whereas the experimentalist sees that "hard parts" of the problem have been left untouched.
>
> Conversely, when theoreticians examine a problem attacked experimentally and spot subproblems for which they recognize theoretical solutions, they may see the work as uninformed and nonscientific. After all, basic, known results of computing have not been applied in this artifact, and so the experimentalist is not doing research, just "hacking." The experimentalist knows that it is the other aspects of the system that represent the research accomplishment, and the fact that it works by using a "wrong" solution implies that the subproblem could not have been too significant anyway.
>
> So, as by the blind men encountering an elephant, impressions are formed about the significance, integrity, and worth of computing research by its practitioners. Although it is natural for researchers to believe their own methodology is better, no claim of superiority can be sustained by either. Fundamental advances in Computer Science & Engineering have been achieved by both experiment and theory. Recognizing that fact promotes tolerance and reduces tensions.

This excerpt does contain a faithful description of the incommunicability that seems to be so frequent in our field. However, it takes for granted that there are two kinds of computer science researcher: the theoretician and the experimentalist.

The author has done five years of "real programming work" in environments where the theoreticians were often laughed at (and these environments were probably the most *illuminated* ones among those where real programming takes place). And during the last four years he has done full-time research work in a milieu where experimentalists "look strange", so to speak. These hostile feelings are sometimes justified. There are, indeed, experimentalists that seem unable to communicate their findings to the other researchers; some even refuse to do it. On the other hand there are theoreticians that like filling the "applications" sections of their papers with conjectures, even though these speculations are not explicitly presented as such.

These observations have convinced the author that being a theoretician and an experimentalist *at the same time* was something worth a try. The risk of being mediocre in both these activities is high, of course, but we must say that we are satisfied with this choice. Having tried, our impression is that theoretical and experimental work do have synergetic effects. As it has already been mentioned, most of our theoretical studies were suggested by practical difficulties. Moreover, the author shamelessly admits that several deficiencies in his theories have been discovered either at the time of writing the actual code (do not know what to write: an under-specified theory), or by observing the results obtained with the implementation (a wrong theory).

If you try to write for the novice
you will communicate with the experts;
otherwise you will communicate with nobody.
— DONALD E. KNUTH, lecturing on *Mathematical Writing*,
Stanford University (1987)

Chapter 2

Preliminaries

Contents

In this chapter we introduce the mathematical concepts and notations that will be used throughout the thesis. The intention is to setup a common language with the reader, and not to explain these concepts in detail. Thus, throughout the thesis we will assume familiarity with the basic notions of lattice theory, semantics of logic programming languages, and abstract interpretation.

2.1 Sets

Let U be a set. The set of all subsets of U will be denoted by $\wp(U)$. The set of all *finite* subsets of U will be denoted by $\wp_{\mathrm{f}}(U)$. The notation $S \subseteq_{\mathrm{f}} T$ stands for $S \in \wp_{\mathrm{f}}(T)$. For $S \subseteq U$ we will denote the complement $U \setminus S$ by $\overline{S}$, when U is clear from the context. For $S, T \subseteq U$ the notation $S \uplus T$ denotes *disjoint union*, emphasizing the fact that $S \cap T = \varnothing$.

2.2 Some Predefined Sets

The sets of all natural, integer, rational, real, and ordinal numbers will be denoted, respectively, by $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{O}$. The first limit ordinal equipotent with the set of natural numbers is denoted by ω.

2.3 Multisets

A *multiset* is a mathematical entity that is like a set except for the fact that it can contain identical elements a finite number of times [Knu80]. A multiset can be specified by listing all the occurrences of its elements. In order to distinguish multisets from sets, special parentheses are used for this purpose. For instance,

$$\wr a, a, b, c, c, c \int$$

denotes the multiset containing two occurrences of a, one occurrence of b and three occurrences of c. The fact that, say, a occurs more than once in $\wr a, a, b, c, c, c \int$ is denoted by $a \Subset \wr a, a, b, c, c, c \int$. The order in which the occurrences are listed is immaterial; thus, for instance,

$$\wr a, a, a, b, b, c, c \int \quad \text{and} \quad \wr c, a, b, a, c, a, b \int$$

denote the same multiset. However, the multiplicity of occurrences of each element is relevant, so

$$\wr a, a, b, c, c, c \int \quad \text{and} \quad \wr a, a, a, b, b, c, c \int$$

are two different multisets. The empty multiset is denoted by $\varnothing$.

If A and B are multisets, so are $A \uplus B$ and $A \Cap B$. An element occurring n times in A and m times in B occurs exactly $n + m$ times in $A \uplus B$ and exactly $\min(n, m)$ times in $A \Cap B$. It is straightforward to show that: $\uplus$ and $\Cap$ are commutative and associative; $\uplus$ distributes over $\Cap$; $\Cap$ is idempotent; the absorption law $A \Cap (A \uplus B) = A$ is satisfied; $\varnothing$ is an identity for $\uplus$, whereas it is a zero for $\Cap$.

The set of multisets whose elements are drawn from some set S will be denoted by $\wp^+(S)$. Conversely, given a multiset M, the set of all elements which occur in M is denoted by $\zeta(M)$. Formally, if $M \in \wp^+(S)$ then

$$\zeta(M) \stackrel{\text{def}}{=} \{\, x \in S \mid x \Subset M \,\}.$$

The set of *finite* multisets built from the set S is denoted by $\wp_{\mathrm{f}}^+(S)$. The *cardinality* of a multiset M is the number of its elements' occurrences and is denoted by $\|M\|$. Thus

$$\bigl\|\wr a, a, b, c, c, c \int\bigr\| = 6.$$

Both $\wp^+(S)$ and $\wp_{\mathrm{f}}^+(S)$ are partially ordered with respect to the ordering defined by

$$A \sqsubseteq B \overset{\text{def}}{\iff} A \Cap B = A.$$

2.4 Cartesian Products and Sequences

By $U^\star$ we will denote the set of all finite sequences of elements drawn from U. The empty sequence is denoted by ε. For $x \in U^\star$, the *length* of x will be denoted by $\# x$, and, for $i = 1, \dots, \# x$, the notation $x[i]$ stands for the i-th element of x. The concatenation of sequences $x_1, x_2 \in U^\star$ is denoted by $x_1 :: x_2$.

Let $S_1, \dots, S_n$ be sets. We will denote elements of $S_1 \times \cdots \times S_n$ by $(e_1, \dots, e_n)$. The *projection mappings* $\pi_i \colon S_1 \times \cdots \times S_n \to S_i$ are defined, for $i = 1, \dots, n$, by

$$\pi_i\bigl((e_1, \dots, e_n)\bigr) \stackrel{\mathrm{def}}{=} e_i.$$

The *liftings* $\pi_i \colon \wp(S_1 \times \cdots \times S_n) \to \wp(S_i)$ given by

$$\pi_i(T) \stackrel{\mathrm{def}}{=} \{\, \pi_i(t) \mid t \in T \,\}$$

will also be used.

2.5 Binary Relations

A set $R \subseteq S \times S$, where S is any set, is called a *binary relation* (or, simply, *relation*) *over* S. Then, for each $A \subseteq S$, the *application of* R *to* A is defined as

$$R(A) \stackrel{\mathrm{def}}{=} \bigl\{\, x_2 \in S \bigm| \exists x_1 \in A \,.\, (x_1, x_2) \in R \,\bigr\}.$$

The *inverse of* R, denoted R^{-1}, is given by

$$R^{-1} \stackrel{\mathrm{def}}{=} \bigl\{\, (x_2, x_1) \in S \times S \bigm| (x_1, x_2) \in R \,\bigr\},$$

whereas the composition of two relations $R_1, R_2 \subseteq S \times S$ is defined much like function's composition:

$$R_2 \circ R_1 \stackrel{\mathrm{def}}{=} \bigl\{\, (x_1, x_3) \in S \times S \bigm| \exists x_2 \in S \,.\, (x_1, x_2) \in R_1, (x_2, x_3) \in R_2 \,\bigr\}.$$

Finally, the *negation* (or *complement*) of a relation R is simply the corresponding set-theoretic notion:

$$\overline{R} \stackrel{\mathrm{def}}{=} (S \times S) \setminus R.$$

2.6 Preorders, Partial and Total Orders

A *preorder* $\preceq$ over a set P is a binary relation which is reflexive and transitive. For $x \in P$, the *downward closure of* x, written $\downarrow x$, is defined by

$$\downarrow x \stackrel{\text{def}}{=} \{\, y \in P \mid y \preceq x \,\}.$$

If $\preceq$ is also antisymmetric, then it is called *partial order*. $\preceq$ is a *total order* if, in addition, for each $x, y \in P$, either $x \preceq y$ or $y \preceq x$. A set P equipped with a partial (resp. total) order $\preceq$ is said to be *partially ordered* (resp., *totally ordered*), and sometimes written $\langle P, \preceq \rangle$. Partially ordered sets are also called *posets*. A subset S of a poset $\langle P, \preceq \rangle$ is said to be a *chain* if it is totally ordered with respect to $\preceq$.

Given a poset $\langle P, \preceq \rangle$ and $S \subseteq P$, $y \in P$ is an *upper bound* for S if and only if $x \preceq y$ for each $x \in S$. An upper bound y for S is the *least upper bound* (or lub) of S if and only if for every upper bound y' for S it is $y \preceq y'$. The lub, when it exists, is unique. In this case we write $y = \operatorname{lub} S$. *Lower bounds* and *greatest lower bounds* are defined dually. $\langle P, \preceq \rangle$ is said *bounded* if it has a minimum and a maximum element.

2.7 Lattices and Semilattices

A poset $\langle L, \preceq \rangle$ such that, for each $x, y \in L$, both $\operatorname{lub}\{x, y\}$ and $\operatorname{glb}\{x, y\}$ exist, is called a *lattice*. In this case, lub and glb are also called, respectively, the *join* and the *meet* operations of the lattice. A poset where only the glb operation is well-defined is called a *meet-semilattice*. A *complete lattice* is a lattice $\langle L, \preceq \rangle$ such that every subset of L has both a least upper bound and a greatest lower bound.

An algebra $\langle L, \wedge, \vee \rangle$ is also called a *lattice* if $\wedge$ and $\vee$ are two binary operations over L which are commutative, associative, idempotent, and satisfy the following *absorption laws*, for each $x, y \in L$: $x \wedge (x \vee y) = x$ and $x \vee (x \wedge y) = x$. The two definitions of lattices are equivalent. This can be seen by setting up the isomorphism given by: $x \preceq y \stackrel{\text{def}}{\iff} x \wedge y = x \stackrel{\text{def}}{\iff} x \vee y = y$, $\operatorname{glb}\{x, y\} \stackrel{\text{def}}{=} x \wedge y$, and $\operatorname{lub}\{x, y\} \stackrel{\text{def}}{=} x \vee y$.

The notions of meet-semilattice and of bounded lattice are imported into the algebraic definition in the natural way. For instance, a *bounded lattice* [BS81] is an algebra $\langle L, \wedge, \vee, \bot, \top \rangle$ such that $\langle L, \wedge, \vee \rangle$ is a lattice and the following two *annihilation* laws are satisfied for each $x \in L$: $x \wedge \bot = \bot$ and $x \vee \top = \top$. A lattice is said *distributive* if it satisfies, for each $x, y, z \in L$, the *distributive laws* $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ and $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$. It is well-known that, for lattices, the distributive laws are equivalent.

2.8 Closure and Kernel Operators

A monotone and idempotent self-map $\rho\colon P \to P$ over a poset $\langle P, \preceq \rangle$ is a *kernel operator* (or *lower closure operator*) if it is *reductive*, that is to say

$$\forall x \in P : \rho(x) \preceq x.$$

If ρ is *extensive*, namely

$$\forall x \in P : x \preceq \rho(x),$$

then it is called *closure operator* (or *upper closure operator*). The reader is referred to [GHK+80] for an extensive treatment of kernel and closure operators.

2.9 Monoids

An algebra $\langle M, \cdot, 1 \rangle$ is a *monoid* if and only if '$\cdot$' is an associative binary operator over M, and $1 \in M$ satisfies the *identity law*: for each $x \in M$, $x \cdot 1 = 1 \cdot x = x$. The monoid $\langle M, \cdot, 1 \rangle$ is said *commutative* or *idempotent* if '$\cdot$' is so.

Chapter 3

A Hierarchy of Constraint Systems

Contents

3.1 Introduction

Many interesting and useful data-flow analyses for constraint logic-based languages are aimed at the detection of *monotonic* properties, that is to say, properties that are preserved as the computation progresses by accumulating constraints along one computation path. Such analyses usually determine the "shape" of the set of solutions of the constraint store at some program points. Analyses that fall in this category include definiteness (groundness), structural information, types closed by instantiation, numerical bounds and relations, symbolic size-relations and so on. The typical examples of non-monotonic properties are freeness and aliasing. Freeness is anti-monotonic: as soon as a variable becomes non-free it will remain such in any stronger constraint store. The aliasing property is subject to a more complex, oscillating evolution: two variables X and Y may not share any other variable at some point, later they can share a common variable Z, eventually Z becomes ground so that X and Y do not share anymore, but suddenly they start sharing another variable W . . . .

A key observation is that monotonic properties can be conveniently expressed by constraints, which are then accumulated in the analysis process much in the same way as during the "concrete" executions. Thus, frameworks of constraint-based languages are, in principle, general enough to encompass several of their own data-flow analyses. Intuitively, this is done by replacing the standard constraint domain with one that is suitable for expressing the desired information. This fundamental aspect was brought to light, for the case of CLP, by Codognet and Filè [CF92] and elaborated by Giacobazzi, Debray, and Levi [GDL92, GDL95]. Giacobazzi *et al.* presented a generalized algebraic semantics for constraint logic programs, which is parameterized with respect to an underlying constraint domain. The main advantages of this approach are that:

1. different instances of CLP can be used to define non-standard semantics for constraint logic programs; and

2. several abstract interpretations of CLP programs can be thus formalized *inside* the CLP paradigm.

In this setting, data-flow analysis is then performed (or at least justified) through abstract interpretation [CC79, CC92a], that is, by "mimicking" the program run-time behavior by "executing" it, in a finite way, on an approximated (abstract) constraint domain.[1]

By following a generalized semantic approach, the concrete and abstract semantics are more easily related, being instances (over two different constraint systems) of the same generalized semantics, which is entirely parametric on a constraint domain. Thus, to ensure correctness, it will be sufficient to exhibit an "abstraction function" α that is a semi-morphism between the constraint domains [BM83, CC92b].

Starting from [GDL95] a more general notion of constraint domain is provided. This allows one to adequately describe both the "logical part" of concrete computations (i.e., answer constraints) and all the monotonic abstract interpretations known to us. In particular our notion of constraint system is able to accommodate approximate inference techniques whose importance relies on very practical considerations, such as representing good compromises between precision and computational efficiency. Some of these techniques, besides being sketched in the examples of this chapter, will be explained in Chapter 5. The new notion of constraint domain requires the introduction of a new generalized semantics framework that is more liberal than the one of [GDL95].

Moreover, and here comes the main point, we show that our constraint domains admit interesting constructions. The most important one consists in upgrading a domain so that it can represent and manipulate *dependencies* among constraints. This is done by regarding a restricted class of cc agents as constraints [Sar93, SRP91]. This construction, among other things, opens up the possibility of combining domains in a novel and interesting way. By following this methodology, the asynchronous interaction between domains can be expressed with all the elegance that derives from the cc framework.

The plan of the chapter is as follows: Section 3.2 explains our generalized semantics for CLP languages, as well as the abstract interpretation framework we employ. Section 3.3 introduces *simple constraint systems*: some important building blocks of the hierarchy. Section 3.4 builds on the previous one presenting standard ways of representing and composing finite constraints: *determinate constraint systems*. In Section 3.5 it is shown how a constraint system is upgraded to incorporate a weak form of disjunction (suitable to monotonic properties) by means of *powerset constraint systems*. Section 3.6 presents a different kind of upgrade: the one needed in order to have the notion of *dependency* built into the constraint system. This is done considering *ask-and-tell constraint systems*. Section 3.7 deals with the interesting problem of combining domains. A technique is shown that consists in applying the *ask-and-tell construction* to a *product constraint sys-*

[1]Incidentally, here is the reason why we do not like the name 'static analysis'. The adjective 'static' means "without executing the program", but we do execute it, though in a non-standard way, over a non-standard domain.

tem. We feel that, indeed, this is one of the more important contributions of this work. Finally, Section 3.8 draws some conclusions and presents some directions for further study.

3.2 A Case Study: CLP

The constraint domains that are the subject of this work are not bound to a particular class of constraint logic-based languages. However, for the sake of clarity and to help the intuition, we will focus on the class of CLP languages [JL87, JM94], which is more and more influential and captures several existing, implemented languages. We will present a *generalized approach* to the semantics of CLP programs of which abstract interpretation is an important instance. This is necessary for a full understanding of how the domains of later sections are employed in data-flow analysis of CLP languages.

3.2.1 CLP: the Syntax

Here we give a precise definition of what we will call a CLP(C) program. Notice that here we are concerned with syntax only. In general, C (the language of *atomic constraints*) is a subset of a first order language L (the language of constraints). Let us start by defining L itself.

Definition 1 (Language of constraints.) *Let V and Λ be two disjoint denumerable sets of variable symbols. Let us also fix two particular isomorphisms between Λ and $\mathbb{N}$, and between V and $\mathbb{N}$. Let $\mathit{Vars} \stackrel{\mathrm{def}}{=} V \cup \Lambda$. Let Ω and Π_C be two finite sets of operation and predicate symbols, respectively, each symbol being characterized with its* arity*. Let also Vars, Ω and Π_C be mutually disjoint. $\mathsf{L} = \mathsf{L}(\mathit{Vars}, \Omega, \Pi_C, \dots)$ is a* language of constraints *if and only if it is any first order language with equality built (by means of standard constructions, possibly with connectives and quantifiers) over the given sets of symbols.*

(The extra-set of variables Λ allows us to simplify the following treatment. We will stipulate that the *heads* of clauses can only contain variable symbols drawn from Λ.)

Now, a CLP language can impose restrictions on the form of constraints that may actually appear in programs. However these restrictions must not destroy too much of the language's expressivity.

Definition 2 (Atomic constraint.) *Given a language of constraints L, any subset C of L is a* language of atomic constraints *if and only if*

1. *it is closed under variable renaming; and*

2. it contains all the equalities '$X = t$', for each $X \in$ Vars and each term t built over Ω and Vars.

Before introducing the full syntax of CLP programs a few remarks about notation are in order. We will denote program variables by means of capital letters (X, Y, ...). Tuples of distinct variable will be denoted by $\bar{X}$, $\bar{Y}$, and so forth. Tuples are always assumed to be of the right cardinality, e.g., if p is a predicate symbol of arity n and we write $p(\bar{X})$, then $\bar{X}$ is an n-tuple. Special tuples denoted by $\vec{\Lambda}$, denoting initial finite segments of Λ, will also be used (recall that we have fixed a total ordering on Λ). In the above hypotheses, by writing $p(\vec{\Lambda})$ we understand that $\vec{\Lambda}$ denotes the n-tuple consisting of the first n variable symbols in Λ. We will also abuse the notation occasionally by treating a tuple as the set of its components.

We can now introduce the notion of CLP(C) program. Notice that, not to exclude in advance any real CLP language, we carefully avoid making any compromising assumption the could preclude the adoption of particular computation and search rules.

Definition 3 (CLP program.) *Let C be a language of atomic constraints with distinguished variable symbols in Λ. Let Π_P be a finite set of predicate symbols, disjoint from the symbols used in C. We will denote by A_P the set of atoms over Π_P, that is*

$$\mathsf{A}_P \stackrel{\text{def}}{=} \bigl\{\, q(\bar{X}) \bigm| q \in \Pi_P, \bar{X} \in \mathit{Vars}^\star \,\bigr\}.$$

A CLP(C) *program P over Π_P is a finite sequence of* rules (or clauses) *of the form*

$$p(\vec{\Lambda}) :- \langle b_1, \dots, b_k \rangle, \quad \text{with } p \in \Pi_P \text{ and } k \geq 0,$$

where, for $1 \leq i \leq k$, $b_i \in \mathsf{C} \cup \mathsf{A}_P$ and $\mathit{vars}(b_i) \cap \Lambda \subseteq \vec{\Lambda}$. $p(\vec{\Lambda})$ is called the head *of the rule, whereas $\langle b_1, \dots, b_k \rangle$ is the* body.

The syntax of any CLP language can be defined in such a way, by augmenting the first order language on which it is based with the set of distinguished variable symbols Λ and transforming each program along the lines of Definition 3 by means of standard techniques. This transformation is always possible by virtue of Definition 2. This *normalization* of programs has the property that predicate symbols are always applied to tuples of distinct variables. Further, all the heads of the rules defining a program predicate p/n have the same variables in the same positions. Observe that the body B of a clause $p(\vec{\Lambda}) :- B$ is a sequence, that is an element of $(\mathsf{C} \cup \mathsf{A}_P)^\star$. As programs themselves are sequences, the semantic constructions will be free to take into account the selection and search rules used in real languages. So far for the syntax, we now examine the (possibly non-standard) semantics of CLP languages.

3.2.2 Non-Standard Semantics for CLP

Here we start from very basic facts. We recognize the existence of four different activities in the execution, and thus in the analysis, of constraint logic programs:

1. different execution paths are explored;
2. along any path, constraints are accumulated in the so-called *constraint store*;
3. the constraint store is recursively subdivided into parts. The activity of imposing restrictions in the way different parts can interact is usually called *hiding.*
4. Pieces of information (parameters) are passed between program rules.

For different (non-standard) semantics, and at different levels of abstraction the degree of correlation among these activities may vary considerably. For instance, while in a "standard" semantic execution paths are explored depending on the satisfiability of the accumulated constraints, in the case of "non-standard" semantic constructions these activities can be somewhat unrelated. Perhaps because the satisfiability check does not make sense in the non-standard interpretation (think about groundness analysis), or because it can be safely moved outside the construction process.

The way we define domains of interpretation for CLP languages is clearly highly dependent on the application we have in mind. Here we give a general description of what are the characteristics of a quite wide class of domains that covers many non-standard semantics for CLP. Then we will obtain our specific class of domains by imposing restrictions on the general scheme. This is important for a full understanding of the hypotheses that are behind our approach.

Suppose we are interested in some properties of programs' *terminating* computations. To this minimal requirement corresponds the following class of interpretation domains.

Definition 4 (Ordered domain.) *An* ordered domain of interpretation *for a* CLP *language is a structure:*

$$\bar{\mathcal{D}} \stackrel{\mathrm{def}}{=} \bigl\langle \mathcal{D}, \sqsubseteq, \preceq, \otimes, \oplus, \bot, \mathbf{1}, \{\bar{\exists}_{\bar{X}}\}_{\bar{X}\in \mathit{Vars}^\star}, \{\mathrm{d}_{\bar{X}\bar{Y}}\}_{\bar{X},\bar{Y}\in \mathit{Vars}^\star} \bigr\rangle,$$

where

- $\mathcal{D}$ *is a set of (not better specified) properties,*
- $\sqsubseteq\, \subseteq \mathcal{D}\times\mathcal{D}$ *and* $\preceq\, \subseteq \mathcal{D}\times\mathcal{D}$ *are relations over* $\mathcal{D}$*,*
- $\otimes\colon \mathcal{D}\times\mathcal{D}\to\mathcal{D}$ *and* $\oplus\colon \mathcal{D}\times\mathcal{D}\to\mathcal{D}$ *are binary operators,*

- $\{\overline{\exists}_{\bar{X}}\}_{\bar{X}\in \mathit{Vars}^\star}$ *is a family of unary operators,*
- $\bot$, $\mathbf{1}$, *and* $\{\mathrm{d}_{\bar{X}\bar{Y}}\}_{\bar{X},\bar{Y}\in \mathit{Vars}^\star}$ *are distinguished elements of* $\mathcal{D}$.

Furthermore

O_1. $\langle \mathcal{D}, \oplus, \bot\rangle$ *is a monoid;*

O_2. $\langle \mathcal{D}, \otimes, \mathbf{1}\rangle$ *is a monoid;*

O_3. $\bot$ *is an* annihilator *for* $\otimes$, *i.e., for each* $D \in \mathcal{D}$, $\bot \otimes D = D \otimes \bot = \bot$*;*

O_4. $\langle \mathcal{D}, \sqsubseteq\rangle$ *is a partial order with minimum element* $\bot$*;*

O_5. $\langle \mathcal{D}, \preceq\rangle$ *is a partial order with maximum element* $\mathbf{1}$.

The $\oplus$ operator models the merging of information coming from different execution paths. The monoidal structure ensures that the composition process is "insensitive to grouping" and that the empty path (which intuitively corresponds to $\bot$) is ignored. Not requiring more than the monoidal structure (e.g., commutativity) means that we have not lost already the ability of taking into account the order in which paths are tried.

The $\otimes$ operator models the constraint accumulation process. Similar considerations apply here as in the case of $\oplus$, with the monoidal unit $\mathbf{1}$ standing, intuitively, for the empty constraint store: the one containing no information at all.

The $\overline{\exists}_{\bar{X}}$ operators represent the *hiding* process: any variable $X \notin \bar{X}$ appearing in the scope of $\overline{\exists}_{\bar{X}}$ is isolated (hidden) from other occurrences of X outside the scope. The "complement sign" that appears on top of $\overline{\exists}_{\bar{X}}$ signifies that we formalize hiding in a dual way with respect to traditional approaches [SRP91, GDL95]. Notice that the exact interpretation of the $\overline{\exists}_{\bar{X}}$ operators remains relatively free. They might stand for projection onto the "variables of interest" $\bar{X}$ (this will be our case), or renaming to fresh variables (e.g., to model what happens in meta-interpreters: any variable not in $\bar{X}$ that appears in the scope is given a fresh name), or for very low-level operations (e.g., creation and disposal of environment-frames in machine-level traces: one slot for each "local variable", that is, each X in scope that is not in $\bar{X}$).

Similar considerations apply also to the distinguished elements representing parameter passing. In our present application the so-called *diagonal elements* $\mathrm{d}_{\bar{X}\bar{Y}}$ will represent, roughly speaking, the fact that the tuples of variables $\bar{X}$ and $\bar{Y}$ are tightly correlated with respect to the properties of interest. When dealing with WAM traces, they could represent, for instance, the multiple assignment of the registers $\bar{X}$ to the *argument registers* $\bar{Y}$ [AK91].

Thus far we have given a mathematical dress to the ingredients that, in our intuition, constitute the computations of CLP programs. We now put ourselves in an abstract interpretation setting.

The non-standard semantics we are interested in will, in general, be captured by some *property transformer* $\Phi_P : \mathcal{D} \to \mathcal{D}$, where P is the program at hand. The purpose of the Φ_P operator, at an intuitive level, is that of exploring the (partial) computation paths of P. It will be designed in such a way that iterated applications of Φ_P to the "null path" $\perp$ corresponds to the iterative re-construction of *all* the computation paths of P. This exploration process is usually monotonic, in that the "knowledge" about the possible paths increases between successive iterates. This relationship between successive iterates is captured (in a program independent way) by the relation $\sqsubseteq$, which is referred to as the *computational order* of the interpretation domain. When this is the case, Φ_P is conceived so that any of its post-fixpoints (with respect to $\sqsubseteq$) is a property taking into account all the the computational possibilities of P and possibly more.

The relation $\preceq$, instead, specifies the relative precision of program properties. $D_1 \preceq D_2$ means that "D_1 is more precise than D_2". In other words, every set of computations that enjoys property D_1 enjoys also property D_2. In the framework of abstract interpretation, $\preceq$ is referred to as the *approximation ordering* of the domain. Notice that the distinction between approximation ordering and computational ordering is important: one has to do with the precision of properties, the other holds between successive iterates of the property transformers. In principle, they could be totally unrelated [CC92b]. (It might be observed that it is not sensible to talk about a computational ordering of the domain without the explicit reference to a particular class of properties' transformers. However, since the domain of interpretation and the transformers are usually chosen at the same time, we found it more convenient to present the computational ordering as part of the domain.)

The objective of the game is now to derive, for any program P, a property that holds for all its computation paths. Generally, depending on the feasibility of the goal and on various other considerations, this will be given by either the least fixpoint or a post-fixpoint (with respect to $\sqsubseteq$) of Φ_P or by some approximations (with respect to $\preceq$) of them. Observe that the choice of a fixpoint presentation for the non-standard semantics is not restrictive in any way [CC95].

A treatment of abstract interpretation for CLP languages in these very general terms is well outside the scope of this chapter. Thus we start imposing restrictions on the notion of interpretation domain given in Definition 4 so as to obtain a specialized version that is suitable for our purposes. This makes clear what is the class of properties that are captured by the hierarchy of domains that will be presented later.

First of all, we are neither interested in the order in which computations are taken, nor in their multiplicities.[2] This amounts to

O'_1. $\langle \mathcal{D}, \oplus, \bot \rangle$ is a *commutative* and *idempotent* monoid.

Since we are interested in characterizing only the (possibly) successful computations, we disregard (finitely) failed computation paths. We thus postulate the existence in $\mathcal{D}$ of a property characterizing failed computations, the fact that extending a failed computation yields a failed computation, and the fact that such computations have to be ignored. More formally,

O_{s_1}. $\mathbf{0} \in \mathcal{D}$,

O_{s_2}. $\mathbf{0}$ is an *annihilator* for $\otimes$,

O_{s_3}. $\mathbf{0}$ is a *unit* for $\oplus$.

Then, we focus on *monotonic* properties, that is, those that are preserved as computation progresses. Since computations progress by *adding* constraints, this means that

O_m. for each $D_1, D_2 \in \mathcal{D}$, $D_1 \otimes D_2 \preceq D_1$ and $D_1 \otimes D_2 \preceq D_2$, must hold.

Further, we restrict our interest to *logical* properties. This means that $\otimes$ is interpreted as logical conjunction. Since, by the previous discussion, $\preceq$ is always interpreted as logical implication, we must have that

O'_2. $\langle \mathcal{D}, \otimes, \mathbf{1} \rangle$ is a *commutative* and *idempotent* monoid.

O_l. for each $D_1, D_2 \in \mathcal{D}$, if $D_1 \preceq D_2$ then $D_1 \preceq D_1 \otimes D_2$.

Finally, since $D_1 \oplus D_2$ must hold for all the paths characterized by D_1 and all the ones characterized by D_2, we enforce

O_p. $D_1 \preceq D_1 \oplus D_2$ and $D_2 \preceq D_1 \oplus D_2$, for each $D_1, D_2 \in \mathcal{D}$.

These assumptions bring us to interpretation domains that are much simpler than those of Definition 4. By O_m and O_l we have that $D_1 \preceq D_2$ holds if and only if $D_1 \otimes D_2 = D_1$. Since $\bot$ and $\mathbf{0}$ are both minimal elements with respect to the ordering $\preceq$ they must coincide. Also, by O_p and the previous discussion, the computational ordering $\sqsubseteq$ must be included in $\preceq$. In fact, in the interpretation domains we will present later, they coincide.

Now the question is: how do we represent the properties of interest? A simple, but far reaching answer was first given in [CF92]: we can represent properties by means of constraints. This opens up the possibility of computing non-standard semantics of CLP, and, in particular, abstract

[2]We refer the reader to [LM95] for an understanding of why and how the ordering can be taken into account in the interpretation domain.

interpretations, *within* the CLP framework. The result of the abstract interpretation of a CLP program P is obtained by "executing" (in a finite way) another CLP program P', strongly related to P, over a non-standard domain. Intuitively, this is done by replacing the standard constraint domain with one suitable for expressing the desired information. This possibility led to the idea of a generalized semantics for CLP programs, proposed in [GDL95]. A generalized semantics is parameterized over the (possibly non-standard) constraint system that constitutes the domain of the computation. The advantages of this approach are that:

1. different instances of CLP can be used to define non-standard semantics for constraint logic programs;

2. the semantics of these instances are all captured within a unified algebraic framework; in particular,

3. many relevant abstract interpretations of CLP programs can be formalized inside the CLP paradigm; and

4. it is easier to correlate any two non-standard semantics, when they are instances of the same parametric construction.

The next section is devoted to the class of ordered domains outlined above.

3.2.3 Constraint Systems

Since we aim at a pervasive treatment, we would like to avoid talking too much about what a constraint is. However, we cannot overlook some basic facts on the relationship between constraints and program variables. The purpose of constraints is, roughly speaking, to restrict the range of values variables can take. For our present objectives the following definition suffices.

Definition 5 (Constraint.) *Let $\mathcal{L}$ be any first-order language with variable symbols in Vars. The class of* constraints over $\mathcal{L}$ *is inductively defined as follows:*

1. *a well-formed formula of $\mathcal{L}$ is a constraint over $\mathcal{L}$;*

2. *any set of constraints over $\mathcal{L}$ is a constraint over $\mathcal{L}$;*

3. *any (meta-level) predicate μ/n applied to n constraints over $\mathcal{L}$ is a constraint over $\mathcal{L}$;*

4. *nothing is a constraint over $\mathcal{L}$ if not by virtue of points 1, 2, or 3.*

For a constraint C, it is natural to ask which variables it talks about. We denote this set of variables by $FV(C)$. Notice that FV stands for *free variables*: this is because there are domains where properties admit bounded occurrences of variables (see Section 3.3.3 on page 47 for an example). Another natural thing is the following: having a constraint C which describes a tuple of variables $\bar{X}$, we would like to say that this same description applies to a different tuple of variables, $\bar{Y}$. We thus introduce the notion of *renaming* and the notation $C[\bar{Y}/\bar{X}]$. More formally, on the relationship between constraints and variables, we can reason inductively as follows.

Definition 6 (All variables, free variables, and renamings.) *With reference to* Definition 5, *if c is a constraint by virtue of point 1 then the notions of* variables of c *and of* free variables of c *are assumed as primitive. These sets of variables are denoted, respectively, by $vars(c)$ and $FV(c)$. An invertible mapping from and to variable symbols that is the identity almost everywhere is called* renaming. *We will use the notation $[\bar{Y}/\bar{X}]$ for renamings, where $\bar{Y}$ and $\bar{X}$ are disjoint tuples of distinct variables. The renaming $[Y/X]$ has no effect on c if $X \notin FV(c)$, whereas variables' capture is avoided by consistent renaming of bound variables. Besides that, for each well-formed formula of $\mathcal{L}$, the constraint $c[\bar{Y}/\bar{X}]$ is assumed as defined.*

If C is a constraint because of point 2 then[3]

$$vars(C) \stackrel{\text{def}}{=} \bigcup_{c \in C} vars(c) \tag{3.1}$$

and

$$FV(C) \stackrel{\text{def}}{=} \bigl\{\, X \in \mathit{Vars} \bigm| \exists c \in C \,.\, \exists Y \in \mathit{Vars} \,.\, c[Y/X] \notin C \,\bigr\}. \tag{3.2}$$

Notice that the definition of $FV(C)$ implicitly depends on $FV(c)$ for $c \in C$. In fact, if $X \notin FV(c)$ then $c[Y/X] = c$ and the condition $c[Y/X] \notin C$ is not satisfied in (3.2) *for that particular choice of c and Y. With the definitions given by* (3.1) *and* (3.2) *the notion of renaming for C is extended in the expected way. The application of a renaming to C is defined element-wise.*

If $\mu(C_1, \ldots, C_n)$ is a constraint by virtue of point 3 then the above notions are extended as they would be in any first-order language, namely, by treating μ as one of the usual logical connectives.

Renamings will always applied carefully so that, when we write $C[\bar{Y}/\bar{X}]$, it is ensured that $FV(C) \cap \bar{Y} = \varnothing$. We will emphasize this fact by saying that $[\bar{Y}/\bar{X}]$ is a renaming for C.

All the members of our hierarchy of domains will turn out to be *constraint systems* in the precise sense stated by the following definition.

[3]This definition of FV is an adaptation of the one of *dependent variables* given by Saraswat [Sar92, Definition 2.3].

Definition 7 (Constraint system.) *Any algebra* $\bar{\mathcal{D}}$ *of the form*

$$\left\langle \mathcal{D}, \otimes, \oplus, \mathbf{0}, \mathbf{1}, \{\bar{\exists}_{\bar{X}}\}_{\bar{X} \in \mathit{Vars}^{\star}}, \{\mathrm{d}_{\bar{X}\bar{Y}}\}_{\bar{X},\bar{Y} \in \mathit{Vars}^{\star}} \right\rangle$$

is a constraint system *if and only if it satisfies the following conditions:*

G_0. *$\mathcal{D}$ is a set of constraints;*

G_1. *$\langle \mathcal{D}, \otimes, \mathbf{1} \rangle$ is a commutative and idempotent monoid;*

G_2. *$\langle \mathcal{D}, \oplus, \mathbf{0} \rangle$ is a commutative and idempotent monoid;*

G_3. *$\mathbf{0}$ is an* annihilator *for $\otimes$, i.e., for each $C \in \mathcal{D}$, $C \otimes \mathbf{0} = \mathbf{0}$;*

G_4. *for each $C_1, C_2 \in \mathcal{D}$, the absorption law $C_1 \otimes (C_1 \oplus C_2) = C_1$ holds;*

G_5. *for each $\bar{X} \in \mathit{Vars}^{\star}$ and $C \in \mathcal{D}$ we have $FV\bigl(\bar{\exists}_{\bar{X}}\, C\bigr) \subseteq \bar{X}$.*

A constraint system induces the relation $\vdash \subseteq \mathcal{D} \times \mathcal{D}$ given, for each $C_1, C_2 \in \mathcal{D}$, by

$$C_1 \vdash C_2 \quad \overset{\text{def}}{\iff} \quad C_1 \otimes C_2 = C_1. \tag{3.3}$$

The relation $\vdash$ is referred to as the approximation ordering *of the constraint system. The notation $C_1 \Vdash C_2$ is a convenient shorthand for $(C_1 \vdash C_2) \wedge (C_1 \neq C_2)$.*

In what follows we will feel free to drop the quantifiers from the notation of the families of projection operators and diagonal elements.

Condition G_4 can be restated as

$$C_1 \vdash C_1 \oplus C_2 \quad \text{and} \quad C_2 \vdash C_1 \oplus C_2.$$

In this form it clearly stands for the correctness of the merge operation, characterizing it as a (not necessarily least) upper bound operator with respect to the approximation ordering.

Hypothesis 8 *Constraint systems must satisfy some other (very technical) conditions related to how they deal with variables. For instance, they do not invent new free variables:*

$$FV(C_1 \otimes C_2) \subseteq FV(C_1) \cup FV(C_2),$$

and similarly for the merge operator. The operators are also generic *in that they are insensible to variable names. This implies that, if $[\bar{Y}/\bar{X}]$ is a renaming for both C_1 and C_2, then*

$$(C_1 \otimes C_2)[\bar{Y}/\bar{X}] = C_1[\bar{Y}/\bar{X}] \otimes C_2[\bar{Y}/\bar{X}].$$

In particular, if we have also $C_1 \vdash C_2$, then $C_1[\bar{Y}/\bar{X}] \vdash C_2[\bar{Y}/\bar{X}]$. All these overwhelmingly reasonable requirements will be taken for granted.

Constraint systems enjoy several properties.

Proposition 9 (Properties of c.s.) *Any constraint system satisfies the following properties, for each $C, C_1, C_2 \in \mathcal{D}$:*

1. $\langle \mathcal{D}, \vdash, \otimes, \mathbf{0}, \mathbf{1} \rangle$ *is a bounded meet-semilattice;*

2. $C \oplus \mathbf{1} = \mathbf{1}$*;*

3. $C_1 \vdash C_1 \oplus (C_1 \otimes C_2)$*;*

4. $C_1 \oplus C_2 = C_2 \implies C_1 \vdash C_2$*.*

Proof

1. First of all, axiom G_1 implies that $\vdash$ is a partial order. Reflexivity, transitivity, and antisymmetry of $\vdash$ come, respectively, from idempotency, associativity, and commutativity of $\otimes$. The ordering definition (3.3) ensures that $\otimes$ is the greatest lower bound operator with respect to $\vdash$. G_3 and G_1 imply that $\mathbf{0}$ and $\mathbf{1}$ are the minimum and maximum elements with respect to $\vdash$.

2. Notice that, for each $C \in \mathcal{D}$, $\mathbf{1} \otimes (\mathbf{1} \oplus C) = \mathbf{1}$ is an instance of G_4, which, by G_1 becomes $\mathbf{1} \oplus C = \mathbf{1}$.

3. By G_4 we have $C_1 \otimes \bigl(C_1 \oplus (C_1 \otimes C_2)\bigr) = C_1$, whence the thesis.

4. Suppose $C_1 \oplus C_2 = C_2$. Then G_4 implies

$$C_1 \otimes C_2 = C_1 \otimes (C_1 \oplus C_2) = C_1,$$

   thus $C_1 \vdash C_2$. □

Observe that $\langle \mathcal{D}, \otimes, \oplus, \mathbf{0}, \mathbf{1} \rangle$, in general, is not a lattice. Both $\otimes$ and $\oplus$ are associative, commutative, and idempotent, but, as stated above, while one of the absorption laws holds (axiom G_4 of Definition 7), only one direction of the dual law is generally valid (property (3) of Proposition 9). In particular, $\oplus$ might be not component-wise monotone with respect to $\vdash$: for each $C_1, C_2 \in \mathcal{D}$, from the obvious relations $C_1 \otimes C_2 \vdash C_2$ and $C_2 \vdash C_2$ we would get $(C_1 \otimes C_2) \oplus C_2 \vdash C_2$. Observe also that $\oplus$ does not distribute, in general, over $\otimes$, as this would imply the equivalence of the two absorption laws. It must be stressed that this flexibility of the $\oplus$ operator is necessary in order to obtain a framework that is general enough to capture several existing analysis domains.

So far for generic constraint systems, we consider now some strengthenings of Definition 7.

Definition 10 (Closed c.s.) *A constraint system is said to be* closed *if and only if*

G_c. *for each family* $\{C_i \in \mathcal{D}\}_{i \in \mathbb{N}}$, *the element*

$$\bigoplus_{i \in \mathbb{N}} C_i \stackrel{\text{def}}{=} C_1 \oplus C_2 \oplus \cdots$$

> *exists and is unique in* $\mathcal{D}$*; moreover, associativity, commutativity, and idempotence of* $\oplus$ *apply to denumerable as well as to finite families of operands.*

So, the operation of merging together the information coming from all the computation paths always makes sense in a closed constraint system. Notice however that property G_c is only necessary when the semantic construction requires it. This will never happen when considering "abstract semantic constructions" formalizing data-flow analyses (which are finite in nature). In these cases the idea of merging infinitely many pieces of information is nonsense in itself. Closedness will instead be required for the constraint systems intended to capture "concrete" program semantics.

Another optional property of constraint systems that we will mention is distributivity.

Definition 11 (Distributive c.s.) *Consider the following conditions:*

G_d. $\langle \mathcal{D}, \otimes, \oplus, \mathbf{0}, \mathbf{1} \rangle$ *is a distributive lattice;*

G_D. *for each* $C \in \mathcal{D}$ *and each family* $\{C_i \in \mathcal{D}\}_{i \in \mathbb{N}}$ *such that the element* $\bigoplus_{i \in \mathbb{N}} C_i$ *exists,*

$$C \otimes \left(\bigoplus_{i \in \mathbb{N}} C_i \right) = \bigoplus_{i \in \mathbb{N}} (C \otimes C_i).$$

A constraint system is said to be distributive *if it satisfies* G_d*. If it satisfies the stronger condition* G_D *then it is called* completely distributive.

Distributivity is useful for proving the equivalence of different abstract semantics constructions used for data-flow analysis. Complete distributivity is required for proving that a concrete semantics corresponds to the operational model of the language [GDL95]. Observe that closed and completely distributive constraint systems are instances of the *closed semi-rings* used in [GDL95].

Definition 12 (Noetherian c.s.) *A constraint system is called* Noetherian *if it satisfies the* ascending chain condition*:*

G_N. *in* $\mathcal{D}$ *every strictly ascending chain,* $C_0 \Vdash C_1 \Vdash C_2 \Vdash \cdots$, *is finite.*

For the abstract semantics constructions we will make use of another class of operators over constraints. These operators were introduced in [CC77] and called *widenings*.

Definition 13 (Widening.) [CC77] *Given a constraint system $\bar{\mathcal{D}}$, a binary operator $\nabla \colon \mathcal{D} \to \mathcal{D}$ is called a* widening for $\bar{\mathcal{D}}$ *if*

W_1. *for each $C_1, C_2 \in \mathcal{D}$ we have $C_1 \vdash C_1 \nabla C_2$ and $C_2 \vdash C_1 \nabla C_2$;*

W_2. *for each increasing chain $C_0 \vdash C_1 \vdash C_2 \vdash \cdots$, the sequence given by $C'_0 \overset{\mathrm{def}}{=} C_0$ and, for $n \geq 1$, $C'_n \overset{\mathrm{def}}{=} C'_{n-1} \nabla C_n$, is stationary after some $k \in \mathbb{N}$.*

Widenings allow to define *convergence acceleration methods* that ensure termination of the "abstract interpreter". However, even when termination is granted anyway (e.g., when the constraint system is Noetherian), these methods are often crucial for achieving *rapid* termination, that is, for obtaining usable data-flow analyzers. More sophisticated methods for convergence acceleration exist. They employ also *narrowing* operators and *families of widening operators* (see [CC92c, CC92b]).

3.2.4 Generalized Semantics

In a generalized semantics setting, the first thing to do is to provide atomic constraints with an interpretation on the chosen constraint system. Suppose that we are interested in deriving information about just two kind of program points: clause's entries and clause's successful exits. In a data-flow analysis setting (where this is often the case) that is to say that we want to derive *call-patterns* and *success-patterns*. In other words, for each clause we want to derive properties of the constraint store that are valid

- whenever the clause is invoked (call-patterns), or
- whenever a computation starting with the invocation of the clause terminates with success (success-patterns).

Observe that call-patterns depend on the ordering of atoms in the body of clauses and on the selection rule employed. By means of program transformations similar to the *magic* one [CD93, DR94] we can obtain the call-patterns of the original program (with respect to the selection rule employed) as success-patterns of the transformed one. These transformations, in fact, besides modifying the clauses of the original program, introduce new clauses that characterize the conditions under which the original clauses are invoked. In the transformed program the ordering of atoms in the clause's bodies is no longer important. Notice that the technique proposed by Debray and Ramakrishnan [DR94], while restricted to logic programs, is more sophisticated than usual transformation approaches, and preserves the connection between call and success patterns. Our generalization of the work of Debray and Ramakrishnan, which is general enough to accommodate the entire CLP framework, will be presented in Chapter 7.

For these reasons we will consider only one kind of program points: clause's exits. Furthermore, in our domains the operation capturing constraint composition is associative, commutative, and idempotent. This means that we can assume without prejudice that all the clauses are of the form

$$p(\vec{\Lambda}) :- \{c_1, \ldots, c_n\} \,\Box\, \{b_1, \ldots, b_k\},$$

where $\{c_1, \ldots, c_n\}$ is a set of atomic constraints, and $\{b_1, \ldots, b_k\}$ is a set of atoms. All the other restrictions imposed by Definition 3 must continue to hold. We now must associate a meaning to the finite sets of atomic constraints that occur in clauses.

Definition 14 (Constraint interpretation.) *Given a language* C *of atomic constraints and a domain of interpretation* $\bar{\mathcal{D}}$*, any computable function* $[\![\cdot]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}} \colon \wp_{\mathrm{f}}(\mathsf{C}) \to \mathcal{D}$ *is a* constraint interpretation of C in $\bar{\mathcal{D}}$.

Then usually one considers, instead of the *syntactic* program P, its *semantic* version over the domain $\bar{\mathcal{D}}$, obtained by interpreting the atomic constraints of clauses through $[\![\cdot]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}}$. Recall that we denote by $P[r]$ the r-th clause of P. for $r = 1, \ldots, \# P$.

Definition 15 (Generalized program.) *When* $\bar{\mathcal{D}}$ *is a constraint system, a* CLP($\bar{\mathcal{D}}$) program *is a sequence of Horn-like formulas of the form*

$$p(\vec{\Lambda}) :- C \,\Box\, \{b_1, \ldots, b_k\},$$

where $C \in \bar{\mathcal{D}}$ *is finitely representable. Given a* CLP(C) *program* P *and a constraint interpretation* $[\![\cdot]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}}$*, the* CLP($\bar{\mathcal{D}}$) *program* $[\![P]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}}$ *is given, for each* $r = 1, \ldots, \# P$*, by*

$$[\![P]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}}[r] \equiv p(\vec{\Lambda}) :- [\![C]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}} \,\Box\, B \quad \overset{\text{def}}{\iff} \quad P[r] \equiv p(\vec{\Lambda}) :- C \,\Box\, B.$$

An interpretation for a program P over a constraint system $\bar{\mathcal{D}}$ is a function from its program points (one for each clause) to $\mathcal{D}$. Notice that we deviate from standard approaches in that our interpretations associate a meaning to *each distinct program rule*, and not to each distinct program predicate.

Definition 16 (Interpretation.) *Let* $\bar{\mathcal{D}}$ *be a constraint system, and* P *a* CLP($\bar{\mathcal{D}}$) *program. An* interpretation for P over $\bar{\mathcal{D}}$ *is any element of*

$$\mathcal{I}^{\bar{\mathcal{D}}}_P \stackrel{\text{def}}{=} \{1, \ldots, \# P\} \to \mathcal{D}.$$

All the operations and relations over $\bar{\mathcal{D}}$ *are extended pointwise to* $\mathcal{I}^{\bar{\mathcal{D}}}_P$*. In particular* $\mathcal{I}^{\bar{\mathcal{D}}}_P$ *is partially ordered by the lifting of* $\vdash$*, i.e., for each* $I_1, I_2 \in \mathcal{I}^{\bar{\mathcal{D}}}_P$*,*

$$I_1 \vdash I_2 \quad \overset{\text{def}}{\iff} \quad \forall r \in \{1, \ldots, \# P\} : I_1(r) \vdash I_2(r).$$

Interpretations will be represented by means of function graphs. It is straightforward to show that all the interesting properties of constraint systems lift smoothly to interpretations.

We are left with the choice of the semantics construction, that is, of the "interpretation transformer". For the purpose of this work we choose a bottom-up construction expressed by a variant of the usual *immediate consequence operator* T_P, taking an interpretation and returning a new interpretation. More specifically, this transformer takes the information arising from a set of computation paths and gives back the information relative to those computation paths that can be obtained by composing in all possible ways, as specified by the program P, the computation paths characterized by the input.

Recall that our set of program variables is $\mathit{Vars} = \Lambda \cup V$, where Λ and V are totally ordered. For $\vec{\Lambda} \in \Lambda^\star$ and $W \subseteq_{\mathrm{f}} \mathit{Vars}$, we denote by $\bar{Y} \ll_{\vec{\Lambda}} W$ the fact that, with respect to the ordering of V, $\bar{Y}$ is a tuple of distinct consecutive variables in V such that $\#\,\bar{Y} = \#\,\vec{\Lambda}$ and the first element of $\bar{Y}$ immediately follows the greatest variable in W.

Definition 17 (Interpretation transformer.) *Let $\bar{\mathcal{D}}$ be a constraint system and P be a* $\mathrm{CLP}(\bar{\mathcal{D}})$ *program. The operator induced by P over $\mathcal{I}_P^{\bar{\mathcal{D}}}$, $T_P^{\bar{\mathcal{D}}} \colon \mathcal{I}_P^{\bar{\mathcal{D}}} \to \mathcal{I}_P^{\bar{\mathcal{D}}}$, is*

$$T_P^{\bar{\mathcal{D}}}(I) \stackrel{\mathrm{def}}{=} \Big\{ \big(r, T_P^{\bar{\mathcal{D}}}(r, I)\big) \Bigm| 1 \leq r \leq \#\,P \Big\},$$

where $T_P^{\bar{\mathcal{D}}} \colon \{1, \ldots, \#\,P\} \times \mathcal{I}_P^{\bar{\mathcal{D}}} \to \mathcal{D}$ is given by

$$T_P^{\bar{\mathcal{D}}}(r, I) \stackrel{\mathrm{def}}{=} \bigoplus \left\{ \exists\!\!\!\exists_{\vec{\Lambda}}\, \tilde{C} \;\middle|\; \begin{array}{l} P[r] \equiv p(\vec{\Lambda}) \mathrel{:-} C \\ \qquad \Box\, \{p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\} \\ \textit{for each } i = 1, \ldots, n\textit{:} \\ \quad P[r_i] \equiv p_i(\vec{\Lambda}_i) \mathrel{:-} C_i \,\Box\, B_{r_i} \\ \quad \bar{Y}_i \ll_{\vec{\Lambda}_i} FV\big(P[r]\big) \cup \bigcup_{k=1}^{i-1} \bar{Y}_k \\ \quad \tilde{C}_i = I(r_i)[\bar{Y}_i/\vec{\Lambda}_i] \\ \quad C'_i = \mathrm{d}_{\bar{X}_i \bar{Y}_i} \otimes \tilde{C}_i \\ \tilde{C} = C \otimes C'_1 \otimes \cdots \otimes C'_n \end{array} \right\}.$$

Notice that, in this construction, the merge operator is applied only to finite sets of operands. In summary, once we have fixed the constraint domain $\bar{\mathcal{D}}$ and the interpretation of atomic constraints $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}}$, the meaning of a $\mathrm{CLP}(\mathsf{C})$ program P over $\bar{\mathcal{D}}$ is encoded into the $T_P^{\bar{\mathcal{D}}}$ operator. Before rushing to require that $T_P^{\bar{\mathcal{D}}}$ must be continuous on the complete lattice $\bar{\mathcal{D}}$ we better have a closer look to our real needs.

3.2.5 Dealing with Non-Standard Semantics

Given our current focus on data-flow analysis of CLP programs, we consider only the typical case in this field. On one hand we have a "concrete" constraint system $\bar{\mathcal{D}}^\natural$: it must capture the properties of interest, it must ensure the existence of the least fixpoint of $T_{P^\natural}^{\bar{\mathcal{D}}^\natural}$, that is, of *the* meaning of each program P. And, of course, this meaning must correspond to the one obtained by means of the top-down construction representing the operational model of the language (namely, some kind of extended SLD-resolution). This last requirement implies, as shown in [GDL95], that $\bar{\mathcal{D}}^\natural$ must be closed and completely distributive, hence a complete lattice.

On the other hand, in data-flow analysis we have an "abstract" constraint system $\bar{\mathcal{D}}^\sharp$. Here we are much less demanding: we simply want to compute in a *finite* way an approximation of a post-fixpoint of $T_{P^\sharp}^{\bar{\mathcal{D}}^\sharp}$ (the least fixpoint might not even exist, or it might be too expensive to compute). And, of course, we need a guarantee that what we compute is a *correct* approximation of the concrete meaning. Finite computability can be ensured, in general, by using a widening operator.

Thus, in this setting, the concrete and abstract iteration sequences defining, respectively, the concrete meaning and approximations of the abstract meaning of programs are quite different.

Definition 18 (Concrete and abstract iteration sequences.) *Consider a closed and completely distributive constraint system $\bar{\mathcal{D}}^\natural$, and a CLP($\bar{\mathcal{D}}^\natural$) program $P^\natural$. The* concrete iteration sequence *for $P^\natural$ is inductively defined as follows, for all ordinals $\kappa \in \mathbb{O}$:*

$$\begin{cases} T_{P^\natural}^{\bar{\mathcal{D}}^\natural} \uparrow 0 \overset{\text{def}}{=} \mathbf{0}^\natural, \\ T_{P^\natural}^{\bar{\mathcal{D}}^\natural} \uparrow (\kappa+1) \overset{\text{def}}{=} T_{P^\natural}^{\bar{\mathcal{D}}^\natural}\bigl(T_{P^\natural}^{\bar{\mathcal{D}}^\natural} \uparrow \kappa\bigr), \\ T_{P^\natural}^{\bar{\mathcal{D}}^\natural} \uparrow \kappa \overset{\text{def}}{=} \bigoplus\nolimits^\natural_{\beta<\kappa}\bigl(T_{P^\natural}^{\bar{\mathcal{D}}^\natural} \uparrow \beta\bigr), \\ \qquad \textit{when } \kappa > 0 \textit{ is a limit ordinal.} \end{cases} \tag{3.4}$$

Let $\bar{\mathcal{D}}^\sharp$ be any constraint system, and let $\nabla^\sharp$ be a widening operator over $\bar{\mathcal{D}}^\sharp$. For a CLP($\bar{\mathcal{D}}^\sharp$) *program $P^\sharp$, the* abstract iteration sequence for $P^\sharp$ with widening $\nabla^\sharp$ *is inductively defined, for $k \in \mathbb{N}$, by the recurrence*

$$\begin{cases} T_{P^\sharp}^{\bar{\mathcal{D}}^\sharp} \Uparrow 0 \overset{\text{def}}{=} \mathbf{0}^\sharp, \\ T_{P^\sharp}^{\bar{\mathcal{D}}^\sharp} \Uparrow (k+1) \overset{\text{def}}{=} \bigl(T_{P^\sharp}^{\bar{\mathcal{D}}^\sharp} \Uparrow k\bigr) \nabla^\sharp T_{P^\sharp}^{\bar{\mathcal{D}}^\sharp}\bigl(T_{P^\sharp}^{\bar{\mathcal{D}}^\sharp} \Uparrow k\bigr). \end{cases} \tag{3.5}$$

Observe that, when $\bar{\mathcal{D}}^\sharp$ is Noetherian or when termination can be ensured in other ways, π_2 (the second projection) can be substituted for $\nabla^\sharp$ in (3.5). In these cases, indeed, the restriction of π_2 to the iterates' values is a widening operator. In other words, for Noetherian domains, the iteration sequence (eq:concrete-iteration) converges in a finite number of steps and thus can be used for data-flow analysis (despite the fact that it has been termed as "concrete").

The proof of following result is standard [GDL95].

Theorem 19 *If $\bar{\mathcal{D}}^\natural$ is a closed and completely distributive constraint system then, for each CLP($\bar{\mathcal{D}}^\natural$) program $P^\natural$ the least fixpoint of $T_{P^\natural}^{\bar{\mathcal{D}}^\natural}$ exists and is given by* $\mathrm{lfp}(T_{P^\natural}^{\bar{\mathcal{D}}^\natural}) = T_{P^\natural}^{\bar{\mathcal{D}}^\natural} \uparrow \omega$.

This theorem is generalizable to the case where $T_{P^\natural}^{\bar{\mathcal{D}}^\natural}$, which is naturally partitioned into a sequence of operators (one for each clause), is evaluated component by component *à la* Gauss-Seidel: at step $n+1$ the approximation for *one* clause is updated by evaluating the corresponding component of $T_{P^\natural}^{\bar{\mathcal{D}}^\natural}$ over the current approximations at step n. In the field of numerical analysis very strong hypotheses are required for ensuring the convergence of Gauss-Seidel iteration methods [Bag95b]. Here, instead, the continuity of $T_{P^\natural}^{\bar{\mathcal{D}}^\natural}$ ensures that each *chaotic iteration strategy* (i.e., such that no component is forgotten indefinitely) based on (3.4) converges to the least fixpoint [Cou78, CC92a].

We now come to the problem of ensuring the correctness of the analysis. We use an *abstraction correspondence* between the concrete and the abstract constraint systems, which induces an abstraction correspondence between the respective semantics [BM83, CC92b].

Definition 20 (Abstraction function.) *Let $\bar{\mathcal{D}}^\natural$ and $\bar{\mathcal{D}}^\sharp$ be two constraint systems as in* Definition 18. *A function $\alpha \colon \mathcal{D}^\natural \to \mathcal{D}^\sharp$ is an* abstraction function *of $\bar{\mathcal{D}}^\natural$ into $\bar{\mathcal{D}}^\sharp$ if and only if*

A_1. *α is a* semi-morphism*, namely, for each $C^\natural, C_1^\natural, C_2^\natural \in \mathcal{D}^\natural$ and $\bar{X}, \bar{Y} \in$ Vars*:*

$$\begin{aligned}
\alpha(C_1^\natural \otimes^\natural C_2^\natural) &\vdash^\sharp \alpha(C_1^\natural) \otimes^\sharp \alpha(C_2^\natural),\\
\alpha(C_1^\natural \oplus^\natural C_2^\natural) &\vdash^\sharp \alpha(C_1^\natural) \oplus^\sharp \alpha(C_2^\natural),\\
\alpha(\mathbf{0}^\natural) &\vdash^\sharp \mathbf{0}^\sharp,\\
\alpha\bigl(\exists^\natural_{\bar{X}}\, C^\natural\bigr) &\vdash^\sharp \exists^\sharp_{\bar{X}}\, \alpha(C^\natural),\\
\alpha\bigl(\mathrm{d}^\natural_{\bar{X}\bar{Y}}\bigr) &\vdash^\sharp \mathrm{d}^\sharp_{\bar{X}\bar{Y}};
\end{aligned}$$

A_2. *for each increasing chain $\{C_j^\natural \in \mathcal{D}^\natural\}_{j\in\mathbb{N}}$ and each $C^\sharp \in \mathcal{D}^\sharp$,*

$$\forall j \in \mathbb{N} : \alpha(C_j^\natural) \vdash^\sharp C^\sharp \quad\Longrightarrow\quad \alpha\Biggl(\bigoplus_{j\in\mathbb{N}}{}^{\natural}\, C_j^\natural\Biggr) \vdash^\sharp C^\sharp;$$

A_3. *for each $C^\natural \in \mathcal{D}^\natural$ and each renaming $[\bar{Y}/\bar{X}]$ for $C^\natural$, it happens that*

$$\alpha(C^\natural)[\bar{Y}/\bar{X}] = \alpha\bigl(C^\natural[\bar{Y}/\bar{X}]\bigr).$$

Any abstraction function $\alpha\colon \mathcal{D}^\natural \to \mathcal{D}^\sharp$ *is extended pointwise to* $\alpha\colon \mathcal{I}_{P^\natural}^{\bar{\mathcal{D}}^\natural} \to \mathcal{I}_{P^\sharp}^{\bar{\mathcal{D}}^\sharp}$*, when* $\# P^\natural = \# P^\sharp$*.*

As anticipated above, one of the beautiful things of the generalized approach is that the abstract meaning of each CLP program can be encoded into another CLP programs. We have thus an *abstract compilation* approach, where the soundness of the *compilation function* $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}^\sharp}$ is expressed, for CLP(C) programs, by the requirement $\alpha \circ [\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}^\natural} \vdash^\sharp [\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}^\sharp}$. Here we lift $\vdash^\sharp$ to functions with the meaning that $\vdash^\sharp$ applies pointwise throughout the function domain.

The following result is crucial for proving the correctness of the methodology.

Lemma 21 *Consider a* CLP(C) *program P, two constraint systems* $\bar{\mathcal{D}}^\natural$ *and* $\bar{\mathcal{D}}^\sharp$*, the constraint interpretations* $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}^\natural}$ *and* $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}^\sharp}$*, the abstraction function* $\alpha\colon \mathcal{D}^\natural \to \mathcal{D}^\sharp$ *such that* $\alpha \circ [\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}^\natural} \vdash^\sharp [\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}^\sharp}$*, the concrete program* $P^\natural = [\![P]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}^\natural}$*, and the abstract program* $P^\sharp = [\![P]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}^\sharp}$*. Then*

$$\forall I^\natural \in \mathcal{I}_{P^\natural}^{\bar{\mathcal{D}}^\natural} : \forall I^\sharp \in \mathcal{I}_{P^\sharp}^{\bar{\mathcal{D}}^\sharp} : \alpha(I^\natural) \vdash^\sharp I^\sharp \implies \alpha\bigl(T_{P^\natural}^{\bar{\mathcal{D}}^\natural}(I^\natural)\bigr) \vdash^\sharp T_{P^\sharp}^{\bar{\mathcal{D}}^\sharp}(I^\sharp).$$

Proof It is enough to show that, for each $r = 1, \ldots, \# P$ we have

$$\forall I^\natural \in \mathcal{I}_{P^\natural}^{\bar{\mathcal{D}}^\natural} : \forall I^\sharp \in \mathcal{I}_{P^\sharp}^{\bar{\mathcal{D}}^\sharp} : \qquad \alpha(I^\natural) \vdash^\sharp I^\sharp \implies \alpha\bigl(T_{P^\natural}^{\bar{\mathcal{D}}^\natural}(r, I^\natural)\bigr) \vdash^\sharp T_{P^\sharp}^{\bar{\mathcal{D}}^\sharp}(r, I^\sharp).$$

Suppose that $I^\natural$ and $I^\sharp$ satisfy the hypothesis, i.e., for each clause r' such that $1 \le r' \le \# P$ we have

$$\alpha(I^\natural(r')) \vdash^\sharp I^\sharp(r'). \tag{3.6}$$

Then, for

$$P^\natural[r] \equiv p(\vec{\Lambda}) \mathrel{:-} C^\natural \mathbin{\square} \bigl\{p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\bigr\},$$

we have

$$T_{P^\natural}^{\bar{\mathcal{D}}^\natural}(r, I^\natural) = \bigoplus\nolimits_{j \in J}^{\natural} \hat{C}_j^\natural$$

and

$$T_{P^\sharp}^{\bar{\mathcal{D}}^\sharp}(r, I^\sharp) = \bigoplus\nolimits_{j \in J}^{\sharp} \hat{C}_j^\sharp,$$

for some finite set of indices J. Moreover, we can stipulate without prejudice that, for each $j \in J$ and each $i = 1, \ldots, n$, the clause r_{ij} has been used to "resolve" against the i-th atom in the body of r in order to produce *both* $\hat{C}^{\natural}_j$ and $\hat{C}^{\sharp}_j$. Thus

$$\hat{C}^{\natural}_j = \exists\!\!\exists^{\natural}_{\vec{\Lambda}}\Bigl(C^{\natural} \otimes^{\natural} \bigotimes_{i=1}^{n}{}^{\natural}(\mathrm{d}^{\natural}_{\bar{X}_i\bar{Y}_i} \otimes^{\natural} \tilde{C}^{\natural}_{ij})\Bigr)$$

and

$$\hat{C}^{\sharp}_j = \exists\!\!\exists^{\sharp}_{\vec{\Lambda}}\Bigl(C^{\sharp} \otimes^{\sharp} \bigotimes_{i=1}^{n}{}^{\sharp}(\mathrm{d}^{\sharp}_{\bar{X}_i\bar{Y}_i} \otimes^{\sharp} \tilde{C}^{\sharp}_{ij})\Bigr),$$

where

$$\tilde{C}^{\natural}_{ij} = I^{\natural}(r_{ij})[\bar{Y}_i/\vec{\Lambda}_i]$$

and

$$\tilde{C}^{\sharp}_{ij} = I^{\sharp}(r_{ij})[\bar{Y}_i/\vec{\Lambda}_i].$$

By the hypotheses we have $\alpha(C^{\natural}) \vdash^{\sharp} C^{\sharp}$, and, using Hypothesis 8, we have also that for each $i = 1, \ldots, n$ and each $j \in J$ it is

$$\begin{aligned}
\alpha\bigl(I^{\natural}(r_{ij})[\bar{Y}_i/\vec{\Lambda}_i]\bigr) &= \alpha\bigl(I^{\natural}(r_{ij})\bigr)[\bar{Y}_i/\vec{\Lambda}_i] \\
&\qquad \text{[by } A_3 \text{ of Definition 20]} \\
&\vdash^{\sharp} I^{\sharp}(r_{ij})[\bar{Y}_i/\vec{\Lambda}_i], \\
&\qquad \text{[by (3.6) and Hypothesis 8]}
\end{aligned}$$

thus $\alpha(\tilde{C}^{\natural}_{ij}) \vdash^{\sharp} \tilde{C}^{\sharp}_{ij}$. Now we can easily conclude:

$$\begin{aligned}
\alpha\Bigl(T^{\bar{\mathcal{D}}^{\natural}}_{P^{\natural}}(r, I^{\natural})\Bigr) &= \alpha\Bigl(\bigoplus_{j\in J}{}^{\natural}\,\hat{C}^{\natural}_j\Bigr) \\
&\vdash^{\sharp} \bigoplus_{j\in J}{}^{\sharp}\,\alpha(\hat{C}^{\natural}_j) \\
&= \bigoplus_{j\in J}{}^{\sharp}\,\alpha\Bigl(\exists\!\!\exists^{\natural}_{\vec{\Lambda}}\Bigl(C^{\natural} \otimes^{\natural} \bigotimes_{i=1}^{n}{}^{\natural}(\mathrm{d}^{\natural}_{\bar{X}_i\bar{Y}_i} \otimes^{\natural} \tilde{C}^{\natural}_{ij})\Bigr)\Bigr) \\
&\vdash^{\sharp} \bigoplus_{j\in J}{}^{\sharp}\,\exists\!\!\exists^{\sharp}_{\vec{\Lambda}}\,\alpha\Bigl(C^{\natural} \otimes^{\natural} \bigotimes_{i=1}^{n}{}^{\natural}(\mathrm{d}^{\natural}_{\bar{X}_i\bar{Y}_i} \otimes^{\natural} \tilde{C}^{\natural}_{ij})\Bigr) \\
&\vdash^{\sharp} \bigoplus_{j\in J}{}^{\sharp}\,\exists\!\!\exists^{\sharp}_{\vec{\Lambda}}\Bigl(C^{\sharp} \otimes^{\sharp} \bigotimes_{i=1}^{n}{}^{\sharp}\Bigl(\alpha\bigl(\mathrm{d}^{\natural}_{\bar{X}_i\bar{Y}_i}\bigr) \otimes^{\sharp} \alpha\bigl(\tilde{C}^{\natural}_{ij}\bigr)\Bigr)\Bigr) \\
&\vdash^{\sharp} \bigoplus_{j\in J}{}^{\sharp}\,\exists\!\!\exists^{\sharp}_{\vec{\Lambda}}\Bigl(C^{\sharp} \otimes^{\sharp} \bigotimes_{i=1}^{n}{}^{\sharp}(\mathrm{d}^{\sharp}_{\bar{X}_i\bar{Y}_i} \otimes^{\sharp} \tilde{C}^{\sharp}_{ij})\Bigr) \\
&= T^{\bar{\mathcal{D}}^{\sharp}}_{P^{\sharp}}\bigl(r, I^{\sharp}\bigr). \qquad \square
\end{aligned}$$

Theorem 22 *Given a* CLP(C) *program P, two constraint systems $\bar{\mathcal{D}}^\natural$ and $\bar{\mathcal{D}}^\sharp$, the constraint interpretations $\llbracket\cdot\rrbracket_{\mathsf{C}}^{\bar{\mathcal{D}}^\natural}$ and $\llbracket\cdot\rrbracket_{\mathsf{C}}^{\bar{\mathcal{D}}^\sharp}$, the abstraction function $\alpha\colon \mathcal{D}^\natural \to \mathcal{D}^\sharp$ such that $\alpha \circ \llbracket\cdot\rrbracket_{\mathsf{C}}^{\bar{\mathcal{D}}^\natural} \vdash^\sharp \llbracket\cdot\rrbracket_{\mathsf{C}}^{\bar{\mathcal{D}}^\sharp}$, and the programs $P^\natural = \llbracket P\rrbracket_{\mathsf{C}}^{\bar{\mathcal{D}}^\natural}$ and $P^\sharp = \llbracket P\rrbracket_{\mathsf{C}}^{\bar{\mathcal{D}}^\sharp}$, the abstract iteration with widening* (3.5) *is eventually stable after $\ell \in \mathbb{N}$ steps and*

$$\alpha\bigl(\mathrm{lfp}(T_{P^\natural}^{\bar{\mathcal{D}}^\natural})\bigr) = \alpha\bigl(T_{P^\natural}^{\bar{\mathcal{D}}^\natural} \uparrow \omega\bigr) \vdash^\sharp \bigl(T_{P^\sharp}^{\bar{\mathcal{D}}^\sharp} \Uparrow \ell\bigr).$$

Proof A straightforward application of a theorem of Cousot and Cousot [CC92b, Proposition 6.20]. Stability of the iteration (3.5) after $\ell \in \mathbb{N}$ comes from the properties of widenings. The rest of the proof is carried on using transfinite induction: the base case is given by property A_1 of abstraction functions, non-limit ordinals are handled through Lemma 21, whereas for limit ordinals property A_2 is exploited. □

Convergence to a post-fixpoint is still ensured when a *chaotic iteration strategy with widening* is employed [Cou78].

We now describe a hierarchy of constraint systems that capture most of the analysis domains used for deriving monotonic properties of programs, as well as the "concrete" collecting semantics they abstract.

The basis is constituted by any constraint system that satisfies the conditions of Definition 7. First we show a way (which, of course, is not the only one) of defining such a constraint system. We start from a set of finite constraints, each expressing some partial information about a program execution's state (i.e., a constraint-store).

3.3 Simple Constraint Systems

A constraint system can be built starting from a set of finite constraints (or *tokens*), each expressing some partial information. We now define a notion of *simple constraint systems* (or s.c.s.), very similar to the one introduced in [SRP91], but with a *totally uninformative* token ($\top$) as in [Sco82].

Definition 23 (Simple constraint system.) *A structure of the form $\langle \mathcal{C}, \vdash, \bot, \top\rangle$ is a* simple constraint system *if $\mathcal{C}$ is a set of constraints, $\bot, \top \in \mathcal{C}$, and $\vdash \subseteq \wp_{\mathrm{f}}(\mathcal{C}) \times \mathcal{C}$ is an* entailment relation *such that, for each $C, C' \in \wp_{\mathrm{f}}(\mathcal{C})$, each $c, c' \in \mathcal{C}$, and $X, Y \in$ Vars:*

E_1. $c \in C \implies C \vdash c$;

E_2. $C \vdash \top$;

E_3. $(C \vdash c) \wedge (\forall c' \in C : C' \vdash c') \implies C' \vdash c$;

E_4. $\{\bot\} \vdash c$;

E_5. $C \vdash c \implies C[Y/X] \vdash c[Y/X]$.

The '$\vdash$' symbol is overloaded to denote also the extended relation $\vdash \subseteq \wp(\mathcal{C}) \times \wp(\mathcal{C})$ *such that, for each* $C, C' \in \wp(\mathcal{C})$,

$$C \vdash C' \quad \overset{\text{def}}{\iff} \quad \forall c' \in C' : \exists C'' \subseteq_{\mathrm{f}} C \,.\, C'' \vdash c'.$$

It is clear that condition E_1 implies reflexivity of $\vdash$, while condition E_3 amounts to transitivity. E_2 qualifies $\top$ as the least informative token: it will be needed just as a "marker" when the *product* of simple constraint systems will be considered (see Section 3.7 and [Sco82]). E_4 ensures that $\mathcal{C}$ is a finitely generable element (see Definition 25). Condition E_5, referred to as *genericity*, states that the entailment is insensible to variables' names.[4]

By axioms E_1 and E_3 of Definition 23 the entailment relation of a simple constraint system is a preorder. Now, instead of considering the quotient poset with respect to the induced equivalence relation, a particular choice of the equivalence classes' representatives is made: closed sets with respect to entailment. This representation is a very convenient domain-independent strong normal form for constraints.

Definition 24 (Elements.) [SRP91] *The* elements *of an s.c.s.* $\langle \mathcal{C}, \vdash, \bot, \top \rangle$ *are the* entailment-closed *subsets of* $\mathcal{C}$*, namely those* $C \subseteq \mathcal{C}$ *such that, whenever* $\exists C' \subseteq_{\mathrm{f}} C \,.\, C' \vdash c$*, then* $c \in C$*. The set of elements of* $\langle \mathcal{C}, \vdash, \bot, \top \rangle$ *is denoted by* $|\mathcal{C}|$.

The poset of elements is thus given by $\langle |\mathcal{C}|, \supseteq \rangle$. Notice that we deviate from [SRP91] in that we order our constraint systems in the dual way, as is customary in abstract interpretation.

Definition 25 (Inference map, finite elements.) *Given a simple constraint system* $\langle \mathcal{C}, \vdash, \bot, \top \rangle$*, the* inference map *of* $\langle \mathcal{C}, \vdash, \bot, \top \rangle$ *is* $\rho \colon \wp(\mathcal{C}) \to \wp(\mathcal{C})$ *given, for each* $C \subseteq \mathcal{C}$*, by*

$$\rho(C) \overset{\text{def}}{=} \{\, c \mid \exists C' \subseteq_{\mathrm{f}} C \,.\, C' \vdash c \,\}.$$

It is well-known that ρ *is a kernel operator, over the complete lattice* $\langle \wp(\mathcal{C}), \supseteq \rangle$*, whose image is* $|\mathcal{C}|$*. The image of the restriction of* ρ *onto* $\wp_{\mathrm{f}}(\mathcal{C})$ *is denoted by* $|\mathcal{C}|_0$*. Elements of* $|\mathcal{C}|_0$ *are called* finitely generated constraints *or simply* finite constraints*.*

[4] In [Sar92] a stronger notion of genericity is used, namely $C[t/X] \vdash c[t/X]$ whenever $C \vdash c$, for any term t. This is too strong for our purposes. For instance, it would force us to treat non-linear numeric constraints in the same way as linear ones in CLP($\mathcal{R}$). See Chapter 5 on this subject.

From here on we will work only with finitely generated constraints, since we are not concerned with infinite behavior of CLP programs.

In general, describing the "standard" semantics of a CLP($\mathcal{X}$) language is done as follows. Let T be the theory that corresponds to the domain $\mathcal{X}$ [JL87]. Let D be an appropriate set of formulas in the vocabulary of T closed under conjunction and existential quantification. Define $\Gamma \vdash c$ if and only if Γ entails c in the logic, with non-logical axioms T. Then $(D, \vdash)$ is the required simple constraint system. For CLP($\mathcal{H}$) (a.k.a. pure Prolog) one takes the Clark's theory of equality (see Section 3.3.3). For CLP($\mathbb{R}$) the theory RCF of real closed fields would do the job.[5] We see now some simple constraint systems.

3.3.1 The Atomic Simple Constraint System

This is probably the simplest useful example of s.c.s. The tokens include variable names. A variable name, when present in a constraint, expresses the fact that the variable has some (unspecified) property. For instance, being definitely bound to a *ground* value. In this case, X is just a shorthand for $ground(X)$. This s.c.s. is thus given by $\mathcal{C} \stackrel{\text{def}}{=} \mathit{Vars} \cup \{\bot, \top\}$ and by the smallest relation $\vdash \subseteq \wp_{\mathrm{f}}(\mathcal{C}) \times \mathcal{C}$ satisfying conditions E_1–E_5 of Definition 23. We will refer to this structure as the *atomic* s.c.s.

A useful extension is to include tokens involving two variable names. These tokens state that the two variables involved share the property of interest: one enjoys it if and only if the other one does. More formally, we have

$$\mathcal{C}' \stackrel{\text{def}}{=} \mathcal{C} \cup \{\, X \leftrightharpoons Y \mid X, Y \in \mathit{Vars} \,\},$$

and the entailment relation is suitably extended to $\mathcal{C}'$ requiring, for each $X, Y, Z \in \mathit{Vars}$:

$$\begin{aligned} \{X \leftrightharpoons Y\} &\vdash Y \leftrightharpoons X; \\ \{X \leftrightharpoons Y, Y \leftrightharpoons Z\} &\vdash X \leftrightharpoons Z; \\ \{X, X \leftrightharpoons Y\} &\vdash Y. \end{aligned}$$

3.3.2 A Simple Constraint System for Simple Types

A slightly more interesting example is when we have more than one monadic (constraint) predicate. This is the case of domains of *simple types* where tokens like $number(X)$ or $atom(X)$ indicate that the variable X is bound to take numerical values, or Herbrand constants, respectively. The reader can easily figure out how to define such a simple constraint system. Figure 3.1

[5] Beware not to confuse CLP($\mathbb{R}$), the *idealized* language over the reals [JM94], with CLP($\mathcal{R}$), the (far from ideal) implemented language and system [JMSY92b].

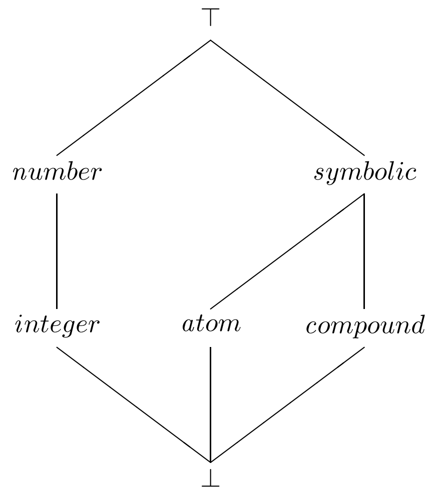


Figure 3.1: A lattice of simple types.

suggests, by means of a Hasse diagram, a possible set of tokens together with its entailment relation.

3.3.3 The Herbrand Simple Constraint System

Since almost any constraint logic-based language is an extension of Prolog, the Herbrand (or finite trees) constraint system plays a fundamental role in the field. If taken seriously, this is a quite complicated constraint system. This is due to the fact that Herbrand constraints in themselves are quantified formulas, which is not the case for most other constraint languages. For our purposes we need a clear separation between "real" program variables (in unification theory called *eliminable variables*) from quantified variables appearing only in the constraints (called *parameters*). We also wish to avoid having to talk *modulo renaming*. For these reasons[6] here we give a quite detailed account of a simple constraint system capturing Herbrand constraints.

Basically, the tokens are systems of equations in solved form (see [LMM88]). Besides that, they are also *minimal* (i.e. they do not contain useless equations, and the scope of parameters is as tight as possible), and *canonical* (i.e. the parameters' names are selected in a unique way). Formally, the set of tokens, $\mathcal{E}$, contains the symbol $\bot$ and all the syntactic objects of the form

$$\lambda\langle Y_1, \dots, Y_p\rangle \,.\, \langle X_1 = t_1, \dots, X_n = t_n\rangle$$

such that

[6]And also because we were not able to find in the literature a precise description (in terms of simple constraint systems) to refer the reader to.

1. $\{X_1 = t_1, \dots, X_n = t_n\}$ is in solved form (i.e. $X_1, \dots, X_n$ are all distinct and do not occur on right-hand sides) with parameters $\{Y_1, \dots, Y_p\} = \bigcup_{i=1}^{n} \mathit{vars}(t_i)$.

2. Equations of the form $X_i = Y_j$ where Y_j does not occur elsewhere are useless and thus forbidden: for each $i \in \{1, \dots, n\}$, if $t_i \equiv Y_j$, then there exists $k \in \{1, \dots, n\}$ with $k \neq i$ such that $Y_j \in \mathit{vars}(t_k)$.

3. The scope of each parameter cannot be restricted to a proper subset of the equations at hand: the graph where $\{1, \dots, n\}$ are the nodes and the edges are

$$\bigl\{\, (i,j) \bigm| i,j = 1, \dots, n \text{ and } \mathit{vars}(t_i) \cap \mathit{vars}(t_j) \neq \varnothing \,\bigr\}$$

is strongly connected.

4. The parameters' names are completely determined by the equations. With respect to the total ordering of variables we have[7]:

   a. $X_1 < X_2 < \cdots < X_n \prec Y_1 \prec Y_2 \prec \cdots \prec Y_p$. This identifies the variable symbols to be used.

   b. They are assigned on a first-seen basis traversing the right sides of the equations in some standard order: if $\langle Y_{i_1}, \dots, Y_{i_l} \rangle$ is the sequence of variables encountered in a depth-first, left-to-right traversal of $\langle t_1, \dots, t_n \rangle$, then[8] for each $h, k \in \{1, \dots, l\}$ with $h < k$

$$\left.\begin{array}{l} \forall h' < h, h' \geq 1 : Y_{i_{h'}} \neq Y_{i_h} \\ \forall k' < k, k' \geq 1 : Y_{i_{k'}} \neq Y_{i_k} \end{array}\right\} \implies Y_{i_h} < Y_{i_k}.$$

Notice that, by selecting the parameter's names in a unique way, we will avoid all the burden of talking "modulo renaming". Tokens of the form $\lambda\varepsilon \,.\, \bar{E}$ can be simply denoted by $\bar{E}$. The token $\lambda\varepsilon \,.\, \varepsilon$ is denoted by $\top$.

The entailment relation $\vdash \subseteq \wp_{\mathrm{f}}(\mathcal{E}) \times \mathcal{E}$ is defined by suitably augmenting the conditions of Definition 23. First we must say how we combine tokens to get other tokens:

$$\{c_1, c_2\} \vdash c, \quad \text{if } c \in \sigma(c_1, c_2), \tag{3.7}$$

where $\sigma \colon \mathcal{E} \times \mathcal{E} \to \wp_{\mathrm{f}}(\mathcal{E})$ is computed as follows.

1. If $c_1 \equiv \bot$ or $c_2 \equiv \bot$ then return $\{\bot\}$. Otherwise, assume that $c_1 \equiv \lambda\bar{Y}_1 \,.\, \bar{E}_1$ and $c_2 \equiv \lambda\bar{Y}_2 \,.\, \bar{E}_2$.

2. Consistently rename the parameters $\bar{Y}_2$ of c_2 apart from $\bar{E}_1$. Let $\lambda\bar{Y}_2' \,.\, \bar{E}_2'$ be the result.

[7]In this context, $X \prec Y$ means that Y immediately follows X in the order.

[8]Notice that we treat $\langle t_1, \dots, t_n \rangle$ as a special term (or tree) with a fictitious root added.

3. Compute the solved form of $\bar{E}_1 \cup \bar{E}'_2$ with parameters $\bar{Y}_1 \cup \bar{Y}'_2$. If the system is unsolvable return $\{\bot\}$. If it is solvable, let $\bar{F} = \{X_1 = t_1, \ldots, X_m = t_m\}$ be the solved form obtained.

