View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by AMS Tesi di Dottorato

Constraint Handling Rules

Compositional Semantics and Program Transformation

Paolo Tacchella

Technical Report UBLCS-2008-04

March 2008

Department of Computer Science

University of Bologna

Mura Anteo Zamboni 7
40127 Bologna (Italy)

The University of Bologna Department of Computer Science Research Technical Reports are available in PDF and gzipped PostScript formats via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` or via WWW at URL `http://www.cs.unibo.it/`. Plain-text abstracts organized by year are available in the directory `ABSTRACTS`.

Recent Titles from the UBLCS Technical Report Series

2006-22 *Broadcasting at the Critical Threshold*, Arteconi, S., Hales, D., October 2006.

2006-23 *Emergent Social Rationality in a Peer-to-Peer System*, Marcozzi, A., Hales, D., October 2006.

2006-24 *Reconstruction of the Protein Structures from Contact Maps*, Margara, L., Vassura, M., di Lena, P., Medri, F., Fariselli, P., Casadio, R., October 2006.

2006-25 *Lambda Types on the Lambda Calculus with Abbreviations*, Guidi, F., November 2006.

2006-26 *FirmNet: The Scope of Firms and the Allocation of Task in a Knowledge-Based Economy*, Mollona, E., Marcozzi, A. November 2006.

2006-27 *Behavioral Coalition Structure Generation*, Rossi, G., November 2006.

2006-28 *On the Solution of Cooperative Games*, Rossi, G., December 2006.

2006-29 *Motifs in Evolving Cooperative Networks Look Like Protein Structure Networks*, Hales, D., Arteconi, S., December 2006.

2007-01 *Extending the Choquet Integral*, Rossi, G., January 2007.

2007-02 *Towards Cooperative, Self-Organised Replica Management*, Hales, D., Marcozzi, A., Cortese, G., February 2007.

2007-03 *A Model and an Algebra for Semi-Structured and Full-Text Queries (PhD Thesis)*, Buratti, G., March 2007.

2007-04 *Data and Behavioral Contracts for Web Services (PhD Thesis)*, Carpineti, S., March 2007.

2007-05 *Pattern-Based Segmentation of Digital Documents: Model and Implementation (PhD Thesis)*, Di Iorio, A., March 2007.

2007-06 *A Communication Infrastructure to Support Knowledge Level Agents on the Web (PhD Thesis)*, Guidi, D., March 2007.

2007-07 *Formalizing Languages for Service Oriented Computing (PhD Thesis)*, Guidi, C., March 2007.

2007-08 *Secure Gossiping Techniques and Components (PhD Thesis)*, Jesi, G., March 2007.

2007-09 *Rich Media Content Adaptation in E-Learning Systems (PhD Thesis)*, Mirri, S., March 2007.

2007-10 *User Interaction Widgets for Interactive Theorem Proving (PhD Thesis)*, Zacchiroli, S., March 2007.

2007-11 *An Ontology-based Approach to Define and Manage B2B Interoperability (PhD Thesis)*, Gessa, N., March 2007.

2007-12 *Decidable and Computational Properties of Cellular Automata (PhD Thesis)*, Di Lena, P., March 2007.

2007-13 *Patterns for descriptive documents: a formal analysis* Dattolo, A., Di Iorio, A., Duca, S., Feliziani, A. A., Vitali, F., April 2007.

2007-14 *BPM + DM = BPDM* Magnani, M., Montesi, D., May 2007.

2007-15 A study on company name matching for database integration, Magnani, M., Montesi, D., May 2007.

2007-16 *Fault Tolerance for Large Scale Protein 3D Reconstruction from Contact Maps*, Vassura, M., Margara, L., di Lena, P., Medri, F., Fariselli, P., Casadio, R., May 2007.

2007-17 *Computing the Cost of BPMN Diagrams*, Magnani, M., Montesi, D., June 2007.

2007-18 *Expressing Priorities, External Probabilities and Time in Process Algebra via Mixed Open/Closed Systems*, Bravetti, M., June 2007.

2007-19 Design and Evaluation of a Wide-Area Distributed Shared Memory Middleware, Mazzucco, M., Morgan, G., Panzieri, F., July 2007.

2007-20 *An Object-based Fault-Tolerant Distributed Shared Memory Middleware* Lodi, G., Ghini, V., Panzieri, F., Carloni, F., July 2007.

2007-21 *Templating Wiki Content for Fun and Profit*, Di Iorio, A., Vitali, F., Zacchiroli, S., August 2007.

2007-22 *EPML: Executable Process Modeling Language*, Rossi, D., Turrini, E., September 2007.

2007-23 *Stream Processing of XML Documents Made Easy with LALR(1) Parser Generators*, Padovani, L., Zacchiroli, S., September 2007.

2007-24 *On the origins of Bisimulation, Coinduction, and Fixed Points*, Sangiorgi, D., October 2007.

2007-25 *Towards a Group Selection Design Pattern*, Hales, D., Arteconi, S., Marcozzi, A., Chao, I., November 2007.

2008-01 *Modelling decision making in fund raising management by a fuzzy knowledge system*, Barzanti, L., Gaspari, M., Saletti, D., February 2008.

2008-02 *Automatic Code Generation: From Process Algebraic Architectural Descriptions to Multi-threaded Java Programs (Ph.D. Thesis)*, Bontà, E., March 2008.

2008-03 Interactive Theorem Provers: issues faced as a user and tackled as a developer (Ph.D. Thesis), Tassi, E., March 2008.

Dottorato di Ricerca in Informatica
Università di Bologna e Padova

Constraint Handling Rules

Compositional Semantics and Program Transformation

Paolo Tacchella

March 2008

Coordinatore:
Ozalp Babaöglu

Tutore:
Maurizio Gabbrielli

Abstract

This thesis intends to investigate two aspects of Constraint Handling Rules (CHR). It proposes a compositional semantics and a technique for program transformation.

CHR is a concurrent committed-choice constraint logic programming language consisting of guarded rules, which transform multi-sets of atomic formulas (constraints) into simpler ones until exhaustion [Frü06] and it belongs to the declarative languages family. It was initially designed for writing constraint solvers but it has recently also proven to be a general purpose language, being as it is Turing equivalent [SSD05a].

Compositionality is the first CHR aspect to be considered. A trace based compositional semantics for CHR was previously defined in [DGM05]. The reference operational semantics for such a compositional model was the original operational semantics for CHR which, due to the propagation rule, admits trivial non-termination.

In this thesis we extend the work of [DGM05] by introducing a more refined trace based compositional semantics which also includes the history. The use of history is a well-known technique in CHR which permits us to trace the application of propagation rules and consequently it permits trivial non-termination avoidance [Abd97, DSGdlBH04]. Naturally, the reference operational semantics, of our new compositional one, uses history to avoid trivial non-termination too.

Program transformation is the second CHR aspect to be considered, with particular regard to the unfolding technique. Said technique is an appealing approach which allows us to optimize a given program and in more detail to improve run-time efficiency or space-consumption. Essentially it consists of a sequence of syntactic program manipulations

which preserve a kind of semantic equivalence called qualified answer [Frü98], between the original program and the transformed ones. The unfolding technique is one of the basic operations which is used by most program transformation systems. It consists in the replacement of a procedure-call by its definition. In CHR every conjunction of constraints can be considered as a procedure-call, every CHR rule can be considered as a procedure and the body of said rule represents the definition of the call. While there is a large body of literature on transformation and unfolding of sequential programs, very few papers have addressed this issue for concurrent languages.

We define an unfolding rule, show its correctness and discuss some conditions in which it can be used to delete an unfolded rule while preserving the meaning of the original program. Finally, confluence and termination maintenance between the original and transformed programs are shown.

This thesis is organized in the following manner. Chapter 1 gives some general notion about CHR. Section 1.1 outlines the history of programming languages with particular attention to CHR and related languages. Then, Section 1.2 introduces CHR using examples. Section 1.3 gives some preliminaries which will be used during the thesis. Subsequentely, Section 1.4 introduces the syntax and the operational and declarative semantics for the first CHR language proposed. Finally, the methodologies to solve the problem of trivial non-termination related to propagation rules are discussed in Section 1.5.

Chapter 2 introduces a compositional semantics for CHR where the propagation rules are considered. In particular, Section 2.1 contains the definition of the semantics. Hence, Section 2.2 presents the compositionality results. Afterwards Section 2.3 expounds upon the correctness results.

Chapter 3 presents a particular program transformation known as unfolding. This transformation needs a particular syntax called annotated which is introduced in Section 3.1 and its related modified operational semantics ω_t' is presented in Section 3.2. Subsequently, Section 3.3 defines the unfolding rule and prove its correctness. Then, in Section 3.4 the problems related to the replacement of a rule by its unfolded version are

discussed and this in turn gives a correctness condition which holds for a specific class of rules. Section 3.5 proves that confluence and termination are preserved by the program modifications introduced.

Finally, Chapter 4 concludes by discussing related works and directions for future work.

Acknowledgements

First of all, I have to thank my father Gildo and my mother Maria Teresa, who gave me life and who have supported me till now. I have to thank my brother Cristian for his friendship and his companionship on the journey.

I would also like to thank Maurizio Gabbrielli for his supervision and continuous help and Maria Chiara Meo for her valuable feedback and corrections.

Furthermore, I have to thank Thom Frühwirth and his Ulm CHR research group especially Frank Raiser, Hariolf Betz, and Marc Meister for their warm welcome, interesting suggestions and precious help during my period abroad.

I have also to thank Tom Schrijvers and the Katholieke Universiteit Leuven CHR group for their welcome and suggestions.

A sincere thank you to Michael Maher and Thom Frühwirth for their detailed revisions which helped very much in the improvement of the thesis.

Heartfelt thanks to the friends who in various ways helped me during these years. In particular: Giovanni, Gabriella, Elisa, Eoghan, Laura, Liberato and Stefania from Verona, Emanuele Lindo from Vicenza, Francesco, Matelda, Aurora, Gianluca, Cesare and Piergiorgio form Bologna, Giampaolo from Rovigo and Silvia, Thomas and Jutta from Ulm (Germany).

Finally, special thanks to Rita from Cascia, Giuseppe from Copertino, Antonio from Padova, ... and last but not least Maria from Nazareth and “Deo Gratias”.

Contents

Abstract **vii**

Acknowledgements **xi**

List of Tables **xv**

List of Figures **xvii**

1 Introduction **1**

1.1 History of programming languages 1

1.1.1 The ’70s languages 2

1.1.2 The ’80s languages 4

1.1.3 The ’90s languages 5

1.2 CHR by example 7

1.3 Preliminaries 10

1.4 The original CHR 11

1.4.1 Syntax 12

1.4.2 Operational semantics 12

1.4.3 Declarative semantics 15

1.5 The propagation problem 17

1.5.1 Unlabelled constraints 17

1.5.2 Labelled constraints: ω_t semantics 20

2 Compositional semantics **25**
2.1 Compositional trace semantics . . . 27
2.2 Compositionality . . . 34
2.3 Correctness . . . 74

3 Program Transformation: Unfolding in CHR **77**
3.1 CHR annotated syntax . . . 81
3.2 The modified semantics ω_t' . . . 84
3.3 The unfolding rule . . . 90
3.4 Safe rule replacement . . . 100
3.5 Confluence and termination . . . 110

4 Related work and conclusions **115**

References **123**

List of Tables

1.1 The transition system Tx for the original semantics 13

1.2 The transition system $Tx - token$ for the CHR semantics that avoids trivial non-termination 20

1.3 The transition system T_{ω_t} for ω_t semantics 22

2.1 The transition system T for compositional semantics 28

3.1 The transition system $T_{\omega'_t}$ for ω'_t semantics 86

List of Figures

3.1 Equivalent program transformation 77

Chapter 1

Introduction

This chapter intends to introduce some common themes that will in turn be used in the rest of the thesis. In particular, CHR syntax and various CHR semantics are introduced. CHR is a general purpose [SSD05a], declarative, concurrent, committed-choice constraint logic programming language consisting of guarded rules, which transform multi-sets of atomic formulas (constraints) into simpler ones to the point of exhaustion [Frü06]. In the following sections we present the initial notion of CHR semantics, as proposed by Frühwirth in [Frü98] that is affected by the trivial non-termination problem. We will continue with an initial attempt to solve this problem, as proposed by Abdennadher in [Abd97] and finally we will introduce the notion of ω_t semantics, as proposed by Duck et al. in [DSGdlBH04] that we will then consider for the remainder of the thesis. Let's start with an historical introduction.

1.1 History of programming languages

We consider that the history of computer science was born during the second half of the 1940s when the first electronic computer appeared. The first programming language, or at least a way to program it, was born with the first computer.

The original programming, using first generation (GL1) languages, was made by turning switches and connecting cables. ENIAC, EDSAC and EDVAC are examples of computers where the programming languages were practically speaking non-existent. In fact, low level machine languages were used for programming. These languges consisted in some binary code descriptions of the operations and of the computational process of the machine itself [GM06, HP90].

The Assembly languages (second generation (GL2)) languages are the first steps in the creation of languages that are closer to human language with respect to previous (GL1) ones. These languages are a symbolic representation of the machine languages. Portability is low because every computer model has its own Assembly language [GM06].

Mainframes are general purpose batch machines which were born at the end of the 1950s. Together with these computers the first hight level languages (GL3) were born. FORTRAN (1957) conceived for scientific applications, ALGOL (1958) developed as an algorithmic family of languages, LISP (1960) projected for list manipulation, COBOL (1960) oriented to business and Simula (1962) devoted to simulation applications are examples of GL3.

1.1.1 The ’70s languages

During the ’70s the microprocessor appeared. In this period software increased its interactive needs. Languages like C (1972), Pascal (1970) and Smalltalk (1970) met these required characteristics. These languages are imperative and object ones.

Declarative languages

Programming, using an imperative paradigm, has to concern itself both with “*what*” results we are interested in and “*how*” to reach them. If a declarative paradigm is used, the programmer has to pay attention only to “*what*” result he is interested in, leaving the interpreter of the language to take care of “*how*” to reach the desired result.

There are two classes of declarative languages, namely the functional and the logical.

ML was born as a Meta Language for a semi-automatic proof system. It was developed by R. Milner and his Edinburgh group. This is a functional declarative language to which imperative characteristics were added. It has a safe static type system and inference-type mechanism.

Prolog

Whereas some ideas of logic programming can also be found in the works of K. Gödel and J. Herbrand, the first strong foundational theories were expounded upon by A. Robinson, who in the 1960s wrote the formal definition of the *resolution* algorithm. This algorithm permitts the proof of theorems of first order logic that do not give an "observable" result which can be seen as the result of the computation. Prolog was the first practical embodiment of the concept of logic programming, that considers the result task. It was conceived in the early 1970s by Alain Colmerauer and his colleagues at the University of Aix-Marseille. The key idea behind logic programming is that computation can be expressed as controlled deduction from declarative statements. Robert Kowalski of the University of Edinburgh collaborated with the Marseille group. He showed that Prolog could be understood as an implementation of the *SLD resolution* (1974). Said resolution is a restricted version of the previous resolution algorithm. SLD resolution proves a formula by explicitly computing the values of the variables that make the formula true. These variables give the computational results at the end of the deductive process. Although the field has developed considerably since those early days, Prolog remains the most fundamental and widely used logic programming language. The first implementation of Prolog was an interpretative programme, written in Fortran by members of Colmerauer's group. Although in some ways quite crude, this implementation was a milestone in several other ways. It established the viability of Prolog. It helped to disseminate the language and it laid the foundations for Prolog implementation technology. A later milestone was reached, by the DEC-10 Prolog system developed at the University of Edinburgh by D.H.D. Warren and colleagues. This system, built on the Marseille implementation technique, operates by introducing the notion of compiling Prolog into a low-level language (in this case DEC-10 machine code), as well as various important space-saving measures.

Warren later refined and abstracted the principles of the DEC-10 Prolog implementation into what is now known as the WAM (Warren Abstract Machine). The ISO standard of Prolog was defined during the 1990s. [AK99, GM06].

1.1.2 The '80s languages

During the 1980s the personal computer appeared. Apple II was probably the first one in 1978, followed in 1984 by the Macintosh PC. In 1981, IBM introduced its first PC and Lotus made the first spreadsheet. During these years languages like C++ and Ada appeared. C++ may be considered as an increment of C. In fact, object-oriented programming now becomes possible. Ada is the first real-time language but CLP (Constraint Logic Programming) had a major impact on CHR history.

Constraint Logic Programming

CLP (Constraint Logic Programming) can be considered a successor of Prolog. There are similiarities between Prolog and CLP rules. Both of them have a single constraint head and a body. The innovation due to CLP, consists mainly in the underlying constraint solver. In fact, while the initial Prolog was able to manage only Horn clauses, CLP permits the manipulation of relations for opportune domains, adding a constraint solving mechanism to classic logic programming. Colmeraurs and his Marseille group were the first to develop a constraint language in 1982 called Prolog II. It permitted the use of equations and disequations for terms (rational trees). After that, in the middle of the 1980s, Prolog III was introduced as an extension of the previous one. Generical constraint on strings, boolean and reals were then premitted. During the same period: CLP(R) with constraints on real numbers was developed at Monash University (Australia) and Jafar and Lassez defined the theoretical aspects of CLP programming. They specifically proved that all the logic languages could be seen as particular instances of CLP. Furthermore, they inherited all the main results of logic programming. Finally Dincbas, Van Hentenryck and others at ECRC defined CHIP, a Prolog extension that permitted various kinds of constraints and in particular constraints on finite domains.

Concurrent Constraint Logic Programming

CCP (Concurrent Constraint Logic Programming) was introduced by V. Saraswat [Sar93] based on ideas of M. J. Maher [Mah87]. It can be seen as an extension of CLP where dynamic scheduling is added. This allows for “processes” or “agents” which communicate through the global constraint store. In particular, user-defined constraints are viewed as processes and a state is regarded as a network of processes linked through shared variables by means of the store. Processes communicate by adding constraints to the store and synchronize by waiting for the store to enable a delay condition. Said condition is called a guard which must be enabled for the rule to be used. There are two kind of condition: the $tell(C)$ and the $ask(C)$ one. The former enables the rule if the constraint C is consistent with the global store while the latter enables the rule when the constraint store implies the constraint C. CLP employs “don't know non-determinism”, which tries every rule until an answer is found. CCP employ the “don't care non-determinism” which applies whatever rule for which the guard is satisfied, without backtracking. A CCP rule is composed of a head, a body like CLP and, unlike CLP itself, by a guard.

1.1.3 The ’90s languages

During the 1990s CHR together with web-oriented languages like HTML and Java appeared.

CHR

CHR can be seen as a successor to CCP. In fact, CHR and CCP are really similiar to each other because both of them are concurrent. Both their rules contain a guard, a body and a head. The main difference consists in the shape of the head because the CHR head can be made up of a conjunction of constraints unlike the one constraint CLP head.

A difference from CLP is the way in which constraints in the goal are rewritten. The unification algorithm is used to perform the binding operation between a CLP rule head and the constraints of a goal while matching substitution is used when CHR is considered to perform the same operation. Matching substitution can be considered as a sort of

simplification of the unification algorithm: a head constraint like $5 =< X$ can unify but can not match with a constraint like $X =< Y$ in the goal.

The first works on CHR appeared in 1992 when T. Früwirth, S. Abdennadher and H. Meuss wrote the first paper, where CHR without the propagation rule was considered [AFM99]. The following year the propagation rule was added. The first survey about CHR was written by T. Früwirth in 1998 [Frü98]. The first solution to trivial non-termination was proposed by S. Abdennadher in 1997 [Abd97] and a solution closer to practical implementation to the same problem was proposed by Duck et al. in 2004 [DSGdlBH04].

Meanwhile, the general interest shown in CHR increased both from the practical and the theoretical side and today various languages like C, Haskell, Java and Curry support CHR implementations.

Prolog was the first language in which CHR was embedded because of the above mentioned similarities [HF00]. Today, many versions of CHR compilers or interpreters for Prolog or logic languages are made. For example, the CHR implementation for HAL [GdlBDMS02], ToyCHR for Sicstus and SWI-Prolog by G. J. Duck in 2004 and the Leuven CHR [Sch05] for SWI-Prolog.

The first Java interpreter for CHR was developed by Abdennadher et al in 2000 and was called JACK (JAva Constraint Kit) [AKSS02]. Soon afterwards two other implementations were proposed namely K.U.Leuven JCHR [VWSD05] by Van Weert et al. and CHORD (CHR with Disjunction) by Vitorino et al. in 2005.

A first implementation of CHR in Haskell was done by G. J. Duck in 2005 and called HaskellCHR. It contains an implementaion of WAM (Warren Abstract Machine) for Haskell. Also a concurrent version of CHR was developed in Haskell by Lam and Sulzmann [SL07].

CHR was also recentely added to Curry by M. Hanus [Han06] and last but by no means least, a fast CHR version for C by Wuille et al. appeared in 2007 [WSD07].

1.2 CHR by example

In this section we will introduce CHR using some examples. A slight variant on the first example is also proposed by the CHR website http://www.cs.kuleuven.ac.be/~dtai/projects/CHR/ as an introduction to CHR for beginners.

Example 1.1 The following CHR program [Frü98] encodes the less-than-or-equal-to constraint, assuming the $=$ predicate as a given built-in constraint

$$\begin{array}{lcll} rfl & @ & X =< Y \Leftrightarrow X = Y | \texttt{true}. & \text{reflexivity} \\ asx & @ & X =< Y, Y =< X \Leftrightarrow X = Y. & \text{antisymmetry} \\ trs & @ & X =< Y, Y =< Z \Rightarrow X =< Z. & \text{transitivity} \\ idp & @ & X =< Y \backslash X =< Y \Leftrightarrow \texttt{true}. & \text{idempotence} \end{array}$$

□

The program of Example 1.1 is made up of four CHR rules. Every rule is made up of an unequivocal *name*, that is written to the left of the symbol "@", and that is usually used to refer to the rule, a *head* that is a conjunction of CHR constraints between the "@" and the "⇒" or "⇔" symbol, or a conjunction of CHR constraints between the "@" and the "⇔" symbol where the symbol "\" is added between the head constraints, an optional *guard* that is contained between the symbols "⇒" or "⇔" and the symbol "|" and finally a *body* that represents the rest of the rule.

It can clearly be observed that there are three kinds of rules: the first two rules (rfl and asx) are called *simplification* rules. In fact, the constraints to which the rules are applied, are simplified with the usually more simple constraint in their body. This kind of rule presents the symbol $\Leftrightarrow$ and no $\backslash$ symbol in the head. The third rule (trs) is called a *propagation* rule because it propagates the meaning of the constraints contained in a state adding the constraints of its body that can be useful to perform the following computational steps. The fourth rule (idp) is called *simpagation* because its behaviour is a combination of simplification and propagation.

Let us now consider the intuitive meaning of each rule: as follows, CHR specifies how $=<$ simplifies, propagates and simpagates as a constraint.

The rfl rule represents the reflexivity relation: $X =< Y$ is logically `true` if it is the case that $X = Y$, which means the guard is satisfied.

The asx rule represents the antisymmetry relation: if we know that both $X =< Y$ and $Y =< X$ then we can replace the previous constraints with the logically equivalent $X = Y$. In this instance no test condition is required.