4. Define the graph

$$G \overset{\text{def}}{=} \bigl\langle \bar{F}, \bigl\{ (e_i, e_j) \bigm| e_i, e_j \in \bar{F} \text{ and } \mathit{vars}(e_i) \cap \mathit{vars}(e_j) \neq \varnothing \bigr\} \bigr\rangle$$

   and let $\{\bar{F}_1, \ldots, \bar{F}_s\}$ be the nodes of the condensed graph G^* (i.e., the strongly connected components of G).

5. From each $\bar{F}_i$ obtain the token $\lambda \bar{Z}_i \mathbin{.} \bar{H}_i$ (by ordering the equations, suitably renaming the parameters and so on).

6. Return $\{ \lambda \bar{Z}_i \mathbin{.} \bar{H}_i \mid i = 1, \ldots, s \}$.

Now the entailment must be closed by anti-instance, i.e.

$$\{c'\} \vdash c, \qquad \text{if } c' \vdash c, \tag{3.8}$$

where $\vdash \subseteq \mathcal{E} \times \mathcal{E}$ is such that

$$\lambda \bar{Y}_1 \mathbin{.} \langle X_1 = t_1, \ldots, X_n = t_n \rangle \vdash \lambda \bar{Y}_2 \mathbin{.} \langle Z_1 = s_1, \ldots, Z_m = s_m \rangle$$

if and only if $\{Z_1, \ldots, Z_m\} \subseteq \{X_1, \ldots, X_n\}$ and there exists a substitution θ such that[9]

$$Z_1 \equiv X_{i_1}, \ldots, Z_m \equiv X_{i_m} \quad \text{and} \quad \langle t_{i_1}, \ldots, t_{i_m} \rangle \equiv \langle s_1, \ldots, s_m \rangle \theta.$$

The effect of a renaming $[\bar{W}/\bar{V}]$ onto a token $\lambda \bar{Y} \mathbin{.} \bar{E}$ is to substitute the non-parameters in $\bar{V}$ with the correspondent ones in $\bar{W}$ (notice that, by definition, $(\bar{W} \cup \bar{Y}) \cap FV(\lambda \bar{Y} \mathbin{.} \bar{E}) = \varnothing$), and then to rename the parameters $\bar{Y}$ (there is a unique way to do that) so to ensure that the result is still a token.

The definition is completed by saying that $\vdash \subseteq \wp_{\mathrm{f}}(\mathcal{E}) \times \mathcal{E}$ is the smallest relation satisfying (3.7), (3.8), and conditions E_1–E_5 of Definition 23. All these complications have the great advantage of being confined here. We will see, in fact, that with this definition equivalence of constraints will amount to the *equality* of the correspondent set of tokens. Similarly, extracting the common information of two Herbrand constraints will be done by taking the intersection of their sets of tokens.

[9]Again, we regard $\langle \ldots \rangle$ as a special function symbol.

3.3.4 Bounds and Relations Analysis

The analysis described in Chapter 5 is based on constraint inference, a variant of constraint propagation [Dav87]. This technique, developed in the field of Artificial Intelligence, has been applied to temporal and spatial reasoning [All83, Sim83, Sim86].

Let us focus our attention on arithmetic domains, where usually the constraints are binary relations over expressions. Let E be the set of arithmetic expressions of interest. Consider also a family I of subsets of $\mathbb{R}$ closed under intersection, that is, for each $I_1, I_2 \in \mathsf{I}$ we also have $I_1 \cap I_2 \in \mathsf{I}$. The set of arithmetic relationships is

$$\mathsf{R} \stackrel{\mathrm{def}}{=} \{=, \neq, \leq <, \geq, >\}$$

and our constraints are given by

$$\begin{aligned} \mathcal{C} \stackrel{\mathrm{def}}{=} & \bigl\{\, e_1 \bowtie e_2 \bigm| e_1, e_2 \in \mathsf{E}, \bowtie \in \mathsf{R} \,\bigr\} \\ & \cup \{\, e \lhd I \mid e \in \mathsf{E}, I \in \mathsf{I} \,\} \cup \{\bot, \top\}. \end{aligned}$$

The meaning of the constraint $e \lhd I$ is the obvious one: any real value the expression e can take is contained in I. Thus $\mathcal{C}$ provides a mixture of qualitative (relationships between expressions) and quantitative knowledge (bounds on the values of the expressions).

The approximate inference techniques we are interested in can be encoded into an entailment relation '$\vdash$' over $\mathcal{C}$. First we need to specify how we deal with intervals: we can intersect them, weaken them, and we detect failure by recognizing the empty ones:

$$\begin{array}{rcll} \{e \lhd I_1, e \lhd I_2\} & \vdash & e \lhd I_1 \cap I_2, & \\ \{e \lhd I\} & \vdash & e \lhd I', & \text{if } I \subseteq I', \\ \{e \lhd I\} & \vdash & \bot, & \text{if } I = \varnothing. \end{array}$$

Two techniques for exploiting pure qualitative information are *symmetric* and *transitive closure*:

$$\begin{array}{rcll} \{e_1 \bowtie e_2\} & \vdash & e_2 \bowtie^{-1} e_1, & \\ \{e_1 \bowtie e_2, e_2 \bowtie' e_3\} & \vdash & e_1 \bowtie'' e_3, & \text{if } \bowtie'' = tc(\bowtie, \bowtie'), \end{array}$$

where $\bowtie^{-1}$ is the inverse of $\bowtie$ (e.g., $<$ is the inverse of $>$, $\geq$ of $\leq$ and so on), and $tc \colon \mathsf{R} \times \mathsf{R} \rightarrowtail \mathsf{R}$ is the partial function individuated by the following table:

tc	$<$	$\leq$	$>$	$\geq$	$=$	$\neq$
$<$	$<$	$<$			$<$	
$\leq$	$<$	$\leq$			$\leq$	
$>$			$>$	$>$	$>$	
$\geq$			$>$	$\geq$	$\geq$	
$=$	$<$	$\leq$	$>$	$\geq$	$=$	$\neq$
$\neq$					$\neq$	

This technique allows the inference of $A < C$ from $A \leq B$ and $B < C$. Of course, qualitative information can be combined and can lead to the detection of inconsistencies:

$$\{e_1 \bowtie e_2, e_1 \bowtie' e_2\} \vdash e_1 \bowtie'' e_2,$$

if $\forall x, y \in \mathbb{R} : (x \bowtie y \wedge x \bowtie' y) \Rightarrow x \bowtie'' y$, and

$$\{e_1 \bowtie e_2, e_1 \bowtie' e_2\} \vdash \bot,$$

whenever $\forall x, y \in \mathbb{R} : \neg(x \bowtie y \wedge x \bowtie' y)$.

A classical quantitative technique is *interval arithmetic*: it allows to infer the variation interval of an expression from the intervals of its subexpressions. Let $f(e_1, \ldots, e_k)$ be any arithmetic expression having $e_1, \ldots, e_k$ as subexpressions. Then

$$\bigl\{f(e_1, \ldots, e_k) \lhd I, e_1 \lhd I_1, \ldots, e_k \lhd I_k\bigr\} \\ \vdash f(e_1, \ldots, e_k) \lhd \ddot{f}(I_1, \ldots, I_k),$$

where $\ddot{f} \colon \mathsf{I}^k \to \mathsf{I}$ is such that for each $x_1 \in I_1, \ldots, x_k \in I_k$, it happens that $f(x_1, \ldots, x_k) \in \ddot{f}(I_1, \ldots, I_k)$. For example,

$$A \lhd [3, 6) \wedge B \lhd [-1, 5] \vdash A + B \lhd [2, 11).$$

Another technique is *numeric constraint propagation*, which consists in determining the relationship between two expressions when their associated intervals do not overlap, except possibly at their endpoints. The associated family of axioms is

$$\{e_1 \lhd I_1, e_2 \lhd I_2\} \vdash e_1 \bowtie e_2, \quad \text{if } \forall x_1 \in I_1, x_2 \in I_2 : x_1 \bowtie x_2.$$

For example, if $A \in (-\infty, 2]$, $B \in [2, +\infty)$, and $C \in [5, 10]$, we can infer that $A \leq B$ and $A < C$. It is also possible to go the other way around, i.e., knowing that $U < V$ may allow to refine the intervals associated to U and V so that they do not overlap. We call this *weak interval refinement*:

$$\{e_1 \bowtie e_2, e_1 \lhd I_1, e_2 \lhd I_2\} \vdash e_1 \lhd I_1',$$

where $I_1' \stackrel{\text{def}}{=} \{\, x_1 \in I_1 \mid \exists x_2 \in I_2 \,.\, x_1 \bowtie x_2 \,\}$. This is an example of local-consistency technique [Mon74, Mac77, Fre78]. In summary, by considering the transitive closure of $\vdash$ and with some minor technical additions we end up with a simple constraint system that characterizes precisely the combination of the above techniques. Other techniques can be easily incorporated (see Chapter 5).

What we have just presented is a watered-down version of the numerical component (presented as a simple constraint system) employed in the China analyzer (see Chapter 5 for the details). Now we are in position to introduce an important class of members of the hierarchy.

3.4 Determinate Constraint Systems

Determinate constraint systems are at the bottom of the hierarchy. Such a construction is uniquely determined by a simple constraint system together with appropriate merge operator and diagonal elements. Notice that, for simplicity, we present only the *finite fragment* of the constraint system, that is, the sub-structure consisting of the finite elements only.

Definition 26 (Determinate constraint system.) *Consider a simple constraint system,* $\mathcal{S} \stackrel{\mathrm{def}}{=} \langle \mathcal{C}, \vdash, \bot, \top \rangle$*. Let* $\mathbf{0}, \mathbf{1} \in |\mathcal{C}|_0$ *and* $\otimes \colon |\mathcal{C}|_0 \times |\mathcal{C}|_0 \to |\mathcal{C}|_0$ *be given, for each* $C_1, C_2 \in |\mathcal{C}|_0$*, by*

$$\begin{aligned} \mathbf{0} &\stackrel{\mathrm{def}}{=} \mathcal{C}, \\ \mathbf{1} &\stackrel{\mathrm{def}}{=} \rho(\varnothing), \\ C_1 \otimes C_2 &\stackrel{\mathrm{def}}{=} \rho(C_1 \cup C_2). \end{aligned}$$

Let $\oplus \colon |\mathcal{C}|_0 \times |\mathcal{C}|_0 \to |\mathcal{C}|_0$ *be an operator satisfying conditions* G_2 *and* G_4 *of* Definition 7*. The projection operators* $\bar{\exists}_{\bar{X}} \colon |\mathcal{C}|_0 \to |\mathcal{C}|_0$ *are given, for each* $\bar{X} \in \mathit{Vars}^\star$ *and each* $C \in |\mathcal{C}|_0$*, by*

$$\bar{\exists}_{\bar{X}}\, C \stackrel{\mathrm{def}}{=} \rho\bigl(\{\, c \in C \bigm| FV(c) \subseteq \bar{X} \,\}\bigr).$$

Finally, let $\{\mathrm{d}_{\bar{X}\bar{Y}}\}_{\bar{X},\bar{Y} \in \mathit{Vars}^\star}$ *be a family of elements of* $|\mathcal{C}|_0$*. We will call the structure* $\langle |\mathcal{C}|_0, \otimes, \oplus, \mathbf{0}, \mathbf{1}, \{\bar{\exists}_{\bar{X}}\}, \{\mathrm{d}_{\bar{X}\bar{Y}}\} \rangle$ *a* determinate constraint system *over* $\mathcal{S}$ *and '*$\oplus$*'.*

The following is an adaptation to our more relaxed hypotheses of a result in [Sar92, Proposition 2.3].

Lemma 27 *Let* $\langle \mathcal{C}, \vdash, \bot, \top \rangle$ *be a simple constraint system. For each* $C \subseteq \mathcal{C}$ *we have* $FV(\rho(C)) \subseteq FV(C)$*.*

Proof Suppose $X \in FV(\rho(C))$. By Equation (3.2) of Definition 6 there exist $c \in \rho(C)$ and $Z \in \mathit{Vars}$ such that $c[Z/X] \notin \rho(C)$. This implies that there exists $C' \subseteq_{\mathrm{f}} C$ such that $C' \vdash c$ (otherwise $c \notin \rho(C)$). Now suppose, towards a contradiction, that $X \notin FV(C)$. This means that for each $c' \in C'$ and each $Y \in \mathit{Vars}$ we have $c'[Y/X] \in C$, and thus $C'[Y/X] \subseteq_{\mathrm{f}} C$. By the genericity axiom E_5 of Definition 23, this implies that for each $Y \in \mathit{Vars}$ we have $C'[Y/X] \vdash c[Y/X]$, and thus that $c[Y/X] \in \rho(C)$, contradicting the hypothesis $X \in FV(\rho(C))$. □

Theorem 28 *Each determinate constraint system is indeed a constraint system. Moreover, for each* $C_1, C_2 \in |\mathcal{C}|_0$*, we have* $C_1 \vdash C_2$ *if and only if* $C_1 \supseteq C_2$*.*

Proof With reference to Definition 7, G_0 is true because of point (2) in Definition 5. G_1 clearly holds by definition of $\otimes$ and ρ. G_2 and G_4 are enforced by the choice of the merge operator. For G_3, if $C \in |\mathcal{C}|_0$ then $C \otimes \mathbf{0} = \rho(C \cup \mathcal{C}) = \rho(\mathcal{C}) = \mathbf{0}$. Then, by definition of $\vdash$, $C_1 \vdash C_2$ if and only if $\rho(C_1 \cup C_2) = C_1$. Thus, if $C_1 \vdash C_2$ we have, by ρ extensivity, $C_1 \cup C_2 \subseteq \rho(C_1 \cup C_2) = C_1$, which implies $C_1 \supseteq C_2$. Conversely, assuming $C_1 \supseteq C_2$, ρ idempotence gives $\rho(C_1 \cup C_2) = \rho(C_1) = C_1$. Finally, G_5 is an immediate consequence of the definition of $\overline{\exists}_{\bar{X}}$ and of Lemma 27. $\square$

The choice of a suitable merge operator, required in addition to an s.c.s. to obtain a determinate constraint system, can be done with relative freedom. This freedom can often be conveniently exploited in order to get a reasonable complexity/precision tradeoff. The same will apply to the *ask-and-tell constraint systems* of Section 3.6.

Observe that, given $C_1, C_2 \in |\mathcal{C}|_0$, there is no *a priori* guarantee that $C = C_1 \cap C_2 \in |\mathcal{C}|_0$. In fact, there are simple constraint systems where this is false[10]. However, since ρ is an algebraic closure operator by its very definition, if it turns out that $C \in |\mathcal{C}|_0$, then there exists $C_0 \subseteq_{\mathrm{f}} C$ such that $\rho(C_0) = C$ [BS81]. That said, defining the merge operator as set intersection works in many cases.

A trivial example of merge operator is the following, whose definition is independent from the simple constraint system at hand:

$$C_1 \oplus C_2 \stackrel{\mathrm{def}}{=} \begin{cases} C_1, & \text{if } C_1 = C_2 \text{ or } C_2 = \mathbf{0}; \\ C_2, & \text{if } C_1 = \mathbf{0}; \\ \mathbf{1}, & \text{otherwise.} \end{cases} \tag{3.9}$$

For a less trivial example, suppose we are approximating subsets of $\mathbb{R}^n$ by means of (closed) convex polyhedra. Of course they will be represented by sets of linear disequations over $x_1, \ldots, x_n$, but, for the purpose of the present example, we will consider the polyhedra themselves. For any convex polyhedra $X, Y \subseteq \mathbb{R}^n$, define $X \vdash Y$ if and only if $X \subseteq Y$ and

$$X \oplus Y \stackrel{\mathrm{def}}{=} \begin{cases} X, & \text{if } X = Y \text{ or } Y = \varnothing; \\ Y, & \text{if } X = \varnothing; \\ \mathrm{bb}(X \cup Y), & \text{otherwise;} \end{cases} \tag{3.10}$$

where $\mathrm{bb}(Z)$ is the smallest "bounding box" containing $Z \subseteq \mathbb{R}^n$, that is,

$$\mathrm{bb}(Z) \stackrel{\mathrm{def}}{=} \left\{ (x_1, \ldots, x_n) \;\middle|\; \begin{array}{l} \forall i = 1, \ldots, n : \\ \qquad \inf \pi_i(Z) \leq x_i \leq \sup \pi_i(Z) \end{array} \right\}.$$

[10]For the interested reader: consider the set of tokens $\{\perp, a, b, \top\} \cup \{t_i\}_{i \in \mathbb{N}}$. Choose the least entailment relation such that: $\{t_i\} \vdash t_j$ if and only if $i \geq j$, $\{a\} \vdash \{t_i\}_{i \in \mathbb{N}}$, $\{b\} \vdash \{t_i\}_{i \in \mathbb{N}}$, and $\{a, b\} \vdash \perp$. It is easy to see that the intersection of the finite elements generated by $\{a\}$ and $\{b\}$, respectively, is $\{t_i\}_{i \in \mathbb{N}}$, which is not finite.

The most precise merge operator is, of course, given by the convex hull, namely,

$$X \oplus Y \overset{\text{def}}{=} \min\bigl\{\, W \subseteq \mathbb{R}^n \bigm| W \supseteq X \cup Y \text{ and } W \text{ is a c.p.} \,\bigr\}. \tag{3.11}$$

Notice that (3.11) satisfies both the absorption laws (thus giving rise to a lattice), (3.9) and (3.10) do not. None of them results in a distributive constraint system. Furthermore, (3.9) and (3.10) are closed, while (3.11) is not.

3.4.1 Definiteness Analysis: *Con*

Consider the extension of the *atomic* simple constraint system, $\mathcal{C}'$, introduced in Section 3.3.1, and apply to it the determinate constraint system construction with

$$C_1 \oplus C_2 \overset{\text{def}}{=} C_1 \cap C_2, \quad \text{for each } C_1, C_2 \in |\mathcal{C}'|.$$

Let also the diagonal elements be given, for each $\bar{X}, \bar{Y} \in \mathit{Vars}^\star$ of the same cardinality, by

$$\mathrm{d}_{\bar{X}\bar{Y}} \overset{\text{def}}{=} \rho\bigl(\{\, \pi_i(\bar{X}) \leftrightarrows \pi_i(\bar{Y}) \mid 1 \leq i \leq \#\bar{X} \,\}\bigr).$$

The resulting domain (a closed and Noetherian d.c.s.) is the simplest one for definiteness analysis, and it was used in early groundness analyzers [Mel85, JS87]. The name *Con* comes from the fact that elements of the form $\{X_1, \ldots, X_n\}$ are usually regarded as the conjunction $X_1 \wedge \cdots \wedge X_n$, meaning that X_1, ... , X_n are definitely bound to a unique value. In this view, $\otimes$ clearly amounts to logical conjunction.

Con is a very weak domain for definiteness analysis. It cannot capture either "aliasing" (apart from the special kind of aliasing arising from parameter passing) or more complex dependencies between variables such as those implied by "concrete" constraints like $A = f(B, C)$ and $A + B + C = 0$. Moreover it cannot represent or exploit disjunctive information.

3.4.2 The *Pattern* Domain

Endowing the Herbrand simple constraint system of Section 3.3.3 with the merge operator

$$C_1 \oplus C_2 \overset{\text{def}}{=} C_1 \cap C_2, \qquad \text{for each } C_1, C_2 \in |\mathcal{E}|_0.$$

and the diagonals (we assume for simplicity that $\pi_i(\bar{X}) \leq \pi_i(\bar{Y})$, for $i = 1$, ... , n)

$$\mathrm{d}_{\bar{X}\bar{Y}} \overset{\text{def}}{=} \rho\left(\left\{\, \lambda\langle Z_i\rangle \,.\, \langle \pi_i(\bar{X}) = Z_i, \pi_i(\bar{Y}) = Z_i\rangle \;\middle|\; \begin{array}{l} 1 \leq i \leq \#\bar{X} \\ \pi_i(\bar{Y}) \prec Z_i \end{array} \right\}\right)$$

yields a closed and Noetherian d.c.s. suitable for structural analysis. Observe that all the finitely generable elements in $|\mathcal{E}|_0 \setminus \mathbf{0}$ are finite, since there is at most one normalized greatest common instance for each finite set of terms (axiom (3.7) of the entailment relation) and only a finite number of (normalized) anti-instances for any given set of terms (axiom (3.8)) [LMM88]. As the ordering is reverse set inclusion, Noetherianity clearly follows.

Since the tokens are normalized and elements are closed by anti-instance, merging two elements of $|\mathcal{E}|_0$ makes implicit the fact that to each eliminable variable in the result will be assigned the least common anti-instance (*lca*) of the terms assigned in the elements to be merged (and all its further anti-instances). In a real implementation, where closure by anti-instance is obviously not performed, this will correspond to the use of some anti-unification algorithm to compute the *lca* of terms assigned to the same eliminable variable.

The *Pattern* domain presented here, which is a reformulation of the ones defined in [Mus90, LCVH92], deals only with pure structural information. In [CLV94] a much more powerful domain is presented, `Pat(`$\Re$`)`, parametric with respect to any abstract domain $\Re$ in their framework (which is not restricted to monotonic properties). It is possible to modify the *Pattern* domain presented in this section so to make it parametric with respect to any constraint system $\bar{\mathcal{D}}$, still obtaining a constraint system. In this way one can achieve the same results of the parametric domain of [CLV94], though in the restricted context of monotonic properties. Of course, the back-side of this restriction is a greater simplicity in the definition of the parametric domain *Pattern*$(\bar{\mathcal{D}})$.

Now that we have seen how many constraint systems can be built, we will show what the induced members of the hierarchy look like.

3.5 Powerset Constraint Systems

For the purpose of program analysis of monotonic properties it is not necessary to represent the "real disjunction" of constraints collected through different computation paths, since we are interested in the common information only. To this end, a weaker notion of disjunction suffices.

We define *powerset constraint systems*, which are instances of a well-known construction: disjunctive completion [CC92b]. For doing that we need some notions from the theory of posets.

Given a poset $\langle L, \bot, \leq \rangle$, the relation $\preceq \subseteq \wp(L) \times \wp(L)$ induced by $\leq$ is given, for each $S_1, S_2 \in \wp(L)$ by

$$(S_1 \preceq S_2) \overset{\text{def}}{\iff} (\forall x_1 \in S_1 : \exists x_2 \in S_2 \,.\, x_1 \leq x_2). \tag{3.12}$$

A subset $S \in \wp(L)$ is called *non-redundant* if and only if $\bot \notin S$ and

$$\forall x_1, x_2 \in S : x_1 \leq x_2 \implies x_1 = x_2. \tag{3.13}$$

The set of non-redundant subsets of L with respect to $\leq$ is denoted by $\wp_{\mathrm{n}}(L, \leq)$. The function $\Omega_L^{\leq} \colon \wp(L) \to \wp_{\mathrm{n}}(L, \leq)$, mapping each set into its non-redundant counterpart is given, for each $S \in \wp(L)$, by

$$\Omega_L^{\leq}(S) \stackrel{\mathrm{def}}{=} S \setminus \{\, x \in S \mid x = \bot \vee \exists x' \in S \,.\, x < x' \,\}. \tag{3.14}$$

Thus, for $S \in \wp(L)$, $\Omega_L^{\leq}(S)$ is the set of maximal elements of S. However, there is no guarantee, in general, that such maximal elements exist: L could be an infinite chain without an upper bound in L, and thus would be mapped to $\varnothing$ by $\Omega_L^{\leq}$. We will denote by $\wp_{\mathrm{c}}(L)$ the set of all those $S \in \wp(L)$ such that, if S contains an infinite chain C, then it also contains an upper bound for C. Observe that $\wp_{\mathrm{f}}(L) \subseteq \wp_{\mathrm{c}}(L)$ and that, if L satisfies the ascending chain condition, then $\wp(L) = \wp_{\mathrm{c}}(L)$.

Proposition 29 *If $\langle L, \bot, \leq \rangle$ is a poset the following hold:*

1. *$\langle \wp_{\mathrm{n}}(L, \leq), \preceq \rangle$ is a poset;*
2. *for each $S \in \wp_{\mathrm{c}}(L)$, we have both $\Omega_L^{\leq}(S) \preceq S$ and $S \preceq \Omega_L^{\leq}(S)$;*
3. *for each family $\{S_i \in \wp(L)\}_{i \in I}$ we have*

$$\Omega_L^{\leq}\Bigl(\bigcup_{i \in I} \Omega_L^{\leq}(S_i)\Bigr) = \Omega_L^{\leq}\Bigl(\bigcup_{i \in I} S_i\Bigr);$$

4. *if $\langle L, \bot, \leq, \wedge \rangle$ is a meet-semilattice then for each $x \in L$ and each $S \in \wp_{\mathrm{c}}(L)$*

$$\Omega_L^{\leq}\bigl(\{\, x \wedge y \mid y \in \Omega_L^{\leq}(S) \,\}\bigr) = \Omega_L^{\leq}\bigl(\{\, x \wedge y \mid y \in S \,\}\bigr).$$

Proof

1. $\preceq$ is easily seen to be preorder relation. For antisymmetry, consider a pair $S_1, S_2 \in \wp_{\mathrm{n}}(L, \leq)$ such that $S_1 \preceq S_2$ and $S_2 \preceq S_1$. If $x_1 \in S_1$ then there exists $x_2 \in S_2$ with $x_1 \leq x_2$. But then again there exists $x_1' \in S_1$ such that $x_1 \leq x_2 \leq x_1'$. By non-redundancy we must have $x_1 = x_2 = x_1'$, so that $x_1 \in S_2$ and $S_1 \subseteq S_2$. A symmetric argument proves that, indeed, $S_1 = S_2$.
2. Observe that $\Omega_L^{\leq}(S) \subseteq S$ and that, if $S \in \wp_{\mathrm{c}}(L)$ and $x \in S$, then either $x \in \Omega_L^{\leq}(S)$ or there exists $y \in \Omega_L^{\leq}(S)$ such that $x < y$.
3. Straightforward.

4. We have that

$$\begin{aligned}
&\Omega_L^{\lessgtr}\bigl(\{\, x \wedge y \mid y \in \Omega_L^{\lessgtr}(S) \,\}\bigr)\\
&\quad = \Omega_L^{\lessgtr}\bigl(\{\, x \wedge y \mid y \in \Omega_L^{\lessgtr}(S) \,\}\\
&\qquad \cup \{\, x \wedge y \bigm| y \in S \setminus \Omega_L^{\lessgtr}(S), \exists y' \in \Omega_L^{\lessgtr}(S) \mathrel{.} x \wedge y \leq x \wedge y' \,\}\bigr)\\
&\quad = \Omega_L^{\lessgtr}\bigl(\{\, x \wedge y \mid y \in \Omega_L^{\lessgtr}(S) \,\} \cup \{\, x \wedge y \bigm| y \in S \setminus \Omega_L^{\lessgtr}(S) \,\}\bigr)\\
&\quad = \Omega_L^{\lessgtr}\bigl(\{\, x \wedge y \mid y \in S \,\}\bigr),
\end{aligned}$$

since, for each $x \in L$, $S \in \wp_{\mathrm{c}}(L)$ and $y \in S \setminus \Omega_L^{\lessgtr}(S)$, there is always a $y' \in \Omega_L^{\lessgtr}(S)$ such that $y < y'$. The condition $\exists y' \in \Omega_L^{\lessgtr}(S) \mathrel{.} x \wedge y \leq x \wedge y'$ above is thus always satisfied by $\wedge$ monotonicity. □

The powerset construction upgrades a domain by considering sets of elements of the base-level domain that are non-redundant with respect to the approximation ordering.

Definition 30 (Powerset constraint systems.) *The* powerset constraint system *over a Noetherian constraint system*

$$\bar{\mathcal{D}} = \langle \mathcal{D}, \otimes, \oplus, \mathbf{0}, \mathbf{1}, \{\bar{\exists}_{\bar{X}}\}, \{\mathrm{d}_{\bar{X}\bar{Y}}\} \rangle$$

is given by

$$\bigl\langle \wp_{\mathrm{n}}(\mathcal{D}, \vdash), \otimes_{\mathrm{P}}, \oplus_{\mathrm{P}}, \mathbf{0}_{\mathrm{P}}, \mathbf{1}_{\mathrm{P}}, \{\bar{\exists}^{\mathrm{P}}_{\bar{X}}\}, \{\mathrm{d}^{\mathrm{P}}_{\bar{X}\bar{Y}}\} \bigr\rangle,$$

where

$$\begin{aligned}
S_1 \otimes_{\mathrm{P}} S_2 &\stackrel{\mathrm{def}}{=} \Omega_{\mathcal{D}}^{\vdash}\bigl(\{\, C_1 \otimes C_2 \mid C_1 \in S_1, C_2 \in S_2 \,\}\bigr),\\
S_1 \oplus_{\mathrm{P}} S_2 &\stackrel{\mathrm{def}}{=} \Omega_{\mathcal{D}}^{\vdash}(S_1 \cup S_2),\\
\mathbf{0}_{\mathrm{P}} &\stackrel{\mathrm{def}}{=} \varnothing,\\
\mathbf{1}_{\mathrm{P}} &\stackrel{\mathrm{def}}{=} \{\mathbf{1}\},\\
\bar{\exists}^{\mathrm{P}}_{\bar{X}}\, S &\stackrel{\mathrm{def}}{=} \Omega_{\mathcal{D}}^{\vdash}\bigl(\{\, \bar{\exists}_{\bar{X}}\, C \mid C \in S \,\}\bigr),\\
\mathrm{d}^{\mathrm{P}}_{\bar{X}\bar{Y}} &\stackrel{\mathrm{def}}{=} \{\mathrm{d}_{\bar{X}\bar{Y}}\}.
\end{aligned}$$

If $\bar{\mathcal{D}}$ *is* any *constraint system, the* finite powerset constraint system *over* $\bar{\mathcal{D}}$ *is*

$$\bigl\langle \wp_{\mathrm{n}}(\mathcal{D}, \vdash) \cap \wp_{\mathrm{f}}(\mathcal{D}), \otimes_{\mathrm{P}}, \oplus_{\mathrm{P}}, \mathbf{0}_{\mathrm{P}}, \mathbf{1}_{\mathrm{P}}, \{\bar{\exists}^{\mathrm{P}}_{\bar{X}}\}, \{\mathrm{d}^{\mathrm{P}}_{\bar{X}\bar{Y}}\} \bigr\rangle,$$

where all the operators are as above.

This double definition reflects the two possible uses of powerset constraint systems. One is to define concrete domains in those cases where the base-level constraint system is Noetherian. The other is when designing abstract domains, where clearly only the finite elements are of interest. In both cases, point (2) of Proposition 29 tells us that, when we deal with monotonic properties, we lose nothing if we restrict ourselves to non-redundant sets in order to capture the non-determinism of CLP languages. Of course, when the base-level c.s. is not Noetherian, one has to consider all the subsets in the design of a concrete domain.

Observe that the powerset construction completely disregards the merge operator of the base-level constraint system. Thus it can be applied to structures weaker than constraint systems, where the merge operator is missing. Instead of devoting a definition to them we rely on the fact that it is always possible to augment such structures with the trivial merge operator defined in (3.9) so to obtain a constraint system in the sense of Definition 7. This way, for example, any simple constraint system can indirectly constitute the basis for a powerset construction.

Theorem 31 *Any powerset constraint system built over a Noetherian c.s. $\bar{\mathcal{D}}$,*

$$\bigl\langle \wp_{\mathrm{n}}(\mathcal{D}, \vdash), \otimes_{\mathrm{P}}, \oplus_{\mathrm{P}}, \mathbf{0}_{\mathrm{P}}, \mathbf{1}_{\mathrm{P}}, \{\exists^{\mathrm{P}}_{\bar{X}}\}, \{\mathrm{d}^{\mathrm{P}}_{\bar{X}\bar{Y}}\}\bigr\rangle,$$

is a closed and completely distributive constraint system, where the ordering is given, for each $S_1, S_2 \in \wp_{\mathrm{n}}(\mathcal{D}, \vdash)$, by

$$S_1 \vdash_{\mathrm{P}} S_2 \iff \forall C_1 \in S_1 : \exists C_2 \in S_2 \,.\, C_1 \vdash C_2. \tag{3.15}$$

For any c.s. $\bar{\mathcal{D}}$, the finite powerset c.s. built over $\bar{\mathcal{D}}$ is a distributive constraint system, where the ordering is given by (3.15)*, for $S_1, S_2 \in \wp_{\mathrm{n}}(\mathcal{D}, \vdash) \cap \wp_{\mathrm{f}}(\mathcal{D})$.*

Proof Let $\bar{\mathcal{D}}$ be a Noetherian c.s. We start by showing that $S_1 \otimes_{\mathrm{P}} S_2$ is the glb of S_1 and S_2 with respect to $\vdash_{\mathrm{P}}$. Clearly $S_1 \otimes_{\mathrm{P}} S_2 \vdash_{\mathrm{P}} S_1$ and $S_1 \otimes_{\mathrm{P}} S_2 \vdash_{\mathrm{P}} S_2$ since, for each $C \in S_1 \otimes_{\mathrm{P}} S_2$ there are some $C_1 \in S_1$ and $C_2 \in S_2$ such that $C_1 \otimes C_2 = C$. This implies $C \vdash C_1$ and $C \vdash C_2$. Suppose now to have $S \in \wp_{\mathrm{n}}(\mathcal{D}, \vdash)$ such that $S \vdash_{\mathrm{P}} S_1$ and $S \vdash_{\mathrm{P}} S_2$. Then $S \vdash_{\mathrm{P}} S_1 \otimes_{\mathrm{P}} S_2$, since

$$\frac{\forall C \in S : \exists C_1 \in S_1 \,.\, C \vdash C_1 \qquad \forall C \in S : \exists C_2 \in S_2 \,.\, C \vdash C_2}{\forall C \in S : \exists C_1 \in S_1, C_2 \in S_2 \,.\, C \vdash C_1 \otimes C_2}$$

and either $C_1 \otimes C_2 \in S_1 \otimes_{\mathrm{P}} S_2$ or there exists $C' \in S_1 \otimes_{\mathrm{P}} S_2$ such that $C_1 \otimes C_2 \vdash C'$. In a similar way $S_1 \oplus_{\mathrm{P}} S_2$ is shown to be the lub of S_1 and S_2 with respect to $\vdash_{\mathrm{P}}$. In fact it is clear that $S_1 \vdash_{\mathrm{P}} S_1 \oplus_{\mathrm{P}} S_2$ and $S_2 \vdash_{\mathrm{P}} S_1 \oplus_{\mathrm{P}} S_2$. Furthermore, if $S \in \wp_{\mathrm{n}}(\mathcal{D}, \vdash)$ is such that $S_1 \vdash_{\mathrm{P}} S$ and $S_2 \vdash_{\mathrm{P}} S$, then, since

$$\frac{\forall C_1 \in S_1 : \exists C_1' \in S \,.\, C_1 \vdash C_1' \qquad \forall C_2 \in S_2 : \exists C_2' \in S \,.\, C_2 \vdash C_2'}{\forall C \in \Omega^{\vdash}_{\mathcal{D}}(S_1 \cup S_2) : \exists C' \in S \,.\, C \vdash C'}$$

we have $S_1 \oplus_{\mathrm{P}} S_2 \vdash_{\mathrm{P}} S$. The minimum and maximum elements of the lattice are clearly $\varnothing$ and $\{\mathbf{1}\}$, respectively. We now show, using points (3) and (4) of Proposition 29, that complete meet-distributivity holds:

$$\begin{aligned}
S \otimes_{\mathrm{P}} \bigoplus_{i \in I}{}_{\mathrm{P}}\, S_i &= \Omega^{\vdash}_{\mathcal{D}}\bigl(\{\, C \otimes C' \mid C \in S, C' \in \Omega^{\vdash}_{\mathcal{D}}(\textstyle\bigcup_{i \in I} S_i) \,\}\bigr) \\
&= \Omega^{\vdash}_{\mathcal{D}}\bigl(\{\, C \otimes C' \mid C \in S, C' \in \textstyle\bigcup_{i \in I} S_i \,\}\bigr) \\
&= \Omega^{\vdash}_{\mathcal{D}}\bigl(\textstyle\bigcup_{i \in I}\{\, C \otimes C' \mid C \in S, C' \in S_i \,\}\bigr) \\
&= \Omega^{\vdash}_{\mathcal{D}}\Bigl(\textstyle\bigcup_{i \in I} \Omega^{\vdash}_{\mathcal{D}}\bigl(\{\, C \otimes C' \mid C \in S, C' \in S_i \,\}\bigr)\Bigr) \\
&= \bigoplus_{i \in I}{}_{\mathrm{P}} (S \otimes_{\mathrm{P}} S_i).
\end{aligned}$$

In particular we have shown that $\langle \wp_{\mathrm{n}}(\mathcal{D}, \vdash), \otimes_{\mathrm{P}}, \oplus_{\mathrm{P}}, \mathbf{0}_{\mathrm{P}}, \mathbf{1}_{\mathrm{P}} \rangle$ is a distributive lattice so that G_1–G_4 of Definition 7 are satisfied. G_0, G_5, and closedness are implicit in the definition. The above proof can be replayed substituting $\wp_{\mathrm{n}}(\mathcal{D})$ with $\wp_{\mathrm{n}}(\mathcal{D}, \vdash) \cap \wp_{\mathrm{f}}(\mathcal{D})$, I with $\{1, 2\}$, and, of course, omitting closedness. $\square$

3.5.1 A Collecting Semantics for Logic Programs

When interested in monotonic properties of logic programs, a suitable collecting semantics can be defined over the domain resulting from the application of the powerset construction to the *Pattern* d.c.s. of Section 3.4.2, which is Noetherian.

3.5.2 Structural Analysis: More than *Pattern*

One possibility for obtaining more precise structural information than possible with *Pattern*, is to use the domain of the previous section together with a suitable widening operator $\nabla \colon \wp_{\mathrm{n}}(|\mathcal{E}|_0)^2 \to \wp_{\mathrm{n}}(|\mathcal{E}|_0)$ to ensure termination.

A very crude widening operator for any powerset c.s. can be defined in terms of the base-level merge operator as follows:

$$S_1 \nabla S_2 \stackrel{\text{def}}{=} \Omega^{\vdash}_{\mathcal{D}}\bigl(\{\textstyle\bigoplus(S_1 \oplus_{\mathrm{P}} S_2)\}\bigr).$$

Of course this destroys, as soon as it is applied, all the extra-precision gained by passing to the powerset. Things can be improved by using derived operators like

$$S_1 \nabla' S_2 \stackrel{\text{def}}{=} \begin{cases} S_1 \nabla S_2, & \text{if } p(S_1, S_2); \\ S_2, & \text{otherwise,} \end{cases}$$

where the predicate $p(S_1, S_2)$ is true if, on passing from S_1 to S_2 (usually S_1 and S_2 are the results of two adjacent iterates computed during the analysis) “something has grown”.

A better widening for the current example of structural analysis can be defined along the following lines. Take $S_1, S_2 \in \wp_{\mathrm{n}}(|\mathcal{E}|_0)$, and let $S = S_1 \oplus_{\mathrm{P}} S_2$. From each $C \in S$ take out all the tokens involving terms of depth greater than some fixed k; call C' the result and S' the set of all the C' so obtained. Now $S_1 \nabla S_2$ is obtained from S' by removing all the redundant elements. This widening roughly corresponds to the use of depth-k approximations as in [ST84, MS88]. It is still not very accurate, as it "cuts" also the structures that have not grown. A more precise widening can be obtained on these lines by restricting the "depth-k cut" to those variables such that the maximum depth of the associated patterns in S_2 is greater than the maximum depth of the associated patterns in S_1.

3.6 Ask-and-Tell Constraint Systems

We now consider constraint systems having additional structure. This additional structure allows to express, at the constraint system level, that the imposition of certain constraints must be delayed until some other constraints are imposed. In [Sar93] similar constructions are called *ask-and-tell constraint systems*. In our construction, ask-and-tell constraint systems are built from constraint systems by regarding some kernel operators as constraints. We follow [Sar93] in considering cc as *the* language framework for expressing and computing with kernel operators.[11] For this reason we will present kernel operators as cc agents. For our current purposes we need only a very simple fragment of the determinate cc language: the one of *finite* cc *agents*. This fragment is described in [SRP91] by means of a declarative semantics. Here we give an operational characterization that is better suited to our needs.

Definition 32 (Finite cc agents: syntax.) *A* finite cc agent *over a constraint system* $\bar{\mathcal{D}} = \langle \mathcal{D}, \otimes, \oplus, \mathbf{0}, \mathbf{1}, \{\bar{\exists}_{\bar{X}}\}, \{\mathrm{d}_{\bar{X}\bar{Y}}\}\rangle$ *is any string generated by the following grammar:*

$$\mathbf{A} ::= \mathrm{tell}(C) \mid \mathrm{ask}(C) \to \mathbf{A} \mid \mathbf{A} \parallel \mathbf{A}$$

where $C \in \mathcal{D}$. *We will denote by* $\mathcal{A}(\bar{\mathcal{D}})$ *the language of such strings. The following explicit definition is also given:*

$$\mathrm{ask}(C_1; \ldots ; C_n) \to \mathbf{A} \equiv \bigl(\mathrm{ask}(C_1) \to \mathbf{A}\bigr) \parallel \cdots \parallel \bigl(\mathrm{ask}(C_n) \to \mathbf{A}\bigr). \quad (3.16)$$

When this will not cause confusion we will freely drop the syntactic sugar, writing C and $C_1 \to C_2$ where $\mathrm{tell}(C)$ and $\mathrm{ask}(C_1) \to \mathrm{tell}(C_2)$ are intended.

[11] At least for a relevant class of them. See Section 3.6.1 for more on this subject.

One of the beautiful properties of kernel operators is that they can be uniquely represented by their range, i.e., the set of their fixed points [GHK$^+$80]. The denotational semantics of finite cc agents over $\bar{\mathcal{D}}$ can thus be expressed by a function $[\![\cdot]\!]\colon \mathcal{A}(\bar{\mathcal{D}}) \to \wp(\mathcal{D})$ defined following [SRP91].

Definition 33 (Semantics of finite cc agents.) *The semantics of finite* cc *agents over a constraint system is given by the following equations:*

$$[\![C]\!] \stackrel{\text{def}}{=} {\downarrow} C, \tag{3.17}$$

$$[\![C \to A]\!] \stackrel{\text{def}}{=} \overline{{\downarrow} C} \cup [\![A]\!], \tag{3.18}$$

$$[\![A \parallel B]\!] \stackrel{\text{def}}{=} [\![A]\!] \cap [\![B]\!]. \tag{3.19}$$

Observe that the actual kernel operator A^K corresponding to a finite agent $A \in \mathcal{A}(\bar{\mathcal{D}})$ can be recovered from $[\![A]\!]$ as

$$A^K \stackrel{\text{def}}{=} \lambda C \,.\, \sup\bigl(({\downarrow} C) \cap [\![A]\!]\bigr). \tag{3.20}$$

The introduction of a syntactic normal form for finite cc agents allows to simplify the subsequent semantic treatment.

Definition 34 (Finite cc agents: syntactic normal form.) *The transformation η over $\mathcal{A}(\bar{\mathcal{D}})$ is defined, by structural induction, as follows, for each $C^a, C^a_1, C^a_2, C^t \in \mathcal{D}$ and $A, A_1, A_2 \in \mathcal{A}(\bar{\mathcal{D}})$:*

$$\begin{aligned}
\eta(C^a \to C^t) &\stackrel{\text{def}}{=} \begin{cases} \mathbf{1} \to \mathbf{1}, & \text{if } C^a \vdash C^t; \\ C^a \to (C^a \otimes C^t), & \text{otherwise;} \end{cases} \\
\eta(C^t) &\stackrel{\text{def}}{=} \mathbf{1} \to C^t; \\
\eta\bigl(C^a_1 \to (C^a_2 \to A)\bigr) &\stackrel{\text{def}}{=} \eta\bigl((C^a_1 \otimes C^a_2) \to A\bigr); \\
\eta\bigl(C^a \to (A_1 \parallel A_2)\bigr) &\stackrel{\text{def}}{=} \eta\bigl((C^a \to A_1) \parallel (C^a \to A_2)\bigr); \\
\eta(A_1 \parallel A_2) &\stackrel{\text{def}}{=} \eta(A_1) \parallel \eta(A_2).
\end{aligned}$$

The following fact is easily proved.

Proposition 35 *The transformation η of* Definition 34 *is well defined. Furthermore, if $A \in \mathcal{A}(\bar{\mathcal{D}})$ then $[\![\eta(A)]\!] = [\![A]\!]$ and $\eta(A)$ is of the form*

$$(C^a_1 \to C^t_1) \parallel \cdots \parallel (C^a_n \to C^t_n).$$

where $C^t_i \Vdash C^a_i$ for each $i = 1, \ldots, n$.

Proof Observe that each agent is in one and only one of the forms appearing in the left-hand sides of the equations. Given an agent, define its *complexity* as the number obtained by summing up the number 3 for each connective (i.e., '$\|$' and '$\rightarrow$') occurring in a guarded context (i.e., in $C \rightarrow A$ the connectives in A are worth 3), and the number 1 for all the other connectives. It is immediate to verify that in the recursive equations of Definition 34 the arguments of η in the right-hand sides have complexity strictly less than the ones in the left-hand sides, while the right-hand sides of non-recursive equation are of the form $C^a \rightarrow C^t$. Thus η is indeed a function, and its range is as stated. The fact that η is semantic-preserving can be shown by total induction on the complexity of agents. The base cases are trivial. For the induction step we have, for instance,

$$\begin{aligned}
[\![C^a \rightarrow (A_1 \parallel A_2)]\!] &= \overline{\downarrow C^a} \cup \big([\![A_1]\!] \cap [\![A_2]\!]\big) \\
&= \big(\overline{\downarrow C^a} \cup [\![A_1]\!]\big) \cap \big(\overline{\downarrow C^a} \cup [\![A_2]\!]\big) \\
&= \Big[\!\!\Big[(C^a \rightarrow A_1) \parallel (C^a \rightarrow A_2)\Big]\!\!\Big] \\
&= \Big[\!\!\Big[\eta\big((C^a \rightarrow A_1) \parallel (C^a \rightarrow A_2)\big)\Big]\!\!\Big] \\
&= \Big[\!\!\Big[\eta\big(C^a \rightarrow (A_1 \parallel A_2)\big)\Big]\!\!\Big]. \qquad \square
\end{aligned}$$

Thus, by considering only agents of the form $\|_{i=1}^{n} C_i^a \rightarrow C_i^t$ we do not lose any generality. We will call elementary agents of the kind $C^a \rightarrow C^t$ *ask-tell pairs.*

Now we express the operational semantics of finite cc agents by means of rewrite rules. An agent in syntactic normal form is rewritten by applying the logical rules of the calculus modulo a structural congruence. This congruence states, intuitively, that we can regard an agent as a set of (concurrent) ask-tell pairs. The semantics of parallel composition stated in (3.19) clearly allows that.

Definition 36 (A calculus of finite cc agents.) *Consider the agent* $\mathbf{1}_{\mathrm{A}} \stackrel{\mathrm{def}}{=} \mathbf{1} \rightarrow \mathbf{1}$. *The* structural congruence *of the calculus is the smallest congruence relation* $\equiv_s$ *such that* $\langle \mathcal{A}(\bar{\mathcal{D}}), \|, \mathbf{1}_{\mathrm{A}} \rangle_{/\equiv_s}$ *is a commutative and idempotent monoid. The reduction rules of the calculus are given in* Figure 3.2. *The relation* $\rho_{\mathrm{A}} \subseteq \mathcal{A}(\bar{\mathcal{D}}) \times \mathcal{A}(\bar{\mathcal{D}})$ *is defined, for each* $A, A' \in \mathcal{A}(\bar{\mathcal{D}})$, *as follows:* $A \ \rho_{\mathrm{A}} \ A'$ *if and only if*

$$(A = A_1) \wedge (A_n = A') \wedge A_1 \mapsto A_2 \mapsto \cdots \mapsto A_n \not\mapsto$$

In the following we will systematically abuse the notation denoting the quotient of $\mathcal{A}(\bar{\mathcal{D}})$ with respect to $\equiv_s$ simply by $\mathcal{A}(\bar{\mathcal{D}})$. Consequently, every assertion concerning $\mathcal{A}(\bar{\mathcal{D}})$ is to be understood *modulo structural congruence.* In particular, an agent will be regarded as the *set* of its ask-tell pairs.

Lemma 37 *The term-rewriting system depicted in* Figure 3.2 *is terminating. Thus, for each* $A \in \mathcal{A}(\bar{\mathcal{D}})$ *there is always an* $A' \in \mathcal{A}(\bar{\mathcal{D}})$ *such that* $A \ \rho_{\mathrm{A}} \ A'$.

Structure

$$\frac{A_1 \equiv_s A_1' \quad A_1' \longmapsto A_2' \quad A_2' \equiv_s A_2}{A_1 \longmapsto A_2} \qquad \frac{A_1 \longmapsto A_1'}{A_1 \parallel A_2 \longmapsto A_1' \parallel A_2}$$

Reduction $r(1,2)$

$$\frac{C_1^a \vdash C_2^a \qquad C_1^a \otimes C_2^t \vdash C_1^t}{(C_1^a \to C_1^t) \parallel (C_2^a \to C_2^t) \longmapsto (C_2^a \to C_2^t)}$$

Deduction $d(1,2)$

$$\frac{C_1^t \vdash C_2^a \qquad C_1^t \nvdash C_2^t}{(C_1^a \to C_1^t) \parallel (C_2^a \to C_2^t) \longmapsto \big(C_1^a \to (C_1^t \otimes C_2^t)\big) \parallel (C_2^a \to C_2^t)}$$

Absorption $a(1,2)$

$$\frac{C_1^a \Vdash C_2^a \qquad C_1^a \nvdash C_2^t \qquad C_1^t \Vdash C_1^a \otimes C_2^t}{(C_1^a \to C_1^t) \parallel (C_2^a \to C_2^t) \longmapsto \big((C_1^a \otimes C_2^t) \to C_1^t\big) \parallel (C_2^a \to C_2^t)}$$

Figure 3.2: Reduction rules for finite cc agents.

Proof Consider an agent in syntactic normal form $A = \|_{i=1}^n A_i$. Each rewriting step results either in the removal of a pair (at most $n-1$ such steps can be performed using the reduction rule), or in the strict strengthening of the ask or tell constraint of a pair. An upper bound for the number of such steps can be obtained by reasoning as follows. The tell constraint of each of the n pairs can be strengthened at most $n-1$ times. Thus at most $n(n-1)$ applications of the deduction rule are possible. Similarly, the absorption rule will be applied at most $n(n-1)$ times. Consequently, A will be maximally rewritten in $O(n^2)$ steps. □

We introduce now, following [SRP91], a normal form for finite cc agents.

Definition 38 (Semantic normal form.) [SRP91] *The agent $A \in \mathcal{A}(\bar{\mathcal{D}})$ is* in semantic normal form *if and only if $A = \mathbf{1}_{\mathrm{A}}$ or $A = \|_{i=1}^n C_i^a \to C_i^t$ and, for each $i, j \in \{1, \dots, n\}$:*

N_1. $C_i^t \Vdash C_i^a$;

N_2. $i \neq j \implies C_i^a \neq C_j^a$;

N_3. $C_i^a \Vdash C_j^a \implies C_i^a \Vdash C_j^t$;

N_4. $C_i^t \vdash C_j^a \implies C_i^t \vdash C_j^t$.

It turns out that this normal form is indeed very strong, whence its name.

Theorem 39 [SRP91] *Two agents $A_1, A_2 \in \mathcal{A}(\bar{\mathcal{D}})$ have the same semantic normal form if and only if $[\![A_1]\!] = [\![A_2]\!]$.*

The purpose of our rewriting system is to put finite cc agents into semantic normal form, preserving their original semantics.

Lemma 40 *For each agent $A \in \mathcal{A}(\bar{\mathcal{D}})$ in syntactic normal form, if $A\ \rho_{\mathrm{A}}\ A'$ then $[\![A]\!] = [\![A']\!]$ and A' is in semantic normal form.*

Proof It is immediate to verify that the rules of the calculus preserve the syntactic normal form, thus A' satisfies condition N_1.

Suppose N_2 is not satisfied, that is, $C_i^a = C_j^a$ and, consequently, $C_i^t \neq C_j^t$, $C_i^t \Vdash C_j^a$, and $C_j^t \Vdash C_i^a$. Either $C_i^t \nvdash C_j^t$ or $C_j^t \nvdash C_i^t$ must hold, so we can apply the deduction rule $d(i,j)$ or the symmetric one $d(j,i)$, respectively.

Suppose that N_3 is not satisfied, namely $C_i^a \Vdash C_j^a$ and $C_i^a \nVdash C_j^t$ (implying $C_i^t \Vdash C_j^a$). There are several cases. If $C_i^a = C_j^t$ then $C_i^t \Vdash C_i^a = C_j^t$, thus $C_j^t \vdash C_i^a$ and $C_j^t \nvdash C_i^t$, which means $d(j,i)$ is applicable. The other possibility is when $C_i^a \nvdash C_j^t$ and either $C_j^t \vdash C_i^t$ or $C_j^t \nvdash C_i^t$. In the former case we have $C_j^t \vdash C_i^t \Vdash C_i^a$, whence $C_i^a \otimes C_j^t = C_j^t \vdash C_i^t$ and $r(i,j)$ can fire. The latter case is further split as follows: if $C_i^t \nvdash C_j^t$ then rule $d(i,j)$ is applicable, otherwise we have $C_i^t \Vdash C_j^t$, which implies $C_i^t \vdash C_i^a \otimes C_j^t$. If this last entailment is strict the absorption rule $a(i,j)$ can fire, if not (namely $C_i^t = C_i^a \otimes C_j^t$), then we can apply $r(i,j)$.

Finally, negating N_4 we obtain $C_i^t \vdash C_j^a$ and $C_i^t \nvdash C_j^t$, which is exactly the applicability condition for $d(i,j)$.

Preservation of semantics is readily verified. For reduction, we show that if $C_1^a \vdash C_2^a$ and $C_1^a \otimes C_2^t \vdash C_1^t$ then

$$[\![(C_1^a \to C_1^t) \parallel (C_2^a \to C_2^t)]\!] = [\![C_2^a \to C_2^t]\!],$$

or, equivalently,

$$\forall C \in \mathcal{D} : [\![C_2^a \to C_2^t]\!]^K(C) \vdash [\![C_1^a \to C_1^t]\!]^K(C).$$

In fact, if $C \nvdash C_1^a$ then $[\![C_1^a \to C_1^t]\!]^K(C) = C$ and the thesis hold by extensivity of $[\![C_2^a \to C_2^t]\!]^K$. If $C \vdash C_1^a$ then $C \vdash C_2^a$ and

$$\begin{aligned}
[\![C_2^a \to C_2^t]\!]^K(C) &= C \otimes C_2^t \\
&= C \otimes C_1^a \otimes C_2^t \\
&\vdash C \otimes C_1^t \\
&= [\![C_1^a \to C_1^t]\!]^K(C).
\end{aligned}$$

The reader is also referred to [SRP91] where less constrained versions of deduction and absorption are reported as laws *L12* and *L11*, respectively. □

Proposition 41 *The term-rewriting system depicted in* Figure 3.2 *is strongly normalizing. Thus the relation ρ_{A} is indeed a function $\rho_{\mathrm{A}} \colon \mathcal{A}(\bar{\mathcal{D}}) \to \mathcal{A}(\bar{\mathcal{D}})$.*

Proof Let $A \in \mathcal{A}(\bar{\mathcal{D}})$ in syntactic normal form, and suppose $A \;\rho_{\mathrm{A}}\; A'$ and $A \;\rho_{\mathrm{A}}\; A''$. By Lemma 40 both A' and A'' are in semantic normal form, and $[\![A']\!] = [\![A]\!] = [\![A'']\!]$. By Theorem 39 we can conclude that $A' = A''$.

The final result is now easily obtained.

Corollary 42 *For $A_1, A_2 \in \mathcal{A}(\bar{\mathcal{D}})$ we have $\rho_{\mathrm{A}}(A_1) = \rho_{\mathrm{A}}(A_2)$ if and only if $[\![A_1]\!] = [\![A_2]\!]$.*

Proof By Proposition 41 $\rho_{\mathrm{A}}(A_1)$ and $\rho_{\mathrm{A}}(A_2)$ are well defined while, by Lemma 40, we have $[\![A_1]\!] = [\![\rho_{\mathrm{A}}(A_1)]\!]$ and $[\![A_2]\!] = [\![\rho_{\mathrm{A}}(A_2)]\!]$ and both $\rho_{\mathrm{A}}(A_1)$ and $\rho_{\mathrm{A}}(A_2)$ are in semantic normal form. Thus $\rho_{\mathrm{A}}(A_1) = \rho_{\mathrm{A}}(A_2)$ if and only if A_1 and A_2 have the same normal form. By Theorem 39 this is equivalent to $[\![A_1]\!] = [\![A_2]\!]$. □

The situation here is almost identical to the one of Definition 25, in that we have a domain-independent strong normal form also for the present class of constraints (i.e., agents) incorporating the notion of dependency.

Definition 43 (Elements.) *The* elements *of $\mathcal{A}(\bar{\mathcal{D}})$ are the fixed points of the* inference map *ρ_{A}. The set of elements of $\mathcal{A}(\bar{\mathcal{D}})$ will be denoted by $|\mathcal{A}(\bar{\mathcal{D}})|$. Thus*

$$|\mathcal{A}(\bar{\mathcal{D}})| \stackrel{\text{def}}{=} \bigl\{\, A \in \mathcal{A}(\bar{\mathcal{D}}) \bigm| \rho_{\mathrm{A}}(A) = A \,\bigr\}.$$

We are now in position to introduce a new class in our hierarchy of constraint systems. Again we present only the finite fragment of the constraint system. Here, the finite elements are precisely the finite cc agents in $|\mathcal{A}(\bar{\mathcal{D}})|$.

Definition 44 (Ask-and-tell constraint system.) *Given a constraint system $\bar{\mathcal{D}} = \langle \mathcal{D}, \otimes, \oplus, \mathbf{0}, \mathbf{1}, \{\exists_{\bar{X}}\}, \{\mathrm{d}_{\bar{X}\bar{Y}}\}\rangle$, let $\mathcal{A} = |\mathcal{A}(\bar{\mathcal{D}})|$. Then let $\mathbf{0}_{\mathrm{A}}, \mathbf{1}_{\mathrm{A}} \in \mathcal{A}$, and $\otimes_{\mathrm{A}} \colon \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ be given, for each $A_1, A_2 \in \mathcal{A}$, by*

$$\mathbf{0}_{\mathrm{A}} \stackrel{\text{def}}{=} \mathbf{1} \to \mathbf{0},$$
$$\mathbf{1}_{\mathrm{A}} \stackrel{\text{def}}{=} \mathbf{1} \to \mathbf{1},$$
$$A_1 \otimes_{\mathrm{A}} A_2 \stackrel{\text{def}}{=} \rho_{\mathrm{A}}(A_1 \parallel A_2).$$

The projection operators $\exists^{\mathrm{A}}_{\bar{X}} \colon \mathcal{A} \to \mathcal{A}$ are given, for each $\bar{X} \subseteq_{\mathrm{f}}$ Vars and $A \in \mathcal{A}$, by

$$\exists^{\mathrm{A}}_{\bar{X}}\, A \stackrel{\text{def}}{=} \rho_{\mathrm{A}}\bigl(A \mid \bar{X}\bigr),$$

where

$$A \mid \bar{X} \overset{\text{def}}{=} \left\{ \left(\bar{\exists}_{\bar{X}}\, C^a \to \bar{\exists}_{\bar{X}}\, C^t\right) \,\middle|\, \begin{array}{l} (C^a \to C^t) \in A \\ \left(\left(\mathbf{1} \to \bar{\exists}_{\bar{X}}\, C^a\right) \otimes_{\mathrm{A}} A\right) \\ \qquad\qquad \vdash_{\mathrm{A}} \left(\mathbf{1} \to C^a\right) \end{array} \right\}.$$

Finally, let $\oplus_{\mathrm{A}} \colon \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ *be an operator satisfying the conditions* G_2 *and* G_4 *of* Definition 7*. For any indexed family* $\{\mathrm{d}^{\mathrm{A}}_{\bar{X}\bar{Y}}\}_{\bar{X},\bar{Y} \in \mathit{Vars}^\star}$ *of elements of* $\mathcal{A}$*, we will call*

$$\left\langle \mathcal{A}(\bar{\mathcal{D}}), \otimes_{\mathrm{A}}, \oplus_{\mathrm{A}}, \mathbf{0}_{\mathrm{A}}, \mathbf{1}_{\mathrm{A}}, \{\bar{\exists}^{\mathrm{A}}_{\bar{X}}\}, \{\mathrm{d}^{\mathrm{A}}_{\bar{X}\bar{Y}}\} \right\rangle$$

an ask-and-tell constraint system *over* $\bar{\mathcal{D}}$ *and* $\oplus_{\mathrm{A}}$*.*

Notice that, as far as the diagonal elements are concerned, we have left complete freedom. This is because, in an ask-and-tell construction, the induced diagonals $\mathrm{d}^{\mathrm{A}}_{\bar{X}\bar{Y}} = \mathbf{1} \to \mathrm{d}_{\bar{X}\bar{Y}}$ are not necessarily a good choice (see Section 3.6.3 for a simple example).

The projection operators are indeed quite complicated. The problem originates from the requirement N_3 of the normal form of Definition 38. This requirement enforces the need of the absorption rule in the calculus. The rule, by strengthening the ask-constraint of pairs, introduces "false dependencies". Consider, for instance, a constraint system where elements include the finite subsets of $\{\, p(X), q(X), r(X) \mid X \in \mathit{Vars} \,\}$ and the operators $\otimes$ and $\oplus$ are set union and intersection, respectively. The non-normalized cc agent over this constraint system

$$A = \mathbf{1} \to \{q(Y)\} \parallel \{p(X)\} \to \{r(X), q(Y)\}$$

is normalized, by means of the absorption rule, to

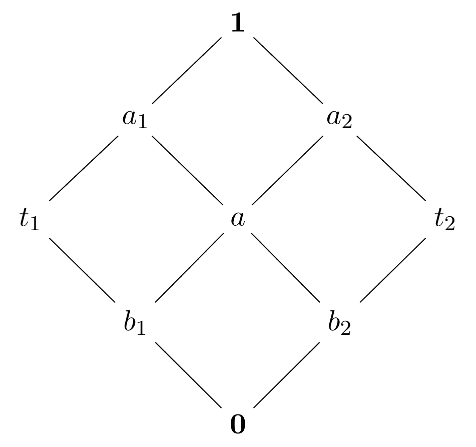
$$A' = \mathbf{1} \to \{q(Y)\} \parallel \{p(X), q(Y)\} \to \{r(X), q(Y)\}.$$

The absorption rule has thus introduced the dependency of $r(X)$ from $q(Y)$, which is indeed false in the context of A' (as it was in the context of A). A definition of the projection operators not taking into account this phenomenon would cause the inaccurate result $\bar{\exists}^{\mathrm{A}}_{X}\, A' = \mathbf{1}_{\mathrm{A}}$. The projection operators given in Definition 44, instead, recognize the false dependency by noting that

$$\{p(X)\} = \bar{\exists}_X \{p(X), q(Y)\}$$

is, in the context of A', equivalent to $\{p(X), q(Y)\}$, that is

$$\mathbf{1} \to \{p(X)\} \parallel A' \quad \vdash_{\mathrm{A}} \quad \mathbf{1} \to \{p(X), q(Y)\}.$$



The agent $\bigl(a_1 \to t_1 \parallel a_2 \to t_2\bigr)$ is normalized. However, even though

$$\bigl(a_1 \to t_1 \parallel a_2 \to t_2\bigr) \vdash_{\mathrm{A}} (a \to \mathbf{0}),$$

we have

$$(a_1 \to t_1) \nvdash_{\mathrm{A}} (a \to \mathbf{0})$$

and

$$(a_2 \to t_2) \nvdash_{\mathrm{A}} (a \to \mathbf{0}).$$

Figure 3.3: The semantic normal form does not help deciding the entailment.

We can thus obtain the expected result $\overline{\exists}^{\mathrm{A}}_{X}\, A' = \{p(X)\} \to \{r(X)\}$. We will see in a moment other problems provoked by the absorption rule and, in turn, by the normal form we employ for agents.

It is interesting to notice that the semantic normal form, while making the task of recognizing equivalent agents trivial (they are equivalent if and only if they have exactly the same pairs), is not so useful in order decide when an agent *entails* another one. An intuitive explanation of this fact is due to Enea Zaffanella, and it is given in Figure 3.3.

Theorem 45 *If $\bar{\mathcal{D}}$ is a constraint system, so is*

$$\bigl\langle |\mathcal{A}(\bar{\mathcal{D}})|, \otimes_{\mathrm{A}}, \oplus_{\mathrm{A}}, \mathbf{0}_{\mathrm{A}}, \mathbf{1}_{\mathrm{A}}, \{\overline{\exists}^{\mathrm{A}}_{\bar{X}}\}, \mathrm{d}^{\mathrm{A}}_{\bar{X}\bar{Y}} \bigr\rangle.$$

Proof An ask-tell pair is a constraint by virtue of point (3) in Definition 5, viewing '$\to$' as a binary (meta-level) predicate. Thus condition G_0 is true by point (2) of the same definition. $\langle \mathcal{A}(\bar{\mathcal{D}}), \otimes_{\mathrm{A}}, \mathbf{1}_{\mathrm{A}} \rangle$ is immediately verified being a commutative and idempotent monoid with zero element $\mathbf{0}_{\mathrm{A}}$, thus satisfying axioms G_1 and G_3. The requirements G_2, G_4, and G_5 are immediate consequences of the definition of ask-and-tell c.s. □

3.6.1 Merge Operators

Even though the ask-and-tell construction is parameterized with respect to a merge operator, it is possible to induce such an operator from the one of the base-level constraint system. Since this is a problematic point we proceed with care.

Suppose that the base-level constraint system $\bar{\mathcal{D}}$ is a lattice. Thus kernel operators over $\bar{\mathcal{D}}$ form again a lattice, where the lub is given, for k_1 and k_2 kernel operators and for each $C \in \mathcal{D}$, by

$$(k_1 \sqcup k_2)(C) \stackrel{\text{def}}{=} k_1(C) \oplus k_2(C), \quad \text{for } C \in \mathcal{D}, \tag{3.21}$$

whose fixed points are

$$(k_1 \sqcup k_2)(\mathcal{D}) = \bigl\{\, C_1 \oplus C_2 \bigm| C_1, C_2 \in k_1(\mathcal{D}) \cup k_2(\mathcal{D}) \,\bigr\}.$$

In terms of kernel operators, as pointed out in [Sar93], this can be thought of as a kind of *determinate disjunction*: $k_1 \sqcup k_2$ gives, on any input C, the strongest common information between k_1 and k_2. The computational significance of this concept has been first recognized in [VSD92a], where determinate disjunction allows for significant improvements in some constraint propagation algorithms.

The problem is that, even when k_1 and k_2 are represented by finite cc agents A_1 and A_2, namely $k_1 = A_1^K$ and $k_2 = A_2^K$, we have no guarantees whatsoever that $k_1 \sqcup k_2$ is representable by a finite cc agent.[12] In other words, (syntactic) finite cc agents are not, in general, closed under the (semantic) lub operation. As a consequence, we must content ourselves with upper bounds (unless we are willing to enrich our representation language with a construct like $A_1 + A_2$ expressing determinate disjunction, and we are not). Observe that the very precise effect of (3.21) can be obtained (at a consequently high cost) by applying a powerset construction (Section 3.5) to the ask-and-tell constraint system considered. This way, when merging two (non-redundant) agents we will keep both of them, thus realizing, in practice, the '+' construct mentioned above. If we do that, obviously, there is no need at all to define a merge operator at the ask-and-tell level.

[12]For the interested reader: consider a constraint system with distinct elements $\{\mathbf{0}, \mathbf{1}\} \cup \{C_i\}_{i \in \mathbb{N}_0}$ and such that $C_i \vdash C_j$ if and only if $1 \leq i \leq j$. The lub of $\mathbf{1} \rightarrow C_0$ and $\mathbf{1} \rightarrow C_1$ is not expressible by a finite cc agent. The same happens if the ordering is such that $C_i \nvdash C_j$ for all i and j.

In our general situation, the base-level constraint system $\bar{\mathcal{D}}$ might not be a lattice, and (3.21) might not define a kernel operator. In these cases, an upper bound on the poset of kernel operators over $\bar{\mathcal{D}}$ can be given as

$$(k_1 \mathbin{\tilde{\sqcup}} k_2)(C) \stackrel{\text{def}}{=} C \otimes \bigl(k_1(C) \oplus k_2(C)\bigr), \quad \text{for } C \in \mathcal{D}, \tag{3.22}$$

which, still, is not guaranteed to correspond to any finite cc agent over $\bar{\mathcal{D}}$.

We stress again that our non-commitment to lattices in the general definition of constraint systems (Section 3.2.3) is not merely dictated by the desire of freely managing the complexity/precision tradeoff. In cases like the one at hand we have no other sensible choice due to representation problems.

Our study of computable merge operators starts with a simple operation merging two (not necessarily normalized) agents into one. This is done, roughly speaking, by taking the meet of the ask constraints, and the merge of the tell constraints.

Definition 46 (Merge operator over agents.) *Consider a constraint system $\bar{\mathcal{D}} = \langle \mathcal{D}, \otimes, \oplus, \mathbf{0}, \mathbf{1}, \{\bar{\exists}_{\bar{X}}\}, \{\mathrm{d}_{\bar{X}\bar{Y}}\}\rangle$, and any two finite cc agents over $\bar{\mathcal{D}}$ in syntactic normal form:*

$$A_1 = \|_{i=1}^{n} C_i^a \to C_i^t \quad \textit{and} \quad A_2 = \|_{j=1}^{m} D_j^a \to D_j^t.$$

Then

$$A_1 \mathbin{\tilde{\oplus}_{\mathrm{A}}} A_2 \stackrel{\text{def}}{=} \mathop{\|}_{i=1}^{n} \mathop{\|}_{j=1}^{m} (C_i^a \to C_i^t) \mathbin{\tilde{\oplus}_{\mathrm{A}}} (D_j^a \to D_j^t), \tag{3.23}$$

where, if we define $C_{ij}^a \stackrel{\text{def}}{=} C_i^a \otimes D_j^a$ and $C_{ij}^t \stackrel{\text{def}}{=} C_i^t \oplus D_j^t$, we have

$$(C_i^a \to C_i^t) \mathbin{\tilde{\oplus}_{\mathrm{A}}} (D_j^a \to D_j^t) \stackrel{\text{def}}{=} \begin{cases} \mathbf{1}_{\mathrm{A}}, & \textit{if } C_{ij}^a \vdash C_{ij}^t; \\ C_{ij}^a \to (C_{ij}^a \otimes C_{ij}^t), & \textit{otherwise.} \end{cases} \tag{3.24}$$

It is easy to see that this syntactic operation corresponds, at the semantic level, to an upper bound.

Proposition 47 *If A_1 and A_2 are as stated in* Definition 46*, then $A_1 \mathbin{\tilde{\oplus}_{\mathrm{A}}} A_2$ is in syntactic normal form. Furthermore, we have both $[\![A_1]\!] \subseteq [\![A_1 \mathbin{\tilde{\oplus}_{\mathrm{A}}} A_2]\!]$ and $[\![A_2]\!] \subseteq [\![A_1 \mathbin{\tilde{\oplus}_{\mathrm{A}}} A_2]\!]$, that is, $A_1 \vdash_{\mathrm{A}} A_1 \mathbin{\tilde{\oplus}_{\mathrm{A}}} A_2$ and $A_2 \vdash_{\mathrm{A}} A_1 \mathbin{\tilde{\oplus}_{\mathrm{A}}} A_2$.*

Proof Preservation of syntactic normal form is clearly guaranteed in (3.24). Then, for each $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, we have $C_i^a \otimes D_j^a \vdash C_i^a$, which implies $\downarrow(C_i^a \otimes D_j^a) \subseteq \downarrow C_i^a$, that is $\overline{\downarrow C_i^a} \subseteq \overline{\downarrow(C_i^a \otimes D_j^a)}$. Similarly, from the correctness of the base-level merge operator, namely $C_i^t \vdash C_i^t \oplus D_j^t$, we get $\downarrow C_i^t \subseteq \downarrow(C_i^t \oplus D_j^t)$. Combining these two relations we obtain, for each i and j,

$$\overline{\downarrow C_i^a} \cup \downarrow C_i^t \subseteq \overline{\downarrow(C_i^a \otimes D_j^a)} \cup \downarrow(C_i^t \oplus D_j^t),$$

and thus, for each $i \in \{1, \dots, n\}$,

$$\overline{\downarrow C_i^a} \cup \downarrow C_i^t \subseteq \bigcap_{j=1}^{m} \bigl(\overline{\downarrow(C_i^a \otimes D_j^a)} \cup \downarrow(C_i^t \oplus D_j^t)\bigr),$$

whence

$$\begin{aligned} [\![A_1]\!] &= \bigcap_{i=1}^{n} [\![C_i^a \to C_i^t]\!] \\ &\subseteq \bigcap_{i=1}^{n} \Bigl[\!\Bigl[\|_{j=1}^{m} C_i^a \otimes D_j^a \to C_i^t \oplus D_j^t \Bigr]\!\Bigr] \\ &= [\![A_1 \mathbin{\tilde{\oplus}_{\mathrm{A}}} A_2]\!] \end{aligned}$$

follows. A symmetric argument proves that $[\![A_2]\!] \subseteq [\![A_1 \mathbin{\tilde{\oplus}_{\mathrm{A}}} A_2]\!]$. □

We now have an obvious merge operator that is completely determined by the underlying, base-level constraint system.

Definition 48 (Canonical ask-and-tell merge operator.) *Let us define $\mathcal{A} \stackrel{\text{def}}{=} |\mathcal{A}(\bar{\mathcal{D}})|$. The operator $\hat{\oplus}_{\mathrm{A}} \colon \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ given, for each $A_1, A_2 \in \mathcal{A}$, by*

$$A_1 \mathbin{\hat{\oplus}_{\mathrm{A}}} A_2 \stackrel{\text{def}}{=} \rho_{\mathrm{A}}(A_1 \mathbin{\tilde{\oplus}_{\mathrm{A}}} A_2)$$

is called the canonical merge operator over $\mathcal{A}$ induced by $\bar{\mathcal{D}}$.

Notice that canonical ask-and-tell constraint systems subsume their base-level c.s., where only "tells" are considered. In fact we have $\eta(C_1) \otimes_{\mathrm{A}} \eta(C_2) = \eta(C_1 \otimes C_2)$ and $\eta(C_1) \mathbin{\hat{\oplus}_{\mathrm{A}}} \eta(C_2) = \eta(C_1 \oplus C_2)$.

Unfortunately, the canonical merge operator turns out to be inaccurate, due to the normal form employed for agents. Consider the ask-and-tell construction applied to the *Con* domain of Section 3.4.1, and the agents in normal form[13]

$$A_1 \stackrel{\text{def}}{=} \mathbf{1} \to Z \parallel XZ \to XYZ \quad \text{and} \quad A_2 \stackrel{\text{def}}{=} X \to XY.$$

It easy to see that the canonical merge operator gives

$$A_1 \mathbin{\hat{\oplus}_{\mathrm{A}}} A_2 \stackrel{\text{def}}{=} \rho_{\mathrm{A}}(A_1 \mathbin{\tilde{\oplus}_{\mathrm{A}}} A_2) = XZ \to XYZ.$$

[13]For simplicity we use juxtaposition instead of the usual set notation.

If we consider the non-normalized agent $A_1' = \mathbf{1} \to Z \parallel X \to XYZ$, we have $[\![A_1']\!] = [\![A_1]\!]$ but $\rho_{\mathrm{A}}(A_1' \tilde{\oplus}_{\mathrm{A}} A_2) = X \to XY$, which is strictly stronger than $A_1 \hat{\oplus}_{\mathrm{A}} A_2$.

The problem can be tracked down, as in the case of the projection operators, to the introduction, by means of the absorption rule, of "unnecessary dependencies" needed to satisfy condition N_3 of the semantic normal form. However, while for projection operators we had a standard solution, here the situation is more difficult. Moreover, looking at the example above, one is caught by the doubt that, perhaps, the choice of the normal form of Definition 38 was a poor one, since it is based on maximal strengthening of both *tell* and *ask* constraints. We are thus lead to the following question:

> Does there exist an alternative normal form for finite cc agents that is based on maximal strengthening of tell constraints and maximal *weakening* of ask constraints?

We might tentatively go as follows. Consider a Noetherian constraint system $\bar{\mathcal{D}}$ and an agent $A = \parallel_{i=1}^{n} C_i^a \to C_i^t$ over $\bar{\mathcal{D}}$ in semantic normal form (Definition 38). For each $j \in \{1, \dots, n\}$, let the agent A_{-j} be given by

$$A_{-j} \stackrel{\mathrm{def}}{=} \mathop{\parallel}_{\substack{i=1 \\ i \neq j}}^{n} C_i^a \to C_i^t,$$

and define the set of constraints

$$D_j \stackrel{\mathrm{def}}{=} \Omega_{\mathcal{D}}^{\vdash}\Bigl(\bigl\{\, C \in \mathcal{D} \bigm| [\![C \to C_j^t \parallel A_{-j}]\!] = [\![A]\!] \,\bigr\}\Bigr).$$

In words, for each ask-tell pair in the original agent A, we consider the set of minimal constraints that can be substituted in the ask part without changing the semantics of A. Observe that, for each $j = 1, \dots, n$, either $D_j = \{C_j^a\}$ or D_j contains only incomparable constraints strictly weaker than C_j^a. Whenever the base-level constraint system $\bar{\mathcal{D}}$ is such to ensure that all the D_j's so obtained will always be finite, then the answer to the above question is positive, and the normal form of A (in this new sense) is given by

$$\mathop{\parallel}_{j=1}^{n} \mathop{\parallel}_{C \in D_j} C \to C_j^t.$$

Such a normal form would give us, in the same spirit of Definition 48, a very precise merge operator while allowing for a simplification of the projection operators. However, even in the cases where this normal form exists, we remain with the problem of computing it. This requires something very different from the rewriting system of Figure 3.2, as we will need mechanisms for

1. *weakening* constraints (now we only *strengthen* them), and
2. *splitting* ask-tell pairs (now we only *combine* them).

Moreover, while our current rewriting system is *local*, in that each rewriting depends on only two pairs, computing the new normal form requires looking at all the agents being normalized. This, together with the fact that splitting can give rise to an exponential number of new pairs, could push us beyond polynomial complexity.

While this is certainly a subject for further study, we now give a general way of defining merge and widening operators for the ask-and-tell construction that are more precise than the canonical merge operator. First of all, let us deal with the problem of constraints' weakening.

Definition 49 (Weakening.) *An operation* $\circledcirc\colon \mathcal{D} \to \mathcal{D}$ *on a constraint system* $\bar{\mathcal{D}} = \langle \mathcal{D}, \otimes, \oplus, \mathbf{0}, \mathbf{1}, \{\bar{\exists}_{\bar{X}}\}, \{\mathrm{d}_{\bar{X}\bar{Y}}\}\rangle$ *that satisfies for each* $C_1, C_2 \in \mathcal{D}$*:*

Q_1. $(C_1 \circledcirc C_2) \otimes C_1 = C_1$,

Q_2. $(C_1 \circledcirc C_2) \otimes C_2 = C_1 \otimes C_2$,

Q_3. $(C_1 \circledcirc C_2) \circledcirc C_2 = C_1 \circledcirc C_2$,

is said a weakening operator for $\bar{\mathcal{D}}$*.*

The intuitive explanation of this axiomatization is as follows. Condition Q_1, which can be restated as $C_1 \vdash C_1 \circledcirc C_2$, means that the weakening operation is correct (it does not add anything). Condition Q_2 states that weakenings are not too aggressive: a weakened constraint can be restored by *adding* what was *taken out*. Q_3 says that taking out twice the same thing is pointless. Observe that these conditions are very weak, while being sufficient for what follows.

A class of powerful weakening operators is defined next.

Definition 50 (Best weakening operator.) *A weakening operator over a constraint system* $\bar{\mathcal{D}}$ *is a* best weakening *if and only if, for each* $C_1, C_2 \in \mathcal{D}$*,*

Q_4. $\forall C \in \mathcal{D} : C \otimes C_2 = C_1 \otimes C_2 \implies C \vdash (C_1 \circledcirc C_2)$.

Notice that conditions Q_2 and Q_4 are sufficient to define best weakenings, as Q_1 and Q_3 necessarily follow. Best weakenings are also monotonic on the first argument and anti-monotonic on the second one.[14]

[14]The connection between our weakenings and the class of *weak relative pseudo-complements* over meet-semilattices, recently introduced in [GPR95], needs to be studied.

Example 51 (Weakening over intervals.) *Consider a domain for numerical bounds analysis based on intervals. For instance, take the simple constraint system of* Section 3.3.4*, restricted to the intervals component, and apply to it the determinate c.s. construction. A weakening operator can be defined along the following lines, considering, for simplicity, only closed intervals:*

$$[\, l_1, u_1 \,] \circledcirc [\, l_2, u_2 \,] \stackrel{\mathrm{def}}{=} [\, l, u \,],$$

where

$$l \stackrel{\mathrm{def}}{=} \begin{cases} -\infty, & \text{if } l_1 \leq l_2; \\ l_1, & \text{otherwise;} \end{cases} \quad \text{and} \quad u \stackrel{\mathrm{def}}{=} \begin{cases} +\infty, & \text{if } u_1 \geq u_2; \\ u_1, & \text{otherwise.} \end{cases}$$

Such an operator is easily verified being a weakening. It is also monotonic on the first argument and anti-monotonic on the second one. All these extra-properties, however, are not necessary for what follows.

The following lemma is useful for what follows.

Lemma 52 *Let $\circledcirc$ be a weakening over a c.s. $\bar{\mathcal{D}}$. For each $C_1, C_2 \in \mathcal{D}$, if $C \in \mathcal{D}$ is such that $C \vdash C_2$ and $C \nvdash C_1$, then $C \nvdash (C_1 \circledcirc C_2)$.*

Proof Suppose that $C \vdash C_2$, $C \nvdash C_1$, and $C \vdash (C_1 \circledcirc C_2)$. In other words, we have $C \otimes C_2 = C$, $C \otimes C_1 \neq C$, $C \otimes (C_1 \circledcirc C_2) = C$, and, by axiom Q_2, $(C_1 \circledcirc C_2) \otimes C_2 = C_1 \otimes C_2$. So,

$$\begin{aligned} C \otimes C_2 &= \bigl(C \otimes (C_1 \circledcirc C_2)\bigr) \otimes C_2 \\ &= C \otimes \bigl((C_1 \circledcirc C_2) \otimes C_2\bigr) \\ &= C \otimes (C_1 \otimes C_2) \\ &= (C \otimes C_2) \otimes C_1 \\ &= C \otimes C_1 \\ &\neq C, \end{aligned}$$

thus reaching the absurd conclusion $C \nvdash C_2$. □

We are now in position to define a class of procedures for weakening the ask constraints of finite cc agents while preserving the semantics. As this operation is somewhat opposite to the absorption rewrite rule of our rewriting system, we call it *de-absorption.* This involves the possibility of *splitting* ask-tell pairs.

Definition 53 (De-absorption step.) *Let $A = \|_{i=1}^{n} C_i^a \to C_i^t$ be an agent in syntactic normal form over a c.s. $\bar{\mathcal{D}}$, and let $\circledcirc$ be a weakening over $\bar{\mathcal{D}}$. Then A' is obtained from A by means of a* de-absorption step based on

$\circledcirc$ *if and only if* $h, k \in \{1, \dots, n\}$ *are such that* $C^a_h \vdash C^a_k$, *the condition* $C^a_h \neq (C^a_h \circledcirc C^t_k) \otimes C^a_k$ *holds, and*

$$A' = \big((C^a_h \circledcirc C^t_k) \otimes C^a_k \to C^t_h\big) \parallel A.$$

Observe that any de-absorption step results in the strict weakening of an ask constraint. In fact, we have $C^a_h \vdash (C^a_h \circledcirc C^t_k)$ by Q_1, and $C^a_h \vdash C^a_k$ by hypothesis, thus $C^a_h \Vdash (C^a_h \circledcirc C^t_k) \otimes C^a_k$.

Lemma 54 *If an agent A' is obtained by a de-absorption step from A, then* $[\![A']\!] = [\![A]\!]$.

Proof In the hypotheses of Definition 53, let $D^a_h \overset{\text{def}}{=} (C^a_h \circledcirc C^t_k) \otimes C^a_k$. We will prove that semantics is *locally* preserved, namely

$$[\![C^a_h \to C^t_h \parallel C^a_k \to C^t_k]\!] = [\![D^a_h \to C^t_h \parallel C^a_k \to C^t_k]\!].$$

From the hypotheses we have $C^t_h \Vdash C^a_h \vdash C^a_k$ and $C^t_h \Vdash D^a_h \vdash C^a_k$. We can thus derive the following facts:

- since $C^t_h \Vdash C^a_k$ we have $\downarrow C^t_h \subset \downarrow C^a_k$ and $\downarrow C^t_h \cap \overline{\downarrow C^a_k} = \varnothing$;
- since $C^a_h \vdash C^a_k$ we have $\overline{\downarrow C^a_k} \subseteq \overline{\downarrow C^a_h}$ and $\overline{\downarrow C^a_h} \cap \overline{\downarrow C^a_k} = \overline{\downarrow C^a_k}$;
- similarly, from $D^a_h \vdash C^a_k$ we have $\overline{\downarrow D^a_h} \cap \overline{\downarrow C^a_k} = \overline{\downarrow C^a_k}$;
- since $C^t_h \Vdash C^a_h$ we have $\downarrow C^t_h \subset \downarrow C^a_h$ and $\downarrow C^t_h \cap \overline{\downarrow C^a_h} = \varnothing$;
- similarly, from $C^t_h \Vdash D^a_h$ we have $\downarrow C^t_h \cap \overline{\downarrow D^a_h} = \varnothing$;
- since $C^t_k \Vdash C^a_k$ we have $\downarrow C^t_k \subset \downarrow C^a_k$ and $\downarrow C^t_k \cap \overline{\downarrow C^a_k} = \varnothing$.