The trs rule represents the transitive relation: if we know both $X =< Y$ and $Y =< Z$ then we can add a redundant constraint $X =< Z$ as a logical consequence.

The idp rule represents the idempotence relation: if there are two constraints $X =< Y$ one of them can be deleted, without changing the meaning.

Let us now consider a practical case where the previous program is applied to the conjunction of constraints $A =< B, A =< B, B =< C, B =< C, C =< A$. The selection of rules introduced in Example 1.2 is a redundant one. Said selection was chosen to show to the reader all the kinds of CHR rules. A more compact example, which computes less-than-or-equal-to, can be found in [Frü98]. The CHR constraints $B =< C$ and $A =< B$ are introduced twice in the considered goal to permit the application of each rule of the proposed Example 1.2 at least once. The reader can verify that the same result can be obtained also if a single copy of the previous constraints is considered.

$\mathbf{A =< B, A =< B, B =< C, B =< C, C =< A}$

$\%A =< B, A =< B$ simpagates in $A =< B$ by idp;

$\mathbf{A =< B, B =< C, B =< C, C =< A}$

$\%A =< B, C =< A$ propagates in $C =< B$ by trs;

$\mathbf{A =< B, B =< C, B =< C, C =< A, C =< B}$

$\%B =< C, C =< B$ simplify in $C = B$ by asx

$\mathbf{A =< B, B =< C, C =< A, C = B}$

$\%B =< C$ simplify in `true` by rfl considered $C = B$

$\mathbf{A =< B, C =< A, C = B}$

$\%A =< B, C =< A$ simplifies in $A = B$ by asx considered $C = B$

$\mathbf{A = B, B = C}$

%is the solution of the computation.

We would like to point out that redundancy as given by CHR propagation, is useful. Otherwise, the second computational step would not be performed and the results would not be achieved. Note also that multiple heads of rules are essential in solving these constraints.

Now a second CHR program is presented that computes the greatest common divisor following the Euclidean algorithm. Said program is introduced to show that, using CHR, extremely compact programs can be written. This program has been published in the above mentioned website.

Example 1.2 This CHR program is made up of a mere two CHR rules. The first one is called *clean-up* and it allows for the deletion of gcd(0) CHR constraints. The second rule, called *compute*, makes the actual computation.

$$\begin{array}{lll} clean-up & @ & gcd(0) \Leftrightarrow true. \\ compute & @ & gcd(N)\backslash gcd(M) \Leftrightarrow 0 < N, N =< M | L\ is\ M\ mod\ N, gcd(L). \end{array}$$

where mod represents the remainder of the integer division between M and N while is represents the assignment of a value to a variable. □

Let us consider an application of Example 1.2 to the whole to numbers 12, 24 and 72.

$\mathbf{gcd(12), gcd(24), gcd(72)}$

$\%gcd(12), gcd(24)$ simpagates as $gcd(0)$ via $compute,$

$\mathbf{gcd(12), gcd(72), gcd(0)}$

$\%gcd(0)$ simplifies as `true` via $clean,$

$\mathbf{gcd(12), gcd(72)}$

$\%gcd(12), gcd(72)$ simpagates as $gcd(6)$ via $compute,$

$\mathbf{gcd(12), gcd(6)}$

$\%gcd(12), gcd(6)$ simpagates as $gcd(0)$ via $compute,$

$\mathbf{gcd(0), gcd(6)}$

$\%gcd(0)$ simplifies as `true` via $clean,$

$\mathbf{gcd(6)}$

$\%$ is the result.

1.3 Preliminaries

In this section we will introduce some notations and definitions which we will need throughout the thesis. Even though we try to provide a self-contained exposition, some familiarity with constraint logic languages and first order logic could be useful (see for example [JM94]). CHR uses two kinds of constraints: the built-in and the CHR ones, also called user-defined.

According to the usual CHR syntax, we assume that a user-defined constraint is a conjunction of atomic user-defined constraints.

On the other hand, built-in constraints are defined by $c ::= d|c \wedge c|\exists_x c$, where d is an atomic formula or atom.

These constraints are handled by an existing solver and we assume that they contain $true$, $false$ (with the obvious meaning) and the equality symbol $=$. The meaning of these constraints is described by a given, first order theory CT which includes the following CET (Clark Equational Theory) in order to describe the $=$ symbol.

Moreover, CET (Clark Equational Theory) is considered for terms and atoms manipulation:

Reflexivity $(\top \rightarrow X = X)$

Symmetry $(X = Y \rightarrow Y = X)$

Transitivity $(X = Y \wedge Y = Z \rightarrow X = Z)$

Compatibility $(X_1 = Y_1 \wedge \ldots \wedge X_n = Y_n \rightarrow f(X_1, \ldots, X_n) = f(Y_1, \ldots, Y_n))$

Decomposition $(f(X_1, \ldots, X_n) = f(Y_1, \ldots, Y_n) \rightarrow X_1 = Y_1 \wedge \ldots \wedge X_n = Y_n)$

Contradiction $(f(X_1, \ldots, X_n) = g(Y_1, \ldots, Y_m) \rightarrow \bot)$ if $f \neq g$ or $n \neq m$

Acyclicity $(X = t \rightarrow \bot)$ if t is function term and X appears in t

If $H = h_1, \ldots, h_k$ and $H' = h'_1, \ldots, h'_k$ are sequences of CHR constraints, the notation $H = H'$ represents the set of equalities $h_1 = h'_1, \ldots, h_k = h'_k$.

$Fv(\phi)$ denotes the free variables appearing in ϕ. The notation $\exists_{-V}\phi$, where V is a set of variables, denotes the existential closure of a formula ϕ with the exception of the variables in V which remain unquantified.

If it is not specified differently, we use c, d to denote built-in constraints, h, k, s, p, q to denote CHR constraints and a, b, g, f to denote both built-in and user-defined constraints, which together are known as constraints. A CHR constraint h can also be labelled with an unequivocal identifier $h\#i$. We will also use the functions *chr($h\#i$)=h* and the function *id($h\#i$)=i*. These functions will be also extended to sets and sequences of identified CHR constraints in the obvious way. If it is not differently specified, the capital versions will be used to denote multi-sets (or sequences) of constraints. Given a goal G, the notational convention $\tilde{G}$ represents sets of identified constraints. In particular, $\tilde{G}$ depicts every possible labelling of the CHR constraints in multi-set G; consequently $G = chr(\tilde{G})$.

We will often use "," rather than $\wedge$ to denote conjunction and we will often consider a conjunction of atomic constraints as a multi-set of atomic constraints.

We denote the concatenation of sequences by $\cdot$ and the set difference operator by $\setminus$. Then, $[n, m]$ with $n, m \in \mathbb{N}$ represents the set of all the natural numbers between n and m (n and m are included). Subsequently, we omit the guard when it is the `true` constraint. Furthermore, we denote by $\mathcal{U}$ the set of user-defined constraints. Finally, multi-set union is represented by the symbol $\uplus$.

1.4 The original CHR

As shown by the following subsection, a *CHR program* consists of a set of rules which can be divided into three types: *simplification*, *propagation* and *simpagation* rules. The first kind of rules is used to rewrite CHR constraints into simpler ones, while the second one allows us to add new redundant constraints which may cause further simplification. Simpagation rules allow us to represent both simplification and propagation rules.

In this section the syntax and the semantics (the operational and the declarative ones), as proposed in [Frü98] are introduced.

1.4.1 Syntax

A CHR program [Frü98] is a finite set of CHR rules. There are three kinds of CHR rules: A **simplification** rule has the following form:

$$r@H \Leftrightarrow D \,|\, B$$

A **propagation** rule has the following form:

$$r@H \Rightarrow D \,|\, B$$

A **simpagation** rule has the following form:

$$r@H_1 \setminus H_2 \Leftrightarrow D \,|\, B,$$

where r is a unique identifier of the rule, H, H_1 and H_2 are sequences of user-defined constraints called *heads*, D is a multi-set (or a sequence) of built-in constraints called *guard* and B is a multi-set (or a sequence) of (built-in and user-defined) constraints called *body*. Both B and D could be empty. A *CHR goal* is a multi-set of (both user-defined and built-in) constraints.

1.4.2 Operational semantics

State

Given a goal G, a multi-set of CHR constraints S and a multi-set of built-in constraints C, and a set of variables $\nu = Fv(G)$, a *state* or *configuration* is represented by $Conf$ and it has the form

$$\langle G, S, C \rangle_\nu.$$

The *initial configuration* has the form

$$\langle G, \emptyset, \mathtt{true} \rangle_\nu.$$

Solve	$\dfrac{CT \models c \wedge C \leftrightarrow C' \text{ and c is a built-in constraint}}{\langle \{c\} \uplus G, S, C\rangle_\nu \longrightarrow \langle G, S, C'\rangle_\nu}$
Introduce	$\dfrac{\text{h is a user-defined (or CHR) constraint}}{\langle \{h\} \uplus G, S, C\rangle_\nu \longrightarrow \langle G, \{h\} \uplus S, C\rangle_\nu}$
Simplify	$\dfrac{r@H_2' \Leftrightarrow D \,\vert\, B \in P \quad x = Fv(H_2') \quad CT \models C \rightarrow \exists_x((H_2 = H_2') \wedge D)}{\langle G, H_2 \uplus S, C\rangle_\nu \longrightarrow \langle B \uplus G, S, (H_2 = H_2') \wedge C, \rangle_\nu}$
Propagate	$\dfrac{r@H_1' \Rightarrow D \,\vert\, B \in P \quad x = Fv(H_1') \quad CT \models C \rightarrow \exists_x((H_1 = H_1') \wedge D)}{\langle G, H_1 \uplus S, C\rangle_\nu \longrightarrow \langle B \uplus G, H_1 \uplus S, (H_1 = H_1') \wedge C\rangle_\nu}$
Simpagate	$\dfrac{r@H_1' \setminus H_2' \Leftrightarrow D \,\vert\, B \in P \quad x = Fv(H_1', H_2') \quad CT \models C \rightarrow \exists_x(H_1, H_2 = H_1', H_2')) \wedge D)}{\langle G, H_1 \uplus H_2 \uplus S, C\rangle_\nu \longrightarrow \langle B \uplus G, H_1 \uplus S, (H_1, H_2 = H_1', H_2') \wedge C\rangle_\nu}$

Table 1.1: The transition system Tx for the original semantics

A *final configuration* has either the form

$$\langle G, S, \texttt{false}\rangle_\nu$$

when it has *failed* or it has the form

$$\langle G, S, C\rangle_\nu$$

when it represents a successful termination, since there are no more applicable rules.

The transition system

Given a program P, the transition relation $\longrightarrow \subseteq \mathit{Conf} \times \mathit{Conf}$ is the least relation which satisfies the rules in Table 1.1 and for the sake of simplicity, we omit indexing the relation with the name of the program.

The **Solve** transition allows us to update the (built-in) constraint store by taking into account a built-in constraint c, contained in the goal. The built-in constraint is moved from the goal to the built-in constraint store.

The **Introduce** transition is used to move a user-defined constraint h from the goal to the CHR constraint store. After this operation, h can be handled by CHR rules.

The **Simplify** transition rewrites user-defined constraints in CHR store, using the simplification rules of program P. All variables of the considered program rule are renamed separately with fresh ones, if needed, in order to avoid variable name clashes before the application of the rule. Simplify transition can work if the current built-in constraint store (C) is strong enough to entail the guard of the rule (D), once the parameter passing (matching substitution) has been performed (this is expressed by the equation $(H_2 = H'_2)$). Note that, due to the existential quantification over the variables x appearing in H'_2, in such a parameter passing, the information flow is from the actual parameter (in H_2) to the formal parameters (H'_2), that is, it is required that the constraints H_2 which have to be rewritten are an instance (which means that $H_2 = H'_2\theta$ and θ is the matching substitution) of the head H'_2. The transition adds the body B of the rule to the current goal, the equation $(H_2 = H'_2)$ to the built-in constraint store and it removes the constraints H_2.

The **Propagate** transition matches user-defined constraints in the CHR store, using the propagation rules of program P. All variables of the program clause (rule) considered are renamed separately with fresh ones, if needed, in order to avoid variable name clashes before the application of the rule. Propagate transition can work if the current built-in constraint store (C) is strong enough to entail the guard of the rule (D), once the parameter passing (matching substitution) has been performed (this is expressed by the equation $(H_1 = H'_1)$). Note that, due to the existential quantification over the variables x appearing in H'_1, in such a parameter passing the information flow is from the actual parameter (in H_1) to the formal parameters (H'_1), that is, it is required that the constraints H_1 which have to be rewritten are an instance (which means that $H_1 = H'_1\theta$ and θ is the matching substitution) of the head H'_1. The transition adds the body B of the rule to the current goal and the equation $(H_1 = H'_1)$ to the built-in constraint store.

The **Simpagate** transition rewrites user-defined constraints in CHR store using the simpagation rules of program P. All variables of the considered program clause are renamed separately with fresh ones, if needed, in order to avoid variable name clashes before the application of the rule. Simpagate transition can work if the current built-in constraint store (C) is strong enough to entail the guard of the rule (D), once the parameter passing (matching substitution) has been performed. This is expressed by the

equation $(H_1, H_2 = H'_1, H'_2)$. Note that, due to the existential quantification over the variables x appearing in H'_1, H'_2, in such a parameter passing the information flow is from the actual parameter (in H_1, H_2) to the formal parameters (H'_1, H'_2), that is, it is required that the constraints H_1, H_2 which have to be rewritten are an instance (which means that $(H_1, H_2) = (H'_1, H'_2)\theta$ and θ is the matching substitution) of the head H'_1, H'_2. The transition adds the body B of the rule to the current goal, the equation $(H_1, H_2 = H'_1, H'_2)$ to the built-in constraint store and it removes the constraints H_2.

We can now point out that the transition system of Table 1.1 can be simplified. The Simpagate rule can simulate both the Simplify and Propagate ones. In fact, the behaviour of:

the Simplify rule	$H'_1 \Rightarrow D \mid B$	is equivalent to	$H'_1 \backslash \emptyset \Leftrightarrow D \mid B$,
the Propagate rule	$H'_2 \Leftrightarrow D \mid B$	is equivalent to	$\emptyset \backslash H'_2 \Leftrightarrow D \mid B$.

1.4.3 Declarative semantics

CHR is concerned with defining constraints and not procedures in their generality and a declarative semantics may be attributed to it.

The logical reading of a CHR program P [Frü98, FA03] is the conjunction of the logical readings of its rules, that is called $\mathcal{P}$, with a *constraint theory* CT.

The CT theory determines the meaning of the built-in constraint symbols appearing in the program and it is expected to include at least an equality constraint $=$ and the basic constraints `true` and `false`.

The logical reading $\mathcal{P}$ of P rules is given by a conjunction of universally quantified logical formulae (one for each rule).

Definition 1.1 *A CHR Simplify rule $H \Leftrightarrow D \mid B$ is a logical equivalence if the guard is satisfied:*

$$\forall \bar{x} \forall \bar{y}((\{d_1\} \uplus \ldots \uplus \{d_j\}) \rightarrow ((h_1, \ldots, h_i) \leftrightarrow \exists \bar{z}(\{b_1\} \uplus \ldots \uplus \{b_k\})))$$

A CHR Propagate rule $H \Rightarrow D \mid B$ is an implication if the guard is satisfied:

$$\forall \bar{x} \forall \bar{y}((\{d_1\} \uplus \ldots \uplus \{d_j\}) \rightarrow ((h_1, \ldots, h_i) \rightarrow \exists \bar{z}(\{b_1\} \uplus \ldots \uplus \{b_k\})))$$

A CHR Simpagate rule $H_1 \setminus H_2 \Leftrightarrow D \mid B$ is a logical equivalence if the guard is satisfied:

$$\forall\bar{x}\forall\bar{y}((\{d_1\}\uplus\ldots\uplus\{d_j\}) \rightarrow ((h_1,\ldots,h_i) \leftrightarrow \exists\bar{z}(\{h_1\}\uplus\ldots\uplus\{h_l\}\uplus\{b_1\}\uplus\ldots\uplus\{b_k\})))$$

where $D = \{d_1\}\uplus\ldots\uplus\{d_j\}$, $H = h_1,\ldots,h_i$, $H_1 = h_1,\ldots,h_l$,$H_2 = h_{l+1},\ldots,h_i$ and $B = \{b_1\}\uplus\ldots\uplus\{b_k\}$ with $\bar{x} = Fv(H)$ (global variables) and $\bar{y} = (Fv(D)\setminus Fv(H))$, $\bar{z} = (Fv(B) \setminus Fv(H))$ (local variables).

The following example considers the previously introduced ones in Section 1.2 and gives the declarative semantics of some CHR rules.

Example 1.3 The CHR simplification rule that encodes the reflexivity relation in Example 1.1

$$rfl@X =< Y \Leftrightarrow X = Y|\mathtt{true}.$$

has the logical reading

$$\forall X, Y(\{X = Y\} \rightarrow (X =< Y \leftrightarrow \mathtt{true})).$$

The CHR propagation rule that encodes the transitivity relation in Example 1.1

$$trs@X =< Y, Y =< Z \Rightarrow X =< Z.$$

has the logical reading

$$\forall X, Y, Z(\mathtt{true} \rightarrow (X =< Y, Y =< Z \rightarrow \{X =< Z\})).$$

The CHR simpagation rule that encodes the computation in Example 1.2

$$compute@gcd(N)\setminus gcd(M) \Leftrightarrow 0 < N, N =< M|L\ is\ M\ mod\ N, gcd(L).$$

has the logical reading

$$\forall N, M(\{0 < N, N =< M\} \rightarrow (gcd(N), gcd(M) \leftrightarrow \exists L\{L\ is\ M\ mod\ N, gcd(L), gcd(N)\})).$$

where CT defines the meaning of $=, <, \mathtt{true}, \mathtt{false}$. □

1.5 The propagation problem

It is of no relevance to the previously proposed semantics in Subsection 1.4.2 that a propagation rule can be applied ad infinitum to a conjunction of constraint H. This can happen until such time as a simplification rule eventually deletes some constaint in H. This problem, known also as trivial non-termination, was solved for the first time in [Abd97], where a multi-set called token store was introduced in the state and subsequently in [DSGdlBH04]. The following subsections present these two approaches.

1.5.1 Unlabelled constraints

The solution to the trivial non-termination problem, as proposed in [Abd97], is an elegant theoretical solution. Proposed semantics needs an update of the token store every time a new user-defined constraint is introduced into the CHR store. Another token store update is performed every time a CHR transition works. The last kind of updates are performed by a normalization function $\mathcal{N}$. Naturally, the application of a propagation rule deletes the associated token.

State

Given a goal G, a multi-set of CHR constraints S and a multi-set of built-in constraint C, a multi-set of tokens T, and a set of variables $\nu = Fv(G)$, a *state* or *configuration* is represented by $Conf_t$ and it has the form

$$\langle G, S, C, T\rangle_\nu .$$

An *initial configuration* has the form

$$\langle G, \emptyset, \texttt{true}, \emptyset\rangle_\nu .$$

A *final configuration* has either the form

$$\langle G, S, \texttt{false}, T\rangle_\nu$$

when it has *failed* or it has the form

$$\langle G, S, C, T\rangle_\nu$$

when it represents a successful termination, as obviously there are no more applicable rules.

The token store

The token store is a set of tokens. A token is made up of the name r of the propagation rule that can be applied, an "@" symbol and the conjunction of constraints in the CHR store to which r can be applied.

Let P be a CHR program, S the current CHR constraint store and h a user-defined constraint. $T_{(h,S)}$ adds to the current token store all the new tokens that can be generated, having taken into consideration the introduction of h in S. Naturally, the multiplicity of the constraints in S is also considered.

$$\begin{aligned} T_{(h,S)} \quad = \quad \{r@H' \quad | \quad & (r@H \Rightarrow D \mid B) \in P, H' \subseteq \{h\} \uplus S, h \in H', \\ & \text{and } H \text{ matches with } H'\}. \end{aligned}$$

When the body of a rule contains more than one constraint, the token store is managed in the following way

$$T_{(h_1,\ldots,h_m,S)} = T_{(h_1,S)} \uplus T_{(h_2,\{h_1\}\uplus S)} \uplus \ldots \uplus T_{(h_m,\{h_1,h_2,\ldots,h_{m-1}\}\uplus S)}$$

Operational semantics

The transition system, which uses the previously described token store, is shown in Table 1.2. The behaviours of the considered transition system are similar to the ones introduced in Subsection 1.4.2, so only the differences will be discussed.

After every transition, a normalization function $\mathcal{N} : State \times State$ is applied. $State$ represents the set of all states. Below, the normalization function $\mathcal{N}$ will be first of all introduced using examples and, after that, giving the formal representation. The application of the $\mathcal{N}$ function to a $State$ will be formally represented by $\mathcal{N}(\langle G, S, C, T\rangle_\nu) = \langle G', S', C', T'\rangle_\nu$.

1. The normalization function deletes the unuseful tokens, that are the ones for which at least a constraint, with which they are associated, has been deleted from the CHR store e.g. $\mathcal{N}(\langle G, g(X), B, \{r@f(W), r'@g(X)\}\rangle) = \langle G, g(X), B, \underline{\{r'@g(X)\}}\rangle_\nu$. This operation is called *token elimination* and it is formally represented by $T' = T \cap T_{(S,State)}$.

2. After having fixed a propagation order on the ν variables and considering the introduction order for the others, it propagates the equality of variables. For example $\mathcal{N}(\langle G, g(X), \{X = Y, Y = Z\}, T\rangle_{\{X,W\}} = (\langle G, \underline{g(Z)}, \underline{\{X = Z, Y = Z\}}, T\rangle_{\{X,W\}}$. This operation is called *equality propagation*. In fact, G', S' and T' derive from G, S and T by replacing all variables X, for which $CT \models \forall(C \rightarrow X = t)$ holds, by the corresponding term t, except if t is a variable that comes after X in the variable order.

3. It projects the useless built-in constraints. For example $(\langle G, H, \underline{\{X = Z, Y = Z, Z = Z\}}, T\rangle_\nu = (\langle G, H, \underline{\{X = Z, Y = Z\}}, T\rangle_\nu$. This operation is called *projection* and it is formally represented by $CT \models \forall((\exists_X C) \leftrightarrow C')$, where X is a variable which appears in C only.

4. Finally, it only unifies the variables that appear in the built-in constraint e.g. if we consider that the following states $(\langle h(W), g(X), \underline{\{X = Z, M = Z\}}, T\rangle_{W,X}$ and $(\langle f(W), m(X), \underline{\{X = Z, N = Z\}}, T\rangle_{W,X}$ are such that $CT \models \exists_M\{X = Z, M = Z\} \leftrightarrow \exists_N\{X = Z, N = Z\}$, then their built-in constraint stores after the application of $\mathcal{N}$ will both be equal to $\underline{\{X = Z, V_0 = Z\}}$. This operation is called *uniqueness* and it is formally represented by $C_1' = C_2'$ where

$$\mathcal{N}(\langle G_1, S_1, C_1, T_1\rangle_\nu) = \langle G_1', S_1', C_1', T_1'\rangle_\nu \text{ and}$$
$$\mathcal{N}(\langle G_2, S_2, C_2, T_2\rangle_\nu) = \langle G_2', S_2', C_2', T_2'\rangle_\nu \text{ and}$$
$$CT \models (\exists X C_1) \leftrightarrow (\exists Y C_2) \text{ and}$$

X and Y are variables which appear only in C_1 and C_2 respectively.