These allow us to state that

$$\begin{aligned}
[\![C^a_h \to C^t_h \parallel C^a_k \to C^t_k]\!] &= \big(\overline{\downarrow C^a_h} \cup \downarrow C^t_h\big) \cap \big(\overline{\downarrow C^a_k} \cup \downarrow C^t_k\big) \\
&= \big(\overline{\downarrow C^a_h} \cap \overline{\downarrow C^a_k}\big) \cup \big(\overline{\downarrow C^a_h} \cap \downarrow C^t_k\big) \cup \\
&\qquad \big(\downarrow C^t_h \cap \overline{\downarrow C^a_k}\big) \cup \big(\downarrow C^t_h \cap \downarrow C^t_k\big) \\
&= \overline{\downarrow C^a_k} \cup \big(\overline{\downarrow C^a_h} \cap \downarrow C^t_k\big) \cup \big(\downarrow C^t_h \cap \downarrow C^t_k\big) \\
&= \overline{\downarrow C^a_k} \uplus \big(\overline{\downarrow C^a_h} \cap \downarrow C^t_k\big) \uplus \big(\downarrow C^t_h \cap \downarrow C^t_k\big),
\end{aligned}$$

and, by the same reasoning,

$$[\![D^a_h \to C^t_h \parallel C^a_k \to C^t_k]\!] = \overline{\downarrow C^a_k} \uplus \big(\overline{\downarrow D^a_h} \cap \downarrow C^t_k\big) \uplus \big(\downarrow C^t_h \cap \downarrow C^t_k\big),$$

where $\uplus$ denotes disjoint union. Since $C^a_h \Vdash D^a_h$ implies

$$\big(\overline{\downarrow C^a_h} \cap \downarrow C^t_k\big) \supseteq \big(\overline{\downarrow D^a_h} \cap \downarrow C^t_k\big),$$

what we are left to show is that

$$\left(\overline{\downarrow C_h^a} \cap \downarrow C_k^t\right) \subseteq \left(\overline{\downarrow D_h^a} \cap \downarrow C_k^t\right),$$

that is to say $\forall C \in \mathcal{D} : (C \vdash C_k^t \wedge C \nvdash C_h^a) \implies C \nvdash D_h^a$. Now Lemma 52 comes in handy, as from $C \vdash C_k^t$ and $C \nvdash C_h^a$ we can derive $C \nvdash (C_h^a \circledcirc C_k^t)$ and thus $C \nvdash (C_h^a \circledcirc C_k^t) \otimes C_k^a = D_h^a$. $\square$

We now define the promised class of procedures.

Definition 55 (De-absorption procedure.) *A* de-absorption procedure *is any algorithm transforming a finite* cc *agent in syntactic normal form, such that it can be characterized as follows:*

Phase 1. *Transform the input agent A into A' by performing any number of de-absorption steps;*

Phase 2. *Transform A' into the output agent A'' by applying the rewriting system of* Figure 3.2 restricted to the structural and reduction rules*.*

A de-absorption procedure will be called maximal *if it applies all the possible de-absorption steps.*

It is now possible to prove the following result.

Theorem 56 *Any de-absorption procedure preserves the semantics of agents.*

Proof Immediate from Lemma 54 and Lemma 40. $\square$

In all those cases where we have a de-absorption procedure that is a function over $|\mathcal{A}(\bar{\mathcal{D}})|$ we have an obvious way to define a merge operator: by applying the *syntactic* merge operator of Definition 48 to de-absorbed agents.

Definition 57 (Merge operator with de-absorption.) *Let $\bar{\mathcal{D}}$ be a constraint system and $\delta_{\circledcirc} \colon |\mathcal{A}(\bar{\mathcal{D}})| \to \mathcal{A}(\bar{\mathcal{D}})$ be a de-absorption procedure. The* merge operator based on $\delta_{\circledcirc}$ *is given, for each $A_1, A_2 \in |\mathcal{A}(\bar{\mathcal{D}})|$, by*

$$A_1 \dot{\oplus}_{\mathrm{A}} A_2 \stackrel{\text{def}}{=} \rho_{\mathrm{A}}\bigl(\delta_{\circledcirc}(A_1) \tilde{\oplus}_{\mathrm{A}} \delta_{\circledcirc}(A_2)\bigr).$$

Any such operator, by virtue of Theorem 56 and Proposition 47, is clearly a merge operator in the sense of Definition 7. De-absorption procedures that are not functions are still useful for designing widening operators.

We now quickly show some examples of ask-and-tell constraint systems. For the more exciting things we have to wait until the next section, where combination of constraint domains are introduced.

3.6.2 More Bounds and Relations Analysis for Numeric Domains

Ask-and-tell constraint systems are suitable for modeling approximate inference techniques that are very useful in a practical setting. Following Section 3.3.4, there is another technique that is used for the analysis described in Chapter 5: *relational arithmetic* [Sim86]. This technique allows to infer constraints on the qualitative relationship of an expression to its arguments. Consider the simple constraint system of Section 3.3.4, and apply to it the determinate construction of Section 3.4. Now apply the ask-and-tell construction to the result. Relational arithmetic can be described by a number of (concurrent) agents. Here are some of them, where x and y are arithmetic expressions, and $\bowtie$ ranges in $\mathsf{R} \overset{\text{def}}{=} \{=, \neq, \leq <, \geq, >\}$:

$$\begin{aligned}
\text{ask}(x \bowtie 0) &\to \text{tell}\bigl((x+y) \bowtie y\bigr)\\
\text{ask}(x > 0 \wedge y > 0 \wedge x \bowtie 1) &\to \text{tell}\bigl((x*y) \bowtie y\bigr)\\
\text{ask}(x > 0 \wedge y < 0 \wedge x \bowtie 1) &\to \text{tell}\bigl(y \bowtie (x*y)\bigr)\\
\text{ask}(x > 0 \wedge y < 0 \wedge x \bowtie -y) &\to \text{tell}\bigl(-1 \bowtie (x/y)\bigr)\\
\text{ask}(x \bowtie y) &\to \text{tell}(e^x \bowtie e^y)
\end{aligned}$$

An example of inference is deducing that $X + 1 \leq Y + 2X + 1$ from the hypotheses $X \geq 0$ and $Y \geq 0$. Notice that there is no restriction to linear constraints.

3.6.3 Definiteness Analysis: ***Def***

The prototypical example of data-flow analysis taking advantage of dependency information is definiteness analysis. In our setting a domain for definiteness can be obtained as follows. Take the *atomic* s.c.s. of Section 3.3.1. Apply to it the determinate construction as outlined in Section 3.4.1. Now apply the ask-and-tell construction to the result, with the merge operator obtained along the lines of Definition 57 choosing:

1. diagonal elements like[15]

$$\mathrm{d}^{\mathrm{A}}_{XY} \overset{\text{def}}{=} \{X\} \to \{X, Y\} \parallel \{Y\} \to \{X, Y\};$$

2. set-theoretic difference as weakening operator;

3. the maximal de-absorption procedure (i.e., the one that applies all the possible de-absorption steps).

[15]Here only in the monadic case, for simplicity.

It can be shown that the domain so obtained is *Def* [Dar91, AMSS]. Its elements can keep track of non-trivial dependencies like the ones induced by symbolic and numeric constraints. For example, the dependencies of $A = f(B, C)$ and $A + B + C = 0$ are captured, respectively, by the agents

$$\{A\} \to \{B, C\} \parallel \{B, C\} \to \{A\},$$

and

$$\{A, B\} \to \{C\} \parallel \{A, C\} \to \{B\} \parallel \{B, C\} \to \{A\}.$$

This example gives us the possibility of pointing out that the entire business of weakenings and de-absorption procedures is not something we can easily avoid. When using definite sentences to represent dependencies, as in our case and in the representations for *Def* studied in [AMSS], obtaining a maximal weakening of the antecedents is crucial for obtaining precise merge operators, let alone for computing the join when it exists. Our present requirement of employing maximal de-absorption corresponds to the requirement, in the representations studied in [AMSS], of the sentences being in *orthogonal* form (which has its costs, since orthogonality must be obtained and preserved by all the domain's operations). In [AMSS] a merge operator is also presented, for the representation $\mathrm{RCNF}_{\mathit{Def}}$, intended to trade precision for efficiency. It does that by not insisting on orthogonality, which in our setting corresponds to the use of a partial de-absorption procedure.

3.6.4 Definiteness Analysis: More than *Pos*

Pos is (like *Def*) a domain of boolean functions [CFW91, AMSS]. It consists precisely of those functions assuming the true value under the *everything-is-true* assignment. In [AMSS] it is shown that *Pos* is strictly more precise than *Def* for groundness analysis. If we apply the powerset construction of Section 3.5 to the ask-and-tell c.s. of the previous section we obtain a very precise (and complex) domain for simple dependencies. In [GR96] it is referred to as $\mho(\mathit{Def})$ (where $\mho$ denotes disjunctive completion) and is shown to be equivalent to $\mho(\mathit{Pos})$. On the other hand, in [FR94] it has been shown that $\mho(\mathit{Pos})$ is strictly more precise than *Pos*, even though this extra-precision is not needed for definiteness analysis.

3.7 Combination of Domains

It is well-known that different data-flow analyses can be combined together. In the framework of abstract interpretation this can be achieved by means of standard constructions such as reduced product and down-set completion [CC79, CC92a]. The key point is that the combined analysis can be more precise than each of the component ones for they can mutually improve each other. However, the degree of cross-fertilization is highly dependent on the degree and quality of interaction taking place among the component domains.

We now propose a general methodology for domain combination with asynchronous interaction. The interaction among domains is asynchronous in that it can occur at any time, or, in other words, it is not synchronized with the domain's operations.

This is achieved by considering ask-and-tell constraint systems built over *product* constraint systems. These constraint systems allow to express communication among domains in a very simple way. They also inherit all the semantic elegance of concurrent constraint programming languages, which provide the basis for their construction. Recently, a methodology for the combination of abstract domains has been proposed in [CLV94], which is directly based on low-level actions such as *tests* and *queries*. While the approach in [CLV94] is immediately applicable to a wider range of analyses (including the ones dealing with non-monotonic properties) the approach we follow here for our restricted set of analyses has the merit of being much more elegant.

We start with a finite set of constraint systems each expressing some properties of interest, and we wish to combine them so to as:

1. perform all the analyses at the same time; and
2. have the domains cooperate to the intent of mutually improving each other.

The first goal is achieved by considering the product of the given constraint systems.

3.7.1 Product Constraint Systems

The product construction over constraint systems is absolutely standard.

Definition 58 (Direct product of constraint systems.) *Given a finite family of constraint systems*

$$\bar{\mathcal{D}}_i = \langle \mathcal{D}_i, \otimes_i, \oplus_i, \mathbf{0}_i, \mathbf{1}_i, \{\bar{\exists}^i_{\bar{X}}\}, \{\mathrm{d}^i_{\bar{X}\bar{Y}}\}\rangle, \quad \textit{for } i = 1, \ldots, n,$$

the product constraint system of $\bar{\mathcal{D}}_1, \ldots, \bar{\mathcal{D}}_n$ *is the algebraic direct product of $\bar{\mathcal{D}}_1, \ldots, \bar{\mathcal{D}}_n$, and is denoted by*

$$\prod_{i=1}^{n} \bar{\mathcal{D}}_i = \langle \mathcal{D}_\times, \otimes_\times, \oplus_\times, \mathbf{0}_\times, \mathbf{1}_\times, \{\bar{\exists}^\times_{\bar{X}}\}, \{\mathrm{d}^\times_{\bar{X}\bar{Y}}\}\rangle.$$

Thus all the operations of a product c.s. are defined coordinate-wise and, in particular, we have

$$\langle C'_1, \ldots, C'_n\rangle \vdash_\times \langle C''_1, \ldots, C''_n\rangle \quad\Longleftrightarrow\quad \forall i = 1, \ldots, n : C'_i \vdash C''_i.$$

An alternative way of obtaining a product constraint system is to start from a collection of simple constraint systems and then to apply the determinate construction.

Definition 59 (Product of simple constraint systems.) *The* product *of a finite family of simple constraint systems,*

$$\mathcal{S}_i = \langle \mathcal{C}_i, \vdash_i, \bot_i, \top_i \rangle, \qquad \textit{for } i = 1, \ldots, n,$$

is the structure given by

$$\prod_{i=1}^{n} \mathcal{S}_i \stackrel{\text{def}}{=} \langle \mathcal{C}_\times, \vdash_\times, \bot_\times, \top_\times \rangle,$$

where the product tokens are

$$\begin{aligned}
\mathcal{C}_\times \stackrel{\text{def}}{=} &\bigl\{ (c_1, \top_2, \ldots, \top_n) \bigm| c_1 \in \mathcal{C}_1 \bigr\} \\
&\cup \bigl\{ (\top_1, c_2, \top_3, \ldots, \top_n) \bigm| c_2 \in \mathcal{C}_2 \bigr\} \\
&\vdots \\
&\cup \bigl\{ (\top_1, \ldots, \top_{n-1}, c_n) \bigm| c_n \in \mathcal{C}_n \bigr\} \\
&\cup \{\bot_\times\}.
\end{aligned}$$

The product entailment is defined as the least relation satisfying conditions E_1–E_5 of Definition 23 *and the following ones, for each $C \in \wp_{\mathrm{f}}(\mathcal{C}_\times)$:*

$$\begin{array}{ccc}
\pi_1(C) \vdash_1 c_1 & \Longrightarrow & C \vdash_\times (c_1, \top_2, \ldots, \top_n), \\
\vdots & \vdots & \vdots \\
\pi_n(C) \vdash_n c_n & \Longrightarrow & C \vdash_\times (\top_1, \ldots, \top_{n-1}, c_n).
\end{array}$$

Finally, $\bot_\times \stackrel{\text{def}}{=} (\bot_1, \ldots, \bot_n)$ *and* $\top_\times \stackrel{\text{def}}{=} (\top_1, \ldots, \top_n)$.

The product simple constraint system so obtained is to be used, together with a suitable merge operator, as the given s.c.s. in the construction of Definition 26. This yields a product constraint system.

Taking the product of constraint systems, we have realized the simplest form of domain combination. It corresponds to the direct product construction of [CC79], allowing for different analyses to be carried out at the same time. Notice that there is no communication at all among the domains.

However, as soon as we consider the ask-and-tell constraint system built over the product, we can express asynchronous communication among the domains in complete freedom. At the very least we would like to have the *smash product* among the component domains. This is realized by the agent

$$\mathop{\big\|}_{i=1}^{n} \mathbf{0}_i \to \mathbf{0}_\times. \tag{3.25}$$

To say it operationally, the *smash* agent globalizes the (local) failure on any of the component domains. This is the only domain-independent agent we have.

Things become much more interesting when instantiated over particular constraint domains. In the CLP($\mathcal{R}$) system [JMSY92b] non-linear constraints (like $X = Y * Z$) are delayed (i.e., not treated by the constraint solver) until they become linear (e.g., until either Y or Z is constrained to take a single value). In standard semantic treatments this is modeled in the operational semantics by carrying over, besides the sequence of goals yet to be solved, a set of delayed constraints. Constraints are taken out from this set (and incorporated into the constraint store) as soon as they become linear.

We believe that this can be viewed in an alternative way that is more elegant, as it easily allows for taking into account the delay mechanism also in the bottom-up semantics, and makes sense from an implementation point of view. The basic claim is the following: CLP($\mathcal{R}$) has *three* computation domains: Herbrand, $\mathbb{R}$ (well, an approximation of it), and *definiteness*.

In other words, it also manipulates, besides the usual ones, constraints of the kind $\mathit{ground}^\natural(X)$, which is interpreted as the variable X being definitively bound to a unique value. We can express the semantics of CLP($\mathcal{R}$) (at a certain level of abstraction) with delay of non-linear constraints by considering the ask-and-tell constraint system over the product of the above three domains. In this view, a constraint of the form $X = Y * Z$ in a program actually corresponds to the agent

$$\mathrm{ask}\bigl(\mathit{ground}^\natural(Y); \mathit{ground}^\natural(Z)\bigr) \rightarrow \mathrm{tell}(X = Y * Z).$$

In fact, any CLP($\mathcal{R}$) user *must* know that $X = Y * Z$ is just a shorthand for that agent! (A similar treatment could probably be done for logic programs with delay declarations.)

Obviously, this cannot be forgotten in abstract constraint systems intended to formalize correct data-flow analyses of CLP($\mathcal{R}$). Referring back to Sections 3.3.4 and 3.6.2, when the abstract constraint system extracts information from non-linear constraints, for example with the agent

$$A = \mathrm{ask}(Y > 0 \wedge Z > 0 \wedge Y \bowtie 1) \rightarrow \mathrm{tell}\bigl((Y * Z) \bowtie Z\bigr)$$

of relational arithmetic, you cannot simply let $X = Y * Z$ stand by itself. By doing this you would incur the risk of *overshooting* the concrete constraint system (thus loosing soundness), which is unable to deduce anything from non-linear constraints. The right thing to do is to combine the numeric abstract constraint system with one for definiteness (by the product and the ask-and-tell constructions) and using, instead of A, the agent

$$A' = \mathrm{ask}\bigl(\mathit{ground}^\sharp(Y); \mathit{ground}^\sharp(Z)\bigr) \rightarrow A.$$

Beware not to confuse $\mathit{ground}^\natural(X)$ with $\mathit{ground}^\sharp(X)$. The first is the *concrete one*: X is definite if and only if $\mathit{ground}^\natural(X)$ is entailed in the current concrete store. In contrast, having $\mathit{ground}^\sharp(X)$ entailed in the *abstract* constraint store at some program point, and assuming a correct definiteness analysis, means that X is certainly bound to a unique value in the concrete computation at that program point. The converse, of course, does not necessarily hold.

Let us see another example. The analysis described in [Han93] aims at the compile-time detection of those non-linear constraints that will become linear at run time. This analysis is important for remedying the limitation of CLP($\mathcal{R}$) to linear constraints by incorporating powerful (and computationally complex) methods from computer algebra as the ones employed in the RISC-CLP(Real) system [Hon93]. With the results of the above analysis this extension can be done in a smooth way: non-linear constraints that are guaranteed to become linear will be simply delayed, while only the other non-linear constraints will be treated with the special solving techniques. Thus, programs not requiring the extra power of these techniques will be hopefully recognized as such, and will not pay any penalties. The analysis of [Han93] is a kind of definiteness. One of its difficulties shows up when considering the simplest non-linear constraint: $X = Y * Z$. Clearly X is definite if Y and Z are such. But we cannot conclude that the definiteness of Y follows from the definiteness of X and Z, as we also need the condition $Z \neq 0$. Similarly, we would like to conclude that X is definite if Y or Z has a zero value. Thus we need approximations of the concrete values of variables (i.e., bounds analysis), something that is not captured by common definiteness analyses while being crucial when dealing with non-linear constraints. If we take the ask-and-tell construction over the product of a constraint system for definiteness with a numerical one, we can solve the problem. The concrete constraint $X = Y * Z$ would be *abstractly compiled* into the agent

$$\begin{aligned}
&\mathrm{ask}\bigl(\mathit{ground}^\sharp(Y) \wedge \mathit{ground}^\sharp(Z)\bigr) \rightarrow \mathrm{tell}\bigl(\mathit{ground}^\sharp(X)\bigr)\\
\parallel\ &\mathrm{ask}(Y = 0; Z = 0) \rightarrow \mathrm{tell}\bigl(\mathit{ground}^\sharp(X)\bigr)\\
\parallel\ &\mathrm{ask}\bigl(\mathit{ground}^\sharp(X) \wedge \mathit{ground}^\sharp(Z) \wedge Z \neq 0\bigr) \rightarrow \mathrm{tell}\bigl(\mathit{ground}^\sharp(Y)\bigr)\\
\parallel\ &\mathrm{ask}\bigl(\mathit{ground}^\sharp(X) \wedge \mathit{ground}^\sharp(Y) \wedge Y \neq 0\bigr) \rightarrow \mathrm{tell}\bigl(\mathit{ground}^\sharp(Z)\bigr).
\end{aligned}$$

Of course, this is significantly more precise than the *Def* formula $X \leftarrow Y \wedge Z$.

We have thus reconciled the two running examples of this chapter (numerical bounds/relations and definiteness) by showing two examples of combination. In fact, when analyzing CLP($\mathcal{R}$) programs, there is a bidirectional flow of information: definiteness information is required for a correct handling of delayed constraints and thus for deriving more precise numerical patterns that, in turn, are used to provide more precise definiteness information. There is another obvious way in which numerical bounds and relations improve definiteness (and any other analysis, indeed): by excluding computation paths that are doomed to fail (this is modeled in a domain-independent way by the *smash agent* seen above). We are thus requiring a quite complicated interaction between domains. It is even more complicated if you consider that the numerical component we have sketched is the combination (in the sense of the present section) of a domain for intervals with one for arithmetic relationships (even though, for simplicity, it was not presented in that way).

3.7.2 Approximating Built-ins Behavior

The techniques we propose are suitable for approximating the behavior of several common built-ins. Consider, for instance, the `functor/3` built-in. Consider a product constraint system with four components: one for *simple types* (for instance, the one of Section 3.3.2), one for definiteness, one incorporating numerical information (including at least signs, e.g., tokens of the kind $X \geq 0$, $X > 0$ and $X = 0$), and one involving symbolic, structural information (e.g., the one of Section 3.4.2). Then, the (success) semantics of `functor(T, F, N)` can be approximated easily and quite precisely by means of the following finite agent over the product:

$$\begin{aligned}
&\mathrm{tell}\bigl(\mathit{symbolic}(T), \mathit{atom}(F), \mathit{ground}(F)\bigr)\\
&\parallel \mathrm{tell}\bigl(\mathit{integer}(N), N \geq 0, \mathit{ground}(N)\bigr)\\
&\parallel \mathrm{ask}\bigl(\mathit{atom}(T); N = 0; T = F\bigr)\\
&\qquad\quad \to \mathrm{tell}\bigl(\mathit{atom}(T), \mathit{ground}(T), N = 0, T = F\bigr)\\
&\parallel \mathrm{ask}\bigl(\mathit{compound}(T); N > 0; T \neq F\bigr)\\
&\qquad\quad \to \mathrm{tell}\bigl(\mathit{compound}(T), N > 0, T \neq F\bigr).
\end{aligned}$$

3.8 Conclusion and Future Work

We have shown a notion of constraint system that is general enough to encompass both the concrete domains of computation of actual constraint logic-based languages, and several of their abstract interpretations useful for data-flow analysis. We have also shown how these constraint systems are integrated within an appropriate framework for the definition of non-standard semantics of constraint logic-based languages. Some significant members of the introduced class of constraint systems have been presented, together with construction techniques that induce a hierarchy of domains. These domains have several nice features, both from a theoretical and an experimental viewpoint.

In particular, we have proposed a general methodology for domain combination with asynchronous interaction. In this kind of combination the communication among domains can be expressed in a very simple way. The methodology also inherits all the semantic elegance of concurrent constraint programming languages.

Ask-and-tell constraint systems have been satisfactorily implemented in the China analyzer. Of course, since we strive (but not too much) for efficient analyzers, we have not implemented them *exactly* as described here. We have, however, a firm theoretical basis on which to base our implementation tricks. We have also implemented domain combinations along the lines of Section 3.7. In China, following an abstract compilation approach, CLP($\mathcal{R}$) constraints are compiled into finite cc agents over a product of domains for simple types, definiteness, numerical bounds and relations, and generic patterns.

Future work includes studying in depth the problem of the semantic normal form for finite cc agents, both in general and in particular cases. The aim is to find more satisfactory solutions to the problem of merging finite cc agents. We also would like to answer the following question: are there variations of these ideas that are applicable also to analysis oriented towards non-monotonic or "non-logical" properties (like variable sharing and freeness)?

Chapter 4

Structural Information Analysis

Contents

4.1 Introduction

It is important to make a clear distinction between the language CLP($\mathcal{H}$, $\mathcal{X}$), where the Herbrand component $\mathcal{H}$ is completely separated from the domain $\mathcal{X}$, and the language CLP($\mathcal{H}_\mathcal{X}$), where $\mathcal{H}$ and $\mathcal{X}$ are somewhat amalgamated. Of course, CLP($\mathcal{H}$, $\mathcal{X}$) languages are simpler and less expressive, though still useful. The simplicity come from the fact that interpreted terms (arithmetic expression, for instance) are not allowed to occur as leaves of

```
length([], 0).
length([H|T], N) :-
    length(T, N-1).
```

Figure 4.1: List-length in CLP($\mathcal{H}$, $\mathcal{N}$).

Herbrand terms. A simple example program that is expressible in a language of the kind CLP($\mathcal{H}$, $\mathcal{N}$), with $\mathcal{N}$ any numeric domain supporting linear equations, is a reversible version of the Prolog *list-length* program: it is given in Figure 4.1.

When interpreted terms are allowed to occur in Herbrand structures more complex programs can be built. For instance, one can express unbounded lists where interpreted terms occur. The program in Figure 4.2 makes use of this possibility. It is a variant of a program from a paper of Alain Colmerauer [Col90]. It solves a rather difficult problem: in the wording of the cited paper, the program is a "program for filling a rectangle of unknown shape by n squares of unknown but different sizes" (*op. cit.*, p. 82).[1]

Independently of the need for "unbounded containers", it is a common CLP idiom to embed interpreted terms into Herbrand terms. As a final example we reproduce in Figure 4.3 a fragment of a program for solving intersection problems involving two solids.

The above discussion is motivated by the fact that several analysis methods presented in the literature, either assume a CLP($\mathcal{X}$) or CLP($\mathcal{H}$, $\mathcal{X}$) language, or tolerate severe precision penalties when interpreted functors occur as leaves of uninterpreted functors, or are rather evasive about how the analysis technique presented is extended from CLP($\mathcal{X}$) to CLP($\mathcal{H}_\mathcal{X}$).

From the experience gained with the first prototype version of CHINA [Bag94] it was clear that, in order to attain a significant precision in the analysis of numerical constraints in CLP($\mathcal{H}_\mathcal{N}$) languages, one must keep at least part of the uninterpreted terms in concrete form. Note that almost any analysis is more precise when this kind of structural information is retained to some extent: in our case the precision loss was just particularly acute.

Cortesi *et al.* [CLV93, CLV94] have a nice proposal in this respect. Using their terminology, they defined a generic abstract domain $\mathtt{Pat}(\Re)$ that automatically upgrades a domain $\Re$ (which must support a certain set of elementary operations) with structural information. Their approach is limited to the analysis of logic programs, but this restriction can be easily removed. However, the presentation in [CLV93] has several drawbacks. First of all, they define a *specific implementation* of the generic structural domain. The implementation is forcedly cluttered with details that make the general prin-

[1]The version in Figure 4.2 is due to Niels Jørgensen.

```
fill_rectangle(A,N,C) :-
    A >= 1,
    all_different(N,C),
    fill([1,0,0,A,1],C,_,[]).

all_different(0,[]).
all_different(N + 1,[[_,_,B]|T]) :-
    N >= 0,
    all_different(N,T),
    different(B,T).

different(_,[]).
different(B,[[_,_,C]|T]) :-
    B <> C,
    different(B,T).

fill([Y0,X1,Y1|L],C,[Y0,X1,Y1|L],C) :-
    Y0 <= Y1.
fill([Y0,X1,Y1|L],[[X1,Y1,B]|C],L3,C3) :-
    Y0 > Y1,
    Y1 + B <= 1,
    place_square([X1,Y1|L],[X1 + B|L1]),
    fill([Y1 + B,X1 + B|L1],C,L2,C2),
    fill([Y0,X1|L2],C2,L3,C3).

place_square([_,Y1,X2,Y2|L],L1) :-
    Y1 = Y2,
    place_square([X2,Y2|L],L1).
place_square([_,_|L],L).
place_square([X1,Y1,X2|L],[X1e,Y1,X2|L]) :-
    X1 < X1e,
    X1e < X2.
```

Figure 4.2: The `fill_rectangle` program.

```
%% Tests d'inclusion pour le solide primitif "sphere".
dedans((X,Y,Z), solide(sphere(Cx,Cy,Cz,R))) :-
    (X-Cx)*(X-Cx) + (Y-Cy)*(Y-Cy) + (Z-Cz)*(Z-Cz) <= R*R.

%% Tests d'inclusion pour le solide primitif "cylindre".
dedans((X,Y,Z), solide(cylindre((X0,Y0,Z0),(X1,Y1,Z1),R))) :-
    % vecteur directeur de l'axe de symetrie
    Vx = X1-X0, Vy = Y1-Y0, Vz = Z1-Z0,
    % le point (Xp,Yp,Zp) est sur l'axe de symetrie,...
    Xp = Vx*T + X0,
    Yp = Vy*T + Y0,
    Zp = Vz*T + Z0,
    % ... a l'interieur du cylindre...
    T >= 0, T <= 1,
    % ... et sur le plan orthogonal a l'axe contenant (X,Y,Z)
    Vx*(X-Xp) + Vy*(Y-Yp) + Vz*(Z-Zp) = 0,
    % contraindre le cylindre
    (X-Xp)*(X-Xp) + (Y-Yp)*(Y-Yp) + (Z-Zp)*(Z-Zp) <= R*R.
```

Figure 4.3: A fragment of `csg.clpr`.

ciples difficult to understand. Moreover, they describe an implementation of the *pattern component* (taking care of representing the terms in concrete form) that appears to be unnecessarily complicated. Their representation of terms and subterms, while responsible for some of the intricacies in the description, does not seem to have any advantage, from the implementation point of view, with respect to more standard representations of terms (such as those employed in the Warren's Abstract Machine and its variants [AK91]). As a consequence, standard notions from unification theory, such as *instance*, *anti-instance*, and *(least) common anti-instance* [LMM88], are never mentioned in [CLV93], while being implicitly present.

On the more technical side, Cortesi *et al.* assume explicitly that, during the analysis, no cyclic pattern (term) will be generated. This assumption, which is indeed standard, makes sense only if we pretend that the analyzed language does not omit the *occur-check* in the unification procedure. Unfortunately, it is well-known that many implemented CLP($\mathcal{H}_\mathcal{X}$) languages (in particular, almost all Prolog systems) do omit the *occur-check*. The real solution to this problem would be to consider the real concrete domain implemented by the vast majority of systems: some kind of *rational trees* [Col84], and not *finite trees*. This, however, is not guaranteed to preserve the existing analysis frameworks and domains: everything ought to be recasted and justified in terms of the "new" concrete domain.

In this chapter we present, for the first time, the rational construction of a generic domain for structural analysis of CLP($\mathcal{H}_\mathcal{X}$) languages: Pattern($\mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$), where the parameter $\mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$ is an abstract domain satisfying certain properties. The formalization of the structural domain is independent from specific implementation techniques: `Pat(ℜ)` (slightly extended and corrected) is a possible implementation of the domain. Reasoning at a higher level of abstraction we are able to appeal to familiar notions of unification theory. One advantage is that we can identify an important parameter (a common anti-instance function, missing in [CLV93]) that gives some control over the precision and computational cost of the resulting generic structural domain. As far as the *occur-check problem* is concerned, we have not been able, so far, to find a general solution that preserves the existing analysis framework. More research is required on this subject. Nonetheless, we will discuss some observations and partial solutions.

It must be stressed that the merit of `Pat(ℜ)` is to define a generic implementation that works on any domain $\Re$ providing a certain set of *elementary*, low-level operations. It is particularly easy to extend an existing domain in order to support the simple operations required. However, this simplicity has a high cost in terms of efficiency: the execution of many isolated small operations over the underlying domain is much more expensive than performing few macro-operations where global effects can be taken into account. The operations that the underlying domain must provide are thus more complicated in our approach. This is not a limitation, if one considers that in the actual implementation even more complex operations are used. For instance, all the *abstract bindings* arising from a bunch of unifications are executed in one shot, instead of one-at-a-time. See near the end of Section 6.7 for more on this subject.

4.2 Preliminaries

We assume that our pervasive set of variable symbols, *Vars*, contains (among others) two infinite, disjoint subsets: $\mathbf{z}$ and $\mathbf{z}'$. Since *Vars* is totally ordered, $\mathbf{z}$ and $\mathbf{z}'$ are as well:

$$\mathbf{z} \stackrel{\text{def}}{=} \bigl(Z_1, Z_2, Z_3, \ldots \tag{4.1}$$

$$\mathbf{z}' \stackrel{\text{def}}{=} \bigl(Z'_1, Z'_2, Z'_3, \ldots \tag{4.2}$$

For any syntactic object o (a term or a tuple of terms) we will denote by $\mathit{vseq}(o)$ the sequence of first occurrences of variables that are found in a depth-first, left-to-right traversal[2] of o. For instance,

$$\mathit{vseq}\Bigl(\bigl(f(g(X), Y), h(X)\bigr)\Bigr) = (X, Y).$$

In order to avoid the burden of talking “modulo renaming” we will make use of two strong normal forms for tuples of terms. Specifically, the set of

[2]Any other *fixed* ordering would be as good for our purposes.

n-tuples in $\mathbf{z}$*-form* is given by

$$\mathbf{T}^n_{\mathbf{z}} \stackrel{\text{def}}{=} \Bigl\{ \bar{t} \in \mathcal{T}^n_{\mathit{Vars}} \Bigm| \mathit{vseq}(\bar{t}\,) = \bigl(Z_1, Z_2, \ldots, Z_{|\mathit{vars}(\bar{t}\,)|}\bigr) \Bigr\}. \tag{4.3}$$

All the tuples in $\mathbf{z}$-form are contained in

$$\mathbf{T}^{\star}_{\mathbf{z}} \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}} \mathbf{T}^n_{\mathbf{z}}. \tag{4.4}$$

By replacing $\mathbf{z}$ with $\mathbf{z}'$ and Z_i with Z'_i in (4.3) and (4.4), we obtain similar definitions for $\mathbf{T}^n_{\mathbf{z}'}$ and $\mathbf{T}^{\star}_{\mathbf{z}'}$.

There is a useful device for toggling between $\mathbf{z}$- and $\mathbf{z}'$-forms. Let $\bar{t} \in \mathbf{T}^n_{\mathbf{z}} \cup \mathbf{T}^n_{\mathbf{z}'}$ and $\bigl|\mathit{vars}(\bar{t}\,)\bigr| = m$. Then

$$\bar{t}' \stackrel{\text{def}}{=} \begin{cases} \bar{t}[Z'_1/Z_1, \ldots, Z'_m/Z_m], & \text{if } \bar{t} \in \mathbf{T}^n_{\mathbf{z}}; \\ \bar{t}[Z_1/Z'_1, \ldots, Z_m/Z'_m], & \text{if } \bar{t} \in \mathbf{T}^n_{\mathbf{z}'}. \end{cases} \tag{4.5}$$

Notice that $\bar{t}'' \stackrel{\text{def}}{=} \bigl(\bar{t}'\bigr)' = \bar{t}$.

We will make use of a *normalization function* $\eta \colon \mathcal{T}^{\star}_{\mathit{Vars}} \to \mathbf{T}^{\star}_{\mathbf{z}}$ such that, for each $\bar{t} \in \mathcal{T}^{\star}_{\mathit{Vars}}$, the resulting tuple $\eta(\bar{t}\,) \in \mathbf{T}^{\star}_{\mathbf{z}}$ is a variant of $\bar{t}$.

Another renaming we will use is the following: for each $\bar{s} \in \mathcal{T}^{\star}_{\mathit{Vars}}$ and each other syntactic object o such that $FV(o) \subset \mathbf{z}$, we write $\varrho_{\bar{s}}(o)$ to denote

$$o[Z_{n+i_1}/Z_{i_1}, \ldots, Z_{n+i_m}/Z_{i_m}],$$

where $n = \bigl|\mathit{vars}(\bar{s})\bigr|$ and $\{Z_{i_1}, \ldots, Z_{i_m}\} = \mathit{vars}(o)$. This device will be useful for concatenating normalized term-tuples, still obtaining a normalized term-tuple. In fact, for each $\bar{s}_1, \bar{s}_2 \in \mathbf{T}^{\star}_{\mathbf{z}}$ we have

$$\bar{s}_1 :: \varrho_{\bar{s}_1}(\bar{s}_2) \in \mathbf{T}^{\star}_{\mathbf{z}}. \tag{4.6}$$

When $\bar{V} \in \mathit{Vars}^m$ and $\bar{t} \in \mathcal{T}^m_{\mathit{Vars}}$ we use $\bigl[\bar{t}/\bar{V}\bigr]$ as a shorthand for the substitution

$$\bigl[\pi_1(\bar{t}\,)/\pi_1(\bar{V}), \ldots, \pi_m(\bar{t}\,)/\pi_m(\bar{V})\bigr].$$

A couple of observations are useful for what follows. If $\bar{s} \in \mathbf{T}^{\star}_{\mathbf{z}}$ and $\bar{u} \in \mathbf{T}^{|\mathit{vars}(\bar{s})|}_{\mathbf{z}}$ then

$$\bar{s}'\bigl[\bar{u}/\,\mathit{vseq}(\bar{s}')\bigr] \in \mathbf{T}^{\star}_{\mathbf{z}}. \tag{4.7}$$

Moreover

$$\mathit{vseq}\Bigl(\bar{s}'\bigl[\bar{u}/\,\mathit{vseq}(\bar{s}')\bigr]\Bigr) = \mathit{vseq}(\bar{u}). \tag{4.8}$$

4.3 Factoring Out Structural Information

A quite general picture for the analysis of a CLP($\mathcal{H}_{\mathcal{X}}$) language is as follows. We want to describe a (possibly infinite) set of constraint stores over a tuple of *variables of interest* $V_1, \ldots, V_k$. These variables represent the arguments of some program predicate. Each constraint store σ can be represented (at some level of abstraction) by a formula of the kind

$$\exists_\Delta \mathrel{.} \bigl(\{V_1 = t_1, \ldots, V_k = t_k\} \land C\bigr), \tag{4.9}$$

such that

$$\{V_1 = t_1, \ldots, V_k = t_k\}, \quad \text{with } t_1, \ldots, t_k \in \mathcal{T}_{\mathit{Vars}}, \tag{4.10}$$

is a system of Herbrand equations in solved form, $C \in \mathcal{D}^\flat_{\mathcal{X}}$ is a constraint on the concrete domain of $\mathcal{X}$, and

$$\Delta \stackrel{\mathrm{def}}{=} \mathit{vars}(C) \cup \mathit{vars}(t_1) \cup \cdots \cup \mathit{vars}(t_k)$$

is such that $\Delta \cap \{V_1, \ldots, V_k\} = \varnothing$. Roughly speaking, the purpose of C is to limit the values that the (quantified) variables occurring in $t_1, \ldots, t_k$ can take.

It must be stressed that the concrete semantics we are outlining, while providing an adequate basis for many abstract interpretations, is "too abstract" when the properties of interest concern the internal workings of the Herbrand constraint solver. An example is *structure-sharing analysis* [WW96], whose aim is to determine those *structure cells* (elementary objects used to represent terms) that are possibly shared by more than one term representation. For example, the system

$$\{V_1 = f(a), V_2 = f(a)\} \tag{4.11}$$

does not say anything about structure sharing: we might have a shared $f/1$ cell, or two distinct ones. In the latter case we might have a shared $a/0$ cell, or two distinct ones. Thus, there are a total of 3 cases that cannot be distinguished by looking at (4.11), the obvious consequence being that we cannot base structure-sharing analysis on a concrete domain made up of representations like (4.9). Similar considerations apply to the choice of $\mathcal{D}^\flat_{\mathcal{X}}$.

Let us assume that we are dealing with analyses where the internal representation of terms is irrelevant. Once variables $V_1, \ldots, V_k$ have been fixed, the Herbrand part of the constraint store (4.9), the system of equations (4.10), can be represented as a k-tuple of terms. Since we want to characterize any set of constraint stores, our concrete domain is

$$\mathcal{D}^\flat_{\mathcal{H}_{\mathcal{X}}} \stackrel{\mathrm{def}}{=} \bigcup_{n \in \mathbb{N}} \wp\bigl(\mathbf{T}^n_{\mathbf{z}} \times \mathcal{D}^\flat_{\mathcal{X}}\bigr). \tag{4.12}$$

An abstract interpretation of $\mathcal{D}^{\flat}_{\mathcal{H}\mathcal{X}}$ can be specified by choosing an abstract domain $\mathcal{D}^{\sharp}_{\mathcal{H}\mathcal{X}}$ and a suitable abstraction function

$$\alpha \colon \mathcal{D}^{\flat}_{\mathcal{H}\mathcal{X}} \to \mathcal{D}^{\sharp}_{\mathcal{H}\mathcal{X}}. \tag{4.13}$$

If $\mathcal{D}^{\sharp}_{\mathcal{H}\mathcal{X}}$ is able to encode enough structural (Herbrand) information from $\mathcal{D}^{\flat}_{\mathcal{H}\mathcal{X}}$ so as to achieve the desired precision, fine. If this is not the case, it is possible to improve the situation by keeping some structural information explicit.

One way of doing that is to perform a change of representation for $\mathcal{D}^{\flat}_{\mathcal{H}\mathcal{X}}$, which is the basis for further abstractions. The new representation is obtained by factoring out some common Herbrand information. The meaning of 'some' is encoded by a function.

$$\begin{tikzcd}
\wp\bigl(\mathbf{T}^{\star}_{\mathbf{z}} \times \mathcal{D}^{\flat}_{\mathcal{X}}\bigr) \arrow[r, "\alpha"] \arrow[d, "\Phi_\phi"'] & \mathcal{D}^{\sharp}_{\mathcal{H}_\mathcal{X}} \\
\mathbf{T}^{\star}_{\mathbf{z}} \times \wp\bigl(\mathbf{T}^{\star}_{\mathbf{z}} \times \mathcal{D}^{\flat}_{\mathcal{X}}\bigr) \arrow[r, "(\mathrm{id},\alpha)"'] & \mathbf{T}^{\star}_{\mathbf{z}} \times \mathcal{D}^{\sharp}_{\mathcal{H}_\mathcal{X}} \arrow[u, dotted, "\alpha'"']
\end{tikzcd}$$

Figure 4.4: Upgrading a domain with structural information.

Definition 60 (Common anti-instance function.) *Any function*

$$\phi\colon \bigcup_{n\in\mathbb{N}} \wp(\mathbf{T}^{n}_{\mathbf{z}}) \to \mathbf{T}^{\star}_{\mathbf{z}'}$$

is called a common anti-instance function *if it satisfies, for each $n \in \mathbb{N}$ and each $\hat{T} \in \wp(\mathbf{T}^{n}_{\mathbf{z}})$:*

1. *$\phi(\hat{T}) \in \mathbf{T}^{n}_{\mathbf{z}'}$;*
2. *if $\phi(\hat{T}) = \bar{r}$ and $\bigl|vars(\bar{r})\bigr| = m$ with $m \geq 0$, then*

$$\forall \bar{t} \in \hat{T} : \exists \bar{u} \in \mathbf{T}^{m}_{\mathbf{z}} \,.\, \bar{r}\bigl[\bar{u}/\,vseq(\bar{r})\bigr] = \bar{t}.$$

In words, $\phi(\hat{T})$ is an *anti-instance*, in $\mathbf{z}'$-form, of each $\bar{t} \in \hat{T}$.

Any choice of ϕ induces a function

$$\Phi_\phi\colon \mathcal{D}^{\flat}_{\mathcal{H}_\mathcal{X}} \to \mathbf{T}^{\star}_{\mathbf{z}} \times \wp\bigl(\mathbf{T}^{\star}_{\mathbf{z}} \times \mathcal{D}^{\flat}_{\mathcal{X}}\bigr), \tag{4.14}$$

which is given, for each $\hat{E}^{\flat} \in \mathcal{D}^{\flat}_{\mathcal{H}_\mathcal{X}}$ by

$$\Phi_\phi(\hat{E}^{\flat}) \stackrel{\mathrm{def}}{=} \left(\bar{s}', \left\{ (\bar{u}, D^{\flat}) \;\middle|\; (\bar{t}, D^{\flat}) \in \hat{E}^{\flat},\quad \bar{s}\bigl[\bar{u}/vseq(\bar{s})\bigr] = \bar{t} \right\}\right), \tag{4.15}$$

where $\bar{s} \stackrel{\mathrm{def}}{=} \phi\bigl(\pi_1(\hat{E}^{\flat})\bigr)$. The corestriction to the image of Φ_ϕ, that is $\Phi_\phi\colon \mathcal{D}^{\flat}_{\mathcal{H}_\mathcal{X}} \to \Phi_\phi\bigl(\mathcal{D}^{\flat}_{\mathcal{H}_\mathcal{X}}\bigr)$, is an isomorphism, the inverse being

$$\Phi_\phi^{-1}\bigl((\bar{s}, \hat{F}^{\flat})\bigr) \stackrel{\mathrm{def}}{=} \left\{ \left(\bar{s}'\bigl[\bar{u}/vseq(\bar{s}')\bigr], D^{\flat}\right) \;\middle|\; (\bar{u}, D^{\flat}) \in \hat{F}^{\flat} \right\}. \tag{4.16}$$

So far, we have just chosen a different representation for $\mathcal{D}^{\flat}_{\mathcal{H}_\mathcal{X}}$, that is $\Phi_\phi\bigl(\mathcal{D}^{\flat}_{\mathcal{H}_\mathcal{X}}\bigr)$. The idea behind structural information analysis is to leave the first component (also called the *pattern component*) of the new representation untouched, while abstracting the second component by means of α, as illustrated in Figure 4.4. The dotted arrow indicates a *residual abstraction function* α'. As we will see in Section 4.5.4, such a function is implicitly required in order to define an important operation over the new abstract domain $\mathbf{T}^{\star}_{\mathbf{Z}} \times \mathcal{D}^{\sharp}_{\mathcal{H}_{\mathcal{X}}}$. Notice that α' may or may not make the diagram of Figure 4.4 commute (although often α' turns out to have this property).

This approach has several advantages. First of all, factoring out common structural information improves the analysis precision, since part of the approximated k-tuples of terms is recorded, in *concrete form*, into the first component of $\mathbf{T}^{\star}_{\mathbf{Z}} \times \mathcal{D}^{\sharp}_{\mathcal{H}_{\mathcal{X}}}$. Secondly, the above construction is adjustable by means of the parameter ϕ. The most precise choice consists in taking ϕ to be a *least common anti-instance* function. For example, the set

$$\hat{S} \stackrel{\mathrm{def}}{=} \Bigl\{ \bigl\langle \bigl(s(0), c(Z_1, nil)\bigr), C_1 \bigr\rangle, \bigl\langle \bigl(s(s(0)), c(Z_1, c(Z_2, nil))\bigr), C_2 \bigr\rangle \Bigr\},$$

where $C_1, C_2 \in \mathcal{D}^{\flat}_{\mathcal{X}}$, is mapped by the Φ_{lca} function onto

$$\Phi_{\mathrm{lca}}(\hat{S}) = \Biggl(\bigl(s(Z_1), c(Z_2, Z_3)\bigr), \\ \Bigl\{ \bigl\langle (0, Z_1, nil), C_1 \bigr\rangle, \bigl\langle \bigl(s(0), Z_1, c(Z_2, nil)\bigr), C_2 \bigr\rangle \Bigr\} \Biggr).$$

At the other side of the spectrum is the possibility of choosing ϕ so that it returns a k-tuples of distinct, new variables for each set of k-tuples of terms. This correspond to a framework where structural information is just discarded. With this choice, $\hat{S}$ would be mapped onto

$$\bigl((Z_1, Z_2), \hat{S}\bigr).$$

In-between these two extremes there are a number of possibilities that help managing the complexity/precision tradeoff. The k-tuples returned by ϕ can be limited in *depth* [ST84, MS88], for instance. More useful is to limit them in *width*, that is, limiting the number of symbols' occurrences. This flexibility allows to design the analysis' domains without caring about structural information: the problem is always to approximate elements of $\wp\bigl(\mathbf{T}^{a}_{\mathbf{Z}} \times \mathcal{D}^{\flat}_{\mathcal{X}}\bigr)$. Whether a is fixed by the arity of a predicate or a is the number of variables occurring in some pattern does not really matter.

4.4 Parametric Structural Analysis

In order to specify the abstract domain for the analysis, we need some assumptions on the concrete $\mathcal{X}$ domain $\mathcal{D}^{\flat}_{\mathcal{X}}$, which represents the $\mathcal{X}$-part of *consistent* constraint stores. One can think about $\mathcal{D}^{\flat}_{\mathcal{X}}$ as made up of first-order sentences: the techniques described in Chapter 3 come in handy for this purpose. In this view, the operator $\otimes\colon \mathcal{D}^{\flat}_{\mathcal{X}} \times \mathcal{D}^{\flat}_{\mathcal{X}} \to \mathcal{D}^{\flat}_{\mathcal{X}}$ corresponds to logical conjunction. Moreover, we assume that it makes sense to talk about the *free variables* of $D^{\flat} \in \mathcal{D}^{\flat}_{\mathcal{X}}$, denoted by $FV(D^{\flat})$. Let $\bar{s}, \bar{t}, \bar{u} \in \mathbf{T}^{\star}_{\mathbf{Z}}$

and $D^\flat, E^\flat \in \mathcal{D}^\flat_\mathcal{X}$ such that $FV(D^\flat) \subseteq \mathit{vars}(\bar{t})$. When we write $(\bar{u}, E^\flat) = \varrho_{\bar{s}}\bigl((\bar{t}, D^\flat)\bigr)$, we mean that $\bar{u} = \varrho_{\bar{s}}(\bar{t})$ and that $E^\flat$ has been obtained from $D^\flat$ by applying the same renaming applied to $\bar{t}$ in order to obtain $\bar{u}$.

Another natural thing to do is projecting a satisfiable store: thus $\exists_{\bar{V}} D^\flat$, where $\bar{V}$ is a tuple (or set) of variables, is assumed to be as defined.

The last thing we need is the ability of adding an equality constraint to a constraint store. Thus $D^\flat[t_1 = t_2]$ is the store obtained from $D^\flat$ by injecting the equation $t_1 = t_2$, *provided that the resulting store is consistent*, otherwise the operation is undefined. Notice that we assume $\mathcal{D}^\flat_\mathcal{X}$ and its operations encode both the proper $\mathcal{X}$*-solver* and the so called *interface* between the *Herbrand engine* and the $\mathcal{X}$-solver [JM94]. In particular, the interface is responsible for *type-checking* of the equations it receives. For example, in CLP($\mathcal{R}$) the interface is responsible for the fact that $X = a$ cannot be consistently added to a constraint store where X was previously classified as numeric.

We now turn our attention to the abstract domain that is the parameter of the generic structural domain. We will denote it simply by $\mathcal{D}^\sharp$, instead of $\mathcal{D}^\sharp_{\mathcal{H}_\mathcal{X}}$. Thus, assuming that $\mathcal{X}$ has been fixed, $\mathcal{D}^\flat_\mathcal{X}$ is indicated just by $\mathcal{D}^\flat$.

Since the aim here is maximum generality, we refer to a very weak abstract interpretation framework.

Definition 61 (Abstract domain.) *An* abstract domain for $\mathcal{H}_\mathcal{X}$ *is any set $\mathcal{D}^\sharp$ equipped with a preorder relation $\sqsubseteq$, an order preserving function*

$$\gamma\colon \mathcal{D}^\sharp \to \mathcal{D}^\flat,$$

an upper-bound operator

$$\oplus\colon \mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \mathcal{D}^\sharp,$$

and a least element $\bot^\sharp$ such that $\gamma(\bot^\sharp) = \varnothing$.

Thus, for each $D^\sharp_1, D^\sharp_2 \in \mathcal{D}^\sharp$, we have both

$$D^\sharp_1 \sqsubseteq D^\sharp_2 \quad \implies \quad \gamma(D^\sharp_1) \subseteq \gamma(D^\sharp_2), \tag{4.17}$$

and

$$\gamma(D^\sharp_1 \oplus D^\sharp_2) \supseteq \gamma(D^\sharp_1) \cup \gamma(D^\sharp_2). \tag{4.18}$$

The structural information construction upgrades any given abstract domain $\mathcal{D}^\sharp$ as follows.

Definition 62 (The Pattern$(\cdot)$ **construction.)** *Let $\mathcal{D}^\sharp$ be an abstract domain. Then*

$$\mathrm{Pattern}(\mathcal{D}^\sharp) \stackrel{\mathrm{def}}{=} \Bigl\{\, (\bar{s}, D^\sharp) \in \mathbf{T}^\star_\mathbf{z} \times \mathcal{D}^\sharp \Bigm| \gamma(D^\sharp) \subseteq \mathbf{T}_\mathbf{z}^{|\mathit{vars}(\bar{s})|} \times \mathcal{D}^\flat \,\Bigr\}.$$

The meaning of each element $(\bar{s}, D) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ *is given by* $\gamma\colon \mathrm{Pattern}(\mathcal{D}^\sharp) \to \wp(\mathbf{T}^\star_{\mathbf{z}} \times \mathcal{D}^\flat)$*:*

$$\gamma\bigl((\bar{s}, D^\sharp)\bigr) \stackrel{\mathrm{def}}{=} \Bigl\{ \Bigl(\bar{s}'\bigl[\bar{u}/\, \mathit{vseq}(\bar{s}')\bigr],\, D^\flat\Bigr) \Bigm| (\bar{u}, D^\flat) \in \gamma(D^\sharp) \Bigr\}.$$

The following result is needed for what follows. It is a straight consequence of Definition 62.

Proposition 63 *Whenever* $(\bar{s}, D^\sharp_1), (\bar{s}, D^\sharp_2) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$

$$\gamma(D^\sharp_1) \subseteq \gamma(D^\sharp_2) \quad\Longrightarrow\quad \gamma\bigl((\bar{s}, D^\sharp_1)\bigr) \subseteq \gamma\bigl((\bar{s}, D^\sharp_2)\bigr).$$

4.5 Operations for the Analysis

In this section we define the operations over $\mathrm{Pattern}(\mathcal{D}^\sharp)$ that are needed for the analysis in a bottom-up framework. In order of appearance into the analysis process:

- we need an operation that takes two descriptions and, roughly speaking, juxtaposes them. This operation, which we call *meet with renaming apart*, is needed when "solving" a clause body with respect to the current interpretation.

- Unification, that realizes "parameter passing". The descriptions that were simply juxtaposed are thus made to communicate with each other.

- When all the goals in a clause body have been solved, projection is used to restrict the abstract description to the tuple of arguments of the clause's head.

- The operation of *remapping* is used to adapt a description to a different, less precise, pattern component. It is used in order to realize various *join* and *widening* operations.

- The *join* operation is parameterized with respect to a common anti-instance function. It is used to merge descriptions arising from the different sets of computation paths explored during the analysis.

- The *comparison* operation is employed by the analyzer in order to check whether a local (to a program clause or predicate) fixpoint has been reached.

The above operations over $\mathrm{Pattern}(\mathcal{D}^\sharp)$ induce the need for other operations on the underlying domain $\mathcal{D}^\sharp$. The latter are specified in the next sections so that the correctness of the analysis can be ensured.

4.5.1 Meet with Renaming Apart

This operation is very simple.

Definition 64 (The rmeet **operation.)** *Let* $\rhd \colon \mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ *be such that, for each* $D_1^\sharp, D_2^\sharp \in \mathcal{D}^\sharp$*,*

$$\gamma(D_1^\sharp \rhd D_2^\sharp) = \left\{ (\bar{r}_1 :: \bar{w}_2, D_1^\flat \otimes E_2^\flat) \;\middle|\; \begin{array}{l} (\bar{r}_1, D_1^\flat) \in \gamma(D_1^\sharp) \\ (\bar{r}_2, D_2^\flat) \in \gamma(D_2^\sharp) \\ (\bar{w}_2, E_2^\flat) = \varrho_{\bar{r}_1}\bigl((\bar{r}_2, D_2^\flat)\bigr) \end{array} \right\}.$$

Then, for each $(\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$*.*

$$\mathrm{rmeet}\bigl((\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp)\bigr) \stackrel{\mathrm{def}}{=} \bigl(\bar{s}_1 :: \varrho_{\bar{s}_1}(\bar{s}_2),\ D_1^\sharp \rhd D_2^\sharp\bigr).$$

The following result is a direct consequence of the definition: there is no precision loss in rmeet.

Theorem 65 *For each* $(\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$*.*

$$\gamma\Bigl(\mathrm{rmeet}\bigl((\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp)\bigr)\Bigr) = \left\{ (\bar{t}_1 :: \bar{u}_2, D_1^\flat \otimes E_2^\flat) \;\middle|\; \begin{array}{l} (\bar{t}_1, D_1^\flat) \in \gamma\bigl((\bar{s}_1, D_1^\sharp)\bigr) \\ (\bar{t}_2, D_2^\flat) \in \gamma\bigl((\bar{s}_2, D_2^\sharp)\bigr) \\ (\bar{u}_2, E_2^\flat) = \varrho_{\bar{t}_1}\bigl((\bar{t}_2, D_2^\flat)\bigr) \end{array} \right\}.$$

4.5.2 Unification with Occur-Check

In this section we assume that the execution mechanism of the language being analyzed performs unifications without omitting the *occur-check*. With this hypothesis (which, unfortunately, is seldom verified) we can easily complete the unification algorithm given in [CLV93]. When the *occur-check* fails in the abstract unification we know that the computation path being analyzed can be safely pruned, because the concrete unification would have failed at that point, if not before. Notice that, for the purpose of the present discussion, the *occur-check* need not be implemented explicitly, that is by making the unification *fail* in the logic programming sense. Since our data-flow analyses provide information of the kind

if control gets to this point, *then* that will hold there,

a more drastic handling of the *occur-check* is acceptable. If we are guaranteed that the concrete system enters either an error state or an infinite loop whenever a cyclic binding is attempted, then the abstract unification

```
procedure unify($\bar{s}, D^\sharp, t, u$)
1: if $t \neq u$ then
2:   if $t = f(t_1, \ldots, t_n)$ and $u = f(u_1, \ldots, u_n)$ then
3:     for all $i = 1, \ldots, n$ do
4:       unify($\bar{s}, D^\sharp, t_i, u_i$)
5:   else if $t = Z_h$ then
6:     if $Z_h$ does not occur in $u$ then
7:       $D^\sharp :=$ bind($\bar{s}, D^\sharp, u, Z_h$) {invokes underlying domain}
8:       $Z_h := u$ {instantiates all the occurrences of $Z_h$}
9:       $\bar{s} := \eta(\bar{s})$ {normalization}
10:    else
11:      $D^\sharp := \bot^\sharp$
12:  else if $u = Z_k$ then
13:    unify($\bar{s}, D^\sharp, u, t$)
14:  else
15:    $D^\sharp := \bot^\sharp$
```

Algorithm 1: Unification for the parametric structural domain.

procedure presented in this section can safely be used. We will discuss later what can be done for those systems where the *occur-check* is, by any means, omitted.

We start with a description $(\bar{s}, D^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ and two terms to be unified, t and u, such that

$$\mathit{vars}\bigl((t, u)\bigr) \subseteq \mathit{vars}(\bar{s}).$$

We then apply the procedure unify, given as Algorithm 1, to $\bar{s}$, $D^\sharp$, t and u.

In the macro-operation $\mathrm{bind}(\bar{s}, D^\sharp, u, Z_h)$, $\bar{s}$ is passed only in order to maintain the connection between the variables in (u, Z_h) and the description $D^\sharp$. We assume, without loss of generality, that whenever $\mathrm{bind}(\bar{s}, D^\sharp, u, Z_h)$ is invoked we have

$$\bigl|\mathit{vars}(\bar{s})\bigr| = m, \quad \text{with } m \geq 0.$$

The result of the operation will be a description $D_1^\sharp$ such that $\gamma(D_1^\sharp) \subseteq \mathbf{T}_{\mathbf{Z}}^{m-1} \times \mathcal{D}^\flat$. This is because, after the binding, Z_h will not be referenced anymore. What remains to be described is the operation of reflecting the binding of Z_h to u into $D^\sharp$ so to obtain $D_1^\sharp$. We will denote this operation by $D^\sharp[u/Z_h]$, and present its variants (depending on whether u is a constant or a number or a variable or a compound term) in the next sections.

Binding to a Constant or a Number

The result of $D^\sharp[k/Z_h]$, where k is a symbolic constant or a number and $h \in \{1, \dots, m\}$ is any $D_1^\sharp \in \mathcal{D}^\sharp$ such that

$$\gamma(D_1^\sharp) \supseteq \Bigl\{ \bigl((t_1, \dots, t_{h-1}, t_{h+1}, \dots, t_m), D^\flat[t_h = k]\bigr) \Bigm| \bigl((t_1, \dots, t_m), D^\flat\bigr) \in \gamma(D^\sharp) \Bigr\}.$$

Notice that

$$vseq\bigl(\bar{s}[k/Z_h]\bigr) = (Z_1, \dots, Z_{h-1}, Z_{h+1}, \dots, Z_m).$$

Similar comments apply also to what follows.

Binding to an Alias

Here we must specify an admissible result, $D_1^\sharp$, for the operation $D^\sharp[Z_i/Z_h]$ with $i, h \in \{1, \dots, m\}$ and $i \neq h$. In order to reduce the complexity of the definition we need some special notation for sequences.

Let U be a set. We define the operation $\cdot \setminus \cdot \colon U^\star \times \wp_\mathrm{f}(U) \to U^\star$ as follows. For each sequence $L \in U^\star$ and each set $S \in \wp_\mathrm{f}(U)$, the sequence $L \setminus U$ is obtained by removing from L all the elements that appear in U. Formally,

$$\varepsilon \setminus S \stackrel{\mathrm{def}}{=} \varepsilon;$$

$$\bigl((x) :: L\bigr) \setminus S \stackrel{\mathrm{def}}{=} \begin{cases} L \setminus S, & \text{if } x \in S; \\ (x) :: (L \setminus S), & \text{if } x \notin S. \end{cases}$$

Let us define $\tau(k) \colon \{1, \dots, m-1\} \to \{1, \dots, m\}$ as

$$\tau(k) \stackrel{\mathrm{def}}{=} \pi_k\Bigl((1, \dots, h-1) :: \bigl((i) \setminus \{1, \dots, h-1\}\bigr) :: \bigl((h+1, \dots, m) \setminus \{i\}\bigr)\Bigr).$$

The transformation τ is such that, for each $k = 1, \dots, m-1$,

$$\pi_k\Bigl(vseq\bigl(\bar{s}[Z_i/Z_h]\bigr)\Bigr) = Z_{\tau(k)}.$$

Now, $D_1^\sharp$ must satisfy

$$\gamma(D_1^\sharp) \supseteq \Bigl\{ \Bigl(\bigl(\pi_{\tau(1)}(\bar{t}), \dots, \pi_{\tau(m-1)}(\bar{t})\bigr), D^\flat\bigl[\pi_h(\bar{t}) = \pi_i(\bar{t})\bigr]\Bigr) \Bigm| (\bar{t}, D^\flat) \in \gamma(D^\sharp) \Bigr\}.$$

Binding to a Compound

Specifying the result of the operation $D^\sharp[u/Z_h]$ is only slightly more complicated. Suppose that

$$vseq(u) = \bigl(Z_{j_1}, \dots, Z_{j_l}\bigr),$$

so that $Z_h \notin \bigl\{Z_{j_1}, \dots, Z_{j_l}\bigr\}$ and the transformation τ is given by

$$\begin{aligned}\tau(k) \overset{\text{def}}{=} \pi_k\Bigl((1, \dots, h-1)& \\ &:: \bigl((j_1, \dots, j_l) \setminus \{1, \dots, h-1\}\bigr) \\ &\qquad :: \bigl((h+1, \dots, m) \setminus \{j_1, \dots, j_l\}\bigr)\Bigr).\end{aligned} \tag{4.19}$$

Then $D^\sharp[u/Z_h]$ is allowed to return any $D_1^\sharp$ such that

$$\gamma(D_1^\sharp) \supseteq \left\{ \Bigl(\bigl(\pi_{\tau(k)}(\bar{t}\,)\bigr)_{k=1}^{m-1}, D_1^\flat\Bigr) \;\middle|\; \begin{array}{l} (\bar{t}, D^\flat) \in \gamma(D^\sharp) \\ \theta = \bigl[u_{j_1}/Z_{j_1}, \dots, u_{j_l}/Z_{j_l}\bigr] \\ \pi_h(\bar{t}\,) = u\theta \\ D_1^\flat = D^\flat\bigl[\pi_{j_1}(\bar{t}\,) = u_{j_1}, \\ \qquad\qquad \dots, \pi_{j_l}(\bar{t}\,) = u_{j_l}\bigr] \end{array} \right\}.$$

The proof of the overall correctness of Algorithm 1 is rather tedious and thus omitted. As observed in [CLV93] it can be obtained by systematic generalization of the proof given in Musumbu's PhD thesis [Mus90].

4.5.3 Projection

This operation consists simply in dropping a suffix of the pattern component, with the consequent projection on the underlying domain.

Definition 66 (The project **operation.)** *Let*

$$\bigl\{\overline{\exists}_i \colon \mathcal{D}^\sharp \to \mathcal{D}^\sharp\bigr\}_{i\in\mathbb{N}}$$

be a family of operations such that, for each $m \in \mathbb{N}$, each $D^\sharp \in \mathcal{D}^\sharp$ with $\gamma(D^\sharp) \subseteq \mathbf{T}_{\mathbf{z}}^m \times \mathcal{D}^\flat$, and for each $j < m$,

$$\gamma\bigl(\overline{\exists}_j\, D^\sharp\bigr) \supseteq \left\{ (\bar{r}, E^\flat) \;\middle|\; \begin{array}{l} (\bar{u}, D^\flat) \in \gamma(D^\sharp) \\ \bar{r} = \bigl(\pi_1(\bar{u}), \dots, \pi_j(\bar{u})\bigr) \\ \Delta = \overline{vars(\bar{r})} \\ E^\flat = \exists_\Delta D^\flat \end{array} \right\}.$$

Then, for each $(\bar{s}, D^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ such that $\bar{s} \in \mathbf{T}_{\mathbf{z}}^n$ and for each $k < n$,

$$\mathrm{project}_k\bigl((\bar{s}, D^\sharp)\bigr) \overset{\text{def}}{=} \Bigl(\underbrace{\bigl(\pi_1(\bar{s}), \dots, \pi_k(\bar{s})\bigr)}_{\bar{t}}, \overline{\exists}_j\, D^\sharp\Bigr),$$

where $j \stackrel{\text{def}}{=} \bigl|vars(\bar{t}\,)\bigr|$.

It is easy to show that project is indeed correct with respect to the obvious concrete operation.

4.5.4 Remapping

Consider a description $(\bar{s}, D^\sharp_{\bar{s}}) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$ and a pattern $\bar{r} \in \mathbf{T}^\star_{\mathbf{z}'}$ such that $\bar{r}$ is an anti-instance of $\bar{s}$. We want to obtain $D^\sharp_{\bar{r}} \in \mathcal{D}^\sharp$ such that

$$\gamma\bigl((\bar{r}', D^\sharp_{\bar{r}})\bigr) \supseteq \gamma\bigl((\bar{s}, D^\sharp_{\bar{s}})\bigr). \tag{4.20}$$

This is what we call *remapping* $(\bar{s}, D^\sharp_{\bar{s}})$ *to* $\bar{r}$.

Definition 67 (The remap **operation.)** *Let* $(\bar{s}, D^\sharp_{\bar{s}})$ *be a description with* $\bar{s} \in \mathbf{T}^k_{\mathbf{z}}$ *and let* $\bar{r} \in \mathbf{T}^k_{\mathbf{z}'}$ *be an anti-instance of* $\bar{s}$*. Assume* $\bigl|vars(\bar{r})\bigr| = m$ *and let* $\bar{u} \in \mathbf{T}^m_{\mathbf{z}}$ *be the unique tuple such that*

$$\bar{r}\bigl[\bar{u}/\,vseq(\bar{r})\bigr] = \bar{s}. \tag{4.21}$$

Then the operation $\mathrm{remap}(\bar{s}, D^\sharp_{\bar{s}}, \bar{r})$ *yields* $D^\sharp_{\bar{r}}$ *such that*

$$\gamma(D^\sharp_{\bar{r}}) \supseteq \Bigl\{ \Bigl(\bar{u}'\bigl[\bar{t}/\,vseq(\bar{u}')\bigr],\, D^\flat\Bigr) \Bigm| (\bar{t}, D^\flat) \in \gamma(D^\sharp_{\bar{s}}) \Bigr\}. \tag{4.22}$$

Observe that the remap function is closely related to the residual abstraction function α' of Figure 4.4. We now prove that the specification of remap meets our original requirement.

Theorem 68 *Let* $(\bar{s}, D^\sharp_{\bar{s}})$ *be a description with* $\bar{s} \in \mathbf{T}^k_{\mathbf{z}}$*. Let also* $\bar{r} \in \mathbf{T}^k_{\mathbf{z}'}$ *be an anti-instance of* $\bar{s}$*. If*

$$D^\sharp_{\bar{r}} = \mathrm{remap}(\bar{s}, D^\sharp_{\bar{s}}, \bar{r})$$

then

$$\gamma\bigl((\bar{r}', D^\sharp_{\bar{r}})\bigr) \supseteq \gamma\bigl((\bar{s}, D^\sharp_{\bar{s}})\bigr).$$

Proof Let us assume the hypotheses of Definition 67. By Definition 62, $\gamma\bigl((\bar{r}, D^\sharp_{\bar{r}})\bigr)$ is given by

$$\Bigl\{ \Bigl(\bar{r}\bigl[\bar{w}/\,vseq(\bar{r})\bigr],\, D^\flat\Bigr) \Bigm| (\bar{w}, D^\flat) \in \gamma(D^\sharp_{\bar{r}}) \Bigr\}. \tag{4.23}$$

Let us define

$$A \stackrel{\text{def}}{=} \Bigl\{ \Bigl(\bar{u}'\bigl[\bar{t}/\,vseq(\bar{u}')\bigr],\, D^\flat\Bigr) \Bigm| (\bar{t}, D^\flat) \in \gamma(D^\sharp_{\bar{s}}) \Bigr\}.$$

By (4.22) we have $\gamma(D^\sharp_{\bar{r}}) \supseteq A$. Thus, by Proposition 63, the set (4.23) contains

$$\Bigl\{ \Bigl(\bar{r}\bigl[\bar{w}/ \mathit{vseq}(\bar{r})\bigr],\, D^\flat\Bigr) \Bigm| (\bar{w}, D^\flat) \in A \Bigr\}.$$

This can be rewritten as

$$\left\{ \Bigl(\bar{r}\bigl[\bar{u}/ \mathit{vseq}(\bar{r})\bigr]\Bigr)'\bigl[\bar{t}/ \mathit{vseq}(\bar{u}')\bigr], D^\flat\Bigr) \;\middle|\; (\bar{t}, D^\flat) \in \gamma(D^\sharp_{\bar{s}}) \right\},$$

which, by (4.21), is equivalent to

$$\left\{ \bar{s}'\bigl[\bar{t}/ \mathit{vseq}(\bar{s}')\bigr], D^\flat\bigr) \;\middle|\; (\bar{t}, D^\flat) \in \gamma(D^\sharp_{\bar{s}}) \right\},$$

since $\mathit{vseq}(\bar{u}) = \mathit{vseq}(\bar{s})$. What we have just written is the definition of $\gamma\bigl((\bar{s}, D^\sharp_{\bar{s}})\bigr)$. $\square$

4.5.5 Join and Widenings

The operation of merging two descriptions turns out to be an easy one, once remapping has been defined. Let $(\bar{s}_1, D^\sharp_1)$ and $(\bar{s}_2, D^\sharp_2)$ be two descriptions with $\bar{s}_1, \bar{s}_2 \in \mathbf{T}^k_{\mathbf{z}}$. The resulting description is $\bigl(\bar{r}', E^\sharp_1 \oplus E^\sharp_2\bigr)$, where $\bar{r} \in \mathbf{T}^k_{\mathbf{z}'}$ is an anti-instance of both $\bar{s}_1$ and $\bar{s}_2$ and

$$E^\sharp_i = \mathrm{remap}(\bar{s}_i, D^\sharp_i, \bar{r}), \qquad \text{for } i = 1, 2.$$

We note again that $\bar{r}$ might be the least common anti-instance of $\bar{s}_1$ and $\bar{s}_2$, or it can be a further approximation of $\mathrm{lca}\{\bar{s}_1, \bar{s}_2\}$: this is one of the degrees of freedom of the framework.

Definition 69 (The join_ϕ operations.) *Let ϕ be any common anti-instance function. The operation (partial function)*

$$\mathrm{join}_\phi \colon \wp_{\mathrm{f}}\bigl(\mathrm{Pattern}(\mathcal{D}^\sharp)\bigr) \rightarrowtail \mathrm{Pattern}(\mathcal{D}^\sharp)$$

is defined as follows. For each $k \in \mathbb{N}$ and each finite family

$$F \stackrel{\mathrm{def}}{=} \bigl\{(\bar{s}_i, D^\sharp_i)\bigr\}_{i \in I}$$

of elements of $\mathrm{Pattern}(\mathcal{D}^\sharp)$ *such that $\bar{s}_i \in \mathbf{T}^k_{\mathbf{z}}$ for each $i \in I$,*

$$\mathrm{join}_\phi(F) \stackrel{\mathrm{def}}{=} \left(\bar{r}', \bigoplus_{i \in I} \mathrm{remap}(\bar{s}_i, D^\sharp_i, \bar{r})\right),$$

where

$$\bar{r} \stackrel{\mathrm{def}}{=} \phi\bigl(\{\bar{s}_i\}_{i \in I}\bigr).$$

Theorem 70 *Let F be as in* Definition 69. *For each common anti-instance function ϕ and each $(\bar{s}_j, D_j^\sharp) \in F$,*

$$\gamma\bigl(\mathrm{join}_\phi(F)\bigr) \supseteq \gamma\bigl((\bar{s}_j, D_j^\sharp)\bigr).$$

Proof Let $j \in I$ and $\bar{r} \stackrel{\mathrm{def}}{=} \phi\bigl(\{\bar{s}_i\}_{i\in I}\bigr)$. Then $\bar{r}$ is an anti-instance of $\bar{s}_j$. Thus

$$\begin{aligned}
\gamma\left(\left(\bar{r}, \bigoplus_{i\in I} \mathrm{remap}(\bar{s}_i, D_i^\sharp, \bar{r})\right)\right) &\supseteq \gamma\bigl((\bar{r}, \mathrm{remap}(\bar{s}_j, D_j^\sharp, \bar{r})\bigr) \\
&\quad \text{[by Def. 61 and Prop. 63]} \\
&\supseteq \gamma\bigl((\bar{s}_j, D_j^\sharp)\bigr) \\
&\quad \text{[by Thm. 68]}
\end{aligned}$$

□

As far as widening operators are concerned, there are several possibilities. First of all, we might want to distinguish between widening in the pattern component and widening on the underlying domain. The former can be defined as any join operation join_ϕ with ϕ different from lca. The latter consists in propagating the widening to the underlying domain. For instance, the following widening operator is the default one applied by CHINA:

$$\mathrm{widen}\bigl((\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp)\bigr) \stackrel{\mathrm{def}}{=} \begin{cases} (\bar{s}_2, D_2^\sharp), & \text{if } \bar{s}_1 \neq \bar{s}_2; \\ (\bar{s}_2, D_1^\sharp \nabla D_2^\sharp), & \text{if } \bar{s}_1 = \bar{s}_2. \end{cases} \tag{4.24}$$

This operator refrains from widening unless the pattern component is stabilized (see the next section to see why it works).

More drastic widenings can be defined. For example, we can exploit the fact that widenings need to be evaluated over $(\bar{s}_1, D_1^\sharp)$ and $(\bar{s}_2, D_2^\sharp)$ only when $\bar{s}_2'$ is an anti-instance of $\bar{s}_1$. This happens because $(\bar{s}_1, D_1^\sharp)$ is the result of the previous iteration, whereas $(\bar{s}_2, D_2^\sharp)$ has been obtained at the current iteration from a join operation that included $(\bar{s}_1, D_1^\sharp)$. Thus another possibility is given by

$$\mathrm{widen}_\phi'\bigl((\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp)\bigr) \stackrel{\mathrm{def}}{=} \bigl(\bar{s}_2, \mathrm{remap}(\bar{s}_1, D_1^\sharp, \bar{s}_2') \nabla D_2^\sharp\bigr). \tag{4.25}$$

It is straightforward to show that (4.24) and (4.25) are correct. Other operators can be defined by using joins and remappings (and don't forget that the underlying domain may have a variety of widening operators to choose from).

4.5.6 Comparing Descriptions

The last operation that is needed in order to put $\mathrm{Pattern}(\mathcal{D}^\sharp)$ at work is for comparing descriptions.

Definition 71 (Approximation ordering.) *The* approximation ordering over $\mathrm{Pattern}(\mathcal{D}^\sharp)$*, denoted by* $\sqsubseteq$*, is defined as follows, for each* $(\bar{s}_1, D_1^\sharp), (\bar{s}_2, D_2^\sharp) \in \mathrm{Pattern}(\mathcal{D}^\sharp)$*:*

$$(\bar{s}_1, D_1^\sharp) \sqsubseteq (\bar{s}_2, D_2^\sharp) \quad \overset{\mathrm{def}}{\iff} \quad \bar{s}_1 = \bar{s}_2 \wedge D_1^\sharp \sqsubseteq D_2^\sharp.$$

It must be stressed that the above approximation ordering is also "approximate", since it does not take into account the peculiarities of $\mathcal{D}^\sharp$. More refined orderings can be obtained in a domain-dependent way, namely, when $\mathcal{D}^\sharp$ has been fixed.

The following result is a trivial consequence of Definition 61 and Proposition 63.

Theorem 72 *If* $(\bar{s}_1, D_1^\sharp) \sqsubseteq (\bar{s}_2, D_2^\sharp)$ *then* $\gamma\bigl((\bar{s}_1, D_1^\sharp)\bigr) \subseteq \bigl((\bar{s}_2, D_2^\sharp)\bigr)$. *Moreover,* $\sqsubseteq$ *is a preorder over* $\mathrm{Pattern}(\mathcal{D}^\sharp)$.

Observe that the ability of comparing descriptions only when they have the same pattern is not restrictive in a data-flow analysis setting. The analyzer, in fact, will only need to compare the descriptions arising from the iteration sequence at two consecutive steps. Moreover, if we denote by $(\bar{s}_n, D_n^\sharp)$ the description at step n, we have

$$(\bar{s}_{i+1}, D_{i+1}^\sharp) = \mathrm{widen}\Biggl((\bar{s}_i, D_i^\sharp), \mathrm{join}_\phi\Bigl(\bigl\{(\bar{s}_i, D_i^\sharp), \dots\bigr\}\Bigr)\Biggr), \tag{4.26}$$

where the widening is possibly omitted. Whether or not the widening has been applied, this implies that $\bar{s}'_{i+1}$ is an anti-instance of $\bar{s}_i$ and

$$\gamma\bigl((\bar{s}_i, D_i^\sharp)\bigr) \subseteq \gamma\bigl((\bar{s}_{i+1}, D_{i+1}^\sharp)\bigr). \tag{4.27}$$

If also the reverse inclusion holds in (4.27) then we have reached a local fixpoint. The analyzer uses the approximate ordering in order to check for this possibility. Namely, it asks whether

$$(\bar{s}_{i+1}, D_{i+1}^\sharp) \sqsubseteq (\bar{s}_i, D_i^\sharp). \tag{4.28}$$

The approximate test, of course, can fail even when equality does hold in (4.27). But this will be a fault of the pattern component only a finite number of times, since $\bar{s}_{i+1}$ is an anti-instance of $\bar{s}_i$ and $\mathbf{T}_{\mathbf{z}}^k$, ordered by the anti-instance relation, has finite height. Thus, there exists $\ell \in \mathbb{N}$ such that, for each $i \geq \ell$, $\bar{s}_i = \bar{s}_\ell$. After the ℓ-th step the accuracy of the approximate ordering is in the hands of $\mathcal{D}^\sharp$.

4.6 What if the Occur-Check is Omitted?

Suppose that the analyzed language omits the *occur-check*, as it is often the case. It has been known since [Col84] that terminating unification procedures without the *occur-check* solve term equations on the domain of *rational trees*. The first question is: are the abstract domains and techniques we all know and use still correct for such languages?

For a precise answer we ought to consider each such domain and the corresponding abstract operations, and prove their correctness with respect to the "new" concrete domain. To our surprise, we were not able to find in the literature any reference to this issue, not even in papers dealing with the analysis of Prolog (which, proverbially, omits the *occur-check*). However, we still hope that this (potentially devastating) problem has not escaped the attention of researchers.

A partial, tentative answer, is that it should not be difficult to prove the correctness, over rational trees and rational unification, of those domains that depend only on the set of variables occurring in terms: the typical domains for groundness and sharing without linearity, for instance. If you only observe the set of variables' occurrences, then a finite term cannot be distinguished from a rational one. The infinite term $f(f(f(\cdots)))$, the rational solution of the equation $X = f(X)$, does not contain any variable and thus is ground. In the rational solution of $X = f(X, Y)$, instead, only the variable Y occurs. If X denotes a ground rational tree then also Y denotes a ground rational tree, and the converse holds too. So, as long as you abstract a concrete equation of the form $X = t$ with the ground dependency

$$X \leftrightarrow \bigwedge\bigl(\mathit{vars}(t) \setminus \{X\}\bigr),$$

the usual domains for groundness should remain correct also when the *occur-check* is omitted in the analyzed language. (See Chapter 6 on the subject of groundness analysis.) This implicit pattern of ignoring the variables' occurrences that are responsible for the introduction of a cycle works whenever only the set of variables occurring in (infinite) terms is relevant. This is because, if $\tilde{t}$ is the most general rational unifier of $X = t$, with X occurring in t, we can write

$$\mathit{vars}(\tilde{t}\,) = \bigl(\mathit{vars}(t) \setminus \{X\}\bigr) \cup \mathit{vars}(\tilde{t}\,). \tag{4.29}$$

The least solution of (4.29) is clearly $\mathit{vars}(\tilde{t}\,) = \mathit{vars}(t) \setminus \{X\}$. Notice that this pattern does not work when the multiplicity of occurrences is relevant, as we will now see.

For a finite term t we say that *t is linear* if and only if t does not contain multiple occurrences of any variable. The linearity property is not very interesting in itself, but it allows us to improve sharing analyses. When two

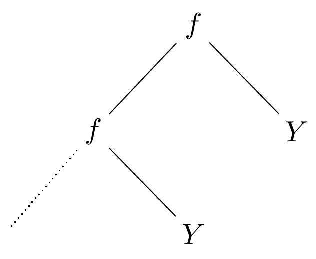


Figure 4.5: The rational tree that solves $X = f(X, Y)$.

terms t and u are unified, knowing that t is linear allows one to conclude that any two variables in u that did not share *before* the unification, will not share *after* the unification. The notion of linearity can easily be extended to rational trees, but care must be taken. When abstracting the constraint $X = f(X, Y)$, for instance, we cannot ignore the occurrence of X in the right-hand side. For a very good reason: this occurrence is responsible for the fact that the result of the unification is a non-linear term. Indeed, the resulting rational tree has an *infinite* number of occurrences of Y, as shown in Figure 4.5. Failure to recognize this fact leads to an unsound sharing analysis: if $\tilde{t}$ is the most general rational unifier of X and $f(X, Y)$, unifying $f(f(\_, A), B)$ with $\tilde{t}$ makes A and B share, as expected.

This is not the place for reconsidering all the domains that have been studied for the analysis of (constraint) logic programs. We do believe that most of them can be easily fixed in order to obtain safe approximations of the concrete domains based on rational trees. So, let us simply assume we have some domain $\mathcal{D}^\sharp$ that is a correct abstraction of a domain of extended rational trees. Can we extend the parametric structural construction so that we can apply it to $\mathcal{D}^\sharp$ and still obtain a correct analysis?

When the analyzed language does not perform the *occur-check* we can no longer *fail* when a cyclic binding is attempted in the pattern component: something else has to be done. Notice that this problem cannot be escaped easily. Even the unreasonable hypothesis that one deals only with *occur-check free* programs[3] does not prevent an analyzer employing `Pat(`$\Re$`)` or Pattern($\mathcal{D}^\sharp$) from trying to build a cyclic pattern (because the analysis was not strong enough to detect an impossible computation path).

The obvious way to go is to have a tuple of rational trees, instead of finite terms, in the pattern component with all the consequent modifications. This, however, might be undesirable because

1. it is more expensive: all the tree visits must employ some marking technique in order to avoid infinite loops; computing common anti-

[3]Note that *occur-check freedom* is an undecidable program property.

instances is more difficult than in the finite case; the same holds for detecting when two rational trees are variant of each other; garbage collection is more complicated and so on;

2. cyclic bindings arise in the analysis of very few programs.[4]

The unlikeliness of the phenomenon, while irrelevant as far as the soundness issues are concerned, seems to discourage the adoption in practice of rational trees for the pattern component: very few programs can benefit from the little extra precision so gained, at a comparatively high cost. We thus prefer leaving the pattern component as it is now: a tuple of finite terms. But, what do we do in the case of cyclic bindings?