The previous examples underlined the multi-sets to which we already referred to specifically.

Solve	$\dfrac{CT \models c \wedge C \leftrightarrow C' \text{ and c is a built-in constraint}}{\langle\{c\} \uplus G, S, C, T\rangle_\nu \longrightarrow \mathcal{N}(\langle G, S, C', T\rangle_\nu)}$
Introduce	$\dfrac{\text{h is a user-defined (or CHR) constraint}}{\langle\{h\} \uplus G, S, C\rangle_\nu \longrightarrow \mathcal{N}(\langle G, \{h\} \uplus S, C, T \uplus T_{(h,S)}\rangle_\nu)}$
Simplify	$\dfrac{r@H_2' \Leftrightarrow D \mid B \in P \quad x = Fv(H_2') \quad CT \models C \rightarrow \exists_x((H_2 = H_2') \wedge D)}{\langle G, H_2 \uplus S, C, T\rangle_\nu \longrightarrow \mathcal{N}(\langle B \uplus G, S, (H_2 = H_2') \wedge C, T\rangle_\nu)}$
Propagate	$\dfrac{r@H_1' \Rightarrow D \mid B \in P \quad x = Fv(H_1') \quad CT \models C \rightarrow \exists_x((H_1 = H_1') \wedge D)}{\langle G, H_1 \uplus S, C, T \uplus \{r@H_1\}\rangle_\nu \longrightarrow \mathcal{N}(\langle B \uplus G, H_1 \uplus S, (H_1 = H_1') \wedge C, T\rangle_\nu)}$
Simpagate	$\dfrac{r@H_1' \setminus H_2' \Leftrightarrow D \mid B \in P \quad x = Fv(H_1', H_2') \quad CT \models C \rightarrow \exists_x(H_1, H_2 = H_1', H_2')) \wedge D)}{\langle G, H_1 \uplus H_2 \uplus S, C, T\rangle_\nu \longrightarrow \mathcal{N}(\langle B \uplus G, H_1 \uplus S, (H_1, H_2 = H_1', H_2') \wedge C, T\rangle_\nu)}$

Table 1.2: The transition system $Tx - token$ for the CHR semantics that avoids trivial non-termination

The $T_{(h,S)}$ relation of Introduce transition adds to the current token store all the possible tokens that can be used, considering the current CHR store S and all the rules of program P.

The right hand side of the rule considered in Simpagate transition is supposed to be not empty, that is $H_2' \neq \emptyset$.

1.5.2 Labelled constraints: ω_t semantics

A more practical solution was proposed in [DSGdlBH04]. In fact, the computation of the new token store every time that a transition happens and twice when the considered transition is an Introduce, decreases the performance. This second approach was more successful in research terms than the previous one. In the following part of this thesis, this second approach is going to be considered. For the sake of simplicity, we omit indexing the relation with the name of the program. First of all, we introduce the new state and the the shape of the new elements of the token store.

State

Given a goal G, a set of CHR constraints S (and its identified version $\tilde{S}$), a set of built-in constraints C, a set of tokens T and a natural number n, a state is represented by $Conf_t$ and it has the form

$$\langle G, \tilde{S}, C, T \rangle_n .$$

Given a goal G, the *initial configuration* has the form

$$\langle G, \emptyset, \texttt{true}, \emptyset \rangle_1 .$$

A *final configuration* has either the form

$$\langle G, \tilde{S}, \texttt{false}, T \rangle_n$$

when it has *failed* or it has the form

$$\langle \emptyset, \tilde{S}, C, T \rangle_n$$

when it represents a successful termination, as there are no more applicable rules.

The token store

Now, the tokens have a different shape from the ones of the previous Subsection 1.5.1. In fact, $r@i_1, \ldots, i_m$ is the new shape where r represents, as in the previous subsection, the name of a rule, but now the identifiers $i_1, \ldots, i_m$ replace the constraints.

The operational semantics

Given a program P, the transition relation $\longrightarrow_{\omega_t} \subseteq Conf_t \times Conf_t$ is the least relation satisfying the rules in Table 1.3. For the sake of simplicity, we omit indexing the relation with the program name.

The **Solve**$_{\omega_t}$ transition allows us to update the (built-in) constraint store, by taking into account a built-in constraint contained in the goal. It moves a built-in constraint from the store to the built-in constraint store. Without the loss of generality, we will assume that $Fv(C') \subseteq Fv(c) \cup Fv(C)$.

Solve$_{\omega_t}$	$$\frac{CT \models c \wedge C \leftrightarrow C' \text{ and c is a built-in constraint}}{\langle \{c\} \uplus G, \tilde{S}, C, T\rangle_n \longrightarrow_{\omega_t} \langle G, \tilde{S}, C', T\rangle_n}$$
Introduce$_{\omega_t}$	$$\frac{\text{h is a user-defined constraint}}{\langle \{h\} \uplus G, \tilde{S}, C, T\rangle_n \longrightarrow_{\omega_t} \langle G, \{h\#n\} \cup \tilde{S}, C, T\rangle_{n+1}}$$
Apply$_{\omega_t}$	$$\frac{r@H_1' \setminus H_2' \Leftrightarrow D \mid B \in P \quad x = Fv(H_1', H_2') \quad CT \models C \rightarrow \exists_x((chr(\tilde{H}_1, \tilde{H}_2) = (H_1', H_2')) \wedge D)}{\langle G, \{\tilde{H}_1\} \cup \{\tilde{H}_2\} \cup \tilde{S}, C, T\rangle_n \longrightarrow_{\omega_t} \langle B \uplus G, \{\tilde{H}_1\} \cup \tilde{S}, (chr(\tilde{H}_1, \tilde{H}_2) = (H_1', H_2')) \wedge C, T'\rangle_n}$$
	where $r@id(\tilde{H}_1, \tilde{H}_2) \notin T$ and $T' = T \cup \{r@id(\tilde{H}_1, \tilde{H}_2)\}$ if $\tilde{H}_2 = \emptyset$ otherwise $T' = T$.

Table 1.3: The transition system T_{ω_t} for ω_t semantics

The **Introduce**$_{\omega_t}$ transition is used to move a user-defined constraint h from the goal to the CHR constraint store, to label h with the first unused identifier n and finally to update the next free identifier to $n+1$. After this operation, h can be handled by applying CHR rules.

The **Apply**$_{\omega_t}$ transition uses the rule $r@H_1' \backslash H_2' \Leftrightarrow D \mid B$ provided that a matching substitution θ exists, such that $(H_1, H_2) = (H_1', H_2')\theta$, D is entailed by the built-in constraint store C of the computation and T does not contain the token $r@id(H_1, H_2)$. CHR constraints of H_2 are deleted. Constraints of the body B of rule r are added to the actual goal G, the built-in constraints $((H_1, H_2) = (H_1', H_2')) \wedge D$ is added to the built-in constraint store and finally the token $r@id(H_1, H_2)$ is added to the token store T, if the right hand side of the head of r, that is H_2', is empty.

Note that, unlike the operations defined for the original CHR (Section 1.4) and the ones defined for the first CHR that manage the trivial non-termination (Subsection 1.5.1), here only three transitions are defined: $Solve_{\omega_t}$, $Introduce_{\omega_t}$ and $Apply_{\omega_t}$. This happens because both a propagation and a simplification rule can be simulated by a simpagation one that is $Apply_{\omega_t}$. In fact, let $H \Rightarrow D|B$ be a propagation rule the equivalent simpagation rule is $H \setminus \emptyset \Leftrightarrow D|B$ and let $H \Leftrightarrow D|B$ be a simplification rule the equivalent simpagation rule is $\emptyset \setminus H \Leftrightarrow D|B$.

Now, given a goal G, the operational semantics, that is going to be introduced in the next chapter, observes the final stores of computations terminating with an empty goal

and an empty user-defined constraint. We call these observables, data sufficient answers, using the terminology of [Frü98], where the first element in the state tuple is the goal store, the second the CHR store, the third the built-in constraints store and the fourth the token store. We also have the last identification number in the state written as subscript.

Definition 1.2 (Data sufficient answers) *Let P be a program and let G be a goal. The set $\mathcal{S}A_P(G)$ of data sufficient answers for the query G in the program P, is defined as follows:*

$$\begin{aligned}\mathcal{S}A_P(G) = \ & \{\exists_{-Fv(G)} d \mid \langle G, \emptyset, \texttt{true}, \emptyset\rangle_1 \longrightarrow^*_{\omega_t} \langle \emptyset, \emptyset, d, T\rangle_n \not\longrightarrow_{\omega_t}\} \\ & \cup \\ & \{\texttt{false} \mid \ \langle G, \emptyset, \texttt{true}, \emptyset\rangle_1 \longrightarrow^*_{\omega_t} \langle G', K, d, T\rangle_n \textit{ and} \\ & \qquad\qquad CT \models d \leftrightarrow \texttt{false}\}.\end{aligned}$$

We can consider a different notion of answer [Frü98]. It is obtained by computations terminating with a user-defined constraint which does not need to be empty.

Definition 1.3 (Qualified answers) *Let P be a program and let G be a goal. The set $\mathcal{Q}A_P(G)$ of qualified answers for the query G in the program P, is defined as follows:*

$$\begin{aligned}\mathcal{Q}A_P(G) = \ & \{\exists_{-Fv(G)} chr(K) \wedge d \mid \langle G, \emptyset, \texttt{true}, \emptyset\rangle_1 \longrightarrow^*_{\omega_t} \langle \emptyset, K, d, T\rangle_n \not\longrightarrow_{\omega_t}\} \\ & \cup \\ & \{\texttt{false} \mid \ \langle G, \emptyset, \texttt{true}, \emptyset\rangle_1 \longrightarrow^*_{\omega_t} \langle G', K, d, T\rangle_n \textit{ and} \\ & \qquad\qquad CT \models d \leftrightarrow \texttt{false}\}.\end{aligned}$$

Unlike the definitions contained in [Frü98], those contained in Definition 1.2 and 1.3, have the new element T, known as the token store. Note that both previous notions of observables characterise an input/output behaviour, since the input constraint is implicitly considered in the goal.

Chapter 2

Compositional semantics

Compositionality is a very antique and interdisciplinary principle. In fact, it can be found in Mathematics, Linguistic Philosophy and Computer Science Semantics. Dummett [Dum73] asserts that it originated with Frege whereas Popper [Pop76] informs us that it can not be found in explicit form in Frege's writings. As a matter of fact, in the works of his youth, Frege only introduces the principle of *contextuality* [Fre84], which declares that one should ask for the meaning of a word only in the context of a sentence, and not in isolation. Said principle is completely different with respect to the contemporaneous idea of compositionality. In contrast, in his last publication, the idea of *compositionality* is introduced, even if informally, in particular in [Fre23]. The above mentioned principle can be represented by the following assertion: The meaning of a compound expression is a function of the meanings of its parts [Jan97].

Till now, a lot of CHR semantics have been defined as per [Frü98, DSGdlBH04, Sch05]. All these semantics, like other versions defined elsewhere, are not compositional with respect to the conjunctions of atoms in a goal. This is somewhat surprising, considering CHR both from the logic programming and the concurrency theory perspective. In fact, in the first case, the reference semantics (the least Herbrand model) as well as other more refined semantics (e.g. s-semantics) enjoy this form of compositionality. When considering CHR as a (committed choice) concurrent language the situation is analogous: conjunction of atoms can naturally be considered as parallel composition, and most semantics in concurrency theory are compositional with respect to parallel composition (as

well as all the other language operators). Indeed generally-speaking, compositionality is a very desirable feature for semantics, as it permits us to manage partially defined components and it can be the basis for defining incremental and modular tools for software analysis and verification. For these reasons in [DGM05] a fixpoint, and-compositional semantics for CHR was defined, which allows us to retrieve the semantics of a conjunctive query from the semantics of its components. This was obtained by using semantic structures based on traces, in a similar manner to what had already been done for data-flow languages [Jon85], imperative concurrent languages [Bro93] and concurrent constraint languages [dBP91]. The semantics defined in [DGM05] uses the operational semantics as defined in [Frü98] as its reference point which, as previously mentioned, allows trivial non-termination.

In this chapter we extend the work of [DGM05] by considering as reference semantics the one defined in [DSGdlBH04] which avoids trivial infinite computations by using the token store, as introduced in Subsection 1.5.2. It should be remembered that it allows us to memorize the history of applied propagation rules and therefore to control their application in order to avoid that the same rule is applied more than once to the same (occurrence of a) constraint in a derivation. As discussed in [DGM05], due to the presence of multiple heads in CHR, the traces needed to obtain compositionality are more complicated than those used for the other concurrent languages above mentioned. In this chapter the need to represent the token store further complicates the semantic model, since when composing two traces, representing two parallel processes (i.e. two jointed atoms), we must ensure that the same propagation rule is not applied twice to the same constraint. The resulting compositional semantics is therefore technically rather complicated, even though the underlying idea is simple.

The new compositional semantics defined in this chapter is proven correct with respect to a slightly different notion of observables, than the one in [DGM05], since now the token store will also be considered.

Some of the results were already published in [GMT06].

2.1 Compositional trace semantics

Given a program P, we say that a semantics $\mathcal{S}_P$ is and-compositional if $\mathcal{S}_P(A, B) = \mathcal{C}(\mathcal{S}_P(A), \mathcal{S}_P(B))$ for a suitable composition operator $\mathcal{C}$ which does not depend on the program P and where A and B are conjunctions of constraints. The presence of multiple heads in CHR makes not and-compositional the semantics which associates with a program P the function $\mathcal{SA}_P$ (Definition 1.2). In fact goals which have the same input/output behavior can behave differently when composed with other goals. Consider for example the program P, consisting of the single rule $r@g, h \Leftrightarrow true|c$ (where c is a built-in constraint). According to Definition 1.2 we have the following result $\mathcal{SA}_P(g) = \mathcal{SA}_P(k) = \emptyset$, while $\mathcal{SA}_P(g, h) = \{\langle \exists_{-Fv(g,h)} c \rangle\} \neq \emptyset = \mathcal{SA}_P(k, h)$, where k is a CHR constraint. An analogous example can be made to show that $\mathcal{QA}$ semantics is also not and-compositional.

In order to solve the problem exemplified above we must first define a new transition system that will then be used to generate the sequences appearing in the compositional model, by using a standard fixpoint construction. This transition system also collects in the semantics the "missing" parts of heads which are needed in order to proceed with the computation. For example, when considering the program P above, we should be able to state that the goal g produces the constraint c, provided that the external environment (i.e. a conjunctive goal) contains the user-defined constraint h. When composing (by using a suitable notion of composition) such a semantics with the one of a goal which contains h, we can verify that the "assumption" h is satisfied and therefore obtains the correct semantics for g, h. In order to model correctly the interaction of different processes we have to use sequences, analogously to what happens with other concurrent paradigms.

Thus, the new transition system we define is $T = (Conf, \longrightarrow_P)$, where configurations in *Conf* are triples of the form $\langle \tilde{G}, c, T \rangle_n$: $\tilde{G}$ is a set of built-in and identified CHR constraints (the goal), c is a (conjunction of) built-in constraint(s) (the store), T is a set of tokens and n is an integer greater or equal to the biggest identifier used either to number a CHR constraint in $\tilde{G}$ or in a token in T. The transition relation $\longrightarrow_P \subseteq Conf \times Conf \times \wp(\mathcal{U})$, where P is a program, is the least relation satisfying the rules in Table 2.1 where

Solve'

$$\frac{CT \models c \wedge d \leftrightarrow d' \text{ and } c \text{ is a built-in constraint}}{\langle \{c\} \cup \tilde{G}, d, T\rangle_n \longrightarrow^{\emptyset}_{P} \langle \tilde{G}, d', T\rangle_n}$$

Apply'

$$\frac{r@H'_1 \setminus H'_2 \Leftrightarrow C \mid B \in P \quad x = Fv(H'_1, H'_2) \quad G \neq \emptyset \quad CT \models c \rightarrow \exists_x((chr(\tilde{H}_1, \tilde{H}_2) = (H'_1, H'_2)) \wedge C)}{\langle \tilde{G} \cup \tilde{G}', c, T\rangle_n \longrightarrow^{K}_{P} \langle I^{n+k+m}_{n+k}(B) \cup \{\tilde{H}_1\} \cup \tilde{G}', (chr(\tilde{H}_1, \tilde{H}_2) = (H'_1, H'_2)) \wedge c, T'\rangle_{n+k+m}}$$

where k and m are the number of CHR constraints in K and in B respectively,
$\{\tilde{G}\} \cup \{I^{n+k}_n(K)\} = \{\tilde{H}_1\} \cup \{\tilde{H}_2\}$, $r@id(\tilde{H}_1, \tilde{H}_2) \notin T$ and
if $\tilde{H}_1 = \emptyset$ then $T' = T$ else $T' = T \cup \{r@id(\tilde{H}_1, \tilde{H}_2)\}$

Table 2.1: The transition system T for compositional semantics

$\wp(A)$ denotes the powerset set of A. Note that we modify the notion of configuration ($Conf_t$) used before by merging the goal store with the CHR store, since we do not need to distinguish between them. Consequently the **Introduce** rule is now useless and we eliminate it. On the other hand, we need the information on the new assumptions, which is added as a label to the transitions.

We need some further notation: given a goal G, we denote by $\tilde{G}$ one of the possible identified versions of G. Moreover, assuming that G contains m CHR-constraints, we define a function $I^{n+m}_n(G)$ which identifies each CHR constraint in G by associating a unique integer in $[n+1, m+n]$ with it, according to the lexicographic order. The identifier association is applied both to the initial goal store, at the beginning of the derivation, and to the bodies of the rules that are added to the goal store during the computation steps. If $m = 0$, we assume that $I^n_n(G) = G$.

Let us now briefly consider the rules in Table 2.1. **Solve'** is essentially the same rule as the one defined in Table 1.3, while the **Apply'** rule is modified to consider assumptions: when reducing a goal G by using a rule having head H, the set of assumptions $K = H \setminus G$ (with $H \neq K$) is used to label the transition. Note that since we apply the function I^{n+k}_n to the assumptions K, each atom in K is associated with an identifier in $[n+1, n+k]$. As before, we assume that program rules use fresh variables to avoid variable name captures. Given a goal G with m CHR-constraints an *initial configuration* has the form $\langle I^m_0(G), \texttt{true}, \emptyset\rangle_m$ where $I^m_0(G)$ is the identified version of the goal.

A *final configuration* has either the form $\langle \tilde{G}, \texttt{false}, T\rangle_n$ if it has *failed* or $\langle \emptyset, c, T\rangle_n$, if it is successful.

The following example shows a derivation obtained by the new transition system.

Example 2.1 Given the goal $(C = 7, A =< B, C =< A, B =< C, B =< C)$ and the program of Example 1.1 by using the transition system of Table 2.1 we obtain the following derivation (where the last step is not a final one):

$$\begin{array}{ll}
\langle\{C = 7, A =< B\#1, C =< A\#2, B =< C\#3, B =< C\#4\}, \mathtt{true}, \emptyset\rangle_4 \rightarrow^{\emptyset} & \mathit{Solve} \\
\langle\{A =< B\#1, C =< A\#2, B =< C\#3, B =< C\#4\}, C = 7, \emptyset\rangle_4 \rightarrow^{\emptyset} & trs@1,3 \\
\langle\{A =< C\#5, A =< B\#1, C =< A\#2, B =< C\#3, B =< C\#4\}, C = 7, \{trs@1,3\}\rangle_5 \rightarrow^{\emptyset} & asx@5,2 \\
\langle\{A = C, A =< B\#1, B =< C\#3, B =< C\#4\}, C = 7, \{trs@1,3\}\rangle_5 \rightarrow^{\emptyset} & \mathit{Solve} \\
\langle\{A =< B\#1, B =< C\#3, B =< C\#4\}, (A = C \wedge C = 7), \{trs@1,3\}\rangle_5 \rightarrow^{\emptyset} & asx@1,3 \\
\langle\{B = C, B =< C\#4\}, (A = C \wedge C = 7), \{trs@1,3\}\rangle_5 \rightarrow^{\emptyset} & \mathit{Solve} \\
\langle\{B =< C\#4\}, (B = C \wedge A = C \wedge C = 7), \{trs@1,3\}\rangle_5 &
\end{array}$$

□

The semantic domain of our compositional semantics is based on sequences which represent derivations obtained by the transition system in Table 2.1. We first consider "concrete" sequences consisting of tuples of the form $\langle\tilde{G}, c, T, m, I_m^{m+k}(K), \tilde{G}', d, T', m'\rangle$. Such a tuple represents exactly a derivation step $\langle\tilde{G}, c, T\rangle_m \longrightarrow_P^K \langle\tilde{G}', d, T'\rangle_{m'}$, where k is the number of CHR atoms in K. The sequences we consider are terminated by tuples of the form $\langle\tilde{G}, c, T, n, \emptyset, \tilde{G}, c, T, n\rangle$, with either $c = \mathtt{false}$ or $\tilde{G}$ is a set of identified CHR constraints, which represent a terminating step (see the precise definition below). Since a sequence represents a derivation, we assume that if

$$\begin{array}{ll}
\ldots & \langle\tilde{G}_i, c_i, T_i, m_i, \tilde{K}_i, \tilde{G}'_i, d_i, T'_i, m'_i\rangle \\
& \langle\tilde{G}_{i+1}, c_{i+1}, T_{i+1}, m_{i+1}, \tilde{K}_{i+1}, \tilde{G}'_{i+1}, d_{i+1}, T'_{i+1}, m'_{i+1}\rangle \ldots
\end{array}$$

appears in a sequence, then $\tilde{G}'_i = \tilde{G}_{i+1}$, $T'_i = T_{i+1}$ and $m'_i \leq m_{i+1}$ hold.

On the other hand, the input store c_{i+1} can be different from the output store d_i produced in a previous step, since we need to perform all the possible assumptions on the constraint c_{i+1} produced by the external environment, in order to obtain a compositional semantics, so we can assume that $CT \models c_{i+1} \rightarrow d_i$ holds. This means that the assumption made on the external environment cannot be weaker than the constraint store produced in the previous step. This reflects the fact that information can be added to the constraint store and cannot be deleted from it. The idea is that $CT \models c_{i+1} \rightarrow d_i$, with $c_{i+1} \neq d_i$, supposes that another sequence, which generates the constraint $c_{i+1} \backslash d_i$ is inserted, into the sequence that we are considering to obtain a real computation, oftentimes leading to

a concrete sequence which does not correspond a real computation. Finally, note that assumptions on user-defined constraints (label K) are made only for the atoms which are needed to "complete" the current goal in order to apply a clause. In other words, no assumption can be made in order to apply clauses whose heads do not share any predicate with the current goal.

Example 2.2 The following is the representation of the derivation of Example 2.1 in terms of concrete sequences:

$$
\begin{array}{l}
\langle \{C = 7, A =< B\#1, C =< A\#2, B =< C\#3, B =< C\#4\}, \mathtt{true}, \emptyset, 4, \emptyset \\
\qquad \{A =< B\#1, C =< A\#2, B =< C\#3, B =< C\#4\}, C = 7, \emptyset, 4\rangle \\
\langle \{A =< B\#1, C =< A\#2, B =< C\#3, B =< C\#4\}, C = 7, \emptyset, 4, \emptyset \\
\qquad \{A =< C\#5, A =< B\#1, C =< A\#2, B =< C\#3, B =< C\#4\}, C = 7, \{trs@1, 3\}, 5\rangle \\
\langle \{A =< C\#5, A =< B\#1, C =< A\#2, B =< C\#3, B =< C\#4\}, C = 7, \{trs@1, 3\}, 5, \emptyset \\
\qquad \{A = C, A =< B\#1, B =< C\#3, B =< C\#4\}, C = 7, \{trs@1, 3\}, 5\rangle \\
\langle \{A = C, A =< B\#1, B =< C\#3, B =< C\#4\}, C = 7, \{trs@1, 3\}, 5, \emptyset \\
\qquad \{A =< B\#1, B =< C\#3, B =< C\#4\}, (A = C, C = 7), \{trs@1, 3\}, 5\rangle \\
\langle \{A =< B\#1, B =< C\#3, B =< C\#4\}, (A = C, C = 7), \{trs@1, 3\}, 5, \emptyset \\
\qquad \{B = C, B =< C\#4\}, (A = C, C = 7), \{trs@1, 3\}, 5\rangle \\
\langle \{B = C, B =< C\#4\}, (A = C, C = 7), \{trs@1, 3\}, 5, \emptyset \\
\qquad \{B =< C\#4\}, (B = C, A = C, C = 7), \{trs@1, 3\}, 5\rangle \\
\langle \{B =< C\#4\}, (B = C, A = C, C = 7), \{trs@1, 3\}, 5, \emptyset \\
\qquad \{B =< C\#4\}, (B = C, A = C, C = 7), \{trs@1, 3\}, 5\rangle
\end{array}
$$

□

We then define formally concrete sequences, which represent derivation steps, performed by using the new transition system as follows:

Definition 2.1 (Concrete sequences) *The set $\mathcal{S}eq$ containing all the possible (concrete) sequences is defined as the set*

$$
\begin{array}{rl}
\mathcal{S}eq = & \{\langle \tilde{G}_1, c_1, T_1, m_1, \tilde{K}_1, \tilde{G}_2, d_1, T'_1, m'_1\rangle\langle \tilde{G}_2, c_2, T_2, m_2, \tilde{K}_2, \tilde{G}_3, d_2, T'_2, m'_2\rangle \cdots \\
& \langle \tilde{G}_n, c_n, T_n, m_n, \emptyset, \tilde{G}_n, c_n, T_n, m_n\rangle \mid \\
& \textit{for each } j,\ 1 \le j \le n \textit{ and for each } i, 1 \le i \le n-1, \\
& \tilde{G}_j \textit{ are identified CHR goals, } \tilde{K}_i \textit{ are sets of identified CHR constraints,} \\
& T_j, T'_i \textit{ are sets of tokens, } m_j, m'_i \textit{ are natural numbers and} \\
& c_j, d_i \textit{ are built-in constraints such that} \\
& T'_i \supseteq T_i,\ T_{i+1} \supseteq T'_i,\ m'_i \ge m_i,\ m_{i+1} \ge m'_i, \\
& CT \models d_i \rightarrow c_i,\ CT \models c_{i+1} \rightarrow d_i \textit{ and} \\
& \textit{either } c_n = \mathtt{false} \textit{ or } \tilde{G}_n \textit{ is a set of identified CHR constraints}\}.
\end{array}
$$

From these concrete sequences we extract some more abstract sequences which are the objects of our semantic domain. If $\langle \tilde{G}, c, T, m, \tilde{K}, \tilde{G}', d, T', m' \rangle$ is a tuple, different from the last one, appearing in a sequence $\delta \in \mathcal{S}eq$, we extract from it a tuple of the form $\langle c, \tilde{K}, \tilde{H}, d \rangle$ where c and d are the input and output store respectively, $\tilde{K}$ are the assumptions and $\tilde{H}$ the stable atoms: the restriction of goal $\tilde{G}$ to the identified constraints, that will not be used any more in δ to fire a rule. The output goal $\tilde{G}'$ is no longer considered. Intuitively, $\tilde{H}$ contains those atoms which are available for satisfying assumptions of other goals, when composing two different sequences, representing two derivations of different goals.

If $\langle c_i, \tilde{K}_i, \tilde{H}_i, d_i \rangle \langle c_{i+1}, \tilde{K}_{i+1}, \tilde{H}_{i+1}, d_{i+1} \rangle$ is in a sequence, we also assume that $\tilde{H}_i \subseteq \tilde{H}_{i+1}$ holds, since the set of those atoms which will not be rewritten in the derivation can only increase. Moreover, if the last tuple in δ is $\langle \tilde{G}, c, T, m, \emptyset, \tilde{G}, c, T, m \rangle$, we extract from it a tuple of the form $\langle c, \tilde{G}, T \rangle$. We can define formally the semantic domain as follows in the next definition but first we must define the abstract sequences that will be used from now on.

Definition 2.2 (Sequences) *The semantic domain $\mathcal{D}$ containing all the possible sequences is defined as the set*

$$
\begin{aligned}
\mathcal{D} = \; & \{ \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1 \rangle \langle c_2, \tilde{K}_2, \tilde{H}_2, d_2 \rangle \ldots \langle c_m, \tilde{H}_m, T \rangle \mid \\
& \quad m \geq 1, \textit{ for each } j,\ 1 \leq j \leq m \textit{ and for each } i, 1 \leq i \leq m-1, \\
& \quad \tilde{H}_j \textit{ and } \tilde{K}_i \textit{ are sets of identified CHR constraints,} \\
& \quad T \textit{ is a set of tokens, } \tilde{H}_i \subseteq \tilde{H}_{i+1} \textit{ and } c_i, d_i \textit{ are built-in constraints} \\
& \quad \textit{such that } CT \models d_i \rightarrow c_i \textit{ and } CT \models c_{i+1} \rightarrow d_i \textit{ hold} \}.
\end{aligned}
$$

In order to define our semantics we need two more notions. The first one is an abstraction operator α, which extracts from the concrete sequences in $\mathcal{S}eq$ (representing exactly derivation steps) the sequences used in our semantic domain. To this aim we need the notion of stable atom.

Definition 2.3 (Stable atoms and Abstraction) *Let*

$$
\delta = \langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n_1' \rangle \ldots \langle \tilde{G}_m, c_m, T_m, n_m, \emptyset, \tilde{G}_m, c_m, T_m, n_m \rangle
$$

be a sequence of derivation steps where we assume that the CHR atoms are identified. We say that an identified atom $g\#l$ is stable in δ if $g\#l$ appears in $\tilde{G}_j$ and the identifier l does not appear in $T_j \setminus T_1$, for each $1 \leq j \leq m$. The abstraction operator $\alpha : \mathcal{S}eq \rightarrow \mathcal{D}$ is then defined inductively as

$$\alpha(\langle \tilde{G}, c, T, n, \emptyset, \tilde{G}, c, T, n\rangle) = \langle c, \tilde{G}, T\rangle$$
$$\alpha(\langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1\rangle \cdot \delta') = \langle c_1, \tilde{K}_1, \tilde{H}, d_1\rangle \cdot \alpha(\delta').$$

where $\tilde{H}$ is the set consisting of all the identified atoms $\tilde{h}$ such that $\tilde{h}$ is stable in $\langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1\rangle \cdot \delta'$.

We should point out at this point that the token store does not only keep trace of the application of the rules with empty lhs (left hand side) of \: the one equivalent of the pure simplification rules.

The following example illustrates the use of the abstraction function α.

Example 2.3 The application of the function α to the (concrete) sequence in Example 2.2 gives the following abstract sequence:

$$\langle \mathtt{true}, \emptyset, \{B =< C\#4\}, C = 7\rangle\ \langle C = 7, \emptyset, \{B =< C\#4\}, C = 7\rangle\ \langle C = 7, \emptyset, \{B =< C\#4\}, C = 7\rangle$$
$$\langle C = 7, \emptyset, \{B =< C\#4\}, (A = C \wedge C = 7)\rangle\ \langle (A = C \wedge C = 7), \emptyset, \{B =< C\#4\}, (A = C \wedge C = 7)\rangle$$
$$\langle (A = C \wedge C = 7), \emptyset, \{B =< C\#4\}, (B = C \wedge A = C \wedge C = 7)\rangle$$
$$\langle (B = C \wedge A = C \wedge C = 7), \{B =< C\#4\}, \{trs@1, 3\}\rangle$$

□

Before defining the compositional semantics we need a further notion of compatibility. To this end, given a sequence of derivation steps

$$\delta = \langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1\rangle \ldots \langle \tilde{G}_m, c_m, T_m, n_m, \emptyset, \tilde{G}_m, c_m, T_m, n_m\rangle$$

and a derivation step $t = \langle \tilde{G}, c, T, n, \tilde{K}, \tilde{G}', d, T', n'\rangle$, we may define

$V_{loc}(t) = Fv(G', d) \setminus Fv(G, c, K)$ (the local variables of t),

$V_{ass}(\delta) = \bigcup_{i=1}^{m-1} Fv(K_i)$ (the variables in the assumptions of δ) and

$V_{loc}(\delta) = \bigcup_{i=1}^{m-1} Fv(G_{i+1}, d_i) \setminus Fv(G_i, c_i, K_i)$ (the local variables of δ, namely the variables, introduced by the clauses used in the derivation δ).

Definition 2.4 (Compatibility) *Let $t = \langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1 \rangle$ a tuple representing a derivation step for the goal G_1 and let*

$$\delta = \langle \tilde{G}_2, c_2, T_2, n_2, \tilde{K}_2, \tilde{G}_3, d_2, T_3, n'_2 \rangle \dots \langle \tilde{G}_m, c_m, T_m, n_m, \emptyset, \tilde{G}_m, c_m, T_m, n_m \rangle$$

be a sequence of derivation steps for G_2. We say that t is compatible with δ if the following holds:

1. $V_{loc}(\delta) \cap Fv(t) = \emptyset$,
2. $V_{loc}(t) \cap V_{ass}(\delta) = \emptyset$ *and*
3. *for* $i \in [2, m]$, $V_{loc}(t) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j)$.

The three conditions of Definition 2.4 reflect the following facts: 1) The clauses in a derivation are renamed apart; 2) The variables in the assumptions are disjointed from the local variables (those of the rules used) in the derivation; 3) Each of the local variables appearing in an input constraint has already appeared in an output constraint. These conditions ensure that, by using the notation of the definition above, if t is compatible with δ then $t \cdot \delta$ is a sequence of derivation steps for G_1. Moreover, the local variables in a derivation δ and in the abstraction of δ are the same (Lemma 2.1). We now have all the tools necessary to define compositional semantics.

Definition 2.5 (Compositional semantics) *Let P be a program and let G be a goal. The compositional semantics of G in the program P, $\mathcal{S}_P : Goals \to \wp(\mathcal{D})$, is defined as*

$$\mathcal{S}_P(G) = \alpha(\mathcal{S}'_P(G))$$

where α is the pointwise extension to sets of the abstraction operator given in Definition 2.3 and $\mathcal{S}'_P : Goals \to \wp(\mathcal{S}eq)$ is defined inductively as follows:

$$\begin{array}{rl} \mathcal{S}'_P(G) = & \{\langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1 \rangle \cdot \delta \in \mathcal{S}eq \mid \\ & \quad \tilde{G}_1 \text{ is an identified version of } G, \\ & \quad CT \not\models c_1 \leftrightarrow \texttt{false},\ \langle \tilde{G}_1, c_1, T_1 \rangle_{n_1} \longrightarrow_P^{K_1} \langle \tilde{G}_2, d_1, T_2 \rangle_{n'_1} \\ & \quad \text{and } \delta \in \mathcal{S}'_P(G_2) \text{ for some } \delta \text{ such that} \\ & \quad \langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1 \rangle \text{ is compatible with } \delta\} \cup \\ & \{\langle \tilde{G}, c, T, n, \emptyset, \tilde{G}, c, T, n \rangle \in \mathcal{S}eq\}. \end{array}$$

It can be observed that $\mathcal{S}'_P(G)$ is also the least fixpoint of the corresponding operator $\Phi \in (Goals \to \wp(\mathcal{S}eq)) \to Goals \to \wp(\mathcal{S}eq)$ defined by

$$\begin{aligned}\Phi(I)(G) = \ & \{\langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1\rangle \cdot \delta \in \mathcal{S}eq \mid \\ & \quad \tilde{G}_1 \text{ is an identified version of } G, \\ & \quad CT \not\models c_1 \leftrightarrow \texttt{false},\ \langle \tilde{G}_1, c_1, T_1\rangle_{n_1} \longrightarrow^{K_1}_{P} \langle \tilde{G}_2, d_1, T_2\rangle_{n'_1} \\ & \quad \text{and } \delta \in I(G_2) \text{ for some } \delta \text{ such that} \\ & \quad \langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1\rangle \text{ is compatible with } \delta\} \cup \\ & \{\langle \tilde{G}, c, T, n, \emptyset, \tilde{G}, c, T, n\rangle \in \mathcal{S}eq\}.\end{aligned}$$

where $I : Goals \to \wp(\mathcal{S}eq)$ stands for a generic interpretation assigning a set of sequences to a goal. The ordering of the set of interpretations $Goals \to \wp(\mathcal{S}eq)$ is that of (point-wise extended) set-inclusion. It is straightforward to check that Φ is continuous (on a CPO), thus standard results ensure that the fixpoint can be calculated by $\sqcup_{n\geq 0}\phi^n(\perp)$, where ϕ^0 is the identity map and for $n > 0$, $\phi^n = \phi \circ \phi^{n-1}$ (see for example [DP90]).

2.2 Compositionality

In this section, we first define the semantic operator $\|$ (Definition 2.9) which allows us to compose two sets of sequences, describing the semantics of two goals, in order to obtain the semantics of their conjunction (i.e. parallel composition). Such an operator consists of two parts. Intuitively, if S_1 and S_2 are the sets we want to compose, first of all every sequence $\sigma_1 \in S_1$ is interleaved with every sequence $\sigma_2 \in S_2$ (Definition 2.6). Then the η operator (Definition 2.8) is applied: this adds all the sequences which can be obtained from the original ones by to eliminating the assumptions which are satisfied by the stable atoms (Definition 2.7).

By using the $\|$ operator we are in position to prove that the semantics defined in the previous section is and-compositional and correct with respect to the observables $\mathcal{SA}_P$.

We now need some more notations. Assuming that $\sigma = \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1\rangle\langle c_2, \tilde{K}_2, \tilde{H}_2, d_2\rangle \cdots \langle c_m, \tilde{H}_m, T\rangle \in \mathcal{D}$ is the abstraction of a sequence for the goal G, we define the overloaded operator $id(\sigma) = id(\bigcup_{i=1}^{m-1} \tilde{K}_i) \cup id(\bigcup_{i=1}^{m} \tilde{H}_i)$ as the set of identification values of

all CHR constraints in σ. We define the following operators, some of which are analogous of those already introduced for concrete sequences:

$V_{ass}(\sigma) = \bigcup_{i=1}^{m-1} Fv(K_i)$ (the variables in the assumptions of σ),

$V_{stable}(\sigma) = Fv(H_m) = \bigcup_{i=1}^{m} Fv(H_i)$ (the variables in the stable sets of σ),

$V_{constr}(\sigma) = \bigcup_{i=1}^{m-1} Fv(d_i) \setminus Fv(c_i)$ (the variables in the output constraints of σ which are not in the corresponding input constraints),

$V_{loc}(\sigma) = (V_{constr}(\sigma) \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup Fv(G))$ (the local variables of a sequence σ, which are the local variables of the derivations δ such that $\alpha(\delta) = \sigma$).

The $\|$ operator which performs the interleaving of two sequences is defined as follows.

Definition 2.6 (Composition) *The operator* $\|: \mathcal{D} \times \mathcal{D} \to \wp(\mathcal{D})$ *is defined as follows. Let* $\sigma_1, \sigma_2 \in \mathcal{D}$ *be sequences for the goals* H *and* G*, respectively, such that* $id(\sigma_1) \cap id(\sigma_2) = \emptyset$ *and*

$$(V_{loc}(\sigma_1) \cup Fv(H)) \cap (V_{loc}(\sigma_2) \cup Fv(G)) = Fv(H) \cap Fv(G). \tag{2.1}$$

Then $\sigma_1 \| \sigma_2$ *is defined by cases as follows:*

1. *If both* σ_1 *and* σ_2 *have length* 1 *and have the same store, say* $\sigma_1 = \langle c, \tilde{H}, T \rangle$ *and* $\sigma_2 = \langle c, \tilde{G}, T' \rangle$*, then*

$$\sigma_1 \| \sigma_2 = \{\langle c, \tilde{H} \cup \tilde{G}, T \cup T' \rangle \in \mathcal{D}\}.$$

2. *If* $\sigma_2 = \langle e, \tilde{G}, T' \rangle$ *has length* 1 *and* $\sigma_1 = \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1 \rangle \cdot \sigma_1'$ *has length* > 1 *then*

$$\sigma_1 \| \sigma_2 = \{\langle c_1, \tilde{K}_1, \tilde{H}_1 \cup \tilde{G}, d_1 \rangle \cdot \sigma \in \mathcal{D} \mid \sigma \in {\sigma'}_1 \| \sigma_2\}.$$

3. *If* $\sigma_1 = \langle c, \tilde{H}, T \rangle$ *has length* 1 *and* $\sigma_2 = \langle e_1, \tilde{J}_1, \tilde{Y}_1, f_1 \rangle \cdot \sigma_2'$ *has length* > 1 *then*

$$\sigma_1 \| \sigma_2 = \{\langle e_1, \tilde{J}_1, \tilde{H} \cup \tilde{Y}_1, f_1 \rangle \cdot \sigma \in \mathcal{D} \mid \sigma \in \sigma_1 \| {\sigma'}_2\}.$$

4. *If both* $\sigma_1 = \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1\rangle \cdot \sigma'_1$ *and* $\sigma_2 = \langle e_1, \tilde{J}_1, \tilde{Y}_1, f_1\rangle \cdot \sigma'_2$ *have length* > 1 *then*

$$\begin{aligned}\sigma_1 \parallel \sigma_2 = \quad & \{\langle c_1, \tilde{K}_1, \tilde{H}_1 \cup \tilde{Y}_1, d_1\rangle \cdot \sigma \in \mathcal{D} \mid \quad \sigma \in {\sigma'}_1 \parallel \sigma_2\} \cup \\ & \{\langle e_1, \tilde{J}_1, \tilde{H}_1 \cup \tilde{Y}_1, f_1\rangle \cdot \sigma \in \mathcal{D} \mid \quad \sigma \in \sigma_1 \parallel {\sigma'}_2\}.\end{aligned}$$

It is worth noting that the condition $id(\sigma_1) \cap id(\sigma_2) = \emptyset$ avoids the capture of identifiers, while the condition (2.1) in Definition 2.6 imposes the fact that there are no common local variables between the two sequences σ_1 and σ_2. This will be used in the proofs to ensure that when using $Apply'$ rules to concrete sequences generating σ_1 and σ_2, no local variables are shared.

Example 2.4 We now show an example of abstract sequences for the goals $(A =< B, C =< A)$ and $(C = 7, B =< C, B =< C)$, using the program in Example 2.1. The goal $(A =< B, C =< A)$ has the derivation $d =$

$$\begin{array}{ll}
\langle\{A =< B\#1, C =< A\#2\}, C = 7, \emptyset\rangle_4 \rightarrow^{\{B=<C\}} & trs@1,5 \\
\langle\{A =< C\#6, B =< C\#5, A =< B\#1, C =< A\#2\}, C = 7, \{trs@1,5\}\rangle_6 \rightarrow^{\emptyset} & asx@6,2 \\
\langle\{A = C, B =< C\#5, A =< B\#1\}, C = 7, \{trs@1,5\}\rangle_6 \rightarrow^{\emptyset} & Solve \\
\langle\{B =< C\#5, A =< B\#1\}, (A = C \wedge C = 7), \{trs@1,5\}\rangle_6 \rightarrow^{\emptyset} & asx@5,1 \\
\langle\{B = C\}, (A = C \wedge C = 7), \{trs@1,5\}\rangle_6 \rightarrow^{\emptyset} & Solve \\
\langle\emptyset, (B = C \wedge A = C \wedge C = 7), \{trs@1,5\}\rangle_6 &
\end{array}$$

and denoting by δ the sequence arising from such a computation d, we obtain the abstract sequence $\alpha(\delta) =$

$$\begin{array}{ll}
\langle C = 7, \{B =< C\#5\}, \emptyset, C = 7\rangle & (a) \\
\langle C = 7, \emptyset, \emptyset, C = 7\rangle & (b) \\
\langle C = 7, \emptyset, \emptyset, (A = C \wedge C = 7)\rangle & (c) \\
\langle (A = C \wedge C = 7), \emptyset, \emptyset, (A = C \wedge C = 7)\rangle & (d) \\
\langle (A = C \wedge C = 7), \emptyset, \emptyset, (B = C \wedge A = C \wedge C = 7)\rangle & (e) \\
\langle (B = C \wedge A = C \wedge C = 7), \emptyset, \{trs@1,5\}\rangle & (f)
\end{array}$$

Moreover, we have the following derivation step for $(C = 7, B =< C, B =< C)$

$$\langle\{C = 7, B =< C\#3, B =< C\#4\}, \mathtt{true}, \emptyset\rangle_4 \rightarrow^{\emptyset} \langle\{B =< C\#3, B =< C\#4\}, C = 7, \emptyset\rangle_4 \ \ Solve$$

and therefore we can say that

$$\begin{array}{rll}
\gamma = & \langle & \{C = 7, B =< C\#3, B =< C\#4\}, \mathtt{true}, \emptyset, 4, \emptyset, \{B =< C\#3, B =< C\#4\}, C = 7, \emptyset, 4\rangle \\
 & \langle & \{B =< C\#3, B =< C\#4\}, (B = C \wedge A = C \wedge C = 7), \emptyset, 4, \emptyset, \\
 & & \{B =< C\#3, B =< C\#4\}, (B = C \wedge A = C \wedge C = 7), \emptyset, 4\rangle
\end{array}$$

is a sequence for $(C = 7, B =< C, B =< C)$. Then $\alpha(\gamma)$ is the following sequence

$$\begin{array}{ll}
\langle \mathtt{true}, \emptyset, \{B =< C\#3, B =< C\#4\}, C = 7\rangle & (g) \\
\langle (B = C \wedge A = C \wedge C = 7), \{B =< C\#3, B =< C\#4\}, \emptyset\rangle & (h)
\end{array}$$

□

The following substitution operator is used in order to satisfy an assumption, by using a stable atom. It allows us to replace the assumption $g\#j$ by the identified stable atom $h\#i$ everywhere in the sequence, and to replace j by the identifier i in every element of token set, provided that token set cardinality does not decrease.