If we restrict ourselves to domains expressing monotonic properties[5], then ignoring the bindings that would cause the introduction of a cycle is a safe (though imprecise) way out: throwing away information is always correct for these domains. In this particular case, lines 9 and 10 of Algorithm 1 on page 98 could simply be removed. This is not true for non-monotonic properties: the cyclic bindings cannot be ignored altogether. As previously noted, omitting communication of the cyclic bindings to the underlying domain $\mathcal{D}^\sharp$ implies, in general, loosing soundness. Let us consider the following simple program:

```
p(X, Y) :-
    Z = f(Z, X, Y),
    Z = f(_, A, A).
```

Most Prolog and CLP systems give the answer `X = Y` to the query `?- p(X, Y).` Consequently, any non-trivial sharing analysis that ignores the cyclic binding is going to obtain wrong results.

4.6.1 Unification without Occur-Check

The above discussion motivates the introduction of a revised unification algorithm, given as Algorithm 2, where we take into account the cyclic bindings both in the pattern component and the underlying domain. Notice how the cyclic binding is avoided in the pattern component: by substituting the dangerous occurrences of Z_h with a new variable. This, of course, is more precise than omitting the binding in the pattern component.

[4]Only one out of more than a hundred third-party programs in the test-suite of China used to provoke a cyclic binding during the analysis. This happened with an early version that did not take some built-ins into account. With the current version this does not happen anymore.

[5]Properties that are preserved as concrete computations progress.

```
procedure unify($\bar{s}, D^\sharp, t, u$)
  if $t \neq u$ then
    if $t = f(t_1, \dots, t_n)$ and $u = f(u_1, \dots, u_n)$ then
      for all $i = 1, \dots, n$ do
        unify($\bar{s}, D^\sharp, t_i, u_i$)
    else if $t = Z_h$ then
      if $Z_h$ does not occur in $u$ then
        $D^\sharp$ := bind($\bar{s}, D^\sharp, u, Z_h$) {invokes underlying domain}
        $Z_h := u$ {instantiates all the occurrences of $Z_h$}
      else
        $D^\sharp$ := cyclic($\bar{s}, D^\sharp, u, Z_h$) {underlying domain}
        $Z_h := u[Z_{m+1}/Z_h]$ {cycle is broken: $m = |vars(\bar{s})|$}
      $\bar{s} := \eta(\bar{s})$ {normalization}
    else if $u = Z_k$ then
      unify($\bar{s}, D^\sharp, u, t$)
    else
      $D^\sharp := \bot^\sharp$
```
Algorithm 2: Revised unification for the structural domain.

Cyclic Binding to a Compound

The operation cyclic$(\bar{s}, D^\sharp, u, Z_h)$ must now be specified. We assume again that $\bigl|vars(\bar{s})\bigr| = m$, with $m \geq 0$. The result of the operation will be a description $D_1^\sharp$ such that

$$\tilde{\gamma}(D_1^\sharp) \subseteq \tilde{\mathbf{T}}_{\mathbf{z}}^m \times \mathcal{D}^\flat,$$

where $\tilde{\mathbf{T}}_{\mathbf{z}}$ denotes the set of all normalized rational trees and $\tilde{\gamma}$ is the new concretization function. Notice that $D_1^\sharp$ denotes an m-tuple of rational trees: Z_h will not be referenced anymore, but a new variable Z_{m+1} has been introduced.

The following definition is reproduced from [Kei94].

Definition 73 (Rational solved form.) [Col82, Col84] *A set of equation is* circular *if it has the form*

$$\left\{\begin{array}{r} X_1 = X_2,\\ X_2 = X_3,\\ \vdots\\ X_{n-1} = X_n,\\ X_n = X_1 \end{array}\right\}.$$

A rational solved form *is a conjunction of equations of the form*

$$\left\{\begin{array}{c} X_1 = t_1, \\ X_2 = t_2, \\ \vdots \\ X_n = t_n \end{array}\right\}$$

not containing any circular subset.

Each rational solved form corresponds to a tuple of rational trees. Conversely, each rational tree can be associated to rational solved form in which "the root" has been identified. Let $\tilde{t}$ be a rational tree, and let $\mathbf{u}$ be a fixed, infinite set of special variables, disjoint from any other set of variables we have mentioned so far. When U is a variable not occurring in $\tilde{t}$, we will denote by $\mathrm{eq}_U(\tilde{t}\,)$ a rational solved form R such that

1. if $\mathrm{lhs}(R)$ and $\mathrm{rhs}(R)$ denote the set of variables on the left-hand sides of the equations in R and the set of terms occurring in the right-hand sides, respectively, then

$$\mathrm{lhs}(R) \cap \mathit{vars}\bigl(\mathrm{rhs}(R)\bigr) \subset \mathbf{u};$$

2. U describes $\tilde{t}$ in R.

For instance, if $\tilde{t}$ is the rational tree depicted in Figure 4.5 then $\mathrm{eq}_U(\tilde{t}\,) = \bigl\{U = f(U, Y)\bigr\}$.

Let us define $\tau(k) \colon \{1, \dots, m\} \to \{1, \dots, m\}$ exactly as we did in (4.19) on page 100 (but note that the domain has changed). Then $D_1^\sharp$ must satisfy

$$\gamma(D_1^\sharp) \supseteq \left\{ \Bigl(\bigl(\pi_{\tau(k)}(\bar{t}\,)\bigr)_{k=1}^{m}, D_1^\flat \Bigr) \;\middle|\; \begin{array}{l} (\bar{t}, D^\flat) \in \tilde{\gamma}(D^\sharp) \\ \{Z_{j_1} = u_{j_1}, \dots, Z_{j_l} = u_{j_l}, \dots\} \\ \quad \text{is a r.s.f. (if any) of} \\ \qquad \mathrm{eq}_U\bigl((\pi_h(\bar{t}\,)\bigr) \cup \{U = u\} \\ D_1^\flat = D^\flat\bigl[\pi_{j_1}(\bar{t}\,) = u_{j_1}, \\ \qquad\qquad \dots, \pi_{j_l}(\bar{t}\,) = u_{j_l}\bigr] \end{array} \right\},$$

where $U \in \mathbf{u}$.

4.7 Conclusion

We have presented the rational construction of a generic domain for structural analysis of $\mathrm{CLP}(\mathcal{H}_{\mathcal{X}})$ languages: $\mathrm{Pattern}(\mathcal{D}^\sharp_{\mathcal{H}_{\mathcal{X}}})$, where the parameter $\mathcal{D}^\sharp_{\mathcal{H}_{\mathcal{X}}}$ is an abstract domain satisfying certain properties, We build on the parameterized `Pat(ℜ)` domain of Cortesi *et al.* [CLV93, CLV94], which is restricted to logic programs. However, while `Pat(ℜ)` is presented as a *specific implementation* of a generic structural domain, our formalization is independent from specific implementation techniques. Reasoning at a higher level of abstraction we are able to fully justify the ideas behind the structural domain. In particular, appealing to familiar notions of unification theory, we can identify an important parameter (a common anti-instance function, missing in [CLV93]) that gives some control over the precision and computational cost of the resulting generic structural domain.

In addition, we have corrected an oversight in the work of Cortesi *et al.*While they assume explicitly that no cyclic binding will be attempted during the analysis, this cannot be granted in any way. It is indeed easy to fix their unification algorithm if one sticks to the standard assumption that the analyzed language does not omit the *occur-check* in the unification procedure. Unfortunately, it is well-known that many implemented CLP($\mathcal{H}_{\mathcal{X}}$) languages (in particular, almost all Prolog systems) do omit the *occur-check*. We have discussed, apparently for the first time, the impact of this problem on data-flow analysis and how some commonly-used abstract domains can be modified in order to ensure precision and correctness. We have also shown how to modify our generic domain for structural information in order to deal with the majority of languages that employ unification without the *occur-check*.

For future work, the more urgent thing to do is to reconsider all the domains that have been proposed for the analysis of (constraint) logic programs and prove that they (or suitable modification of them) are correct with respect to rational trees and rational unification. This might be a non-trivial task, if one considers that there seems to be no published proof of the correctness of the *Sharing* domain [JL89], not even in the case of finite trees [Zaf95][6].

[6]Enea Zaffanella in a desperate search for the proof, got in touch, in the fall of 1995, with three of the major authors on the subject of aliasing analysis. The result was disheartening: nobody knew where the proof could be found.

Chapter 5

Range and Relations Analysis

Contents

5.1 Introduction

There are several CLP and cc languages that incorporate some kind of numeric domain. Here is a (certainly incomplete) list including the particular numeric domain(s) employed:

CHIP: numeric constraints over finite domains, linear rational constraints [DVS+88];

CLP($\mathcal{R}$): real linear arithmetic (so to speak[1]), delay mechanism for non-linear constraints [JM87, JMSY92b];

QUAD-CLP($\mathcal{R}$): extends CLP($\mathcal{R}$) with the handling of quadratic constraints, rewriting them so that they can actually be decided upon or generating a conservative approximation for them (while still delaying them) [PB94];

CIAL: linear arithmetic on real intervals; it employs a linear constraint solver (based on preconditioned interval Gauss-Seidel method) in addition to the interval narrowing solver [CL94, LL94];

clp(Q, R): real[2] and rational linear arithmetic, linear disequations included, delay mechanism for non-linear constraints [Hol95];

Prolog-III: linear rational arithmetic [Col90];

Prolog-IV: an ISO-compliant replacement for Prolog-III;

Trilogy: linear arithmetic both over the integers and over the reals [Vod88a, Vod88b];

[1]The CLP($\mathcal{R}$) system implements a domain based on floating-point numbers where rounding errors are simply disregarded. While the relevant literature unanimously regards CLP($\mathcal{R}$) as a CLP language, we follow this usage with embarrassment. Indeed, the 'L' in CLP($\mathcal{R}$) has no meaning whatsoever.

[2]The same comments as for CLP($\mathcal{R}$) apply.

```
sumto(0, 0).        % clause 1
sumto(N, S) :-      % clause 2
    integer(N),
    N >= 1,
    N = N1 + 1,
    S = N + S1,
    sumto(N1, S1).

?- integer(N), sumto(N, S).
```

Figure 5.1: Sum of the first `N` naturals in CLP($\mathcal{N}$).

CAL: (*Contrainte Avec Logique*) non-linear constraints over complex numbers [ASS$^+$88, SA89]. A parallel version exists and is called GDCC [AH92];

RISC-CLP(Real): non-linear constraints over (algebraic) real numbers [Hon92, Hon93, Cap93];

CLP(BNR): non-linear arithmetic on real intervals plus equations over the integers [OV93];

CLP(F): domains include integers, reals, real-valued functions of one variable, and vectors of domain elements; function variables are constrained by functional equations and by putting interval constraints on the values of their derivatives at points and intervals [Hic94];

Newton: non-linear interval arithmetic with equations and inequalities, provides both constrained and unconstrained optimization primitives [BMV94];

`clp(FD)`: finite domains *à la* CHIP [DC93];

Echidna: finite domains and real intervals arithmetic [HSS$^+$92];

`cc(FD)`: finite domains [VSD92a, VSD92b];

AKL(FD): finite domains [CJH94, Jan94].

Roughly speaking, the target of the data-flow analysis we present is the derivation of numeric constraints that, at some program point p, are *redundant*. This means that they are guaranteed to hold whenever control reaches p. Consider, for instance, the self-contained program and query in Figure 5.1. On exit from clause 2 the following constraints (and infinitely many others) are redundant:

$$\mathtt{N} \in \mathbb{N}, \qquad \mathtt{N} \geq 1, \tag{5.1}$$

$$\mathtt{S} \in \mathbb{N}, \qquad \mathtt{S} \geq 1, \tag{5.2}$$

$$\mathtt{N} \leq \mathtt{S}, \tag{5.3}$$

$$\mathtt{S} \leq \mathtt{N}^2, \qquad \mathtt{S} = \mathtt{N}^2 + \mathtt{N}. \tag{5.4}$$

Redundant constraints can be further classified as follows:

- *truly redundant constraints*: they are in the program's text, but they are either implied by the constraints accumulated before reaching p (in this case they could be ignored) or they will be implied by the other constraints collected through any successful computation from p (in which case they can be subject to simplified treatment). We call these constraints *past redundant* and *future redundant*, respectively[3]. On entry to clause 2 the constraint `integer(N)` is past redundant: any invocation of clause 2 happens in the context of a constraint store that entails it. The constraints `integer(N)` and `N >= 1` are future redundant: any successful computation starting from clause 2 ends up with a constraint store entailing them. These facts can be established by means of a simple inductive argument.

- *implicit constraints*: they are not present in the program's text, but they are guaranteed to hold if the computation arrives at p: thus the constraints (5.2)–(5.4) are implicit on exit from clause 2.

Notice that adding to a clause a constraint c that is implicit on exit from that clause would result in c being future redundant. As a last remark, the analysis which is the subject of this chapter is able to infer that the constraints (5.1)–(5.3) belong to the above categories.

Since in this work we restrict ourselves to considering only clause entry and clause (successful) exit as program points, the expressions *numeric call-patterns* and *numeric success-patterns* can also be used to denote redundant constraints at those points. However, the implicit/redundant terminology helps in understanding the applications.

Our interest in automated detection of implicit and redundant numeric constraints is motivated by the wide range of applications they have in semantics-based program manipulation. Moreover, while analysis techniques devoted to the discovery of implicit constraints over some Herbrand universe are well-known (e.g., depth-k abstractions [ST84, MS88], types [JB92, VCL94] and so on), in the field of numeric domains very little has been done. After our original proposal about employing constraint propagation techniques [BGL92, BGL93], Marriott and Stuckey envisaged the use of two

[3] Jørgensen *et al.* use a more restrictive notion of future redundancy [JMM91].

domains: a simple one for *sign analysis*, *Sign*, and a (much) more complex domain, *CHull*, based on convex polyhedra. Following [BGL92, BGL93], Janssens *et al.* presented three interval-based approximations for the numerical leaves of (extended) Herbrand terms [JBE94]. The main merit of [JBE94] is that it contains the first published proposal on how to approximate the propagation of numerical information through unification constraints. Janssens *et al.* have also implemented their first approximations giving an indication that useful approximations of numerical values are feasible. These indications were reinforced, using a more complex numeric domain, in [Bag94]. Very recently, what probably is the first implementation of *CHull*, the Stan system, has been briefly described [Han96]. Stan, however, is limited to purely-numerical CLP languages.

We present a general methodology for the detection of redundant numeric constraints. The techniques we use for reasoning about arithmetic constraints come from the world of Artificial Intelligence, and are known under the generic name of *constraint propagation* [Dav87]. Notice that we do not commit ourselves to any specific CLP language, even though all the languages mentioned above can be profitably analyzed with the techniques we propose. In particular, we allow and reason about linear and non-linear constraints, integer, rational, and real numbers as well as domains based on intervals.

5.2 What Redundant Constraints Are For

In this section we show a number of applications for redundant constraints. Some of them are domain-independent, but we concentrate on redundant constraints over numeric domains. We will see that the range of situations where they prove to be useful is quite wide. It should then be clear that their automatic detection is very important for the whole field of semantics-based manipulation of CLP programs. The first four subsections are devoted to applications related to the compilation of CLP programs. Traditionally, this is one of the major interest areas for data-flow analysis. The remaining two subsections describe applications of redundant constraints to the improvement of other data-flow analyses.

It must be observed that here we are dealing with some kind of information that *might* be useful in the compilation process by allowing for the production of faster code. In this work the focus is on techniques for automatic deduction of run-time program properties. Deciding in which cases and to what extent this information gives rise to actual speedups is a completely different and complicated issue. While some compile-time transformation (e.g., domain reduction, determinacy exploitation, and call-graph simplification) can be definitely recognized as optimizations, others (like constraint anticipation or *refinement* [MS93]) require careful consideration as their usefulness depends on several factors. Deep knowledge of the constraint solver and extensive experimentation might be necessary for this purpose. One must also be aware of the interplay between different optimizations, as the combination of two transformations is not guaranteed to be an optimization, even though the original ones were such, when applied in isolation. Another point to be taken into account is that a transformation applied in one point of the program might prevent the application of another transformation at another program point. When one of these situations occur a tradeoff is faced and some kind of compromise must be found with the aim of maximizing benefits. Sometimes knowledge about the particular system at hand can be enough to definitely conclude that a particular (combination of) optimization(s) is worthwhile. In other cases the decisions about what transformations to apply, and where, are more complex and must be dealt with more flexibly by the compiler and/or the user. Techniques devoted to the solution of these problems include:

- performing complexity/cost analysis and using the results as a base for compilation decisions;
- two-phase compilation, e.g., compiling first an instrumented version of the program, profiling it with some "representative" input, and using the profile data to drive the final compilation step;
- simple heuristics, e.g., by analyzing the program's call graph, and deserving more work/optimization to code which is more deeply nested;
- user intervention by means of program annotations; in this task the user can be assisted by providing him with information originating from the above techniques.

5.2.1 Domain Reduction

In CLP systems supporting finite domains, like CHIP, `clp(FD)`, and Echidna, variables can range over finite sets of integer numbers. These sets must be specified by the programmer. There are combinatorial problems, such as n-queens, where this operation is trivial: variables denoting row or column indexes range over $\{1, \ldots, n\}$. For other problems, like scheduling, the ranges of variables are not so obvious. Leaving the user alone in the (tedious) task of specifying the lower and upper bounds for any variable involved in the problem is inadvisable. On one hand the user can give bounds that are too tight, thus loosing solutions. On the other hand he can exceed in being conservative by specifying bounds that are too loose. In that case he will incur inefficiency, as finite domains constraint solvers work by gradually eliminating values from variable's ranges.

A solution to this problem is either to assist the user during program development or to provide him with a compiler able to tighten the bounds

```
mc(N, N-10) :-     % clause 1
    N > 100.
mc(N, M) :-        % clause 2
    N <= 100,
    mc(N+11, U),
    mc(U, M).
```

Figure 5.2: McCarthy's 91-function in CLP($\mathcal{N}$).

he has specified. In this case the programmer can take the relaxing habit of being conservative, relying on the compiler's ability of achieving domain reduction. Whatever the programmer's habit is, domain reduction at compile-time can be an important optimization as possibly many inconsistent values can be removed once and for all from the variable's domains. This has to be contrasted with the situation where these inconsistent values are removed over and over again during the computation.

The following example is somewhat unnatural, but it shows how it can be difficult for an unassisted human to provide good variable's bounds. In contrast, it shows how relatively tight bounds can be "hidden" in a program and how they can be "discovered" by means of data-flow analysis. The program in Figure 5.2 is a CLP($\mathcal{N}$) version of the McCarthy's 91-function. If the program is complemented with the domain declaration

```
domain mc([0 .. 200], [0 .. 200]).
```

the China analyzer is able to derive the following success-patterns for `mc(A, B)`:

$$\begin{array}{lll} 101 \le \mathtt{A} \le 200, & 91 \le \mathtt{B} \le 190, & \text{for clause 1;} \\ 0 \le \mathtt{A} \le 100, & 91 = \mathtt{B}, & \text{for clause 2.} \end{array}$$

Notice how the analyzer correctly infers that any successful derivation from the second clause must end up with an answer constraint entailing $\mathtt{A} = 91$. The same pattern is derived with the more liberal declaration[4]

```
domain mc([0 .. 200], _).
```

With no domain declaration at all the inferred patterns are

$$\begin{array}{lll} \mathtt{A} \ge 101, & \mathtt{B} \ge 91, & \text{for clause 1;} \\ \mathtt{A} \le 100, & \mathtt{B} = 91, & \text{for clause 2.} \end{array}$$

[4]The underscore sign '_' stands, in this context, for the set of all the integer numbers.

5.2.2 Extracting Determinacy

In the history of efficient Prolog execution a major role has been played by the avoidance of unnecessary backtracking, since this is the principal source of inefficiency. These efforts go back to the WAM [War83] with the indexing mechanism used to reduce *shallow backtracking*. A more general way of avoiding backtracking is to use global analysis for detecting conditions under which clauses may succeed in a program (determinacy analysis). Run-time tests to check this conditions may allow for the elimination of *choice points* or, at least, for the reduction of backtracking search (determinacy exploitation).

Notice that backtracking in CLP can be significantly more complex than in Prolog. The reason is that in CLP languages it is not enough to store a reference to variables that have become bound since the last choice point creation, and to unbind them on backtracking. In CLP it is generally necessary to record changes to constraints, as expressions appearing in them can assume different forms while the computation proceeds [JM94].

We show here, more or less following the exposition in [DRRS93], how redundant constraints can be used for determinacy discovery and exploitation. Consider a CLP program P and a clause R in P of the form

$$R: \quad p(\bar{X}) :- c \,\Box\, q_1(\bar{X}_1), \ldots, q_n(\bar{X}_n). \tag{5.5}$$

Suppose now that data-flow analysis of P computes the success-pattern ϕ for clause R. Substituting the clause

$$R': \quad p(\bar{X}) :- \phi \wedge c \,\Box\, q_1(\bar{X}_1), \ldots, q_n(\bar{X}_n). \tag{5.6}$$

for R yields a program that is logically equivalent to P. Suppose also the analysis derives that ψ is a *correct call-pattern* for the atom $p(\bar{X})$. This means that, whenever $p(\bar{X})$ is selected in a *successful* computation path, ψ holds.[5] Dawson *et al.* define the *clause condition* of R as $\Phi = \psi \wedge \phi$. The clause condition Φ is a necessary condition for R to succeed when $p(\bar{X})$ is invoked from a successful context. Every successful computation calling R is such that, on successful exit from R, the accumulated constraints entail Φ. In other words, if the concrete constraint store on entry to clause R is incompatible with Φ, then the current computation branch can safely be abandoned. This fact can be captured by rewriting clause R into

$$R'': \quad p(\bar{X}) :- \Phi \wedge c \,\Box\, q_1(\bar{X}_1), \ldots, q_n(\bar{X}_n). \tag{5.7}$$

Let now P'' be the program obtained by transforming each clause of the form (5.5) into the form (5.7) as explained. It turns out that P and P'' are logically equivalent, and that the exposed clause conditions can be used for

```
fib(N, F) :-              % clause 1
    N = 0, F = 1.
fib(N, F) :-              % clause 2
    N = 1, F = 1.
fib(N, F) :-              % clause 3
    N > 1,
    F = F1 + F2,
    fib(N-1, F1),
    fib(N-2, F2).
```

Figure 5.3: Fibonacci's sequence in CLP($\mathcal{N}$).

detecting and exploiting determinacy in the compilation of P.[6]

Suppose the predicate p is defined in P by clauses $R_1, \ldots, R_m$ with respective success-patterns $\phi_1, \ldots, \phi_m$ and clause conditions $\Phi_1, \ldots, \Phi_m$. What can happen is that, for each $i, j = 1, \ldots, n$ with $i \neq j$, the constraint $\phi_i \wedge \phi_j$ is unsatisfiable. In this case p is *deterministic*, namely, for each actual call-pattern that is strong enough we can select the unique clause that might bring the computation to success. When the above condition fails, we may still have that $\Phi_i \wedge \Phi_j$ is unsatisfiable whenever $i \neq j$. Thus p might not be deterministic in itself, but we are guaranteed that in P it is always used in a determinate way.

In both cases, when the conditions are simple enough to be checked, it is possible to avoid the creation of a choice point jumping directly to the unique clause that has a chance of success. Weaker assumptions still allow to exclude clauses from search by partitioning the set of clauses into "mutually incompatible" subsets. Of course, determinacy exploitation requires the existence of an adequate indexing mechanism [HM89, RRW90, CRR92]. As an example consider the famous CLP program, reproduced in Figure 5.3, expressing the Fibonacci sequence. China derives the following success-patterns for the `fib` program:

$\mathtt{N} = 0$,	$\mathtt{F} = 1$,	for clause 1;
$\mathtt{N} = 1$,	$\mathtt{F} = 1$,	for clause 2;
$\mathtt{N} \in \{2, 3, \ldots\}$,	$\mathtt{F} \in \{2, 3, \ldots\}$,	for clause 3.

(The same patterns would have been derived even if `N > 1` did not appear in clause 3. China can also derive `N <= F` for clause 3.) Notice that, when `fib`

[6]Notice that correct call-patterns are, in general, stronger than "plain" call-patterns. This is because only successful computation paths are considered for correct call-patterns. As a consequence, the present discussion remains valid even if plain call-patterns are considered.

is called with the a definite (or ground) first argument, a simple test allows to select the appropriate clause without creating a choice point. When `fib` is called with its second argument instantiated then a similar test allows at least to discriminate between clause 3, if $\mathtt{F} \geq 2$ (no choice point), and clauses 1 and 2, if $\mathtt{F} = 1$ (a choice point is necessary). In both cases some calls can be made to hit an immediate failure instead of proceeding deeper before failing or looping forever, e.g., `fib(1.5, X)`, `fib(X, -1)`.

5.2.3 Static Call Graph Simplification

Suppose we are given a methodology for approximate deduction of implicit constraints. Then, if the approximate constraint system has a non-trivial notion of consistency[7], we also have a methodology for approximate consistency checking which we can use for control-flow analysis of CLP programs. By soundness, when *false* is derived as an implicit constraint we can safely conclude that the original set of constraints was unsatisfiable, and that the computation branch responsible for this state of affairs cannot possibly lead to any success.

This information can be employed at compile-time to generate a simplified call graph for the program at hand. Let

$$R: \quad p(\bar{X}) :- c \,\Box\, q_1(\bar{Y}_1), \dots , q_n(\bar{Y}_n).$$

be a program clause, and let the predicate q_i be defined by clauses R_{i_1}, …, $R_{i_{m_i}}$, for $i = 1, \dots, n$. While performing the analysis we may discover that whenever we use clause R_{i_j}, with $1 \leq j \leq m_i$, to resolve with $q_i(\bar{Y}_i)$, we end up with an unsatisfiable constraint. In this case we can drop the edge from the $q_i(\bar{Y}_i)$ call in the above clause to R_{i_j} from the syntactic call graph of the program. This simplification can be used for generating faster code.[8] We illustrate this point by means of an example. CHINA, when presented with the `fib` program of Figure 5.3, produces the following call graph representation:

$$3 :- \bigl\langle \{2, 3\}, \{1, 2, 3\} \bigr\rangle. \tag{5.8}$$

The above notation can be read as follows:

> if a call to clause 3 has to succeed, then only clauses 2 and 3 can successfully used to resolve with the first recursive call, while the

[7]This is not the case for, say, groundness analysis, where abstract constraints are always satisfiable. A non-trivial approximation of inconsistency can be obtained with the domains that will presented later in this chapter, with the domain for simple types of Section 3.3.2 on page 46, with structural information analysis as in Chapter 4, with depth-k abstractions, types and so forth.

[8]We do not want to make a strong claim, but we have not found any published proposal for this optimization besides [Bag94].

```
fib/3_1:     try_me_else fib/3_2
             < code for clause 1 >
fib/3_2:     retry_me_else fib/3_3
fib/3_2a:    < code for clause 2 >
fib/3_3:     trust_me
fib/3_3a:    ...
             call fib/3_2_3
             call fib/3_1
             ...

fib/3_2_3:   try fib/3_2a
             trust fib/3_3a
```

Figure 5.4: Fragment of WAM-like code for the 3rd clause of `fib` to be executed when the 1st argument is not definite.

> second recursive call is unrestricted, i.e., all the clauses for `fib` can lead to success.

In other words, when treating the first recursive call of clause 3, clause 1 can be forgotten. This information allows for search space reduction without any overhead, in the case where the third clause of `fib` is called with the first argument uninstantiated, that is, when search is not avoidable. Figure 5.4 shows how this can be achieved by means of a simple compilation scheme in the setting of the WAM and its extensions [JMSY92a].

Notice how this simple transformation reduces the amount of backtracking. In fact, every time clause 3 is invoked a pointless call to clause 1 is avoided, with the consequent saving of one backtracking. It is easy to think about more involved examples where the pruned computation branch would have proceeded deeper before failing, thus wasting more work. When the number of applicable clauses is found to be one, a choice point can be avoided, thus achieving greater savings. In these cases determinacy is exploited without any run-time effort. The optimization is always achieved without any time overhead at the price of at most a small, constant increase of space usage for additional code. An example where choice point creations are avoided is the following:

```
square(0, 0).              % clause 1
square(X+1, Y+2*X+1) :-    % clause 2
    square(X, Y).

pythagoras(X, Y, Z) :-     % clause 3
    X < Y,
    Y < Z,
                SX + SY = SZ,
                square(X, SX),
                square(Y, SY),
                square(Z, SZ).
```

The detected call graph in this case is

$$\begin{aligned}&2 :- \{1, 2\},\\ &3 :- \{1, 2\}, \{2\}, \{2\},\end{aligned}$$

showing that the second and third procedure calls of clause 3 can be compiled as direct jumps to the code for clause 2.

The strong points of call graph simplification are the following:

- only a small space overhead is incurred by the optimized program;
- no time overhead: the optimized program will *never* be slower than the unoptimized one;
- almost no additional cost for the analysis: CHINA simply records which clauses have been successfully used to resolve with each atom in the clauses' bodies.

As a final remark, notice that the simplified call graph for `fib` given in (5.8) is actually obtained by abstracting away the relational information from

$$3 :- \bigl\{\langle 2, 1\rangle, \langle 3, 2\rangle, \langle 3, 3\rangle\bigr\}. \tag{5.9}$$

In addition to the fact that clause 1 can be disregarded when dealing with the first recursive call in clause 3, (5.9) specifies that, when clause 2 is selected to resolve with the first recursive call, then only clause 1 is a sensible choice for the second recursive call. Similarly, when clause 3 is recursively called first, the subsequent call should ignore clause 1. In summary, while the static call graph of `fib` contains 9 edges, the simplified call graph represented by (5.9) specifies that only 3 of them can lead to a successful computation. This additional information, which, again, is cheaply obtained as a byproduct of the analysis, can also be useful. Its direct exploitation at run-time requires a mechanism whereby the choice of one clause to resolve with a body atom influences the choices of the clauses for subsequent atoms. Another application concerns guiding program transformations such as unfolding: a partial evaluator would immediately know that clause 3 can be unfolded in at most three different, sensible ways.[9] Finally, the simplified call graph can be used in order to simplify subsequent analyses of the same program.

[9]A small experiment with the CLP($\mathcal{R}$) system: the queries `fib(19, F)` and `fib(N, 6765)` took 1.16 and 3.65 seconds, respectively. By unfolding clause 3 a tenfold speedup is obtained: 0.1 and 0.38 seconds, respectively.

```
mortgage(P, T, I, B, MP) :-         % clause 1
    T = 1,
    B = P*(I + 1) - MP.
mortgage(P, T, I, B, MP) :-         % clause 2
    T > 1,
    P >= 0,
    mortgage(P*(I + 1) - MP, T - 1, I, B, MP).
```

Figure 5.5: Standard mortgage relationship in CLP($\mathcal{R}$).

5.2.4 Future-Redundant Constraints

As mentioned in the introduction, we say that a constraint is *future redundant* if, after the satisfiability check, adding or not adding it to the *current constraint* (i.e., the constraint accumulated so far in the computation), will not affect any answer constraints. Consider the `mortgage` program in Figure 5.5. In any derivation from the second clause the constraint $\mathtt{T}' = \mathtt{T} - 1 \wedge \mathtt{T}' > 1$ or $\mathtt{T}' = \mathtt{T} - 1 \wedge \mathtt{T}' = 1$ will be encountered, and both imply $\mathtt{T} > 1$. Thus `T > 1` in the second clause is future redundant. If `T` is uninstantiated, not adding the future redundant constraint reduces the "size" of the current constraint, thus reducing the complexity of any subsequent satisfiability check. A dramatic speed-up is obtainable thanks to this optimization [JMM91]. Notice that the definition of future redundant constraint given in [JMM91] is more restrictive than ours and, while allowing a stronger result for the equivalence of the optimized program with respect to the original one, it fails to capture situations which can be important not only for compilation purposes (see Section 5.2.5 below). As an example, consider a version of the `fib` program in Figure 5.3 on page 119, where the constraint `F >= 2` has been added in the recursive clause. This is future redundant for our definition (and is recognized as such by the analyzer), while it is not for the definition in [JMM91].

5.2.5 Improving any Other Analysis

Detecting redundant constraints improves the precision (and sometimes also the efficiency) of any other analysis. This is due to the ability, described in Section 5.2.3, of discovering computation branches which are dead. The obvious implication is that these branches can be safely excluded from analysis: the result is better precision, because the *merge-over-all-paths* operation needed for ensuring soundness [CC77] has potentially a less dramatic effect, and possibly improved efficiency, because less branches need to be analyzed.

We illustrate the first point by means of an example. Consider the following predicate definition:

```
r(X, Y, Z) :-
    Y < X,
    Z = 0.
r(X, Y, Z) :-
    Y >= X,
    Z = Y - X.
```

This defines the so called *ramp function*, and is one of the linear piecewise functions which are used to build simple mathematical models of valuing options and other financial instruments such as stocks and bonds [LMY87]. Suppose we are interested in groundness analysis of a program containing the above clauses. A standard groundness analyzer cannot derive any useful success-pattern for a call to `r(X, Y, Z)` where nothing is known about the groundness of variables, even though the "real" call-pattern implied `Y` $<$ `X`. In contrast, in the same situation a definiteness analyzer employing also a numerical domain (or supplemented with the simplified call graph described in Section 5.2.3) can deduce the success-pattern *ground*(`Z`). This is due to the ability of recognizing that, in the mentioned context, only the first clause is applicable. Indeed, this is what happens in the analysis of the `option` program distributed with the CLP($\mathcal{R}$) system.

Improving the Results of Some Other Analyses

The previous section showed how our analysis can generally improve the others. There are, however, more specific situations where its results may be of help. For example, the freeness analysis proposed in [DJBC93] can be greatly improved by the detection of future redundant constraints. In fact, their abstraction is such that constraints like $N \geq 0$ "destroy" (often unnecessarily) the freeness of N. This kind of constraints are very commonly used as clause guards, and many of them can be recognized as being future redundant. This information imply that they do not need to be abstracted, with the corresponding precision gain.

The analysis described in [Han93] aims at the compile-time detection of those non-linear constraints, which are delayed in the CLP($\mathcal{R}$) implementation, that will become linear at run time. This analysis is important for remedying the limitation of CLP($\mathcal{R}$) to linear constraints by incorporating powerful (and computationally complex) methods from computer algebra as the ones employed in RISC-CLP(Real). With the results of the above analysis this extension can be done in a smooth way: non-linear constraints which are guaranteed to become linear will be simply delayed, while only the other non-linear constraints will be treated with the special solving techniques. Thus, programs not requiring the extra power of these techniques will be hopefully recognized as such, and will not pay any penalties. The analysis of [Han93] is a kind of definiteness. One of its difficulties shows up when considering the simplest non-linear constraint: $X = Y * Z$. Clearly X

is definite if Y and Z are such. But we cannot conclude that the definiteness of Y follows from the one of X and Z, as we need also the condition $Z \neq 0$. Similarly, we would like to conclude that X is definite if Y or Z have a zero value. It should then be clear how the results of the analysis we propose can be of help: by providing approximations of the concrete values of variables, something which is not captured by common definiteness analyses while being crucial when dealing with non-linear constraints.

5.3 Numbers as Leaves of Terms

For the purpose of bounds and relation analysis of CLP($\mathcal{H}_\mathcal{N}$) languages we need ways to approximate both numbers and multisets of numbers. The need for numbers is obvious: we want to represent and compute approximations of the values which numeric variables can take. While this would be enough for simple CLP($\mathcal{N}$) and CLP($\mathcal{H}, \mathcal{N}$) languages, this is not the case for CLP($\mathcal{H}_\mathcal{N}$) languages, where numerical values can occur as leaves of extended Herbrand terms.

An important observation is that the numerical properties we are about to introduce are *monotonic*. If, at some program point, the numerical leaves of terms that can be bound to a variable X are constrained to be, say, greater than 1, they will continue to do so as the computation progresses. An important consequence is that we can formalize concrete constraint stores as possibly infinite sets of tuples of ground terms. For example, the constraint store containing

$$\begin{aligned}
A &= \mathtt{c}(X, Y, Z),\\
B &= \mathtt{k}(\mathtt{s}(Cx, Cy, Cz, R)),\\
R * R &\geq (X - Cx) * (X - Cx) + (Y - Cy) * (Y - Cy)\\
&\qquad + (Z - Cz) * (Z - Cz),
\end{aligned}$$

and projected onto the pair of variables (A, B) is represented by the following infinite set of pairs of ground terms:

$$\left\{ \bigl(\mathtt{c}(x, y, z), \mathtt{k}(\mathtt{s}(c_x, c_y, c_z, r))\bigr) \;\middle|\; \begin{array}{l} x, y, z, c_x, c_y, c_z, r \in \mathbb{R} \\ r^2 \geq (x - c_x)^2 + (y - c_y)^2 \\ \qquad +(z - c_z)^2 \end{array} \right\}.$$

The overall analysis domain that is the subject of this chapter is of the kind Pattern(something), where Pattern($\cdot$) is the generic structural domain of Chapter 4. Describing that 'something' is the purpose of the following sections.

5.4 A Sequence of Approximations

We will now start describing how we abstract a set of tuples of terms for the purpose of range and relations analysis. This is done by setting up chain of subsequent approximations starting from the very concrete domain constituted by the complete lattice

$$\mathcal{D}^{\flat\!\flat} \overset{\mathrm{def}}{=} \wp(\mathcal{T}^m) \tag{5.10}$$

ordered by set inclusion.

First of all, we abstract each ground term with the *multiset* of its numerical leaves.[10]

Definition 74 (Numerical leaves.) *The multiset of* numerical leaves of a term *is given by the function* $\mathrm{nl}^+ \colon \mathcal{T} \to \wp_\mathrm{f}^+(\mathbb{R})$ *defined as follows, for each* $t \in \mathcal{T}$*:*

$$\mathrm{nl}^+(t) \overset{\mathrm{def}}{=} \begin{cases} \wr t \int, & \textit{if } t \textit{ is a number;} \\ \biguplus_{i=1}^{a} \mathrm{nl}^+(t_i), & \textit{if } t = f(t_1, \dots, t_a); \\ \varnothing, & \textit{otherwise.} \end{cases}$$

This brings us consider the intermediate domain

$$\mathcal{D}^\flat \overset{\mathrm{def}}{=} \wp\bigl(\wp_\mathrm{f}^+(\mathbb{R})^m\bigr), \tag{5.11}$$

still ordered by set inclusion, which is connected to $\mathcal{D}^{\flat\!\flat}$ by means of the abstraction function

$$\lambda \hat{T} \in \wp(\mathcal{T}^m) \, . \, \Bigl\{ \bigl(\mathrm{nl}^+(t_1), \dots, \mathrm{nl}^+(t_m)\bigr) \Bigm| (t_1, \dots, t_m) \in \hat{T} \Bigr\}. \tag{5.12}$$

The function (5.12) is a complete join-morphism by its very definition. Consequently, we have just defined a Galois connection between $\mathcal{D}^{\flat\!\flat}$ and $\mathcal{D}^\flat$.

We now face the problem of representing sets of m-tuples of finite multisets of real numbers. The first approximation which comes to mind consists in flattening the multisets and getting rid of the outermost powerset construction (an approximation often employed in abstract interpretation). This would lead to consider the domain

$$\mathcal{D}_\mathrm{q}^\natural \overset{\mathrm{def}}{=} \wp(\mathbb{R})^m, \tag{5.13}$$

which is an abstraction of $\mathcal{D}^\flat$ through the abstraction function $\alpha_\mathrm{q} \colon \mathcal{D}^\flat \to \mathcal{D}_\mathrm{q}^\natural$ given by

$$\lambda \hat{M} \in \mathcal{D}^\flat \, . \, \left(\bigcup_{\bar{M} \in \hat{M}} \zeta\bigl(\pi_1(\bar{M})\bigr), \dots, \bigcup_{\bar{M} \in \hat{M}} \zeta\bigl(\pi_m(\bar{M})\bigr) \right). \tag{5.14}$$

[10]The reader who has skipped the chapter on mathematical background might want to go back to Section 2.3 on page 18 where multiset notation is explained.

Of course, if we aim at a practical data-flow analysis, we would need to approximate $\wp(\mathbb{R})^m$ further: by using m-tuples of intervals, for instance. However, even the abstraction (5.14) alone implies an excessive loss of precision. For example, the set

$$\hat{E} \stackrel{\text{def}}{=} \Bigl\{ \bigl(\Lbag 1 \Rbag, \Lbag 2 \Rbag, \Lbag 3 \Rbag\bigr), \bigl(\Lbag 4 \Rbag, \Lbag 5 \Rbag, \Lbag 6 \Rbag\bigr) \Bigr\} \tag{5.15}$$

is abstracted by (5.14) into

$$\alpha_{\mathrm{q}}(\hat{E}) = \bigl(\{1,4\}, \{2,5\}, \{3,6\}\bigr), \tag{5.16}$$

which is also the abstraction of, say,

$$\Bigl\{ \bigl(\varnothing, \Lbag 5 \Rbag, \Lbag 3, 6 \Rbag\bigr), \bigl(\Lbag 1, 4 \Rbag, \Lbag 2 \Rbag, \varnothing\bigr) \Bigr\}.$$

We have thus lost information on the cardinalities of the multisets which form the tuples of $\hat{E}$, as well as relational information on these multisets. While it is true that each m-tuple $\bar{E} \in \hat{E}$ satisfies

$$\forall x \in \pi_1(\bar{E}) : \forall y \in \pi_2(\bar{E}) : \forall z \in \pi_3(\bar{E}) : x < y < z,$$

this cannot be concluded by looking at $\alpha_{\mathrm{q}}(\hat{E})$ only.

The above discussion is fairly obvious, of course, since information loss is the very essence of proper abstraction. The point is that we do not want to lose too much precision neither on the cardinalities of the multisets nor on the relationships which exist between the elements of each multiset. The reasons why these kinds of information are important for our application will become more clear later. However, they can be appreciated immediately by means of a couple of observations. First of all, qualitative constraints of the kind $X < Y$ arise frequently in constraint programming. They can be combined with quantitative constraints like, say, $X \geq 2$ in order to obtain new qualitative information: $Y > 2$ in the present example. So, relational information is important.

Then, suppose we have three sets of terms: T_1, T_2, and T_3. Suppose also we know that all the numerical leaves that occur in T_1 are numerically less than those of T_2, which, in turn, are less than those of T_3. Can we conclude that all the numerical leaves of T_1 are less than those of T_3? The answer is

"yes, *provided that T_2 has at least one numerical leaf*".

There are other possible inferences which are based on some *non-emptiness* condition. However, in order to capture the property of non-emptiness for multisets, non-emptiness alone is not strong enough. Suppose you have an unknown term

$$t \stackrel{\text{def}}{=} f(t_1, t_2),$$

where t_1 and t_2 are two further unknown terms. If you only know that t has a non-empty multiset of numerical leaves then all you can say is that t_1 *or* t_2 have at least one numerical leave. Stated differently, if you can reason in terms of non-emptiness only, nothing can be concluded: t_1 might not have any numerical leave, and the same can happen for t_2, though not at the same time. If cardinalities of the multisets can be expressed a more refined reasoning is possible, since the number of numerical leaves of t, u_1, and u_2 are tied by the formula

$$\bigl\|\mathrm{nl}^+(t)\bigr\| = \bigl\|\mathrm{nl}^+(t_1)\bigr\| + \bigl\|\mathrm{nl}^+(t_2)\bigr\|.$$

Similar situations occur frequently in abstract interpretation. For instance, if the *groundness* of variables has to inferred precisely one must use a strictly more powerful domain capturing *groundness dependencies*.

Stated that cardinalities and binary relations are important to us, we supplement $\mathcal{D}^\natural_\mathrm{q}$ with two further domains,

$$\mathcal{D}^\natural_\mathrm{c} \stackrel{\mathrm{def}}{=} \wp\bigl(\mathbb{N}\bigr)^m \tag{5.17}$$

and

$$\mathcal{D}^\natural_\mathrm{r} \stackrel{\mathrm{def}}{=} \wp\bigl(\mathbb{R}^2\bigr)^{m(m-1)/2}. \tag{5.18}$$

The abstraction $\alpha_\mathrm{c} \colon \mathcal{D}^\flat \to \mathcal{D}^\natural_\mathrm{c}$ keeps track of cardinalities. It is given by

$$\lambda \hat{M} \in \mathcal{D}^\flat \,.\, \biggl(\Bigl\{\, \bigl\|\pi_1(\bar{M})\bigr\| \Bigm| \bar{M} \in \hat{M} \,\Bigr\}, \dots, \Bigl\{\, \bigl\|\pi_m(\bar{M})\bigr\| \Bigm| \bar{M} \in \hat{M} \,\Bigr\}\biggr). \tag{5.19}$$

The additional domain $\mathcal{D}^\natural_\mathrm{r}$, instead, records the Cartesian product (a binary relation) between the values occurring in the i-th and the j-th multiset, for $i < j$. This restriction to the set of indexes

$$I_m \stackrel{\mathrm{def}}{=} \bigl\{\, (i,j) \bigm| i,j = 1,\dots,m, \quad i < j \,\bigr\} \tag{5.20}$$

is in order to avoid redundancies. Clearly, $S_i \times S_i$ does not convey more information that S_i itself, and for any two sets S_i and S_j we have $S_j \times S_i = (S_i \times S_j)^{-1}$. Observe that excluding useless redundancies allows us, among other things, to avoid the trouble of specifying consistency conditions among the redundant pieces. Once one has specified a bijection

$$(\tilde{\imath}, \tilde{\jmath}) \colon I_m \to \bigl\{1,\dots,m(m-1)/2\bigr\}, \tag{5.21}$$

such as the one given in Figure 5.6, the abstraction $\alpha_\mathrm{r} \colon \mathcal{D}^\flat \to \mathcal{D}^\natural_\mathrm{r}$ can be expressed as

$$\lambda \hat{M} \in \mathcal{D}^\flat \,.\, \Bigl(R\bigl(1, \hat{M}\bigr), \dots, R\bigl(m(m-1)/2, \hat{M}\bigr)\Bigr), \tag{5.22}$$

$$\begin{aligned}
\tilde{\jmath}(k) &\stackrel{\text{def}}{=} \left\lfloor \frac{\sqrt{8k-7}+1}{2} \right\rfloor + 1, && \text{for } k = 1, \ldots, \frac{m(m-1)}{2};\\
\tilde{\imath}(k) &\stackrel{\text{def}}{=} k - \frac{\bigl(\tilde{\jmath}(k)-1\bigr)\bigl(\tilde{\jmath}(k)-2\bigr)}{2}, && \text{for } k = 1, \ldots, \frac{m(m-1)}{2};\\
(\tilde{\imath},\tilde{\jmath})^{-1}(i,j) &\stackrel{\text{def}}{=} i + \frac{(j-1)(j-2)}{2}, && \text{for } (i,j) \in I_m.
\end{aligned}$$

Figure 5.6: A bijection $(\tilde{\imath},\tilde{\jmath})\colon I_m \to \bigl\{1, \ldots, m(m-1)/2\bigr\}$.

where, for each $k = 1, \ldots, m(m-1)/2$,

$$R(k, \hat{M}) \stackrel{\text{def}}{=} \bigcup_{\bar{M} \in \hat{M}} \zeta\bigl(\pi_{\tilde{\imath}(k)}(\bar{M})\bigr) \times \zeta\bigl(\pi_{\tilde{\jmath}(k)}(\bar{M})\bigr), \tag{5.23}$$

Now, our first approximation of $\mathcal{D}^\flat$ is given by the product of $\mathcal{D}^\natural_{\mathrm{c}}$, $\mathcal{D}^\natural_{\mathrm{q}}$, and $\mathcal{D}^\natural_{\mathrm{r}}$, namely

$$\mathcal{D}^\natural \stackrel{\text{def}}{=} \mathcal{D}^\natural_{\mathrm{c}} \times \mathcal{D}^\natural_{\mathrm{q}} \times \mathcal{D}^\natural_{\mathrm{r}}. \tag{5.24}$$

This is clearly more precise than $\mathcal{D}^\natural_{\mathrm{q}}$ alone. The domain $\mathcal{D}^\natural$ includes some redundancy, since many of its elements represent the same element of $\mathcal{D}^\flat$. This situation could be rectified by considering, instead of the straight product, the *reduced product* of $\mathcal{D}^\natural_{\mathrm{c}}$, $\mathcal{D}^\natural_{\mathrm{q}}$, and $\mathcal{D}^\natural_{\mathrm{r}}$ [CC79, CC92a]. We will not do that, however, since we proceed by performing a further approximation step on $\mathcal{D}^\natural$.

Actually, we consider a family of such approximations. Each member of the family is obtained by providing separate approximations for $\mathcal{D}^\natural_{\mathrm{c}}$, $\mathcal{D}^\natural_{\mathrm{q}}$, and $\mathcal{D}^\natural_{\mathrm{r}}$. The considered family of abstract domains is thus given by any

$$\mathcal{D}^\sharp(\mathsf{A}_{\mathbb{R}}, \mathsf{A}_{\mathbb{N}}, \mathsf{A}_{\mathbb{R}^2}) \stackrel{\text{def}}{=} \mathsf{A}_{\mathbb{N}}^m \times \mathsf{A}_{\mathbb{R}}^m \times \mathsf{A}_{\mathbb{R}^2}^{m(m-1)/2}, \tag{5.25}$$

where $\mathsf{A}_{\mathbb{R}}$, $\mathsf{A}_{\mathbb{N}}$, and $\mathsf{A}_{\mathbb{R}^2}$ are approximations of $\wp(\mathbb{N})$, $\wp(\mathbb{R})$, and $\wp(\mathbb{R}^2)$, respectively. We will show later how we can approximate the reduction of the product (5.25), thus obtaining an approximation of the reduced product of $\mathcal{D}^\natural_{\mathrm{c}}$, $\mathcal{D}^\natural_{\mathrm{q}}$, and $\mathcal{D}^\natural_{\mathrm{r}}$.

The next three sections are devoted to the introduction of the classes of approximations we consider for the representation of sets of cardinalities, sets of real numbers, and binary relations between such sets. We will give the properties they must satisfy for our purposes together with some representative examples.

5.5 Approximations for Sets of Reals

This first kind of approximation conveys *quantitative information* on the values that numerical leaves can take. Here is a definition that, adopting the closure operator approach to abstract interpretation, spells out precisely what we require for this purpose.

Definition 75 ($\mathbb{R}$-approximation.) *A* $\mathbb{R}$-approximation *is any algebra of the form*[11]

$$\dot{\mathsf{A}}_{\mathbb{R}} \stackrel{\text{def}}{=} \langle \mathsf{A}_{\mathbb{R}}, \boxplus, \boxminus_1, \boxminus_2, \boxast, \boxslash, \boxed{\sin}, \dots \rangle,$$

where

1. $\mathsf{A}_{\mathbb{R}} \subseteq \wp(\mathbb{R})$ *is an abstraction of* $\wp(\mathbb{R})$*, namely, there exists an operator*[12] $\mathsf{A}_{\mathbb{R}} \in \mathrm{uco}\bigl(\wp(\mathbb{R}), \subseteq\bigr)$ *such that* $\mathsf{A}_{\mathbb{R}} = \mathsf{A}_{\mathbb{R}}\bigl(\wp(\mathbb{R})\bigr)$*. Moreover,* $\varnothing \in \mathsf{A}_{\mathbb{R}}$*.*

2. $\boxplus$, $\boxminus_1$, $\boxminus_2$, $\boxast$, $\boxslash$, $\boxed{\sin}$, … *are functions in* $\mathsf{A}_{\mathbb{R}}^k \to \mathsf{A}_{\mathbb{R}}$*, for some* $m \in \mathbb{N}$*, approximating the linear extensions of their obvious (non-boxed) counterparts in* $\mathbb{R}^k \to \mathbb{R}$*. For instance,* $\boxplus \colon \mathsf{A}_{\mathbb{R}}^2 \to \mathsf{A}_{\mathbb{R}}$ *is such that, for each* $A_1, A_2 \in \mathsf{A}_{\mathbb{R}}$*,*

$$\{\, x_1 + x_2 \mid x_1 \in A_1, x_2 \in A_2 \,\} \subseteq A_1 \boxplus A_2.$$

We will use the symbols $\otimes$ *and* $\oplus$ *to denote the* glb *(set intersection) and the* lub *operators, respectively, of the complete lattice* $(\mathsf{A}_{\mathbb{R}}, \subseteq)$*.*

The reader who wonders why we have not defined, for instance,

$$A_1 \boxplus A_2 \stackrel{\text{def}}{=} \mathsf{A}_{\mathbb{R}}\bigl(\{\, x_1 + x_2 \mid x_1 \in A_1, x_2 \in A_2 \,\}\bigr),$$

is referred to the next section, where we will briefly review a well-known $\mathbb{R}$-approximation.

Sometimes one needs approximations for subsets of the extended reals $\mathbb{R}_\infty \stackrel{\text{def}}{=} \mathbb{R} \cup \{-\infty, +\infty\}$, instead of just the reals. This can happen because the language to be analyzed employs IEEE-754 and/or IEEE-854 extended arithmetic[13]. Some extra complications arise (especially in the implementation) when dealing with the extended reals.

Another reason for dealing with the extended reals is when one of the results which are sought from the analysis is *possible overflow* information. In this case the values $+\infty$ and $-\infty$ indicate a potential arithmetic overflow in the computation.

Actually, we are about to introduce $\mathbb{R}_\infty$*-approximations* because we need them in order to define an $\mathbb{R}^2$-approximation called *bounded quotients* (see Section 5.7.3 on page 139).

[11] The subscripts of $\boxminus_1$ and $\boxminus_2$ denote the arity: they are the approximations of unary and binary minus, respectively.

[12] Notice how we overload $\mathsf{A}_{\mathbb{R}}$ to denote both the approximation's domain and the upper closure operator that maps each subset of the reals into the least element of the domain that contains it. Thus, $\mathsf{A}_{\mathbb{R}} \colon \mathbb{R} \to \mathsf{A}_{\mathbb{R}}$ is a sort of *constructor* for $\mathsf{A}_{\mathbb{R}} \subseteq \wp(\mathbb{R})$. We will adopt the same convention for all the approximations that follow.

[13] SICStus Prolog implements such a language [SIC95].

Definition 76 ($\mathbb{R}_\infty$-approximation.) *A* $\mathbb{R}_\infty$-approximation *is an algebra of the form*

$$\dot{\mathsf{A}}_{\mathbb{R}_\infty} \stackrel{\mathrm{def}}{=} \langle \mathsf{A}_{\mathbb{R}_\infty}, \boxplus, \boxminus_1, \boxminus_2, \circledast, \boxslash, \boxed{\sin}, \ldots \rangle,$$

where

1. $\mathsf{A}_{\mathbb{R}_\infty} \subseteq \wp(\mathbb{R}_\infty)$*, with* $\varnothing \in \mathsf{A}_{\mathbb{R}_\infty}$*, is an abstraction of* $\wp(\mathbb{R}_\infty)$ *through the operator* $\mathsf{A}_{\mathbb{R}_\infty} \in \mathrm{uco}\bigl(\wp(\mathbb{R}_\infty), \subseteq\bigr)$.
2. $\boxplus$, $\boxminus_1$, $\boxminus_2$, $\circledast$, $\boxslash$, $\boxed{\sin}$, … *are functions in* $\mathsf{A}_{\mathbb{R}_\infty}^k \to \mathsf{A}_{\mathbb{R}_\infty}$ *approximating the linear extensions of their non-boxed counterparts in* $\mathbb{R}_\infty^k \to \mathbb{R}_\infty$.
3. *Moreover, all the above functions satisfy the following property: for each* $\boxed{\mathrm{op}}\colon \mathsf{A}_{\mathbb{R}_\infty}^k \to \mathsf{A}_{\mathbb{R}_\infty}$ *and each* $A_1, \ldots, A_k \in \mathsf{A}_{\mathbb{R}_\infty}$*, if it happens that* $+\infty \in \boxed{\mathrm{op}}(A_1, \ldots, A_k)$ *then there must exist* $x_1 \in A_1$, … , $x_k \in A_k$ *such that* $\mathrm{op}(x_1, \ldots, x_k) = +\infty$*. Likewise for* $-\infty$.

The same notational conventions introduced for $\mathbb{R}$*-approximations will be used.*

Note that condition 3 above imposes a limit on the crudeness of the approximate operators. In words, the extra-symbols $+\infty$ and $-\infty$ cannot be thrown in the results without reasons, but only when the linear extension of the approximated operation would have produced them. As we will see, this requirement imposes some extra care in the actual definition and implementation of a $\mathbb{R}_\infty$-approximation.

5.5.1 Intervals

One commonly used $\mathbb{R}$-approximation is constituted by intervals. They have several advantages [AH83, Cle87, Moo66, Dav87]:

1. they have a compact representation. Usually two numbers drawn from some subset of $\mathbb{R} \cup \{-\infty, +\infty\}$ are enough, plus possibly two booleans to indicate closedness or openness at each end.
2. Intervals are closed under intersection; and
3. the image of an interval under a continuous function is itself an interval.

Definition 77 (Open/closed intervals.) *Let* $\mathsf{B} \subseteq \mathbb{R}$ *be any set of real numbers. The set* B_∞ *of* boundaries *over* B *is given by*

$$\mathsf{B}_\infty \stackrel{\mathrm{def}}{=} \mathsf{B} \cup \{-\infty, +\infty\},$$

and is totally ordered by the natural extension of $<$ *over* B_∞*. Consider the set*

$$\mathsf{I}_\mathsf{B} \stackrel{\text{def}}{=} \bigl\{\, \{ x \in \mathbb{R} \mid l \bowtie_l x \bowtie_u u \} \bigm| l, r \in \mathsf{B}_\infty, \quad \bowtie_l, \bowtie_r \in \{<, \leq\} \,\bigr\}.$$

The $\otimes$ *operator over* I_B *is set intersection, while* $\oplus$ *is given by the convex hull, that is, for each* $I_1, I_2 \in \mathsf{I}_\mathsf{B}$*,*

$$I_1 \oplus I_2 \stackrel{\text{def}}{=} \bigl\{ x \in \mathbb{R} \bigm| \exists l, u \in I_1 \cup I_2 \,.\, l \leq x \leq u \bigr\}.$$

By endowing I_B *with a family of arithmetic operators as required by* Definition 75 *we obtain a* $\mathbb{R}$*-approximation. We will denote it by* $\dot{\mathsf{I}}_\mathsf{B}$ *and call it an* interval approximation over B*.*

Suppose that $\{0, 1\} \subseteq \mathsf{B}$ and that $-x \in \mathsf{B}$ whenever $x \in \mathsf{B}$. Then it is possible to define the $\boxplus$ and $\circledast$ operations so that I_B is a commutative ringoid with identity. Namely [KM81], for each $I, I_1, I_2 \in \mathsf{I}_\mathsf{B}$:

$$\begin{aligned} I_1 \boxplus I_2 &= I_2 \boxplus I_1, & I \boxplus \{0\} &= I, \\ I_1 \boxast I_2 &= I_2 \boxast I_1, & I \boxast \{1\} &= I, \\ I \boxast \{0\} &= \{0\}, \end{aligned}$$

$\{-1\}$ is the unique interval which is the additive inverse of $\{1\}$, the multiplicative inverse of itself, and distributes with respect to $\boxplus$ and $\boxast$. In case B is unbounded, $\boxplus$ and $\boxast$ can be made to be associative. Distributivity, however, does not hold, even in the extreme case where B is $\mathbb{R}$ itself.

In practice, one chooses B to be a computable set of numbers, e.g., some family of rational or floating point numbers. Rational numbers would allow exact arithmetic computations to be performed, assuming the availability of arbitrarily large integers. Exact computations, however, have the disadvantage that the gcd operations, while taking time, cannot prevent the formation of huge numerators and denumerators in the boundaries' representations. As usual, one can trade precision for efficiency by considering only interval boundaries with acceptably compact ratios [EF92].

Of course, one can consider open intervals only or closed intervals only. Some authors advocate this last possibility for the following reasons [Dav87]:

1. any bounded set of values admits a unique minimal closed interval which contains it;
2. numbers are a special case of closed interval;
3. the image of a closed bounded interval under a continuous function is a bounded closed interval.

These considerations, unfortunately, have little value in practice. The reason is that our boundary values must be machine representable. For instance, if we employ single-precision IEEE floating point numbers, we can represent the bounded set of values $\{2\}$ as $[2, 2]$, but the best representation we can get for $\{\sqrt{2}\}$, the image of $[2, 2]$ under the continuous function $\sqrt{\cdot} : \mathbb{R}^+ \to \mathbb{R}^+$, is something like $(1.414213538\cdots, 1.414213657\cdots)$. Moreover, the set of strictly positive real numbers is not representable by means of a closed interval. In summary, we believe that considering a mixture of open, closed, and half-open intervals is generally a good idea.

A useful variation on the theme, which we will exploit later, is to consider a mixture of real and integer-valued intervals.

Definition 78 (Real/integer intervals.) *Let* $\dot{\mathsf{I}}_{\mathsf{B}}$ *be an interval approximation over* B*. The* real/integer interval approximation over I_{B} *is given by the carrier*

$$\mathsf{I}_{\mathsf{B}}^{\mathbb{N}} \overset{\text{def}}{=} \mathsf{I}_{\mathsf{B}} \cup \{\, I \cap \mathbb{N} \mid I \in \mathsf{I}_{\mathsf{B}} \,\}$$

augmented with all the operators of I_{B}*, suitably extended to* $\mathsf{I}_{\mathsf{B}}^{\mathbb{N}}$ *so to satisfy the requirements of* Definition 75*. We will use the notation* $[l \mathinner{..} u]$*, with* $l, u \in \mathsf{B} \cap \mathbb{N}$ *for the finite sets in* $\mathsf{I}_{\mathsf{B}}^{\mathbb{N}}$*. The remaining elements of* $\{\, I \cap \mathbb{N} \mid I \in \mathsf{I}_{\mathsf{B}} \,\}$ *will be denoted by* $(-\infty \mathinner{..} u]$*,* $[l \mathinner{..} +\infty)$*, and* $(-\infty \mathinner{..} +\infty)$*.*

The mathematically inclined might be surprised (or upset) by the fact that we use the word "interval" to denote also non-convex subsets of the reals: we simply find this terminology convenient.

We would now like to give an example explaining why, in Definition 75, we allow for some crudeness on the part of the arithmetic operators. Consider division in $\dot{\mathsf{I}}_{\mathsf{B}}^{\mathbb{N}}$. Defining $\boxslash$ in the most precise way, namely as $\mathsf{I}_{\mathsf{B}}^{\mathbb{N}} \circ (\cdot/\cdot)$, we would be able to obtain

$$[6 \mathinner{..} 6] \boxslash [1 \mathinner{..} 3] \overset{\text{def}}{=} \mathsf{I}_{\mathsf{B}}^{\mathbb{N}}\Bigl(\bigl\{\, x/y \bigm| x \in [6 \mathinner{..} 6], y \in [1 \mathinner{..} 3] \,\bigr\}\Bigr) = [2 \mathinner{..} 6]. \tag{5.26}$$

However, one normally uses a more efficient division operator, which avoids the divisibility checks at the price of a small precision penalty. Such an operator would yield the cruder result $[6 \mathinner{..} 6] \boxslash [1 \mathinner{..} 3] = [2, 6]$, for instance. So we do not want to make any *a priori* commitment to maximum precision. Nonetheless, we might do wish to be precise. This point gives us an opportunity for a brief digression.

The abstract interpretation practitioner might have protested because we did not present the real/integer interval approximation as a reduced product. Indeed, the carrier of $\dot{\mathsf{I}}_{\mathsf{B}}^{\mathbb{N}}$ can be obtained as the reduced product between an interval domain and a domain for *integrality* with two elements: '*integer*' and '*don't know*'. But what about the arithmetic operations? Following the reduced product approach, the division operator cannot obtain the same result of (5.26). In fact, $[6,6]$ divided by $[1,3]$ gives $[2,6]$, *integer* divided by *integer* yields *don't know*, and there is no way for $[2,6]$ and *don't know* to be reduced to $[2\,..\,6]$. This limitation is undesirable, since one might well want to implement (5.26). In summary, our choices (both here and in general) are justified by the desire of retaining maximum freedom in the actual implementations. In this respect, we condemn both excessive commitment to precision and to sloppiness equally.

Of course, the examples of $\mathbb{R}$-approximations we have shown are not exhaustive. One can consider some family of sets of intervals, also called *multi-intervals*, for instance. The theory of abstract interpretation offers a great variety of possibilities for designing the carrier of the approximations, starting from some base domains, through standard constructions like reduced product, down-set completion, auto-dependencies and so forth. This remark clearly applies also to the approximations for cardinalities and binary relations we are about to present.

5.6 Approximations for Cardinalities

For the purpose of representing multiset cardinalities any approximation of $\wp(\mathbb{N})$ satisfying some minimal requisites would do.

Definition 79 ($\mathbb{N}$-approximation.) *A* $\mathbb{N}$-approximation *is any algebra of the form*

$$\check{\mathsf{A}}_{\mathbb{N}} \stackrel{\text{def}}{=} \langle \mathsf{A}_{\mathbb{N}}, \boxplus, \boxminus_2 \rangle,$$

where

1. $\mathsf{A}_{\mathbb{N}} \subseteq \wp(\mathbb{N})$ *is an abstraction of* $\wp(\mathbb{N})$, *the corresponding upper closure operator being* $\mathsf{A}_{\mathbb{N}} \in \operatorname{uco}\bigl(\wp(\mathbb{N}), \subseteq\bigr)$. *Moreover,* $\mathsf{A}_{\mathbb{N}}$ *contains* $\varnothing$, $\{0\}$, *and* $\mathbb{N} \setminus \{0\}$.

2. $\boxplus$ *and* $\boxminus_2$ *are correct approximations of the linear extensions of* $+, -\colon \mathbb{N}^2 \to \mathbb{N}$.

Again, we will use the symbols $\otimes$ *and* $\oplus$ *to denote the* glb *(set intersection) and the* lub *operators, respectively, of the complete lattice* $\mathsf{A}_{\mathbb{N}}$.

5.6.1 An Example

As an example of $\mathbb{N}$-approximation we introduce now a simple domain of intermediate precision between *signs* and *integer intervals*. In lack of a better name, we will refer to this trivial exercise in abstract interpretation as the *cardinalities approximation*.

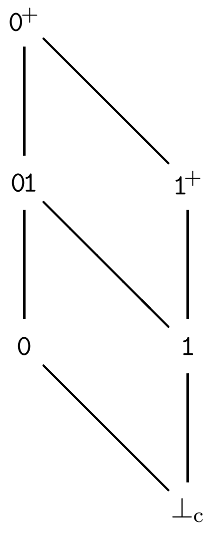


Figure 5.7: Hasse diagram of the cardinalities' lattice.

Definition 80 (Cardinalities.) *The* cardinality approximation *is given by* $\big\langle\{\bot_c, \mathsf{0}, \mathsf{1}, \mathsf{01}, \mathsf{1}^+, \mathsf{0}^+\}, \boxplus, \boxminus_2\big\rangle$, *where*

$$\bot_c \stackrel{\text{def}}{=} \varnothing, \qquad \mathsf{0} \stackrel{\text{def}}{=} \{0\}, \qquad \mathsf{1} \stackrel{\text{def}}{=} \{1\},$$
$$\mathsf{01} \stackrel{\text{def}}{=} \{0, 1\}, \qquad \mathsf{1}^+ \stackrel{\text{def}}{=} \mathbb{N} \setminus \{0\}, \qquad \mathsf{0}^+ \stackrel{\text{def}}{=} \mathbb{N}.$$

The complete lattice which arises from this definition is depicted in Figure 5.7, whereas the $\boxplus$ *and* $\boxminus_2$ *operators are summarized in Tables 5.1 and 5.2 on page 137.*

5.7 Approximations for Numerical Relations

The approximations of relations carry *qualitative information* on numerical values. Since there is an interplay between qualitative and quantitative information, our approximations of relations are parametric with respect to a $\mathbb{R}$-approximation. Besides that the following definition is quite similar to the ones we have already seen, the key principle (sound approximation) being exactly the same.

The operations we need to approximate here are for composing, building, and restricting (or applying) binary relations.

Definition 81 ($\mathbb{R}^2$-approximation.) *Let* $\mathsf{A}_\mathbb{R}$ *be any a set approximation. A* $\mathbb{R}^2$-approximation *is any algebra of the form*

$$\ddot{\mathsf{A}}_{\mathbb{R}^2} \stackrel{\text{def}}{=} \langle \mathsf{A}_{\mathbb{R}^2}, \mathsf{A}_\mathbb{R}, \boxdot, \boxtimes, \boxbar \rangle,$$

where

1. $\mathsf{A}_{\mathbb{R}^2} \subseteq \wp(\mathbb{R}^2)$ *is an abstraction of* $\wp(\mathbb{R}^2)$ *such that* $\varnothing \in \mathsf{A}_{\mathbb{R}^2}$*. Moreover,* $R^{-1} \in \mathsf{A}_{\mathbb{R}^2}$ *whenever* $R \in \mathsf{A}_{\mathbb{R}^2}$*. The associated upper closure operator will be denoted by* $\mathsf{A}_{\mathbb{R}^2}$*.*

2. $\boxtimes \colon \mathsf{A}_{\mathbb{R}} \times \mathsf{A}_{\mathbb{R}} \to \mathsf{A}_{\mathbb{R}^2}$ *is such that, for each* $A_1, A_2 \in \mathsf{A}_{\mathbb{R}}$*,*
$$A_1 \boxtimes A_2 \supseteq A_1 \times A_2.$$

3. $\boxdot \colon \mathsf{A}_{\mathbb{R}^2} \times \mathsf{A}_{\mathbb{R}^2} \to \mathsf{A}_{\mathbb{R}^2}$ *is such that, for each* $R_1, R_2 \in \mathsf{A}_{\mathbb{R}^2}$*,*
$$R_1 \boxdot R_2 \supseteq R_1 \circ R_2.$$

4. $\boxbar \colon \mathsf{A}_{\mathbb{R}^2} \times \mathsf{A}_{\mathbb{R}} \to \mathsf{A}_{\mathbb{R}}$ *is such that, for each* $R \in \mathsf{A}_{\mathbb{R}^2}$ *and each* $A \in \mathsf{A}_{\mathbb{R}}$*,*
$$R \boxbar A \supseteq R(A).$$

The $\boxbar$ *operation is called* refinement*. Again, we will use the symbols* $\otimes$ *and* $\oplus$ *to denote the* glb *(set intersection) and the* lub *operators, respectively, of the complete lattice* $\mathsf{A}_{\mathbb{R}^2}$*.*

We now show three representative classes of $\mathbb{R}^2$-approximation. As usual, many other alternatives exist.

5.7.1 Ordering Relationships

One of the simplest form of $\mathbb{R}^2$-approximation is constituted by *ordering relationships.*

Definition 82 (Ordering relationships.) *The set*
$$\mathsf{O} \stackrel{\mathrm{def}}{=} \{\varnothing, =, <, >, \leq, \geq, \neq, \mathbb{R}^2\},$$
where
$$\begin{aligned}
= &\stackrel{\mathrm{def}}{=} \{\, (x,y) \in \mathbb{R}^2 \mid x = y \,\}, & \neq &\stackrel{\mathrm{def}}{=} \{\, (x,y) \in \mathbb{R}^2 \mid x \neq y \,\},\\
< &\stackrel{\mathrm{def}}{=} \{\, (x,y) \in \mathbb{R}^2 \mid x < y \,\}, & \leq &\stackrel{\mathrm{def}}{=} \{\, (x,y) \in \mathbb{R}^2 \mid x \leq y \,\},\\
> &\stackrel{\mathrm{def}}{=} \{\, (x,y) \in \mathbb{R}^2 \mid x > y \,\}, & \geq &\stackrel{\mathrm{def}}{=} \{\, (x,y) \in \mathbb{R}^2 \mid x \geq y \,\},
\end{aligned}$$
is the carrier of a family of $\mathbb{R}^2$*-approximations which we will collectively refer to as* ordering relationships*. The* $\otimes$ *and* $\oplus$ *operators are given by set-theoretic intersection and union, respectively. Thus,* O *has the simple lattice structure depicted in* Figure 5.8*. Let* $\dot{\mathsf{A}}_{\mathbb{R}}$ *be any* $\mathbb{R}$*-approximation. The* $\boxtimes$ *and* $\boxbar$ *operators (which depend on* $\dot{\mathsf{A}}_{\mathbb{R}}$*) are chosen freely, as long as the requirements of* Definition 81 *are satisfied. The most precise composition operator, namely* $\mathsf{O} \circ (\cdot \circ \cdot)$*, is so simple that it constitutes a compulsory choice. The resulting* $\boxdot$ *operator is shown in* Table 5.3*. Any* $\mathbb{R}^2$*-approximation so obtained will be denoted by* $\ddot{\mathsf{O}}(\dot{\mathsf{A}}_{\mathbb{R}})$ *and called* ordering relationships over $\dot{\mathsf{A}}_{\mathbb{R}}$*.*

$\boxplus$	$\bot_c$	0	1	01	1^+	0^+
$\bot_c$	$\bot_c$	$\bot_c$	$\bot_c$	$\bot_c$	$\bot_c$	$\bot_c$
0	$\bot_c$	0	1	01	1^+	0^+
1	$\bot_c$	1	1^+	1^+	1^+	1^+
01	$\bot_c$	01	1^+	0^+	1^+	0^+
1^+	$\bot_c$	1^+	1^+	1^+	1^+	1^+
0^+	$\bot_c$	0^+	1^+	0^+	1^+	0^+

Table 5.1: The $\boxplus$ operator on cardinalities.

$\boxminus_2$	$\bot_c$	0	1	01	1^+	0^+
$\bot_c$	$\bot_c$	$\bot_c$	$\bot_c$	$\bot_c$	$\bot_c$	$\bot_c$
0	$\bot_c$	0	$\bot_c$	0	$\bot_c$	0
1	$\bot_c$	1	0	01	0	01
01	$\bot_c$	01	0	01	0	01
1^+	$\bot_c$	1^+	0^+	0^+	0^+	0^+
0^+	$\bot_c$	0^+	0^+	0^+	0^+	0^+

Table 5.2: The $\boxminus_2$ operator on cardinalities: $x \boxminus y$ is at the intersection between the x-th row and the y-th column.

$$\begin{tikzcd}
 & \mathbb{R}^2 \arrow[dl, no head] \arrow[d, no head] \arrow[dr, no head] & \\
\leq \arrow[d, no head] \arrow[dr, no head] & \neq \arrow[dl, no head] \arrow[dr, no head] & \geq \arrow[dl, no head] \arrow[d, no head] \\
< \arrow[dr, no head] & = \arrow[d, no head] & > \arrow[dl, no head] \\
 & \varnothing &
\end{tikzcd}$$

Figure 5.8: Hasse diagram of the ordering relationships lattice.

$\boxdot$	$<$	$\leq$	$>$	$\geq$	$=$	$\neq$
$<$	$<$	$<$	$\mathbb{R}^2$	$\mathbb{R}^2$	$<$	$\mathbb{R}^2$
$\leq$	$<$	$\leq$	$\mathbb{R}^2$	$\mathbb{R}^2$	$\leq$	$\mathbb{R}^2$
$>$	$\mathbb{R}^2$	$\mathbb{R}^2$	$>$	$>$	$>$	$\mathbb{R}^2$
$\geq$	$\mathbb{R}^2$	$\mathbb{R}^2$	$>$	$\geq$	$\geq$	$\mathbb{R}^2$
$=$	$<$	$\leq$	$>$	$\geq$	$=$	$\neq$
$\neq$	$\mathbb{R}^2$	$\mathbb{R}^2$	$\mathbb{R}^2$	$\mathbb{R}^2$	$\neq$	$\mathbb{R}^2$

Table 5.3: The $\boxdot$ operator on ordering relationships.

5.7.2 Bounded Differences

The next $\mathbb{R}^2$-approximation we consider comes (as it is the case for many other things in this chapter) from the field of Artificial Intelligence. It allows to express the relative values of two quantities by means of constraints of the form $x - y \in S$, with $S \subseteq \mathbb{R}$, whence the name *bounded differences* [Dav87]. In AI these approximations proved to be particularly useful for systems which need to place events on a time-line, such as TMM [Dea85] and other planning programs [Ver83]. In our particular formal treatment, any $\mathbb{R}$-approximation induces a system of bounded differences.

Definition 83 (Bounded differences.) *Let $\dot{\mathsf{A}}_{\mathbb{R}}$ be an $\mathbb{R}$-approximation. Consider, for each $A \in \mathsf{A}_{\mathbb{R}}$, the set*

$$\mathsf{d}(A) \stackrel{\text{def}}{=} \bigl\{\, (x, y) \in \mathbb{R}^2 \bigm| (x - y) \in A \,\bigr\},$$

and then

$$\mathsf{D}(\mathsf{A}_{\mathbb{R}}) \stackrel{\text{def}}{=} \bigl\{\, \mathsf{d}(A) \bigm| A \in \mathsf{A}_{\mathbb{R}} \,\bigr\}.$$

By construction, $\mathsf{A}_{\mathbb{R}}$ is isomorphic to $\mathsf{D}(\mathsf{A}_{\mathbb{R}})$, the isomorphism being given by $\mathsf{d}\colon \mathsf{A}_{\mathbb{R}} \to \mathsf{D}(\mathsf{A}_{\mathbb{R}})$. In particular, $\mathsf{d}(\varnothing) = \varnothing$ and $\mathsf{d}(\mathbb{R}) = \mathbb{R}^2$. Moreover, several operations over bounded differences are defined in terms of the operations of $\dot{\mathsf{A}}_{\mathbb{R}}$. Thus, for each $A_1, A_2 \in \mathsf{A}_{\mathbb{R}}$:

$$\begin{aligned}
\mathsf{d}(A_1) \otimes \mathsf{d}(A_2) &\stackrel{\text{def}}{=} \mathsf{d}(A_1 \otimes A_2),\\
\mathsf{d}(A_1) \oplus \mathsf{d}(A_2) &\stackrel{\text{def}}{=} \mathsf{d}(A_1 \oplus A_2),\\
\mathsf{d}(A_1) \boxdot \mathsf{d}(A_2) &\stackrel{\text{def}}{=} \mathsf{d}(A_1 \boxplus A_2),\\
A_1 \boxtimes A_2 &\stackrel{\text{def}}{=} \mathsf{d}(A_1 \boxminus_2 A_2),
\end{aligned}$$

If $\boxbar$ is a refinement operator satisfying Definition 81*, it can be shown that*

$$\ddot{\mathsf{D}}(\dot{\mathsf{A}}_{\mathbb{R}}) \stackrel{\text{def}}{=} \bigl\langle \mathsf{D}(\mathsf{A}_{\mathbb{R}}), \mathsf{A}_{\mathbb{R}}, \boxdot, \boxtimes, \boxbar \bigr\rangle$$

is indeed an $\mathbb{R}^2$*-approximation. We will call such an approximation* bounded differences over $\dot{\mathsf{A}}_{\mathbb{R}}$.

Observe that a plausible definitions of the refinement operator, in terms of $\mathsf{A}_{\mathbb{R}}$ operators, is the following:

$$\mathsf{d}(A_1) \boxbar A_2 \overset{\text{def}}{=} A_2 \boxminus_2 A_1,$$

where $A_1, A_2 \in \mathsf{A}_{\mathbb{R}}$. However, generally speaking, this does not automatically define neither the most precise refinement operator, nor an efficient one. Of course, the more precise is the $\mathbb{R}$-approximation which is used as the basis, the more precise is the resulting system of bounded differences. Observe also that Definition 83 is somewhat unnecessarily restrictive, in that it assumes that one uses the same $\mathbb{R}$-approximation both *per se* (namely, as a way for approximating sets of values) and as the basis for the bounded differences construction. Of course, one can make a different choice: for example using intervals for bounded differences together with, say, some family of multi-intervals.

As a final remark, notice that any conventional interval approximation, such as the one introduced in Definition 77, is not strong enough in order to obtain, through the bounded-differences construction, an $\mathbb{R}^2$-approximation capturing all the ordering relationships of $\ddot{\mathsf{O}}$. In fact, while any decent interval approximation $\dot{\mathsf{I}}_{\mathsf{B}}$ is such that $\mathsf{D}(\dot{\mathsf{I}}_{\mathsf{B}})$ can represent $=$, $<$, $>$, $\leq$, and $\geq$ (by means of $[0,0]$, $(-\infty,0)$, $(0,+\infty)$, $(-\infty,0]$, and $[0,+\infty)$, respectively), it cannot represent $\neq$. Indeed, $\neq$ would correspond to $(-\infty,0) \cup (0,+\infty)$, which usually does not belong to I_{B}. The solution is to adopt an $\mathbb{R}$-approximation which is able to capture constraints of the kind $x \neq 0$. A suitable variation on the *intervals theme* is given by

$$\mathsf{I}_{\mathsf{B}}^{\neq 0} \overset{\text{def}}{=} \mathsf{I}_{\mathsf{B}} \cup \bigl\{\, I \setminus \{0\} \bigm| I \in \mathsf{I}_{\mathsf{B}} \,\bigr\}, \tag{5.27}$$

which, if endowed with reasonable $\boxplus$ and $\boxminus_2$ operators, is such that $\ddot{\mathsf{D}}\bigl(\dot{\mathsf{I}}_{\mathsf{B}}^{\neq 0}\bigr)$ is strictly more precise than $\ddot{\mathsf{O}}$, that is, $\mathsf{O} \subset \mathsf{D}\bigl(\mathsf{I}_{\mathsf{B}}^{\neq 0}\bigr)$.

5.7.3 Bounded Quotients

Consider the extended reals $\mathbb{R}_\infty \overset{\text{def}}{=} \mathbb{R} \cup \{-\infty, +\infty\}$, with the arithmetic operations extended in such a way that, for each $b \in \mathbb{R}$ with $b > 1$,

$$\begin{aligned}
\frac{0}{0} &\overset{\text{def}}{=} 1, & \frac{x}{0} &\overset{\text{def}}{=} +\infty, \quad \text{for each } x > 0,\\
\log_b 0 &\overset{\text{def}}{=} -\infty, & \log_b(+\infty) &\overset{\text{def}}{=} +\infty,\\
(+\infty) - (+\infty) &\overset{\text{def}}{=} 0, & (-\infty) - (-\infty) &\overset{\text{def}}{=} 0.
\end{aligned}$$

This is what is needed to ensure that, for each $x, y \in \mathbb{R}_\infty$,

$$\log_b \left|\frac{x}{y}\right| = \log_b|x| - \log_b|y|.$$

Once one has fixed a base $b > 1$, a system of *bounded quotients* captures relations of the form $\log_b|x| - \log_b|y| \in S$. As such, it is quite similar to a system of bounded differences. In the field of Artificial Intelligence, systems of bounded differences, restricted to positive quantities, have been used for temporal reasoning [AK85]. In our case quantities can be negative, whence the need for taking the absolute value, or can be zero, which is why we need to consider the extended structure of the reals.

Definition 84 (Bounded quotients.) *Let us fix $b \in \mathbb{R}$ such that $b > 1$, and let $\dot{\mathsf{A}}_{\mathbb{R}_\infty}$ and $\dot{\mathsf{A}}_{\mathbb{R}}$ be an $\mathbb{R}_\infty$-approximation and an $\mathbb{R}$-approximation, respectively. Consider the function $\mathsf{q} \colon \mathsf{A}_{\mathbb{R}_\infty} \to \mathbb{R}^2$ given, for each $A \in \mathsf{A}_{\mathbb{R}_\infty}$, by*

$$\mathsf{q}(A) \stackrel{\text{def}}{=} \bigl\{\, (x, y) \in \mathbb{R}^2 \bigm| \log_b|x| - \log_b|y| \in A \,\bigr\}.$$

Then, on the set $\mathsf{Q}(\mathsf{A}_{\mathbb{R}_\infty}) \stackrel{\text{def}}{=} \bigl\{\, \mathsf{q}(A) \bigm| A \in \mathsf{A}_{\mathbb{R}_\infty} \,\bigr\}$, we consider the following operations, for each $A_1, A_2 \in \mathsf{A}_{\mathbb{R}_\infty}$:

$$\begin{aligned}
\mathsf{q}(A_1) \otimes \mathsf{q}(A_2) &\stackrel{\text{def}}{=} \mathsf{q}(A_1 \otimes A_2),\\
\mathsf{q}(A_1) \oplus \mathsf{q}(A_2) &\stackrel{\text{def}}{=} \mathsf{q}(A_1 \oplus A_2),\\
\mathsf{q}(A_1) \boxdot \mathsf{q}(A_2) &\stackrel{\text{def}}{=} \mathsf{q}(A_1 \boxplus A_2).
\end{aligned}$$

If we are also given two functions $\boxtimes \colon \mathsf{A}_{\mathbb{R}} \times \mathsf{A}_{\mathbb{R}} \to \mathsf{Q}(\mathsf{A}_{\mathbb{R}_\infty})$ and $\boxbar \colon \mathsf{Q}(\mathsf{A}_{\mathbb{R}_\infty}) \times \mathsf{A}_{\mathbb{R}} \to \mathsf{A}_{\mathbb{R}}$ such that

$$\begin{aligned}
A_1 \boxtimes A_2 &\supseteq A_1 \times A_2,\\
\mathsf{q}(A_1) \boxbar A_2 &\supseteq \bigl\{\, y \in \mathbb{R} \bigm| \exists x \in A_2 \mathrel{.} \exists z \in A_1 \mathrel{.} y = \pm b^{-z}|x| \,\bigr\},
\end{aligned}$$

then $\ddot{\mathsf{Q}}(\dot{\mathsf{A}}_{\mathbb{R}_\infty}, \dot{\mathsf{A}}_{\mathbb{R}}) \stackrel{\text{def}}{=} \bigl\langle \mathsf{Q}(\mathsf{A}_{\mathbb{R}_\infty}), \mathsf{A}_{\mathbb{R}}, \boxdot, \boxtimes, \boxbar \bigr\rangle$, called bounded quotients over $\dot{\mathsf{A}}_{\mathbb{R}_\infty}$ and $\dot{\mathsf{A}}_{\mathbb{R}}$, *is an $\mathbb{R}^2$-approximation.*

5.8 Approximations are Constraints

As we have seen, our family of domains for range and relations analysis has, by now, three degrees of freedom. Once we have chosen an $\mathbb{R}$-approximation, $\mathsf{A}_{\mathbb{R}}$, an $\mathbb{N}$-approximation, $\mathsf{A}_{\mathbb{N}}$, and an $\mathbb{R}^2$-approximation, $\mathsf{A}_{\mathbb{R}^2}$, our base domain is the complete lattice

$$\mathcal{D}_n^\sharp \stackrel{\text{def}}{=} \mathcal{D}_n^\sharp(\mathsf{A}_{\mathbb{R}}, \mathsf{A}_{\mathbb{N}}, \mathsf{A}_{\mathbb{R}^2}) \stackrel{\text{def}}{=} \mathsf{A}_{\mathbb{R}}^n \times \mathsf{A}_{\mathbb{N}}^n \times \mathsf{A}_{\mathbb{R}^2}^{n(n-1)/2} \tag{5.28}$$

The $\mathsf{A}_{\mathbb{N}}$ component carries information on multiset cardinality, $\mathsf{A}_{\mathbb{R}}$ conveys *quantitative information* on the values of multisets' elements, whereas $\mathsf{A}_{\mathbb{R}^2}$ is for *qualitative information* on that values.