Definition 2.7 (Substitution operators) *Let T be a token set, S be a set of identified atoms, $id_1, \ldots, id_o$, $id'_1, \ldots, id'_o$ be identification values and let $g_1\#id_1, \ldots, g_o\#id_o$, $h_1\#id'_1, \ldots, h_o\#id'_o$ be identified atoms.*
Moreover, let $\sigma = \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1\rangle\langle c_2, \tilde{K}_2, \tilde{H}_2, d_2\rangle \cdots \langle c_m, \tilde{H}_m, T\rangle \in \mathcal{D}$.

- *$T' = T[id_1/id'_1, \ldots, id_o/id'_o]$ is the token set obtained from T, by substituting each occurrence of the identifier id_l with id'_l, for $1 \leq l \leq o$. The operation is defined if T and T' have the same cardinality (namely, there are no elements in T which collapse when we apply the substitution).*

- *$S[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]$ is the set of identified atoms obtained from S, by substituting each occurrence of the identified atom $g_l\#id_l$ with $h_l\#id'_l$, for $1 \leq l \leq o$.*

- *$\sigma' = \sigma\,[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]$ is defined only if $T' = T[id_1/id'_1, \ldots, id_o/id'_o]$ is defined and in this case*

$$\sigma' = \langle c_1, \tilde{K}'_1, \tilde{H}'_1, d_1\rangle\langle c_2, \tilde{K}'_2, \tilde{H}'_2, d_2\rangle \cdots \langle c_m, \tilde{H}'_m, T'\rangle \in \mathcal{D},$$

with $1 \leq l \leq m-1$, $1 \leq p \leq m$, $\tilde{K}'_l = \tilde{K}_l[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]$ and $\tilde{H}'_p = \tilde{H}_p[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]$.

The η operator performs the satisfaction of assumptions, using stable atoms in the composed sequence, as previously mentioned.

Definition 2.8 (η operator) *Let $\tilde{W}$ be a set of identified CHR atoms, let σ be a sequence in $\mathcal{D}$ of the form*

$$\langle c_1, \tilde{K}_1, \tilde{H}_1, d_1\rangle\langle c_2, K_2, H_2, d_2\rangle \ldots \langle c_m, \tilde{H}_m, T\rangle.$$

We denote by $\sigma \setminus \tilde{W} \in \mathcal{D}$ *the sequence*

$$\langle c_1, \tilde{K}_1, \tilde{H}_1 \setminus \tilde{W}, d_1\rangle\langle c_2, \tilde{K}_2, \tilde{H}_2 \setminus \tilde{W}, d_2\rangle \ldots \langle c_m, \tilde{H}_m \setminus \tilde{W}, T\rangle$$

(where the sets difference $\tilde{H}_j \setminus \tilde{W}$ *considers identifications, with* $1 \leq j \leq m$*).*

The operator $\eta : \wp(\mathcal{D}) \to \wp(\mathcal{D})$ *is defined as follows. Given* $S \in \wp(\mathcal{D})$, $\eta(S)$ *is the least set, satisfying the following conditions:*

- $S \subseteq \eta(S)$*;*
- *if* $\sigma' \cdot \langle c, \tilde{K}, \tilde{H}, d\rangle \cdot \sigma'' \in \eta(S)$ *and two sets of identified atoms exist, namely* $\tilde{K}' = \{g_1\#id_1, \ldots, g_o\#id_o\} \subseteq \tilde{K}$ *and* $\tilde{W} = \{h_1\#id'_1, \ldots, h_o\#id'_o\} \subseteq \tilde{H}$ *such that*

  1. *for* $1 \leq l \leq o$, $CT \models (c \wedge g_l) \leftrightarrow (c \wedge h_l)$ *and*
  2. $\bar{\sigma} = ((\sigma' \cdot \langle c, \tilde{K} \setminus \tilde{K}', \tilde{H}, d\rangle \cdot \sigma'') \setminus \tilde{W})\,[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]$ *is defined,*

  then $\bar{\sigma} \in \eta(S)$*.*

Note that Definition 2.8 introduces an upper closure operator[1] which satisfies a set of sequences S, by adding new sequences where redundant assumptions can be removed. An assumption $g\#i$ in $\tilde{K}$ can be removed if $h\#j$ appears as a stable atom in $\tilde{H}$ and the built-in store c implies that g is equivalent to h. Once a stable atom is “consumed” for satisfying an assumption, it is removed from the sets of stable atoms of all the tuples appearing in the sequence, to avoid multiple uses of the same atom.

We can finally define the composition operator $\|$ on sets of sequences. To simplify the notation we denote with the symbol $\|$ both the operator acting on sequences in addition to the one acting on sets of sequences.

[1] $S \subseteq \eta(S)$ holds by definition, and it is easy to see that $\eta(\eta(S)) = \eta(S)$ holds and that $S \subseteq S'$ implies $\eta(S) \subseteq \eta(S')$.

Definition 2.9 (Set of sequence composition) *The composition of sets of sequences* $\|$: $\wp(\mathcal{D}) \times \wp(\mathcal{D}) \rightarrow \wp(\mathcal{D})$ *is defined by:*

$$
\begin{array}{lll}
S_1 \parallel S_2 = & \{\sigma \in \mathcal{D} \mid & \textit{there exists } \sigma_1 \in S_1 \textit{ and } \sigma_2 \in S_2 \textit{ such that} \\
& & \sigma = \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1 \rangle \cdots \langle c_m, \tilde{H}_m, T \rangle \in \eta(\sigma_1 \parallel \sigma_2), \\
& & (V_{loc}(\sigma_1) \cup V_{loc}(\sigma_2)) \cap V_{ass}(\sigma) = \emptyset \textit{ and for } i \in [1, m] \\
& & (V_{loc}(\sigma_1) \cup V_{loc}(\sigma_2)) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j)\}.
\end{array}
$$

The first condition on variables ensures that local variables of σ, that are the ones used in the derivation of which σ is abstraction, are different from the ones used by assumptions of σ. The second condition ensures that σ is an abstraction of a derivation that satisfies condition 3 of Definition 2.4 (Compatibility).

Example 2.5 We now consider an application of Definition 2.9, by using the two abstract sequences of Example 2.4, showing that this composition gives us the sequence in Example 2.3. First of all, we compose the abstract sequences $\alpha(\delta)$ and $\alpha(\gamma)$ of Example 2.4 by using Definition 2.6 and obtaining (among others) the following interleaved sequence:

$$
\begin{array}{ll}
\langle \mathtt{true}, \emptyset, \{B =< C\#3, B =< C\#4\}, C = 7 \rangle & g(a) \\
\langle C = 7, \{B =< C\#5\}, \{B =< C\#3, B =< C\#4\}, C = 7 \rangle & a(h) \\
\langle C = 7, \emptyset, \{B =< C\#3, B =< C\#4\}, C = 7 \rangle & b(h) \\
\langle C = 7, \emptyset, \{B =< C\#3, B =< C\#4\}, (A = C, C = 7) \rangle & c(h) \\
\langle (A = C, C = 7), \emptyset, \{B =< C\#3, B =< C\#4\}, (A = C, C = 7) \rangle & d(h) \\
\langle (A = C, C = 7), \emptyset, \{B =< C\#3, B =< C\#4\}, (B = C, A = C, C = 7) \rangle & e(h) \\
\langle (B = C, A = C, C = 7), \{B =< C\#3, B =< C\#4\}, \{trs@1, 5\} \rangle & f \text{ and } h
\end{array}
$$

where $g(a)$ means that the tuple g and the stable atoms of tuple (a) have been used (and analogously for the other steps), until the last step of interleaving, that uses f and h, closes the composition. The application of Definition 2.8, by also using Definition 2.7, substitutes the constraint labelled by the identifier $\#5$ for the one labelled by $\#3$. This is represented by crossing both constraints.

$$
\begin{array}{ll}
\langle \mathtt{true}, \emptyset, \{\cancel{B =< C\#3}, B =< C\#4\}, C = 7 \rangle & g(a) \\
\langle C = 7, \{\cancel{B =< C\#\ \not{5} \rightarrow 3}\}, \{\cancel{B =< C\#3}, B =< C\#4\}, C = 7 \rangle & a(h) \\
\langle C = 7, \emptyset, \{\cancel{B =< C\#3}, B =< C\#4\}, C = 7 \rangle & b(h) \\
\langle C = 7, \emptyset, \{\cancel{B =< C\#3}, B =< C\#4\}, (A = C, C = 7) \rangle & c(h) \\
\langle (A = C, C = 7), \emptyset, \{\cancel{B =< C\#3}, B =< C\#4\}, (A = C, C = 7) \rangle & d(h) \\
\langle (A = C, C = 7), \emptyset, \{\cancel{B =< C\#3}, B =< C\#4\}, (B = C, A = C, C = 7) \rangle & e(h) \\
\langle (B = C, A = C, C = 7), \{\cancel{B =< C\#3}, B =< C\#4\}, \{trs@1, \not{5} \rightarrow 3\} \rangle & f \text{ and } h
\end{array}
$$

Note that another application of Definition 2.8 is possible by satisfying the assumption with the stable atom $B =< C\#4$. Definition 2.8 then also produces, among others, the

following sequence:

$$\begin{array}{ll}
\langle \mathtt{true}, \emptyset, \{B =< C\#4[3]\}, C = 7\rangle & g(a) \\
\langle C = 7, \emptyset, \{B =< C\#4[3]\}, C = 7\rangle & a(h) \\
\langle C = 7, \emptyset, \{B =< C\#4[3]\}, C = 7\rangle & b(h) \\
\langle C = 7, \emptyset, \{B =< C\#4[3]\}, (A = C, C = 7)\rangle & c(h) \\
\langle (A = C, C = 7), \emptyset, \{B =< C\#4[3]\}, (A = C, C = 7)\rangle & d(h) \\
\langle (A = C, C = 7), \emptyset, \{B =< C\#4[3]\}, (B = C, A = C, C = 7)\rangle & e(h) \\
\langle (B = C, A = C, C = 7), \{B =< C\#4[3]\}, \{trs@1, 3[4]\}\rangle & f \text{ and } h
\end{array}$$

which is equal to the one found in Example 2.3. Here $4[3]$ represents the two possibilities, namely, the constraint labelled by $\#5$ is satisfied by the one labelled by $\#3$ or by $\#4$, respectively. □

Using this notion of composition of sequences we can show that the semantics $\mathcal{S}_P$ is compositional. Before proving the compositionality theorem we need some technical lemmata.

Now four main lemmas, which are directly used in the theorem, and other minor lemmata, which are used to prove the main lemmata, are given.

Let us introduce some further notations that are used in the following lemmas. Given a sequence γ, where $\gamma \in \mathcal{S}eq \cup \mathcal{D}$, we will denote by $length(\gamma)$, $Inc(\gamma)$, $Ass(\gamma)$ and $Stable(\gamma)$ the length of the sequence γ, the set of the input constraints of γ, the set of non-identified assumptions of γ and the set of non-identified atoms in the last goal of γ respectively. We denote by ε the empty sequence. Moreover, let $\delta \in \mathcal{S}eq$ be a sequence of derivation steps

$$\delta = \langle \tilde{B}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{B}_2, d_1, T_2, n_1'\rangle \ldots \langle \tilde{B}_m, c_m, T_m, n_m, \emptyset, \tilde{B}_m, c_m, T_m, n_m\rangle.$$

We denote by $InG(\delta)$, $Intok(\delta)$ and $Inid(\delta)$ the identified goal $\tilde{B}_1$, the token set T_1 and the counter n_1, respectively.

Finally, we denote by $Aloc(\delta)$ the set of the CHR-atoms in the (renamed) clauses, used in a derivation represented by δ.

Now, let $\tilde{W}$ be a set of identified CHR-constraints, such that for each $i \in id(\tilde{W})$ and $j \in id(\tilde{B}_1)$, we can say that $i \leq n_1$ and $i \neq j$. We denote by $\delta \oplus \tilde{W}$ the sequence

$$\langle (\tilde{B}_1, \tilde{W}), c_1, T_1, n_1, \tilde{K}_1, (\tilde{B}_2, \tilde{W}), d_1, T_2, n_1'\rangle \cdots$$
$$\langle (\tilde{B}_m, \tilde{W}), c_m, T_m, n_m, \emptyset, (\tilde{B}_m, \tilde{W}), c_m, T_m, n_m\rangle.$$

The following first main lemma states that, considering a sequence δ in a concrete semantics, the free variables in the assumptions and the local variables in δ are the same as the ones in the abstraction of δ.

Lemma 2.1 *Let G be a goal, $\delta \in \mathcal{S}'_P(G)$ and let $\sigma = \alpha(\delta)$. Then $V_r(\delta) = V_r(\sigma)$ holds, where $r \in \{\, ass,\ loc \,\}$.*

Proof Let us consider the following two sequences (where $\tilde{G}_1$ is an identified version of G):

$$\delta = \langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1 \rangle \dots \langle \tilde{G}_m, c_m, T_m, n_m, \emptyset, \tilde{G}_m, c_m, T_m, n_m \rangle$$

and

$$\sigma = \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1 \rangle \dots \langle c_m, \tilde{H}_m, T_m \rangle,$$

where $\tilde{H}_m = \tilde{G}_m$.

The proof follows:

$$V_{ass}(\delta) = \bigcup_{i=1}^{m-1} Fv(K_i) = V_{ass}(\sigma) \tag{2.2}$$

Now, we will prove that $V_{loc}(\delta) = V_{loc}(\sigma)$. The proof is by induction on $m = length(\delta)$.

We now recall the definitions of V_{loc}:

$$V_{loc}(t) = Fv(G_2, d_1) \setminus Fv(G_1, c_1, K_1)$$

$$V_{loc}(\delta) = \bigcup_{i=1}^{m-1} Fv(G_{i+1}, d_i) \setminus Fv(G_i, c_i, K_i)$$

$$V_{loc}(\sigma) = (V_{constr}(\sigma) \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup Fv(G))$$

$m = 1$) In this case $\delta = \langle \tilde{G}, c, T, n, \emptyset, \tilde{G}, c, T, n \rangle$, $\sigma = \langle c, \tilde{G}, T \rangle$, and therefore, by definition $V_{loc}(\delta) = V_{loc}(\sigma) = \emptyset$.

$m \geq 1$) Let

$$\delta = \langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1 \rangle \langle \tilde{G}_2, c_2, T_2, n_2, \tilde{K}_2, \tilde{G}_3, d_2, T_3, n'_2 \rangle \cdots \langle \tilde{G}_m, c_m, T_m, n_m, \emptyset, \tilde{G}_m, c_m, T_m, n_m \rangle.$$

By definition of $\mathcal{S}'_P(G)$, there exists $\delta' \in \mathcal{S}'_P(G_2)$ such that

$$t = \langle \tilde{G}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1 \rangle$$

is compatible with δ' and $\delta = t \cdot \delta' \in Seq$.

By inductive hypothesis, we can say that $V_{loc}(\delta') = V_{loc}(\sigma')$, where $\sigma' = \alpha(\delta')$. Moreover, by definition of function α, $\sigma = \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1 \rangle \cdot \sigma'$, where $\tilde{H}_1$ is the set consisting of all the identified atoms that are stable in δ.

By definition of V_{loc} and by inductive hypothesis

$$\begin{aligned} V_{loc}(\delta) &= \bigcup_{i=1}^{m-1} Fv(G_{i+1}, d_i) \setminus Fv(G_i, c_i, K_i) \\ &= V_{loc}(\delta') \cup (Fv(G_2, d_1) \setminus Fv(G_1, c_1, K_1)) \\ &= V_{loc}(\sigma') \cup (Fv(G_2, d_1) \setminus Fv(G_1, c_1, K_1)). \end{aligned} \quad (2.3)$$

Moreover, by definition of $V_{loc}(\sigma)$ and since $V_{stable}(\sigma) = V_{stable}(\sigma')$, we can say that

$$V_{loc}(\sigma') = (V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma') \cup Fv(G_2)). \quad (2.4)$$

Therefore, considering the equations (2.3) and (2.4), using the properties of $\cup$ and observing that $Fv(G_2) \cap Fv(G_1, c_1, K_1) = Fv(G_2) \cap Fv(G_1, K_1)$ because of the behaviour of $Solve'$ and $Apply'$[2] we are in position to write:

$$\begin{aligned} V_{loc}(\delta) = & ((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma') \cup Fv(G_2))) \\ & \cup (Fv(G_2) \setminus Fv(G_1, K_1)) \cup (Fv(d_1) \setminus Fv(G_1, c_1, K_1)). \end{aligned} \quad (2.5)$$

[2] **Solve'** : $Fv(G_2) \cap Fv(G_1, c_1, K_1) = Fv(G_2) \cap Fv(G_1)$;
Apply' : fresh variable of the rule can not be in c_1.

Now, let $x \in Fv(K_1)$. By definition $x \in Fv(t)$, since t is compatible with δ' and by point 1 of Definition 2.4 (Compatibility), that is $V_{loc}(\delta') \cap Fv(t) = \emptyset$, we can say that $x \notin V_{loc}(\delta') = V_{loc}(\sigma')$ and therefore considering (2.4) we can add x to $V_{ass}(\sigma') \cup Fv(G_2)$ without any problem. Then by (2.5)

$$\begin{aligned} V_{loc}(\delta) \;&=\; ((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup Fv(G_2))) \\ &\quad \cup (Fv(G_2) \setminus Fv(G_1, K_1)) \;\cup\; (Fv(d_1) \setminus Fv(G_1, c_1, K_1)). \end{aligned} \tag{2.6}$$

We will now only consider the first part of the equation found in (2.6): by properties of $\cup$, and considering that the variables that we can add using $Fv(G_2) \cap Fv(G_1, K_1)$ instead of $Fv(G_2)$ are yet added by $Fv(G_2) \setminus Fv(G_1, K_1)$, we can say that

$$\begin{aligned} &((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup Fv(G_2))) \;\cup \\ &(Fv(G_2) \setminus Fv(G_1, K_1)) \;= \\ &((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup (Fv(G_2) \cap Fv(G_1, K_1)))) \;\cup \\ &(Fv(G_2) \setminus Fv(G_1, K_1)) \;= \\ &((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup (Fv(G_2) \cap Fv(G_1)))) \;\cup \\ &(Fv(G_2) \setminus Fv(G_1, K_1)), \end{aligned} \tag{2.7}$$

where the last equality follows by observing that $Fv(K_1) \subseteq V_{ass}(\sigma)$.

Now let $x \in Fv(G_1) \setminus Fv(G_2)$ and let us assume that $x \in V_{constr}(\sigma') \cup V_{stable}(\sigma)$. Then $x \notin V_{loc}(\delta') = V_{loc}(\sigma')$ because $x \in Fv(t)$ and by Definition 2.4 point 1 (Compatibility) $V_{loc}(\delta') \cap Fv(t) = \emptyset$. Therefore since $x \notin Fv(G_2)$, by considering the equation found in (2.4) we can say that $x \in V_{ass}(\sigma') \subseteq V_{ass}(\sigma)$. According to the previous results and through (2.6) and (2.7), we can say that

$$\begin{aligned} V_{loc}(\delta) \;&=\; ((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup Fv(G_1))) \\ &\quad \cup (Fv(G_2) \setminus Fv(G_1, K_1)) \;\cup\; (Fv(d_1) \setminus Fv(G_1, c_1, K_1)). \end{aligned} \tag{2.8}$$

Now let $x \in (Fv(d_1) \setminus Fv(c_1)) \cap V_{ass}(\sigma')$. Since by point 2 of Definition 2.4

(Compatibility) $V_{loc}(t) \cap V_{ass}(\sigma') = \emptyset$, we can say that $x \in Fv(G_1, K_1)$. Then

$$\begin{aligned} Fv(d_1) \setminus Fv(G_1, c_1, K_1) &= (Fv(d_1) \setminus Fv(c_1)) \setminus Fv(G_1, K_1) \\ &= (Fv(d_1) \setminus Fv(c_1)) \setminus (Fv(G_1, K_1) \cup V_{ass}(\sigma')) \\ &= (Fv(d_1) \setminus Fv(c_1)) \setminus (Fv(G_1) \cup V_{ass}(\sigma)). \end{aligned}$$

Considering the previous result we can further say that

$$\begin{aligned} V_{loc}(\delta) &= ((V_{constr}(\sigma) \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup Fv(G_1))) \cup \\ & \quad (Fv(G_2) \setminus Fv(G_1, K_1)). \end{aligned} \tag{2.9}$$

Finally you may observe that if $x \in Fv(G_2) \setminus Fv(G_1, K_1)$, then an $Apply'$ step occurred, $x \in V_{loc}(t)$ and therefore, by Definition 2.4 of compatibility at point 2, $x \notin V_{ass}(\sigma)$. Moreover, according to point 3 of Definition 2.4 (Compatibility), that says for $i \in [2, m], V_{loc}(t) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j)$, we can conclude that $x \in V_{constr}(\sigma) \cup V_{stable}(\sigma)$. In fact, if no derivation step in δ uses an atom $g \in G_2$ containing the variable x, then $x \in Fv(G_m) = V_{stable}(\sigma)$. Otherwise, by definition of the derivation step, there is a least index $k \in [1, m = 1]$ such that $x \in Fv(d_k)$ and according to point 3 of Definition 2.4 (compatibility) $x \notin Fv(c_k)$, so x will appear in $Fv(d_k) \setminus Fv(c_k) \subseteq V_{constr}(\sigma)$. Then as seen in (2.9), by the previous result and by definition of V_{loc},

$$V_{loc}(\delta) = ((V_{constr}(\sigma) \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup Fv(G_1))) = V_{loc}(\sigma)$$

the thesis thus holds.

□

The identifiers are introduced to the constraints only to distinguish two different occurrences of the same atom, so we can freely rename them without any unforeseen difficulty. We recall that a renaming is a substitution of the form $[j_1/i_1, \ldots, j_o/i_o]$, where $j_1, \ldots, j_o$ are distinct identification values and $i_1, \ldots, i_o$ is a permutation of $j_1, \ldots, j_o$. We will use $\rho,\ \rho', \ldots$ to denote renamings.

Definition 2.10 (Index rename) *Let $\sigma, \sigma_1, \sigma_2 \in \mathcal{D}$ and let $S_1, S_2 \in \wp(\mathcal{D})$.*

- *Let* $\rho = [j_1/i_1, \ldots, j_o/i_o]$ *be a renaming.* $\sigma\rho$ *is the sequence obtained from* σ, *by substituting each occurrence of the identification value* j_l *with the corresponding* i_l, *for* $l \in [1, o]$.

- $\sigma_1 \simeq \sigma_2$ *if there exists a renaming* ρ *such that* $\sigma_1 = \sigma_2\rho$.

- $S_1 \ll S_2$ *if for each* $\sigma_1 \in S_1$ *there exists* $\sigma_2 \in S_2$ *such that* $\sigma_1 \simeq \sigma_2$.

- $S_1 \simeq S_2$ *if* $S_1 \ll S_2$ *and* $S_2 \ll S_1$.

We can also note that if $\sigma_1 \simeq \sigma_2$ there therefore exists a renaming $\rho = [i_1/j_1, \ldots i_o/j_o]$ such that $\sigma_1 = \sigma_2\rho$. There also exists a renaming $\rho^{-1}[j_1/i_1, \ldots j_o/i_o]$ such that $\sigma_1\rho^{-1} = \sigma_2$.