One way to obtain a description in $\mathcal{D}_n^\sharp$ for a predicate p in a program P, would be to execute all the possible computation paths of P, collecting all the tuples of ground terms that represent the possible successes of p. If the set of all tuples is $\hat{T}$ and we take a common anti-instance s of $\hat{T}$ with n variables, we know how, by means of the sequence of abstractions above, to obtain the desired element of $\mathcal{D}_n^\sharp$. This, of course, is impossible, in general.

Something else must be done, and the domain $\mathcal{D}_n^\sharp$ turns out to be too weak for our purposes. Before discussing this topic, we better equip ourselves with a language that will make easier talking about the elements of $\mathcal{D}_n^\sharp$.

We can think about each element of $\mathcal{D}_n^\sharp$ as a safe description of an n-tuple of multisets that is, in general, unknown. Let us call this tuple that we have in mind $\hat{X}$. It is important to understand that the tuple $\hat{X}$ is defined elsewhere: here $\hat{X}$ is just a variable symbol standing for a *particular, unspecified* tuple of multisets of numbers. When our approximations will be put at work each $\hat{X}$ will be given a precise definition, like the following:

$$\hat{X} \stackrel{\text{def}}{=} \bigl(\mathrm{nl}^+(t_1), \ldots, \mathrm{nl}^+(t_n)\bigr)$$

for some $(t_1, \ldots, t_n)$ belonging to an unknown set $\hat{T}$. The set $\hat{T}$, though entirely defined[14], is unknown, so is $(t_1, \ldots, t_n)$, and so will be $\hat{X}$. After all, the objective of the game is to derive some information about what $\hat{T}$ looks like.

So, $\hat{X}$ is a variable ranging over n-tuples of multisets. What we will do is to put constraints on $\hat{X}$ in order to restrict the set of n-tuples of multisets in its range. By doing this we will indirectly constrain the tuple $(t_1, \ldots, t_n)$ and thus $\hat{T}$ itself.

What is needed is a *language* for constraining $\hat{X}$ based on our approximations for cardinalities, sets, and relations. For instance, we could write

$$\zeta\bigl(\pi_1(\hat{X})\bigr) \subseteq [1, 3] \tag{5.29}$$

meaning that X can stand for any n-tuple of multisets of the reals such that the elements of the first multiset are all included in $[1, 3]$. We can use any element of a chosen $\mathbb{R}$-approximation instead of $[1, 3]$. If we write also

$$\bigl\|\pi_1(\hat{X})\bigr\| \in \{1, 5\} \tag{5.30}$$

we further specify that the first multiset must have cardinality 1 or 5. Once we have fixed an $\mathbb{N}$-approximation we have a language for expressing sets of

[14]For instance, "$\hat{T}$ is the set of n-tuples of terms to which a certain n-tuple of variables can be bound when the computation reaches a particular *control point* following one of some set of computation paths assuming that the resulting constraint store is consistent."

cardinalities. Finally, we might also specify that

$$\zeta\bigl(\pi_1(\hat{X})\bigr) \times \zeta\bigl(\pi_2(\hat{X})\bigr) \subseteq \bigl\{\, (x,y) \in \mathbb{R}^2 \bigm| x < y \,\bigr\}, \tag{5.31}$$

thus saying that the elements of the first multiset are less than those (if any) of the second multiset. Any $\mathbb{R}^2$-approximation provides us with a language for specifying binary relations.

The notation used in (5.29)–(5.31) is quite cumbersome. We will use the following equivalent formulation instead:

$$\begin{aligned} \hat{X}_1^{\mathrm{q}} &\subseteq [1,3], && \text{instead of (5.29);} \\ \hat{X}_1^{\mathrm{c}} &\subseteq \{1,5\}, && \text{instead of (5.30);} \\ \hat{X}_{1,2}^{\mathrm{r}} &\subseteq \bigl\{\, (x,y) \in \mathbb{R}^2 \bigm| x < y \,\bigr\}, && \text{instead of (5.31).} \end{aligned}$$

What we have done can be thought as if we had "exploded" the variable symbol $\hat{X}$ into several variable symbols. These are, for each $i, j \in \{1, \dots, n\}$,

$$\hat{X}_i^{\mathrm{c}}, \hat{X}_i^{\mathrm{q}}, \hat{X}_{ij}^{\mathrm{r}}.$$

Suppose $\hat{X}$ stands for the tuple

$$(M_1, \dots, M_n),$$

then

- $\hat{X}_i^{\mathrm{c}}$ stands for the singleton set of natural numbers containing the cardinality of M_i, that is $\bigl\{\|M_i\|\bigr\}$;
- $\hat{X}_i^{\mathrm{q}}$ stands for the set of values occurring in M_i, namely $\zeta(M_i)$;
- $\hat{X}_{ij}^{\mathrm{r}}$ stands for $\zeta(M_i) \times \zeta(M_j)$.

It is easy to associate each element of $\mathcal{D}_n^\sharp$ to a set of constraints of the form

$$v \subseteq K,$$

where v is any of $\hat{X}_i^{\mathrm{c}}$, $\hat{X}_i^{\mathrm{q}}$, or $\hat{X}_{ij}^{\mathrm{r}}$, and K is an element of the corresponding $\mathbb{N}$-, $\mathbb{R}$-, or $\mathbb{R}^2$-approximation. On the other hand, to each set of constraints of this form corresponds an element of $\mathcal{D}_n^\sharp$. However, with our syntax we can express more than this. We can say, for instance,

$$\hat{X}_i^{\mathrm{q}} \subseteq \hat{X}_j^{\mathrm{q}},$$

meaning that the i-th multiset does not contain more numeric values than those occurring in the j-th multiset. As we will see, we are moving to an abstract domain that is much more powerful than $\mathcal{D}_n^\sharp$: a space of *kernel operators* over $\mathcal{D}_n^\sharp$. For this purpose, we will do the following:

1. define a class of *expressions* over our special variable symbols and approximations;

2. define a class of *basic constraints* (or *tokens*) based on these expressions;

3. define a class of *entailment relations* between the basic constraints;

4. define a class of *determinate constraint systems*: as seen in Chapter 3, these are induced by the *simple constraint system* constituted by a set of basic constraints together with a suitable entailment relation and merge operator;

5. define *ask-and-tell* constraint systems over these d.c.s.;

6. use finite cc agents over the above ask-and-tell c.s. in order to capture the relevant aspects of concrete computations.

Let us start with expressions. The punctilious reader will forgive us if we do not use different symbols for the syntactic operators and the corresponding semantic operations.

Definition 85 (Abstract expressions.) *The set of* cardinality expressions, E_c, *is the language generated by the following grammar:*

$$\begin{aligned} c ::= {} & N && \textit{with } N \subseteq \mathsf{A}_\mathbb{N} \\ \mid {} & \hat{X}_i^\mathrm{c} && \textit{with } i \in \{1, \ldots, n\} \\ \mid {} & c_1 \otimes c_2 && \textit{with } c_1, c_2 \in \mathsf{E}_\mathrm{c} \\ \mid {} & c_1 \oplus c_2 && \\ \mid {} & c_1 \boxplus c_2 && \\ \mid {} & c_1 \boxminus_2 c_2 && \end{aligned}$$

Similarly, the sets of quantity expressions, $\mathsf{E}_\mathrm{q} \ni q$, *and* relationship expressions, $\mathsf{E}_\mathrm{r} \ni r$, *are given by*

$$
\begin{array}{rlll}
q ::= & S & & \textit{with } S \subseteq \mathsf{A}_\mathbb{R} \\
\mid & \hat{X}^\mathrm{q}_i & & \textit{with } i \in \{1, \dots, n\} \\
\mid & q_1 \otimes q_2 & & \textit{with } q_1, q_2 \in \mathsf{E}_\mathrm{q} \\
\mid & q_1 \oplus q_2 & & \\
\mid & r \boxbar q & & \textit{with } r \in \mathsf{E}_\mathrm{r} \\
\mid & q_1 \boxplus q_2 & & \\
\mid & \boxminus_1 q & & \\
\mid & q_1 \boxminus_2 q_2 & & \\
\mid & q_1 \boxast q_2 & & \\
\mid & q_1 \boxslash q_2 & & \\
\mid & \boxed{\sin}\, q & & \\
\vdots & & & \textit{(all the other operators)}
\end{array}
$$

and

$$
\begin{array}{rlll}
r ::= & R & & \textit{with } R \subseteq \mathsf{A}_{\mathbb{R}^2} \\
\mid & \hat{X}^\mathrm{r}_{ij} & & \textit{with } i, j \in \{1, \dots, n\} \textit{ and } i \neq j \\
\mid & r^{-1} & & \\
\mid & r_1 \otimes r_2 & & \textit{with } r_1, r_2 \in \mathsf{E}_\mathrm{r} \\
\mid & r_1 \oplus r_2 & & \\
\mid & r_1 \boxcircle r_2 & & \\
\mid & q_1 \boxtimes q_2 & & \textit{with } q_1, q_2 \in \mathsf{E}_\mathrm{q}
\end{array}
$$

The set of all such expressions is denoted by

$$
\mathsf{E} \stackrel{\mathrm{def}}{=} \mathsf{E}_\mathrm{c} \cup \mathsf{E}_\mathrm{q} \cup \mathsf{E}_\mathrm{r}. \tag{5.32}
$$

We will denote by E_0 *the set of* constant expressions, *namely, the subset of* E *containing all the expressions where no variable symbols occur.*

In what follows we will always assume that everything is "well-typed". Thus, when we write $e_1 \otimes e_2$ and $e_1 \in \mathsf{E}_\mathrm{q}$ we are implicitly stating that $e_2 \in \mathsf{E}_\mathrm{q}$. In order to simplify the exposition we also introduce the set

$$
\mathsf{A} \stackrel{\mathrm{def}}{=} \mathsf{A}_\mathbb{N} \cup \mathsf{A}_\mathbb{R} \cup \mathsf{A}_{\mathbb{R}^2} \tag{5.33}
$$

that will be used freely under the assumption of well-typedness. So, for $K_1, K_2 \in \mathsf{A}$, writing $K_1 \subseteq K_2$ implies that K_1 and K_2 belong to the same set, among the three that are united in (5.33).

Needless to say, constant expressions can be evaluated in the obvious way.

Definition 86 (Evaluation of expressions.) *The function*

$$\mathrm{eval}\colon \mathsf{E}_0 \to \mathsf{A}$$

is defined, by structural induction, in the obvious way. For instance, for each $e_1, e_2 \in \mathsf{E}_0$,

$$\begin{aligned}
\mathrm{eval}(K) &\stackrel{\mathrm{def}}{=} K, && \text{if } K \in \mathsf{A};\\
\mathrm{eval}(e_1 \otimes e_2) &\stackrel{\mathrm{def}}{=} \mathrm{eval}(e_1) \otimes \mathrm{eval}(e_2),\\
\mathrm{eval}(e_1 \oplus e_2) &\stackrel{\mathrm{def}}{=} \mathrm{eval}(e_1) \oplus \mathrm{eval}(e_2),\\
\mathrm{eval}(e_1^{-1}) &\stackrel{\mathrm{def}}{=} \mathrm{eval}(e_1)^{-1}\\
\mathrm{eval}(e_1 \boxplus e_2) &\stackrel{\mathrm{def}}{=} \mathrm{eval}(e_1) \boxplus \mathrm{eval}(e_2),\\
&\dots
\end{aligned}$$

Expressions need also to be manipulated syntactically. The (usually limited) algebraic capabilities of the system are encoded by a relation.

Definition 87 (Safe approximation.) *A relation* $\preccurlyeq \subseteq \mathsf{E} \times \mathsf{E}$ *is a* safe approximation relation *if*

$$\begin{aligned}
K_1 &\preccurlyeq K_2, && \text{for each } K_1, K_2 \in \mathsf{A} \text{ such that } K_1 \subseteq K_2;\\
e_1 &\preccurlyeq e_1 \oplus e_2, && \text{for each } e_1, e_2 \in \mathsf{E}.
\end{aligned}$$

Moreover, whenever

$$e_1 \preccurlyeq e_2$$

if $vars\bigl((e_1, e_2)\bigr) = \{v_1, \dots, v_k\}$ *and* $K_i \in \mathsf{A}$ *is a constant of the same type of* v_i, *with* $i = 1, \dots, k$, *then*

$$\mathrm{eval}\bigl(e_1[K_1/v_1, \dots, K_k/v_k]\bigr) \subseteq \mathrm{eval}\bigl(e_2[K_1/v_1, \dots, K_k/v_k]\bigr).$$

By the first condition the system is able to relate constants: a very minimal requirement. The second condition will be needed to define a sensible merge operator. The last condition, instead, is a guarantee of soundness: if $e_1 \preccurlyeq e_2$ the functions $f_{e_1}, f_{e_2}\colon \mathsf{A}^k \to \mathsf{A}$, encoded by e_1 and e_2 through eval and substitution, are such that $f_{e_1} \subseteq f_{e_2}$.

Definition 88 (Abstract constraints.) *An* abstract numeric constraint *is any formula taking one of the following forms:*

$$\begin{aligned}
\hat{X}_i^{\mathrm{c}} &\subseteq e_{\mathrm{c}}, && \text{with } e_{\mathrm{c}} \in \mathsf{E}_{\mathrm{c}};\\
\hat{X}_i^{\mathrm{q}} &\subseteq e_{\mathrm{q}}, && \text{with } e_{\mathrm{q}} \in \mathsf{E}_{\mathrm{q}};\\
\hat{X}_{ij}^{\mathrm{r}} &\subseteq e_{\mathrm{c}}, && \text{with } e_{\mathrm{r}} \in \mathsf{E}_{\mathrm{r}}.
\end{aligned}$$

Constraints not containing any variable symbol on their left-hand side are called basic constraints. *We assume that the left-hand sides of basic constraints consist of a single constant (namely, we assume that* constant folding *is always performed). The set of all the abstract constraints is denoted by* C.

The entailment relation is parametric with respect to a safe approximation relation.

Definition 89 (Abstract entailment.) *Let* $\preccurlyeq$ *be a safe approximation relation. The* entailment relation over $\preccurlyeq$, $\vdash \subseteq \wp_{\mathrm{f}}(\mathsf{C}) \times \mathsf{C}$ *is the minimal relation satisfying all the conditions of* Definition 23 *on page 44, plus the the following axiom schemata, where* v *(possibly subscripted) is any of* $\hat{X}_i^{\mathrm{c}}$, $\hat{X}_i^{\mathrm{q}}$, *or* $\hat{X}_{ij}^{\mathrm{r}}$, *for* i *and* j *in* $\{1, \ldots, n\}$ *with* $i \neq j$.
Constraints can be wakened:

$$\{v \subseteq e_1\} \vdash v \subseteq e_2, \qquad \text{if } e_1 \preccurlyeq e_2. \tag{5.34}$$

Constraints can be combined: by "conjuncting" expressions,

$$\{v \subseteq e_1, v \subseteq e_2\} \vdash v \subseteq e_1 \otimes e_2, \tag{5.35}$$

and by substituting expressions for some variables:

$$\left\{\begin{array}{c} v \subseteq e \\ v_1 \subseteq e_1 \\ \vdots \\ v_s \subseteq e_s \end{array}\right\} \vdash v \subseteq e[e_1/v_1, \ldots, e_k/v_k]. \tag{5.36}$$

Here the expression e_i *has the same type of* v_i, *for each* $i = 1, \ldots, s$. *Information is propagated from cardinalities to quantities,*

$$\{\, \hat{X}_i^{\mathrm{c}} \subseteq \{0\} \,\} \vdash \hat{X}_i^{\mathrm{q}} \subseteq \varnothing, \tag{5.37}$$

and back,

$$\{\, \hat{X}_i^{\mathrm{q}} \subseteq \varnothing \,\} \vdash \hat{X}_i^{\mathrm{c}} \subseteq \{0\}. \tag{5.38}$$

Unsatisfiability is detected and globalized: for cardinalities,

$$\{\, \hat{X}_i^{\mathrm{c}} \subseteq \varnothing \,\} \vdash \bot, \tag{5.39}$$

and for relations,

$$\left\{\begin{array}{c} \hat{X}_{ij}^{\mathrm{r}} \subseteq \varnothing \\ \hat{X}_i^{\mathrm{c}} \subseteq \mathbb{N} \setminus \{0\} \\ \hat{X}_j^{\mathrm{c}} \subseteq \mathbb{N} \setminus \{0\} \end{array}\right\} \vdash \bot. \tag{5.40}$$

The justification behind (5.39) is that a multiset must have a cardinality. Thus $\hat{X}_i^{\mathrm{c}} \subseteq \varnothing$ indicates an inconsistency. For (5.40), two sets of number are in the empty relation if and only if one or the other is empty. Thus, if both are non-empty the overall system is inconsistent.

By Definition 89 the structure $(\mathsf{C}, \vdash)$ is a simple constraint system. It is thus natural to apply the determinate constraint system construction described in Definition 26 on page 52.

Definition 90 *The constraint system* J *is the determinate constraint system built over* $(\mathsf{C}, \vdash)$ *with set intersection as the merge operator.*

An important observation is that the role of set intersection as a merge operator is controlled by the safe approximation relation $\preccurlyeq$ that has been chosen. Suppose we have two abstract constraints $C', C'' \in \mathsf{J}$ such that

$$\begin{aligned} C' &\supset \bigl\{\hat{X}_1^{\mathrm{q}} \subseteq e', \hat{X}_2^{\mathrm{q}} \subseteq \hat{X}_3^{\mathrm{q}} \boxplus \hat{X}_4^{\mathrm{q}}\bigr\}, \\ C'' &\supset \bigl\{\hat{X}_1^{\mathrm{q}} \subseteq e'', \hat{X}_2^{\mathrm{q}} \subseteq \hat{X}_4^{\mathrm{q}} \boxplus \hat{X}_3^{\mathrm{q}}\bigr\}. \end{aligned}$$

Given the minimal requirements we have imposed on $\preccurlyeq$, we know, since C' and C'' are closed under entailment, that

$$C' \cap C'' \supset \bigl\{\hat{X}_1^{\mathrm{q}} \subseteq e' \oplus e'', \hat{X}_2^{\mathrm{q}} \subseteq (\hat{X}_3^{\mathrm{q}} \boxplus \hat{X}_4^{\mathrm{q}}) \oplus (\hat{X}_4^{\mathrm{q}} \boxplus \hat{X}_3^{\mathrm{q}})\bigr\}.$$

Even if $\boxplus$ is commutative, without the $\preccurlyeq$ relation being such that

$$\hat{X}_i^{\mathrm{q}} \boxplus \hat{X}_j^{\mathrm{q}} \preccurlyeq \hat{X}_j^{\mathrm{q}} \boxplus \hat{X}_i^{\mathrm{q}}$$

we cannot have

$$C' \cap C'' \supset \{\hat{X}_2^{\mathrm{q}} \subseteq \hat{X}_3^{\mathrm{q}} \boxplus \hat{X}_4^{\mathrm{q}}\}.$$

This is how the algebraic capabilities of the system are reflected into the merge operation.

As we will see, the J constraint system is still not strong enough for our purposes. We thus define *finite* cc *agents* over J as in Section 3.6 on page 60, and move to an ask-and-tell constraint system,

Definition 91 *The constraint system* K *is the ask-and-tell constraint system built over* J *together with a suitable ask-and-tell merge operator, as in* Definition 44 on page 65.

For the choice of the merge operator some possibilities are described in Section 3.6.1 on page 68.

The ask-and-tell abstract domain K is quite expressive. Now the question is how do we use it? Or, in other words, where do the agents come from? The answer is that there are a variety of agents that come from different sources and serve different purposes:

implicit agents: agents expressing consistency conditions on $\hat{X}$;

numeric agents: agents that arise from concrete numeric expressions or are abstractions of concrete numeric constraints:

binding agents: agents that express parameter passing;

remapping agents: agents that arise in remapping;

The next few sections are devoted to describing them.

5.9 Implicit Agents

As the name says it, implicit agents do not arise as abstraction of concrete constraints or operations. Rather, they encode consistency conditions on the various pieces of information that are carried by abstract constraints. These conditions, which are based on quantitative and qualitative arithmetic reasoning, allow to approximate the reduced product of the three components we started with: $\mathsf{A}_{\mathbb{N}}$, $\mathsf{A}_{\mathbb{R}}$, and $\mathsf{A}_{\mathbb{R}^2}$.

5.9.1 Transitive Closure

A purely qualitative inference technique is known under the historical name of *transitive closure* [All83, Sim86]. It performs the approximate composition of known relationships in order to infer new relationships. Formally speaking, transitive closure is justified by the following inference rule, where $A, B, C \in \wp^+(\mathbb{R})$ and $R_1, R_2 \in \mathbb{R}^2$:

$$\frac{\begin{array}{c}\forall x \Subset A : \forall y \Subset B : x\ R_1\ y \\ \forall y \Subset B : \forall z \Subset C : y\ R_2\ z \\ B \neq \varnothing\end{array}}{\forall x \Subset A : \forall z \Subset C : x\ (R_1 \circ R_2)\ z} \tag{5.41}$$

Observe that the premise $B \neq \varnothing$, ensuring the existence of at least one "pivot element", is essential for the soundness of the inference. Rule (5.41) is approximated by the agents

$$\mathrm{ask}\bigl(\hat{X}^{\mathrm{c}}_{j} \subseteq \mathbb{N} \setminus \{0\}\bigr) \to \mathrm{tell}\bigl(\hat{X}^{\mathrm{r}}_{ik} \subseteq \hat{X}^{\mathrm{r}}_{jk} \boxdot \hat{X}^{\mathrm{r}}_{ij}\bigr), \tag{5.42}$$

for each $\{i, j, k\} \subseteq \{1, \ldots, n\}$.

Examples of inferences are the following, assuming that B is *non-empty*):

- $A < C$ from $A \leq B$ and $B < C$, if ordering relationships are adopted;

- $A - C \in [0, 1]$ from $A - B = -1$ and $B - C \in [1, 2]$, when $\mathsf{A}_{\mathbb{R}^2}$ is a system of bounded differences[15];
- $|A/C| \in [1, e]$ from $|A/B| = e^{-1}$ and $|B/C| \in [e, e^2]$, employing bounded quotients[16].

5.9.2 Quantity Refinement

The technique we call *quantity refinement* allows for the inference of quantitative information from qualitative information. Quantity refinement applies an approximation of the inference rule

$$\frac{\forall x \Subset A : \forall y \Subset B : x \mathrel{R} y \qquad A \neq \varnothing}{\zeta(B) \subseteq R\bigl(\zeta(A)\bigr)} \tag{5.43}$$

Again, the premise about non-emptiness is crucial. In fact, whenever $A = \varnothing$, the first premise of (5.43) holds vacuously for any relation $R \subseteq \mathbb{R}^2$. This, clearly, does not allow to conclude that $B = \varnothing$, since $R(\varnothing) = \varnothing$.

The repeated application of the inference rule (5.43) is an instance of *network consistency technique* [Mon74, Mac77, Fre78]. Such techniques consist in removing from the domains of network nodes (quantity labels, in our case) values that cannot appear in a solution, or, differently stated, that do not satisfy the *global constraint* represented by the network. More precisely, (5.43) expresses an *arc-consistency* step. All the arc-consistency techniques are based on an observation, due to Fikes [Fik70, Mac77], that can be rephrased as follows:

> Let u and v be two variables whose domains are D_u and D_v, respectively, and suppose we know that a certain relation R holds between u and v: formally we have the constraints $u \in D_u$, $v \in D_v$, and $u \mathrel{R} v$. If $x \in D_u$ and there is no $y \in D_v$ such that $(x, y) \in R$, then we can replace D_u with $D'_u = D_u \setminus \{x\}$ in the above set of constraints without changing its semantics. When we have gone through this process for each $x \in D_u$, ending up with
>
> $$D^*_u \stackrel{\text{def}}{=} D_u \setminus \bigl\{\, x \in D_u \bigm| \nexists y \in D_v \;.\; (x, y) \in R \,\bigr\},$$
>
> then the arc (u, v) (or, better, the relation R between u and v) is consistent with D^*_u and D_v. Consistency of the arc (v, u) is enforced in the same way, swapping u for v and considering R^{-1} instead of R.

[15]For instance, take $\mathsf{A}_{\mathbb{R}^2}$ to be $\mathsf{D}(\mathsf{I}_\mathsf{B})$ where the boundaries on which I_B is based include -1, 0, 1, and 2.

[16]For instance, $\mathsf{Q}(\mathsf{I}_\mathsf{B})$ where I_B is as in the previous example and assuming natural logarithms.

In our setting, the approximation of rule (5.43) is given by the agents

$$\mathrm{ask}\bigl(\hat{X}_i^{\mathrm{c}} \subseteq \mathbb{N} \setminus \{0\}\bigr) \to \mathrm{tell}\bigl(\hat{X}_j^{\mathrm{q}} \subseteq \hat{X}_{ij}^{\mathrm{r}} \boxbar \hat{X}_i^{\mathrm{c}}\bigr), \tag{5.44}$$

for each $i, j = 1, \ldots, n$ with $i \neq j$.

As an example of inference, let us assume that $\mathcal{D}_n^\sharp$ is based on $\mathsf{I}_{\mathsf{B}}^{\mathbb{N}}$ and O. Then, from $A > B$, B non-empty, $A \in [1 \mathrel{..} 6]$, and $B \in [4, 9]$, quantity refinement yields both $A \in [5 \mathrel{..} 6]$ and $B \in [4, 6)$. In general, when using intervals and ordering relationships, knowing the relation between A and B may allow to refine the intervals associated to A and B. It must be clear that (5.44) is only an approximation of an arc-consistency step. In fact, there is no *a priori* guarantee that the $\mathbb{R}$-approximation employed is expressive enough to effectively remove all the "impossible values", given a particular choice of $\mathbb{R}^2$-approximation. In the setting of intervals with ordering relationships this phenomenon can be appreciated by considering that the approximate refinement rule (5.44) cannot deduce anything, in any case, from $A \in [2 \mathrel{..} 2]$, $B \in [1, 3]$, and $A \neq B$.

5.9.3 Numeric Constraint Propagation

As quantity refinement exploit qualitative knowledge in order to obtain quantitative information, it is also possible to go the other way around. In [Sim86] an instance of this this process was named *numeric constraint propagation*. This technique consists in determining the relationship between two quantities when enough qualitative information is available. Formally, it is as simple as

$$\frac{}{\forall x \in A : \forall y \in B : x \ \bigl(\zeta(A) \times \zeta(B)\bigr)\ y} \tag{5.45}$$

which is then approximated by the agents

$$\mathrm{tell}\bigl(\hat{X}_{ij}^{\mathrm{r}} \subseteq \hat{X}_i^{\mathrm{q}} \boxtimes \hat{X}_j^{\mathrm{q}}\bigr), \tag{5.46}$$

for each $i, j = 1, \ldots, n$ with $i \neq j$.

When using intervals and ordering relationships, numeric constraint propagation consists in determining the relationship between two quantities when their associated intervals do not overlap, except possibly at their endpoints. For example, if $A \in (-\infty, 2]$, $B \in [2, +\infty)$, and $C \in [5, 10]$, we can infer that $A \leq B$ and $A < C$.

5.10 Numeric Agents

Numeric agents result from the abstract compilation of concrete clauses. For a rough description of the essence of this process, we help ourselves with the following two clauses, excerpted from the `fill_rectangle` program of Figure 4.2 on page 87.

```
fill([Y0,X1,Y1|L],[[X1,Y1,B]|C],L3,C3) :-     % clause 1
    Y0 > Y1,
    Y1 + B <= 1,
    place_square([X1,Y1|L],[X1 + B|L1]),
    fill([Y1 + B,X1 + B|L1],C,L2,C2),
    fill([Y0,X1|L2],C2,L3,C3).

place_square([_,Y1,X2,Y2|L],L1) :-           % clause 2
    Y1 = Y2,
    place_square([X2,Y2|L],L1).
```

The first thing to do is to assign to each *non-anonymous* program variable and to each numeric expression a *quantity*. A quantity is characterized by its *formula* and its *index*: a position within the tuple of multisets $\hat{X}$. We will denote by $\langle f\rangle_i$ the quantity corresponding to the formula f, which has been assigned the index i. Thus, the first step of the compilation is to assign the quantities

$$\bigl\{\langle\texttt{Y0}\rangle_1, \langle\texttt{X1}\rangle_2, \langle\texttt{Y1}\rangle_3, \langle\texttt{L}\rangle_4, \langle\texttt{B}\rangle_5, \langle\texttt{C}\rangle_6, \langle\texttt{L3}\rangle_7, \langle\texttt{C3}\rangle_8, \langle\texttt{Y1 + B}\rangle_9, \langle\texttt{X1 + B}\rangle_{10}, \langle\texttt{L1}\rangle_{11}, \langle\texttt{L2}\rangle_{12}, \langle\texttt{C2}\rangle_{13}\bigr\} \tag{5.47}$$

to clause 1, and the quantities

$$\bigl\{\langle\texttt{Y1}\rangle_1, \langle\texttt{X2}\rangle_2, \langle\texttt{Y2}\rangle_3, \langle\texttt{L}\rangle_4, \langle\texttt{L1}\rangle_5\bigr\} \tag{5.48}$$

to clause 2. It is important to remark that, as far as the compilation process is concerned, clauses are considered in isolation, one at a time, and no information is propagated from one clause to another.

Clause 1 is thus associated to a 13-tuple of multisets of numbers, one for each of its quantities. So, when dealing with clause 1, $\hat{X}_1^{\mathrm{c}}$ is the cardinality of the multiset of numbers that might appear in the term bound to `Y0` just before clause 1 terminates its execution with success, having followed one particular computation path. Similarly, $\hat{X}_4^{\mathrm{q}}$ is the set of numbers that might occur in the term bound to `L` etc.

5.10.1 Cardinality Agents

The next step in the compilation process is to examine the constraints appearing in the clause and the quantities' formulas in order to identify numeric variables. Variables appearing either on a numeric constraints, or in a numeric expression are certainly numeric. In other words, no computation of the clause can succeed if they are bound to a non-numeric term. Notice that also variables appearing in *aliasing constraints*, like `X = Y`, can be classified as numeric, if `X` or `Y` has been recognized as such. Thus, this local analysis requires a small fixpoint computation, or, if you prefer, the computation of a transitive closure. Quantities whose formula contains a numeric variable are classified as numeric.

For instance, in clause 1 of our small example, the quantities

$$\langle \texttt{Y0} \rangle_1, \langle \texttt{X1} \rangle_2, \langle \texttt{Y1} \rangle_3, \langle \texttt{B} \rangle_5, \langle \texttt{Y1 + B} \rangle_9, \langle \texttt{X1 + B} \rangle_{10},$$

are recognized as numeric, whereas no numeric quantities can be detected in clause 2. Now, for each numeric quantity we produce an agent of the form

$$\hat{X}_i^{\mathrm{c}} \subseteq \mathsf{A}_{\mathbb{N}}\bigl(\{1\}\bigr), \tag{5.49}$$

where i corresponds to the index of that quantity. This says, with the precision allowed by $\mathsf{A}_{\mathbb{N}}$, that the cardinality of our i-th multiset is 1, or, at the very least, that the i-th multiset is not empty.

As far as clause 2 is concerned, notice that the aliasing constraint `Y1 = Y2` does not say anything about the "type" of `Y1` and `Y2`. The fact that in any computation starting from the main program predicate (`fill_rectangle/3`, see Figure 4.2 on page 87) the constraint `Y1 = Y2` will play the role of a numeric equation is, of course, irrelevant.[17] Nonetheless, the constraint `Y1 = Y2` does give rise to a couple of cardinality agents. With reference to the set of quantities given in (5.48), they are

$$\hat{X}_1^{\mathrm{c}} \subseteq \hat{X}_3^{\mathrm{c}} \qquad \text{and} \tag{5.50}$$

$$\hat{X}_3^{\mathrm{c}} \subseteq \hat{X}_1^{\mathrm{c}}. \tag{5.51}$$

Whatever term `Y1` stands for, `Y2` stands for the same thing: this fact propagates to the cardinalities of the multisets of numbers that might be present in this term.

5.10.2 Constraint Agents

Constraint agents arise from a straightforward analysis of the constraints appearing in program clauses. With reference to the set of quantities (5.47), for clause 1 we get

$$\hat{X}_{1,3}^{\mathrm{r}} \subseteq \mathsf{A}_{\mathbb{R}^2}\Bigl(\bigl\{\, (x,y) \in \mathbb{R}^2 \bigm| x > y \,\bigr\}\Bigr), \tag{5.52}$$

from `Y0 > Y1`, and

$$\hat{X}_9^{\mathrm{q}} \subseteq \mathsf{A}_{\mathbb{R}}\bigl((-\infty, 1]\bigr), \tag{5.53}$$

from `Y1 + B <= 1`. For clause 2, instead, we have the constraint agent

$$\hat{X}_{1,3}^{\mathrm{r}} \subseteq \mathsf{A}_{\mathbb{R}^2}\Bigl(\bigl\{\, (x,x) \in \mathbb{R}^2 \,\bigr\}\Bigr). \tag{5.54}$$

[17] There are, however, CLP languages providing "typed" equations.

This shows us that it is always a good idea to employ an $\mathbb{R}^2$-approximation at least as strong as ordering relationships. This implies that bounded quotients approximation should not be used alone, but, for instance, in some kind of product with ordering relationships themselves.

When $\mathsf{A}_{\mathbb{R}^2}$ does contain ordering relationships,[18] the constraint agents for clause 1

$$\hat{X}_1^{\mathrm{q}} \subseteq (>) \boxbar \hat{X}_3^{\mathrm{q}} \qquad \text{and} \tag{5.55}$$

$$\hat{X}_3^{\mathrm{q}} \subseteq (<) \boxbar \hat{X}_1^{\mathrm{q}}, \tag{5.56}$$

that constitute a sound approximation of `Y0 > Y1`, are superfluous. In fact, they are subsumed by the combination of the quantity refinement agents

$$\begin{aligned} &\operatorname{ask}\bigl(\hat{X}_3^{\mathrm{c}} \subseteq \mathbb{N} \setminus \{0\}\bigr) \to \operatorname{tell}\bigl(\hat{X}_1^{\mathrm{q}} \subseteq \hat{X}_{3,1}^{\mathrm{r}} \boxbar \hat{X}_3^{\mathrm{c}}\bigr) \qquad \text{and} \\ &\operatorname{ask}\bigl(\hat{X}_1^{\mathrm{c}} \subseteq \mathbb{N} \setminus \{0\}\bigr) \to \operatorname{tell}\bigl(\hat{X}_3^{\mathrm{q}} \subseteq \hat{X}_{1,3}^{\mathrm{r}} \boxbar \hat{X}_1^{\mathrm{c}}\bigr), \end{aligned}$$

the cardinality agents

$$\begin{aligned} &\hat{X}_1^{\mathrm{c}} \subseteq \mathsf{A}_{\mathbb{N}}\bigl(\{1\}\bigr) \qquad \text{and} \\ &\hat{X}_3^{\mathrm{c}} \subseteq \mathsf{A}_{\mathbb{N}}\bigl(\{1\}\bigr), \end{aligned}$$

and the constraint agent (5.52). Indeed, the above combination is stronger than (5.55) and (5.56). In the latter the relationships are given by constant relation expressions, and thus they cannot be strengthened. In the combination, instead, the relation expression $\hat{X}_{1,3}^{\mathrm{r}}$ can be strengthened by effect of other agents (think about using bounded differences, or using ordering relationships supposing we had `Y0 >= Y1` instead of `Y0 > Y1`).

A similar argument holds for clause 2 too, where one might be tempted to produce the agents

$$\hat{X}_1^{\mathrm{q}} \subseteq \hat{X}_3^{\mathrm{q}} \qquad \text{and} \tag{5.57}$$

$$\hat{X}_3^{\mathrm{q}} \subseteq \hat{X}_1^{\mathrm{q}} \tag{5.58}$$

from the concrete constraint `Y1 = Y2`. Such agents will never be needed when using ordering relationships: an assumption that, from now on, we will take for granted.

The above discussion raises another point: if good "abstract code" has to be produced, the compilation process needs to be careful about the interplay among different inference techniques and to take into account the family of approximations employed. We will see other applications of this general principle.

[18] As stated in Definition 82 on page 136, $< \stackrel{\text{def}}{=} \bigl\{ (x,y) \in \mathbb{R}^2 \bigm| x < y \bigr\}$ and likewise for the other ordering relationships.

Since we are about to drop clause 2 from our running example, it is time to unveil a little lie. In fact, for ease of exposition we are describing a simplified compilation process. In CHINA, for instance, quantities which are tied by an equality constraint, such as `Y1 = Y2` in clause 2, are merged into only *one* quantity. This means that, in reality, a quantity corresponds to a *set* of formulas. This way the set of quantities for clause 2 would be something like $\bigl\{\langle \mathtt{Y1}, \mathtt{Y2}\rangle_1, \langle \mathtt{X2}\rangle_2, \langle \mathtt{L}\rangle_3, \langle \mathtt{L1}\rangle_4\bigr\}$ and the agents (5.50), (5.51), (5.54), (5.57), and (5.58) would never be generated.

5.10.3 Quantity Arithmetic Agents

A classical quantitative technique is *interval arithmetic* which allows to infer the variation interval of an expression from the intervals of its sub-expressions. An example inference is the deduction of $A \cdot B \in (-6, 30)$ from $A \in [3, 6)$ and $B \in [-1, 5]$. Since our $\mathbb{R}$-approximation might be something different from straight intervals, we call this inference technique *quantity arithmetic.*

The abstract compiler examines the quantities of clause 1, in particular

$$\langle \mathtt{X1}\rangle_2, \langle \mathtt{Y1}\rangle_3, \langle \mathtt{B}\rangle_5, \langle \mathtt{Y1\ +\ B}\rangle_9, \langle \mathtt{X1\ +\ B}\rangle_{10},$$

and produces the agents

$$\hat{X}^{\mathrm{q}}_9 \subseteq \hat{X}^{\mathrm{q}}_3 \boxplus \hat{X}^{\mathrm{q}}_5 \qquad \text{and} \tag{5.59}$$

$$\hat{X}^{\mathrm{q}}_{10} \subseteq \hat{X}^{\mathrm{q}}_2 \boxplus \hat{X}^{\mathrm{q}}_5, \tag{5.60}$$

on the grounds that

$$(\mathtt{X1\ +\ B}) = (\mathtt{X1}) + (\mathtt{B}) \qquad \text{and} \tag{5.61}$$

$$(\mathtt{Y1\ +\ B}) = (\mathtt{Y1}) + (\mathtt{B}). \tag{5.62}$$

But (5.61) can be implicitly manipulated so to obtain

$$(\mathtt{X1}) = (\mathtt{X1\ +\ B}) - (\mathtt{B}) \qquad \text{and} \tag{5.63}$$

$$(\mathtt{B}) = (\mathtt{X1\ +\ B}) - (\mathtt{X1}), \tag{5.64}$$

and the same can be done for (5.62). Thus the abstract compiler can emit also the agents

$$\hat{X}^{\mathrm{q}}_3 \subseteq \hat{X}^{\mathrm{q}}_9 \boxminus \hat{X}^{\mathrm{q}}_5, \tag{5.65}$$

$$\hat{X}^{\mathrm{q}}_5 \subseteq \hat{X}^{\mathrm{q}}_9 \boxminus \hat{X}^{\mathrm{q}}_3, \tag{5.66}$$

$$\hat{X}^{\mathrm{q}}_2 \subseteq \hat{X}^{\mathrm{q}}_{10} \boxminus \hat{X}^{\mathrm{q}}_5, \qquad \text{and} \tag{5.67}$$

$$\hat{X}^{\mathrm{q}}_5 \subseteq \hat{X}^{\mathrm{q}}_{10} \boxminus \hat{X}^{\mathrm{q}}_2. \tag{5.68}$$

Even though in our example only linear arithmetic expressions occur, we can, of course, treat non-linear expressions in a very similar way. The next technique, instead, is limited to linear constraints.

5.10.4 Linear Refinement Agents

Consider clause 1, where the quantities include $\langle \mathtt{Y1} \rangle_3$ and $\langle \mathtt{B} \rangle_5$. The constraint `Y1 + B <= 1` can be transformed, by means of trivial algebraic manipulations, into

$$(\mathtt{Y1}) \leq 1 - (\mathtt{B}) \qquad \text{and} \tag{5.69}$$
$$(\mathtt{B}) \leq 1 - (\mathtt{Y1}), \tag{5.70}$$

which can then be approximated by means of the agents[19]

$$\hat{X}^{\mathrm{q}}_3 \subseteq (\geq) \boxplus \Big(\mathsf{A}_{\mathbb{R}}\big(\{1\}\big) \boxminus \hat{X}^{\mathrm{q}}_1\Big) \qquad \text{and} \tag{5.71}$$
$$\hat{X}^{\mathrm{q}}_1 \subseteq (\geq) \boxplus \Big(\mathsf{A}_{\mathbb{R}}\big(\{1\}\big) \boxminus \hat{X}^{\mathrm{q}}_3\Big). \tag{5.72}$$

This simple pattern can be applied to any linear constraint. The agents (5.71) and (5.72) also suggest that it is a good idea to employ $\mathbb{R}$-approximations providing reasonably tight overestimates for the singleton sets containing the numeric constants that can appear in programs.

When using bounded differences a very common form of constraint is subject to a simplified treatment. This form is

```
X1 = X - 1,
```

and can be found in all sorts of inductive definitions. The agent

$$\hat{X}^{\mathrm{r}}_{i_{\mathtt{X}},i_{\mathtt{X1}}} \subseteq \big\{\, (x,y) \;\big|\; x,y \in \mathbb{R}, \quad x - y = 1 \,\big\}$$

achieves, through quantity refinement, the same effect of the linear refinement agents that can be derived from `X1 = X - 1`. Notice that this applies also to inequality constraints like `X1 < X - 1`. It also applies in the context of the previous section, when our set of quantities includes, say, the formulæ `X+1` and `X`.

5.10.5 Relational Arithmetic Agents

Another important technique is *relational arithmetic* [Sim86] that infers constraints on the qualitative relationship of an expression to its arguments. It can be given by a number of axiom schemata that are *strongly dependent* on the $\mathbb{R}^2$-approximation employed. For instance, when using ordering re-

[19]Reminder: we are assuming that $\mathsf{A}_{\mathbb{R}^2}$ includes ordering relationships.

lationships the following are valid, for each $\bowtie \in \{=, \neq, \leq <, \geq, >\}$:

$$x \bowtie 0 \iff (x+y) \bowtie y \tag{5.73}$$

$$x \bowtie y \iff (x+z) \bowtie (y+z) \tag{5.74}$$

$$x \bowtie y \iff e^x \bowtie e^y \tag{5.75}$$

$$(x > 0 \land y > 0) \implies \begin{cases} x \bowtie 1 \implies (x * y) \bowtie y \\ y \bowtie 1 \implies (x * y) \bowtie x \end{cases} \tag{5.76}$$

$$(x > 0 \land y < 0) \implies \bigl(x \bowtie -y \implies -1 \bowtie (x/y)\bigr) \tag{5.77}$$

Notice that this is only a small selection of the many axiom schemata on which relational arithmetic can be based. Observe also that there is no restriction to linear constraints.

If we consider clause 1 of our running example, whose quantities include

$$\langle \texttt{X1} \rangle_2, \langle \texttt{Y1} \rangle_3, \langle \texttt{B} \rangle_5, \langle \texttt{Y1 + B} \rangle_9, \langle \texttt{X1 + B} \rangle_{10},$$

then axioms (5.73) and (5.74) justify the introduction of the following relational arithmetic agents:

$$\begin{aligned} \hat{X}^{\mathrm{r}}_{10,2} &\subseteq \hat{X}^{\mathrm{q}}_5 \boxtimes \mathsf{A}_{\mathbb{R}}\bigl(\{0\}\bigr), \\ \hat{X}^{\mathrm{r}}_{10,5} &\subseteq \hat{X}^{\mathrm{q}}_2 \boxtimes \mathsf{A}_{\mathbb{R}}\bigl(\{0\}\bigr), \\ \hat{X}^{\mathrm{r}}_{9,3} &\subseteq \hat{X}^{\mathrm{q}}_5 \boxtimes \mathsf{A}_{\mathbb{R}}\bigl(\{0\}\bigr), \\ \hat{X}^{\mathrm{r}}_{9,5} &\subseteq \hat{X}^{\mathrm{q}}_3 \boxtimes \mathsf{A}_{\mathbb{R}}\bigl(\{0\}\bigr), \\ \hat{X}^{\mathrm{r}}_{10,9} &\subseteq \hat{X}^{\mathrm{r}}_{2,3}, \\ \hat{X}^{\mathrm{r}}_{2,3} &\subseteq \hat{X}^{\mathrm{r}}_{10,9}. \end{aligned}$$

Observe that the axiom schemata (5.73) and (5.74) hold also when the $\mathbb{R}^2$-approximation employed consists of bounded differences. Notice also that a reasonable system of bounded differences is likely to satisfy

$$A \boxtimes \mathsf{A}_{\mathbb{R}}\bigl(\{0\}\bigr) = \mathsf{d}(A),$$

for each $A \in \mathsf{A}_{\mathbb{R}}$ (see Definition 83 on page 138).

5.11 Binding Agents

Here we deal with the problem of reflecting into an abstract constraint (namely, a collection of agents) the changes that are induced by the unification of two terms. Starting with an ask-and-tell constraint C, we must produce another constraint C' as specified in Sections 4.5.2 on page 97, for the case where the occur-check is performed at the concrete level, and 4.6.1 on page 107, when the occur-check is omitted. The new agent C' will be obtained by adding some new agents to C, by closing the result with respect to the inference map, and by projecting away some information.

So we have an unknown n-tuple of multisets of real numbers, $\hat{X}$, that describes an unknown set of n-tuples of ground terms $\hat{T}$. Let us arbitrarily choose

$$(t_1, \dots, t_n) \in \hat{T}.$$

All what we know is that $t_h = u$, for some term u. We distinguish the following cases along the lines of Section 4.5.2: for each case the agents to be added to C are specified.

Binding to a Symbolic Constant

A symbolic constant does not contain any numerical leaf. Thus we can add the agent

$$\hat{X}^{\mathrm{c}}_h \subseteq \{0\}. \tag{5.78}$$

Binding to a Number

A number u has a single numerical leaf: u itself. The agents

$$\hat{X}^{\mathrm{c}}_h \subseteq \mathsf{A}_{\mathbb{N}}\bigl(\{1\}\bigr) \qquad \text{and} \tag{5.79}$$

$$\hat{X}^{\mathrm{q}}_h \subseteq \mathsf{A}_{\mathbb{R}}\bigl(\{u\}\bigr) \tag{5.80}$$

specify this fact.

Binding to an Alias

Here we know that $u = t_j$ for some $j \in \{1, \dots, n\}$ with $j \neq h$. This means that t_h and t_j are identical, which can be abstracted by

$$\hat{X}^{\mathrm{c}}_h \subseteq \hat{X}^{\mathrm{c}}_j, \tag{5.81}$$

$$\hat{X}^{\mathrm{c}}_j \subseteq \hat{X}^{\mathrm{c}}_h, \tag{5.82}$$

$$\hat{X}^{\mathrm{q}}_h \subseteq \hat{X}^{\mathrm{q}}_j, \qquad \text{and} \tag{5.83}$$

$$\hat{X}^{\mathrm{q}}_j \subseteq \hat{X}^{\mathrm{q}}_h. \tag{5.84}$$

Relations are clearly inherited, too. Thus we can add the following agents, for each $m = \{1, \dots, n\} \setminus \{h, j\}$:

$$\hat{X}^{\mathrm{r}}_{hm} \subseteq \hat{X}^{\mathrm{r}}_{jm}, \tag{5.85}$$

$$\hat{X}^{\mathrm{r}}_{jm} \subseteq \hat{X}^{\mathrm{r}}_{hm}. \tag{5.86}$$

Binding to a Compound

In this case u is obtained by instantiation from a “skeleton term” s that is a compound term. More precisely, we can assume, without loss of generality, that $\mathit{vars}(s) = \{Y_{j_1}, \dots, Y_{j_l}\}$, with $l \geq 0$ and

$$\{j_1, \dots, j_l\} \subset \{1, \dots, n\} \setminus \{h\},$$

and that

$$u = t_h = s\bigl[t_{j_1}/Y_{j_1}, \dots, t_{j_l}/Y_{j_l}\bigr]. \tag{5.87}$$

Let us consider the extension of $\mathrm{nl}^+ \colon \mathcal{T} \to \wp_{\mathrm{f}}^+(\mathbb{R})$ to possibly non-ground terms: $\mathrm{nl}^+ \colon \mathcal{T}_{\mathit{Vars}} \to \wp_{\mathrm{f}}^+(\mathbb{R})$ is obtained by considering that a variable is *not* a numerical leaf. Then, Equation (5.87) implies

$$\mathrm{nl}^+(t_h) = \mathrm{nl}^+(s) \uplus \biguplus_{i=1}^{l} \mathrm{nl}^+\bigl(t_{j_i}\bigr). \tag{5.88}$$

This implies both

$$\bigl\|\mathrm{nl}^+(t_h)\bigr\| = \|\mathrm{nl}^+(s)\| + \sum_{i=1}^{l} \bigl\|\mathrm{nl}^+\bigl(t_{j_i}\bigr)\bigr\| \tag{5.89}$$

and

$$\zeta\bigl(\mathrm{nl}^+(t_h)\bigr) = \zeta\bigl(\mathrm{nl}^+(s)\bigr) \cup \bigcup_{i=1}^{l} \zeta\Bigl(\mathrm{nl}^+\bigl(t_{j_i}\bigr)\Bigr). \tag{5.90}$$

Now, Equation 5.89 is equivalent, for each $k = 1, \dots, l$, to

$$\bigl\|\mathrm{nl}^+\bigl(t_{j_k}\bigr)\bigr\| = \bigl\|\mathrm{nl}^+(t_h)\bigr\| - \left(\|\mathrm{nl}^+(s)\| + \sum_{\substack{i=1\\ i \neq k}}^{l} \bigl\|\mathrm{nl}^+\bigl(t_{j_i}\bigr)\bigr\| \right) \tag{5.91}$$

whereas (5.90) implies, again for each $k = 1, \dots, l$,

$$\zeta\Bigl(\mathrm{nl}^+\bigl(t_{j_k}\bigr) \subseteq \zeta\bigl(\mathrm{nl}^+(t_h)\bigr). \tag{5.92}$$

If we stipulate that $c_s \stackrel{\mathrm{def}}{=} \bigl\|\mathrm{nl}^+(s)\bigr\|$ and $q_s \stackrel{\mathrm{def}}{=} \zeta\bigl(\mathrm{nl}^+(s)\bigr)$, we have justified the addition to C of the following agents:

$$\hat{X}^{\mathrm{c}}_h \subseteq \mathsf{A}_{\mathbb{N}}\bigl(\{c_s\}\bigr) \boxplus \mathop{\boxplus}_{i=1}^{l} \hat{X}^{\mathrm{c}}_{j_i}, \tag{5.93}$$

$$\hat{X}^{\mathrm{c}}_{j_k} \subseteq \hat{X}^{\mathrm{c}}_h \boxminus \left(\mathsf{A}_{\mathbb{N}}\bigl(\{c_s\}\bigr) \boxplus \mathop{\boxplus}_{\substack{i=1\\ i\neq k}}^{l} \hat{X}^{\mathrm{c}}_{j_i} \right), \qquad \text{for } k = 1, \ldots, l, \tag{5.94}$$

$$\hat{X}^{\mathrm{q}}_h \subseteq \mathsf{A}_{\mathbb{R}}(q_s) \oplus \bigoplus_{i=1}^{l} \hat{X}^{\mathrm{q}}_{j_i}, \tag{5.95}$$

$$\hat{X}^{\mathrm{q}}_{j_k} \subseteq \hat{X}^{\mathrm{q}}_h, \qquad \text{for } k = 1, \ldots, l. \tag{5.96}$$

For relations we reason as follows. Equation (5.92) implies that if the numerical leaves of t_h are in some relation with the numerical leaves of, say, t_m, then this is true also for the subset of numerical leafs that are in t_{j_k}. This justifies the agents

$$\hat{X}^{\mathrm{r}}_{j_k m} \subseteq \hat{X}^{\mathrm{r}}_{hm} \tag{5.97}$$

for each $k = 1, \ldots, l$ and each $m = 1, \ldots, n$ with $m \neq j_k$. On the other hand, the numeric leaves of t_h can be obtained by "putting together" the pieces indicated by (5.90). If we join the relations that hold between each piece and the leaves of t_m we obtain a relation that holds for the entire collection. We thus add the agents

$$\hat{X}^{\mathrm{r}}_{hm} \subseteq \bigl(\mathsf{A}_{\mathbb{R}}(q_s) \boxtimes \hat{X}^{\mathrm{q}}_m\bigr) \oplus \bigoplus_{i=1}^{l} \hat{X}^{\mathrm{r}}_{j_i m}, \tag{5.98}$$

for each $m \in \{1, \ldots, n\} \setminus \{h, j_1, \ldots, j_l\}$.

Cyclic Binding to a Compound

In this case, the handling of cyclic binding turns out to be easy. However, we need an approximation for subsets of the extended naturals $\mathbb{N} \cup \{\infty\}$. This $\mathbb{N}_\infty$-approximation (whose simple definition is omitted) is needed for cardinalities, since a cyclic term may have an infinite number of numerical leaves.

With this change, cyclic bindings are very similar to normal bindings. They are thus handled by means of the agents described in the previous paragraph, where $\{j_1, \ldots, j_l\}$ has been replaced by $\{j_1, \ldots, j_l\} \setminus \{h\}$.

5.12 Making It Practical

We have set up a very precise domain for range and relations analysis. Indeed, being obtained from a non-trivial constraint system by means of the ask-and-tell construction, this domain is much too complex to be practical. We attack this complexity from two sides:

1. on one side, all the agents constituted by basic constraints (see Definition 88 on page 145) and all the implicit agents can be efficiently represented and dealt with by means of *constraint networks*;

2. on the other side, we can apply the golden rule of abstract interpretation: information can be safely thrown away at any time.

5.12.1 Representing Basic Constraints and Implicit Agents

The term *constraint network* is very general, and has been used to define a wide variety of structures. Here we define a particular kind of constraint networks that suits our purposes.

Definition 92 (Constraint network.) *Let* $\mathsf{A}_{\mathbb{N}}$, $\mathsf{A}_{\mathbb{R}}$, *and* $\mathsf{A}_{\mathbb{R}^2}$ *be approximations for* $\mathbb{N}$, $\mathbb{R}$, *and* $\mathbb{R}^2$, *respectively. A* constraint network *is a structure of the form*

$$(n, \ell, \dot{\ell}, \ddot{\ell}),$$

where

1. $n \in \mathbb{N}$; *the members of* $Q_n \stackrel{\text{def}}{=} \{1, \ldots, n\}$ *are called* quantities; *an element of* $R_n \stackrel{\text{def}}{=} \{\, (i, j) \mid i, j \in Q_n, i \neq j \,\}$ *is called a* proper pair *of the constraint network;*

2. $\ell \colon Q_n \to \mathsf{A}_{\mathbb{N}}$ *assigns a* cardinality *to each quantity;*

3. $\dot{\ell} \colon Q_n \to \mathsf{A}_{\mathbb{R}}$ *assigns a* range *to each quantity;*

4. $\ddot{\ell} \colon R_n \to \mathsf{A}_{\mathbb{R}^2}$ *assigns a* relation *to each proper pair.*

In practice, a constraint network is a complete, directed, and labelled graph where we have two labels for the nodes and one label for the edges. Each constraint network corresponds to an element of $\mathcal{D}_n^\sharp$ as given in (5.28) on page 140. Constraint networks are also a compact representation for any set of basic constraints.

A constraint network can be inconsistent.

Definition 93 (Consistency.) *Let* $\mathcal{Q} \stackrel{\text{def}}{=} (n, \ell, \dot{\ell}, \ddot{\ell})$ *be a constraint network.* $\mathcal{Q}$ *is said* inconsistent *if at least one of the following conditions holds:*

1. *there exists* $i \in Q_n$ *such that* $\ell(i) = \varnothing$;

2. *there exists* $i \in Q_n$ *such that* $\ell(i) \geq 1$ *and* $\dot{\ell}(i) = \varnothing$;

3. there exist $(i, j) \in R_n$ *such that* $\ddot{\ell}(i, j) = \varnothing$ *and either* $\ell(i) \geq 1$ *or* $\ell(j) \geq 1$.

The constraint network $\mathcal{Q}$ *is said* consistent *otherwise.*

Another advantage of constraint networks is that all the implicit agents need not be represented explicitly. The kernel operators they denote are simply realized by graph algorithms. In the following presentation of the algorithms we will make use of the notation $\hat{\ell}(o) := e$, where $\hat{\ell}$ is one of ℓ, $\dot{\ell}$, or $\ddot{\ell}$, o is a node or a pair, depending on $\hat{\ell}$, and e is an expressions. The meaning of $\hat{\ell}(o) := e$ can expressed, using Hoare logic, as follows:

$$\begin{gathered}\bigl\{\hat{\ell}(o) = K\bigr\}\\ \hat{\ell}(o) := e\\ \bigl\{\hat{\ell}(o) = K \otimes e\bigr\}\end{gathered}$$

Transitive Closure

A simple algorithm for applying transitive closure up to quiescence can be obtained by a suitable variation of Warshall/Warren algorithm for graph closure [War62, War75]. A first, naive version of it is given as Algorithm 3.

```
Require: any constraint network $(n, \ell, \dot{\ell}, \ddot{\ell})$
Ensure: $\forall i, j, k \in Q_n : \ell(j) \geq 1 \implies \ddot{\ell}(i, k) \subseteq \ddot{\ell}(j, k) \boxdot \ddot{\ell}(i, j)$
  for all $j \in Q_n$ do
    if $\ell(j) \geq 1$ then
      for all $i \in Q_n$ do
        for all $k \in Q_n$ do
          $\ddot{\ell}(i, k) := \ddot{\ell}(j, k) \boxdot \ddot{\ell}(i, j)$
```

Algorithm 3: Naive algorithm for *transitive closure.*

Quantity Refinement

An almost obvious algorithm which applies quantity refinement as much as possible is Algorithm 4 on the next page. Perhaps, the only non-obvious part is the need for the outer iteration. To see why this is necessary, let us modify the example above. Suppose that we have three quantities: $A \in [1 \mathinner{..} 6]$ and $B, C \in [4, 9]$. Suppose also that we have the relations $A > B \geq C$ and $A \ \mathbb{R}^2 \ C$. If we apply refinement to B and C or to A and C we cannot deduce anything new. Then we can apply refinement to A and B, obtaining $A \in [5 \mathinner{..} 6]$ and $B \in [4, 6)$, as we have seen before. But now we must apply refinement to B and C again in order to obtain $C \in [4, 6)$.

Indeed, Algorithm 4 is naive under several aspects. First of all it considers also the pairs from which no refinement is possible. This is easily fixed by changing line 5 so to read as

```
Require: any constraint network $(n, \ell, \dot{\ell}, \ddot{\ell})$
Ensure: $\forall i, j \in Q_n : \ell(i) \geq 1 \implies \dot{\ell}(j) \subseteq \ddot{\ell}(i, j) \boxbar \dot{\ell}(i)$
1: repeat
2:    $changed := false$
3:    for all $i \in Q_n$ do
4:       if $\ell(i) \geq 1$ then
5:          for all $j \in Q_n$ such that $i \neq j$ do
6:             $\dot{\ell}(j) := \ddot{\ell}(i, j) \boxbar \dot{\ell}(i)$
7:             if $\dot{\ell}(j)$ has changed then
8:                $changed := true$
9: until $changed = false$
```

Algorithm 4: Naive algorithm for *quantity refinement.*

for all $j \in Q_n$ such that $i \neq j$ and $\ddot{\ell}(i, j) \neq \mathbb{R}^2$ **do**

What we have obtained is an algorithm called AC-1 [Mac77]. More serious arc-consistency algorithms originate from the *Waltz filtering algorithm* [Wal75], which was later named AC-2. A simpler and more general version of AC-2, known as AC-3, is given as Algorithm 5. It improves on AC-1 by

```
Require: any constraint network $(n, \ell, \dot{\ell}, \ddot{\ell})$
Ensure: $\forall i, j \in Q_n : \ell(i) \geq 1 \implies \dot{\ell}(j) \subseteq \ddot{\ell}(i, j) \boxbar \dot{\ell}(i)$
  $S := \{ (i, j) \mid i, j \in Q_n, \ell(i) \geq 1, \ddot{\ell}(i, j) \neq \mathbb{R}^2 \}$
  while $S \neq \varnothing$ do
     select $(h, k) \in S$
     $S := S \setminus \{(h, k)\}$
     $\dot{\ell}(k) := \ddot{\ell}(h, k) \boxbar \dot{\ell}(h)$
     if $\dot{\ell}(k)$ has changed and $\ell(k) \geq 1$ then
        $S := S \cup \{ (k, l) \mid l \in Q_n, \ddot{\ell}(k, l) \neq \mathbb{R}^2 \}$
```

Algorithm 5: AC-3 algorithm for *quantity refinement.*

avoiding repeated refinement using quantities which have not changed.

Numeric Constraint Propagation

The algorithm for numeric constraint propagation given as Algorithm 6 on the facing page is self-explanatory.

The remaining agents can be "interpreted" in the standard way (see, e.g., [VSD92a]) thus implementing the associated kernel operators. One thing worth noticing, though it is well-known, is that the functional composition of two kernel operators is not necessarily a kernel operator. Thus, the algorithms for transitive closure, quantity refinement, numeric constraint

```
Require: any constraint network $(n, \ell, \dot{\ell}, \ddot{\ell})$
Ensure: $\forall i, j \in Q_n : i \neq j \implies \ddot{\ell}(i,j) \subseteq \dot{\ell}(i) \boxtimes \dot{\ell}(j)$
1: for all $i \in Q_n$ do
2:   for all $j \in Q_n$ such that $i \neq j$ do
3:     $\ddot{\ell}(i,j) := \dot{\ell}(i) \boxtimes \dot{\ell}(j)$
```

Algorithm 6: Algorithm for *numeric-constraint-propagation.*

propagation, and for the interpretation of all the other agents must be enclosed in an outer loop and iterated, in principle, until quiescence. However, depending on the chosen approximations for cardinalities, sets, and binary relations, quiescence is not guaranteed to happen. This brings us to the next section.

5.12.2 Widenings

Most members of the family of numerical domains we have presented[20] either have infinite height, or their height can be considered infinite by any practical means. Thus widening operators are undoubtedly required in order to ensure or speed-up the convergence of the analysis. We have widenings in several of the domain components: widenings for the $\mathbb{N}$-, $\mathbb{R}$-, and $\mathbb{R}^2$-approximations, serve as a base for designing widening operators at the higher levels.

However, the practicality of the domain relies also on *unary widenings*: unary operators that allow losing information without losing correctness. They typically consist in throwing away agents. Since our domain is monotonic this is always safe. An example where this is absolutely necessary was announced at the end of the previous section. Suppose we have the set of agents constituted by

$$\hat{X}_1^{\mathrm{q}} \subseteq [0, +\infty), \tag{5.99}$$

$$\hat{X}_2^{\mathrm{q}} \subseteq [0, +\infty), \tag{5.100}$$

$$\hat{X}_1^{\mathrm{q}} \subseteq \hat{X}_2^{\mathrm{q}} \boxplus [1, 1], \tag{5.101}$$

$$\hat{X}_2^{\mathrm{q}} \subseteq \hat{X}_1^{\mathrm{q}} \boxplus [1, 1]. \tag{5.102}$$

They give rise to an infinite (or virtually infinite) sequence of refinements. Even though this kind of situation occurs rarely, a unary widening is needed. In the current version of China the system sets a watchdog timer before entering the "agents interpreter". In the above situation, a timeout condition occurs and the execution of a unary widening is triggered. This widening will identify the loop constituted by (5.101) and (5.102) in the dependency graph of the interpreter and will arbitrarily kill one or both the guilty agents.

Unary widenings are also used in other places. In general, they are employed in order to throttle complex agents. One extreme possibility is

[20]We have a specific domain for each possible choice of the various parameters.

to remove all the non-basic agents after each meet operation of the ask-and-tell constraint system. This clearly results in a very fast analysis, but all the "global effects" captured by complex agents are lost. Alternatively, heuristics can be used in order to decide which agents to evict. Unary widenings are also applied during the join operation. For example, the first domain of Janssens *et al.* [JBE94], if recasted in our setting, prescribes the removal of all the non-basic agents before each join operation.

5.13 Conclusion

We have shown that the compile-time detection of redundant numeric constraints in CLP programs can play a crucial role in the field of semantics based program manipulation. This is especially true for the area of optimized compilation, where they can enable major performance leaps. Besides reviewing known optimizations, we have proposed a novel optimization based on *call-graph simplification.* The nice thing about the simplified call-graph optimization is that it is an optimization in a strict sense: in exchange for a small increase in code size, the optimized program is guaranteed to be at least as fast as the unoptimized one. Moreover, this optimization helps when the known indexing techniques, that are normally used to reduce the search space, fail.

Even though the analysis of numeric constraints has many important application, and despite the fact that almost every existing CLP language incorporates some kind of numeric domain, this research topic is relatively unexplored. We have described a sequence of approximations for characterizing, by means of ranges and relations, the values of the numerical leaves that appear in Herbrand terms. In particular, we have illustrated for the first time the need for cardinality information. For each approximation needed for the analysis, we have presented different alternatives that allow for dealing with the complexity/precision tradeoff in several ways.

We have then presented a sophisticated numeric domain, which is obtained (among other things) by means of the ask-and-tell construction of Chapter 3. The ask-and-tell constraint system constitutes a very convenient formalism for expressing both

- efficient reasoning techniques originating from the world of artificial intelligence, where approximate deduction holds the spotlight since the origins; and
- the abstract operations of the domain.

In practice, we have defined a family of concurrent languages that serve as target-languages in an abstract compilation approach. One notable advantage is that the correctness proof is constructive: every abstract agent has been introduced at the end of an argument that shows its correctness.

The future work is of experimental nature. We have developed, in the context of the China analyzer, a modular implementation of the numeric domain presented in this chapter. Modularity is essential, due to the many degrees of freedom that characterize the construction. Up to now, the emphasis of the development has been on correctness, but keeping into account that efficiency must finally be obtained. We have just started working on the optimization of the domain's components. Then we will turn to the experimentation of several widening strategies: something that, today, is a completely empirical activity.

Chapter 6

Definiteness Analysis

Contents

6.1 Introduction

The task of *groundness analysis* (or *definiteness* analysis as it is also referred to) is to derive, for all the program points of interest, whether a certain variable is bound to a unique value (or *ground*). This kind of information is very important: it allows substantial optimizations to be performed at compile-time, and is also crucial for most semantics-based program manipulation tools. Moreover, many other analyses are made much more precise by the availability of groundness information. For these reasons, the subject of groundness analysis for (constraint) logic programs has been widely studied. After the early attempts, some classes of Boolean functions have been recognized as constituting good abstract domains for groundness analysis [CFW91, CH93, AMSS, HCC95]. In particular, the set of *positive Boolean functions*, (namely, those functions that assume the true value under the valuation assigning true to all variables), which is denoted by *Pos*, allows to express Boolean properties of program variables where the property of one variable may depend on that property of other variables. For groundness analysis, since variables can be bound to terms containing other variables, the groundness of one variable may depend on the groundness of other variables. *Pos* has been recognized as the most precise domain for capturing the kind of dependencies arising in groundness analysis.

This ability to express dependencies makes analysis based on *Pos* very precise, but also makes it relatively expensive, as many operations on Boolean formulas have exponential worst case complexity. Armstrong *et al.* [AMSS94a] analyzed many representations of positive Boolean formulas for abstract interpretation, and found *Reduced Ordered Binary Decision Diagrams* (ROBDDs) to give the best performance.

ROBDDs [Bry86], also known as *Bryant graphs*, are a representation for Boolean functions supporting many efficient operations. Because of this, they have often been used to implement *Pos* for abstract interpretation. Indeed, ROBDDs are general enough to represent *all* Boolean functions. However, nobody has yet succeeded in exploiting the (seemingly very small) peculiarities of positive functions in order to obtain a more efficient implementation.

Several authors have reported successful experiences using ROBDDs for groundness analysis (see, e.g., [CH93, AMSS]). However, in the literature there is no reference to the problem of detecting, as efficiently as possible, those variables which are deemed to be ground in the context of a ROBDD. This is not surprising, since most currently implemented analyzers need to derive this information only *at the end* of the analysis and only for presentation purposes. In these cases efficiency is not a problem and the simple approaches are sufficient. Things are very different when this information is required *during* the analysis. This need arises when dealing with languages which employ some sort of *delay mechanism*, which is typically based on groundness conditions. One of these languages is CLP($\mathcal{R}$) [JMSY92b], where non-linear constraints are delayed until they become linear; only then are they sent to the constraint solver. In the context of our work on data-flow analysis for CLP($\mathcal{R}$) we thus faced the following problem: in programs with many non-linear constraints, the abstract interpreter spends a lot of time deciding whether a constraint is delayed or not. In the early implementations of the China analyzer this kind of information (which is needed quite often) was derived using the ROBDD package itself (see Section 6.5). This had the advantage of making possible the use of untouched, readily-available ROBDD software, while having the big disadvantage of inefficiency.

In this chapter we introduce and study the problem of quick detection of ground variables using ROBDDs. We first propose an easy, even though not completely satisfactory, solution. We then take a different approach where we represent *Pos* functions in a hybrid way: ground variables are represented explicitly, while ROBDDs come into play only for dependency and disjunctive information. This solution uses the more efficient representation for each kind of information: "surely ground variables" are best represented by means of sets (bit-vectors, at the implementation level), whereas ROBDDs are used only for "conditional" and "disjunctive" information. In such a way, besides making the information about ground variables readily available, we can keep the ROBDDs generated during the analysis as small as possible. This promises to be a win, given that most real programs (together with their typical call-patterns) exhibit a high percentage of variables which are ground at the program points of interest. Notice that Boolean functions are used in the more general context of *dependency analysis*, including *finiteness analysis* for deductive database languages [BDM92] and *suspension analysis* for concurrent (constraint) logic programming languages [FGMP95]. The techniques we propose might be useful also in these contexts. However, this is something we have not studied yet. In Section 6.2 we briefly review the usage of Boolean functions for groundness analysis of (constraint) logic programs (even though we assume familiarity on this subject). Section 6.3 presents the main motivations of this work. Binary-decision trees and diagrams, and the problem of extracting *sure groundness information* from them are introduced in sections 6.4 and 6.5. In Section 6.6 we show a first non-trivial solution to the problem, while Section 6.7 introduces the hybrid domain. The results of the experimental evaluation are reported in Section 6.8. Section 6.9 concludes with some final remarks.

6.2 Boolean Functions for Groundness Analysis

After the early approaches to groundness analysis [Mel85, JS87], which suffered from serious precision drawbacks, the use Boolean functions has become customary in the field. The reason is that Boolean functions allow to capture in a very precise way the *groundness dependencies* which are implicit in unification constraints such as $z = f(g(x), y)$: the corresponding Boolean function is $(x \wedge y) \leftrightarrow z$, meaning that z is ground if and only if x and y are so. They also capture dependencies arising from other constraint domains: for instance, $x + 2y + z = 4$ can be abstracted as $((x \wedge y) \to z) \wedge ((x \wedge z) \to y) \wedge ((y \wedge z) \to x)$. We now introduce Boolean valuations and functions in a way which is suitable for what follows. *Vars* is a fixed denumerable set of variable symbols.

Definition 94 (Boolean valuations.) *The set of* Boolean valuations over *Vars is given by* $\mathcal{A} \stackrel{\mathrm{def}}{=} \mathit{Vars} \to \{0,1\}$. *For each* $a \in \mathcal{A}$, *each* $x \in \mathit{Vars}$, *and each* $c \in \{0,1\}$ *the valuation* $a[c/x] \in \mathcal{A}$ *is given, for each* $y \in \mathit{Vars}$, *by*

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

For $X = \{x_1, x_2, \dots\} \subseteq \mathit{Vars}$, *we write* $a[c/X]$ *for* $a[c/x_1][c/x_2]\cdots$.

Definition 95 (Boolean functions.) *The set of* Boolean function over *Vars is* $\mathcal{F} \stackrel{\text{def}}{=} \mathcal{A} \to \{0,1\}$*. The distinguished elements* $\top, \bot \in \mathcal{F}$ *are the functions defined by* $\top \stackrel{\text{def}}{=} \lambda a \in \mathcal{A} \,.\, 1$ *and* $\bot \stackrel{\text{def}}{=} \lambda a \in \mathcal{A} \,.\, 0$*. For* $f \in \mathcal{F}$*,* $x \in \mathit{Vars}$*, and* $c \in \{0,1\}$*, the function* $f[c/x] \in \mathcal{F}$ *is given, for each* $a \in \mathcal{A}$*, by* $f[c/x](a) \stackrel{\text{def}}{=} f\bigl(a[c/x]\bigr)$*. When* $X \subseteq \mathit{Vars}$*,* $f[c/X]$ *is defined in the obvious way.*

The question whether a Boolean function f forces a particular variable x to be true (which is what, in the context of groundness analysis, we call *sure groundness information*) is equivalent to the question whether $f \to x$ is a tautology (namely, $f \to x = \top$). In what follows we will also need the notion of *dependent variables* of a function.

Definition 96 (Dependent and true variables.) *For* $f \in \mathcal{F}$*, the set of* variables on which f depends *and the set of* variables necessarily true for f *are given, respectively, by*

$$\begin{aligned}
\mathit{vars}(f) &\stackrel{\text{def}}{=} \bigl\{\, x \in \mathit{Vars} \bigm| \exists a \in \mathcal{A} \,.\, f\bigl(a[0/x]\bigr) \neq f\bigl(a[1/x]\bigr) \,\bigr\},\\
\mathit{true}(f) &\stackrel{\text{def}}{=} \bigl\{\, x \in \mathit{vars}(f) \bigm| \forall a \in \mathcal{A} : f(a) = 1 \implies a(x) = 1 \,\bigr\}.
\end{aligned}$$

Two classes of Boolean functions which are suitable for groundness analysis are known under the names of *Def* and *Pos* (see [AMSS] for details). *Pos* consists precisely of those functions assuming the true value under the *everything-is-true* assignment (i.e., $f \in \mathit{Pos}$ if and only if $f \in \mathcal{F}$ and $f[1/\mathit{Vars}] = \top$). *Pos* is strictly more precise than *Def* for groundness analysis [AMSS]. The reason is that the elements of *Pos* allow to maintain disjunctive information which is, instead, lost in *Def*.

6.3 Combination of Domains and Reactivity

It is well-known that different data-flow analyses can be combined together. In the framework of abstract interpretation this can be achieved by means of standard constructions such as reduced product and down-set completion [CC79, CC92a]. The key point is that the combined analysis can be more precise than each of the component ones for they can mutually improve each other. However, the degree of cross-fertilization is highly dependent on the degree and quality of interaction taking place among the component domains. For the limited purpose of this discussion, when we talk about *combination of domains* we refer to the following situation: we have several distinct (both conceptually and at the implementation level) analysis' domains and, for the sake of ensuring correctness or improving precision, there must be a flow of information between them. This can be formalized in different ways. A methodology for the combination of abstract domains has been proposed in [CLV94]. It is based onto low level actions such as *tests* and *queries*. Basically, the component domains have the ability of querying other domains for some kind of information. Of course, they must also be able to respond to queries from other domains. For instance, the operations of a domain for numerical information might ask a domain for groundness whether a certain variable is guaranteed to be ground or not. Another way of describing this kind of interaction is the one proposed in [Bag97]. Here the interaction among domains is asynchronous in that it can occur at any time, or, in other words, it is not synchronized with the domain's operations. This is achieved by considering so called *ask-and-tell constraint systems* built over *product constraint systems*. These constraint systems allow to express communication among domains in a very simple way. They also inherit all the semantic elegance of concurrent constraint programming languages [Sar93], which provide the basis for their construction. We will now see, staying on an intuitive level and following the approach of [Bag97] for simplicity, examples of how these combinations look like.

In the CLP($\mathcal{R}$) system [JMSY92b] non-linear constraints (like $X = Y*Z$) are delayed (i.e., not treated by the constraint solver) until they become linear (e.g., until either Y or Z are constrained to take a single value). Obviously, this cannot be forgotten in abstract constraint systems intended to formalize correct data-flow analyses of CLP($\mathcal{R}$). When the abstract constraint system is able to extract numerical information from non-linear constraints (such as the one proposed in [BGL93]), the abstraction $\alpha_N(X = Y * Z)$ cannot be used without considering the groundness of Y and Z. By doing this you would incur the risk of *overshooting* the concrete constraint system (thus loosing soundness), which is unable to deduce anything from non-linear constraints. The right thing to do is to combine the numeric abstract constraint system with one for groundness and using, instead of the abstraction $\alpha_N(X = Y * Z)$, the agent

$$A \stackrel{\text{def}}{=} \text{ask}\bigl(\mathit{ground}(Y); \mathit{ground}(Z)\bigr) \rightarrow \alpha_N(X = Y * Z). \tag{6.1}$$

The intuitive reading is that the abstract constraint system is not allowed to do anything with $X = Y * Z$ until Y *or* (this is the intuitive reading of the semicolon) Z are ground. In this way, all the abstractions of non-linear constraints are "disabled" until their wake-up conditions are met (in the abstract, which, given a sound groundness analysis, implies that these conditions are met also at the concrete level). The need for interaction between groundness and numerical domains does not end here. Consider again the constraint $X = Y * Z$: clearly X is definite if Y and Z are so. But we cannot conclude that the groundness of Y follows from the one of X and Z, as we need also the condition $Z \neq 0$. Similarly, we would like to conclude that X is definite if Y or Z have a zero value. Thus we need approximations of the concrete values of variables (i.e., bounds analysis), something which is not captured by common groundness analyses while being crucial when dealing with non-linear constraints. In the approach of [Bag97] $X = Y * Z$ would be *abstractly compiled* into an agent of the form[1]

$$\begin{aligned}
A \parallel\ &\mathrm{ask}\bigl(\mathit{ground}(Y) \wedge \mathit{ground}(Z)\bigr) \to \mathrm{tell}\bigl(\mathit{ground}(X)\bigr)\\
&\parallel \mathrm{ask}(Y = 0; Z = 0) \to \mathrm{tell}\bigl(\mathit{ground}(X)\bigr)\\
&\parallel \mathrm{ask}\bigl(\mathit{ground}(X) \wedge \mathit{ground}(Z) \wedge Z \neq 0\bigr) \to \mathrm{tell}\bigl(\mathit{ground}(Y)\bigr)\\
&\parallel \mathrm{ask}\bigl(\mathit{ground}(X) \wedge \mathit{ground}(Y) \wedge Y \neq 0\bigr) \to \mathrm{tell}\bigl(\mathit{ground}(Z)\bigr),
\end{aligned}$$

where A is the agent given in Equation (6.1). Of course, this is much more precise than the *Pos* formula $X \leftarrow Y \wedge Z$, which is all you can say about the groundness dependencies of $X = Y * Z$ if you do not have any numerical information. It is clear from these examples that, when analyzing CLP($\mathcal{R}$) programs there is a bidirectional flow of information: groundness information is required for a correct handling of delayed constraints and thus for deriving more precise numerical patterns which, in turn, are used to provide more precise groundness information. Indeed, we are requiring a quite complicated interaction between domains.

Another application of groundness analysis with fast access to ground variables is for *aliasing analysis.* The most popular domain for this kind of analysis is *Sharing* [JL89]. Without going into details, its strength over the previous approaches [JS87, Deb89] comes from the fact that it keeps track of groundness dependencies. In fact, *Sharing* has, as far as groundness information is concerned, the same power of *Def.* When *Pos* is used for groundness, using *Sharing* for aliasing at the same time is a waste: *Sharing* spends time and space for keeping track of groundness, which is already done, and more precisely, by *Pos.* A possible solution is to adopt a variation of the domains proposed in [JS87, Deb89] (which are much less computationally expensive than *Sharing*) and to combine it with *Pos.* We are currently working along this line. This, however, is beyond the scope of this work.

Whatever conceptual methodology you follow to realize the combination of any domain with one for groundness, a key component for the efficiency is that the implementation of the latter must be *reactive.* By this we mean that: (a) it must react quickly to external queries about the groundness of variables; and, (b) it must absorb quickly groundness notifications coming from other domains.

[1]We choose this form of presentation for clarity. It is clear that this agent will be itself compiled to something different. For instance, the second agent of the parallel composition will "live" in the groundness component, if the latter is able to capture the indicated dependency.

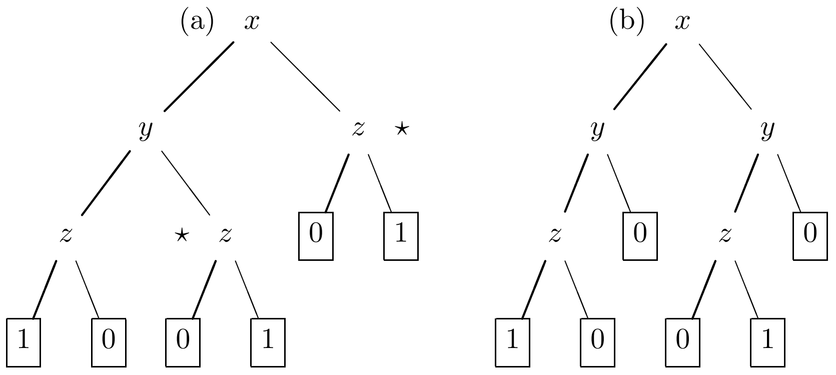


Figure 6.1: OBDTs for $(x \wedge y) \leftrightarrow z$ (a) and $(x \leftrightarrow z) \wedge y$ (b).

6.4 Binary Decision Trees and Diagrams

Binary decision trees (BDTs) and diagrams (BDDs) are well-known representations of Boolean functions [Bry86, Bry92]. Binary decision trees, such as the ones presented in Fig. 6.1 are binary trees where non-terminal nodes are labeled with variable names, while terminal nodes are labeled with the Boolean constants 0 or 1. The value of the represented function, for a given assignment of Boolean values to variables, can be recovered by following a particular path from the root: at any non-terminal node labeled with a variable v, the thick branch is taken if v is assigned to 1, otherwise the thin branch is taken. The terminal node reached by this walk on the tree is the function value. For a non-terminal node n, we will call the node connected to n by means of the thick (resp. thin) edge the *true* (resp. *false*) *successor of n*. A BDD is a directed acyclic graph which can be thought of as obtained from a BDT by collapsing identical subtrees. With reference to Fig. 6.1 (a), the subtrees marked with '$\star$' can be collapsed, as well as all the terminal nodes having the same label. The action of collapsing identical subtrees does not change the represented function. Given a total ordering on the variable symbols, an *ordered binary decision tree* (OBDT) is a BDT where the sequence of variables (associated to non-terminals) encountered in any path from the root is strictly increasing. The trees depicted in Fig. 6.1 are indeed OBDTs where the ordering is such that $x \prec y \prec z$. Applying the very same restriction to BDDs we obtain the notion of *ordered binary decision diagram*, or OBDD.