Now, four minor lemmata, which are used to prove the second main lemma, are introduced. The proof of the following three minor lemmas is straightforward by definition of derivation and of $\simeq$.

The following lemma states that we can obtain two concrete sequences that differ only from the same fixed subset of tokens in each tuple.

Lemma 2.2 *Let* G *be a goal,* $\delta \in \mathcal{S}'_P(G)$ *such that*

$$\begin{aligned}\delta = \; & \langle \tilde{G}, c_1, T_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T_2, n'_1\rangle\langle \tilde{G}_2, c_2, T_2, n_2, \tilde{K}_2, \tilde{G}_3, d_2, T_3, n'_2\rangle \\ & \cdots \langle \tilde{G}_m, c_m, T_m, n_m, \emptyset, \tilde{G}_m, c_m, T_m, n_m\rangle,\end{aligned}$$

where $\tilde{G}$ *is an identified version of* G. *Let* $T'_1 \subseteq T_1$. *Then there exists a derivation* $\delta' \in \mathcal{S}'_P(G)$

$$\begin{aligned}\delta' = \; & \langle \tilde{G}, c_1, T'_1, n_1, \tilde{K}_1, \tilde{G}_2, d_1, T'_2, n'_1\rangle\langle \tilde{G}_2, c_2, T'_2, n_2, \tilde{K}_2, \tilde{G}_3, d_2, T'_3, n'_2\rangle \\ & \cdots \langle \tilde{G}_m, c_m, T'_m, n_m, \emptyset, \tilde{G}_m, c_m, T'_m, n_m\rangle,\end{aligned}$$

such that $T_m \setminus T'_m = T_1 \setminus T'_1$.

The following lemma proves that every concrete sequence δ, that is obtained from the goal (H, G), where the first step is made by an $Apply'$ rule, can be obtained, with the sole exception of first tuple and an index renaming, from the goal H, assuming in the first step the constraints in G, that are all used in the $Apply'$ rule.

Lemma 2.3 *Let H, G be goals and let $\delta \in \mathcal{S}'_P(H,G)$ such that*

$$\begin{aligned}\delta &= \langle(\tilde{H},\tilde{G}),c_1,T_1,n_1,\tilde{K}_1,\tilde{R}_2,d_1,T_2,n'_1\rangle\langle\tilde{R}_2,c_2,T_2,n_2,\tilde{K}_2,\tilde{R}_3,d_2,T_3,n'_2\rangle \\ &\quad\cdots\langle\tilde{R}_m,c_m,T_m,n_m,\emptyset,\tilde{R}_m,c_m,T_m,n_m\rangle \\ &= \langle(\tilde{H},\tilde{G}),c_1,T_1,n_1,\tilde{K}_1,\tilde{R}_2,d_1,T_2,n'_1\rangle\cdot\delta_1\end{aligned}$$

where $\tilde{H} = (\tilde{H}', \tilde{H}'')$ and $\tilde{G}$ are identified versions of $H = (H', H'')$ and G, respectively, $H'' \neq \emptyset$ and the first tuple of the sequence δ represents a derivation step s, which is made by an **Apply'** *rule and uses only all the atoms in $(\tilde{H}'', \tilde{G})$. There then exists a derivation $\delta' \in \mathcal{S}'_P(H)$,*

$$\begin{aligned}\delta' &= \langle\tilde{H},c_1,T_1,n_1,\tilde{K}'_1\cup\tilde{G}',\tilde{R}'_2,d_1,T'_2,l'_1\rangle\langle\tilde{R}'_2,c_2,T'_2,l_2,\tilde{K}'_2,\tilde{R}'_3,d_2,T'_3,l'_2\rangle \\ &\quad\cdots\langle\tilde{R}'_m,c_m,T'_m,l_m,\emptyset,\tilde{R}'_m,c_m,T'_m,l_m\rangle, \\ &= \langle\tilde{H},c_1,T_1,n_1,\tilde{K}'_1\cup\tilde{G}',\tilde{R}'_2,d_1,T'_2,l'_1\rangle\cdot\delta'_1\end{aligned}$$

and there further exists a renaming ρ such that $\delta'_1 = \delta_1\rho$, $\tilde{K}'_1 = \tilde{K}_1\rho$ and $\tilde{G}' = \tilde{G}\rho$.

The next Lemma says that if we can add a set of identified constraints to the goal store (such that there are sufficient free indices) of a concrete sequence then the sequence obtained is a concrete sequence too.

Lemma 2.4 *Let G be a goal, $\tilde{W}$ be a set of identified atoms and let $\delta \in \mathcal{S}'_P(G)$ such that $\delta \oplus \tilde{W}$ is defined and $Fv(\tilde{W}) \cap V_{loc}(\delta) = \emptyset$. Then $\delta \oplus \tilde{W} \in \mathcal{S}'_P(G, chr(\tilde{W}))$.*

The following not immediate lemma proves that we can obtain the same concrete semantics both from a goal and from one part of it, where at most only different identifiers can be used. It is used in Lemma 2.6. We then impose some characteristics to arrive at the desired result.

Lemma 2.5 *Let P be a program and let H and G be two goals such that there exists a derivation step*

$$s = \langle(\tilde{H},\tilde{G}),c_1,T_1\rangle_{n_1} \longrightarrow_P^{K_1} \langle(\tilde{B},\tilde{G}),d_1,T_2\rangle_{n'_1},$$

where $\tilde{H}$ and $\tilde{G}$ are identified versions of H and G respectively and only the atoms in $\tilde{H}$ are rewritten in s.
Assume that there exists $\delta \in \mathcal{S}'_P(H,G)$ such that $\delta = t \cdot \delta' \in Seq$, where

$$t = \langle(\tilde{H},\tilde{G}), c_1, T_1, n_1, \tilde{K}_1, (\tilde{B},\tilde{G}), d_1, T_2, n'_1\rangle,$$

$\delta' \in \mathcal{S}'_P(B,G)$ and t is compatible with δ'. Moreover, assume that there exists $\delta'_1 \in \mathcal{S}'_P(B)$ and $\delta'_2 \in \mathcal{S}'_P(G)$, such that

1. *$InG(\delta'_1) = \tilde{B}$, $InG(\delta'_2) = \tilde{G}$, $Intok(\delta'_1) = Intok(\delta'_2) = T_2$,*

   for $i \in \{1,2\}$, $Inid(\delta'_i) \geq n'_1$, $V_{loc}(\delta'_i) \subseteq V_{loc}(\delta')$ and $Inc(\delta'_i) \subseteq Inc(\delta')$.

2. *$Ass(\delta'_1) \subseteq Ass(\delta') \cup Aloc(\delta'_2) \cup InG(\delta'_2)$ and $Ass(\delta'_2) \subseteq Ass(\delta') \cup Aloc(\delta'_1) \cup InG(\delta'_1)$,*

3. *$\alpha(\delta'_1) \parallel \alpha(\delta'_2)$ is defined and that there exists $\sigma' \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta'_2))$ such that $\sigma' \simeq \alpha(\delta')$.*

Thus, $\delta_1 = t' \cdot \delta'_1 \in S'_P(H)$, where $t' = \langle\tilde{H}, c_1, T_1, n_1, \tilde{K}_1, \tilde{B}, d_1, T_2, n'_1\rangle$, $\alpha(\delta_1) \parallel \alpha(\delta'_2)$ is defined and there exists $\sigma \in \eta(\alpha(\delta_1) \parallel \alpha(\delta'_2))$ such that $\sigma \simeq \alpha(\delta)$.

Proof In the following proof we assume that

$$\begin{aligned}
\delta'_1 &= \langle\tilde{B}_1, e_1, T_2, h_1, \tilde{M}_1, \tilde{B}_2, f_1, T'_2, h'_1\rangle \cdots \langle\tilde{B}_l, e_l, T'_l, h_l, \emptyset, \tilde{B}_l, e_l, T'_l, h_l\rangle \\
\delta'_2 &= \langle\tilde{G}_1, r_1, S_2, j_1, \tilde{N}_1, \tilde{G}_2, s_1, S'_2, j'_1\rangle \cdots \langle\tilde{G}_p, r_p, S'_p, j_p, \emptyset, \tilde{G}_p, r_p, S'_p, j_p\rangle \\
\delta' &= \langle\tilde{R}_2, c_2, T_2, n_2, \tilde{K}_2, \tilde{R}_3, d_2, T_3, n'_2\rangle \cdots \langle\tilde{R}_m, c_m, T_m, n_m, \emptyset, \tilde{R}_m, c_m, T_m, n_m\rangle,
\end{aligned}$$

where $B_1 = \tilde{B}$, $G_1 = \tilde{G}$, $R_2 = (\tilde{B},\tilde{G})$ and $e_l = r_p = c_m$ (our sequence needs the last condition to close the composition (cmp. Definition 2.6)). The following then holds.

(a) t' represents the derivation step $s' = \langle\tilde{H}, c_1, T_1\rangle_{n_1} \longrightarrow_P^{K_1} \langle\tilde{B}, d_1, T_2\rangle_{n'_1}$. The proof is straightforward, by observing that t represents the derivation step

$$s = \langle(\tilde{H},\tilde{G}), c_1, T_1\rangle_{n_1} \longrightarrow_P^{K_1} \langle(\tilde{B},\tilde{G}), d_1, T_2\rangle_{n'_1},$$

which uses only atoms in $\tilde{H}$.

(b) $\delta_1 \in \mathcal{S}'_P(H)$**.** By considering the previous point, by hypothesis and by definition of $\mathcal{S}'_P(H)$, we have to prove that $\delta_1 \in \mathcal{S}eq$ and that Definition 2.4 is satisfied, according to the hypothesis $InG(\delta'_1) = \tilde{B}$, $Intok(\delta'_1) = T_2$, $Inid(\delta'_1) = h_1 \geq n'_1$ and $Inc(\delta'_1) \subseteq Inc(\delta')$ (and then $CT \models instore(\delta'_1) \rightarrow instore(\delta')$). Moreover, since $\delta = t \cdot \delta' \in Seq$, we can say that $CT \models instore(\delta') \rightarrow d_1$ and therefore $CT \models instore(\delta'_1) \rightarrow d_1$ through transitivity. Then we have only to prove that t' is compatible with δ'_1 and so that the three conditions of Definition 2.4 hold.

The following points then hold:

1. According to the hypothesis $V_{loc}(\delta'_1) \subseteq V_{loc}(\delta')$ and the construction $Fv(t') \subseteq Fv(t)$. Then $V_{loc}(\delta'_1) \cap Fv(t') \subseteq V_{loc}(\delta') \cap Fv(t) = \emptyset$, where the last equality follows since t is compatible with δ'.

2. We can say that:

$$
\begin{aligned}
& V_{loc}(t') \cap V_{ass}(\delta'_1) \subseteq \\
& \qquad \text{(since } V_{loc}(t') = V_{loc}(t) \text{ through the construction and} \\
& \qquad Ass(\delta'_1) \subseteq Ass(\delta') \cup Aloc(\delta'_2) \cup InG(\delta'_2) \text{ as per hypothesis)} \\
& V_{loc}(t) \cap (V_{ass}(\delta') \cup V_{loc}(\delta'_2) \cup Fv(G)) \subseteq \\
& \qquad \text{(since through the hypothesis } V_{loc}(\delta'_2) \subseteq V_{loc}(\delta') \text{ and} \\
& \qquad V_{loc}(t) \cap Fv(G) = \emptyset \text{ as per construction)} \\
& V_{loc}(t) \cap (V_{ass}(\delta') \cup V_{loc}(\delta')) = \\
& \qquad \text{(since } t \text{ is compatible with } \delta' \text{ and } V_{loc}(t) \subseteq Fv(t)) \\
& \emptyset
\end{aligned}
$$

3. We have to prove that with $1 \leq i \leq l$,

$$V_{loc}(t') \cap Fv(e_i) \subseteq \bigcup_{j=1}^{i-1} Fv(f_j) \cup Fv(d_1).$$

First of all, observe that since t is compatible with δ', per construction $x \in V_{loc}(t') = V_{loc}(t)$ and for $i \in \{1, 2\}$, $V_{loc}(\delta'_i) \subseteq V_{loc}(\delta')$ we can say that

$$x \notin Fv(H) \cup Fv(G) \cup V_{loc}(\delta'_1) \cup V_{loc}(\delta'_2) \cup V_{ass}(\delta'). \tag{2.10}$$

Moreover, through the hypothesis $Inc(\delta'_1) \subseteq Inc(\delta')$, there therefore exists the least index $h \in [2, m]$ such that $e_i = c_h$. Therefore, since, by construction, $V_{loc}(t') = V_{loc}(t)$ and, by hypothesis, t is compatible with δ', considering a generic variable $x \in V_{loc}(t') \cap Fv(e_i)$, we have that

$$x \in \bigcup_{j=1}^{h-1} Fv(d_j).$$

Then, to prove the thesis, we have to prove that if $x \in \bigcup_{j=1}^{h-1} Fv(d_j)$ then $x \in \bigcup_{j=1}^{i-1} Fv(f_j) \cup Fv(d_1)$. If $x \in Fv(d_1)$ then the thesis holds.

Let us assume that $x \notin Fv(d_1)$, $x \in \bigcup_{j=2}^{h-1} Fv(d_j)$ and let k be the least index $j \in [2, h-1]$ such that $x \in Fv(d_j)$. Now, we have two possibilities:

(a) d_k is an output constraint of δ'_1, i.e. there exists $j \in [1, i-1]$ such that $d_k = f_j$, then we have the proof.

(b) d_k is an output constraint of δ'_2, namely there exists $w \in [1, p]$ such that $d_k = s_w$. Then, since k is the least index j such that $x \in Fv(d_j)$, $x \in V_{loc}(t)$ and, by hypothesis, t is compatible with δ', we have that $x \notin Fv(c_k)$ and therefore $x \notin Fv(r_w)$.

Moreover, since by (2.10) and by point 2 of the hypothesis, $x \notin Fv(G) \cup V_{loc}(\delta'_2) \cup V_{ass}(\delta'_2)$, we can say that $x \notin Fv(G_w)$.

Then by definition of derivation step, we have a contradiction, since $x \in Fv(s_w) \setminus (Fv(r_w) \cup Fv(G_w) \cup V_{loc}(\delta'_2) \cup V_{ass}(\delta'_2))$.

(c) $\alpha(\delta_1) \parallel \alpha(\delta'_2)$ **is defined.** Now we consider Definition 2.6. First of all, observe that $id(\delta_1) \cap id(\delta'_2) = \emptyset$ since $\alpha(\delta'_1) \parallel \alpha(\delta'_2)$ is defined (and therefore $id(\delta'_1) \cap id(\delta'_2) = \emptyset$) and since by hypothesis $Inid(\delta'_2) = j_1 \geq n'_1$. Then we only have to prove that

$$(V_{loc}(\alpha(\delta_1)) \cup Fv(H)) \cap (V_{loc}(\alpha(\delta'_2)) \cup Fv(G)) = Fv(H) \cap Fv(G).$$

By Lemma 2.1

$$V_{loc}(\alpha(\delta_1)) = V_{loc}(\alpha(\delta'_1)) \cup V_{loc}(t'). \tag{2.11}$$

and since $\alpha(\delta'_1) \parallel \alpha(\delta'_2)$ is defined, we can say that

$$(V_{loc}(\alpha(\delta'_1)) \cup Fv(B)) \cap (V_{loc}(\alpha(\delta'_2)) \cup Fv(G)) = Fv(B) \cap Fv(G).$$

From the above, we can therefore maintain that

$$V_{loc}(\alpha(\delta_1')) \cap (V_{loc}(\alpha(\delta_2')) \cup Fv(G)) = \emptyset. \tag{2.12}$$

Now observe that, since t is compatible with δ' (Definition 2.4 at point 1): $V_{loc}(\delta') \cap Fv(t) = \emptyset$ holds, $V_{loc}(t') = V_{loc}(t)$ per construction and, according to Lemma 2.1, we can conclude that $V_{loc}(t') \cap V_{loc}(\alpha(\delta')) = \emptyset$. Furthermore, through the hypothesis $V_{loc}(\alpha(\delta_2')) \subseteq V_{loc}(\alpha(\delta'))$ and according to the definition of t, we have that $Fv(G) \cap V_{loc}(t') = Fv(G) \cap V_{loc}(t) = \emptyset$. Then

$$\begin{aligned} &V_{loc}(\alpha(\delta_1)) \cap (V_{loc}(\alpha(\delta_2')) \cup Fv(G)) = \\ &(V_{loc}(\alpha(\delta_1')) \cup V_{loc}(t')) \cap (V_{loc}(\alpha(\delta_2')) \cup Fv(G)) = \emptyset. \end{aligned} \tag{2.13}$$

Finally, since t is compatible with δ' as in Definition 2.4 point 1: $V_{loc}(\delta') \cap Fv(t) = \emptyset$), as per construction $Fv(H) \subseteq Fv(t)$ and the hypothesis $V_{loc}(\alpha(\delta_2')) \subseteq V_{loc}(\alpha(\delta'))$ we can say that

$$Fv(H) \cap V_{loc}(\alpha(\delta_2')) \subseteq Fv(H) \cap V_{loc}(\alpha(\delta')) = \emptyset \tag{2.14}$$

and that the thesis thus holds for (2.11), (2.13), (2.14) and for the properties of set operators.

(d) There exists $\sigma \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2'))$ **such that** $\sigma \simeq \alpha(\delta)$**.** By inductive hypothesis $\alpha(\delta') \simeq \sigma' \in \eta(\alpha(\delta_1') \parallel \alpha(\delta_2'))$. Through the definition of $\simeq$, there exists a renaming ρ such that

$$\alpha(\delta') \quad = \quad \sigma'\rho. \tag{2.15}$$

Since by hypothesis $InG(\delta') = (InG(\delta_1'), InG(\delta_2'))$ and $Intok(\delta_1') = Intok(\delta_2') = Intok(\delta')$, without loss of generality, we can assume that

$$t'\rho = t'. \tag{2.16}$$

Moreover, by definition of $\parallel$ there exists $\sigma_1 \in \alpha(\delta_1') \parallel \alpha(\delta_2')$ such that $\sigma' \in \eta(\{\sigma_1\})$ and

$$\langle c_1, \tilde{K}_1, (\tilde{J}_1 \cup \tilde{Y}_1), d_1 \rangle \cdot \sigma_1 \in \alpha(\delta_1) \parallel \alpha(\delta_2'),$$

where $\tilde{J}_1$ is the set of atoms in $\tilde{H}$ which are not rewritten in δ_1 and $\tilde{Y}_1$ the set of atoms in $\tilde{G}$ which are not rewritten in δ'_2.

Let us denote the following symbols where:

- $\tilde{J}_2$ is the set of atoms in $\tilde{B}$ which are not rewritten in δ'_1,
- $\tilde{W}_1$ is the set of atoms in $(\tilde{H}, \tilde{G})$ which are not rewritten in δ and
- $\tilde{W}_2$ is the set of atoms in $(\tilde{B}, \tilde{G})$ which are not rewritten in δ'.

According to the definition of α,

$$\alpha(\delta) \quad = \quad \langle c_1, \tilde{K}_1, \tilde{W}_1, d_1 \rangle \cdot \alpha(\delta'). \tag{2.17}$$

According to the definition of η and since $\sigma' \in \eta(\{\sigma_1\})$,

$$\langle c_1, \tilde{K}_1, (\tilde{J}_1 \cup \tilde{Y}_1) \setminus S, d_1 \rangle \cdot \sigma' \in \eta(\alpha(\delta_1) \parallel \alpha(\delta'_2)), \tag{2.18}$$

where the sets difference $(\tilde{J}_1 \cup \tilde{Y}_1) \setminus S$ considers identification values and S is such that $(\tilde{J}_2 \cup \tilde{Y}_1) \setminus S = \tilde{W}_2$. Since $(\tilde{J}_1 \cup \tilde{Y}_1) \subseteq (\tilde{J}_2 \cup \tilde{Y}_1)$, we can assume that $\tilde{W} = (\tilde{J}_1 \cup \tilde{Y}_1) \setminus S = (\tilde{J}_1 \cup \tilde{Y}_1) \cap \tilde{W}_2$. Then by definition, $\tilde{W}$ contains all and solely the atoms in $(\tilde{H}, \tilde{G})$ which are not rewritten in t and in δ' and therefore $\tilde{W} = \tilde{W}_1$.

Therefore by (2.18)

$$\langle c_1, \tilde{K}_1, \tilde{W}_1, d_1 \rangle \cdot \sigma' \in \eta(\alpha(\delta_1) \parallel \alpha(\delta'_2)).$$

Then

$$\begin{aligned} (\langle c_1, \tilde{K}_1, \tilde{W}_1, d_1 \rangle \cdot \sigma')\, \rho \quad &= \quad \text{(by (2.16))} \\ \langle c_1, \tilde{K}_1, \tilde{W}_1, d_1 \rangle \cdot (\sigma' \rho) \quad &= \quad \text{(by (2.15))} \\ \langle c_1, \tilde{K}_1, \tilde{W}_1, d_1 \rangle \cdot \alpha(\delta') \quad &= \quad \text{(by (2.17))} \\ \alpha(\delta) \end{aligned}$$

and this completes the proof.

□

The following second main lemma states that, fixed a concrete sequence derived by the goal (H, G), there then exist two concrete sequences that are derived from H and G, for which there further exists an abstract composition that is equal to the abstraction of the fixed sequence.

Lemma 2.6 *Let P be a program, H and G be two goals and assume that $\delta \in \mathcal{S}'_P(H,G)$. Then both $\delta_1 \in \mathcal{S}'_P(H)$ and $\delta_2 \in \mathcal{S}'_P(G)$ exist, and $\sigma \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$ such that, for $i = 1,2$, $V_{loc}(\delta_i) \subseteq V_{loc}(\delta)$ and $\sigma \simeq \alpha(\delta)$.*

Proof We can now construct by induction on $l = length(\delta)$, two sequences $\delta \uparrow_{(H,G)}= (\delta_1, \delta_2)$, where $\delta_1 \in \mathcal{S}'_P(H)$, $\delta_2 \in \mathcal{S}'_P(G)$ and $i \in \{1,2\}$ with the following characteristics:

1. $InG(\delta) = (InG(\delta_1), InG(\delta_2)), V_{loc}(\delta_i) \subseteq V_{loc}(\delta), Inid(\delta_i) \geq Inid(\delta), Intok(\delta_i) = Intok(\delta)$ and $Inc(\delta_i) \subseteq Inc(\delta)$ (and therefore $CT \models instore(\delta_i) \rightarrow instore(\delta)$);
2. $Ass(\delta_1) \subseteq Ass(\delta) \cup Aloc(\delta_2) \cup InG(\delta_2)$ and $Ass(\delta_2) \subseteq Ass(\delta) \cup Aloc(\delta_1) \cup InG(\delta_1)$;
3. $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined and $\alpha(\delta) \simeq \sigma \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$ (where identifiers of atoms in σ are resorted with respect to the δ ones).