Definition 97 (BDTs and OBDTs.) *A* binary decision tree *is any string generated by the grammar*

$$\mathrm{BDT} ::= \mathbf{0} \mid \mathbf{1} \mid \mathbf{ite}(v, \mathrm{BDT}, \mathrm{BDT})$$

where $v \in Vars$. The set of all BDTs is denoted by $\mathcal{B}$. The semantics of

BDTs is expressed by the function $[\![\cdot]\!] \colon \mathcal{B} \to \mathcal{F}$*, defined as follows:*

$$[\![\mathbf{0}]\!] \stackrel{\mathrm{def}}{=} \bot, \qquad [\![\mathbf{1}]\!] \stackrel{\mathrm{def}}{=} \top, \qquad [\![\mathbf{ite}(v, b_1, b_0)]\!] \stackrel{\mathrm{def}}{=} \mathit{ite}\bigl(v, [\![b_1]\!], [\![b_0]\!]\bigr),$$

where for each $w \in \mathit{Vars}$*,* $f_1, f_0 \in \mathcal{F}$*, and each* $a \in \mathcal{A}$*,*

$$\mathit{ite}(w, f_1, f_0)(a) \stackrel{\mathrm{def}}{=} \begin{cases} f_1(a), & \text{if } a(w) = 1; \\ f_0(a), & \text{if } a(w) = 0. \end{cases}$$

The subset $\mathcal{B}_o \subseteq \mathcal{B}$ *of* ordered BDTs (OBDTs) *is defined by the following recurrent equation:*

$$\mathcal{B}_o \stackrel{\mathrm{def}}{=} \{\mathbf{0}, \mathbf{1}\} \cup \bigl\{\, \mathbf{ite}(v, b_1, b_0) \bigm| v \in \mathit{Vars}, \quad b_1, b_0 \in \{\mathbf{0}, \mathbf{1}\} \,\bigr\}$$
$$\cup \left\{ \mathbf{ite}(v, b_1, b_0) \;\middle|\; \begin{array}{l} \forall i = 0, 1 : b_i \in \mathcal{B}_o \wedge \\ \quad \exists w \in \mathit{Vars} \,.\, \exists b'_1, b'_0 \in \mathcal{B}_o \,. \\ \qquad b_i = \mathbf{ite}(w, b'_1, b'_0) \Rightarrow v \prec w \end{array} \right\}.$$

We will deliberately confuse a BDT with the Boolean function it represents. In particular, for $b \in \mathcal{B}$, when we write $\mathit{vars}(b)$ or $\mathit{true}(b)$ what we really mean is $\mathit{vars}([\![b]\!])$ or $\mathit{true}([\![b]\!])$. This convention of referring to the semantics simplifies the presentation and should not cause problems.

A *reduced ordered binary decision diagram*, or ROBDD, is an OBDD such that:

1. there are no duplicate terminal nodes;
2. there are no duplicate non-terminal nodes (i.e., nodes having the same label and the same true and false successors);
3. there are no redundant tests, that is each non-terminal node has distinct true and false successors.

Any OBDD can be converted into a ROBDD by repeatedly applying the reduction rules corresponding to the above properties: collapsing all the duplicate nodes into one and removing all the redundant tests, redirecting edges in the obvious way. Application of these rules does not change the represented functions. ROBDDs have one very important property: they are *canonical*. This means that, for each fixed variables' ordering, two ROBDDs represent the same function if and only if they are identical.

The nice computational features of ROBDDs make them suitable for implementing *Pos* (see, e.g., [CH93, AMSS]), even though ROBDDs are clearly able to represent any Boolean function. Here we deal formally only with OBDTs, since our results do not need all the properties of ROBDDs. Indeed, since every OBDT is an OBDD and the reduction rules do not change the represented Boolean function, everything we say about OBDTs is true also for ROBDDs.

6.5 Is x Ground?

Capturing dependency and disjunctive information is essential for precise groundness analysis. However, this kind of information is only needed for maintaining precise intermediate results *during* the analysis. Instead, the only information that is relevant for the user of the analysis' results is whether a certain variable is guaranteed to be ground at a certain point or not. When combinations of domains are considered, as explained in Section 6.3, it is vital to recover the set of ground variables quickly even during the analysis *itself*. The problem of deriving this sure groundness information from ROBDDS has not been tackled in previous works [CFW91, CH93, AMSS]. Basically we know about five ways of doing that:

1. Given $x \in \mathit{Vars}$ and a ROBDD representation b of a Boolean function f, use the ROBDD package to test whether $f \to x$ is a tautology, that is, whether $f \to x$ is equivalent to $\top$. This test can be performed in $O(|b|)$ time. The main advantage of this solution is that it does not require any change to the ROBDD package. One of the drawbacks is that the reduction of $f \to x$ causes the creation and disposal of "spurious" nodes which must be dealt with by the system[2].

2. Given $x \in \mathit{Vars}$ and a ROBDD representation b of a Boolean function f, the information whether x is forced to 1 by f is obtained by visiting b. The answer is affirmative if (a) there is at least one node in b labeled with x; and, (b) each node in b labeled with x has its false branch equal to $\mathbf{0}$. This method has still linear complexity, requires the incorporation of the visit into the ROBDD package, and it is *read-only* in that is does not involve the creation of any node.

3. Another possibility is to visit the ROBDD representation b to derive, in one step, the set G of *all* the variables which are forced to 1. We will see how this can be done in Section 6.6.

4. A variation of the previous method consists in avoiding visiting b, while obtaining exactly the same information, by modifying ROBDD's nodes so that every node records the set of variables which are forced to true by the Boolean function it represents. Section 6.6 explains how this method of keeping explicit the information about true variables can be easily implemented.

5. The last method is based on a quite radical, though very simple, solution. Intuitively, it is based on the idea of keeping the information about true variables totally separate from dependency and disjunctive information. True variables are represented naturally by means of

[2]This is due to technical details which are vital for efficient and realistic implementations. This is, however, beyond the scope of this work.

sets whereas only the dependency and disjunctive information is maintained by means of ROBDDs. This will be explained in Section 6.7.

6.6 Extracting Sure Groundness from ROBDDs

Here is the only property of OBDTs (and thus of ROBDDs) we need.

Definition 98 (Weak normal form.) *A BDT $b \in \mathcal{B}$ is said to be* in weak normal form *if and only if either $b = \mathbf{0}$ or $b = \mathbf{1}$, or there exist $b_1, b_0 \in \mathcal{B}$ such that $b = \mathbf{ite}(v, b_1, b_0)$, $v \notin \mathit{vars}(b_1) \cup \mathit{vars}(b_0)$, and both b_1 and b_0 are in weak normal form.*

Proposition 99 *Each OBDT $b \in \mathcal{B}_o$ is in weak normal form.*

Proof By induction on the structure of b: the base cases $b = \mathbf{0}$ and $b = \mathbf{1}$ are trivial. If $b = \mathbf{ite}(v, b_1, b_0)$ then b_1 and b_0 are members of $\mathcal{B}_o$, and thus are in weak normal form by the induction hypothesis. Moreover, by the definition of $\mathcal{B}_o$, v does not occur neither in b_1 nor in b_0. But then the definition of $[\![\cdot]\!]$ ensures that, for each $a \in \mathcal{A}$, $[\![b_1]\!][0/v] = [\![b_1]\!][1/v]$ and likewise for b_0. The thesis now follows. $\square$

This is indeed the distinctive property of "free BDDs", also called "1-time branching programs", where no ordering is required but each path from the root is allowed to test a variable only once [Bry92].

Theorem 100 *Let $b = \mathbf{ite}(v, b_1, b_0)$ be in weak normal form. Then we have*

$$\mathit{true}(b) = \begin{cases} \mathit{true}(b_0), & \text{if } b_1 = \mathbf{0}; \\ \{v\} \cup \mathit{true}(b_1), & \text{if } b_1 \neq \mathbf{0} \text{ and } b_0 = \mathbf{0}; \\ \mathit{true}(b_1) \cap \mathit{true}(b_0), & \text{otherwise.} \end{cases}$$

Lemma 101 *Let $b = \mathbf{ite}(v, b_1, b_0)$ in weak normal form. Then $\mathit{vars}(b) = V \cup \mathit{vars}(b_1) \cup \mathit{vars}(b_0)$, where*

$$V \stackrel{\text{def}}{=} \begin{cases} \{v\} & \text{if } b_1 \neq b_0, \\ \varnothing & \text{otherwise.} \end{cases}$$

Proof Let $x \in \mathit{vars}(b)$, namely, there exists $a \in \mathcal{A}$ such that $b\bigl(a[0/x]\bigr) \neq b\bigl(a[1/x]\bigr)$. If $x = v$ then $b_1 \neq b_0$ and $x \in V$. Otherwise, there are still two cases: either $a(v) = 1$ or $a(v) = 0$. In the first case, $b\bigl(a[0/x]\bigr) = b_1\bigl(a[0/x]\bigr)$ and $b\bigl(a[1/x]\bigr) = b_1\bigl(a[1/x]\bigr)$, thus $x \in \mathit{vars}(b_1)$. The second case is handled symmetrically, yielding the conclusion $x \in \mathit{vars}(b_0)$. We have thus shown that $\mathit{vars}(b) \subseteq V \cup \mathit{vars}(b_1) \cup \mathit{vars}(b_0)$.

We now prove that the reverse inclusion also holds. Let us consider $x \in \mathit{vars}(b_1)$. Since b is in weak normal form, it must be $x \neq v$. Take $a \in \mathcal{A}$ such that $b_1\bigl(a[0/x]\bigr) \neq b_1\bigl(a[1/x]\bigr)$, and let $b' = b[1/w]$. Then, clearly, $b'\bigl(a[0/x]\bigr) \neq b'\bigl(a[1/x]\bigr)$ and $x \in \mathit{vars}(b)$. Symmetrically, taking $x \in \mathit{vars}(b_0)$ we obtain again that $x \in \mathit{vars}(b)$. If we suppose that $x \in V \cup \mathit{vars}(b_1) \cup \mathit{vars}(b_0)$ but $x \notin \mathit{vars}(b_1) \cup \mathit{vars}(b_0)$, then we have $x = v$ and $b_1 \neq b_0$. So, there exists $a \in \mathcal{A}$ such that $b_1(a) \neq b_0(a)$. But then $b\bigl(a[1/x]\bigr) = b_1\bigl(a[1/x]\bigr) = b_1(a)$ and $b\bigl(a[0/x]\bigr) = b_0\bigl(a[0/x]\bigr) = b_0(a)$, thus $b\bigl(a[0/x]\bigr) \neq b\bigl(a[1/x]\bigr)$ and $x \in \mathit{vars}(b)$. □

PROOF of Theorem 100. First of all, let us restate the claim in an equivalent, though more convenient way. Let $b = \mathbf{ite}(v, b_1, b_0)$ be in weak normal form. The theorem says that

$$\mathit{true}(b) = V \cup V_1 \cup V_0 \cup V_{10},$$

where

$$V \stackrel{\text{def}}{=} \begin{cases} \{V\} & \text{if } b_1 \neq \mathbf{0} \text{ and } b_0 = \mathbf{0},\\ \varnothing & \text{otherwise;} \end{cases}$$
$$V_1 \stackrel{\text{def}}{=} \begin{cases} \mathit{true}(b_1) & \text{if } b_0 = \mathbf{0},\\ \varnothing & \text{otherwise;} \end{cases}$$
$$V_0 \stackrel{\text{def}}{=} \begin{cases} \mathit{true}(b_0) & \text{if } b_1 = \mathbf{0},\\ \varnothing & \text{otherwise;} \end{cases}$$
$$V_{10} \stackrel{\text{def}}{=} \mathit{true}(b_1) \cap \mathit{true}(b_0).$$

The following table shows that V, V_1, V_0, and V_{10} are pair-wise disjoint.

$b_1 \in$	$b_0 \in$	$V =$	$V_1 =$	$V_0 =$	$V_{10} =$
$\{\mathbf{0}\}$	$\{\mathbf{0}\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
$\mathcal{B} \setminus \{\mathbf{0}\}$	$\{\mathbf{0}\}$	$\{v\}$	$\mathit{true}(b_1)$	$\varnothing$	$\varnothing$
$\{\mathbf{0}\}$	$\mathcal{B} \setminus \{\mathbf{0}\}$	$\varnothing$	$\varnothing$	$\mathit{true}(b_0)$	$\varnothing$
$\mathcal{B} \setminus \{\mathbf{0}\}$	$\mathcal{B} \setminus \{\mathbf{0}\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\mathit{true}(b_1) \cap \mathit{true}(b_0)$

In particular, V and V_1 are disjoint due to the fact that b is in weak normal form. We now show that $V \cup V_1 \cup V_0 \cup V_{10} \subseteq \mathit{true}(b)$. If we suppose that $x \in V \cup V_1 \cup V_0 \cup V_{10}$, the following cases are mutually exclusive, and all imply that $x \in \mathit{true}(b)$.

$x \in V$**:** this means $x = v$, $b_1 \neq \mathbf{0}$, and $b_0 = \mathbf{0}$. Since $b_1 \neq b_0$ we have, by Lemma 101, $x \in \mathit{vars}(b)$. Also, if $a \in \mathcal{A}$ is such that $b(a) = 1$ it must be $a(x) = 1$.

$x \in V_1$: so $b_1 \neq \mathbf{0}$ and $b_0 = \mathbf{0}$. From Lemma 101 we have that $x \in \mathit{true}(b_1) \subseteq \mathit{vars}(b_1) \subseteq \mathit{vars}(b)$. Now, if $a \in \mathcal{A}$ is such that $b(a) = 1$, it must be $b_1(a) = 1$, but, since $x \in V_1 = \mathit{true}(b_1)$, this implies $b(x) = 1$.

$x \in V_0$: this case is symmetric to the previous one.

$x \in V_{10}$: that is $x \in \mathit{true}(b_1) \cap \mathit{true}(b_0)$, $b_1 \neq 0$, $b_0 \neq 0$, and $x \neq v$. In particular, $x \in \mathit{true}(b_1) \subseteq \mathit{vars}(b_1) \subseteq \mathit{vars}(b)$. Let $a \in \mathcal{A}$ such that $b(a) = 1$. If $a(v) = 1$, then $b(a) = b_1(a) = 1$ and thus $a(x) = 1$, since $x \in \mathit{true}(b_1)$. If, instead, $a(v) = 0$, then $b(a) = b_0(a) = 1$ and so $a(x) = 1$, since $x \in \mathit{true}(b_0)$.

It remains to be shown that $\mathit{true}(b) \subseteq V \cup V_1 \cup V_0 \cup V_{10}$. We assume that $x \in \mathit{true}(b)$, namely that $x \in \mathit{vars}(b)$ and

$$\forall a \in \mathcal{A} : \bigl(b(a) = 1 \implies a(x) = 1\bigr).$$

Observe that the existence of such an x ensures that b_1 and b_0 cannot be both equal to $\mathbf{0}$. We distinguish the following cases:

$x = v$: this means $b_1 \neq \mathbf{0}$ and $b_0 = \mathbf{0}$, that is $x \in V$.

$x \neq v$: this case is further split as follows:

$b_1 = \mathbf{0}$: for $a \in \mathcal{A}$,

$$\begin{aligned} b_0(a) = 1 &\implies b\bigl(a[0/v]\bigr) = 1 \\ &\implies a[0/w](x) = 1 \\ &\implies a(x) = 1, \end{aligned}$$

and thus, $\forall a \in \mathcal{A} : b_0(a) = 1 \Rightarrow a(x) = 1$. By Lemma 101 $x \in \{v\} \cup \mathit{vars}(b_1) \cup \mathit{vars}(b_0)$. Since $x \neq v$ and $b_1 = \mathbf{0}$, this implies $x \in \mathit{vars}(b_0)$. In conclusion, $x \in \mathit{true}(b_0) = V_0$.

$b_0 = \mathbf{0}$: handled symmetrically.

$b_1 \neq \mathbf{0}$ and $b_0 \neq \mathbf{0}$: observe that, for each $a \in \mathcal{A}$, $b(a) = 1$ is equivalent to

$$\bigl(a(v) = 1 \land b_1(a) = 1\bigr) \lor \bigl(a(v) = 0 \land b_0(a) = 1\bigr).$$

Suppose, towards a contradiction, that

$$\exists a \in \mathcal{A} \;.\; b_1(a) = 1 \land a(x) = 0.$$

We would have $b\bigl(a[1/v]\bigr) = 1$ and $a[1/v](x) = 0$ which is in contrast with the assumption $x \in \mathit{true}(b)$. Thus,

$$\forall a \in \mathcal{A} : \bigl(b_1(a) = 1 \implies a(x) = 1\bigr).$$

Now suppose, *ad absurdum*, that $x \notin \mathit{vars}(b_1)$, namely

$$\forall a \in \mathcal{A} : b_1\bigl(a[1/x]\bigr) = b_1\bigl(a[0/x]\bigr).$$

Let $a \in \mathcal{A}$ such that $b(a) = 1$ and $b(v) = 1$ (such an a must exist, since $b_1 \neq \mathbf{0}$). Now,

$$b(a) = b_1(a) = b_1\bigl(a[0/x]\bigr) = b\bigl(a[0/x]\bigr) = 1,$$

but, clearly, $a[0/x](x) = 0$ which contradicts the assumption $x \in \mathit{true}(b)$. We can thus conclude that $x \in \mathit{vars}(b_1)$ and, by the previous reasoning, that $x \in \mathit{true}(b_1)$. A symmetric argument proves that also $x \in \mathit{true}(b_0)$ and thus $x \in V_{10}$. □

This theorem gives us at least two ways of deriving sure groundness information from ROBDDs. One is by implementing a bottom-up visit, collecting true variables as indicated. Another one, which is more in the spirit of a reactive implementation, is based on a modification of the node structure which is used to represent ROBDDs. In standard implementations, a non-terminal node n has one field $n.V$ which holds the test variable, plus two fields $n.T$ and $n.F$ which are references to the nodes which are the roots of the true and false branch, respectively. All the nodes are created by means of a function $\mathit{create}(v, @n_1, @n_0)$, taking a variable symbol and two references to (already created) nodes, and returning a reference to the newly created node. We can modify this state of things by adding to the node structure a field $n.G$, containing the set of true variables for the function represented by the ROBDD rooted at n, and by modifying the creation function to initialize $n.G$ as indicated by Theorem 100.

6.7 A New, Hybrid Implementation for *Pos*

The observation of many constraint logic programs shows that the percentage of variables which are found to be ground during the analysis, for typical invocations, is as high as 80%. This suggests that representing *Pos* elements simply by means of ROBDDs, as in [CH93, AMSS], is probably not the best thing we can do. Here we propose a hybrid implementation where each *Pos* element is represented by a pair: the first component is the set of true variables (just as in the domain used in early groundness analyzers [Mel85, JS87]); the second component is a ROBDD. In each element of this new representation there is no redundancy: the ROBDD component does not contain any information about true variables. In fact, as we will see, the hybrid representation has the property that ROBDDs are used only in what they are good for: keeping track of dependencies and disjunctive information. True variables, instead are more efficiently represented by means

of sets. The hybrid representation has two major advantages: (a) it is *reactive* in the sense of Section 6.3; and, (b) it allows for keeping the ROBDDs small, during the analysis, when many variables come out to be true, as it is often the case. Consider Fig. 6.1 (b): the information about y being a true variable (besides not being readily available) requires two nodes. In more involved cases, the information about trueness of a variable coming late in the ordering can be scattered over a large number of nodes. Notice that, while having many true variables, in a straight ROBDD implementation, means that the *final* ROBDDs will be very similar to a linear chain of nodes, the intermediate steps still require the creation (and disposal) of complex (and costly) ROBDDs. This phenomenon is avoided as much as possible in the hybrid implementation.
(By $\wp_{\mathrm{f}}(\mathit{Vars})$ we denote the set of all *finite* subsets of *Vars*.)

Definition 102 (Hybrid representation.) *The* hybrid representation *for Pos is*

$$\mathcal{G} \stackrel{\mathrm{def}}{=} \bigl\{\, \langle G, b\rangle \bigm| G \in \wp_{\mathrm{f}}(\mathit{Vars}), b \in \mathcal{B}_o, \mathit{vars}(b) \cap G = \mathit{true}(b) = \varnothing \,\bigr\}.$$

The meaning of $\mathcal{G}$'s elements is given by the overloading $[\![\cdot]\!] \colon \mathcal{G} \to \mathcal{F}$:

$$\bigl[\!\bigl[\langle G, b\rangle\bigr]\!\bigr] \stackrel{\mathrm{def}}{=} \bigwedge(G) \wedge [\![b]\!],$$

where $\bigwedge\{x_1, \ldots, x_n\} \stackrel{\mathrm{def}}{=} x_1 \wedge \cdots \wedge x_n$ *and* $\bigwedge \varnothing \stackrel{\mathrm{def}}{=} \top$.

Now, we briefly review the operations we need over *Pos* (and thus over $\mathcal{G}$) for the purpose of groundness analysis. The constraint accumulation process requires computing the logical conjunction of two functions, the merge over different computation paths amounts to logical disjunction, whereas projection onto a designated set of variables is handled through existential quantification. Functions of the kind $x \leftrightarrow (y_1, \ldots, y_m)$, for $m \geq 0$, accommodate both abstract *mgu*s and the combination operation in domains like $\mathtt{Pat}(\mathit{Pos})$ [CLV94].

Before introducing the $\mathcal{G}$'s operations we introduce, by means of Table 6.1, the needed operations over OBDTs and ROBDDs, their complexity and semantics, as well as the correspondent operations over $\mathcal{G}$. In what follows we will refer to some operations on OBDTs whose meaning and complexity is specified in the table. The restriction operation $b[1/V]$ (also called *valuation* or *co-factoring*) is used for maintaining the invariant specified in Definition 102. In the definition of the abstract operators used in groundness analysis, the functions of the form $x \leftrightarrow (y_1, \ldots, y_m)$ are always *conjuncted* with some other function. For this reason we provide a family of specialized operations $(x, V) \overset{\leftrightarrow}{\otimes} \colon \mathcal{G} \to \mathcal{G}$, indexed over variables and finite sets of variables. The operation $(x, V) \overset{\leftrightarrow}{\otimes}$ builds a representation for $\bigl(x \leftrightarrow \bigwedge(V)\bigr) \wedge f$, given one for f.

$\mathcal{B}_o$ op	Complexity	Meaning	$\mathcal{G}$ op	
$b_1 \ddot{\wedge} b_2$	$O(\lvert b_1\rvert \lvert b_2\rvert)$	$[\![b_1]\!] \wedge [\![b_2]\!]$	$g_1 \otimes g_2$	
$b_1 \ddot{\vee} b_2$	$O(\lvert b_1\rvert \lvert b_2\rvert)$	$[\![b_1]\!] \vee [\![b_2]\!]$	$g_1 \oplus g_2$	
$\ddot{\exists}_{\overline{V}}\, b$	$O\big(\lvert b\rvert^{2^{\lvert V\rvert}}\big)$	$\exists_{\overline{V}}[\![b]\!]$	$\exists_{\overline{V}}\, g$	Note: $V \subseteq vars(b)$.
$\ddot{\bigwedge}(V)$	$O(\lvert V\rvert)$	$\bigwedge(V)$		Note: $V \neq \varnothing$.
$x \ddot{\leftrightarrow} V$	$O(\lvert V\rvert)$	$x \leftrightarrow \bigwedge(V)$		Note: $x \notin V$.
$b[1/V]$	$O(\lvert b\rvert)$	$[\![b]\!][1/V]$		

Table 6.1: Operations defined over $\mathcal{B}_o$ and $\mathcal{G}$.

Definition 103 (Operations over $\mathcal{G}$.) *The operation $\otimes\colon \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ is defined, for each $\langle G_1, b_1\rangle, \langle G_2, b_2\rangle \in \mathcal{G}$, as follows:*

$$\langle G_1, b_1\rangle \otimes \langle G_2, b_2\rangle \stackrel{\text{def}}{=} \eta\Big(G_1 \cup G_2, b_1\big[1/(G_2 \setminus G_1)\big] \ddot{\wedge} b_2\big[1/(G_1 \setminus G_2)\big]\Big),$$

where, for each $G \in \wp_{\mathrm{f}}(Vars)$ and $b \in \mathcal{B}_o$ such that $G \cap vars(b) = \varnothing$,

$$\eta(G, b) \stackrel{\text{def}}{=} \begin{cases} \langle G, b\rangle, & \text{if } true(b) = \varnothing; \\ \eta\Big(G \cup true(b), b\big[1/\, true(b)\big]\Big), & \text{otherwise.} \end{cases} \tag{6.2}$$

The join operation $\oplus\colon \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ is given by

$$\langle G_1, b_1\rangle \oplus \langle G_2, b_2\rangle \stackrel{\text{def}}{=} \Big\langle G_1 \cap G_2, \big(b_1 \ddot{\wedge} \ddot{\bigwedge}(G_1 \setminus G_2)\big) \ddot{\vee} \big(b_2 \ddot{\wedge} \ddot{\bigwedge}(G_2 \setminus G_1)\big)\Big\rangle.$$

For each $\langle G, b\rangle \in \mathcal{G}$, each $V \in \wp_{\mathrm{f}}(Vars)$, and $x \in Vars$, the unary operations $\exists_{\overline{V}}\colon \mathcal{G} \to \mathcal{G}$ and $(x, V)\overset{\leftrightarrow}{\otimes}\colon \mathcal{G} \to \mathcal{G}$ are given by

$$\exists_{\overline{V}}\langle G, b\rangle \stackrel{\text{def}}{=} \big\langle G \setminus V, \ddot{\exists}_{\overline{V}}\, b\big\rangle$$

and

$$(x, V)\overset{\leftrightarrow}{\otimes}\langle G, b\rangle \stackrel{\text{def}}{=} \begin{cases} \eta\Big(G \cup V, b\big[1/(V \setminus G)\big]\Big), & \text{if } x \in G; \\ \eta\big(G \cup \{x\}, b[1/x]\big), & \text{if } V \subseteq G; \\ \eta\Big(G, b \ddot{\wedge} \big(x \ddot{\leftrightarrow} (V \setminus G)\big)\Big), & \text{if } x \notin G \text{ and } V \nsubseteq G. \end{cases}$$

The following result holds almost by definition.

Theorem 104 *The operations of* Definition 103 *are well-defined. Moreover, for each $g, g_1, g_2 \in \mathcal{G}$, each $V \in \wp_f(\mathit{Vars})$, and $x \in \mathit{Vars}$,*

$$\begin{aligned} [\![g_1 \otimes g_2]\!] &= [\![g_1]\!] \wedge [\![g_2]\!], & [\![g_1 \oplus g_2]\!] &= [\![g_1]\!] \vee [\![g_2]\!], \\ \Big[\!\!\Big[\exists_{\overline{V}}\, g\Big]\!\!\Big] &= \exists_{\overline{V}} [\![g]\!], & \Big[\!\!\Big[(x, V) \overset{\leftrightarrow}{\otimes} g\Big]\!\!\Big] &= \big(x \leftrightarrow \textstyle\bigwedge(V)\big) \wedge [\![g]\!]. \end{aligned}$$

Definition 105 (Restriction operation.) *For each $b \in \mathcal{B}$, each $x \in \mathit{Vars}$, and each $c \in \{0,1\}$ the* restriction of b with respect to x and c *is the BDT $b[c/x]$ defined as follows:*

$$\begin{aligned} \mathbf{0}[c/x] &\stackrel{\text{def}}{=} \mathbf{0}; \\ \mathbf{1}[c/x] &\stackrel{\text{def}}{=} \mathbf{1}; \\ \mathbf{ite}(v, b_1, b_0)[c/x] &\stackrel{\text{def}}{=} \begin{cases} b_c[c/x], & \text{if } x = v; \\ \mathbf{ite}\big(v, b_1[c/x], b_0[c/x]\big), & \text{otherwise.} \end{cases} \end{aligned}$$

This restriction operation on BDTs is easily seen to be the syntactic counterpart of the operation on Boolean functions given in Definition 95.

Proposition 106 *For each $b \in \mathcal{B}$, each $x \in \mathit{Vars}$, and each $c \in \{0,1\}$,*

$$\Big[\!\!\Big[b[c/x]\Big]\!\!\Big] = [\![b]\!][c/x].$$

Proof By induction on the structure of b. The base cases $b = \mathbf{0}$ and $b = \mathbf{1}$ are immediate from definitions 95 and 105. For the induction step, suppose $b = \mathbf{ite}(v, b_1, b_0)$. If $x = v$ then

$$\begin{aligned} \Big[\!\!\Big[b[c/v]\Big]\!\!\Big] &= \Big[\!\!\Big[b_c[c/v]\Big]\!\!\Big] \\ &= [\![b_c]\!][c/v] \\ &= \mathit{ite}\big(v, [\![b_1]\!], [\![b_0]\!]\big)[c/v] \\ &= [\![b]\!][c/v]. \end{aligned}$$

If $x \neq v$ then

$$\begin{aligned} \Big[\!\!\Big[b[c/x]\Big]\!\!\Big] &= \Big[\!\!\Big[\mathbf{ite}(v, b_1[c/x], b_0[c/x])\Big]\!\!\Big] \\ &= \mathit{ite}\Big(v, \Big[\!\!\Big[b_1[c/x]\Big]\!\!\Big], \Big[\!\!\Big[b_0[c/x]\Big]\!\!\Big]\Big) \\ &= \mathit{ite}\big(v, [\![b_1]\!][c/x], [\![b_0]\!][c/x]\big) \\ &= \mathit{ite}\big(v, [\![b_1]\!], [\![b_0]\!]\big)[c/x] \\ &= [\![b]\!][c/x]. \end{aligned}$$

□

Lemma 107 *For each $b \in \mathcal{B}$, each $x \in$ Vars, and each $c \in \{0,1\}$ we have that $x \notin vars\bigl(b[c/x]\bigr)$.*

Proof Let $a \in \mathcal{A}$. Then

$$\begin{aligned}
\bigl[\!\bigl[b[c/x]\bigr]\!\bigr]\bigl(a[0/x]\bigr) &= [\![b]\!][c/x]\bigl(a[0/x]\bigr)\\
&= [\![b]\!]\bigl(a[0/x][c/x]\bigr)\\
&= [\![b]\!]\bigl(a[c/x]\bigr)\\
&= [\![b]\!]\bigl(a[1/x][c/x]\bigr)\\
&= [\![b]\!][c/x]\bigl(a[1/x]\bigr)\\
&= \bigl[\!\bigl[b[c/x]\bigr]\!\bigr]\bigl(a[1/x]\bigr). \qquad \square
\end{aligned}$$

PROOF of Theorem 104 (sketch). We first see that, if $\langle G, b\rangle$ is the result of one of the operations of Definition 103, then we have $true(b) = \varnothing$. We use Lemma 107 and other simple facts about Boolean functions. Of course, we also exploit the assumption that the semantics of OBDT operations is as given in Table 6.1. For the $\otimes$ and $(x, V)\overset{\leftrightarrow}{\otimes}$ operators this is true by the definition of η given in Equation (6.2). Also, $true\bigl(\exists_{\bar V} b\bigr) \subseteq true(b)$, which handles the case of the $\exists_{\bar V}$ operators. For the $\oplus$ operator, from

$$\begin{aligned}
true(b_1) = true(b_2) &= \varnothing,\\
vars(b_1) \cap G_1 = vars(b_2) \cap G_2 &= \varnothing,
\end{aligned}$$

we obtain

$$\begin{aligned}
true\bigl(b_1 \mathbin{\ddot\wedge} \ddot{\textstyle\bigwedge}(G_1 \setminus G_2)\bigr) &= G_1 \setminus G_2,\\
true\bigl(b_2 \mathbin{\ddot\wedge} \ddot{\textstyle\bigwedge}(G_2 \setminus G_1)\bigr) &= G_2 \setminus G_1.
\end{aligned}$$

Since, for each $b', b'' \in \mathcal{B}$, $true(b' \mathbin{\ddot\vee} b'') = true(b') \cap true(b'')$ the thesis follows.

Now we see that, if an operator gives back $\langle G, b\rangle$, then we have $G \cap vars(b) = \varnothing$. It is immediate from the definition that any non-recursive application $\eta(G, b)$ is such that $G \cap vars(b) = \varnothing$. Thus, by Lemma 107,

$$\begin{aligned}
&\bigl(G \cup true(b)\bigr) \cap vars\Bigl(b\bigl[1/\,true(b)\bigr]\Bigr)\\
&\qquad = \bigl(G \cup true(b)\bigr) \cap \bigl(vars(b) \setminus true(b)\bigr)\\
&\qquad = G \cap \bigl(vars(b) \setminus true(b)\bigr)\\
&\qquad = \varnothing,
\end{aligned}$$

and also the recursive applications of η enjoy the same property. This completes the proof for the $\otimes$ and $(x, V)\overset{\leftrightarrow}{\otimes}$ operators. For the $\oplus$ operator, just observe that

$$\mathit{vars}\Bigl(\bigl(b_1 \mathbin{\ddot{\wedge}} \ddot{\bigwedge}(G_1 \setminus G_2)\bigr) \mathbin{\ddot{\vee}} \bigl(b_2 \mathbin{\ddot{\wedge}} \ddot{\bigwedge}(G_2 \setminus G_1)\bigr)\Bigr) \subseteq \\ \mathit{vars}(b_1) \cup \mathit{vars}(b_2) \cup (G_1 \setminus G_2) \cup (G_2 \setminus G_1).$$

Then, since $\mathit{vars}\bigl(\exists_{\overline{V}}\, b\bigr) = \mathit{vars}(b) \cap V$ and $(G \setminus V) \cap \bigl(\mathit{vars}(b) \cap V\bigr) = \varnothing$ we have the thesis for $\exists_{\overline{V}}$.

The statement about the operations' semantics is routine. □

Notice that $\mathcal{G}$ operations make use of the $\mathcal{B}_o$ (ROBDD) operations only when strictly necessary. When this happens, expensive operations like $\ddot{\wedge}$ and $\ddot{\vee}$ are invoked with operands of the smallest possible size. In particular, we exploit the fact that the restriction operation is relatively cheap. However, we cannot avoid searching for true variables, as the $\otimes$ and $(x, V) \overset{\leftrightarrow}{\otimes}$ operators need that. For this purpose, the procedure implicit in Theorem 100 comes in handy. In programs where many variables are ground the ROBDDs generated will be kept small, and so also the cost of searching will be diminished. As a final remark, observe that in a real implementation the operations which are executed can be further optimized. Without entering into details, the basic analysis step, for what concerns groundness and in a bottom-up framework, generates *macro-operations* of the form

$$(x_1, V_1) \overset{\leftrightarrow}{\otimes} \bigl((x_2, V_2) \overset{\leftrightarrow}{\otimes} \bigl(\cdots((x_n, V_n) \overset{\leftrightarrow}{\otimes} (g_1 \otimes g_2 \otimes \cdots \otimes g_m))\cdots\bigr)\bigr).$$

These operations can be greatly simplified by first collecting all the true variables in the g_i's in one sweep (a bunch of set unions) and iterating through the $(x_i, V_i) \overset{\leftrightarrow}{\otimes}$ indexes for collecting further true variables. Then the ROBDDs which occur in the macro-operation are restricted using the collected true variables and, at the end of this process, the ROBDD package is invoked over the simplified arguments. Only then we search for further true variables in the resulting ROBDD.

6.8 Experimental Evaluation

The ideas presented here have been experimentally validated in the context of the development of the China analyzer [Bag94]. China is a data-flow analyzer for CLP($\mathcal{H}_\mathcal{N}$) languages (i.e. Prolog, CLP($\mathcal{R}$), `clp(FD)` and so forth) written in Prolog and C++. It performs bottom-up analysis deriving information on both call and success patterns by means of program transformations and optimized fixpoint computation techniques.

The assessment of the hybrid domain has been done in a quite radical way. In fact, we have compared the standard, pure ROBDD-based implementation of *Pos* against the hybrid domain on the following problem: *deriving, once for each clause's evaluation, a Boolean vector indicating which*

	Analysis time (sec)			N. of BDD nodes		
Program	STD	HYB	S/H	STD	HYB	S/H
CS	1.06	0.6	1.77	12387	391	31.7
Disj	1.06	0.6	1.77	72918	176	414.3
DNF	5.17	4.4	1.18	5782	111	52.1
Gabriel	1.13	0.74	1.53	28634	10472	2.73
Kalah	3.92	2.02	1.94	43522	645	67.5
Peep	6.13	5.52	1.11	176402	128332	1.37
PG	0.37	0.25	1.48	3732	86	43.4
Plan	0.59	0.5	1.18	1736	65	26.7

Table 6.2: Experimental results obtained with the China analyzer.

variables are known to be ground and which are not. This is a very minimal demand for each analysis requiring the knowledge about definitely ground variables *during* the analysis. We have thus performed the analysis of a number of programs on a domain similar to `Pat`(*Pos*) [CLV94], switching off all the other domains currently supported by China[3]. `Pat`($\Re$) is a generic structural domain which is parametric with respect to any abstract domain $\Re$. Roughly speaking, `Pat`($\Re$) associates to each variable the following information:

- a *pattern*, that is to say, the principal functor and subterms which are bound to the variable;
- the "properties" of the variable, which are delegated to the $\Re$ domain (the two implementations of *Pos*, in our case).

As reported in [CLV93], `Pat`(*Pos*) is a very precise domain for groundness analysis.

The experimental results are reported in Table 6.2. The table gives, for each program, the analysis times and the number of ROBDD nodes allocated for the standard implementation (STD) and the hybrid one (HYB), respectively. It also shows the ratio STD/HYB for the above mentioned figures (S/H). The computation times have been taken on a 80486DX4 machine with 32 MB of RAM running Linux 1.3.64. The tested programs have become standard for the evaluation of data-flow analyzers. They are a cutting-stock program `CS`, the generate and test version of a disjunctive scheduling program `Disj`, a program to put Boolean formulas in disjunctive normal form `DNF`, the `Browse` program `Gabriel` taken from Gabriel benchmark, an alpha-beta procedure `Kalah`, the peephole optimizer of SB-Prolog

[3]Namely, numerical bounds and relations, aliasing, and polymorphic types

`Peep`, a program `PG` written by W. Older to solve a particular mathematical problem, and the famous planning program `Plan` by D.H.D. Warren.

The results indicate that the hybrid implementation outperforms the standard one in both time and space efficiency. The systematic speed-up obtained was not expected. Indeed, we were prepared to content ourselves with a moderate slow-down which would have been recovered in the reactive combinations. The space figures show that we have achieved significant (and sometimes huge) savings in the number of allocated ROBDD nodes. With the hybrid domain we are thus able to keep the ROBDDs which are created and traversed during the analysis as small as possible. This phenomenon is responsible for the speed-up. It seems that, even for programs characterized by not-so-many ground variables, there are always enough ground variables to make the hybrid implementation competitive. This can be observed, for instance, in the case of the `Peep` program, which was analyzed with a non-ground, most-general input pattern. The following additional observations are important for a full understanding of Table 6.2:

1. we are not comparing against a poor standard implementation of *Pos*, as can be seen by comparing the analysis times with those of [CLV93]. The ROBDD package we are using is fine-tuned: it employs separate caches for the main operations (with hit-rates in the range 95%–99% for almost all programs), specialized and optimized versions of the important operations over ROBDDs, as well as aggressive memory allocation strategies. Indeed, we were led to the present work by the apparent impossibility of further optimizing the standard implementation. Moreover, the hybrid implementation has room for improvement, especially for what concerns the handling of bit-vectors.

2. We are not taking into account the cost of garbage-collection for ROBDD nodes. In particular, the sizes of the relevant data-structures were chosen so that the analysis of the tested programs could run to completion without any node deallocation or reallocation.

3. The Boolean vectors computed during our test analyses are what is necessary for, say, the quick handling of delayed constraints and goals, and the efficient simplification of aliasing information. However, the experiment does not take into account the inevitable gains which are a consequence of the fast access to ground variables. Furthermore, in a truly reactive combination, the set of ground variables is not needed only at the end of each clause's evaluation (this is the optimistic hypothesis under which we conducted the experimentation), but at each body-atom evaluation for each clause. In this context the hybrid implementation, due to its incrementality, is even more favored with respect to the standard one (which is not incremental at all).

6.9 Conclusion

We have studied the problem, given an implementation of *Pos* based on ROBDDs, of determining as efficiently as possible the set of variables which are forced to true in the abstract representation. We have explained why, for the sake of realizing reactive combinations of domains, it is important to detect these variables (which correspond to *ground* ones at the concrete level) as quickly as possible. This problem has not been treated before in the literature [CH93, AMSS, HCC95]. After reviewing the *naïf* approaches, we have presented a simple method of detecting all the true variables in a ROBDD representation at once. We have then proposed a novel hybrid representation for Boolean functions. This representation is designed in a way to take advantage from the observation that most programs (together with their typical queries) have a high percentage of variables which are deemed to be ground at the program points of interest. With the new representation, not only the information about true (ground) variables is always readily available (instead of being scattered all over the ROBDDs), but we are also able to keep the usage of (expensive) ROBDDs at a minimum. This is clearly important for efficiency reasons. In fact, we have presented the experimental results obtained with a prototype implementation of the hybrid domain which outperforms, from any point of view, the standard implementation based on ROBDDs only. Surprisingly enough, we have thus been able to assess the superiority of the hybrid domain even for those cases where fast access to ground variables is not important.

Chapter 7

Precise Detection of Call-Patterns

Contents

7.1 Introduction

In data-flow analysis of constraint logic programs we are often interested in deriving information about just two kinds of program points: clause's entries and clause's successful exits. Using a terminology that has become customary in the field, that is to say that we want to derive *call-patterns* and *success-patterns*. In other words, for each clause we want to derive properties of the constraint store that are valid

1. whenever the clause is invoked (call-patterns); and

2. whenever a computation starting with the invocation of the clause terminates with success (success-patterns).

In principle, call-patterns can be reconstructed, to a limited extent, from the success-patterns. This, however, often implies significant precision losses. As the precision of call-patterns is very important for many applications, their direct computation is desirable. Top-down analysis methods are usually advocated for this purpose, since the standard execution strategy of (constraint) logic programs is top-down. Alternatively, methods based on program transformation and bottom-up analysis can be employed. This approach, however, can result in a loss of precision because the connection between call- and success-patterns is not preserved.

In a recent work, Debray and Ramakrishnan [DR94] introduced a bottom-up analysis technique for logic programs based on program transformation. They showed, among other things, that their bottom-up analysis is at least as precise (on both call- and success-patterns) as any top-down abstract interpretation using the same abstract domain and abstract operators.

The basic idea behind the approach of Debray and Ramakrishnan is to employ a variation of the *Magic Templates* algorithm [Ram88]. Nothing new yet, since this follows a line of research on bottom-up analysis methods based on the *Magic Sets* or similar transformations [Kan93, CDY94, Nil91]. However, as several authors [CDY94, Nil91, DR94] have pointed out, a direct application of *Magic Templates* can result in a loss of precision. While both call- and success-patterns are derived, the connection between them is lost. As a simple example, let us consider a program P containing the following CLP($\mathcal{R}$) clause, R:

```
transistor_state(Beta, Ib, Ic, Ie):-
    Ib >= 0,
    Ic = Beta*Ib,
    Ie+Ib+Ic = 0.
```

Suppose we apply the magic transformation to P and analyze the result employing a domain for range and relations, like those presented in Chapter 5, and a domain for groundness. The analysis of P can discover two different call-patterns (perhaps coming from two different call sites) for predicate `transistor_state`/4:

$$D_1^{\mathrm{c}} \overset{\mathrm{def}}{=} \bigl(\mathtt{Beta} > 0\bigr),$$
$$D_2^{\mathrm{c}} \overset{\mathrm{def}}{=} \bigl(\mathtt{Beta} > 0 \land \mathit{ground}(\mathtt{Beta})\bigr).$$

Analyzing clause R with respect to these call-patterns we can obtain the success-patterns[1]

$$D_1^{\mathrm{s}} \overset{\mathrm{def}}{=} \bigl(\mathtt{Beta} > 0 \land \mathtt{Ib} \geq 0\bigr), \tag{7.1}$$
$$D_2^{\mathrm{s}} \overset{\mathrm{def}}{=} D_1^{\mathrm{s}} \land \bigl(\mathtt{Ic} \geq 0 \land \mathtt{Ie} \leq 0\bigr). \tag{7.2}$$

Now, if the success-patterns are merged, obtaining D_1^{s}, we have clearly lost information. With the standard approach, i.e., by maintaining two separate relations for call- and success-patterns, even if we keep both D_1^{s} and D_2^{s}, all that can be said may be summarized as

$$\begin{aligned}\mathrm{calls}_R &\stackrel{\mathrm{def}}{=} \{D_1^{\mathrm{c}}, D_2^{\mathrm{c}}\},\\ \mathrm{succs}_R &\stackrel{\mathrm{def}}{=} \{D_1^{\mathrm{s}}, D_2^{\mathrm{s}}\}.\end{aligned} \tag{7.3}$$

Having lost the connection between call- and success-patterns, when confronted with a call-pattern, say,

$$D^{\mathrm{c}} \stackrel{\mathrm{def}}{=} (\mathtt{Beta} = 60),$$

which is stronger than D_2^{c}, we must content ourselves with the success-pattern

$$D^{\mathrm{s}} \stackrel{\mathrm{def}}{=} D^{\mathrm{c}} \wedge D_1^{\mathrm{s}}.$$

In fact, D^{c} is "compatible" with both D_1^{s} and D_2^{s}. Stated differently, D_1^{s} and D_2^{s} are both possible as success-patterns for D^{c}. In order to obtain the strictly more precise result

$$E^{\mathrm{s}} \stackrel{\mathrm{def}}{=} D^{\mathrm{c}} \wedge D_2^{\mathrm{s}}$$

we must keep the relation between call- and success-patterns explicitly. This means that for clause R we must record, instead of the two relations given in (7.3), something like

$$\mathrm{callsuccs}_R \stackrel{\mathrm{def}}{=} \bigl\{D_1^{\mathrm{c}} \mapsto D_1^{\mathrm{s}}, D_2^{\mathrm{c}} \mapsto D_2^{\mathrm{s}}\bigr\}. \tag{7.4}$$

The intuitive reading of (7.4) is as follows:

> all the concrete call-patterns of clause R are described by either D_1^{c} or D_2^{c}. Moreover, every call-pattern that is described by D_i^{c} gives rise to a concrete success-pattern that is described by D_i^{s}, for $i = 1$, 2.

In this particular example, all the concrete call-patterns described by D^{c} are also described by D_1^{c} and D_2^{c}. Formally:

$$\gamma(D^{\mathrm{c}}) \subseteq \gamma(D_1^{\mathrm{c}}) \cap \gamma(D_2^{\mathrm{c}}).$$

As a consequence, we are allowed to conclude that the ensuing success-patterns are described by *both* D_1^{s} and D_2^{s}. Thus we can conclude that

$$E^{\mathrm{s}} = D^{\mathrm{c}} \wedge D_2^{\mathrm{s}} = D^{\mathrm{c}} \wedge D_1^{\mathrm{s}} \wedge D_2^{\mathrm{s}}$$

[1]Notice that `Ic = Beta*Ib` is a non-linear constraint.

```
p(X) :-
    q(a,X),
    q(X,b).

q(X,Y).

?- p(X).
```

Figure 7.1: Program showing some difficulties with the magic transformation.

is a correct success-pattern for clause R corresponding to the call-pattern D^{c}.

The above example does not tell the entire story, and the reason is that we have considered an *abstract* domain, which, in addition, turns out to be *monotonic*[2]. Indeed, when dealing with monotonic properties only, the problem can be understood as a lack of expressivity of the domain, rather than a deficiency of the transformation approach. In this case, in fact, the information loss can be avoided to the extent the domain employed can represent the *dependencies* of the success-patterns from the call-patterns. For the above example, the ask-and-tell combination of the numerical domain with the groundness domain (see Section 3.7 on page sec:combining-domains) would allow to obtain the success-pattern

$$E_1^{\mathrm{s}} \stackrel{\mathrm{def}}{=} \bigl(\mathtt{Beta} > 0 \land \mathtt{Ib} \geq 0\bigr) \\ \qquad\qquad \parallel \mathrm{ask}\bigl(\mathit{ground}(\mathtt{Beta})\bigr) \rightarrow \mathrm{tell}\bigl(\mathtt{Ic} \geq 0 \land \mathtt{Ie} \leq 0\bigr),$$

instead of D_1^{s} as given by (7.1).

The limitations of the transformation approach are more serious when the analysis concerns (also) non-monotonic properties and, in this case, the theory developed in Chapter 3 is not applicable. The next example provides a very compelling argument: it shows that a straightforward application of magic transformation, followed by a standard bottom-up evaluation, causes information loss *already at the concrete level*.

Consider the program and goal in Figure 7.1.[3] The bottom-up evaluation of the transformed program on the standard domain of Herbrand equality

[2]Notice how the call-pattern D^{c} is "preserved" in the success-patterns D^{s} and E^{s}. This, of course, will not always happen for non-monotonic domains.

[3]This nice example is due to an anonymous referee.

constraints yields the following sets of call- and success-patterns[4]:

$$
\begin{aligned}
\text{calls} &\overset{\text{def}}{=} \bigl\{p(X), q(a, X), q(X, b), q(b, b)^*\bigr\}, \\
\text{succs} &\overset{\text{def}}{=} \bigl\{p(X), p(a)^*, p(b)^*, q(a, X), q(X, b), q(b, b)^*\bigr\}.
\end{aligned}
\tag{7.5}
$$

Here our inability to re-couple call- and success-patterns is self-evident. However, the real problem is that the patterns marked with '*' will never be computed by a top-down evaluation of the original program! This precision loss at the concrete level is clearly reflected at the abstract level. No matter how refined is the domain for freeness analysis we might employ, the fact that top-down evaluation of `?-p(X)` succeeds with a free argument cannot be derived.

The approach of [DR94], which is restricted to ordinary logic programs, solves the problems outlined above. It consists in transforming the source program by means of *Magic Templates*. Then an "explicated" version of the transformed program is built. In the explicated version the arguments of each clause head and body atom are duplicated: one copy represents the tuple of arguments at the moment of the call (the "calling arguments"), whereas the other copy represents the arguments at the moment of return (the "return arguments"). The "explication" of clause's bodies makes explicit the operational aspects of the execution of logic programs: unification in particular. The solution of [DR94] is not generalizable to the entire class of CLP programs, since it exploits some peculiar properties of the Herbrand constraint system, where a tuple of terms is a strong normal form for constraints. This way, and by duplicating the arguments, they are able to describe the partial functions that represent the connection between call- and success-patterns *in the clauses heads.* Of course, this cannot be done for generic CLP languages.

In this work we aim at generalizing the overall result of [DR94] to the unrestricted case of constraint logic programs: "*the abstract interpretation of languages with a top-down execution strategy need not itself be top-down.*" A bottom-up analysis framework has some advantages over top-down ones. It is simpler, it is easier to guarantee termination without special devices such as *memoization.* This, however, is beyond the scope of this work.

We present a generalized framework for the semantics of the entire class of CLP languages. The approach, which is still based on program transformation and bottom-up evaluation, is able to capture both call- and success-patterns without loss of precision at the concrete level. The framework can accommodate a very wide range of non-standard semantics and, in particular, abstract interpretations. We employ domains of partial functions that are parametric with respect to a given interpretation domain. These partial

[4]Observe that, for simplicity, we are exploiting the fact that tuples of terms constitute a normal form for Herbrand equality constraints.

functions express both call-patterns, success-patterns, and the connection between them in a very convenient way. Moreover, the entire construction is almost automatic: given any abstract domain we can define a *functional representation* over it and define an iteration sequence that computes the desired approximation of the abstract semantics. The only domain-dependent thing is (part of) the design of a suitable *widening operator* [CC77, CC92b] ensuring termination.

7.2 The Magic-Templates Algorithm

Our objective is to capture the top-down control strategy of CLP languages by means of a program transformation followed by a bottom-up evaluation. The transformation we employ is a generalization of *Magic Set* [BR87, BMSU86] that is applied by an instance of the *Magic Templates* algorithm. This algorithm is applicable to any language where programs are made up by Horn-like clauses: CLP in our case. The basic idea is to introduce auxiliary clauses, called *magic clauses*. They are intended to "compute" the constraints "passed" from one atom to another in the original clauses, following the control strategy of the language's implementation model (namely, extended SLD-resolution). Roughly speaking, each magic clause describes a procedure call in the body of an original clause. The *Magic Templates* algorithm also prescribes the addition of an atom (the *magic atom*, which is defined by the magic clauses) in the body of the original clauses. This atom forces the bottom-up evaluation to take into account the conditions under which the clause is invoked. In an evaluation over the standard (or concrete) domain this can be thought of as a "filter" that reduces the search space, preventing the generation of irrelevant facts, i.e., those that cannot be produced by any execution, standing a particular set of initial goals. In a data-flow analysis setting, the added atom is what makes possible a *goal-dependent analysis* with the effect of improving the accuracy. The original clauses with the magic atom added are called *modified clauses*. Here is the *Magic Templates* algorithm, instantiated over CLP languages and assuming a left-to-right selection rule.[5]

Definition 108 (Magic Templates.) [Ram88] *Let P be a CLP program and G be a set of* initial goals*. Starting from P and G, we build a new program P_G^{m}. Initially P_G^{m} is empty. Assuming that in P no predicate symbol has* '`magic_`' *as a prefix, we perform the following steps:*

1. *for each predicate symbol p/n in P (where n denotes the arity), create a new predicate symbol* `magic_`*p/n.*

[5]For the knowledgeable reader: in our case the *adorned* program coincides with the original program [Ram88].

2. *For each clause R of P, add the modified version of R to P_G^{m}. Let $p(\bar{t})$ be the head of R. The modified version is obtained by adding* `magic_`$p(\bar{t})$ *in the leftmost position of the body of R.*

3. *For each clause R in P and each atom $q_i(\bar{t}_i)$ in the body of R, add a magic clause to P_G^{m}: the head is* `magic_`$q_i(\bar{t}_i)$*, whereas the body consists of all the constraints and atoms preceding $q_i(\bar{t}_i)$ in the modified version of R.*

4. *For each goal g in G and each atom $q_i(\bar{t}_i)$ in g, add a magic clause to P_G^{m}: the head is* `magic_`$q_i(\bar{t}_i)$ *whereas the body consists of all the constraints and atoms preceding $q_i(\bar{t}_i)$ in g.*

For a predicate symbol p we will use the more comfortable notation `m_`p instead of `magic_`p.

As an example, *Magic Templates* transforms the program of Figure 7.1 on page fig:magic-example into the program of Figure 7.2. For a CLP example, the CLP($\mathcal{N}$) program in Figure 5.2 on page fig:numeric-mc91 is transformed as in Figure 7.3.

7.3 Generalized Semantics

As shown in Chapter 3, the characteristics of the CLP framework [JL87] allow for the definition of a generalized algebraic semantics. However, the framework presented in Chapter 3 is limited to (possibly non-standard) interpretations based on *monotonic* properties. For this class of properties the similarities between the standard and non-standard interpretations are striking, as monotonicity allows to employ constraints also to represent the non-standard properties. For our purposes we need a more general framework, capturing also non-monotonic properties, which is introduced in the following sections.

7.3.1 Interpretation Domains

Here we generalize the semantic treatment for CLP languages presented in Section 3.2 on page sec:case-study-CLP. While several things carry through without change to the case where non-monotonic properties are considered, we can no longer assume that the domain of interpretation is a constraint system. We thus introduce a class of algebraic structures called *interpretation domains.*

The carrier of an interpretation domain is a set of *properties*, or *descriptions.* In our setting, we assume that properties deal with *program variables.* Properties usually describe the instantiation status of program variables, at some level of abstraction. As we did in Section 3.2.3 on page sec:constraint-systems, we make explicit some (very reasonable) assumptions about the

```
p(X) :-                       m_q(a, X) :-
    m_p(X),                       m_p(X).
    q(a,X),                   m_q(X, b) :-
    q(X,b).                       m_p(X),
                                  q(a, X).

q(X,Y) :-                     m_p(X).
    m_q(X, Y).
```

Figure 7.2: The “magic version” of the program in Figure 7.1.

```
mc(N, N-10) :-
    m_mc(N, N-10),
    N > 100.
mc(N, M) :-
    m_mc(N, M),
    N <= 100,
    mc(N+11, U),
    mc(U, M).

m_mc(N+11, U) :-
    m_mc(N, M),
    N <= 100.
m_mc(U, M) :-
    m_mc(N, M),
    N <= 100,
    mc(N+11, U).
```

Figure 7.3: The “magic version” of the program in Figure 5.2 on page fig:numeric-mc91.

relationships of properties to program variables. We still assume the existence of two denumerable sets of variable symbols, V and Λ, which are disjoint. We also denote the set of all variables by $\mathit{Vars} \stackrel{\mathrm{def}}{=} V \uplus \Lambda$. For a property D, we will indicate by $FV(D)$ the set variables D talks about. The notion of *renaming* remains also unchanged: if a property D describes a tuple of variables $\bar{X}$, $D[\bar{Y}/\bar{X}]$ is that very same property applied to $\bar{Y}$, if $\bar{Y}$ and $\bar{X}$ are disjoint tuples of distinct variables. The renaming $[Y/X]$ has no effect on D if $X \notin FV(D)$, whereas variable capture is avoided by consistent renaming of bound variables. We will say that $[\bar{Y}/\bar{X}]$ is a renaming *for D* if and only if $FV(D) \cap \bar{Y} = \varnothing$.

Definition 109 (Interpretation domain.) *Any algebra $\bar{\mathcal{D}}$ of the form*

$$\left\langle \mathcal{D}, \trianglelefteq, \otimes, \oplus, \mathbf{0}, \mathbf{1}, \{\bar{\exists}_{\bar{X}}\}_{\bar{X} \in \mathit{Vars}^\star}, \{\mathrm{d}_{\bar{X}\bar{Y}}\}_{\bar{X}, \bar{Y} \in \mathit{Vars}^\star} \right\rangle$$

is an interpretation domain *if it satisfies the following conditions:*

I_0. *$\mathcal{D}$ is a set of properties;*

I_1. *$\trianglelefteq$ is a preorder over $\mathcal{D}$;*

I_2. *for each $D \in \mathcal{D}$, $D \trianglelefteq \mathbf{1}$;*

I_3. *$\otimes$ is an associative binary operator over $\mathcal{D}$;*

I_4. *$\langle \mathcal{D}, \oplus, \mathbf{0} \rangle$ is a commutative and idempotent monoid;*

I_5. *$\mathbf{0}$ is an* annihilator *for $\otimes$, namely, for each $D \in \mathcal{D}$ we have $D \otimes \mathbf{0} = \mathbf{0} \otimes D = \mathbf{0}$;*

I_6. *for each $\bar{X} \in \mathit{Vars}^\star$ we have $\bar{\exists}_{\bar{X}} \colon \mathcal{D} \to \mathcal{D}$ and $FV\bigl(\bar{\exists}_{\bar{X}} D\bigr) \subseteq \bar{X}$ for each $D \in \mathcal{D}$;*

I_7. *for each $\bar{X}, \bar{Y} \in \mathit{Vars}^\star$, $\mathrm{d}_{\bar{X}\bar{Y}} \in \mathcal{D}$ and $\mathrm{d}_{\bar{X}\bar{Y}} = \mathrm{d}_{\bar{Y}\bar{X}}$.*

The relation $\trianglelefteq$ is referred to as the *approximation ordering* of the interpretation domain. It specifies the relative precision of the properties in $\mathcal{D}$: $D_1 \trianglelefteq D_2$ means that D_1 is more precise than D_2 or, alternatively, that D_2 is a safe approximation of D_1. The $\oplus$ operator models the merging of information coming from different execution paths. Its properties ensure that the merging process is insensitive to grouping, ordering and multiplicity. Moreover, empty or failed paths (which intuitively correspond to $\mathbf{0}$) are ignored. The $\otimes$ operator models the constraint accumulation process. The element $\mathbf{1}$ describes, intuitively, the empty constraint store, i.e., the one containing no information at all. The $\bar{\exists}_{\bar{X}}$ operators represent the *hiding* process: any variable $X \notin \bar{X}$ appearing in the scope of $\bar{\exists}_{\bar{X}}$ is isolated (hidden) from other occurrences of X outside the scope. The "complement sign" that appears on top of $\bar{\exists}$ signifies that we formalize hiding in a dual way with respect to traditional approaches [SRP91, GDL95]. The distinguished elements $\mathrm{d}_{\bar{X}\bar{Y}}$ represent parameter passing.

Indeed, interpretation domains must satisfy some other (very technical) conditions related to how they deal with variables. The interested reader can find these additional requirements in Hypothesis 121 on page hypo:magic-id-axioms.

Notice that the above definition specifies the bare essentials we require from a (non-standard) domain that is suitable for the abstract interpretation of CLP languages. This way we can capture abstract domains that are very weak, despite their importance for data-flow analysis. It is worth noticing that any constraint system, in the sense of Definition 7 on page def:constraint-system is an interpretation domain. The converse, of course, does not hold.

However, in a generalized framework such as the one we are describing, we need also structures that are able to capture the "concrete" (or standard) semantics of programs. Here, of course, we are much more demanding, since we want to prove the equivalence between different semantic constructions (e.g., the operational, top-down construction and the bottom-up construction). For this purpose, we now introduce a class of "strong" interpretation domains. With the domains in this class one can characterize the standard semantics of any CLP language.

Definition 110 (Strong interpretation domain.) *An interpretation domain $\bar{\mathcal{D}}$ is a* strong interpretation domain *if and only if it satisfies the following conditions, for each $\bar{X}, \bar{Y} \in Vars^\star$, each renaming ρ, and $D, D_1, D_2 \in \mathcal{D}$:*

S_1. *$\otimes$ is commutative and idempotent;*

S_2. *$D \otimes \bar{\exists}_{\bar{X}} D = D$;*

S_3. *for each family $\{D_i \in \mathcal{D}\}_{i\in\mathbb{N}}$, $\bigoplus_{i\in\mathbb{N}} D_i \stackrel{\mathrm{def}}{=} D_1 \oplus D_2 \oplus \cdots$ exists and is unique in $\mathcal{D}$; moreover, associativity, commutativity, and idempotence of $\oplus$ apply to denumerable as well as to finite families of operands;*

S_4. *$\bar{\exists}_{\bar{X}}\bigl(\bar{\exists}_{\bar{X}} D_1 \otimes D_2\bigr) = \bar{\exists}_{\bar{X}}\bigl(D_1 \otimes \bar{\exists}_{\bar{X}} D_2\bigr) = \bar{\exists}_{\bar{X}} D_1 \otimes \bar{\exists}_{\bar{X}} D_2$;*

S_5. *if $\bar{X}\rho = \bar{Y}\rho = \bar{Y}$ then $\bar{\exists}_{\bar{Y}}(D\rho) = \bar{\exists}_{\bar{Y}}\bigl(D \otimes \mathrm{d}_{\bar{X}\bar{Y}}\bigr)$;*

S_6. *$\bar{\exists}_{\bar{X}}\bigl(\bar{\exists}_{\bar{X}} D\bigr) = \bar{\exists}_{\bar{X}} D$, namely, $\bar{\exists}_{\bar{X}}$ is idempotent;*

S_7. *if $FV(D_1) \cap FV(D_2) \subseteq \bar{Y}$ and $FV(D_1) \cap \bar{X} \subseteq \bar{Y}$ then*

$$\bar{\exists}_{\bar{X}}(D_1 \otimes D_2) = \bar{\exists}_{\bar{X}}\bigl(\bar{\exists}_{\bar{Y}} D_1 \otimes D_2\bigr).$$

Any strong interpretation domain is a closed constraint system. Notice that here, for the first time, we needed to give a rather precise specification of the projection operators and the parameter passing elements. These conditions are clearly satisfied by any concrete constraint domain, and are necessary in order to ensure the equivalence between the semantic constructions we will present later.

7.3.2 Functional Representations

In the following sections, we will present semantic characterizations of CLP capturing both call- and success-patterns and maintaining the connection between them. For any such characterization, and given an interpretation domain $\bar{\mathcal{D}}$, the "meaning" of a program clause R is a partial function $\mu_R \colon \mathcal{D} \rightarrowtail \wp(\mathcal{D})$. The idea is that descriptions appearing in the domain of μ_R, $\mathrm{dom}(\mu_R)$, are the possible call-patterns for R, whereas, for each $D \in \mathrm{dom}(\mu_R)$, $\mu_R(D)$ describes the possible outcomes of a call to R started in a constraint store described by C.

Partial functions of this kind can be conveniently represented by subsets of $\mathcal{D} \times \mathcal{D}$ [Bou92]. However, since we are concerned with successful computations only, we will use subsets of $\bigl(\mathcal{D} \setminus \{\mathbf{0}\}\bigr) \times \mathcal{D}$ to represent the partial functions of interest. Instead of the usual notation for pairs $(D^{\mathrm{c}}, D^{\mathrm{s}})$ we will use the perspicuous notation $(D^{\mathrm{c}} \mapsto D^{\mathrm{s}})$ for emphasis. Intuitively, a pair $(D^{\mathrm{c}} \mapsto D^{\mathrm{s}})$ represent the set of all computations starting with a (consistent) constraint store described by D^{c} and terminating (successfully or not) in a constraint store described by D^{s}.

Before introducing the functional representation we need to define a binary predicate over properties. It expresses the notion of *compatibility* of two properties. In a bottom-up construction what happens is that (characterizations of) partial computations are, in a sense, concatenated in order to obtain (characterizations of) longer computations. Suppose we know that some partial computation starting with the invocation of predicate p, in the context of a constraint store satisfying D_1, arrives at a point where predicate q is called with a constraint store characterized by property D_2. Suppose we know also that, whenever predicate q is called with a constraint satisfying D_3, the computation succeeds with a constraint described by D_4. If we blindly concatenate the above partial computations, then we may obtain a computation that cannot exist in any top-down derivation. In fact, the above hypotheses do not constitute the evidence that a computation from p starting in a constraint satisfying D_1 succeeds with a constraint satisfying D_4. In our construction we will require that D_2 and D_3 are *compatible*.

Definition 111 (Compatibility predicate.) *Let $\bar{\mathcal{D}}^\natural$ be a* strong *interpretation domain. Then, $\between^\natural \subseteq \mathcal{D} \times \mathcal{D}$ is given for each $D_1^\natural, D_2^\natural \in \mathcal{D}^\natural$, by*

$$D_1^\natural \between^\natural D_2^\natural \quad \overset{\mathrm{def}}{\iff} \quad D_1^\natural = D_2^\natural.$$

If, instead, $\bar{\mathcal{D}}^\sharp$ is an interpretation domain intended to abstract another interpretation domain $\bar{\mathcal{D}}^\natural$ trough a concretization function $\gamma\colon \mathcal{D}^\sharp \to \wp(\mathcal{D}^\natural)$, then $\between^\sharp \subseteq \mathcal{D}^\sharp \times \mathcal{D}^\sharp$ satisfies the following safety requirement:

$$\forall D_1^\sharp, D_2^\sharp \in \mathcal{D} : \bigl((D_1^\sharp \between^\sharp D_2^\sharp)\bigr) \vee \bigl(\gamma(D_1^\sharp) \cap \gamma(D_2^\sharp) = \varnothing\bigr).$$

The $\between$ predicates will be used in order to limit the extent to which the bottom-up construction concatenates partial computations that are not concatenatable. This is the essence of the problem, illustrated above with the program of Figure 7.1, affecting the standard approaches employing the magic transformation followed by bottom-up evaluation.

The above definition makes clear the different notions of compatibility in the concrete case, where strong interpretation domains are used to characterize the concrete semantics, and in the abstract case. In the former case, we want a correct and complete criterion for deciding whether two partial computations can be concatenated, still obtaining a partial computation: the first computation must end up with a constraint store that is exactly the starting constraint store of the second computation.

In the latter case, when $\bar{\mathcal{D}}^\sharp$ is an analysis' domain, we must content ourselves with an approximation. Suppose that $\bar{\mathcal{D}}^\sharp$ embodies both groundness and freeness information. Clearly, a partial computation ending up with a constraint where X is free cannot be prolonged with a partial computation starting with X ground, since no satisfiable constraint can make X both ground and free. If we are employing a domain for ranges and relations of and among numerical variables, we can certainly conclude that a partial computation ending with $X \geq 0$ cannot be extended with another partial computation that starts with $X < 0$.

At the implementation level, $\between^\sharp$ admits different degrees of sophistication, and may involve several domains. Suppose we have domains for aliasing, freeness, and structural information. While the aliasing domain provides *definite non-sharing* information, the combination of structural information and freeness provides some *definite sharing* information. Suppose that the aliasing component of $D_1^\sharp$ implies "X and Y do not share a common variable". Suppose also that the structural component of $D_2^\sharp$ contains $X = f(Z)$ and $Y = g(a, Z)$ and that the freeness component reveals that Z is indeed free. A refined implementation would discover this inconsistency making $D_1^\sharp \between^\sharp D_2^\sharp$ false. Definition 111, however, imposes only the correctness of $\between^\sharp$, the precision being left to efficiency considerations that are outside the scope of the present discussion.

Definition 112 (Functional representation.) *Let $\bar{\mathcal{D}}$ be an interpretation domain. The* functional representation *over $\bar{\mathcal{D}}$ is defined as:*

$$\bar{\mathcal{D}}_{\mathrm{F}} \stackrel{\mathrm{def}}{=} \bigl\langle \mathcal{D}_{\mathrm{F}}, \trianglelefteq_{\mathrm{F}}, \otimes_{\mathrm{F}}, \mathbf{0}_{\mathrm{F}}, \{\exists^{\mathrm{F}}_{\bar{X}}\}_{\bar{X} \in \mathit{Vars}^\star} \bigr\rangle,$$

where

$$\begin{aligned}
\mathcal{D}_{\mathrm{F}} &\stackrel{\mathrm{def}}{=} \wp\Big(\big(\mathcal{D} \setminus \{\mathbf{0}\}\big) \times \mathcal{D}\Big),\\
\mathbf{0}_{\mathrm{F}} &\stackrel{\mathrm{def}}{=} \varnothing,\\
\bar{\exists}^{\mathrm{F}}_{\bar{X}}\, S &\stackrel{\mathrm{def}}{=} \Big\{ \big(\bar{\exists}_{\bar{X}}\, D^{\mathrm{c}} \mapsto \bar{\exists}_{\bar{X}}\, D^{\mathrm{s}}\big) \;\Big|\; (D^{\mathrm{c}} \mapsto D^{\mathrm{s}}) \in S \Big\}.
\end{aligned}$$

The relation $\trianglelefteq_{\mathrm{F}} \subseteq \mathcal{D}_{\mathrm{F}} \times \mathcal{D}_{\mathrm{F}}$ is given for each $S_1, S_2 \in \mathcal{D}_{\mathrm{F}}$, by

$$\begin{aligned}
S_1 \trianglelefteq_{\mathrm{F}} S_2 \stackrel{\mathrm{def}}{\iff} {} & \forall (D_1^{\mathrm{c}} \mapsto D_1^{\mathrm{s}}) \in S_1 :\\
& \quad \exists (D_2^{\mathrm{c}} \mapsto D_2^{\mathrm{s}}) \in S_2 \;.\\
& \qquad (D_1^{\mathrm{c}} \trianglelefteq D_2^{\mathrm{c}}) \wedge (D_1^{\mathrm{s}} \trianglelefteq D_2^{\mathrm{s}}).
\end{aligned}$$

Finally, if we define

$$\tilde{\mathcal{D}}_{\mathrm{F}} \stackrel{\mathrm{def}}{=} \mathcal{D} \cup \mathcal{D}_{\mathrm{F}},$$

the binary operator $\otimes_{\mathrm{F}} : \tilde{\mathcal{D}}_{\mathrm{F}} \times \tilde{\mathcal{D}}_{\mathrm{F}} \to \tilde{\mathcal{D}}_{\mathrm{F}}$ is given as follows, for each $S, S_1, S_2 \in \mathcal{D}_{\mathrm{F}}$ and each $D, D_1, D_2 \in \mathcal{D}$:

$$\begin{aligned}
S_1 \otimes_{\mathrm{F}} S_2 &\stackrel{\mathrm{def}}{=} \left\{ D_1^{\mathrm{c}} \mapsto D_1^{\mathrm{s}} \otimes D_2^{\mathrm{s}} \;\middle|\; \begin{array}{l} (D_1^{\mathrm{c}} \mapsto D_1^{\mathrm{s}}) \in S_1 \\ (D_2^{\mathrm{c}} \mapsto D_2^{\mathrm{s}}) \in S_2 \\ (\bar{\exists}_{FV(D_2^{\mathrm{c}})}\, D_1^{\mathrm{s}}) \between D_2^{\mathrm{c}} \\ D_1^{\mathrm{s}} \otimes D_2^{\mathrm{s}} \neq \mathbf{0} \end{array} \right\},\\
D \otimes_{\mathrm{F}} S &\stackrel{\mathrm{def}}{=} \left\{ D \mapsto D \otimes D^{\mathrm{s}} \;\middle|\; \begin{array}{l} (D^{\mathrm{c}} \mapsto D^{\mathrm{s}}) \in S \\ \big(\bar{\exists}_{FV(D^{\mathrm{c}})}\, D\big) \between D^{\mathrm{c}} \\ D \otimes D^{\mathrm{s}} \neq \mathbf{0} \end{array} \right\},\\
S \otimes_{\mathrm{F}} D &\stackrel{\mathrm{def}}{=} \left\{ D^{\mathrm{c}} \mapsto D^{\mathrm{s}} \otimes D \;\middle|\; \begin{array}{l} (D^{\mathrm{c}} \mapsto D^{\mathrm{s}}) \in S \\ D^{\mathrm{s}} \otimes D \neq \mathbf{0} \end{array} \right\},\\
D_1 \otimes_{\mathrm{F}} D_2 &\stackrel{\mathrm{def}}{=} D_1 \otimes D_2.
\end{aligned}$$

The notion of $FV(\cdot)$ and renaming are extended pointwise to functional domains, preserving all the needed properties.

It is possible to show (see Proposition 122 on page prop:functional-domains-properties) that the following hold: $\mathbf{0}_{\mathrm{F}}$ is an annihilator for $\otimes_{\mathrm{F}}$; $\trianglelefteq_{\mathrm{F}}$ is a preorder with minimum element $\mathbf{0}_{\mathrm{F}}$; $\otimes_{\mathrm{F}}$ is componentwise monotonic with respect to $\subseteq$, and completely left- and right-distributive with respect to $\cup$; finally, $\bar{\exists}^{\mathrm{F}}_{\bar{X}}$ is monotonic with respect to $\subseteq$ and continuous with respect to $\cup$.

7.3.3 Generalized CLP Programs

We assume, for simplicity, that clauses are of the form

$$p(\bar{X}) :- \{c_1, \dots, c_n\} \,\Box\, (b_1, \dots, b_k),$$

where $\{c_1, \dots, c_n\}$ is a (conjunctive) set of atomic constraints and $(b_1, \dots, b_k)$ is a sequence of atoms. Notice that the parentheses around $(b_1, \dots, b_k)$ will be often omitted for simplicity. When $\otimes$ is commutative this assumption can be made without restrictions. The reader can easily figure out how to modify (at the price of some complications) the subsequent definitions in order to accommodate non-commutative $\otimes$ operators.

It is also convenient to consider, instead of a *syntactic* CLP($\mathcal{X}$) program P, its *semantic* version over some interpretation domain $\bar{\mathcal{D}}$. This is obtained by interpreting the set of constraints appearing in clauses through a function $[\![\cdot]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}} \colon \wp_{\mathrm{f}}(\mathsf{C}) \to \mathcal{D}$, where C is the language of (syntactic) atomic constraints for CLP($\mathcal{X}$).

Another important simplification can be obtained if we assume, without loss of generality, that the heads of program clauses are normalized. It is for this purpose that we have an additional, totally ordered set of variable symbols, $\Lambda = \{A_1, A_2, A_3, \dots\}$, disjoint from V. The head of each program clause defining a predicate p/n will always be $p(A_1, \dots, A_n)$, and denoted by $p(\vec{\Lambda})$, where $\vec{\Lambda}$ means "an initial segment of Λ of the appropriate length." This device will avoid much of the burden of talking "modulo renaming".

A CLP($\bar{\mathcal{D}}$) *program* is a *sequence* of Horn-like formulas of the form

$$p(\vec{\Lambda}) :- D \,\Box\, b_1, \dots, b_k,$$

where $D \in \mathcal{D}$. Given a CLP($\mathcal{X}$) program P and a constraint interpretation $[\![\cdot]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}}$, the CLP($\bar{\mathcal{D}}$) program $[\![P]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}}$ is given, for each $r = 1, \dots, \# P$, by

$$[\![P]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}}[r] = \bigl(p(\vec{\Lambda}) :- [\![C]\!]^{\bar{\mathcal{D}}}_{\mathsf{C}} \,\Box\, B\bigr) \quad \overset{\text{def}}{\iff} \quad P[r] = \bigl(p(\vec{\Lambda}) :- C \,\Box\, B\bigr).$$

7.3.4 The Functional Semantics

Given any functional domain $\bar{\mathcal{D}}_{\mathrm{F}}$, our semantics associates an element of $\mathcal{D}_{\mathrm{F}}$ to each program clause.

Definition 113 (Functional interpretation.) *Let $\bar{\mathcal{D}}$ be an interpretation domain, let P be a* CLP($\bar{\mathcal{D}}$) *program, and let G be a set of* CLP($\bar{\mathcal{D}}$) *goals. A* functional interpretation for P over $\bar{\mathcal{D}}$ *is any element I of*

$$\mathcal{I}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P,G} \overset{\text{def}}{=} \{1, \dots, \# P\} \to \mathcal{D}_{\mathrm{F}}$$

such that, for each $r = 1, \dots, \# P$, if the head of $P[r]$ is $p(\vec{\Lambda})$ then $FV\bigl(I(r)\bigr) \subseteq \vec{\Lambda}$. The operations and relations of $\bar{\mathcal{D}}_{\mathrm{F}}$ are extended pointwise to $\mathcal{I}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P,G}$ in the obvious way. Functional interpretations will be represented by means of function graphs.

It is straightforward to show that all the interesting properties of functional representations lift smoothly to interpretations.

7.3.5 Top-Down (Operational) Construction

The notion of derivation for CLP($\bar{\mathcal{D}}$) programs is quite standard. Notice that we restrict ourselves to the *leftmost* selection rule.

Definition 114 (Finite derivation.) *Given any interpretation domain $\bar{\mathcal{D}}$, let P be a CLP($\bar{\mathcal{D}}$) program and G be an initial goal for P. A* finite derivation of P from G *is a sequence of goals $(G_0, G_1, \dots, G_n)$, together with a sequence $(R_1, \dots, R_k)$ of variants of clauses in P that are* renamed apart, *where $G_0 = G$ and, for each $i > 0$ we have:*

1. *$G_i = D_i \,\Box\, p_{i_1}(\bar{X}_{i_1}), \dots, p_{i_n}(\bar{X}_{i_n})$ with $D_i \neq \mathbf{0}$;*
2. *$R_1, \dots, R_h$ are the variants used in the derivation up to G_i;*
3. *$R_{h+1} = \bigl(p_{i_1}(\bar{Y}) \text{ :- } D \,\Box\, B\bigr)$ is a variant of $P[r]$ such that*[6]
$$FV(R_{h+1}) \cap \left(FV(G_0) \cup \bigcup_{j=1}^{h} FV(R_j) \right) = \varnothing;$$
4. *$G_{i+1} = D_i \otimes \mathrm{d}_{\bar{X}_{i_1}\bar{Y}} \otimes D \,\Box\, B :: \bigl(p_{i_2}(\bar{X}_{i_2}), \dots, p_{i_n}(\bar{X}_{i_n})\bigr)$.*

Each transition from G_i to G_{i+1} is called a derivation step *and is indicated by $G_i \leadsto_P G_{i+1}$. When $\bar{\mathcal{D}}$ is the standard constraint domain for the CLP language at hand, the constraint D_i associated to G_i is called the* constraint store *at step i. A* successful derivation *is a finite derivation where the last goal is constituted by constraints only. The derivation is* finitely failed *otherwise.*

The only important point about the operational construction is that call-patterns are recorded, by means of functional representation's elements, together with the corresponding success-patterns.

Definition 115 (Operational semantics.) *Given a domain $\bar{\mathcal{D}}$, a* CLP($\bar{\mathcal{D}}$) *program P, and a set G of* CLP($\bar{\mathcal{D}}$) *goals, we define $\mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P, G) \in \mathcal{I}_{P,G}^{\bar{\mathcal{D}}_{\mathrm{F}}}$ as*

$$\mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P, G) \stackrel{\text{def}}{=} \Bigl\{ \bigl(r, \mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P, G, r)\bigr) \Bigm| 1 \le r \le \# P \Bigr\},$$

[6]This static condition is sufficient to ensure correct renaming apart. This is because we assume (see Hypothesis 121 on page hypo:magic-id-axioms) that $\otimes$ does not introduce new variables: $FV(D_1 \otimes D_2) \subseteq FV(D_1) \cup FV(D_2)$, for each $D_1, D_2 \in \mathcal{D}$ (in other words, $\otimes$ is *relevant*). If $\otimes$ is not relevant such a renaming-apart condition cannot be given statically, i.e., before actually starting the derivation.

where

$$\mathcal{O}_{\bar{\mathcal{D}}_\mathrm{F}}(P, G, r) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{r|l} & G_0 \in G \\ & R = \bigl(p(\bar{Y}) \;{:}{-}\; D \,\square\, B_R\bigr) \\ & \quad \textit{is a variant of } P[r] \\ \bigl((\bar{\exists}_{\bar{X}}\, D^\mathrm{c})[\vec{\Lambda}/\bar{X}] \mapsto & G_0 \rightsquigarrow_P^* D^\mathrm{c} \,\square\, p(\bar{X}) :: B \\ (\bar{\exists}_{\bar{X}}\, D^\mathrm{s})[\vec{\Lambda}/\bar{X}]\bigr) & D^\mathrm{c} \neq \mathbf{0} \\ & D^\mathrm{c} \,\square\, p(\bar{X}) \rightsquigarrow_{\{R\}} G' \\ & \qquad\qquad \rightsquigarrow_P^* \; D^\mathrm{s} \,\square\, B' \\ & D^\mathrm{s} \neq \mathbf{0} \implies B' = \epsilon \end{array} \right\}.$$

In words, here is what happens. We take any goal G_0 in our set of initial goals G and a suitable variant R of clause $P[r]$. Suppose there is a finite derivation from G_0 to a goal where the constraint is D^c and the first atom, $p(\bar{X})$, has the same predicate symbol, p, of $P[r]$ (the sequence B of atoms yet to be solved after the call to p does not matter). Suppose also that D^c is consistent, and consider a partial derivation from $D^\mathrm{c} \,\square\, p(\bar{X})$ starting with an application of clause R (arriving at a configuration G', which we do not care about), and continuing until a configuration, say, $D^\mathrm{s} \,\square\, B'$ is reached, where B' is the sequence of atoms yet to be solved. The last condition says an important thing about the *partiality* of this last derivation:

- either $D^\mathrm{s} = \mathbf{0}$, in which case the derivation is failed (B' might be non-empty or not); nonetheless, clause $P[r]$ has been invoked with call-pattern D^c;

- or $B' = \epsilon$, in which case the derivation is indeed total and successful; D^s is thus the success-pattern corresponding to an invocation of clause $P[r]$ with call-pattern D^c.

Notice that properties are normalized using the distinguished variable symbols in Λ.

7.3.6 Bottom-Up (Fixpoint) Construction

Given CLP($\mathcal{X}$) program P, a set of CLP($\mathcal{X}$) goals G, and an interpretation domain $\bar{\mathcal{D}}$, let P_G be the generalized CLP($\bar{\mathcal{D}}$) program obtained by transforming P as specified by Definition 108 and interpreting the constraints over $\bar{\mathcal{D}}$. The bottom-up construction for P is defined over P_G.

For a program P and each $i = 1, \ldots, \# P$ we assume that $P_G[i]$ is the modified clause corresponding to $P[i]$. Moreover, we represent the bodies of the clauses of P_G as a triple: magic atom (if any), property (describing the original clause's constraints), and original body atoms. Thus, each clause of P_G has the general form

$$\hat{p}(\vec{\Lambda}) :- (M, D, B)$$

where $\hat{p}$ is a (possibly magic) atom, M is a set of magic atoms that is either empty or a singleton, $D \in \mathcal{D}$ is a property, and B is a finite, possibly empty sequence of atoms. This representation allows us to generalize the following three cases: modified clauses,

$$p(\vec{\Lambda}) :- \Bigl(\bigl\{\mathtt{m\_}p(\vec{\Lambda})\bigr\}, D, \bigl(p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\bigr)\Bigr);$$

magic clauses arising from the transformation of a goal,

$$\mathtt{m\_}p(\vec{\Lambda}) :- \Bigl(\varnothing, D, \bigl(p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\bigr)\Bigr);$$

and other magic clauses,

$$\mathtt{m\_}p(\vec{\Lambda}) :- \Bigl(\bigl\{\mathtt{m\_}q(\bar{X})\bigr\}, D, \bigl(p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\bigr)\Bigr).$$

Remember that our set of program variables is $\mathit{Vars} = V \uplus \Lambda$, where Λ and V are totally ordered. For $\vec{\Lambda} \in \Lambda^\star$ and $W \subseteq_{\mathrm{f}} \mathit{Vars}$, we denote by $\bar{Y} \ll_{\vec{\Lambda}} W$ the fact that, with respect to the ordering of V, $\bar{Y}$ is a tuple of distinct consecutive variables in V such that $\# \bar{Y} = \# \vec{\Lambda}$ and the first element of $\bar{Y}$ immediately follows the greatest variable in W.

Definition 116 (Interpretation transformer.) *Let P be any* CLP($\mathcal{X}$) *program and G a set of* CLP($\mathcal{X}$) *goals. Let $\bar{\mathcal{D}}_{\mathrm{F}}$ be the functional representation over some domain $\bar{\mathcal{D}}$. Let P_G be the CLP($\bar{\mathcal{D}}$) program obtained by applying the Magic Templates algorithm to P and G and interpreting the result over $\bar{\mathcal{D}}$ through $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}}$. The operator induced by P_G over $\mathcal{I}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}$,*

$$T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} : \mathcal{I}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} \to \mathcal{I}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}},$$

is given by

$$T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(I) \stackrel{\mathrm{def}}{=} \Bigl\{ \bigl(r, T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r, I)\bigr) \Bigm| 1 \leq r \leq \# P_G \Bigr\}, \quad \textit{for each } I \in \mathcal{I}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}},$$

where $T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} : \{1, \ldots, \# P_G\} \times \mathcal{I}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} \to \mathcal{I}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}$ is defined as follows: if

$$P_G[r] = \hat{p}(\vec{\Lambda}) :- \Bigl(M, D, \bigl(p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\bigr)\Bigr),$$

then

$$T^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r, I) \overset{\mathrm{def}}{=}$$

$$\bigcup \left\{ \exists^{\mathrm{F}}_{\vec{\Lambda}}\, \eta_r(S) \;\middle|\; \begin{array}{l} \textit{if } M = \big\{\mathtt{m\_}q(\bar{X})\big\} \\ \quad P_G[r_0] = \mathtt{m\_}q(\vec{\Lambda}_0) \mathrel{:-} T_0 \\ \quad \bar{Y} \ll_{\vec{\Lambda}_0} FV(P_G[r]) \cup \Lambda \\ \quad S_0 = \varpi_r\big(I(r_0), \vec{\Lambda}_0, \bar{X}, \bar{Y}\big) \\ \textit{else } S_0 = \{\mathbf{1} \mapsto \mathbf{1}\} \\ \textit{for each } i = 1, \ldots, n\textit{:} \\ \quad P_G[r_i] = p_i(\vec{\Lambda}_i) \mathrel{:-} T_i \\ \quad \bar{Y}_i \ll_{\vec{\Lambda}_i} FV(P_G[r]) \cup \Lambda \\ \qquad\qquad\qquad \cup\, \bar{Y} \cup \bigcup_{k=1}^{i-1} \bar{Y}_k \\ \quad S_i = \big(I(r_i)[\bar{Y}_i/\vec{\Lambda}_i]\big) \\ S = \overleftarrow{\otimes}_{\mathrm{F}}\big(S_0, D, \mathrm{d}_{\bar{X}_1\bar{Y}_1}, S_1, \\ \qquad\qquad \ldots, \mathrm{d}_{\bar{X}_n\bar{Y}_n}, S_n\big) \end{array} \right\},$$

where $\overleftarrow{\otimes}_{\mathrm{F}} \colon \tilde{\mathcal{D}}^{\star}_{\mathrm{F}} \to \tilde{\mathcal{D}}_{\mathrm{F}}$ *performs the left-associative composition, through* $\otimes_{\mathrm{F}}$*, of the sequence of descriptions constituting its argument. The auxiliary function* $\varpi_r \colon \bar{\mathcal{D}}_{\mathrm{F}} \times (\mathit{Vars}^{\star})^3 \rightarrowtail \bar{\mathcal{D}}_{\mathrm{F}}$ *is given, for each* $S \in \mathcal{D}_{\mathrm{F}}$*, and each* $\vec{\Lambda}_0, \bar{X}, \bar{Y} \in \mathit{Vars}^{\star}$ *of the same cardinality, by*

$$\varpi_r(S, \vec{\Lambda}_0, \bar{X}, \bar{Y}) \overset{\mathrm{def}}{=} \begin{cases} S[\bar{Y}/\vec{\Lambda}_0] \otimes_{\mathrm{F}} \mathrm{d}_{\bar{X}\bar{Y}}, & \textit{if } P_G[r] \textit{ is magic;} \\ S, & \textit{if } P_G[r] \textit{ is modified.} \end{cases}$$

The function $\eta_r \colon \tilde{\mathcal{D}}_{\mathrm{F}} \to \bar{\mathcal{D}}_{\mathrm{F}}$*, instead, is such that, for each* $D \in \mathcal{D}$*,*

$$\eta_r(D) \overset{\mathrm{def}}{=} \{D \mapsto D\},$$

whereas for each $S \in \mathcal{D}_{\mathrm{F}}$ *we have*

$$\eta_r(S) \overset{\mathrm{def}}{=} \big\{\, (D^{\mathrm{s}} \mapsto D^{\mathrm{s}}) \;\big|\; (D^{\mathrm{c}} \mapsto D^{\mathrm{s}}) \in S \,\big\},$$

if $P_G[r]$ *is magic, and*

$$\eta_r(S) \overset{\mathrm{def}}{=} S,$$

if $P_G[r]$ *is modified.*

Notice that we have provided a uniform treatment for both the magic and modified clauses. This required the introduction of the functions η_r and ϖ_r that encapsulate all the differences in the treatment of the two kinds of clauses. While η_r is used to make the success-pattern of magic clauses act as a call-pattern for modified clauses (the magic clauses serve the only purpose of characterizing the *call-conditions* for the original clauses), ϖ_r is used to rename and to performing parameter passing for the call-success pairs associated to magic atoms, but only in the body of magic clauses. Here is a more "operational" explanation of how $T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r, I)$ is computed.

1. Take a normalized version of the r-th clause in P_G. This can be a modified clause (if $1 \leq R \leq \# P$), or it can be a magic clause (if $\# P < r \leq \# P_G$).

2. If the r-th clause has no magic atom in the body ($M = \varnothing$), the *call-description* S_0 is simply $\{\mathbf{1} \mapsto \mathbf{1}\}$. Skip to step 7.

3. Otherwise M must be a singleton, say, $\bigl\{\mathtt{m\_}q(\bar{X})\bigr\}$. If r corresponds to a modified clause, then $\mathtt{m\_}q(\bar{X})$ is indeed $\mathtt{m\_}p(\vec{\Lambda})$. If, instead, r corresponds to a magic clause, then $\mathtt{m\_}q(\bar{X})$ can be different from $\mathtt{m\_}p(\vec{\Lambda})$.

4. Take a normalized version of any magic clause describing the magic predicate $\mathtt{m\_}q$. Suppose this is a version of the r_0-th clause.

5. Take a tuple of variables whose cardinality is the same as the the arity of q (same cardinality of $\vec{\Lambda}_0$), disjoint from the free variables occurring in the version of the r-th clause selected at step 1, and without variables belonging to the special set of variables Λ.

6. Take the functional description associated to r_0 in the current interpretation I and call it $\tilde{S}_0$. If r corresponds to a magic clause $\tilde{S}_0$ needs to be renamed and to have the parameter passing element added to it. If r corresponds to a clause $\tilde{S}_0$ needs nothing. Let S_0 be the outcome of this step.

7. Choose clauses $r_1, \ldots, r_n$ to "resolve" against the other atoms in the body of the r-th clause.

8. For each $i = 1, \ldots, n$, let S_i the description associated to the r_i-th clause in the current interpretation I, suitably renamed.

9. Put the pieces together applying $\otimes_{\mathrm{F}}$ in a left-associative way. Call S the result.

10. If r corresponds to a magic clause, transform S suitably: calls to magic clauses are meaningless, whereas the successes of magic clauses represent calls to ordinary clauses. If r, corresponds to a modified clause, do nothing.

11. Project the outcome of step 10 onto the variables $\vec{\Lambda}$. Put the result into the set of functional descriptions we are building.

12. Repeat until all the choices for $r_0, r_1, \dots, r_n$ in the above process have been made in all the possible ways. This union of all the description so obtained is the new interpretation for the r-th clause.

Now, the *concrete iteration sequence* [CC92b] that is used for the bottom-up construction of the semantics is inductively defined as follows, for each ordinal $\kappa \in \mathbb{O}$:

$$\begin{cases} T_{P_G}^{\bar{\mathcal{D}}_F} \uparrow 0 \stackrel{\text{def}}{=} \mathbf{0}_F, \\ T_{P_G}^{\bar{\mathcal{D}}_F} \uparrow (\kappa+1) \stackrel{\text{def}}{=} T_{P_G}^{\bar{\mathcal{D}}_F}\bigl(T_{P_G}^{\bar{\mathcal{D}}_F} \uparrow \kappa\bigr), \\ T_{P_G}^{\bar{\mathcal{D}}_F} \uparrow \kappa \stackrel{\text{def}}{=} \bigcup_{\beta<\kappa}\bigl(T_{P_G}^{\bar{\mathcal{D}}_F} \uparrow \beta\bigr), \\ \qquad \text{when } \kappa > 0 \text{ is a limit ordinal.} \end{cases}$$

The $\subseteq$ relation is the *computational order* of the functional representation [CC92b], capturing the relationship between successive iterates in the iteration sequence. Notice that this is one of the cases in abstract interpretation where the approximation and computational orderings are different. In [Scu96] it is shown that $T_{P_G}^{\bar{\mathcal{D}}_F}$ is continuous on the complete lattice $\bigl(\mathcal{I}_{P_G}^{\bar{\mathcal{D}}_F}, \subseteq\bigr)$. This implies the existence of the least fixpoint of $T_{P_G}^{\bar{\mathcal{D}}_F}$ and allows us to define the result of the bottom-up semantic construction as

$$\mathcal{F}_{P_G}^{\bar{\mathcal{D}}_F} \stackrel{\text{def}}{=} \operatorname{lfp}\Bigl(T_{P_G}^{\bar{\mathcal{D}}_F}\Bigr) = T_{P_G}^{\bar{\mathcal{D}}_F} \uparrow \omega. \tag{7.6}$$

Moreover, we have the following important result.

Theorem 117 (Equivalence.) *Let $\bar{\mathcal{D}}$ be a* strong *interpretation domain and $\bar{\mathcal{D}}_F$ the functional representation over $\bar{\mathcal{D}}$. Given a* CLP($\mathcal{X}$) *program $\tilde{P}$, a set $\tilde{G}$ of* CLP($\mathcal{X}$) *goals, and a constraint interpretation $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}}$, let P and G be the* CLP($\bar{\mathcal{D}}$) *program and goals obtained from $\tilde{P}$ and $\tilde{G}$ through $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}}$. Let also P_G be the* CLP($\bar{\mathcal{D}}$) *program obtained from $\tilde{P}$, $\tilde{G}$, and $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}}$ as in* Definition 116*. Then, if we define $\sigma\colon \mathcal{I}_{P_G}^{\bar{\mathcal{D}}_F} \to \mathcal{I}_{P_G}^{\bar{\mathcal{D}}_F}$ as*

$$\lambda I \in \mathcal{I}_{P_G}^{\bar{\mathcal{D}}_F} \,.\, \lambda r \in \{1, \dots, \# P\} \,. \\ \bigl\{\, (D^{c} \mapsto D^{s}) \bigm| (D^{c} \mapsto D^{s}) \in I(r),\ D^{s} \neq \mathbf{0} \,\bigr\}$$

we have

$$\sigma\bigl(\mathcal{O}_{\bar{\mathcal{D}}_F}(P, G)\bigr) = \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_F} \restriction \{1, \dots, \# P\}.$$

Moreover, if we denote by calls(r) *the subset of $\{\# P+1, \dots, \# P_G\}$ containing the indices of the magic clauses corresponding to calls of the predicate* *defined by the r-th clause, then, for each $r = 1, \ldots, \# P$,*

$$(D \mapsto \mathbf{0}) \in \mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P, G)(r) \implies \exists r' \in \mathrm{calls}(r) \,.\, (D \mapsto D) \in \mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r').$$

What σ does is just to strip the *failure call-patterns*, i.e., elements of the form $(D^{\mathrm{c}} \mapsto \mathbf{0})$, from the functional interpretation obtained through the top-down construction. The restriction to $\{1, \ldots, \# P\}$ in the first statement is also necessary, since the transformed program contains also the magic clauses that were used to derive call-patterns.

Roughly speaking, a call-pattern is said to be *correct* if it brings to at least one successful computation [GM92]. Of course, observing correct call-patterns only (instead of *any* call-pattern) does not provide much information for the typical applications of data-flow analysis. The first statement of the theorem says that all and only the *possibly correct* call-patterns[7] for the r-th original clause can be found in $\mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r)$. The second statement specifies that any other call-pattern for the r-th clause is contained in $\mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r')$, for some index r' such that, if the r-th clause defines predicate p, then the r'-th clause defines predicate `m_`p.

We have thus a semantic construction that, for the purpose of capturing both call- and success-patterns, is as good as the operational one. Moreover, this construction can be easily abstracted for the purpose of data-flow analysis. This is the subject of the following section.

7.4 Abstract Interpretation

The problem of ensuring the correctness of the analysis is dealt with much in the same way as in Section 3.2.5 on page sec:hierarchy-non-standard-semantics.

Definition 118 (Abstraction function.) *Let $\bar{\mathcal{D}}^\natural$ and $\bar{\mathcal{D}}^\sharp$ be two interpretation domains. A function $\alpha \colon \mathcal{D}^\natural \to \mathcal{D}^\sharp$ is an* abstraction function *of $\bar{\mathcal{D}}^\natural$ into $\bar{\mathcal{D}}^\sharp$ if and only if:*

A_1. *α is a* semi-morphism, *namely, for each $D^\natural, D^\natural_1, D^\natural_2 \in \mathcal{D}^\natural$ and $\bar{X}, \bar{Y} \in$ Vars*:*

$$\begin{aligned}
\alpha(D^\natural_1 \otimes^\natural D^\natural_2) &\trianglelefteq^\sharp \alpha(D^\natural_1) \otimes^\sharp \alpha(D^\natural_2),\\
\alpha(D^\natural_1 \oplus^\natural D^\natural_2) &\trianglelefteq^\sharp \alpha(D^\natural_1) \oplus^\sharp \alpha(D^\natural_2),\\
\alpha(\mathbf{0}^\natural) &\trianglelefteq^\sharp \mathbf{0}^\sharp,\\
\alpha\big(\exists\!\!\!\exists^\natural_{\bar{X}}\, D^\natural\big) &\trianglelefteq^\sharp \exists\!\!\!\exists^\sharp_{\bar{X}}\, \alpha(D^\natural),\\
\alpha\big(\mathrm{d}^\natural_{\bar{X}\bar{Y}}\big) &\trianglelefteq^\sharp \mathrm{d}^\sharp_{\bar{X}\bar{Y}};
\end{aligned}$$

[7]If $\bar{\mathcal{D}}$ is an abstract domain we might not be able to detect all the failures.

A_2. for each increasing chain $\{D^\natural_j \in \mathcal{D}^\natural\}_{j\in\mathbb{N}}$, and each $D^\sharp \in \mathcal{D}^\sharp$,

$$\forall j \in \mathbb{N} : \alpha(D^\natural_j) \trianglelefteq^\sharp D^\sharp \quad \Longrightarrow \quad \alpha\Biggl(\bigoplus_{j\in\mathbb{N}}{}^\natural D^\natural_j\Biggr) \trianglelefteq^\sharp D^\sharp;$$

A_3. for each $D^\natural \in \mathcal{D}^\natural$ and each renaming $[\bar{Y}/\bar{X}]$ for $D^\natural$, we have

$$\alpha(|D^\natural)[\bar{Y}/\bar{X}] = \alpha\bigl(D^\natural[\bar{Y}/\bar{X}]\bigr).$$

Each abstraction function $\alpha\colon \mathcal{D}^\natural \to \mathcal{D}^\sharp$ is extended to $\alpha\colon {\mathcal{D}_\mathrm{F}}^\natural \to {\mathcal{D}_\mathrm{F}}^\sharp$ point-wise: for each $S \in {\mathcal{D}_\mathrm{F}}^\natural$,

$$\alpha(S) \stackrel{\mathrm{def}}{=} \Bigl\{ \bigl(\alpha(D^\natural_1) \mapsto \alpha(D^\natural_2)\bigr) \Bigm| (D^\natural_1 \mapsto D^\natural_2) \in S \Bigr\}. \tag{7.7}$$

It can be shown that the extended function is an abstraction function along the lines of Definition 118.

When we use a functional representation as an abstract domain for data-flow analysis, we must ensure termination. If the domain that is used as a base for the functional construction is finite, also the corresponding functional representation is so. In any case we guarantee or accelerate termination by using widening operators [CC77, CC92b]. Here we give a simplified definition of this device for convergence enforcement and acceleration, in the case of the functional representation.

Definition 119 (Widening.) [CC77] *Given a functional representation $\bar{\mathcal{D}}^\sharp_\mathrm{F}$, a binary operator $\nabla^\sharp_\mathrm{F}\colon {\mathcal{D}_\mathrm{F}}^\sharp \to {\mathcal{D}_\mathrm{F}}^\sharp$ is called a* widening for $\bar{\mathcal{D}}^\sharp_\mathrm{F}$ *if*

W_1. for each $S_1, S_2 \in {\mathcal{D}_\mathrm{F}}^\sharp$ we have $S_1 \trianglelefteq_\mathrm{F} S_1 \nabla^\sharp_\mathrm{F} S_2$ and $S_2 \trianglelefteq_\mathrm{F} S_1 \nabla^\sharp_\mathrm{F} S_2$;

W_2. for each increasing chain $S_0 \subseteq S_1 \subseteq S_2 \subseteq S_3 \subseteq \cdots$, the sequence given by $S'_0 \stackrel{\mathrm{def}}{=} S_0$ and, for $n \geq 1$, $S'_n \stackrel{\mathrm{def}}{=} S'_{n-1} \nabla^\sharp_\mathrm{F} S_n$, is stationary after some $k \in \mathbb{N}$.

The following result is an application of a theorem in [CC92b, Proposition 6.20]. The soundness of the approach clearly follows.

Theorem 120 *Let P and G be a* CLP($\mathcal{X}$) *program and a set of* CLP($\mathcal{X}$) *goals, respectively. Let $\bar{\mathcal{D}}^\natural$ and $\bar{\mathcal{D}}^\sharp$ be two interpretation domains, and let $[\![\cdot]\!]^{\bar{\mathcal{D}}^\natural}_\mathsf{C}$ and $[\![\cdot]\!]^{\bar{\mathcal{D}}^\sharp}_\mathsf{C}$ be two constraint interpretations for C (the language of constraints of $\mathcal{X}$). Let $\alpha\colon \bar{\mathcal{D}}^\natural \to \bar{\mathcal{D}}^\sharp$ be an abstraction function such that $\alpha \circ [\![\cdot]\!]^{\bar{\mathcal{D}}^\natural}_\mathsf{C} \trianglelefteq^\sharp [\![\cdot]\!]^{\bar{\mathcal{D}}^\sharp}_\mathsf{C}$. Let $P^\natural$ be the* CLP($\bar{\mathcal{D}}^\natural$) *program obtained by applying the Magic Templates algorithm to P and G and interpreting the result over $\bar{\mathcal{D}}$ through $[\![\cdot]\!]^{\bar{\mathcal{D}}^\natural}_\mathsf{C}$. Likewise, $P^\sharp$ is obtained from $\tilde{P}$, $\tilde{G}$, and $[\![\cdot]\!]^{\bar{\mathcal{D}}^\sharp}_\mathsf{C}$. Let also $\nabla^\sharp_\mathrm{F}$ a* *widening operator over* $\bar{\mathcal{D}}^\sharp_{\mathrm{F}}$. *Consider the* abstract iteration sequence for $P^\sharp$ with widening $\nabla^\sharp_{\mathrm{F}}$ *inductively defined by*

$$\begin{cases} T^{\bar{\mathcal{D}}^\sharp}_{P^\sharp} \Uparrow 0 \stackrel{\text{def}}{=} \mathbf{0}^\sharp_{\mathrm{F}}, \\ T^{\bar{\mathcal{D}}^\sharp}_{P^\sharp} \Uparrow (k+1) \stackrel{\text{def}}{=} (T^{\bar{\mathcal{D}}^\sharp}_{P^\sharp} \Uparrow k) \nabla^\sharp_{\mathrm{F}} T^{\bar{\mathcal{D}}^\sharp}_{P^\sharp} (T^{\bar{\mathcal{D}}^\sharp}_{P^\sharp} \Uparrow k), \quad \textit{for } k \in \mathbb{N}. \end{cases} \tag{7.8}$$

The abstract iteration is eventually stable after $\ell \in \mathbb{N}$ *steps and*

$$\alpha\Bigl(\mathrm{lfp}(T^{\bar{\mathcal{D}}^\natural}_{P^\natural})\Bigr) = \alpha\Bigl(T^{\bar{\mathcal{D}}^\natural}_{P^\natural} \uparrow \omega\Bigr) \trianglelefteq^\sharp_{\mathrm{F}} (T^{\bar{\mathcal{D}}^\sharp}_{P^\sharp} \Uparrow \ell).$$

Thus we have (again) an *abstract compilation* approach, where the soundness of the *compilation function* $[\![\cdot]\!]^{\bar{\mathcal{D}}^\sharp}_{\mathsf{C}}$ is expressed by the requirement $\alpha \circ [\![\cdot]\!]^{\bar{\mathcal{D}}^\natural}_{\mathsf{C}} \trianglelefteq^\sharp [\![\cdot]\!]^{\bar{\mathcal{D}}^\sharp}_{\mathsf{C}}$

Concerning the precision of the analysis, observe that the abstract iteration given above, is much simplified. In reality, especially with “domains” like the functional ones we employ, one uses *a family of widening operators*. In principle, it is possible to use a different widening operator at each iteration step. Roughly speaking, widening operators coming early in the sequence do not lose much precision. Later in the sequence, when the *shape* of the fixpoint is more likely to have been established, more aggressive widening operators come into play [Bou92]. There is not much more to say, in general, since the design of widening operators (and the overall precision of the analysis) is highly dependent from the actual domain employed (see, e.g., [CH78]).

7.5 Conclusion

We have presented a generalized framework for the semantics of the entire class of CLP languages. The approach, which is based on the *Magic Templates* program transformation and bottom-up evaluation, is able to capture both call- and success-patterns without loss of precision. The framework can accommodate a very wide range of non-standard semantics and, in particular, abstract interpretations. A remarkable feature is that the entire construction is almost automatic: given any abstract domain we provide a standard way of upgrading it and using it to obtain a precise bottom-up computation of the abstract semantics. The only thing that must be dealt with in a domain-dependent way is providing suitable widening operators so to ensure the convergence of the method.

7.6 Proof of the Main Result

In this section we present the (overly complicated) proof of the main result of this chapter. It would be very nice to find a simpler proof, of course.

As mentioned previously, Definition 109 on page def:interpretation-domain does not specify some important, though very reasonable, requirements that interpretation domains must satisfy. Here we make them explicit.

Hypothesis 121 (Further axioms for i.d.) *The operators of an interpretation domain $\bar{\mathcal{D}}$ "do not invent new free variables". That is to say that, for each $\bar{X}, \bar{Y} \in Vars^{\star}$ and each $C_1, C_2 \in \bar{\mathcal{D}}$:*

H_1. *we have*

$$FV(C_1 \otimes C_2) \subseteq FV(C_1) \cup FV(C_2),$$
$$FV(C_1 \oplus C_2) \subseteq FV(C_1) \cup FV(C_2);$$

H_2. $FV\big(\mathrm{d}_{\bar{X}\bar{Y}}\big) \subseteq \bar{X} \cup \bar{Y}$.

Moreover, the operators are generic*, or, in other words, they are insensible to variables' names:*

H_3. *if $[\bar{Y}/\bar{X}]$ is a renaming for both C_1 and C_2, then*

$$(C_1 \otimes C_2)[\bar{Y}/\bar{X}] = C_1[\bar{Y}/\bar{X}] \otimes C_2[\bar{Y}/\bar{X}]$$

and likewise for the other operators;

H_4. *if $[\bar{Y}/\bar{X}]$ is a renaming for C, then*

$$\big(\bar{\exists}_{\bar{X}}\, C\big)[\bar{Y}/\bar{X}] = \bar{\exists}_{\bar{Y}}\big(C[\bar{Y}/\bar{X}]\big);$$

H_5. *if $[\bar{Z}/\bar{Y}]$ is a renaming for C, then*

$$\big(C[\bar{X}/\bar{Y}]\big)[\bar{Z}/\bar{X}] = C[\bar{Z}/\bar{Y}];$$

H_6. *if $[\bar{Z}/\bar{Y}]$ is a renaming for C, $\bar{X} \cap \bar{Y} = \varnothing$, and $\bar{Z} \cap \bar{X} = \varnothing$, then*

$$\bar{\exists}_{\bar{X}}\, C = \bar{\exists}_{\bar{X}}\big(C[\bar{Z}/\bar{Y}]\big).$$

H_7. *if $[\bar{Y}/\bar{X}]$ is a renaming for both C_1 and C_2, then*

$$C_1 \trianglelefteq C_2 \quad \Longrightarrow \quad C_1[\bar{Y}/\bar{X}] \trianglelefteq C_2[\bar{Y}/\bar{X}].$$

Proposition 122 (Functional representation's properties.) *Let $\bar{\mathcal{D}}$ be any domain, and $\bar{\mathcal{D}}_{\mathrm{F}}$ the functional representation built over it. Then the following properties hold:*

1. *for each $S \in \mathcal{D}_{\mathrm{F}}$ we have $\mathbf{0}_{\mathrm{F}} \otimes_{\mathrm{F}} S = S \otimes_{\mathrm{F}} \mathbf{0}_{\mathrm{F}} = \mathbf{0}_{\mathrm{F}}$;*

2. *$\langle \mathcal{D}_{\mathrm{F}}, \trianglelefteq_{\mathrm{F}} \rangle$ is a preorder with minimum element $\mathbf{0}_{\mathrm{F}}$;*

3. *$\otimes_{\mathrm{F}} \colon \mathcal{D}_{\mathrm{F}} \times \mathcal{D}_{\mathrm{F}} \to \mathcal{D}_{\mathrm{F}}$ (i.e., $\otimes_{\mathrm{F}}$ restricted to $\mathcal{D}_{\mathrm{F}}$) is componentwise monotonic with respect to $\subseteq$ and completely left- and right-distributive with respect to $\cup$;*

4. *$\bar{\exists}^{\mathrm{F}}_{\bar{X}}$ is monotonic with respect to $\subseteq$ and continuous with respect to $\cup$;*

5. *for each family $\{S_j \in \mathcal{D}_{\mathrm{F}}\}_{j\in\mathbb{N}}$ and each $S \in \mathcal{D}_{\mathrm{F}}$,*

$$\forall j \in \mathbb{N} : S_j \trianglelefteq_{\mathrm{F}} S \quad \Longrightarrow \quad \bigcup_{j\in\mathbb{N}} S_j \trianglelefteq_{\mathrm{F}} S.$$

Proof The statements are all trivial consequences of the definition. We just show that $\otimes_{\mathrm{F}}$ is completely distributive with respect to $\cup$. Consider a family $\{S_i \in \mathcal{D}_{\mathrm{F}}\}_{i\in\mathbb{N}}$. We must show that

$$S \otimes_{\mathrm{F}} \bigcup_{i\in\mathbb{N}} S_i = \bigcup_{i\in\mathbb{N}} (S \otimes_{\mathrm{F}} S_i)$$

and

$$\left(\bigcup_{i\in\mathbb{N}} S_i\right) \otimes_{\mathrm{F}} S = \bigcup_{i\in\mathbb{N}} (S_i \otimes_{\mathrm{F}} S).$$

For each $k \in \mathbb{N}$, from $S_k \subseteq \bigcup_{i\in\mathbb{N}} S_i$ and the monotonicity of $\otimes_{\mathrm{F}}$ with respect to $\subseteq$ we get

$$(S \otimes_{\mathrm{F}} S_k) \subseteq S \otimes_{\mathrm{F}} \left(\bigcup_{i\in\mathbb{N}} S_i\right),$$

and thus

$$\bigcup_{i\in\mathbb{N}} (S \otimes_{\mathrm{F}} S_i) \subseteq S \otimes_{\mathrm{F}} \left(\bigcup_{i\in\mathbb{N}} S_i\right).$$

By the same kind of reasoning we derive also

$$\bigcup_{i\in\mathbb{N}} (S_i \otimes_{\mathrm{F}} S) \subseteq \left(\bigcup_{i\in\mathbb{N}} S_i\right) \otimes_{\mathrm{F}} S.$$

For the reverse inclusions, by definition of $\otimes_{\mathrm{F}}$ we have that for each $(D^{\mathrm{c}} \mapsto D^{\mathrm{s}}) \in S \otimes_{\mathrm{F}} \bigcup_{i\in\mathbb{N}} S_i$ there must exists $h \in \mathbb{N}$ such that $(D^{\mathrm{c}} \mapsto D^{\mathrm{s}}) \in S \otimes_{\mathrm{F}} S_h$. We can thus conclude that

$$S \otimes_{\mathrm{F}} \left(\bigcup_{i\in\mathbb{N}} S_i\right) \subseteq \bigcup_{i\in\mathbb{N}} (S \otimes_{\mathrm{F}} S_i),$$

and, by a similar argument, that

$$\Big(\bigcup_{i\in\mathbb{N}} S_i\Big) \otimes_{\mathrm{F}} S \subseteq \bigcup_{i\in\mathbb{N}} (S_i \otimes_{\mathrm{F}} S).$$

□

Lemma 123 *For each $C_1, C_2 \in \mathcal{D}$ and each $\bar{X}, \bar{Y}, \bar{Z} \in \mathit{Vars}^\star$, if the conditions*

$$FV(C_1) \cap FV(C_2) = \varnothing, \qquad FV(C_1) \cap \bar{X} \subseteq \bar{Y}, \qquad FV(C_1) \cap \bar{Z} = \varnothing,$$

hold, then $\bar{\exists}_{\bar{X}}\big(C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}} \otimes C_2\big) = \bar{\exists}_{\bar{X}}\big(\bar{\exists}_{\bar{Y}}\, C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}} \otimes C_2\big)$.

Proof By properties H_1 and H_2 of Hypothesis 121 we have that

$$FV\big(\mathrm{d}_{\bar{Y}\bar{Z}} \otimes C_2\big) \subseteq FV(C_2) \cup \bar{Y} \cup \bar{Z},$$

then

$$\begin{aligned}
&FV(C_1) \cap FV\big(\mathrm{d}_{\bar{Y}\bar{Z}} \otimes C_2\big)\\
&\qquad \subseteq \big(FV(C_1) \cap FV(C_2)\big) \cup \big(FV(C_1) \cap \bar{Y}\big) \cup \big(FV(C_1) \cap \bar{Z}\big)\\
&\qquad = FV(C_1) \cap \bar{Y}\\
&\qquad [\text{since } FV(C_1) \cap FV(C_2) = \varnothing \text{ and } FV(C_1) \cap \bar{Z} = \varnothing]\\
&\qquad \subseteq \bar{Y}
\end{aligned}$$

Thus, since $FV(C_1) \cap \bar{X} \subseteq \bar{Y}$, condition S_7 of Definition 110 implies the thesis. □

Lemma 124 *For each $C_1, C_2 \in \mathcal{D}$ and each $\bar{X}, \bar{Y}, \bar{Z} \in \mathit{Vars}^\star$, if the conditions*

$$FV(C_1) \cap \bar{Y} = \varnothing, \qquad \bar{Y} \cap \bar{X} = \varnothing, \qquad FV(C_2) \cap \bar{Z} = \varnothing,$$

hold, then

$$\bar{\exists}_{\bar{X}}\big(C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}} \otimes \bar{\exists}_{\bar{Y}}\, C_2\big) = \bar{\exists}_{\bar{X}}\big(C_1 \otimes \bar{\exists}_{\bar{Z}}(C_2[\bar{Z}/\bar{Y}])\big).$$

Proof By properties H_1 and H_2 of Hypothesis 121 we have that

$$FV\big(\mathrm{d}_{\bar{Y}\bar{Z}} \otimes \bar{\exists}_{\bar{Y}}\, C_2\big) \subseteq \bar{Y} \cup \bar{Z},$$

and then

$$\begin{aligned}
FV(C_1) \cap FV\bigl(\mathrm{d}_{\bar{Y}\bar{Z}} \otimes \bar{\exists}_{\bar{Y}}\, C_2\bigr) &\subseteq FV(C_1) \cap (\bar{Y} \cup \bar{Z}) \\
&= \bigl(FV(C_1) \cap \bar{Y}\bigr) \cup \bigl(FV(C_1) \cap \bar{Z}\bigr) \\
&= FV(C_1) \cap \bar{Z} \\
&\text{[since } FV(C_1) \cap \bar{Y} = \varnothing\text{]} \\
&\subseteq \bar{Z}.
\end{aligned}$$

Moreover,

$$\begin{aligned}
FV\bigl(\mathrm{d}_{\bar{Y}\bar{Z}} \otimes \bar{\exists}_{\bar{Y}}\, C_2\bigr) \cap \bar{X} &\subseteq \bigl(\bar{Y} \cup \bar{Z}\bigr) \cap \bar{X} \\
&= \bigl(\bar{Y} \cap \bar{X}\bigr) \cup \bigl(\bar{Z} \cap \bar{X}\bigr) \\
&= \bar{Z} \cap \bar{X} \\
&\text{[since } \bar{Y} \cap \bar{X} = \varnothing\text{]} \\
&\subseteq \bar{Z}.
\end{aligned}$$

Then, by condition S_7 of Definition 110, we can write:

$$\begin{aligned}
\bar{\exists}_{\bar{X}}\bigl(C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}} \otimes \bar{\exists}_{\bar{Y}}\, C_2\bigr) &= \bar{\exists}_{\bar{X}}\Bigl(C_1 \otimes \bar{\exists}_{\bar{Z}}\bigl(\mathrm{d}_{\bar{Y}\bar{Z}} \otimes \bar{\exists}_{\bar{Y}}\, C_2\bigr)\Bigr) \\
&= \bar{\exists}_{\bar{X}}\Bigl(C_1 \otimes \bar{\exists}_{\bar{Z}}\Bigl(\bigl(\bar{\exists}_{\bar{Y}}\, C_2\bigr)[\bar{Z}/\bar{Y}]\Bigr)\Bigr) \\
&\text{[by } S_5 \text{ with } [\bar{Z}/\bar{Y}] = \rho\text{]} \\
&= \bar{\exists}_{\bar{X}}\Bigl(C_1 \otimes \bar{\exists}_{\bar{Z}}\Bigl(\bar{\exists}_{\bar{Z}}\bigl(C_2[\bar{Z}/\bar{Y}]\bigr)\Bigr)\Bigr) \\
&\text{[by } H_4 \text{ since } FV(C_2) \cap \bar{Z} = \varnothing\text{]} \\
&= \bar{\exists}_{\bar{X}}\Bigl(C_1 \otimes \bar{\exists}_{\bar{Z}}\bigl(C_2[\bar{Z}/\bar{Y}]\bigr)\Bigr). \\
&\text{[by } S_6 \text{ (i.e., idempotence)]}
\end{aligned}$$

□

Lemma 125 *For each $C_1, C_2 \in \mathcal{D}$, each $\bar{X}, \bar{Y}, \bar{Z} \in Vars^\star$, and each renaming ρ such that $\bar{Y}\rho = \bar{Z}\rho = \bar{Z}$*

$$\bar{\exists}_{\bar{X}}\bigl(C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}} \otimes \bar{\exists}_{\bar{Z}}(C_1\rho) \otimes C_2\bigr) = \bar{\exists}_{\bar{X}}\bigl(C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}} \otimes C_2\bigr).$$

Proof By S_5 we have $\bar{\exists}_{\bar{Z}}(C_1\rho) = \bar{\exists}_{\bar{Z}}\bigl(C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}}\bigr)$, thus

$$\begin{aligned}
&\bar{\exists}_{\bar{X}}\bigl(C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}} \otimes \bar{\exists}_{\bar{Z}}(C_1\rho) \otimes C_2\bigr) \\
&\qquad = \bar{\exists}_{\bar{X}}\bigl(C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}} \otimes \bar{\exists}_{\bar{Z}}\bigl(C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}}\bigr) \otimes C_2\bigr) \\
&\qquad = \bar{\exists}_{\bar{X}}\bigl(C_1 \otimes \mathrm{d}_{\bar{Y}\bar{Z}} \otimes C_2\bigr) \\
&\qquad \text{[by } S_2\text{]}
\end{aligned}$$

□

Lemma 126 *For each initial goal G_0, consider the partial derivation*

$$\begin{aligned} G_0 &\rightsquigarrow^*_P \hat{C}^{\mathrm{c}} \,\square\, p(\bar{X}) :: B \\ &\rightsquigarrow_{\{R_k\}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \,\square\, \bigl(p_1(\bar{Y}_1), \dots, p_n(\bar{Y}_n)\bigr) :: B \\ &\rightsquigarrow^*_P \hat{C}^{\mathrm{s}} \,\square\, \bigl(p_i(\bar{Y}_i), \dots, p_n(\bar{Y}_n)\bigr) :: B, \end{aligned} \tag{7.9}$$

where

$$\hat{C}^{\mathrm{s}} = \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{i=1}^{n} (\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes C_i).$$

and $R_k \equiv p(\bar{Y}) :- C\rho \,\square\, p_1(\bar{Y}_1), \dots, p_n(\bar{Y}_n)$ is a variant of $P[r]$ such that $FV(R_k) \cap \bigl(FV(G_0) \cup \bigcup_{j=1}^{k-1} FV(R_j)\bigr) = \varnothing$, and $\bar{U}\rho = \bar{Y}$, $\bar{U}_1\rho = \bar{Y}_1$, ..., $\bar{U}_n\rho = \bar{Y}_n$. Then,

$$\begin{aligned} \overline{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{s}} &= \overline{\exists}_{\bar{X}} \bigl(\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{i=1}^{n} (\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes C_i)\bigr) \\ &= \overline{\exists}_{\bar{X}} \bigl(\overline{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{i=1}^{n} (\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes \overline{\exists}_{\bar{X}_i}\, C_i)\bigr). \end{aligned} \tag{7.10}$$

Proof By the derivation (7.9) there must exist derivations

$$\begin{aligned} \mathbf{1} \,\square\, p_i(\bar{Y}_i) &\rightsquigarrow_{\{R_{k+h_i}\}} \mathbf{1} \otimes \mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes C'_i\rho_i \,\square\, B_{r_i} \\ &\rightsquigarrow^*_P \mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes C_i \,\square\, \varnothing \end{aligned} \tag{7.11}$$

such that $h_i > 0$ and $R_{k+h_i} = \bigl(p_i(\bar{X}_i) :- C'_i\rho_i \,\square\, B_{r_i}\bigr)$ is a variant of $P[r_i]$ with $FV(R_{k+h_i}) \cap \bigl(FV(G_0) \cup \bigcup_{j=1}^{k+h_i-1} FV(R_j)\bigr) = \varnothing$. Obviously, $h_i < h_{i+1}$. Observe also that in (7.11) we have used the same variants employed in (7.9). This implies that $FV(C_i) \cap \bar{Y}_i = \varnothing$.

Consider the expressions

$$\begin{aligned} \overline{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{s}} &= \overline{\exists}_{\bar{X}} \Bigl(\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes \overbrace{C\rho \otimes \bigotimes_{i=1}^{n} \bigl(\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes C_i\bigr)}^{E}\Bigr) \\ &= \overline{\exists}_{\bar{X}} \bigl(\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes E\bigr). \end{aligned} \tag{7.12}$$

We apply Lemma 123 to (7.12). The hypotheses are readily checked exploiting point 3 of Definition 114 on page def:finite-derivation:

$$\begin{aligned} FV(C) \cap \bar{X} &\subseteq \bar{X}; \\ FV(C) \cap FV(E) &\subseteq \left(\bigcup_{j=1}^{k-1} FV(R_j)\right) \cap \left(\bigcup_{j=k}^{N} FV(R_j)\right) \\ &= \varnothing; \\ FV(C) \cap \bar{Y} &\subseteq \left(\bigcup_{j=1}^{k-1} FV(R_j)\right) \cap \left(\bigcup_{j=k}^{N} FV(R_j)\right) \\ &= \varnothing. \end{aligned}$$

Thus we obtain

$$\begin{aligned}\bar{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{s}} &= \bar{\exists}_{\bar{X}}\Big(\bar{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes E\Big)\\ &= \bar{\exists}_{\bar{X}}\Big(\bar{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{i=1}^{n} (\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes C_i)\Big). \qquad (7.13)\end{aligned}$$

We transform (7.13) by applying Lemma 123 again. In this case we must prove that, for each $i = 1, \dots, n$,

$$\bar{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{s}} = \bar{\exists}_{\bar{X}}\Big(\bar{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{i=1}^{n} (\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes \bar{\exists}_{\bar{X}_i}\,C_i)\Big). \qquad (7.14)$$

Consider, for each $i = 1, \dots, n$, the expression

$$\begin{aligned}E_i = \Big(&\bar{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho\\ &\otimes \bigotimes_{j=1}^{i-1} (\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes \bar{\exists}_{\bar{X}_j}\,C_j) \otimes \bigotimes_{j=i}^{n} (\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes C_j)\Big)\end{aligned}$$

The right-hand sides of (7.13) and (7.14) are given by E_1 and E_{n+1}, respectively. It is possible to prove that $E_1 = E_{n+1}$ by showing that $E_1 = E_2$, $E_2 = E_3$, … , $E_n = E_{n+1}$. The equality $E_i = E_{i+1}$ can be obtained by using Lemma 125. To this purpose, we consider the following expression for E_i:

$$\begin{aligned}E_i = &\overbrace{\Big(\bar{\exists}_{\bar{X}}\,C \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{j=1}^{i-1} (\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes \bar{\exists}_{\bar{X}_j}\,C_j)}^{C^{i-1}}\\ &\otimes \mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes C_i \otimes \underbrace{\bigotimes_{j=i+1}^{n} (\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes C_j)\Big)}_{C^{i+1}} \qquad (7.15)\end{aligned}$$

By $\otimes$ commutativity and associativity, we can write

$$E_i = C_i \otimes \mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes (C^{i-1} \otimes C^{i+1}). \qquad (7.16)$$

The hypotheses of Lemma 123 are checked, for each $i = 1, \dots, n$, as follows (still using point 3 of Definition 114):

$$\begin{aligned}
FV(C_i) \cap \bar{X} &\subseteq \Biggl(\bigcup_{j=k+h_i}^{k+h_{i+1}-1} FV(R_j) \Biggr) \cap \Biggl(\bigcup_{j=1}^{k} FV(R_j) \Biggr) \\
&= \varnothing \\
&\subseteq \bar{Y}_i; \\
FV(C_i) \cap FV(C^{i-1} &\otimes C^{i+1}) \\
&\subseteq FV(C_i) \cap \bigl(FV(C^{i-1}) \cup FV(C^{i+1}) \bigr) \\
&= \bigl(FV(C_i) \cap FV(C^{i-1}) \bigr) \cup \bigl(FV(C_i) \cap FV(C^{i+1}) \bigr) \\
&\subseteq \Biggl(\Biggl(\bigcup_{j=k+h_i}^{k+h_{i+1}-1} FV(R_j) \Biggr) \cap \Biggl(\bigcup_{j=1}^{k+h_i-1} FV(R_j) \Biggr) \Biggr) \\
&\cup \Biggl(\Biggl(\bigcup_{j=k+h_i}^{k+h_{i+1}-1} FV(R_j) \Biggr) \cap \Biggl(\bigcup_{j=k+h_{i+1}}^{N} FV(R_j) \Biggr) \Biggr) \\
&= \varnothing; \\
FV(C_i) \cap \bar{X}_i &\subseteq \Biggl(\bigcup_{j=k+h_i}^{k+h_{i+1}-1} FV(R_j) \Biggr) \cap FV(R_k) \\
&= \varnothing.
\end{aligned}$$

Thus, by Lemma 123 we can write, for each $i = 1, \ldots, n$,

$$E_i = \bar{\exists}_{\bar{X}_i} C_i \otimes \mathrm{d}_{\bar{Y}_i \bar{X}_i} \otimes (C^{i-1} \otimes C^{i+1}).$$

By substituting the expressions for C^{i-1} and C^{i+1} we have the thesis.

Theorem 127 (117) (Equivalence.) *Let $\bar{\mathcal{D}}$ be a* strong *interpretation domain and $\bar{\mathcal{D}}_{\mathrm{F}}$ the functional representation over $\bar{\mathcal{D}}$. Given a* CLP($\mathcal{X}$) *program $\tilde{P}$ and a set $\tilde{G}$ of* CLP($\mathcal{X}$) *goals, a constraint interpretation $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}}$, let P and G be the* CLP($\bar{\mathcal{D}}$) *program and goals obtained from $\tilde{P}$ and $\tilde{G}$ through $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}}$. Let also P_m^G be the* CLP($\bar{\mathcal{D}}$) *program obtained from $\tilde{P}$, $\tilde{G}$, and $[\![\cdot]\!]_{\mathsf{C}}^{\bar{\mathcal{D}}}$ as in* Definition 116*. Then*

$$\sigma\Bigl(\mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P, G)\Bigr) = \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} \restriction \{1, \ldots, \# P\}.$$

Moreover, for each $r \in \{1, \ldots, \# P\}$,

$$\begin{aligned}
&(C \mapsto \mathbf{0}) \in \mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P, G)(r) \implies \\
&\qquad \exists r' \in \bigl\{ \# P + 1, \ldots, \# P_m^G \bigr\} \,.\, (C \mapsto C) \in \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r'). \qquad (7.17)
\end{aligned}$$

Proof Since the proof, while not difficult, is quite complicated, we will adhere strictly to the following notational conventions. We will focus on the the r-th clause of the program, $P[r]$. It is given by

$$P[r] \stackrel{\text{def}}{=} \Big(p(\bar{U}) \mathrel{:-} C \mathbin{\Box} p_1(\bar{U}_1), \ldots , p_n(\bar{U}_n)\Big).$$

In the top-down derivations we will consider, in particular, a variant of $P[r]$, obtained from $P[r]$ through the application of a renaming ρ:

$$P[r]\rho \stackrel{\text{def}}{=} \Big(p(\bar{Y}) \mathrel{:-} C\rho \mathbin{\Box} p_1(\bar{Y}_1), \ldots , p_n(\bar{Y}_n)\Big),$$

whence it is clear that $\bar{U}\rho = \bar{Y}$, $\bar{U}_1\rho = \bar{Y}_1$, … , $\bar{U}_n\rho = \bar{Y}_n$.

Staten that p_i always denotes the predicate symbol of the i-th atom goal in the body of $P[r]$, we will also talk about n clauses in P defining p_i, for each $i = 1, \ldots , n$. The indexes for these clauses are $r_1, \ldots , r_n$, respectively. When, for some $i = 1, \ldots , n$, we use a variant of r_i in a derivation, we will write it as

$$p_i(\bar{X}_i) \mathrel{:-} C_i\rho_i \mathbin{\Box} B_i.$$

So far for the original program P and the variants of its clauses, we now deal with the magic program P_m^G. Recall that our transformation has the following property: the modified clause corresponding to $P[r]$ is $P_m^G[r]$; the magic clauses are in the range $\# P + 1, \ldots , \# P_m^G$. Moreover, the magic clauses are normalized so that the variable symbols that appear in their heads are initial segments of Λ, denoted by $\vec{\Lambda}$. Thus, the modified clause $P_m^G[r]$ is

$$p(\vec{\Lambda}) \mathrel{:-} \Big\langle \{\mathtt{m\_}p(\vec{\Lambda})\}, C\ddot{\rho}, \big(p_1(\bar{W}_1), \ldots , p_n(\bar{W}_n)\big) \Big\rangle,$$

for some renaming $\ddot{\rho}$ such that

$$\bar{U}\ddot{\rho} = \vec{\Lambda}, \bar{U}_1\ddot{\rho} = \bar{W}_1, \ldots , \bar{U}_n\ddot{\rho} = \bar{W}_n.$$

In the magic program we will also have n magic clauses, one for each call in the body of $P[r]$. These magic clauses will have indexes $r'_1, \ldots , r'_n$, and will define the predicates $\mathtt{m\_}p_1, \ldots , \mathtt{m\_}p_n$, respectively. For each $i = 1, \ldots , n$, the clause $P_m^G[r'_i]$ is

$$\mathtt{m\_}p_i(\vec{\Lambda}_i) \mathrel{:-} \Big\langle \{\mathtt{m\_}p(\bar{V}^i)\}, C\ddot{\rho}_i, \big(p_1(\bar{V}_1^i), \ldots , p_{i-1}(\bar{V}_{i-1}^i)\big) \Big\rangle,$$

where $\bar{U}\ddot{\rho}_i = \bar{V}^i$, $\bar{U}_1\ddot{\rho}_i = \bar{V}_1^i$, … , $\bar{U}_{i-1}\ddot{\rho}_i = \bar{V}_{i-1}^i$, and $\bar{U}_i\ddot{\rho}_i = \vec{\Lambda}_i$.

In the proof we will consider, among other things, a derivation such that:

1. it starts with a goal G_0 drawn from the set G of initial goals;

2. after a certain number of derivation steps (possibly 0), it performs a step employing a variant of $P[r]$;
3. it performs other derivation steps and stops as soon as one of the following condition holds:
   - a goal with an unsatisfiable constraint store is reached;
   - the last call in the body of $P[r]$ "returns".

Such a derivation will be written as follows:

$$\begin{array}{ll} G_0 \rightsquigarrow_P^* & \hat{C}^{\mathrm{c}} \,\Box\, p(\bar{X}) :: B \\ \rightsquigarrow_{\{r\}} & \hat{C}_1^{\mathrm{c}} \,\Box\, \bigl(p_1(\bar{Y}_1), \ldots, p_n(\bar{Y}_n)\bigr) :: B \\ \rightsquigarrow_P^* & \hat{C}_1^{\mathrm{s}} \,\Box\, \bigl(p_2(\bar{Y}_2), \ldots, p_n(\bar{Y}_n)\bigr) :: B \\ = & \hat{C}_2^{\mathrm{c}} \,\Box\, \bigl(p_2(\bar{Y}_2), \ldots, p_n(\bar{Y}_n)\bigr) :: B \\ \rightsquigarrow_P^* & \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \vdots & \qquad\qquad\vdots \\ \rightsquigarrow_P^* & \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \rightsquigarrow_P^* & \hat{C}_m^{\mathrm{c}} \,\Box\, \bigl(p_m(\bar{Y}_m), \ldots, p_n(\bar{Y}_n)\bigr) :: B \\ \rightsquigarrow_P^* & \hat{C}^{\mathrm{s}} \,\Box\, B' :: B, \end{array} \tag{7.18}$$

with the condition that either $\hat{C}^{\mathrm{s}} = \mathbf{0}$ or $\hat{C}^{\mathrm{s}} \neq \mathbf{0}$ and $B' = \epsilon$. We use this prototypical partial derivation in order to define the following symbols:

- the k-th goal in (7.18) is G_k;
- the variant of clause used to rewrite G_k is denoted by R_k, with $k \geq 0$ (see Definition 114 on page def:finite-derivation);
- $\hat{C}^{\mathrm{c}}$ is the constraint store at the time of the considered call to $P[r]$;
- if $\hat{C}^{\mathrm{s}} \neq \mathbf{0}$ then $\hat{C}^{\mathrm{s}}$ is the constraint store on exit to the above call;
- for each $i = 1, \ldots, n$, if the call to p_i in the body of $P[r]$ takes place in (7.18), then the constraint store at the time of the call is $\hat{C}_i^{\mathrm{c}}$;
- for each $i = 1, \ldots, n$, if the call mentioned above succeeds, then the constraint store on exit is $\hat{C}_i^{\mathrm{s}}$; in that case we have also $\hat{C}_i^{\mathrm{s}} = \hat{C}_{i+1}^{\mathrm{c}}$ if $i < n$.

In order to ascertain that these hypothesis are clear, please make sure you understand that in (7.18) we have $\hat{C}_1^{\mathrm{c}} = \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho$.

Still on notation. Our convention is that, while descriptions arising from *top-down* (partial) derivation are characterized by a "hat accent", as in $\hat{C}_{\mathrm{sub}}^{\mathrm{sup}}$, the non-accented version $C_{\mathrm{sub}}^{\mathrm{sup}}$ denotes the same description *projected* and *renamed* onto $\vec{\Lambda}$. For instance, in what follows we will use the (possibly subscripted) symbols $\hat{C}^{\mathrm{s}}$ and C^{s} that are correlated by

$$C^{\mathrm{s}} = \bigl(\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{s}}\bigr)[\vec{\Lambda}/\bar{X}].$$

We will also use "reversed hats" to denote descriptions arising from the *bottom-up* computations, $\check{C}^{\mathrm{s}}$ for instance. With this notation, one of the objectives of the proof will be to show, for example, that

$$\bigl(\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{s}}\bigr)[\vec{\Lambda}/\bar{X}] = \bigl(\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{s}}\bigr) = C^{\mathrm{s}}.$$

We also assume, without loss of generality, that variables employed in top-down derivations are totally distinct from those used in the bottom-up construction.[8]

All the properties indicated by H_i, for $i = 1, \ldots, 7$ refer to Hypothesis 121 on page hypo:magic-id-axioms.

First Inclusion

We start by showing the inclusion

$$\sigma\Bigl(\mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P,G)\Bigr) \subseteq \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} \restriction \{1,\ldots,\#\,P\}$$

and the implication (7.17), for each $r \in \{1,\ldots,\#\,P\}$. We need to show that for each $G_0 \in G$:

(a) if $G_0 = \bigl(\hat{C}^{\mathrm{c}} \mathbin{\Box} p(\bar{X}) :: B\bigr)$, then there exists a clause r' in P_m^G, describing the first call in G_0, such that

$$(C^{\mathrm{c}} \mapsto C^{\mathrm{c}}) \in \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r'), \quad \text{where} \quad C^{\mathrm{c}} \stackrel{\mathrm{def}}{=} \bigl(\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}}\bigr)[\vec{\Lambda}/\bar{X}].$$

Moreover, suppose there exists a derivation

$$\begin{aligned} G_0 &\leadsto_P^m \hat{C}^{\mathrm{c}} \mathbin{\Box} p(\bar{X}) :: B \\ &\leadsto_{\{R_k\}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \mathbin{\Box} \bigl(p_1(\bar{Y}_1),\ldots,p_n(\bar{Y}_n)\bigr) :: B \\ &\leadsto_P^h \hat{C}_i \mathbin{\Box} \bigl(p_i(\bar{Y}_i),\ldots,p_n(\bar{Y}_n)\bigr) :: B, \end{aligned}$$

where $m,h \in \mathbb{N}$, $R_k = \bigl(p(\bar{Y}) \mathrel{:-} C\rho \mathbin{\Box} p_1(\bar{Y}_1),\ldots,p_n(\bar{Y}_n)\bigr)$ is a variant of $P[r]$ such that

$$FV(R_k) \cap \Bigl(FV(G_0) \cup \Bigl(\bigcup_{j=1}^{k-1} FV(R_j)\Bigr)\Bigr) = \varnothing. \tag{7.19}$$

[8]This assumption is not restrictive considering that we work on expressions deriving from two distinct mechanisms (top-down and bottom-up). A way to avoid collisions is, for instance, to use variables with an even index for the top-down derivations, reserving those with an odd index for the bottom-up construction.

Then there exists a magic clause r_i', obtained from $P[r]$ (i.e., describing the i-th call in the body of $P[r]$), such that

$$(C_i^{\mathrm{c}} \mapsto C_i^{\mathrm{c}}) \in \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r_i'),$$

where $C_i^{\mathrm{c}} \stackrel{\mathrm{def}}{=} \bigl(\bar{\exists}_{\bar{Y}_i}\, \hat{C}_i^{\mathrm{c}}\bigr)[\vec{\Lambda}_i/\bar{Y}_i]$.

(b) Suppose there exists a derivation

$$\begin{aligned} G_0 &\leadsto_P^m \hat{C}^{\mathrm{c}} \,\square\, p(\bar{X}) :: B \\ &\leadsto_{\{R_k\}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \,\square\, \bigl(p_1(\bar{X}_1), \dots, p_n(\bar{X}_n)\bigr) :: B \\ &\leadsto_P^l \hat{C}^{\mathrm{s}} \,\square\, B' :: B, \end{aligned}$$

where $m, l \in \mathbb{N}$ and $R_k = \bigl(p(\bar{Y}) \text{ :- } C\rho \,\square\, p_1(\bar{Y}_1), \dots, p_n(\bar{Y}_n)\bigr)$ is a variant of $P[r]$ renamed apart as in (7.19). Then, if $\hat{C}^{\mathrm{s}} \neq \mathbf{0}$ and $B' = \epsilon$, we have

$$(C^{\mathrm{c}} \mapsto C^{\mathrm{s}}) \in \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r),$$

where

$$C^{\mathrm{c}} \stackrel{\mathrm{def}}{=} \bigl(\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}}\bigr)[\vec{\Lambda}/\bar{X}]\bigr)$$

and

$$C^{\mathrm{s}} \stackrel{\mathrm{def}}{=} \bigl(\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{s}}\bigr)[\vec{\Lambda}/\bar{X}]\bigr).$$

Otherwise, if $\hat{C}^{\mathrm{s}} = \mathbf{0}$ then

$$\exists r' \in \bigl\{\, \# P + 1, \dots, \# P_m^G \bigr\} \,.\, (C^{\mathrm{c}} \mapsto C^{\mathrm{c}}) \in \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r').$$

The proof is by total induction on the length N of the considered derivations.

Base case (a). Let $N = 0$. The derivation coincides with the initial *query* $G_0 = \bigl(\hat{C}^{\mathrm{c}} \,\square\, p(\bar{X}) :: B\bigr)$. By definition of magic program, the first call in G_0 corresponds to the normalized clause $P_m^G[r'] = \bigl(\mathtt{m\_}p(\vec{\Lambda}) \text{ :- } \hat{C}^{\mathrm{c}}[\vec{\Lambda}/\bar{X}]\bigr)$. So, by $\eta_{r'}$ definition, $(C^{\mathrm{c}} \mapsto C^{\mathrm{c}})$ belongs to $\bigl(T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} \uparrow 1\bigr)(r')$ and thus to $\mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r')$.

Base case (b). Let $N = 1$ (a success requires at least one derivation step). Consider the partial derivation

$$G_0 = \bigl(\hat{C}^{\mathrm{c}} \,\Box\, p(\bar{X}) :: B\bigr) \rightsquigarrow_{\{R_1\}} \overbrace{\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho}^{\hat{C}^{\mathrm{s}}} \,\Box\, B \tag{7.20}$$

where $R_1 = \bigl(p(\bar{Y}) :\!-\; C\rho \,\Box\, \varnothing\bigr)$ is a variant of $P[r]$ renamed apart from G_0. The renaming ρ is such that $\bar{U}\rho = \bar{Y}$. By the base case (a), there is a clause r' in P_m^G, corresponding to the first call in G_0, such that

$$(C^{\mathrm{c}} \mapsto C^{\mathrm{c}}) \in \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r'). \tag{7.21}$$

Let $P_m^G[r] = \bigl(p(\vec{\Lambda}) :\!-\; \bigl\langle\{\mathtt{m\_}p(\vec{\Lambda})\}, C\ddot{\rho}, \varnothing\bigr\rangle\bigr)$, where $\bar{U}\ddot{\rho} = \vec{\Lambda}$. Let $\hat{C}^{\mathrm{s}} = \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho$ the partial answer constraint in (7.20).

If $\hat{C}^{\mathrm{s}} = \mathbf{0}$ then $(C^{\mathrm{c}} \mapsto \mathbf{0}) \in \mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P, G)(r)$, and, by (7.21), the thesis (7.17) holds.

If $\hat{C}^{\mathrm{s}} \neq \mathbf{0}$ we reason as follows. Consider the function $T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}$ as given in Definition 116 on page def:TP. In the computation of $T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r, \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}})$ we have

$$S' = \varpi_r\bigl(\mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r'), \vec{\Lambda}, \vec{\Lambda}, \bar{Z}\bigr), \tag{7.22}$$

where $\bar{Z} \ll_{\vec{\Lambda}} FV(P_m^G[r]) \cup \Lambda$, and thus

$$S = S' \otimes_{\mathrm{F}} C\ddot{\rho}.$$

By manipulating (7.22), taking into account (7.21), and considering that $P_m^G[r]$ is a modified clause, we obtain $(C^{\mathrm{c}} \mapsto C^{\mathrm{c}}) \in S'$.

Observe that $T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(\mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}) = \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}$, since $\mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}$ is a fixpoint; therefore $\bar{\exists}_{\vec{\Lambda}}\, \eta_r(S) = \bar{\exists}_{\vec{\Lambda}}\, S \subseteq \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r)$. By $\otimes_{\mathrm{F}}$ distributivity with respect to $\cup$ we have that

$$\bigl(C^{\mathrm{c}} \mapsto \bar{\exists}_{\vec{\Lambda}}(\underbrace{C^{\mathrm{c}} \otimes C\ddot{\rho}})\bigr) \in \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r) \tag{7.23}$$

provided that, by $\otimes_{\mathrm{F}}$ definition,

$$C^{\mathrm{c}} \otimes C\ddot{\rho} \neq \mathbf{0}. \tag{7.24}$$

In other words, for (7.23) to hold, we must ensure that the indicated pair is not "discarded" by the $\otimes_{\mathrm{F}}$ operator, i.e., (7.24) holds. We will now show that

$$\bar{\exists}_{\vec{\Lambda}}(C^{\mathrm{c}} \otimes C\ddot{\rho}) = \bigl(\bar{\exists}_{\bar{X}}(\underbrace{\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho}_{\hat{C}^{\mathrm{s}}})\bigr)[\vec{\Lambda}/\bar{X}]. \tag{7.25}$$

This way, since $\hat{C}^{\mathrm{s}} \neq \mathbf{0}$, we will also prove that (7.24) and (7.23) hold. The validity of (7.25) is proved as follows:

$$
\begin{aligned}
&\big(\bar{\exists}_{\bar{X}}(\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho)\big)[\vec{\Lambda}/\bar{X}] \\
&\quad = \big(\bar{\exists}_{\bar{X}}(\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho)\big)[\vec{\Lambda}/\bar{X}] \\
&\quad \text{[by Lemma 126]} \\
&\quad = \bar{\exists}_{\vec{\Lambda}}\Big((\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho)[\vec{\Lambda}/\bar{X}]\Big) \\
&\quad \text{[by } H_4\text{]} \\
&\quad = \bar{\exists}_{\vec{\Lambda}}\big((\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}})[\vec{\Lambda}/\bar{X}] \otimes \mathrm{d}_{\vec{\Lambda}\bar{Y}} \otimes C\rho[\vec{\Lambda}/\bar{X}]\big) \\
&\quad \text{[by } H_3\text{]} \\
&\quad = \bar{\exists}_{\vec{\Lambda}}\Big(\bar{\exists}_{\vec{\Lambda}}\big(\hat{C}^{\mathrm{c}}[\vec{\Lambda}/\bar{X}]\big) \otimes \mathrm{d}_{\vec{\Lambda}\bar{Y}} \otimes C\rho\Big) \\
&\quad \text{[by } H_4 \text{ and since } FV(C\rho) \cap \bar{X} \subseteq FV(R_1) \cap FV(G_0) = \varnothing\text{]} \\
&\quad = \bar{\exists}_{\vec{\Lambda}}\big(\hat{C}^{\mathrm{c}}[\vec{\Lambda}/\bar{X}]\big) \otimes \bar{\exists}_{\vec{\Lambda}}\big(\mathrm{d}_{\vec{\Lambda}\bar{Y}} \otimes C\rho\big) \\
&\quad \text{[by } S_4\text{]} \\
&\quad = \bar{\exists}_{\vec{\Lambda}}\big(\hat{C}^{\mathrm{c}}[\vec{\Lambda}/\bar{X}]\big) \otimes \bar{\exists}_{\vec{\Lambda}}\big(C\rho[\vec{\Lambda}/\bar{Y}]\big) \\
&\quad \text{[by } S_5\text{, since } \bar{Y}(\rho[\vec{\Lambda}/\bar{Y}]) = \vec{\Lambda}\text{]} \\
&\quad = \bar{\exists}_{\vec{\Lambda}}\Big(\bar{\exists}_{\vec{\Lambda}}\big(\hat{C}^{\mathrm{c}}[\vec{\Lambda}/\bar{X}]\big) \otimes C\ddot{\rho}\Big) \\
&\quad \text{[by } S_4 \text{ and since } \ddot{\rho} = \rho[\vec{\Lambda}/\bar{Y}]\text{]} \\
&\quad = \bar{\exists}_{\vec{\Lambda}}(C^{\mathrm{c}} \otimes C\ddot{\rho}) \\
&\quad \text{[by def. of } C^{\mathrm{c}}\text{]}
\end{aligned}
$$

Thus the thesis, i.e., $(C^{\mathrm{c}} \mapsto C^{\mathrm{s}}) \in \mathcal{F}_{R_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r)$, follows by (7.25).

Inductive case (b). Assume the thesis is valid for derivations whose length is less than N. We prove that it holds also for derivations of length N (case (b)). Consider a partial derivation

$$
\begin{aligned}
G_0 &\leadsto_P^m \hat{C}^{\mathrm{c}} \,\square\, p(\bar{X}) :: B \\
&\leadsto_{\{R_k\}} \underbrace{\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \,\square\, \big(p_1(\bar{Y}_1), \ldots, p_n(\bar{Y}_n)\big) :: B}_{G'} \\
&\leadsto_P^l \hat{C}^{\mathrm{s}} \,\square\, B' :: B,
\end{aligned}
\tag{7.26}
$$

where $m + l + 1 = N$ and

$$R_k = \Big(p(\bar{Y}) \mathrel{:-} C\rho \,\square\, \big(p_1(\bar{Y}_1), \ldots, p_n(\bar{Y}_n)\big)\Big)$$

is a variant of $P[r]$ such that

$$FV(R_k) \cap \left(FV(G_0) \cup \bigcup_{j=1}^{k-1} FV(R_j)\right) = \varnothing$$

and $\bar{U}\rho = \bar{Y}$, $\bar{U}_1\rho = \bar{Y}_1$, ... , $\bar{U}_n\rho = \bar{Y}_n$. This derivation is responsible for the fact that

$$(C^{\mathrm{c}} \mapsto C^{\mathrm{s}}) \in \mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P, G)(r). \tag{7.27}$$

If $\hat{C}^{\mathrm{s}} = \mathbf{0}$ then $(C^{\mathrm{c}} \mapsto \mathbf{0}) \in \mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P, G)(r)$. This holds because there is a derivation whose length is either

- zero, in which case we must have $G_0 = \bigl(\hat{C}^{\mathrm{c}} \mathbin{\square} p(\bar{X}) :: B\bigr)$. By the inductive hypothesis there exists r' such that $P^G_m[r'] = \bigl(\mathtt{m\_}p(\vec{\Lambda}) \mathrel{:-} \hat{C}^{\mathrm{c}}[\vec{\Lambda}/\bar{X}]\bigr)$ and

  $$(C^{\mathrm{c}} \mapsto C^{\mathrm{c}}) \in \mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P^G_m}(r'). \tag{7.28}$$

  Or the length of the derivation is

- greater than zero. This derivation is of the form

  $$\begin{aligned} G_0 &\rightsquigarrow^{m_1}_{P} C' \mathbin{\square} q(\bar{W}) :: B_0 \\ &\rightsquigarrow_{\{R_p\}} \tilde{C} \mathbin{\square} B_1 :: p(\bar{X}) :: B_2 :: B_0 \\ &\rightsquigarrow^{m_2}_{P} \hat{C}^{\mathrm{c}} \mathbin{\square} p(\bar{X}) :: B_2 :: B_0 \end{aligned}$$

  where $m_1 + m_2 + 1 = m < N$, R_p is a variant of some program clause $P[r'']$. $P^G_m[r']$ is the magic clause describing the j-th call to p in the body of r'', where $j = \# B_1 + 1$. By the inductive hypothesis (7.28) holds in this case too.

Suppose, instead, that $\hat{C}^{\mathrm{s}} \neq \mathbf{0}$. By Definition 115 we have that, in (7.26), $B' = \epsilon$ and $\hat{C}^{\mathrm{s}}$ is the constraint store at the successful exit from the indicated call to $P[r]$. We will now rewrite $\hat{C}^{\mathrm{s}}$, in order to show that the pair in (7.27) belongs also to $\mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P^G_m}(r)$. A consequence of (7.26) is that, for each $i = 1, \ldots, n$, there must exist derivations

$$\begin{aligned} \mathbf{1} \mathbin{\square} p_i(\bar{Y}_i) &\rightsquigarrow_{\{R_{k+h_i}\}} \mathbf{1} \otimes \mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes C_i\rho_i \mathbin{\square} B_{r_i} \\ &\rightsquigarrow^{k_i}_{P} \mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes \grave{C}^{\mathrm{s}}_i \mathbin{\square} \varnothing \end{aligned} \tag{7.29}$$

such that[9] $k_i < l$, $h_i > 0$ and $R_{k+h_i} = \bigl(p_i(\bar{X}_i) \mathrel{:-} C_i\rho_i \mathbin{\square} B_{r_i}\bigr)$ is a variant of $P[r_i]$ with

$$FV(R_{k+h_i}) \cap \left(FV(G_0) \cup \bigcup_{j=1}^{k+h_i-1} FV(R_j) \right) = \varnothing. \tag{7.30}$$

Observe that the variants used in the derivation (7.26) up to the call of p_i in G', for each $i = 1, \ldots, n$, are $R_1, \ldots, R_{k+h_i-1}$. Obviously, $h_i < h_{i+1}$.

[9]Notice that $\sum_{i=1}^{n}(k_i + 1) = l$.

Observe also that in (7.29) we have used the same variants employed in (7.26). This implies that $FV(\grave{C}^{\mathrm{s}}_i) \cap \bar{Y}_i = \varnothing$. Thus we can write

$$\hat{C}^{\mathrm{s}} = \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{i=1}^{n} (\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes \grave{C}^{\mathrm{s}}_i). \tag{7.31}$$

We have thus obtained our final formulation for the partial answer constraint of the partial derivation (7.20):

$$\begin{aligned}
\bigl(\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{s}}\bigr)[\vec{\Lambda}/\bar{X}] &= \bar{\exists}_{\bar{X}}\Bigl(\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{i=1}^{n} \bigl(\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes \grave{C}^{\mathrm{s}}_i\bigr)\Bigr)[\vec{\Lambda}/\bar{X}] \\
&= \bar{\exists}_{\bar{X}}\Bigl(\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \\
&\qquad \otimes \bigotimes_{i=1}^{n} \bigl(\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes \bar{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\bigr)\Bigr)[\vec{\Lambda}/\bar{X}] \\
&\quad \text{[by Lemma 126]}
\end{aligned} \tag{7.32}$$

We will now show that the bottom-up construction derives, for $P[r]$, a pair that, taking into account (7.32), is identical to the pair in (7.27).

Recall that the partial derivation (7.26) employs the following variant of $P[r]$:

$$R_k = \Bigl(p(\bar{Y}) \mathrel{:\!-} C\rho \,\Box\, \bigl(p_1(\bar{Y}_1), \ldots, p_n(\bar{Y}_n)\bigr)\Bigr).$$

By definition of magic program, there exists a renaming $\ddot{\rho}$ with $\bar{U}\ddot{\rho} = \vec{\Lambda}$, $\bar{U}_1\ddot{\rho} = \bar{W}_1$, ..., $\bar{U}_n\ddot{\rho} = \bar{W}_n$, such that P^G_m contains the following clauses derived from $P[r]$:

$$P^G_m[r] = \Bigl(p(\vec{\Lambda}) \mathrel{:\!-} \Bigl\langle \{\mathtt{m\_}p(\vec{\Lambda})\}, C\ddot{\rho}, \bigl(p_1(\bar{W}_1), \ldots, p_n(\bar{W}_n)\bigr)\Bigr\rangle\Bigr), \tag{7.33}$$

and, for each $i = 1$, ..., n,

$$P^G_m[r'_i] = \Bigl(\mathtt{m\_}p_i(\vec{\Lambda}_i) \mathrel{:\!-} \Bigl\langle \{\mathtt{m\_}p(\bar{V}^i)\}, C\ddot{\rho}, \bigl(p_1(\bar{V}^i_1), \ldots, p_{i-1}(\bar{V}^i_{i-1})\bigr)\Bigr\rangle\Bigr). \tag{7.34}$$

By the inductive hypothesis we have that:

- $\bigl(C^{\mathrm{c}} \mapsto C^{\mathrm{c}}\bigr) \in \mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r')$ (as stated in (7.28));
- for each $i = 1$, ..., n, by (7.26) we have that

$$\begin{aligned}
G_0 &\rightsquigarrow^{m}_{P} \hat{C}^{\mathrm{c}} \,\Box\, p(\bar{X}) :: B \\
&\rightsquigarrow_{\{R_k\}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \,\Box\, \bigl(p_1(\bar{Y}_1), \ldots, p_n(\bar{Y}_n)\bigr) :: B \\
&\rightsquigarrow^{h}_{P} \hat{C}^{\mathrm{c}}_i \,\Box\, \bigl(p_i(\bar{Y}_i), \ldots, p_n(\bar{Y}_n)\bigr) :: B
\end{aligned}$$

is a derivation whose length is less than N, where

$$\hat{C}^{\mathrm{c}}_i = \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{j=1}^{i-1} (\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes \grave{C}^{\mathrm{s}}_j),$$

each $\grave{C}^{\mathrm{s}}_j$ being obtained from the derivations in (7.29). Thus, there exists a clause in P^G_m (precisely, clause r'_i as given in (7.34)) such that $\bigl(C^{\mathrm{c}}_i \mapsto C^{\mathrm{c}}_i\bigr) \in \mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r'_i)$.

- For each $i = 1, \ldots, n$, by (7.26) we have that

$$\begin{aligned} G_0 &\rightsquigarrow^m_P \hat{C}^{\mathrm{c}} \,\square\, p(\bar{X}) :: B \\ &\rightsquigarrow^{h+1}_P \hat{C}^{\mathrm{c}}_i \,\square\, \bigl(p_i(\bar{Y}_i), \ldots, p_n(\bar{Y}_n)\bigr) :: B \\ &\rightsquigarrow_{\{R_{k+h_i}\}} \hat{C}^{\mathrm{c}}_i \otimes \mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes C_i\rho_i \\ &\qquad\qquad \square\, B_i :: \bigl(p_{i+1}(\bar{Y}_{i+1}), \ldots, p_n(\bar{Y}_n)\bigr) :: B \\ &\rightsquigarrow^{k_i}_P \hat{C}^{\mathrm{s}}_i \,\square\, \bigl(p_{i+1}(\bar{Y}_{i+1}), \ldots, p_n(\bar{Y}_n)\bigr) :: B \end{aligned}$$

  is a partial derivation of length less than N, where R_{k+h_i} is a variant of $P[r_i]$ (satisfying the conditions described in (7.30)), $\hat{C}^{\mathrm{s}}_i \neq \mathbf{0}$ (since $\hat{C}^{\mathrm{s}} \neq \mathbf{0}$) and

$$\hat{C}^{\mathrm{s}}_i = \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{j=1}^{i} (\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes \grave{C}^{\mathrm{s}}_j). \tag{7.35}$$

  Thus, $(C^{\mathrm{c}}_i \mapsto C^{\mathrm{s}}_i) \in \mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r_i)$.

By Lemma 126 we have that, for each $i = 1, \ldots, n$:

$$\bar{\exists}_{\bar{X}_i} \hat{C}^{\mathrm{s}}_i = \bar{\exists}_{\bar{X}_i}\Bigl(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{j=1}^{i} \bigl(\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes \bar{\exists}_{\bar{X}_j} \grave{C}^{\mathrm{s}}_j\bigr)\Bigr), \tag{7.36}$$

and thus

$$\begin{aligned} &\bigl(\bar{\exists}_{\bar{X}_i} \hat{C}^{\mathrm{s}}_i\bigr)[\vec{\Lambda}_i/\bar{X}_i] = \\ &\qquad \Bigl(\bar{\exists}_{\bar{X}_i}\bigl(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{j=1}^{i} (\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes \bar{\exists}_{\bar{X}_j} \grave{C}^{\mathrm{s}}_j)\bigr)\Bigr)[\vec{\Lambda}_i/\bar{X}_i]. \end{aligned} \tag{7.37}$$

With reference to Definition 116, the evaluation of $T^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r, \mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G})$ gives:

$$S' = \varpi_r\bigl(\mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r'), \vec{\Lambda}, \vec{\Lambda}, \bar{Z}\bigr), \tag{7.38}$$

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

$$S_i = (\mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r_i))[\bar{Z}_i/\vec{\Lambda}_i] \tag{7.39}$$

where

$$\bar{Z} \ll_{\vec{\Lambda}} FV(P_m^G[r]) \cup \Lambda, \tag{7.40}$$

and

$$\bar{Z}_i \ll_{\vec{\Lambda}_i} FV(P_m^G[r]) \cup \Lambda \cup \bar{Z} \cup \bar{Z}_1 \cup \cdots \cup \bar{Z}_{i-1}. \tag{7.41}$$

Thus

$$S = S' \otimes_{\mathrm{F}} C\ddot{\rho} \otimes_{\mathrm{F}} \bigotimes_{i=1}^{n}{}_{\mathrm{F}} (\mathrm{d}_{\bar{W}_i \bar{Z}_i} \otimes_{\mathrm{F}} S_i) \tag{7.42}$$

Again, by manipulating (7.38), taking into account (7.28) and considering that $P_m^G[r]$ is a modified clause, we obtain $(C^{\mathrm{c}} \mapsto C^{\mathrm{c}}) \in S'$; moreover, for each $i = 1, \ldots, n$, $(C_i^{\mathrm{c}} \mapsto C_i^{\mathrm{s}}) \in S_i$.

We also have $\bar{\exists}_{\vec{\Lambda}}\, \eta_r(S) = \bar{\exists}_{\vec{\Lambda}}\, S \subseteq \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r)$, since $T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(\mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}) = \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}$. From $\otimes_{\mathrm{F}}$ distributivity with respect to $\cup$ and from (7.37) we obtain

$$\bigl(C^{\mathrm{c}} \mapsto \bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{s}}\bigr) \in \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r), \tag{7.43}$$

where the success component $\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{s}}$ can be written as follows:

$$\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{s}} \overset{\mathrm{def}}{=} \bar{\exists}_{\vec{\Lambda}} \biggl(C^{\mathrm{c}} \otimes C\ddot{\rho} \otimes \bigotimes_{i=1}^{n} \Bigl(\mathrm{d}_{\bar{W}_i \bar{Z}_i} \otimes \bigl(\bar{\exists}_{\vec{\Lambda}_i}\, \check{C}_i^{\mathrm{s}}\bigr)[\bar{Z}_i/\vec{\Lambda}_i]\Bigr)\biggr), \tag{7.44}$$

provided that, by $\otimes_{\mathrm{F}}$ definition, $\check{C}^{\mathrm{s}} \neq \mathbf{0}$ and, considering that, for each $i = 1, \ldots, n$, it is

$$FV\Bigl(\bigl(\bar{\exists}_{\vec{\Lambda}}\, \check{C}_i^{\mathrm{c}}\bigr)[\bar{Z}_i/\vec{\Lambda}_i]\Bigr) \subseteq \bar{Z}_i,$$

we have

$$\bar{\exists}_{\bar{Z}_i} \overbrace{\biggl(C^{\mathrm{c}} \otimes C\ddot{\rho} \otimes \bigotimes_{j=1}^{i-1} \Bigl(\mathrm{d}_{\bar{W}_j \bar{Z}_j} \otimes \bigl(\bar{\exists}_{\vec{\Lambda}_j}\, \check{C}_j^{\mathrm{s}}\bigr)[\bar{Z}_j/\vec{\Lambda}_j]\Bigr) \otimes \mathrm{d}_{\bar{W}_i \bar{Z}_i}\biggr)}^{C^i} = \bigl(\bar{\exists}_{\vec{\Lambda}}\, \check{C}_i^{\mathrm{c}}\bigr)[\bar{Z}_i/\vec{\Lambda}_i]. \tag{7.45}$$

Now we prove that (7.45) holds. First we note that

$$\big(\overline{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{c}}_i\big)[\bar{Z}_i/\vec{\Lambda}_i] = \big(\overline{\exists}_{\bar{Y}_i}\, \hat{C}^{\mathrm{c}}_i\big)[\bar{Z}_i/\bar{Y}_i] \tag{7.46}$$

[by the inductive hypothesis and H_5]

$$= \Big(\overline{\exists}_{\bar{Y}_i}\Big(\overline{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{j=1}^{i-1}\Big(\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes \overline{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_i\Big)\Big)\Big)[\bar{Z}_i/\bar{Y}_i]$$

[derives from (7.36)]

$$= \overline{\exists}_{\bar{Z}_i}\Big(\Big(\overline{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{j=1}^{i-1}\Big(\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes \overline{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_i\Big)\Big)[\bar{Z}_i/\bar{Y}_i]\Big) \tag{7.47}$$

[by H_4]

Moreover,

$$\big(\overline{\exists}_{\vec{\Lambda}_i}\, \check{C}^{\mathrm{s}}_i\big)[\bar{Z}_i/\vec{\Lambda}_i] = \big(\overline{\exists}_{\bar{X}_i}\, \hat{C}^{\mathrm{s}}_i\big)[\vec{\Lambda}_i/\bar{X}_i][\bar{Z}_i/\vec{\Lambda}_i]$$

$$= \Big(\overline{\exists}_{\bar{X}_i}\big(\hat{C}^{\mathrm{c}}_i \otimes \mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes \overline{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\big)\Big)[\bar{Z}_i/\bar{X}_i]$$

[by (7.29) and (7.36)]

$$= \Big(\overline{\exists}_{\bar{X}_i}\big(\hat{C}^{\mathrm{c}}_i \otimes \mathrm{d}_{\bar{Y}_i\bar{X}_i}\big) \otimes \overline{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\Big)[\bar{Z}_i/\bar{X}_i]$$

[by axiom S_4 of Definition 110]

$$= \Big(\overline{\exists}_{\bar{X}_i}\big(\hat{C}^{\mathrm{c}}_i[\bar{X}_i/\bar{Y}_i]\big) \otimes \overline{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\Big)[\bar{Z}_i/\bar{X}_i]$$

[by S_5, since $\bar{X}_i \cap \bar{Y}_i = \varnothing$]

$$= \Big(\overline{\exists}_{\bar{X}_i}\big(\hat{C}^{\mathrm{c}}_i[\bar{X}_i/\bar{Y}_i]\big)\Big)[\bar{Z}_i/\bar{X}_i] \otimes \Big(\overline{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\Big)[\bar{Z}_i/\bar{X}_i]$$

[by H_3]

$$= \big(\overline{\exists}_{\bar{Y}_i}\, \hat{C}^{\mathrm{c}}_i\big)[\bar{X}_i/\bar{Y}_i][\bar{Z}_i/\bar{X}_i] \otimes \big(\overline{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\big)[\bar{Z}_i/\bar{X}_i]$$

[by H_4, since $FV(\hat{C}^{\mathrm{c}}_i) \cap \bar{X}_i = \varnothing$]

$$= \big(\overline{\exists}_{\bar{Y}_i}\, \hat{C}^{\mathrm{c}}_i\big)[\bar{Z}_i/\bar{Y}_i] \otimes \big(\overline{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\big)[\bar{Z}_i/\bar{X}_i] \tag{7.48}$$

[by H_5, since $\bar{Z}_i \cap \bar{Y}_i = \varnothing$ and $FV(\hat{C}^{\mathrm{c}}_i) \cap \bar{Z}_i = \varnothing$]

Substituting (7.48) for $\big(\overline{\exists}_{\vec{\Lambda}_i}\, \check{C}^{\mathrm{s}}_i\big)[\bar{Z}_i/\vec{\Lambda}_i]$ in the over-braced expression in

(7.45), we obtain, for $i = 1, \ldots, n$:

$$
\begin{aligned}
&\bar{\exists}_{\bar{Z}_i} C^i \\
&= \bar{\exists}_{\bar{Z}_i}\Bigg(C^{\mathrm{c}} \otimes C\ddot{\rho} \\
&\quad \otimes \bigotimes_{j=1}^{i-1}\Big(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \big(\bar{\exists}_{\bar{Y}_j}\, \hat{C}^{\mathrm{c}}_j\big)[\bar{Z}_j/\bar{Y}_j] \otimes \big(\bar{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_j\big)[\bar{Z}_j/\bar{X}_j]\Big) \otimes \mathrm{d}_{\bar{W}_i\bar{Z}_i}\Bigg) \\
&= \bar{\exists}_{\bar{Z}_i}\Bigg(\Big(C^{\mathrm{c}} \otimes C\ddot{\rho} \\
&\quad \otimes \bigotimes_{j=1}^{i-1}\Big(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \big(\bar{\exists}_{\bar{Z}_j}\, \hat{C}^{\mathrm{c}}_j\big)[\bar{Z}_j/\bar{Y}_j] \otimes \big(\bar{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_j\big)[\bar{Z}_j/\bar{X}_j]\Big)\Big)[\bar{Z}_i/\bar{W}_i]\Bigg) \\
&\text{[by } S_5 \text{ with } [\bar{Z}_i/\bar{W}_i] = \rho] \\
&= \bar{\exists}_{\bar{Z}_i}\Bigg(\Big((\bar{\exists}_{\bar{X}}\, C^{\mathrm{c}})[\vec{\Lambda}/\bar{X}] \otimes C\ddot{\rho} \\
&\quad \otimes \bigotimes_{j=1}^{i-1}\Big(\mathrm{d}_{\bar{W}_j\bar{Z}_i} \otimes \big(\bar{\exists}_{\bar{Y}_j}\, \hat{C}^{\mathrm{c}}_j\big)[\bar{Z}_j/\bar{Y}_j] \otimes \big(\bar{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_j\big)[\bar{Z}_j/\bar{X}_j]\Big)\Big)[\bar{Z}_i/\bar{W}_i]\Bigg) \\
&\text{[by definition of } C^{\mathrm{c}}] \\
&= \bar{\exists}_{\bar{Z}_i}\Bigg(\Big(\bar{\exists}_{\bar{X}}\, C^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\vec{\Lambda}} \\
&\quad \otimes \overbrace{C\ddot{\rho} \otimes \bigotimes_{j=1}^{i-1}\Big(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \big(\bar{\exists}_{\bar{Y}_j}\, \hat{C}^{\mathrm{c}}_j\big)[\bar{Z}_j/\bar{Y}_j] \otimes \big(\bar{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_j\big)[\bar{Z}_j/\bar{X}_j]\Big)}^{F} \\
&\qquad\qquad \Big)[\bar{Z}_i/\bar{W}_i]\Bigg) \\
&\text{[by Lemma 124 since } FV(F) \cap \bar{X} = FV(\hat{C}^{\mathrm{c}}) \cap \vec{\Lambda} = \bar{X} \cap \vec{\Lambda} = \bar{Z}_i \cap \bar{X}_i = \varnothing\ ]
\end{aligned}
\tag{7.49}
$$

Now, we consider the renaming

$$\rho_{i-1} \stackrel{\mathrm{def}}{=} [\bar{Y}/\vec{\Lambda}]\,;[\bar{Y}_1/\bar{W}_1]\,;\cdots;[\bar{Y}_n/\bar{W}_n]\,;[\bar{X}_1/\bar{Z}_1]\,;\cdots;[\bar{X}_{i-1}/\bar{Z}_{i-1}].$$

By (7.49) we can write

$$
\begin{aligned}
\bar{\exists}_{\bar{Z}_i} C^i = \bar{\exists}_{\bar{Z}_i} \Bigg(\bigg(\Big(\bar{\exists}_{\bar{X}} C^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\vec{\Lambda}} \otimes C\ddot{\rho} \\
\otimes \bigotimes_{j=1}^{i-1} \Big(\mathrm{d}_{\bar{W}_j \bar{Z}_j} \otimes \big(\bar{\exists}_{\bar{Y}_j} \hat{C}^{\mathrm{c}}_j \big)[\bar{Z}_j/\bar{Y}_j] \\
\otimes \big(\bar{\exists}_{\bar{X}_j} \grave{C}^{\mathrm{s}}_j \big)[\bar{Z}_j/\bar{X}_j] \Big) \Big) [\bar{Z}_i/\bar{W}_i] \bigg) \rho_{i-1} \Bigg)
\end{aligned}
$$

$\left[\text{by } H_6\text{, since } FV(C^i) \cap \mathrm{cod}(\rho_{i-1}) = \bar{Z}_i \cap \mathrm{cod}(\rho_{i-1}) = \bar{Z}_i \cap \mathrm{dom}(\rho_{i-1}) = \varnothing\right]$

$$
\begin{aligned}
= \bar{\exists}_{\bar{Z}_i} \bigg(\Big(\bar{\exists}_{\bar{X}} C^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\ddot{\rho}\rho_{i-1} \\
\otimes \bigotimes_{j=1}^{i-1} \Big(\mathrm{d}_{\bar{Y}_j \bar{X}_j} \otimes \big(\bar{\exists}_{\bar{Y}_j} \hat{C}^{\mathrm{c}}_j \big)[\bar{X}_j/\bar{Y}_j] \otimes \bar{\exists}_{\bar{X}_j} \grave{C}^{\mathrm{s}}_j \Big) \Big) [\bar{Z}_i/\bar{Y}_i] \bigg)
\end{aligned}
\tag{7.50}
$$

[by H_3 and then applying ρ_{i-1}]

$$
\begin{aligned}
= \bar{\exists}_{\bar{Z}_i} \bigg(\Big(\bar{\exists}_{\bar{X}} C^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \\
\otimes \bigotimes_{j=1}^{i-1} \Big(\mathrm{d}_{\bar{Y}_j \bar{X}_j} \otimes \big(\bar{\exists}_{\bar{Y}_j} \hat{C}^{\mathrm{c}}_j \big)[\bar{X}_j/\bar{Y}_j] \otimes \bar{\exists}_{\bar{X}_j} \grave{C}^{\mathrm{s}}_j \Big) \Big) [\bar{Z}_i/\bar{Y}_i] \bigg)
\end{aligned}
$$

$\left[\text{since } \ddot{\rho}\rho_{i-1} = \rho[\bar{X}_1/\bar{Z}_1] \cdots [\bar{X}_{i-1}/\bar{Z}_{i-1}] \text{ and } FV(C) \cap \bar{Z}_k = \varnothing\text{, for each } k = 1, \dots, i-1\right]$

We will now show that

$$
\begin{aligned}
\bar{\exists}_{\bar{Z}_i} C^i = \bar{\exists}_{\bar{Z}_i} \bigg(\Big(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \\
\otimes \bigotimes_{j=1}^{i-1} \Big(\mathrm{d}_{\bar{Y}_j \bar{X}_j} \otimes \bar{\exists}_{\bar{X}_j} \grave{C}^{\mathrm{s}}_j \Big) \Big) [\bar{Z}_i/\bar{Y}_i] \bigg),
\end{aligned}
\tag{7.51}
$$

where the right-hand side of (7.51) is the same as (7.47). This is done by considering the generic expression

$$
\begin{aligned}
E_k \stackrel{\mathrm{def}}{=} \bar{\exists}_{\bar{Z}_i} \bigg(\Big(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{j=1}^{k-1} \Big(\mathrm{d}_{\bar{Y}_j \bar{X}_j} \otimes \bar{\exists}_{\bar{X}_j} \grave{C}^{\mathrm{s}}_j \Big) \\
\otimes \bigotimes_{j=k}^{i-1} \Big(\mathrm{d}_{\bar{W}_j \bar{Z}_j} \otimes \big(\bar{\exists}_{\bar{Y}_j} \hat{C}^{\mathrm{c}}_j \big)[\bar{Z}_j/\bar{Y}_j] \otimes \bar{\exists}_{\bar{X}_j} \grave{C}^{\mathrm{s}}_j \Big) \Big) [\bar{Z}_i/\bar{Y}_i] \bigg).
\end{aligned}
\tag{7.52}
$$

The right-hand sides of (7.50) and (7.51) are given by E_1 and E_i, respectively. It is possible to prove that $E_1 = E_i$ by showing that $E_1 = E_2$, $E_2 = E_3$, $\dots$, $E_{i-1} = E_i$. The equality $E_k = E_{k+1}$ can be obtained using

Lemma 125. To this purpose, we consider the following expression for E_k

$$\begin{aligned}E_k \overset{\text{def}}{=} \bar{\exists}_{\bar{Z}_k} \Biggl(\Biggl(& \overbrace{\Bigl(\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{j=1}^{k-1} \Bigl(\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes \bar{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_j\Bigr)}^{D} \\ & \otimes \mathrm{d}_{\bar{Y}_k\bar{X}_k} \otimes \bigl(\bar{\exists}_{\bar{Y}_k}\, \hat{C}^{\mathrm{c}}_k\bigr)[\bar{Z}_k/\bar{Y}_k] \otimes \bigl(\bar{\exists}_{\bar{X}_k}\, \grave{C}^{\mathrm{s}}_k\bigr)[\bar{Z}_k/\bar{X}_k] \\ \otimes \bigotimes_{j=k+1}^{i-1} & \Bigl(\mathrm{d}_{\bar{Y}_j\bar{X}_j} \otimes \bigl(\bar{\exists}_{\bar{Y}_j}\, \hat{C}^{\mathrm{c}}_j\bigr)[\bar{Z}_j/\bar{Y}_j] \otimes \bigl(\bar{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_j\bigr)[\bar{Z}_j/\bar{X}_j]\Bigr)\Biggr)[\bar{Z}_i/\bar{Y}_i]\Biggr), \end{aligned} \tag{7.53}$$

By (7.46) and (7.47) we have that, for each $k = 1, \ldots, i$,

$$\bigl(\bar{\exists}_{\bar{Y}_k}\, \hat{C}^{\mathrm{c}}_i\bigr)[\bar{Z}_k/\bar{Y}_k] = \bigl(\bar{\exists}_{\bar{Y}_k}\, D\bigr)[\bar{Z}_k/\bar{Y}_k],$$

thus Lemma 125 applies giving the desired result.