The two sequences composed using $\parallel$ operator enjoy the above properties. These properties are also needed to apply Lemma 2.5 that we will need to prove the current lemma.

(l=1) In this case $\delta = \langle(\tilde{H}, \tilde{G}), c, T, n, \emptyset, (\tilde{H}, \tilde{G}), c, T, n\rangle$, so

$$\begin{aligned} \delta \uparrow_{(H,G)} &= (\langle\tilde{H}, c, T, n, \emptyset, \tilde{H}, c, T, n\rangle,\ \langle\tilde{G}, c, T, n, \emptyset, \tilde{G}, c, T, n\rangle) \\ &= (\delta_1, \delta_2) \end{aligned}$$

where $\delta_1 \in \mathcal{S}'_P(H)$, $\delta_2 \in \mathcal{S}'_P(G)$. Note that, by definition of sequence, $id(\tilde{H}) \cap id(\tilde{G}) = \emptyset$ and by construction $V_{loc}(\delta_1) = V_{loc}(\delta_2) = \emptyset$, so $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined. Then

$$\begin{aligned} &\alpha(\delta_1) = \langle c, \tilde{H}, T\rangle,\ \alpha(\delta_2) = \langle c, \tilde{G}, T\rangle \text{ and} \\ &\alpha(\delta) = \sigma = \langle c, (\tilde{H}, \tilde{G}), T\rangle \in \alpha(\delta_1) \parallel \alpha(\delta_2). \end{aligned}$$

Furthermore, the following holds

1. $InG(\delta) = (\tilde{H}, \tilde{G}) = (InG(\delta_1), InG(\delta_2)), V_{loc}(\delta) = V_{loc}(\delta_i) = \emptyset$ so $V_{loc}(\delta_i) \subseteq V_{loc}(\delta) = \emptyset$, $Inid(\delta_i) = Inid(\delta)$, $Intok(\delta_i) = T = Intok(\delta)$ and $Inc(\delta_i) = Inc(\delta)$;

2. $Ass(\delta_1) = \emptyset$ so $Ass(\delta_1) \subseteq Ass(\delta) \cup Aloc(\delta_2) \cup InG(\delta_2)$ and $Ass(\delta_2) = \emptyset$ so $Ass(\delta_2) \subseteq Ass(\delta) \cup Aloc(\delta_1) \cup InG(\delta_1)$);
3. $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined and $\alpha(\delta) \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$: the proof is straightforward by definition of $\parallel$.

(l>1) If $\delta \in \mathcal{S}'_P(H,G)$, by definition

$$\delta = \langle(\tilde{H},\tilde{G}), c_1, T_1, n_1, \tilde{K}_1, \tilde{B}_2, d_1, T_2, n'_1\rangle \cdot \delta',$$

where $\tilde{H}$, $\tilde{G}$ and $\tilde{B}_2$ are identified versions of the goals H, G and B_2, respectively, $id(\tilde{H}) \cap id(\tilde{G}) = \emptyset$, $\delta' \in \mathcal{S}'_P(B_2)$ and $t = \langle(\tilde{H},\tilde{G}), c_1, T_1, n_1, \tilde{K}_1, \tilde{B}_2, d_1, T_2, n'_1\rangle$ is compatible with δ'. We recall that, by definition, the tuple t represents a derivation step

$$s = \langle(\tilde{H},\tilde{G}), c_1, T_1\rangle_{n_1} \longrightarrow_P^{K_1} \langle\tilde{B}_2, d_1, T_2\rangle_{n'_1}.$$

Now we distinguish various cases according to the structure of the derivation step s.

Solve' If the derivation step s uses a $Solve'$ rule we can assume, without loss of generality, that $H = (c, H')$ and $\tilde{H} = (c, \tilde{H}')$ so:

$$s = \langle(\tilde{H},\tilde{G}), c_1, T_1\rangle_{n_1} \rightarrow_P^{\emptyset} \langle(\tilde{H}',\tilde{G}), d_1, T_1\rangle_{n_1},$$

$CT \models c_1 \wedge c \leftrightarrow d_1$, $t = \langle(\tilde{H},\tilde{G}), c_1, T_1, n_1, \emptyset, (\tilde{H}',\tilde{G}), d_1, T_1, n_1\rangle$ and $\delta' \in \mathcal{S}'_P(H',G)$. Furthermore, $\alpha(\delta) = \langle c_1, \emptyset, \tilde{W}, d_1\rangle \cdot \alpha(\delta')$ where $\tilde{W}$ is the first stable identified atoms set of $\alpha(\delta')$, because the application of $Solve'$ does not modify the next stable identified atoms set.

By inductive hypothesis there exists $\delta'_1 \in \mathcal{S}'_P(H')$ and $\delta_2 \in \mathcal{S}'_P(G)$ such that $\delta' \uparrow_{(H',G)}= (\delta'_1, \delta_2)$ and $\alpha(\delta') \simeq \sigma' \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta_2))$. We may now define:

$$\delta \uparrow_{(H,G)}= (\delta_1, \delta_2) \text{ where } \delta_1 = \langle\tilde{H}, c_1, T_1, n_1, \emptyset, \tilde{H}', d_1, T_1, n_1\rangle \cdot \delta'_1.$$

By definition, $\langle\tilde{H}, c_1, T_1\rangle_{n_1} \rightarrow_P^{\emptyset} \langle\tilde{H}', d_1, T_1\rangle_{n_1}$ represents a derivation step for H and so we can write the tuple $t' = \langle\tilde{H}, c_1, T_1, n_1, \emptyset, \tilde{H}', d_1, T_1, n_1\rangle$,

$Fv(d_1) \subseteq Fv(H) \cup Fv(c_1)$ and therefore $V_{loc}(t') = \emptyset$, then the following holds:

1. By inductive hypothesis $InG(\delta_2) = \tilde{G}$ and therefore $InG(\delta) = (\tilde{H}, \tilde{G}) = (InG(\delta_1), InG(\delta_2))$. Now, let $i \in \{1, 2\}$. $V_{loc}(\delta_i) \subseteq V_{loc}(\delta')$ through inductive hypothesis and as per construction; $V_{loc}(\delta') = V_{loc}(\delta)$ by previous observation (that is $V_{loc}(t') = \emptyset$), then $V_{loc}(\delta_i) \subseteq V_{loc}(\delta)$.
   $Intok(\delta_i) = T_1 = Intok(\delta)$, $Inid(\delta_1) = n_1 = Inid(\delta)$ and by inductive hypothesis $Inid(\delta_2) \geq Inid(\delta') \geq Inid(\delta)$, where the last inequality follows from the definition of sequence.
   By inductive hypothesis and as per construction $Inc(\delta_i) \subseteq Inc(\delta') \cup \{c_1\} = Inc(\delta)$.
2. By construction $Ass(\delta_1) = Ass(\delta'_1)$ and $Ass(\delta) = Ass(\delta')$ and through inductive hypothesis $Ass(\delta'_1) \subseteq Ass(\delta') \cup Aloc(\delta_2) \cup InG(\delta_2)$. Then $Ass(\delta_1) \subseteq Ass(\delta) \cup Aloc(\delta_2) \cup InG(\delta_2)$.
   Furthermore through inductive hypothesis, $Ass(\delta_2) \subseteq Ass(\delta') \cup Aloc(\delta'_1) \cup InG(\delta'_1)$ and as per construction $Ass(\delta') \cup Aloc(\delta'_1) \cup InG(\delta'_1) = Ass(\delta) \cup Aloc(\delta_1) \cup InG(\delta_1)$.
3. The proof follows according to Lemma 2.5 and by inductive hypothesis.

Apply' - only atoms of H In the derivation step s we use the $Apply'$ rule and we assume that only atoms deriving from $H = (H', H'')$ are used: $H'' \neq \emptyset$ is used by $Apply'$ rule and $\tilde{H} = (\tilde{H}', \tilde{H}'')$.

In this case we can assume that

$$s = \langle (\tilde{H}, \tilde{G}), c_1, T_1 \rangle_{n_1} \rightarrow_P^{K_1} \langle (\tilde{H}', \tilde{B}, \tilde{G}), d_1, T_2 \rangle_{n'_1}$$

so $\delta' \in \mathcal{S}'_P(H', B, G)$ and $t = \langle (\tilde{H}, \tilde{G}), c_1, T_1, n_1, \tilde{K}_1, (\tilde{H}', \tilde{B}, \tilde{G}), d_1, T_2, n'_1 \rangle$.

By inductive hypothesis there exist $\delta'_1 \in \mathcal{S}'_P(H', B)$ and $\delta'_2 \in \mathcal{S}'_P(G)$ such that $\delta' \uparrow_{((H',B),G)} = (\delta'_1, \delta'_2)$ and $\alpha(\delta') \simeq \sigma' \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta'_2))$. As per the definition of $\uparrow$, $Intok(\delta'_2) = T_2 \supseteq T_1$.

Thus, according to Lemma 2.2, there exists a derivation $\delta_2 \in \mathcal{S}'_P(G)$ such that

$$V(\delta_2) = V(\delta'_2), \text{ for } V \in \{length, Aloc, \ Ass, \ V_{loc}, \ Stable\},$$

$$Intok(\delta_2) = T_1 \text{ and } \alpha(\delta') \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta_2)). \tag{2.19}$$

We may then define:

$$\delta \uparrow_{(H,G)} = (\delta_1, \delta_2) \text{ where } \delta_1 = \langle \tilde{H}, c_1, T_1, n_1, \tilde{K}_1, (\tilde{H}', \tilde{B}), d_1, T_2, n'_1 \rangle \cdot \delta'_1$$

By definition $s' = \langle \tilde{H}, c_1, T_1 \rangle_{n_1} \longrightarrow_P^{K_1} \langle (\tilde{H}', \tilde{B}), d_1, T_2 \rangle_{n'_1}$ is a derivation step for H, $t' = \langle \tilde{H}, c_1, K_1, T_1, n_1, (\tilde{H}', \tilde{B}), d_1, T_2, n'_1 \rangle$ represents the derivation step s' and $V_{loc}(t') = V_{loc}(t)$. Now the following holds, with $i \in \{1, 2\}$:

1. By construction $InG(\delta) = (InG(\delta_1), InG(\delta_2)) = (\tilde{H}, \tilde{G})$. Through inductive hypothesis, construction, property of union and by (2.19), $V_{loc}(\delta_i) \subseteq V_{loc}(\delta') \cup V_{loc}(t)$.
   Moreover, through inductive hypothesis, the definition of δ, (2.19) and as per construction $V_{loc}(\delta') \cup V_{loc}(t) = V_{loc}(\delta)$ and $Inc(\delta_i) \subseteq Inc(\delta') \cup \{c_1\} = Inc(\delta)$ so $V_{loc}(\delta_i) \subseteq V_{loc}(\delta)$ and $Inc(\delta_i) \subseteq Inc(\delta)$. $Inid(\delta_1) = n_1 = Inid(\delta)$ and $Inid(\delta_2) \geq Inid(\delta') \geq Inid(\delta)$. Finally, through construction and inductive hypothesis $Intok(\delta_i) = T_1 = Intok(\delta)$.

2. By inductive hypothesis, (2.19) and construction,

$$\begin{aligned} Ass(\delta_1) &= Ass(\delta'_1) \cup \{K_1\} \\ &\subseteq Ass(\delta') \cup Aloc(\delta_2) \cup InG(\delta_2) \cup \{K_1\} \\ &= Ass(\delta) \cup Aloc(\delta_2) \cup InG(\delta_2) \end{aligned}$$

   and

$$\begin{aligned} Ass(\delta_2) &= Ass(\delta'_2) \\ &\subseteq Ass(\delta') \cup Aloc(\delta'_1) \cup InG(\delta'_1) \\ &\subseteq Ass(\delta) \cup Aloc(\delta_1) \cup InG(\delta_1). \end{aligned}$$

3. The proof follows according to Lemma 2.5 and inductive hypothesis.

Apply' - only atoms of G The proof is the same as that of the previous case and is thus omitted.

Apply' - atoms of H and G In the derivation step s we use the $Apply'$ rule and let us further assume that in s some atoms deriving both from H and G are used.

In this case, we can assume that $H = (H', H'')$, $G = (G', G'')$, $H'' \neq \emptyset$, $G'' \neq \emptyset$, $\tilde{H} = (\tilde{H}', \tilde{H}'')$, $\tilde{G} = (\tilde{G}', \tilde{G}'')$ and $(\tilde{H}'', \tilde{G}'')$ are the atoms in the goal $(\tilde{H}, \tilde{G})$, which are used in s. Moreover let $\tilde{G}_1$ be the atoms in the head of the clause used in the derivation step s, such that $\tilde{G}''$ are unified with $\tilde{G}_1$ in s.

$$s = \langle (\tilde{H}, \tilde{G}), c_1, T_1 \rangle_{n_1} \longrightarrow_P^{K_1} \langle (\tilde{H}', \tilde{G}', \tilde{B}), d_1, T_2 \rangle_{n_1'},$$

so $\delta' \in \mathcal{S}'_P(H', G', B)$ and $t = \langle (\tilde{H}, \tilde{G}), c_1, T_1, n_1, \tilde{K}_1, (\tilde{H}', \tilde{G}', \tilde{B}), d_1, T_2, n_1' \rangle$. Moreover $\alpha(\delta) = \langle c_1, \tilde{K}_1, \tilde{W}, d_1 \rangle \cdot \alpha(\delta')$, where $\tilde{W}$ is the set of stable atoms of δ' restricted to the atoms in $(\tilde{H}', \tilde{G}')$.

Using the same arguments of the previous point both $\delta_1' \in \mathcal{S}'_P(H, G'')$ and $\delta_2' \in \mathcal{S}'_P(G')$ exist such that $\delta \uparrow_{((H,G''),G')} = (\delta_1', \delta_2')$.

Now, observe that, according to Lemma 2.3 and the definition of $\uparrow$, there exists $\delta_1 \in \mathcal{S}'_P(H)$ such that $InG(\delta_1) = \tilde{H}$, $Ass(\delta_1) = Ass(\delta_1') \cup \{G''\}$,

$$\alpha(\delta_1') = \langle c_1, \tilde{K}_1, \tilde{W}_1, d_1 \rangle \cdot \sigma_1, \;\; \alpha(\delta_1) = \langle c_1, \tilde{K}_1' \cup \tilde{G}_2, \tilde{W}_1', d_1 \rangle \cdot \sigma_1',$$
$$V(\delta_1) = V(\delta_1') \text{ for } V \in \{Intok,\ Inid,\ V_{loc},\ Inc,\ Stable,\ Aloc\} \quad (2.20)$$

where $\sigma_1 \simeq \sigma_1'$, and $\tilde{K}_1'$, $\tilde{W}_1'$ and $\tilde{G}_2$ are an identified version of K_1, of W_1 and of G'', respectively.

Moreover, since $\delta \in \mathcal{S}'_P(H, G)$ and $V_{loc}(\delta_2') \subseteq V_{loc}(\delta)$, we can say that $Fv(G'') \cap V_{loc}(\delta_2') = \emptyset$ and for each $i \in id(\tilde{G}'')$ and $j \in id(\tilde{G}')$, we can further say that $i \leq n_1$ and $i \neq j$. Then according to Lemma 2.4, we can assume that $\delta_2 = \delta_2' \oplus \tilde{G}'' \in S'_P(G)$. By construction $InG(\delta_2) = \tilde{G}$, $Stable(\delta_2) = Stable(\delta_2') \cup \{G''\}$ and

$$V(\delta_2) = V(\delta_2') \text{ for } V \in \{Intok,\ Inid,\ V_{loc},\ Inc,\ Ass,\ Aloc\}. \quad (2.21)$$

Without loss of generality, we can choose δ_1 and δ_2 in such a way that $id(\delta_1) \cap id(\delta_2) = \emptyset$.

Now, observe that, according to the definition of $\uparrow$ and inductive hypothesis, $\alpha(\delta'_1) \parallel \alpha(\delta'_2)$ is defined and therefore $V_{loc}(\delta'_1) \cap V_{loc}(\delta'_2) = \emptyset$. Moreover, since $\{G_1\} \subseteq Aloc(\delta'_1)$ and therefore $Fv(G_1) \subseteq V_{loc}(\delta'_1)$, and as per (2.21), we can conclude that

$$Fv(G_1) \cap V_{loc}(\delta'_2) = \emptyset. \tag{2.22}$$

Then, we can assume, without loss of generality, that

$$\{G''\} \cap Ass(\delta'_2) \quad \subseteq \quad Ass(\delta). \tag{2.23}$$

In fact, let $g \in (\{G''\} \cap Ass(\delta'_2)) \setminus Ass(\delta)$, since by definition of $\uparrow$ there exists $\bar{\sigma} \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta'_2))$ such that $\alpha(\delta) \simeq \bar{\sigma}$ and $g \notin Ass(\delta)$, by definition of $\parallel$, there exists a stable atom g' either in δ'_1 or δ'_2 such that, in order to obtain $\bar{\sigma}$, the assumption g is satisfied with the stable atom g'.

Then, according to (2.22), we can substitute each occurrence of g in the assumptions of δ'_2 with the corresponding element in G_1 (namely, with the atom $g_1 \in G_1$ such that $CT \models c_1 \wedge g \leftrightarrow c_1 \wedge g_1$).

We are now in a position to define

$$\delta \uparrow_{(H,G)} = (\delta_1, \delta_2).$$

Therefore, the following holds, with $i \in \{1, 2\}$:

1. By construction $InG(\delta) = (InG(\delta_1), InG(\delta_2))$. By definition of $\uparrow$ and by the previous observation $V_{loc}(\delta_i) = V_{loc}(\delta'_i) \subseteq V_{loc}(\delta)$, $Intok(\delta_i) = Intok(\delta'_i) = Intok(\delta)$, $Inid(\delta_i) = Inid(\delta'_i) \geq Inid(\delta)$ and $Inc(\delta_i) = Inc(\delta'_i) \subseteq Inc(\delta)$.
2. By definition of $\uparrow$, by construction and by (2.20),

$$\begin{array}{rcl} Ass(\delta_1) & = & Ass(\delta'_1) \cup \{G''\} \\ & \subseteq & Ass(\delta) \cup Aloc(\delta'_2) \cup InG(\delta'_2) \cup \{G''\} \\ & = & Ass(\delta) \cup Aloc(\delta_2) \cup InG(\delta_2). \end{array}$$

Moreover, according to (2.20), the definition of $\uparrow$, construction and (2.21),

$$\begin{aligned} Ass(\delta_2) &= Ass(\delta_2') \\ &\subseteq Ass(\delta) \cup Aloc(\delta_1') \cup InG(\delta_1') \\ &= Ass(\delta) \cup Aloc(\delta_1) \cup InG(\delta_1) \cup \{G''\} \\ &= Ass(\delta) \cup Aloc(\delta_1) \cup InG(\delta_1), \end{aligned}$$

where the last equality follows as per (2.23).

3. $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined and $\sigma \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$ exists such that $\alpha(\delta) \simeq \sigma$. The proof that $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined follows by observing that, by definition of derivation, $V_{loc}(\delta_1') \cap Fv(G'') = \emptyset$, by construction for $i \in \{1, 2\}$, $V_{loc}(\delta_i) = V_{loc}(\delta_i')$ and by definition of $\uparrow$, $\alpha(\delta_1') \parallel \alpha(\delta_2')$ is already defined.

   At this point, we will prove that $\alpha(\delta) \simeq \sigma \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$. First of all, observe that through construction, (2.20) and (2.21) for each $\bar{\sigma}_1 \in \eta(\alpha(\delta_1') \parallel \alpha(\delta_2'))$ there exists $\bar{\sigma}_2 \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$ such that $\bar{\sigma}_1 \simeq \bar{\sigma}_2$ (namely, $\eta(\alpha(\delta_1') \parallel \alpha(\delta_2')) \ll \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$).

   Moreover, by definition of $\uparrow$, $\alpha(\delta) \simeq \bar{\sigma} \in \eta(\alpha(\delta_1') \parallel \alpha(\delta_2'))$. Then the proof follows through the transitivity of $\simeq$.

□

There now follow three minor lemmas. Said lemmata are used to prove Lemma 2.10. In the following, given a derivation

$$\delta = \langle \tilde{R}_1, c_1, T_1, n_1, \tilde{K}_1, \tilde{R}_2, d_1, T_2, n_1' \rangle \cdots \langle \tilde{R}_m, c_m, T_m, n_m, \emptyset, \tilde{R}_m, c_m, T_m, n_m \rangle,$$

we define $id(\delta) = id(\bigcup_{i=1}^{m} \tilde{R}_i) \cup id(\bigcup_{i=1}^{m-1} \tilde{K}_i)$.

The following lemma considers a derivation step s and replaces assumptions of s with unused constraints in the input goal of s.

Lemma 2.7 *Let P be a program and let R be a goal, such that there exists a derivation step $s = \langle \tilde{R}, c, T, n, \tilde{L}_1, \tilde{R}', d, T', n' \rangle$ for R. We suppose that $\tilde{L}_1$ has k CHR constraints. Assume that there exists*

$$\tilde{L}' = \{h_1\#id_1', \ldots, h_o\#id_o'\} \subseteq \tilde{R} \text{ and } \tilde{L} = \{g_1\#id_1, \ldots, g_o\#id_o\} \subseteq \tilde{L}_1$$

such that

- *the identified atoms in* $\tilde{L}'$ *are not used by* s,
- *for each* $j \in [1, o]$, $CT \models c \wedge h_j \leftrightarrow c \wedge g_j$ *and*
- $T'[id_1/id'_1, \ldots, id_o/id'_o]$ *is defined.*

There then exists a derivation step

$$s' = \langle \tilde{R}, c, T, n, \tilde{L}'_1, \tilde{R}'', d, T'', n'' \rangle,$$

whereby

- $\{n+1, \ldots, n+k\} = id(\tilde{L}_1)$, $\rho = [n+1/j_1, \ldots, n+k/j_k]$ *is a renaming,*
- $\tilde{L}'_1 = (\tilde{L}_1 \setminus \tilde{L})\rho$,
- $\tilde{R}'' = (\tilde{R}' \setminus \tilde{L}')[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho$,
- $T'' = T'[id_1/id'_1, \ldots, id_o/id'_o]\rho$, $n'' \leq n'$ *and*
- $V_{loc}(s) = V_{loc}(s')$.

Proof The proof is straightforward as per the definition of derivation step.
□

The following lemma substitutes constraints, that are in the input goal and eventually also in the output goal, with other constraints having a different label. The context, in which the following lemma is used, is the propagation of the substitution of an assumption with a stable atom in all of the computational steps of a sequence.