We will now show that

$$\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{s}} = \bar{\exists}_{\vec{\Lambda}}\Bigl(C^{\mathrm{c}} \otimes C\ddot{\rho} \otimes \bigotimes_{i=1}^{n} \Bigl(\mathrm{d}_{\bar{W}_i\bar{Z}_i} \otimes \bigl(\bar{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\bigr)[\bar{Z}_i/\bar{X}_i]\Bigr)\Bigr). \tag{7.54}$$

This is done by considering that, by the inductive hypothesis, we can write:

$$\begin{aligned}\bigl(\bar{\exists}_{\vec{\Lambda}_i}\, \check{C}^{\mathrm{s}}_i)\bigr)[\bar{Z}_i/\vec{\Lambda}_i] &= \bigl(\bar{\exists}_{\bar{Y}_i}\, \hat{C}^{\mathrm{c}}_i\bigr)[\bar{Z}_i/\bar{Y}_i] \otimes \bigl(\bar{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\bigr)[\bar{Z}_i/\bar{X}_i] \\ &\quad \text{[by (7.48)]} \\ &= \bigl(\bar{\exists}_{\vec{\Lambda}_i}\, \check{C}^{\mathrm{c}}_i\bigr)[\bar{Z}_i/\vec{\Lambda}_i] \otimes \bigl(\bar{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\bigr)[\bar{Z}_i/\bar{X}_i] \\ &\quad \text{[by inductive hypothesis]}\end{aligned} \tag{7.55}$$

Now, we consider the generic expression

$$\begin{aligned}E_i \overset{\text{def}}{=} \bar{\exists}_{\vec{\Lambda}}\Bigl(C^{\mathrm{c}} \otimes C\ddot{\rho} \otimes \bigotimes_{j=1}^{i-1} \Bigl(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \bigl(\bar{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_j\bigr)[\bar{Z}_j/\bar{X}_j]\Bigr) \\ \otimes \bigotimes_{j=i}^{n} \Bigl(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \bigl(\bar{\exists}_{\vec{\Lambda}_j}\, \check{C}^{\mathrm{c}}_j\bigr)[\bar{Z}_j/\bar{Y}_j] \otimes \bigl(\bar{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_j\bigr)[\bar{Z}_j/\bar{X}_j]\Bigr)\Bigr).\end{aligned} \tag{7.56}$$

The right-hand sides of (7.44) and (7.54) are, considering (7.55), given by E_1 and E_{n+1}, respectively. It is possible to prove that $E_1 = E_n$ by showing that $E_1 = E_2$, $E_2 = E_3$, ..., $E_n = E_{n+1}$. The equality $E_i = E_{i+1}$ can be obtained using S_2 of Definition 110. To this purpose, we consider the following expression for E_i

$$\begin{aligned}E_i \overset{\text{def}}{=} \bar{\exists}_{\vec{\Lambda}}\Bigl(& \overbrace{C^{\mathrm{c}} \otimes C\ddot{\rho} \otimes \bigotimes_{j=1}^{i-1} \Bigl(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \bigl(\bar{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_j\bigr)[\bar{Z}_j/\bar{X}_j]\Bigr) \otimes \mathrm{d}_{\bar{W}_i\bar{Z}_i}}^{C^i} \\ & \otimes \bigl(\bar{\exists}_{\vec{\Lambda}_i}\, \check{C}^{\mathrm{c}}_i\bigr)[\bar{Z}_i/\vec{\Lambda}_i] \otimes \bigl(\bar{\exists}_{\bar{X}_i}\, \grave{C}^{\mathrm{s}}_i\bigr)[\bar{Z}_i/\bar{X}_i] \\ \otimes \bigotimes_{j=i+1}^{n} & \Bigl(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \bigl(\bar{\exists}_{\vec{\Lambda}_j}\, \check{C}^{\mathrm{c}}_j\bigr)[\bar{Z}_j/\vec{\Lambda}_j] \otimes \bigl(\bar{\exists}_{\bar{X}_j}\, \grave{C}^{\mathrm{s}}_j\bigr)[\bar{Z}_j/\bar{X}_j]\Bigr)\Bigr),\end{aligned} \tag{7.57}$$

By (7.45), for each $i = 1, \ldots, n$,:

$$\big(\bar{\exists}_{\vec{\Lambda}_i} \check{C}_i^{\mathrm{c}}\big)[\bar{Z}_j/\vec{\Lambda}_i] = \bar{\exists}_{\bar{Z}_i} C^i$$

thus S_2 applies giving the desired result. It remains to be proved that $C^{\mathrm{s}} = \bar{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{s}}$. For this purpose we define

$$\rho_n \stackrel{\text{def}}{=} [\vec{\Lambda}/\bar{Y}] \,;\, [\bar{W}_1/\bar{Y}_1] \,;\, \cdots \,;\, [\bar{W}_n/\bar{Y}_n] \,;\, [\bar{Z}_1/\bar{X}_1] \,;\, \cdots \,;\, [\bar{Z}_n/\bar{X}_n],$$

then

$$\begin{aligned}
C^{\mathrm{s}} &= \bar{\exists}_{\bar{X}}\Big(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{i=1}^{n} \big(\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes \bar{\exists}_{\bar{X}_i} \grave{C}_i^{\mathrm{s}}\big)\Big)[\vec{\Lambda}/\bar{X}] \\
&\quad\text{[by (7.32)]} \\
&= \bar{\exists}_{\bar{X}}\Big(\Big(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{i=1}^{n} \big(\mathrm{d}_{\bar{Y}_i\bar{X}_i} \otimes \bar{\exists}_{\bar{X}_i} \grave{C}_i^{\mathrm{s}}\big)\Big)\rho_n\Big)[\vec{\Lambda}/\bar{X}] \\
&\quad\text{[by } H_6\text{, since } FV(\hat{C}^{\mathrm{s}}) \cap \mathrm{cod}(\rho_n) = \bar{X} \cap \mathrm{cod}(\rho_n) = \bar{X} \cap \mathrm{dom}(\rho_n) = \varnothing] \\
&= \bar{\exists}_{\bar{X}}\Big(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\vec{\Lambda}} \\
&\qquad\qquad \otimes C\ddot{\rho} \otimes \bigotimes_{i=1}^{n} \big(\mathrm{d}_{\bar{W}_i\bar{Z}_i} \otimes \big(\bar{\exists}_{\bar{X}_i} \grave{C}_i^{\mathrm{s}}\big)[\bar{Z}_i/\bar{X}_i]\big)\Big)[\vec{\Lambda}/\bar{X}] \\
&\quad\text{[by } H_3 \text{ and the definition of } \rho_n] \\
&= \bar{\exists}_{\vec{\Lambda}}\Bigg(\big(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}}\big)[\vec{\Lambda}/\bar{X}] \otimes \mathrm{d}_{\vec{\Lambda}\vec{\Lambda}} \\
&\qquad\qquad \otimes \overbrace{\Big(C\ddot{\rho} \otimes \bigotimes_{i=1}^{n} \Big(\mathrm{d}_{\bar{W}_i\bar{Z}_i} \otimes \big(\bar{\exists}_{\bar{X}_i} \grave{C}_i^{\mathrm{s}}\big)[\bar{Z}_i/\bar{X}_i]\Big)\Big)}^{G}[\vec{\Lambda}/\bar{X}]\Bigg) \qquad (7.58) \\
&\quad\text{[by } H_4 \text{ and } H_3] \\
&= \bar{\exists}_{\vec{\Lambda}}\Big(C^{\mathrm{c}} \otimes C\ddot{\rho} \otimes \bigotimes_{i=1}^{n} \big(\mathrm{d}_{\bar{W}_i\bar{Z}_i} \otimes \big(\bar{\exists}_{\bar{X}_i} \grave{C}_i^{\mathrm{s}}\big)[\bar{Z}_i/\bar{X}_i]\big)\Big) \\
&\quad\text{[by definition of } C^{\mathrm{c}} \text{ and since } FV(G) \cap \bar{X} = \varnothing] \\
&= \bar{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{s}} \\
&\quad\text{[by (7.54)]}
\end{aligned}$$

Therefore, we have the thesis: $(C^{\mathrm{c}} \mapsto C^{\mathrm{s}}) \in \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r)$. The induction step for (a) can be carried out in a similar way.

Second Inclusion

We now show that $\mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} \restriction \{1, \ldots, \# P\} \subseteq \mathcal{O}_{\bar{\mathcal{D}}_{\mathrm{F}}}(P, G)$. Staten differently:

(a) if there exists a magic clause $r' \in \{\# P + 1, \ldots, \# P_m^G\}$ in P_m^G such that $\big(\bar{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{c}} \mapsto \bar{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{c}}\big) \in \mathcal{F}_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(r')$, then there exists a partial derivation from a goal $G_0 \in G$ of the form $G_0 \leadsto_P^* \hat{C}^{\mathrm{c}} \,\Box\, p(\bar{X}) :: B$, such that

$$\bar{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{c}} = \big(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}}\big)[\vec{\Lambda}/\bar{X}]$$

(b) If there exists a modified clause r of P^G_m such that the pair $\bigl(\bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} \mapsto \bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{s}}\bigr)$ belongs to $\mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r)$, then there exists a partial derivation from a goal $G_0 \in G$ of the form

$$\begin{aligned} G_0 &\rightsquigarrow^*_P \hat{C}^{\mathrm{c}} \,\square\, p(\bar{X}) :: B \\ &\rightsquigarrow_{\{R_k\}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \,\square\, \bigl(p_1(\bar{Y}_1),\ldots,p_n(\bar{Y}_n)\bigr) :: B \\ &\rightsquigarrow^*_P \hat{C}^{\mathrm{s}} \,\square\, B, \qquad \text{with } \hat{C}^{\mathrm{s}} \neq \mathbf{0}, \end{aligned}$$

where $R_k = \bigl(p(\bar{Y}) \mathrel{:\!-} C\rho \,\square\, p_1(\bar{Y}_1),\ldots,p_n(\bar{Y}_n)\bigr)$ is a variant of $P[r]$ such that $FV(R_k) \cap \Bigl(FV(G_0) \cup \bigl(\bigcup_{j=1}^{k-1} FV(R_j)\bigr)\Bigr) = \varnothing$. Moreover we have

$$\bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} = \bigl(\bar{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{c}}\bigr)[\vec{\Lambda}/\bar{X}] \quad \text{and} \quad \bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{s}} = \bigl(\bar{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{s}}\bigr)[\vec{\Lambda}/\bar{X}]$$

The proof has the following structure. The hypotheses say that for some $r \in \{1,\ldots,\#\,P^G_m\}$ we have some $e \in \mathcal{D}_{\mathrm{F}}$ such that e belongs to $\mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(r)$. Since

$$\mathcal{F}^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G} = \bigcup_{i \in \mathbb{N}} T^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G} \uparrow i,$$

there must exist $k \in \mathbb{N}$ such that $e \in \bigl(T^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G} \uparrow k\bigr)(r)$. The proof continues by induction on k.

Base case (a). For $k = 1$ there must exist r' such that

$$\bigl(\bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} \mapsto \bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}}\bigr) \in \bigl(T^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}(\varnothing)\bigr)(r').$$

Thus, by definition of $T^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}$, we have $P^G_m[r'] = \bigl(\mathtt{m\_}p(\vec{\Lambda}) \mathrel{:\!-} \check{C}^{\mathrm{c}}\bigr)$. By the definition of magic program, $P^G_m[r']$ describes the first call in an initial goal $G_0 \in G$ of the form $G_0 = \bigl(\hat{C}^{\mathrm{c}} \,\square\, p(\bar{X}) :: B\bigr)$, where $\bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} = \bigl(\bar{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{c}}\bigr)[\vec{\Lambda}/\bar{X}] = C^{\mathrm{c}}$.

Base case (b). Here we take $k = 2$, since each modified clause has at least an atom: the magic one. If, for some r, we have

$$\bigl(\bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} \mapsto \bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{s}}\bigr) \in \bigl(T^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G} \uparrow 2\bigr)(r), \tag{7.59}$$

then $P^G_m[r] = \bigl(p(\vec{\Lambda}) \mathrel{:\!-} \langle \mathtt{m\_}p(\vec{\Lambda}), C\ddot{\rho}, \varnothing \rangle\bigr)$ with

$$\bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{s}} = \bar{\exists}_{\vec{\Lambda}}\bigl(\bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} \otimes C\ddot{\rho}\bigr) \tag{7.60}$$

where $\bar{U}\ddot{\rho} = \vec{\Lambda}$ and $\check{C}^{\mathrm{s}} \neq \mathbf{0}$, by $\otimes_{\mathrm{F}}$ definition (case (ii)). Thus, by $T^{\bar{\mathcal{D}}_{\mathrm{F}}}_{P_G}$ definition, there exists a clause r' defining $\mathtt{m\_}p$ such that $\bigl(\bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} \mapsto \bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}}\bigr) \in$

$\big(T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}(\varnothing)\big)(r')$. By the base case (a) there exists a initial goal $G_0 = \big(\hat{C}^{\mathrm{c}} \,\square\, p(\bar{X}) :: B\big)$ such that (7.60) can be rewritten as

$$\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{s}} = \bar{\exists}_{\vec{\Lambda}}(C^{\mathrm{c}} \otimes C\ddot{\rho}). \tag{7.61}$$

Consider the clause $P[r]$ from which $P_m^G[r]$ was generated. Let $R_1 = \big(p(\bar{Y}) \mathrel{:-} C\rho \,\square\, \varnothing\big)$ be a variant of $P[r]$ such that $FV(G_0) \cap FV(R_1) = \varnothing$. Since $\hat{C}^{\mathrm{c}} \neq \mathbf{0}$ by base case (a) there must exist a partial derivation

$$G_0 = \big(\hat{C}^{\mathrm{c}} \,\square\, p(\bar{X}) :: B\big) \rightsquigarrow_{\{R_1\}} \overbrace{\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho}^{\hat{C}^{\mathrm{s}}} \,\square\, B \tag{7.62}$$

We now prove that the partial answer constraint of (7.62), suitably renamed, is the success component of the pair in (7.59), namely

$$\bar{\exists}_{\vec{\Lambda}}\big(C^{\mathrm{c}} \otimes C\ddot{\rho}\big) = \Big(\bar{\exists}_{\bar{X}}\big(\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho\big)\Big)[\vec{\Lambda}/\bar{X}], \tag{7.63}$$

and thus that we also have $\hat{C}^{\mathrm{s}} \neq \mathbf{0}$. Equation (7.63) is identical to (7.25), thus (7.63) holds, as we saw while proving the first implication.

Inductive case (b). We now consider the case $k > 2$. If for some r we have that

$$\big(\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{c}} \mapsto \bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{s}}\big) \in \big(T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} \uparrow k\big)(r)$$

and

$$P_m^G[r] = \Big(p(\vec{\Lambda}) \mathrel{:-} \Big\langle \{\mathtt{m\_}p(\vec{\Lambda})\}, C\ddot{\rho}, \big(p_1(\bar{W}_1), \dots, p_n(\bar{W}_n)\big)\Big\rangle\Big),$$

then we must have, by $T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}}$ definition, that

$$\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{s}} = \bar{\exists}_{\vec{\Lambda}}\Big(\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{c}} \otimes C\ddot{\rho} \otimes \bigotimes_{i=1}^{n} \big(\mathrm{d}_{\bar{W}_i\bar{Z}_i} \otimes \big(\bar{\exists}_{\vec{\Lambda}_i}\, \check{C}_i^{\mathrm{s}}\big)[\bar{Z}_i/\vec{\Lambda}_i]\big)\Big), \tag{7.64}$$

where, by $\otimes_{\mathrm{F}}$ definition, $\check{C}^{\mathrm{s}} \neq \mathbf{0}$ and for each $i = 1, \dots, n$,

$$\bar{\exists}_{\bar{Z}_i}\left(\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{c}} \otimes C\ddot{\rho} \otimes \bigotimes_{j=1}^{i-1} \Big(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \big(\bar{\exists}_{\vec{\Lambda}_j}\, \check{C}_j^{\mathrm{s}}\big)[\bar{Z}_i/\vec{\Lambda}_i]\Big) \otimes \mathrm{d}_{\bar{W}_i\bar{Z}_i}\right) = \big(\bar{\exists}_{\vec{\Lambda}_i}\, \check{C}_i^{\mathrm{c}}\big)[\bar{Z}_i/\vec{\Lambda}_i]. \tag{7.65}$$

Moreover, for each $i = 1, \dots, n$, we have

$$\bar{Z}_i \ll_{\vec{\Lambda}_i} FV(P_m^G[r]) \cup \Lambda \cup \bar{Z}_1 \cup \cdots \cup \bar{Z}_{i-1},$$

and $\check{C}^{\mathrm{c}}$, $\check{C}_1^{\mathrm{s}}$, …, $\check{C}_n^{\mathrm{s}}$ are given by the inductive hypotheses as follows.

1. There exists r' such that $P_m^G[r']$ describes a call to p such that

   $$\big(\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{c}} \mapsto \bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{c}}\big) \in \big(T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} \uparrow (k-1)\big)(r')$$

   By the inductive hypothesis, there exists a derivation

   $$G_0 \leadsto_P^* \hat{C}^{\mathrm{c}} \,\square\, p(\bar{X}) :: B \tag{7.66}$$

   such that $G_0 \in G$ and $\bar{\exists}_{\vec{\Lambda}}\, \check{C}^{\mathrm{c}} = \big(\bar{\exists}_{\bar{X}}\, \hat{C}^{\mathrm{c}}\big)[\vec{\Lambda}/\bar{X}] = C^{\mathrm{c}}$.

2. For each $i = 1, \ldots, n$, there exists r_i such that

   $$\big(\bar{\exists}_{\vec{\Lambda}_i}\, \check{C}_i^{\mathrm{c}} \mapsto \bar{\exists}_{\vec{\Lambda}_i}\, \check{C}_i^{\mathrm{s}}\big) \in \big(T_{P_G}^{\bar{\mathcal{D}}_{\mathrm{F}}} \uparrow (k-1)\big)(r_i).$$

   By the inductive hypothesis, for $i = 1, \ldots, n$, there exists a derivation from $G_0^i \in G$

   $$\begin{aligned} G_0^i \leadsto_P^* G_{k_i}^i &= \big(\hat{C}_i^{\mathrm{c}} \,\square\, p_i(Y_i^i) :: B^i\big) \\ &\leadsto_{\{R_{k+h_i}^i\}} \hat{C}_i^{\mathrm{c}} \otimes \mathrm{d}_{\bar{Y}_i^i \bar{X}_i^i} \otimes C_i \rho_i^i \,\square\, B_{r_i}^i :: B^i \\ &\leadsto_P^* \hat{C}_i^{\mathrm{s}} \,\square\, B^i, \end{aligned} \tag{7.67}$$

   where $R_{k+h_i}^i = \big(p_i(\bar{X}_i^i) :\!-\; C_i \rho_i^i \,\square\, B_{r_i}^i\big)$ is a variant of $P[r_i]$ such that $FV(R_{k+h_i}^i) \cap \big(FV(G_0) \cup \big(\bigcup_{j=1}^{k+h_i-1} FV(R_j^i)\big)\big) = \varnothing$. Furthermore,

   $$\bar{\exists}_{\vec{\Lambda}_i}\, \check{C}_i^{\mathrm{c}} = \big(\bar{\exists}_{\bar{Y}_i^i}\, \hat{C}_i^{\mathrm{c}}\big)[\vec{\Lambda}_i/\bar{Y}_i^i] = C_i^{\mathrm{c}}$$

   and

   $$\bar{\exists}_{\vec{\Lambda}_i}\, \check{C}_i^{\mathrm{s}} = \big(\bar{\exists}_{\bar{Y}_i^i}\, \hat{C}_i^{\mathrm{s}}\big)[\vec{\Lambda}_i/\bar{Y}_i^i] = C_i^{\mathrm{s}}.$$

   Now, by (7.67), for each $i, \ldots, n$, there must exist the following derivation:

   $$\mathbf{1} \,\square\, p_i(\bar{Y}_i^i) \leadsto_{\{R_{k+h_i}^i\}} \mathbf{1} \otimes \mathrm{d}_{\bar{Y}_i^i \bar{X}_i^i} \otimes C_i \rho_i^i \,\square\, B_{r_i}^i \leadsto_P^* \grave{C}_i^{\mathrm{s}} \,\square\, \varnothing, \tag{7.68}$$

   and $\hat{C}_i^{\mathrm{s}} = \hat{C}_i^{\mathrm{c}} \otimes \grave{C}_i^{\mathrm{s}}$. Observe that in (7.68) we have used the same variants employed in (7.67). This implies that $FV(\grave{C}_i^{\mathrm{s}}) \cap \bar{Y}_i^i = \varnothing$.

Then we have:

$$
\begin{aligned}
\bigl(\overline{\exists}_{\bar{Y}_i^i}\,\hat{C}_i^{\mathrm{s}}\bigr)[\vec{\Lambda}_i/\bar{Y}_i^i] &= \bigl(\overline{\exists}_{\bar{Y}_i^i}\bigl(\hat{C}_i^{\mathrm{c}}\otimes\grave{C}_i^{\mathrm{s}}\bigr)\bigr)[\vec{\Lambda}_i/\bar{Y}_i^i]\\
&\quad\text{[by the above result]}\\
&= \bigl(\overline{\exists}_{\bar{Y}_i^i}\bigl(\overline{\exists}_{\bar{Y}_i^i}\,\hat{C}_i^{\mathrm{c}}\otimes\grave{C}_i^{\mathrm{s}}\bigr)\bigr)[\vec{\Lambda}_i/\bar{Y}_i^i]\\
&\quad\left[\begin{array}{l}\text{by } S_7 \text{ since } FV(\hat{C}_i^{\mathrm{c}})\cap FV(\grave{C}_i^{\mathrm{s}})\subseteq\bar{Y}_i^i \text{ and}\\ FV(\hat{C}_i^{\mathrm{c}})\cap\bar{Y}_i^i\subseteq\bar{Y}_i^i\end{array}\right]\\
&= \bigl(\overline{\exists}_{\bar{Y}_i^i}\,\hat{C}_i^{\mathrm{c}}\bigr)[\vec{\Lambda}_i/\bar{Y}_i^i]\otimes\bigl(\overline{\exists}_{\bar{Y}_i^i}\,\grave{C}_i^{\mathrm{s}}\bigr)[\vec{\Lambda}_i/\bar{Y}_i^i] \qquad (7.69)\\
&\quad\text{[by } S_4 \text{ and } H_3\text{]}\\
&= \overline{\exists}_{\vec{\Lambda}_i}\,\check{C}_i^{\mathrm{c}}\otimes\bigl(\overline{\exists}_{\bar{Y}_i^i}\,\grave{C}_i^{\mathrm{s}}\bigr)[\vec{\Lambda}_i/\bar{Y}_i^i]\\
&\quad\text{[by inductive hypothesis 2]}\\
&= \bigl(\overline{\exists}_{\bar{Y}_i^i}\,\hat{C}_i^{\mathrm{s}}\bigr)[\vec{\Lambda}_i/\bar{Y}_i^i][\bar{Z}_i/\vec{\Lambda}_i]\\
&\quad\text{[by } H_5\text{]}
\end{aligned}
$$

Thus,

$$
\begin{aligned}
\bigl(\overline{\exists}_{\bar{Y}_i^i}\,\hat{C}_i^{\mathrm{s}}\bigr)[\bar{Z}_i/\bar{Y}_i^i] &= \bigl(\overline{\exists}_{\vec{\Lambda}_i}\,\check{C}_i^{\mathrm{c}}\bigr)[\bar{Z}_i/\vec{\Lambda}_i]\otimes\bigl(\overline{\exists}_{\bar{Y}_i^i}\,\grave{C}_i^{\mathrm{s}}\bigr)[\vec{\Lambda}_i/\bar{Y}_i^i][\bar{Z}_i/\vec{\Lambda}_i].\\
&\quad\text{[by (7.69) and } H_3\text{]}\\
&= \bigl(\overline{\exists}_{\vec{\Lambda}_i}\,\check{C}_i^{\mathrm{c}}\bigr)[\bar{Z}_i/\vec{\Lambda}_i]\otimes\bigl(\overline{\exists}_{\bar{Y}_i^i}\,\grave{C}_i^{\mathrm{s}}\bigr)[\bar{Z}_i/\bar{Y}_i^i]\\
&\quad\text{[by } H_5\text{]}
\end{aligned}
\tag{7.70}
$$

We will now show that

$$
\overline{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{s}} = \overline{\exists}_{\vec{\Lambda}}\Bigl(\overline{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}}\otimes C\ddot{\rho}\otimes\bigotimes_{i=1}^{n}\Bigl(\mathrm{d}_{\bar{W}_i\bar{Z}_i}\otimes\bigl(\overline{\exists}_{\bar{Y}_i^i}\,\grave{C}_i^{\mathrm{s}}\bigr)[\bar{Z}_i/\bar{Y}_i^i]\Bigr)\Bigr). \tag{7.71}
$$

Consider the generic expression

$$
\begin{aligned}
E_i \overset{\mathrm{def}}{=} \overline{\exists}_{\vec{\Lambda}}\Biggl(&\overline{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}}\otimes C\ddot{\rho}\otimes\bigotimes_{j=1}^{i}\Bigl(\mathrm{d}_{\bar{W}_j\bar{Z}_j}\otimes\bigl(\overline{\exists}_{\vec{\Lambda}_j}\,\check{C}_j^{\mathrm{s}}\bigr)[\bar{Z}_j/\vec{\Lambda}_j]\Bigr)\\
&\otimes\bigotimes_{j=i+1}^{n}\Bigl(\mathrm{d}_{\bar{W}_j\bar{Z}_j}\otimes\bigl(\overline{\exists}_{\bar{Y}_j^j}\,\grave{C}_j^{\mathrm{s}}\bigr)[\bar{Z}_j/\bar{Y}_j^j]\Bigr)\Biggr).
\end{aligned}
\tag{7.72}
$$

The right-hand sides of (7.64) and (7.71) are given by E_n and E_0, respectively. It is possible to prove that $E_n = E_0$ by showing that $E_n = E_{n-1}$, $E_{n-1} = E_{n-2}$, ... , $E_1 = E_0$. The equality $E_i = E_{i-1}$ can be obtained using S_2 of Definition 110. For this purpose, we consider, taking into account

(7.70), the following expression for E_i

$$E_i \stackrel{\text{def}}{=} \bar{\exists}_{\vec{\Lambda}}\Biggl(\overbrace{\bar{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} \otimes C\ddot{\rho} \otimes \bigotimes_{j=1}^{i-1}\Bigl(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \bigl(\bar{\exists}_{\vec{\Lambda}_j}\,\check{C}_j^{\mathrm{s}}\bigr)[\bar{Z}_j/\vec{\Lambda}_j]\Bigr)}^{C^{i-1}} \\ \otimes \mathrm{d}_{\bar{W}_i\bar{Z}_i} \otimes \bigl(\bar{\exists}_{\vec{\Lambda}_i}\,\check{C}_i^{\mathrm{c}}\bigr)[\bar{Z}_i/\vec{\Lambda}_i] \otimes \bigl(\bar{\exists}_{\bar{Y}_i^i}\,\grave{C}_i^{\mathrm{s}}\bigr)[\bar{Z}_i/\bar{Y}_i^i] \\ \otimes \bigotimes_{j=i+1}^{n}\Bigl(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \bigl(\bar{\exists}_{\bar{Y}_j^j}\,\grave{C}_j^{\mathrm{s}}\bigr)[\bar{Z}_j/\bar{Y}_j^j]\Bigr)\Biggr). \tag{7.73}$$

By (7.65), for each $i = 1, \ldots, n$ we have

$$\bar{\exists}_{\bar{Z}_i}\,C^{i-1} = \bigl(\bar{\exists}_{\vec{\Lambda}_i}\,\check{C}_i^{\mathrm{c}}\bigr)[\bar{Z}_i/\vec{\Lambda}_i],$$

thus S_2 applies giving the desired result.

The aim is now to show that we can reconstruct an existing derivation whose answer constraint is the right-hand side of (7.71), by composing the partial derivations described in (7.66) and (7.67).

We show this by total induction on the number of calls, n. We start by considering clause $P[r]$, which originated the modified clause $P_m^G[r]$, and the derivation (7.66):

$$G_0 \rightsquigarrow_P^* \hat{C}^{\mathrm{c}} \,\Box\, p(\bar{X}) :: B.$$

Since $\hat{C}^{\mathrm{c}} \neq \mathbf{0}$ (by the inductive hypothesis), there exists a variant of $P[r]$,

$$R_k = \Bigl(p(\bar{Y}) \mathrel{:-} C\rho \,\Box\, \bigl(p_1(\bar{Y}_1), \ldots, p_n(\bar{Y}_n)\bigr)\Bigr),$$

where $\bar{U}\rho = \bar{Y}$, $\bar{U}_1\rho = \bar{Y}_1$, … ,$\bar{U}_n\rho = \bar{Y}_n$, and

$$FV(R_k) \cap \Bigl(FV(G_0) \cup \bigcup_{i=1}^{k-1} FV(R_i)\Bigr) = \varnothing,$$

such that we have

$$\begin{aligned} G_0 &\rightsquigarrow_P^* G_k \\ &= \hat{C}^{\mathrm{c}} \,\Box\, p(\bar{X}) :: B \\ &\rightsquigarrow_{\{R_k\}} \underbrace{\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho}_{\hat{C}_1^{\mathrm{c}}} \,\Box\, \bigl(p_1(\bar{Y}_1), \ldots, p_n(\bar{Y}_n)\bigr) :: B \end{aligned} \tag{7.74}$$

If we project $\hat{C}_1^{\mathrm{c}}$ onto the relevant variables $\bar{Y}_1$, we have that

$$\begin{aligned} \bar{\exists}_{\bar{Y}_1}\,\hat{C}_1^{\mathrm{c}} &= \bar{\exists}_{\bar{Y}_1}\bigl(\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho\bigr) \\ &= \bar{\exists}_{\bar{Y}_1}\bigl(\bar{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho\bigr) \\ &\qquad \text{[by Lemma 126]} \end{aligned} \tag{7.75}$$

We show that this call-pattern, suitably renamed onto $\vec{\Lambda}_1$, is exactly the call-pattern considered in the bottom-up construction. We consider a renaming ρ'_1 such that

$$\vec{\Lambda}\rho'_1 = \bar{Y}, \bar{W}_1\rho'_1 = \bar{Y}_1, \ldots, \bar{W}_n\rho'_1 = \bar{Y}_n.$$

By (7.65) we can write

$$\begin{aligned}
\overline{\exists}_{\vec{\Lambda}_1} \check{C}^{\mathrm{c}}_1 &= \Big(\overline{\exists}_{\bar{Z}_1} \big(\overline{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{c}} \otimes C\ddot{\rho} \otimes \mathrm{d}_{\bar{W}_1\bar{Z}_1}\big)\Big)[\vec{\Lambda}_1/\bar{Z}_1] \\
&= \Big(\overline{\exists}_{\bar{Z}_1} \Big(\big(\overline{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{c}} \otimes C\ddot{\rho}\big)[\bar{Z}_1/\bar{W}_1]\Big)[\vec{\Lambda}_1/\bar{Z}_1] \\
&\quad \text{[by } S_5 \text{ with } [\bar{Z}_1/\bar{W}_1] = \rho\text{]} \\
&= \Big(\overline{\exists}_{\bar{W}_1} \big(\overline{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{c}} \otimes C\ddot{\rho}\big)\Big)[\bar{Z}_1/\bar{W}_1][\vec{\Lambda}_1/\bar{Z}_1] \\
&\quad \text{[by } H_4\text{]} \\
&= \Big(\overline{\exists}_{\bar{W}_1} \big(\overline{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{c}} \otimes C\ddot{\rho}\big)\Big)[\vec{\Lambda}_1/\bar{W}_1] \\
&\quad \text{[by } H_5\text{]} \\
&= \Big(\overline{\exists}_{\bar{W}_1} \big((\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}})[\vec{\Lambda}/\bar{X}] \otimes C\ddot{\rho}\big)\Big)[\vec{\Lambda}_1/\bar{W}_1] \\
&\quad \text{[by inductive hypothesis 1]} \\
&= \Big(\overline{\exists}_{\bar{W}_1} \big(\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\vec{\Lambda}} \otimes C\ddot{\rho}\big)\Big)[\vec{\Lambda}_1/\bar{W}_1] \\
&\quad \begin{bmatrix}\text{by Lemma 124, since } FV(C\ddot{\rho}) \cap \bar{X} = \varnothing \\ \text{and } FV(\hat{C}^{\mathrm{c}}) \cap \vec{\Lambda} = \bar{X} \cap \vec{\Lambda} = \bar{X} \cap \bar{W}_i = \varnothing\end{bmatrix} \\
&= \Big(\overline{\exists}_{\bar{W}_1} \big(\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\vec{\Lambda}} \otimes C\ddot{\rho}\big)\rho'_1[\bar{W}_1/\bar{Y}_1]\Big)[\vec{\Lambda}_1/\bar{W}_1] \\
&\quad \text{[by } H_6\text{, since } FV(\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\vec{\Lambda}} \otimes C\ddot{\rho}) \cap \mathrm{cod}(\rho'_1[\bar{W}_1/\bar{Y}_1]) = \varnothing\text{]} \\
&= \Big(\overline{\exists}_{\bar{W}_1} \Big(\big(\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho\big)\Big)[\bar{W}_1/\bar{Y}_1]\Big)[\vec{\Lambda}_1/\bar{W}_1] \\
&\quad \text{[by the definitions of } \rho'_1, \rho, \text{ and } \ddot{\rho}\text{]} \\
&= \Big(\Big(\overline{\exists}_{\bar{Y}_1} \big(\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho\big)\Big)[\bar{W}_1/\bar{Y}_1]\Big)[\vec{\Lambda}_1/\bar{W}_1] \\
&\quad \text{[by } H_4\text{]} \\
&= \Big(\overline{\exists}_{\bar{Y}_1} \big(\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho\big)\Big)[\vec{\Lambda}_1/\bar{Y}_1] \\
&\quad \text{[by } H_5\text{]} \\
&= \big(\overline{\exists}_{\bar{Y}_1} \hat{C}^{\mathrm{c}}_1\big)[\vec{\Lambda}_1/\bar{Y}_1]
\end{aligned}$$

Then we have that $\overline{\exists}_{\vec{\Lambda}_i} \check{C}^{\mathrm{c}}_1 = \big(\overline{\exists}_{\bar{Y}_1} \hat{C}^{\mathrm{c}}_1\big)[\vec{\Lambda}_1/\bar{Y}_1] = \big(\overline{\exists}_{\bar{Y}^1_1} \hat{C}^{\mathrm{c}}_1\big)[\vec{\Lambda}_1/\bar{Y}^1_1]$ by the inductive hypothesis 2.

The goal $G^1_{k_1}$ in (7.67) (for $i = 1$) is (a variant of) G_k in (7.74). Therefore, we can continue the derivation as in (7.67), but using a different variant of clause $P[r_1]$:

$$\begin{aligned} G_0 \leadsto_P^* G_k &= \bigl(\hat{C}^{\mathrm{c}} \mathbin{\Box} p(\bar{X}) :: B\bigr) \\ &\leadsto_{\{R_k\}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \mathbin{\Box} \bigl(p_1(\bar{Y}_1), \dots, p_n(\bar{Y}_n)\bigr) :: B \\ &\leadsto_P^* \hat{C}_1^{\mathrm{s}} \mathbin{\Box} \bigl(p_2(\bar{Y}_2), \dots, p_n(\bar{Y}_n)\bigr) :: B, \end{aligned}$$

so we have $\bigl(\bar{\exists}_{\bar{Y}_1} \hat{C}_1^{\mathrm{s}}\bigr)[\vec{\Lambda}_1/\bar{Y}_1] = \bar{\exists}_{\vec{\Lambda}_1} \check{C}_1^{\mathrm{s}}$. Now, suppose that what we have just proved for the first call holds for all the calls up to the $(i-1)$-th one. This means that

$$\begin{array}{ll} G_0 \leadsto_P^* & \hat{C}^{\mathrm{c}} \mathbin{\Box} p(\bar{X}) :: B \\ \quad \leadsto_{\{R_k\}} & \hat{C}_1^{\mathrm{c}} \mathbin{\Box} \bigl(p_1(\bar{Y}_1), \dots, p_n(\bar{Y}_n)\bigr) :: B \\ \quad \leadsto_P^* & \hat{C}_1^{\mathrm{s}} \mathbin{\Box} \bigl(p_2(\bar{Y}_2), \dots, p_n(\bar{Y}_n)\bigr) :: B \\ \quad = & \hat{C}_2^{\mathrm{c}} \mathbin{\Box} \bigl(p_2(\bar{Y}_2), \dots, p_n(\bar{Y}_n)\bigr) :: B \\ \quad \leadsto_P^* & \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \quad \vdots & \qquad\qquad\qquad \vdots \\ \quad \leadsto_P^* & \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \quad \leadsto_P^* & \hat{C}_{i-1}^{\mathrm{c}} \mathbin{\Box} \bigl(p_{i-1}(\bar{Y}_{i-1}), \dots, p_n(\bar{Y}_n)\bigr) :: B \\ \quad \leadsto_P^* & \hat{C}_{i-1}^{\mathrm{s}} \mathbin{\Box} \bigl(p_i(\bar{Y}_i), \dots, p_n(\bar{Y}_n)\bigr) :: B, \end{array} \tag{7.76}$$

for each $j \leq i-1$ we have

$$\begin{aligned} \bar{\exists}_{\vec{\Lambda}_j} \check{C}_j^{\mathrm{c}} &= \bigl(\bar{\exists}_{\bar{Y}_j} \hat{C}_j^{\mathrm{c}}\bigr)[\vec{\Lambda}_j/\bar{Y}_j], \\ \bar{\exists}_{\vec{\Lambda}_j} \check{C}_j^{\mathrm{s}} &= \bigl(\bar{\exists}_{\bar{Y}_j} \hat{C}_j^{\mathrm{s}}\bigr)[\vec{\Lambda}_j/\bar{Y}_j] \end{aligned}$$

where

$$\begin{aligned} \bar{\exists}_{\bar{Y}_j} \hat{C}_j^{\mathrm{c}} &= \bar{\exists}_{\bar{Y}_j} \Bigl(\hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{k=1}^{j-1} \bigl(\mathrm{d}_{\bar{Y}_k\bar{X}_k} \otimes \grave{C}_k^{\mathrm{s}}\bigr)\Bigr) \\ &= \bar{\exists}_{\bar{Y}_j} \Bigl(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{k=1}^{j-1} \bigl(\mathrm{d}_{\bar{Y}_k\bar{X}_k} \otimes \bar{\exists}_{\bar{X}_k} \grave{C}_k^{\mathrm{s}}\bigr)\Bigr) \\ &\text{[by Lemma 126]} \end{aligned}$$

Similarly, we have also[10]

$$\bar{\exists}_{\bar{Y}_j} \hat{C}_j^{\mathrm{s}} = \bar{\exists}_{\bar{Y}_j} \Bigl(\bar{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \otimes \bigotimes_{k=1}^{j} \bigl(\mathrm{d}_{\bar{Y}_k\bar{X}_k} \otimes \bar{\exists}_{\bar{X}_k} \grave{C}_k^{\mathrm{s}}\bigr)\Bigr)$$

We show that, by (7.76), there exists a derivation

$$\begin{aligned} G_0 &\leadsto_P^* \hat{C}_{i-1}^{\mathrm{s}} \mathbin{\Box} \bigl(p_i(\bar{Y}_i), \dots, p_n(\bar{Y}_n)\bigr) :: B, \\ &= \hat{C}_i^{\mathrm{c}} \mathbin{\Box} \bigl(p_i(\bar{Y}_i), \dots, p_n(\bar{Y}_n)\bigr) :: B \\ &\leadsto_P^* \hat{C}_i^{\mathrm{s}} \mathbin{\Box} \bigl(p_{i+1}(\bar{Y}_{i+1}), \dots, p_n(\bar{Y}_n)\bigr) :: B \end{aligned}$$

[10]We assume that, for each $j = 1, \dots, i-1$, the variant of $P[r_j]$ employed is $R_{k+h_j} = \bigl(p_j(\bar{X}_j) \mathrel{:-} C_j\rho_j \mathbin{\Box} B_{r_j}\bigr)$. However, the only important thing here is that the renaming-apart conditions must hold.

such that $\big(\overline{\exists}_{\bar{Y}_i}\,\hat{C}^{\mathrm{c}}_i\big)[\vec{\Lambda}_i/\bar{Y}_i] = \overline{\exists}_{\vec{\Lambda}_i}\,\check{C}^{\mathrm{c}}_i$ and $\big(\overline{\exists}_{\bar{Y}_i}\,\hat{C}^{\mathrm{s}}_i\big)[\vec{\Lambda}_i/\bar{Y}_i] = \overline{\exists}_{\vec{\Lambda}_i}\,\check{C}^{\mathrm{s}}_i$. We consider a renaming ρ'_i such that

$$\vec{\Lambda}\rho'_i = \bar{Y}, \bar{W}_1\rho'_i = \bar{Y}_1, \ldots, \bar{W}_n\rho'_i = \bar{Y}_n, \bar{Z}_1\rho'_i = \bar{X}_1, \ldots, \bar{Z}_i\rho'_i = \bar{X}_i.$$

Notice that

$$\ddot{\rho}\rho'_i = \rho[\bar{X}_1/\bar{Z}_1]\cdots\rho[\bar{X}_i/\bar{Z}_i].$$

Then we can write

$$\begin{aligned}
\overline{\exists}_{\vec{\Lambda}_i}\,\check{C}^{\mathrm{c}}_i &= \Big(\overline{\exists}_{\bar{Z}_i}\Big(\overline{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} \otimes C\ddot{\rho} \\
&\qquad \otimes \bigotimes_{j=1}^{i-1}\big(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \big(\overline{\exists}_{\bar{X}_j}\,\grave{C}^{\mathrm{s}}_j\big)[\bar{Z}_j/\bar{X}_j]\big) \otimes \mathrm{d}_{\bar{W}_i\bar{Z}_i}\Big)\Big)[\vec{\Lambda}_i/\bar{Z}_i] \\
&\text{[by (7.65)]} \\
&= \Big(\overline{\exists}_{\bar{Z}_i}\Big(\Big(\overline{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} \otimes C\ddot{\rho} \\
&\qquad \otimes \bigotimes_{j=1}^{i-1}\big(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \big(\overline{\exists}_{\bar{X}_j}\,\grave{C}^{\mathrm{s}}_j\big)[\bar{Z}_j/\bar{X}_j]\big)\Big)[\bar{Z}_i/\bar{W}_i]\Big)\Big)[\vec{\Lambda}_i/\bar{Z}_i] \\
&\text{[by } S_5 \text{ with } [\bar{Z}_i/\bar{W}_i] = \rho] \\
&= \Big(\overline{\exists}_{\bar{W}_i}\Big(\overline{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} \otimes C\ddot{\rho} \\
&\qquad \otimes \bigotimes_{j=1}^{i-1}\big(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \big(\overline{\exists}_{\bar{X}_j}\,\grave{C}^{\mathrm{s}}_j\big)[\bar{Z}_j/\bar{X}_j]\big)\Big)\Big)[\bar{Z}_i/\bar{W}_i][\vec{\Lambda}_i/\bar{Z}_i] \\
&\text{[by } H_4] \\
&= \Big(\overline{\exists}_{\bar{W}_i}\Big(\overline{\exists}_{\vec{\Lambda}}\,\check{C}^{\mathrm{c}} \otimes C\ddot{\rho} \\
&\qquad \otimes \bigotimes_{j=1}^{i-1}\big(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \big(\overline{\exists}_{\bar{X}_j}\,\grave{C}^{\mathrm{s}}_j\big)[\bar{Z}_j/\bar{X}_j]\big)\Big)\Big)[\vec{\Lambda}_i/\bar{W}_i] \\
&\text{[by } H_5] \\
&= \Bigg(\overline{\exists}_{\bar{W}_i}\Big(\Big(\big(\overline{\exists}_{\bar{X}}\,\hat{C}^{\mathrm{c}}\big)[\vec{\Lambda}/\bar{X}] \otimes C\ddot{\rho} \\
&\qquad \otimes \bigotimes_{j=1}^{i-1}\big(\mathrm{d}_{\bar{W}_j\bar{Z}_j} \otimes \big(\overline{\exists}_{\bar{X}_j}\,\grave{C}^{\mathrm{s}}_j\big)[\bar{Z}_j/\bar{X}_j]\big)\Big)\rho'_i[\bar{W}_i/\bar{Y}_i]\Big)\Bigg)[\vec{\Lambda}_i/\bar{W}_i] \\
&\text{[by definition of } C^{\mathrm{c}} \text{ and } H_6\text{, since } \rho'_i[\bar{W}_i/\bar{Y}_i] \text{ is a renaming for } \check{C}^{\mathrm{c}}_i]
\end{aligned}$$

$$= \Bigg(\overline{\exists}_{\bar{W}_i} \Big(\big(\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}}\big)[\bar{Y}/\bar{X}] \otimes C\rho$$

$$\otimes \bigotimes_{j=1}^{i-1} \Big(\mathrm{d}_{\bar{Y}_j \bar{X}_j} \otimes \overline{\exists}_{\bar{X}_j} \grave{C}^{\mathrm{s}}_j\Big)[\bar{W}_i/\bar{Y}_i]\Big)[\vec{\Lambda}_i/\bar{W}_i]\Bigg)$$

[by H_3 and definition of ρ', ρ and $\ddot{\rho}$]

$$= \Big(\overline{\exists}_{\bar{Y}_i} \Big(\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}}$$

$$\overbrace{\otimes\, C\rho \otimes \bigotimes_{j=1}^{i-1} \big(\mathrm{d}_{\bar{Y}_j \bar{X}_j} \otimes \overline{\exists}_{\bar{X}_j} \grave{C}^{\mathrm{s}}_j\big)}^{H}\Big)\Big)[\vec{\Lambda}_i/\bar{Y}_i]$$

$\left[\begin{array}{l}\text{by } H_4 \text{ and Lemma 124, since } FV(H) \cap \bar{X} = \\ FV(\hat{C}^{\mathrm{c}}) \cap \bar{Y} = \bar{X} \cap \bar{Y} = \bar{X} \cap \bar{Y}_i = \varnothing\end{array}\right]$

$$= \big(\overline{\exists}_{\bar{Y}_i} \hat{C}^{\mathrm{s}}_{i-1}\big)[\vec{\Lambda}_i/\bar{Y}_i]$$

We can thus conclude that there exists a derivation

$$\begin{aligned} G_0 &\rightsquigarrow^*_P \hat{C}^{\mathrm{c}} \,\Box\, p(\bar{X}) :: B \\ &\rightsquigarrow_{\{R_k\}} \hat{C}^{\mathrm{c}} \otimes \mathrm{d}_{\bar{X}\bar{Y}} \otimes C\rho \,\Box\, \big(p_1(\bar{Y}_1), \ldots, p_n(\bar{Y}_n)\big) :: B \\ &\rightsquigarrow^*_P \hat{C}^{\mathrm{s}} \,\Box\, B, \end{aligned}$$

such that $\big(\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{c}}\big)[\vec{\Lambda}/\bar{X}] = \overline{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{c}}$ and $\big(\overline{\exists}_{\bar{X}} \hat{C}^{\mathrm{s}}\big)[\vec{\Lambda}/\bar{X}] = \overline{\exists}_{\vec{\Lambda}} \check{C}^{\mathrm{s}}$, which implies $\hat{C}^{\mathrm{s}} \neq \mathbf{0}$.

The inductive case for (a) can be proved analogously. □

Chapter 8

Conclusion

Back in the introduction we have stated our (admittedly ambitious) objectives concerning the analysis of constraint logic-based languages. As the reader will have noticed, there is still much work to do. Whereas we are rather satisfied about the current status of our theoretical treatment, we did not have enough time to complete the experimental part of the project. Indeed, we underestimated both

- the effort required in order to set up the theory: we thought that many of the "needed pieces" were already there, but a closer analysis often revealed that this was not the case; and
- the effort required in order to implement the CHINA.

All the (families) of domains described in this thesis have been implemented and tested, here included a generic implementation of the hierarchy of domains described in Chapter 3. However, up to now the emphasis of the development has been mainly on correctness, though keeping into account that efficiency should finally be obtained. The work on the optimization of the domains has just started. While the generic structural domain of Chapter 4 and the groundness domain of Chapter 6 are already highly optimized, much work is still needed for the numerical domains of Chapter 5. For the latter, extensive experimentation is required. Since our actual implementation (and the theory on which it is based) has several degrees of freedom, finding the right compromise between different techniques is not easy. Similar comments apply to Chapter 7, where we need considerably more empirical comparison among different domains and among different widening policies. Even the domain-independent part of CHINA requires more work. For instance, it applies a sub-optimal version of the *magic* transformation that causes the unnecessary repetition of some analysis' work. We are currently modifying the analyzer in order to employ the more efficient *Supplementary Magic-Set* transformation described in [BR87]. More work is also necessary in the field of fixpoint computation strategies. Just one example that, furthermore, appears not to have been tackled from the theoretical point of view: while there are studies devoted to fixpoint iteration strategies that try to minimize both the analysis work and the application of unnecessary widenings [Bou93, Sch96], no one seems to have studied the strategies that might allow for the efficient application of *narrowings* [Cou96].

For future work of a more speculative nature, we would like to pursue the possibility of *programming the analysis* that the work described in chapters 3 and 5 seem to open up. We feel that this ability is a step forward towards our dream about *user-defined analysis' domains*. Here, by 'user' we mean any programmer, not just the designer of data-flow analyzers. What we have today are particular program analyzers that use a specific domain of properties. While domains for important, *program independent*, *low-level* properties such as groundness, aliasing and so forth, are well-known, very little has been done for *high-level*, *user-defined* properties. We envisage the possibility of having development environments where the program analysis component is able to deal with properties that are not known *a priori*.

For instance, Prolog programmers develop their own data-structures by freely using the first-order objects of the language. Suppose that a user defines a particular kind of structure: say, a kind of tree. Now, knowing that the representation of some tree is ground might not be enough. Perhaps one wants to know whether a certain tree is almost perfectly balanced, or some other property of the particular structure that *the user* has devised. Obviously, the analyzer cannot be endowed with all the domains for each property. The user should instead be allowed to provide the system with a description of the properties of interest. The possibility, which is exploited in our approach, of programming both the abstraction function and the abstract domain, might be the right way to go.

Any writing can be improved.
But eventually you have to put something out the door.
— DONALD E. KNUTH, lecturing on *Mathematical Writing*,
Stanford University (1987)

Bibliography

[AH83] G. Alefeld and J. Herzberger. *Introduction to Interval Computation.* Academic Press, New York, 1983.

[AH92] A. Aiba and R. Hasegawa. Constraint logic programming systems — CAL, GDCC and their constraint solvers. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'92)*, pages 113–131, Tokyo, Japan, 1992. ICOT.

[AK85] J. F. Allen and H. A. Kautz. A model of naive temporal reasoning. In J. R. Hobbs and R. Moore, editors, *Formal Theories of the Commonsense World*, pages 251–268. Ablex, Norwood, NJ, 1985.

[AK91] H. Aït-Kaci. *Warren's Abstract Machine. A Tutorial Reconstruction.* The MIT Press, 1991.

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[AMSS] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of Boolean functions for dependency analysis. To appear in *Science of Computer Programming.* A previous version of this work is available in [AMSS94b].

[AMSS94a] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Boolean functions for dependency analysis: Algebraic properties and efficient representation. In B. Le Charlier, editor, *Static Analysis: Proceedings of the First International Symposium*, number 864 in Lecture Notes in Computer Science, pages 266–280. Springer-Verlag, Berlin, 1994.

[AMSS94b] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of Boolean functions for dependency analysis. Technical Report 94/211, Dept. Computer Science, Monash University, Melbourne, 1994.

[Apt92] K. Apt, editor. *Logic Programming: Proceedings of the Joint International Conference and Symposium on Logic Programming*, MIT Press Series in Logic Programming, Washington, USA, 1992. The MIT Press.

[ASS$^+$88] A. Aiba, K. Sakai, Y. Sato, D. J. Hawley, and R. Hasegawa. The constraint logic programming language CAL. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88)*, pages 263–276, Tokyo, Japan, 1988. ICOT.

[Bag92] R. Bagnara. Interpretazione astratta di linguaggi logici con vincoli su domini finiti. M.Sc. dissertation, University of Pisa, July 1992. In Italian.

[Bag94] R. Bagnara. On the detection of implicit and redundant numeric constraints in CLP programs. In M. Alpuente, R. Barbuti, and I. Ramos, editors, *Proceedings of the "1994 Joint Conference on Declarative Programming (GULP-PRODE '94)"*, pages 312–326, Peñíscola, Spain, September 1994.

[Bag95a] R. Bagnara. Constraint systems for pattern analysis of constraint logic-based languages. In M. Alpuente and M. I. Sessa, editors, *Proceedings of the "1995 Joint Conference on Declarative Programming (GULP-PRODE '95)"*, pages 581–592, Marina di Vietri, Italy, September 1995.

[Bag95b] R. Bagnara. A unified proof for the convergence of Jacobi and Gauss-Seidel methods. *SIAM Review*, 37(1):93–97, 1995.

[Bag96a] R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. Technical Report TR-96-10, Dipartimento di Informatica, Università di Pisa, 1996.

[Bag96b] R. Bagnara. A reactive implementation of *Pos* using ROBDDs. In Kuchen and Swierstra [KS96], pages 107–121.

[Bag96c] R. Bagnara. Straight ROBDDs are not the best for *Pos*. In P. Lucio, M. Martelli, and M. Navarro, editors, *Proceedings of the "1996 Joint Conference on Declarative Programming (APPIA-GULP-PRODE '96)"*, pages 493–496, Donostia-San Sebastián, Spain, July 1996.

[Bag97] R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. To appear in *Science of Computer Programming*. The full version is available in [Bag96a]. A previous version of this work was published in [Bag95a], 1997.

[BCSZ96] R. Bagnara, M. Comini, F. Scozzari, and E. Zaffanella. The *AND*-compositionality of CLP computed answer constraints. In P. Lucio, M. Martelli, and M. Navarro, editors, *Proceedings of the "1996 Joint Conference on Declarative Programming (APPIA-GULP-PRODE '96)"*, pages 355–366, Donostia-San Sebastián, Spain, July 1996.

[BDM92] P. Bigot, S. K. Debray, and K. Marriott. Understanding finiteness analysis using abstract interpretation. In Apt [Apt92], pages 735–749.

[BGL92] R. Bagnara, R. Giacobazzi, and G. Levi. Static analysis of CLP programs over numeric domains. In M. Billaud, P. Castéran, MM. Corsini, K. Musumbu, and A. Rauzy, editors, *Actes "Workshop on Static Analysis '92"*, volume 81–82 of *Bigre*, pages 43–50, Bordeaux, September 1992. Atelier Irisa, IRISA Campus de Beaulieu. Extended abstract.

[BGL93] R. Bagnara, R. Giacobazzi, and G. Levi. An application of constraint propagation to data-flow analysis. In *Proceedings of "The Ninth Conference on Artificial Intelligence for Applications"*, pages 270–276, Orlando, Florida, March 1993. IEEE Computer Society Press, Los Alamitos, CA.

[Ble74] W. W. Bledsoe. The sup-inf method in Presburger arithmetic. Memo ATP-18, Math. Dept., University of Texas at Austin, Austin, 1974.

[BM83] R. Barbuti and A. Martelli. A structured approach to semantics correctness. *Science of Computer Programming*, 3:279–311, 1983.

[BMSU86] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 1–15, 1986.

[BMV94] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(intervals) revisited. In Bruynooghe [Bru94], pages 124–138.

[Bou92] F. Bourdoncle. Abstract interpretation by dynamic partitioning. *Journal of Functional Programming*, 2(4):407–435, 1992.

[Bou93] F. Bourdoncle. Sémantiques des langages impératifs d'ordre supérieur et interprétation abstraite. Technical Report PRL Research Report 22, DEC Paris Research Laboratory, 1993.

[BR87] C. Beeri and R. Ramakrishnan. On the power of Magic. In *Proceedings of the ACM-SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1987.

[Bru94] M. Bruynooghe, editor. *Logic Programming: Proceedings of the 1994 International Symposium*, MIT Press Series in Logic Programming, Ithaca, NY, USA, 1994. The MIT Press.

[Bry86] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[BS81] S. Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Springer-Verlag, Berlin, 1981.

[Cap93] Olga Caprotti. RISC-CLP($\mathcal{R}$-*Trees*): RISC-CLP(Real) handles symbolic functions. In A. Miola, editor, *DISCO'93: International Symposium on Design and Implementation of Symbolic Computation Systems*. Springer-Verlag, Berlin, 1993.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, 1979.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation and applications to logic programs. *Journal of Logic Programming*, 13(2&3):103–179, 1992.

[CC92b] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[CC92c] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming*, volume 631 of *Lecture*

Notes in Computer Science, pages 269–295, Leuven, Belgium, 1992. Springer-Verlag, Berlin.

[CC95] P. Cousot and R. Cousot. Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game theoretic form. In P. Wolper, editor, *Proceedings of the 7th International Conference on Computer-Aided Verification (CAV '95)*, volume 939 of *Lecture Notes in Computer Science*, pages 293–308. Springer-Verlag, Berlin, 1995. Invited paper.

[CD93] M. Codish and B. Demoen. Analysing logic programs using "Prop"-ositional logic programs and a magic wand. In D. Miller, editor, *Logic Programming: Proceedings of the 1993 International Symposium*, pages 114–129, Vancouver, Canada, 1993. The MIT Press.

[CDY90] M. Codish, D. Dams, and E. Yardeni. Bottom-up abstract interpretation of logic programs. Technical Report CS90-24, Weizmann Institute of Science, Dept of appl. maths and comp. sci., 1990.

[CDY94] M. Codish, D. Dams, and E. Yardeni. Bottom-up abstract interpretation of logic programs. *Theoretical Computer Science*, 124(1):93–125, 1994. An earlier version of this work appeared in [CDY90].

[CF92] P. Codognet and G. Filé. Computations, abstractions and constraints. In *Proceedings of the Fourth IEEE International Conference on Computer Languages*. IEEE Computer Society Press, 1992.

[CFW91] A. Cortesi, G. Filé, and W. Winsborough. *Prop* revisited: Propositional formula as abstract domain for groundness analysis. In *Proc. Sixth IEEE Symp. on Logic In Computer Science*, pages 322–327. IEEE Computer Society Press, 1991.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 84–96, 1978.

[CH93] B. Le Charlier and P. Van Hentenryck. Groundness analysis for Prolog: Implementation and evaluation of the domain *Prop*. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 99–110. ACM Press, 1993.

[CJH94] B. Carlson, S. Janson, and S. Haridi. AKL(FD): A concurrent language for FD programming. In Bruynooghe [Bru94], pages 521–535.

[CL94] C. K. Chiu and J. H. M. Lee. Towards practical interval constraint solving in logic programming. In Bruynooghe [Bru94], pages 109–123.

[Cle87] J. G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.

[CLV93] A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Conceptual and software support for abstract domain design: Generic structural domain and open product. Technical Report CS-93-13, Brown University, Providence, RI, 1993.

[CLV94] A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Combinations of abstract domains for logic programming. In *Conference Record of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 227–239, Portland, Oregon, 1994.

[Col82] A. Colmerauer. Prolog and Infinite Trees. In K. L. Clark and S. Å. Tärnlund, editors, *Logic Programming, APIC Studies in Data Processing*, volume 16, pages 231–251. Academic Press, New York, 1982.

[Col84] A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'84)*, pages 85–99, Tokyo, Japan, 1984. ICOT.

[Col90] A. Colmerauer. An introduction to Prolog-III. *Communications of the ACM*, 33(7):69–90, 1990.

[Cou78] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* Thèse d'Ètat, Université Scientifique et Médicale de Grenoble, Grenoble, France, March 1978.

[Cou96] P. Cousot. Efficient narrowing strategies. Personal communication, September 1996.

[CRR92] T. Chen, I. V. Ramakrishnan, and R. Ramesh. Multistage indexing algorithms for speeding Prolog execution. In Apt [Apt92], pages 639–653.

[Dar91] P. W. Dart. On derived dependencies and connected databases. *Journal of Logic Programming*, 11(1&2):163–188, 1991.

[Dav87] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.

[DC93] D. Diaz and P. Codognet. A minimal extension of the WAM for `clp(FD)`. In D. S. Warren, editor, *Proc. Tenth Int'l Conf. on Logic Programming*, pages 774–790. The MIT Press, 1993.

[DE73] G. B. Dantzig and B. C. Eaves. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory* (A), 14:288–297, 1973.

[Dea85] T. Dean. Temporal imagery: An approach to reasoning about time for planning and problem solving. Technical Report 433, Yale University, New Haven, CT, 1985.

[Deb89] S. K. Debray. Static inference of modes and data dependencies in logic programs. *ACM Transactions on Programming Languages and Systems*, 11(3):418–450, 1989.

[DJBC93] V. Dumortier, G. Janssens, M. Bruynooghe, and M. Codish. Freeness analysis in the presence of numerical constraints. In *Tenth International Conference on Logic Programming*, pages 100–115. The MIT Press, June 1993.

[DR94] S. K. Debray and R. Ramakrishnan. Abstract interpretation of logic programs using magic transformations. *Journal of Logic Programming*, 18(2):149–176, 1994.

[DRRS93] S. Dawson, C. R. Ramakrishnan, I. V. Ramakrishnan, and R. C. Sekar. Extracting determinacy in logic programs. In Warren [War93], pages 424–438.

[DVS$^+$88] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'88)*, pages 693–702, Tokyo, Japan, 1988. ICOT.

[EF92] I. Z. Emiris and R. J. Fateman. Towards an efficient implementation of interval arithmetic. Technical Report UCB/CSD 92/693, Computer Science Division (EECS), University of California at Berkeley, 1992.

[FGMP95] M. Falaschi, M. Gabbrielli, K. Marriott, and C. Palamidessi. Confluence and concurrent constraint programming. In V. S.

Alagar and M. Nivat, editors, *Proceedings of the Fourth International Conference on Algebraic Methodology and Software Technology (AMAST'95)*, volume 936 of *Lecture Notes in Computer Science*, pages 531–545. Springer-Verlag, Berlin, 1995.

[Fik70] R. E. Fikes. REF-ARF: A system for solving problems stated as procedures. *Artificial Intelligence*, 1:27–120, 1970.

[FR94] G. Filé and F. Ranzato. Improving abstract interpretations by systematic lifting to the powerset. In Bruynooghe [Bru94], pages 655–669.

[Fre78] E. C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21(11):958–966, 1978.

[GDL92] R. Giacobazzi, S. K. Debray, and G. Levi. A generalized semantics for constraint logic programs. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'92)*, pages 581–591, Tokyo, Japan, 1992. ICOT.

[GDL95] R. Giacobazzi, S. K. Debray, and G. Levi. Generalized semantics and abstract interpretation for constraint logic programs. *Journal of Logic Programming*, 25(3):191–247, 1995.

[GHK$^+$80] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices.* Springer-Verlag, Berlin, 1980.

[GM92] M. Gabbrielli and M. C. Meo. Fixpoint semantics for partial computed answer substitutions and call patterns. In H. Kirchner and G. Levi, editors, *Algebraic and Logic Programming, Proceedings of the Third International Conference*, volume 632 of *Lecture Notes in Computer Science*, pages 84–99, Volterra, Italy, 1992. Springer-Verlag, Berlin.

[GPR95] R. Giacobazzi, C. Palamidessi, and F. Ranzato. Weak relative pseudo-complements of closure operators. Technical Report LIX/RR/95/04, Laboratoire d'Informatique, École Polytechnique, Paris, 1995. Available on WWW at URL `http://www.di.unipi.it/~giaco`.

[GR96] R. Giacobazzi and F. Ranzato. Compositional optimization of disjunctive abstract interpretations. In H. R. Nielson, editor, *Proceedings of the 1996 European Symposium on Programming*, volume 1058 of *Lecture Notes in Computer Science*, pages 141–155. Springer-Verlag, Berlin, 1996.

[Han93] M. Hanus. Analysis of nonlinear constraints in CLP($\mathcal{R}$). In Warren [War93], pages 83–99.

[Han96] M. Handjieva. STAN: A static analyzer for CLP($\mathcal{R}$) based on abstract interpretation. In R. Cousot and D. A. Schmidt, editors, *Static Analysis: Proceedings of the 3rd International Symposium*, volume 1145 of *Lecture Notes in Computer Science*, pages 383–384, Aachen, Germany, 1996. Springer-Verlag, Berlin. System description.

[HCC95] P. Van Hentenryck, A. Cortesi, and B. Le Charlier. Evaluation of the domain PROP. *Journal of Logic Programming*, 23(3):237–278, 1995. Extended version of [CH93].

[Hic94] T. Hickey. CLP(F) and constrained ODE's. In *Proceedings of the 1994 Workshop on Constraints and Modelling*, Ithaca, USA, 1994.

[HM89] T. Hickey and S. Mudambi. Global compilation of Prolog. *Journal of Logic Programming*, 7(3):193–230, 1989.

[HMO91] R. Helm, K. Marriott, and M. Odersky. Spatial query optimization: from boolean constraints to range queries. Technical Report RC 17231, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, 1991.

[Hol95] C. Holzbaur. OFAI clp(q,r) manual, edition 1.3.3. Technical Report TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna, 1995.

[Hon92] H. Hong. Non-linear real constraints in constraint logic programming. In H. Kirchner and G. Levi, editors, *Algebraic and Logic Programming, Proceedings of the Third International Conference*, volume 632 of *Lecture Notes in Computer Science*, pages 201–212, Volterra, Italy, 1992. Springer-Verlag, Berlin.

[Hon93] H. Hong. RISC-CLP(Real): Logic programming with non-linear constraints over the reals. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research.* The MIT Press, 1993.

[HSS$^+$92] W. Havens, S. Sidebottom, G. Sidebottom, J. Jones, and R. Ovans. Echidna: a constraint logic programming shell. In *Proceedings of the 1992 Pacific Rim International Conference on Artificial Intelligence*, Seoul, 1992.

[Jan94] S. Janson. *AKL - A Multiparadigm Programming Language.* PhD thesis, Uppsala University, Sweden, June 1994. Also available in the SICS Dissertation Series: SICS/D–14–SE.

[JB92] G. Janssens and M. Bruynooghe. Deriving descriptions of possible values of program variables by means of abstract interpretation. *Journal of Logic Programming*, 13(2&3):205–258, 1992.

[JBE94] G. Janssens, M. Bruynooghe, and V. Englebert. Abstracting numerical values in CLP(H, N). In M. Hermenegildo and J. Penjam, editors, *Proc. Sixth Int'l Symp. on Programming Language Implementation and Logic Programming*, volume 844 of *Lecture Notes in Computer Science*, pages 400–414. Springer-Verlag, Berlin, 1994.

[JL87] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 111–119. ACM Press, 1987.

[JL89] D. Jacobs and A. Langen. Accurate and efficient approximation of variable aliasing in logic programs. In E. L. Lusk and R. A. Overbeek, editors, *Logic Programming: Proceedings of the North American Conference*, MIT Press Series in Logic Programming, pages 154–165, Cleveland, Ohio, USA, 1989. The MIT Press.

[JM87] J. Jaffar and S. Michaylov. Methodology and implementation of a CLP system. In J.-L. Lassez, editor, *Proceedings of the Fourth International Conference on Logic Programming*, MIT Press Series in Logic Programming, pages 196–218, Melbourne, Australia, 1987. The MIT Press.

[JM94] J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19&20:503–582, 1994.

[JMM91] N. Jørgensen, K. Marriot, and S. Michaylov. Some global compile-time optimizations for CLP($\mathcal{R}$). In Saraswat and Ueda [SU91], pages 420–434.

[JMSY92a] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. An abstract machine for CLP($\mathcal{R}$). In *Proceedings of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation*, volume 27 of *SIGPLAN Notices*, pages 128–139, San Francisco, California, 1992. Association for Computing Machinery.

[JMSY92b] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[JS87] N. D. Jones and H. Søndergaard. A semantics-based framework for the abstract interpretation of Prolog. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*, chapter 6, pages 123–142. Ellis Horwood Ltd, West Sussex, England, 1987.

[Kan90] T. Kanamori. Abstract interpretation based on Alexander templates. Technical Report TR-549, ICOT, Tokyo, Japan, 1990.

[Kan93] T. Kanamori. Abstract interpretation based on Alexander templates. *Journal of Logic Programming*, 15(1&2):31–54, 1993. An earlier version of this work appeared in [Kan90].

[KB88] R. A. Kowalski and K. A. Bowen, editors. *Logic Programming: Proceedings of the Fifth International Conference and Symposium on Logic Programming*, MIT Press Series in Logic Programming, Seattle, USA, 1988. The MIT Press.

[Kei94] T. Keisu. *Tree Constraints*. PhD thesis, The Royal Institute of Technology, Stockholm, Sweden, May 1994. Also available in the SICS Dissertation Series: SICS/D–16–SE.

[KM81] U. W. Kulisch and W. L. Miranker. *Computer Arithmetic in Theory and Practice*. Academic Press, New York, 1981.

[Knu80] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, second edition, 1980.

[KS96] H. Kuchen and S. D. Swierstra, editors. *Programming Languages: Implementations, Logics and Programs, Proceedings of the Eighth International Symposium*, volume 1140 of *Lecture Notes in Computer Science*, Aachen, Germany, 1996. Springer-Verlag, Berlin.

[Lag85] J. C. Lagarias. The computational complexity of simultaneous Diophantine approximation problems. *SIAM Journal of Computing*, 14(1):196–209, 1985.

[LCVH92] B. Le Charlier and P. Van Hentenryck. Experimental evaluation of a generic abstract interpretation algorithm for Prolog. In *Proceedings of the Fourth IEEE International Conference on Computer Languages*, pages 137–146. IEEE Computer Society Press, 1992.

[LL94] J. H. M. Lee and T. W. Lee. A WAM-based abstract machine for interval constraint logic programming and the multiple-trailing problem. In *Proceedings of the Sixth IEEE International Conference on Tools with Artificial Intelligence*, New Orleans, 1994.

[LM95] G. Levi and D. Micciancio. Analysis of pure PROLOG programs. In M. Alpuente and M. I. Sessa, editors, *Proceedings of the "1995 Joint Conference on Declarative Programming (GULP-PRODE '95)"*, pages 521–532, Marina di Vietri, Italy, September 1995.

[LMM88] J.-L. Lassez, M. J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988.

[LMY87] C. Lassez, K. McAloon, and R. Yap. Constraint logic programming and option trading. *IEEE Expert*, 2(3), 1987.

[Mac77] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mel85] C. S. Mellish. Some global optimizations for a Prolog compiler. *Journal of Logic Programming*, 2(1):43–66, 1985.

[MF85] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[Mon74] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[Moo66] R. E. Moore. *Interval Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1966.

[MS88] K. Marriott and H. Søndergaard. On describing success patterns of logic programs. Technical Report 12, The University of Melbourne, 1988.

[MS93] K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM Letters on Programming Languages and Systems*, 2(4):181–196, 1993.

[Mus90] K. Musumbu. *Interprétation Abstraite des Programmes Prolog.* PhD thesis, Facultés Universitaires Notre-Dame de la Paix – Namur Institut d'Informatique, Belgium, September 1990.

[Nil91] U. Nilsson. Abstract interpretation: A kind of Magic. In J. Małuszyński and M. Wirsing, editors, *Proceedings of the 3rd International Symposium on Programming Language Implementation and Logic Programming*, volume 528 of *Lecture Notes in Computer Science*, pages 299–310, Passau, Germany, 1991. Springer-Verlag, Berlin.

[NRC94] Academic careers for experimental computer scientists and engineers. Report of the Committee on Academic Careers for Experimental Computer Scientists, Computer Science and TeleCommunication Board, Commission on Physical Sciences, Mathematics, and Applications — U.S.A. National Research Council, 1994.

[OV93] W. Older and A. Vellino. Constraints arithmetic on real intervals. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research.* The MIT Press, 1993.

[PB94] G. Pesant and M. Boyer. QUAD-CLP($\mathcal{R}$): Adding the power of quadratic constraints. In A. Borning, editor, *Principles and Practice of Constraint Programming: Proceedings of the Second International Workshop*, volume 874 of *Lecture Notes in Computer Science*, pages 95–108, Rosario, Orcas Island, USA, 1994. Springer-Verlag, Berlin.

[Pug92] W. Pugh. A practical algorithm for exact array dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992.

[Ram88] R. Ramakrishnan. Magic Templates: A spellbinding approach to logic programs. In Kowalski and Bowen [KB88], pages 140–159.

[Ric68] D. Richardson. Some undecidable problems involving elementary functions of a real variable. *Journal of Symbolic Logic*, 33:514–520, 1968.

[RRW90] R. Ramesch, I. V. Ramakrishnan, and D. S. Warren. Automata-driven indexing of Prolog clauses. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 281–290, 1990.

[SA89] K. Sakai and A. Aiba. CAL: A theoretical background of constraint logic programming and its applications. *Journal of Symbolic Computation*, 8:589–603, 1989.

[Sar92] V. A. Saraswat. The category of constraint systems is Cartesian-closed. In *Proceedings, Seventh Annual IEEE Symposium on*

Logic in Computer Science, pages 341–345, Santa Cruz, California, 1992. IEEE Computer Society Press.

[Sar93] V. A. Saraswat. *Concurrent Constraint Programming.* The MIT Press, Cambridge, Mass., 1993.

[Sch96] E. Schön. On the computation of fixpoints in static program analysis with an application to analysis of AKL. Technical Report R95:06, Swedish Institute of Computer Science, 1996.

[Sco82] D. Scott. Domains for denotational semantics. In M. Nielsen and E. M. Schmidt, editors, *Proc. ninth Int. Coll. on Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 577–613. Springer-Verlag, Berlin, 1982.

[Scu96] M. C. Scudellari. Analisi bottom-up di programmi logici con vincoli basata su trasformazioni *Magic-Set.* M.Sc. dissertation, R. Bagnara and G. Levi supervisors, University of Pisa, April 1996. In Italian.

[Sho77] R. E. Shostak. On the SUP-INF method in for proving Presburger formulas. *Journal of the ACM*, 24(4):529–543, 1977.

[Sho81] R. E. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28(4):769–779, 1981.

[SIC95] Swedish Institute of Computer Science, Programming Systems Group. *SICStus Prolog User's Manual*, release 3 #0 edition, 1995.

[Sim83] R. Simmons. Representing and reasoning about change in geologic interpretation. Technical Report 749, MIT AI Laboratory, 1983.

[Sim86] R. Simmons. Commonsense arithmetic reasoning. In T. Kehler and S. Rosenschein, editors, *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, volume 1, pages 118–124, Philadelphia, PA, 1986. AAAI Press / The MIT Press. Distributed by Morgan Kaufmann.

[SRP91] V. A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundation of concurrent constraint programming. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 333–353. Association for Computing Machinery, 1991.

[SS80] G. J. Sussman and G. L. Steele. CONSTRAINTS: a language for expressing almost hierarchical descriptions. *Artificial Intelligence*, 14(1):1–39, 1980.

[ST84] T. Sato and H. Tamaki. Enumeration of success patterns in logic programs. *Theoretical Computer Science*, 34:227–240, 1984.

[SU91] V. Saraswat and K. Ueda, editors. *Logic Programming: Proceedings of the 1991 International Symposium*, MIT Press Series in Logic Programming, San Diego, USA, 1991. The MIT Press.

[Sut63] I. E. Sutherland. SKETCHPAD: A man-machine graphical communication system. Technical Report 296, MIT Lincoln Labs, 1963.

[Tay90] A. Taylor. LIPS on a MIPS: Results from a Prolog compiler for a RISC. In D. H. D. Warren and P. Szeredi, editors, *Logic Programming: Proceedings of the Seventh International Conference on Logic Programming*, MIT Press Series in Logic Programming, pages 174–185, Jerusalem, Israel, 1990. The MIT Press.

[VCL94] P. Van Hentenryck, A. Cortesi, and B. Le Charlier. Type analysis of Prolog using type graphs. In *Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation*, volume 29 of *SIGPLAN Notices*, pages 337–348, Orlando, Florida, 1994. Association for Computing Machinery.

[VD92] P. VanRoy and A. M. Despain. High-performance logic programming with the Aquarius Prolog compiler. *IEEE Computer*, 25(1):54, 1992.

[Ver83] S. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Trans. Patt. Anal. Mach. Intell.*, 5(3):246–267, 1983.

[Vod88a] P. Voda. The constraint language Trilogy: Semantics and computation. Technical report, Complete Logic Systems, North Vancouver, BC, Canada, 1988.

[Vod88b] P. Voda. Types of Trilogy. In Kowalski and Bowen [KB88], pages 580–589.

[VSD92a] P. Van Hentenryck, A. V. Saraswat, and Y. Deville. Constraint logic programming over finite domains: the design, implementation, and applications of `cc(fd)`. Technical report, Brown University, Providence, RI, 1992.

[VSD92b] P. Van Hentenryck, H. Simonis, and M. Dincbas. Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58:113–159, 1992.

[Wal75] D. Waltz. Understanding line drawings of scenes with shadows. In P. Winston, editor, *The Psychology of Computer Vision*, chapter 2. McGraw-Hill, New York, 1975.

[War62] S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

[War75] H. S. Warren. A modification of Warshall's algorithm for the transitive closure of binary relations. *Communications of the ACM*, 18(4):218–220, 1975.

[War83] D. H. Warren. An abstract Prolog instruction set. Technical Report Note 309, SRI International, 1983.

[War93] D. S. Warren, editor. *Logic Programming: Proceedings of the Tenth International Conference on Logic Programming*, MIT Press Series in Logic Programming, Budapest, Hungary, 1993. The MIT Press.

[WW96] G. Weyer and W. Winsborough. Annotated structure shape graphs for abstract analysis of Prolog. In Kuchen and Swierstra [KS96], pages 92–106.

[Yem79] Y. Yemini. Some theoretical aspects of position-location problems. In *Proceedings of the 20th Symposium on the Foundations of Computer Science*, pages 1–7, 1979.

[Zaf95] E. Zaffanella. Sharing correctness. Personal communication, December 1995.