Lemma 2.8 *Let* P *be a program.* R *be a goal,*

$$s = \langle \tilde{R}_1, c, T_1, n, \tilde{L}_1, \tilde{R}_2, d, T_2, n' \rangle$$

be a derivation step for R, *where* $\tilde{R}_1$ *is an identified version of* R *and let the two sets of identified atoms* $\tilde{L}' = \{h_1\#id'_1, \ldots, h_o\#id'_o\}$ *and* $\tilde{L} = \{g_1\#id_1, \ldots, g_o\#id_o\}$ *be such that for each* $j \in [1, o]$ *the following then holds*

- $CT \models c \wedge h_j \leftrightarrow c \wedge g_j$,
- $id'_j \notin id(\tilde{R}_1) \cup id(\tilde{R}_2) \cup id(\tilde{L}_1)$
- *Either* $g_j \# id_j \in \tilde{R}_1$ *or* $id_j \notin id(\tilde{R}_1) \cup id(\tilde{R}_2) \cup id(\tilde{L}_1)$ *and*
- $T_2[id_1/id'_1, \ldots, id_o/id'_o]$ *is defined.*

There then exists a derivation step

$$s' = \langle \tilde{R}'_1, c, T'_1, n, \tilde{L}_1, \tilde{R}'_2, d, T'_2, n' \rangle$$

where for $i \in \{1, 2\}$, $\tilde{R}'_i = \tilde{R}_i[g_1 \# id_1/h_1 \# id'_1, \ldots, g_o \# id_o/h_o \# id'_o]$,
$T'_i = T_i[id_1/id'_1, \ldots, id_o/id'_o]$ *and* $V_{loc}(s) = V_{loc}(s')$.

Proof The proof is straightforward as per the definition of derivation step.
□

Before the following lemma we need the extension of the substitution operator defined in Definition 2.7 for the previously mentioned concrete sequences:

Definition 2.11 *What is defined in Definition 2.7 is considered as a given. Let the sequence*

$$\delta = \langle \tilde{R}_1, c_1, T_1, n_1, \tilde{L}_1, \tilde{R}_2, d_1, T_2, n'_1 \rangle \cdots \langle \tilde{R}_m, c_m, T_m, n_m, \emptyset, \tilde{R}_m, c_m, T_m, n_m \rangle,$$

then $\delta' = \delta[g_1 \# id_1/h_1 \# id'_1, \ldots, g_o \# id_o/h_o \# id'_o]$ *with for each* $j \in [1, o]$, $id_j \leq n_1$ *and* $id'_j \leq n_1$ *is:*

$$\delta' = \langle \tilde{R}^*_1, c_1, T^*_1, n_1, \tilde{L}^*_1, \tilde{R}^*_2, d_1, T^*_2, n'_1 \rangle \cdots \langle \tilde{R}^*_m, c_m, T^*_m, n_m, \emptyset, \tilde{R}^*_m, c_m, T^*_m, n_m \rangle.$$

where $\tilde{R}^*_i = \tilde{R}_i[g_1 \# id_1/h_1 \# id'_1, \ldots, g_o \# id_o/h_o \# id'_o]$, $T^*_i = T_i[id_1/id'_1, \ldots, id_o/id'_o]$ *and* $\tilde{L}^*_j = \tilde{L}_j[g_1 \# id_1/h_1 \# id'_1, \ldots, g_o \# id_o/h_o \# id'_o]$ *with* $1 \leq i \leq m$, $1 \leq j \leq m - 1$.

The following lemma substitutes constraints of the goal with other constraints that differ only for the identifier used. The context in which it will be used is the substitution of assumptions with stable atoms in a sequence.

Lemma 2.9 *Let P be a program and R be a goal,*

$$\delta = \langle \tilde{R}_1, c_1, T_1, n_1, \tilde{L}_1, \tilde{R}_2, d_1, T_2, n'_1 \rangle \cdots \langle \tilde{R}_m, c_m, T_m, n_m, \emptyset, \tilde{R}_m, c_m, T_m, n_m \rangle \in S'_P(R),$$

where $\tilde{R}_1$ is an identified version of R, and let $\tilde{L}' = \{h_1\#id'_1, \ldots, h_o\#id'_o\}$ and let $\tilde{L} = \{g_1\#id_1, \ldots, g_o\#id_o\}$ be two sets of identified atoms such that for each $j \in [1, o]$ the following holds

- $id_j \leq n_1$ *and* $id'_j \leq n_1$
- $CT \models c \wedge h_j \leftrightarrow c \wedge g_j$,
- $id'_j \not\in id(\delta)$
- *Either* $g_j\#id_j \in \tilde{R}_1$ *or* $id_j \not\in id(\delta)$ *and*
- $T_m[id_1/id'_1, \ldots, id_o/id'_o]$ *is defined.*

Then $\delta[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o] \in \mathcal{S}'_P(R')$,
where $R' = chr(\tilde{R}_1[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o])$.

Proof The proof is straightforward as per Lemma 2.8 and by induction on the length of δ.
□

The following third main lemma states that once fixed, two concrete sequences, that are derived from the goals H and G, there exists a concrete sequence, that is derived from (H, G), then said abstraction is equal to the abstraction of the composition of the given two sequences.

Lemma 2.10 *Let P be a program, let H and G be two goals and assume that $\delta_1 \in \mathcal{S}'_P(H)$ and $\delta_2 \in \mathcal{S}'_P(G)$ are two sequences such that the following holds:*

1. $\alpha(\delta_1) \parallel \alpha(\delta_2)$ *is defined,*
2. $\sigma = \langle c_1, \tilde{K}_1, \tilde{W}_1, d_1 \rangle \cdots \langle c_m, \tilde{W}_m, T_m \rangle \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$,
3. $(V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma) = \emptyset$,

4. *For* $i \in [1, m]$, $(V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j)$.

Then there exists $\delta \in \mathcal{S}'_P(H, G)$ *such that* $\alpha(\delta) \simeq \sigma$.

Proof The proof follows by induction on the length of δ. First of all, two concrete sequences $\delta_1 \in \mathcal{S}'_p(H)$ and $\delta_2 \in \mathcal{S}'_p(G)$, which are elements of $\mathcal{S}eq$, and their composition, are considered. Furthermore, it is proven that the first concrete tuple, whose abstraction provides the first abstract sequence of the composed abstract one, represents a derivation step for (H, G). Afterwards, it will be proven that the inductive abstract sequence enjoys properties of the four points in the hypothesis of the lemma. Then the existence of $\delta \in \mathcal{S}'_p(H, G)$ follows by compatibility. Finally, the existence of a rename ρ such that $\sigma = \alpha(\delta)\rho$ is proven.

First of all, observe that since $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined, we can assume without loss of generality, that the following holds:

- $InG(\delta_1) = \tilde{H}$, $InG(\delta_2) = \tilde{G}$. $Intok(\delta_1) = T'_1$, $Intok(\delta_2) = T''_1$ and $Inid(\delta_1) = p_1$ and $Inid(\delta_2) = q_1$ such that
- for each $h \in id(\tilde{H})$ and $k \in id(\tilde{G})$, $h \neq k$, $h \leq q_1$, $k \leq p_1$ and
- for each $j \in [1, l]$ and $r@i_1, \ldots, i_l \in T'_1$, $\{i_1, \ldots, i_l\} \not\subseteq id(\delta_2)$ and $i_j \leq q_1$ and for each $r@i_1, \ldots, i_l \in T''_1$, $\{i_1, \ldots, i_l\} \not\subseteq id(\delta_1)$ and $i_j \leq p_1$.

In the remaining part of the proof, given two derivations $\delta_1 \in \mathcal{S}'_P(H)$ and $\delta_2 \in \mathcal{S}'_P(G)$, which verify the previous conditions, we can construct by induction on the $l = length(\sigma)$ a derivation $\delta \in \mathcal{S}'_P(H, G)$ such that the following conditions hold

1. $InG(\delta) = (\tilde{H}, \tilde{G}) = (InG(\delta_1), InG(\delta_2))$, $V_{loc}(\delta) \subseteq V_{loc}(\delta_1) \cup V_{loc}(\delta_2)$, $Intok(\delta) = T'_1 \cup T''_1$, $Inid(\delta) = n_1$, where n_1 is the minimum between p_1 and q_1, $Inc(\delta) \subseteq Inc(\delta_1) \cup Inc(\delta_2)$,
2. $Ass(\delta) \subseteq Ass(\delta_1) \cup Ass(\delta_2)$ and
3. there exists a renaming ρ such that $\sigma = \alpha(\delta)\rho$ (and therefore $\sigma \simeq \alpha(\delta)$) and $\rho(id) = id$ for each $id \leq Inid(\delta)$.

($l = 1$) In this case $\delta_1 = \langle \tilde{H}, c, T', p, \emptyset, \tilde{H}, c, T', p \rangle$, $\delta_2 = \langle \tilde{G}, c, T'', q, \emptyset, \tilde{G}, c, T'', q \rangle$, $\alpha(\delta_1) = \langle c, \tilde{H}, T' \rangle$, $\alpha(\delta_2) = \langle c, \tilde{G}, T'' \rangle$, $\sigma = \langle c, (\tilde{H}, \tilde{G}), T' \cup T'' \rangle$ and $\delta = \langle (\tilde{H}, \tilde{G}), c, T, n, \emptyset, (\tilde{H}, \tilde{G}), c, T, n \rangle$, where $T = T' \cup T''$ and n is the minimum between p and q.

($l > 1$) Without loss of generality, we can assume that

$$\delta_1 = t' \cdot \delta'_1,\ \delta_2 = \langle \tilde{G}, e_1, T''_1, q_1, \tilde{J}_1, \tilde{G}_2, f_1, T''_2, q'_1 \rangle \cdot \delta'_2$$

where the following holds

- $t' = \langle \tilde{H}, c_1, T'_1, p_1, \tilde{L}_1, \tilde{H}_2, d_1, T'_2, p'_1 \rangle$, $\delta'_1 \in \mathcal{S}'_P(H_2)$, t' is compatible with δ'_1 and $\sigma_1 = \alpha(\delta_1) = \langle c_1, \tilde{L}_1, \tilde{N}_1, d_1 \rangle \cdot \alpha(\delta'_1)$ where $\tilde{N}_1$ is the set of stable atoms for σ_1.
- $\delta'_2 \in \mathcal{S}'_P(G_2) \cup \varepsilon$ and if $\delta'_2 \in \mathcal{S}'_P(G_2)$ then $\sigma_2 = \alpha(\delta_2) = \langle e_1, \tilde{J}_1, \tilde{M}_1, f_1 \rangle \cdot \alpha(\delta'_2)$ and $\tilde{M}_1$ is the set of stable atoms for σ_2, else $\sigma_2 = \alpha(\delta_2) = \langle e_1, \tilde{M}_1, T''_1 \rangle$ and $\tilde{M}_1 = \tilde{G}$.
- $\sigma \in \eta(\langle c_1, \tilde{L}_1, \tilde{N}_1 \cup \tilde{M}_1, d_1 \rangle \cdot \sigma')$ and $\sigma' \in \eta(\alpha(\delta'_1) \parallel \sigma_2)$.

In this case, without loss of generality, we can assume that $p'_1 \leq q_1$.

As per the definition of η, there exist the sets of identified atoms $\tilde{L}', \tilde{L}'', \tilde{L}$ such that

$$\tilde{L} = \{g_1\#id_1, \ldots, g_o\#id_o\} \subseteq \tilde{L}_1 \text{ and}$$
$$\tilde{L}' = \{h_1\#id'_1, \ldots, h_o\#id'_o\} \subseteq ((\tilde{N}_1 \cup \tilde{M}_1) \setminus \tilde{L}''),$$

where

1. $\tilde{L}''$ is the set of stable atoms of $(\tilde{N}_1 \cup \tilde{M}_1)$ used in $\eta(\alpha(\delta'_1) \parallel \sigma_2)$, in order to obtain σ'.
2. for $1 \leq j \leq o$, $CT \models (c_1 \wedge g_j) \leftrightarrow (c_1 \wedge h_j)$ and
3. $\sigma = (\langle c_1, \tilde{K}_1, \tilde{W}_1, d_1 \rangle \cdot (\sigma' \setminus \tilde{L}'))\,[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]$ is defined, where $\tilde{K}_1 = \tilde{L}_1 \setminus \tilde{L}$ and $\tilde{W}_1 = (\tilde{N}_1 \cup \tilde{M}_1) \setminus (\tilde{L}' \cup \tilde{L}'')$.

Now observe that the following holds:

- Since $t' = \langle \tilde{H}, c_1, T'_1, p_1, \tilde{L}_1, \tilde{H}_2, d_1, T'_2, p'_1 \rangle$ represents a derivation step for H and since by hypothesis for each $k \in id(\tilde{G})$, $k \leq p_1$ and for each $r@i_1, \ldots, i_l \in T''_1$, $\{i_1, \ldots, i_l\} \not\subseteq id(\delta_1)$, we can conclude that

$$t'' = \langle (\tilde{H}, \tilde{G}), c_1, T'_1 \cup T''_1, p_1, \tilde{L}_1, (\tilde{H}_2, \tilde{G}), d_1, T'_2 \cup T''_1, p'_1 \rangle$$

represents a derivation step for (H, G).

- Since

$$\sigma = (\langle c_1, \tilde{K}_1, \tilde{W}_1, d_1 \rangle \cdot (\sigma' \setminus \tilde{L}')) \, [g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]$$

is defined and by definition of $\|$, we can say that $(T'_2 \cup T''_1)[id_1/id'_1, \ldots, id_o/id'_o]$ is defined

- As per previous observations and Lemma 2.7,

$$t = \langle (\tilde{H}, \tilde{G}), c_1, T'_1 \cup T''_1, p_1, \tilde{K}'_1, \tilde{B}, d_1, T'', p''_1 \rangle$$

represents a derivation step for (H, G), where

  - $\{p_1 + 1, \ldots, p_1 + k\} = id(\tilde{L}_1)$ and $\rho_1 = [p_1 + 1/j_1, \ldots, p_1 + k/j_k]$ is a renaming,
  - $\tilde{K}'_1 = (\tilde{L}_1 \setminus \tilde{L})\rho_1 = \tilde{K}_1\rho_1$,
  - $\tilde{B} = ((\tilde{H}_2, \tilde{G}) \setminus \tilde{L}')[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho_1$,
  - $T'' = (T'_2 \cup T''_1)[id_1/id'_1, \ldots, id_o/id'_o]\rho_1$, $p''_1 \leq p'_1$ and $V_{loc}(t) = V_{loc}(t')$.

Moreover, the following holds:

$\alpha(\delta'_1) \parallel \alpha(\delta_2)$ **is defined.** Since $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined, we can assume that $id(\delta_1) \cap id(\delta_2) = \emptyset$ and so $id(\delta'_1) \cap id(\delta_2) = \emptyset$. Then by definition, we have only to prove that

$$(V_{loc}(\alpha(\delta'_1)) \cup Fv(H_2)) \cap (V_{loc}(\alpha(\delta_2)) \cup Fv(G)) = Fv(H_2) \cap Fv(G).$$

First of all, observe that since $V_{loc}(\alpha(\delta'_1)) \subseteq V_{loc}(\alpha(\delta_1))$ and $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined, we can say that $V_{loc}(\alpha(\delta'_1)) \cap (V_{loc}(\alpha(\delta_2)) \cup Fv(G)) = \emptyset$ and $(Fv(H) \cup V_{loc}(\alpha(\delta_1))) \cap (V_{loc}(\alpha(\delta_2))) = \emptyset$.

Now, observe that according to the definition of derivation

$$Fv(H_2) \subseteq Fv(H) \cup V_{loc}(\alpha(\delta_1)) \cup Fv(L_1)$$

and therefore, $Fv(H_2) \cap V_{loc}(\alpha(\delta_2)) = Fv(L_1) \cap V_{loc}(\alpha(\delta_2))$. Then as per previous observations

$$\begin{aligned}(V_{loc}(\alpha(\delta_1')) \cup Fv(H_2)) \cap (V_{loc}(\alpha(\delta_2)) \cup Fv(G)) &= \\ (Fv(H_2) \cap Fv(G)) \cup (Fv(L_1) \cap V_{loc}(\alpha(\delta_2))).\end{aligned}$$

We now, assume that there exists $x \in Fv(L_1) \cap V_{loc}(\alpha(\delta_2))$ and that $g \in L_1$ is such that $x \in Fv(g)$. Since as seen in Point 3. of the hypothesis $V_{loc}(\alpha(\delta_2)) \cap V_{ass}(\sigma) = \emptyset$, we can maintain that $g \notin K_1$ and therefore there exists $g' \in G$ such that $CT \models c_1 \wedge g \leftrightarrow c_1 \wedge g'$. Now, observe that, since $g' \in G$ and $x \in V_{loc}(\alpha(\delta_2))$, we can say that $x \notin Fv(g')$ and therefore, we can say that $x \in Fv(c_1)$ and $CT \not\models \exists_x c_1 \leftrightarrow c_1$. Then, since by definition of $\|$, $CT \models e_1 \rightarrow c_1$, either $x \in Fv(e_1)$ or $CT \models e_1 \leftrightarrow \texttt{false}$. In both cases $x \notin V_{loc}(\alpha(\delta_2))$ and it then follows that $Fv(L_1) \cap V_{loc}(\alpha(\delta_2)) = \emptyset$.

$\sigma' = \langle c_2, \tilde{K}_2, \tilde{W}_2 \cup \tilde{L}', d_2\rangle \cdots \langle c_m, \tilde{W}_m \cup \tilde{L}', T_m\rangle \in \eta(\alpha(\delta_1') \| \alpha(\delta_2))$**.** The proof is straightforward, by definition of $\|$.

$(V_{loc}(\alpha(\delta_1')) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma') = \emptyset$**.** According to the definition, the hypothesis and Lemma 2.1, we can say that

$$\begin{aligned}(V_{loc}(\alpha(\delta_1')) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma') &\subseteq \\ (V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma) &= \emptyset.\end{aligned}$$

for $i \in [2, m]$**,** $(V_{loc}(\alpha(\delta_1')) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) \subseteq \bigcup_{j=2}^{i-1} Fv(d_j)$. To prove this statement, observe that as per the hypothesis and Lemma 2.1, where $i \in [2, m]$,

$$\begin{aligned}(V_{loc}(\alpha(\delta_1')) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) &\subseteq \\ (V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) &\subseteq \\ \textstyle\bigcup_{j=1}^{i-1} Fv(d_j).\end{aligned}$$

Let $i \in [2, m]$, such that there exists $x \in (V_{loc}(\alpha(\delta_1')) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) \cap Fv(d_1)$. As per the hypothesis $x \notin Fv(c_1)$. Then, since $x \in Fv(d_1) \subseteq$

$Fv(t')$ and t' is compatible with δ'_1, we may conclude that $x \notin V_{loc}(\alpha(\delta'_1))$ and therefore $x \in V_{loc}(\alpha(\delta_2))$. As per Lemma 2.1 and since $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined, we can say that $x \notin Fv(H)$ and therefore, by definition of derivation, we can say that $CT \not\models \exists_x d_1 \leftrightarrow d_1$. According to the definition of $\parallel$, $CT \models e_1 \rightarrow d_1$ and therefore, since $x \in Fv(d_1)$ and $CT \not\models \exists_x d_1 \leftrightarrow d_1$, either $x \in Fv(e_1)$ or $CT \models e_1 \leftrightarrow \texttt{false}$. In both the cases $x \notin V_{loc}(\alpha(\delta_2))$. As in previous observations

$$(V_{loc}(\alpha(\delta'_1)) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) \subseteq \bigcup_{j=2}^{i-1} Fv(d_j)$$

and we are therefore arrived at the thesis.

Moreover, via construction the following holds

- $Intok(\delta'_1) = T'_2$ and $Inid(\delta'_1) = p_2$. Since according to the definition of the derivation $p_2 \geq p_1$ and the hypothesis for each $k \in id(\tilde{G})$, $k \leq p_1$, we can say that for each $h \in id(\tilde{H}_2)$, $h \neq k$ and $k \leq p_2$. Moreover, without loss of generality, we can assume that for each $h \in id(\tilde{H}_2)$, $h \leq q_1$.
- according to the definition of the derivation, if $T'_2 \neq T'_1$, $T'_2 = T'_1 \cup \{r@id_1, \dots, id_l\}$ such that $\{id_1, \dots, id_l\} \subseteq id(\delta_1)$. Then since by hypothesis $id(\delta_1) \cap id(\delta_2) = \emptyset$ and for each $r@i_1, \dots, i_l \in T'_1$, $\{i_1, \dots, i_l\} \not\subseteq id(\delta_2)$, we can conclude that for each $r@i_1, \dots, i_l \in T'_2$, $\{i_1, \dots, i_l\} \not\subseteq id(\delta_2)$.

Through previous results and by inductive hypothesis, we can say that there $\delta' \in \mathcal{S}'_P(H_2, G)$ exists such that

1. $InG(\delta') = (InG(\delta'_1), InG(\delta_2)) = (\tilde{H}_2, \tilde{G})$, $V_{loc}(\delta') \subseteq V_{loc}(\delta'_1) \cup V_{loc}(\delta_2)$, $Intok(\delta') = T'_2 \cup T''_1$, $Inid(\delta') = m_2$, where m_2 is the minimum between p_2 and q_1, $Inc(\delta') \subseteq Inc(\delta'_1) \cup Inc(\delta_2)$,
2. $Ass(\delta') \subseteq Ass(\delta'_1) \cup Ass(\delta_2)$ and
3. there exists a renaming ρ' such that $\sigma' = \alpha(\delta')\rho'$ (and therefore $\sigma' \simeq \alpha(\delta')$) and $\rho'(id) = id$ for each $id \leq Inid(\delta')$.

Moreover, according to the definition of η, $\tilde{L}' \subseteq (\tilde{H}, \tilde{G})$ is a set of atoms which are stable in δ'. Let $\delta'' \in \mathcal{S}'_P(R')$ be the derivation obtained from δ' by deleting from each goal in δ' the atoms in $\tilde{L}'$, where $\tilde{R}' = (\tilde{H}_2, \tilde{G}) \setminus \tilde{L}'$ and $R' = chr(\tilde{R}')$.

Via construction and Lemma 2.9

$$\bar{\delta} = \delta''[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho_1 \in S'_P(R),$$

where

- $InG(\bar{\delta}) = \tilde{R} = \tilde{R}'[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho_1$, $R = chr(\tilde{R})$,
- $Intok(\bar{\delta}) = (T'_2 \cup T''_1)[id_1/id'_1, \ldots, id_o/id'_o]\rho_1$ and
- $V_{loc}(\bar{\delta}) = V_{loc}(\delta'') = V_{loc}(\delta')$.

Let us denote by δ the sequence $t \cdot \bar{\delta}$.

Then, to prove the thesis, we have to prove that $t \cdot \bar{\delta} \in \mathcal{S}eq$, t is compatible with $\bar{\delta}$ (and therefore $\delta \in \mathcal{S}'_P(H, G)$), $V_{loc}(\delta) \subseteq V_{loc}(\delta_1) \cup V_{loc}(\delta_2)$, $Inc(\delta) \subseteq Inc(\delta_1) \cup Inc(\delta_2)$, $Ass(\delta) \subseteq Ass(\delta_1) \cup Ass(\delta_2)$ and $\sigma \simeq \alpha(\delta)$.

($t \cdot \bar{\delta} \in \mathcal{S}eq$). By construction, we have only to prove that $CT \models instore(\bar{\delta}) \rightarrow d_1$. The proof is straightforward, since via construction either $instore(\bar{\delta}) = instore(\delta'_1)$ or $instore(\bar{\delta}) = instore(\delta_2)$.

(t is compatible with $\bar{\delta}$). The following holds.

1. $V_{loc}(\bar{\delta}) \cap Fv(t) = \emptyset$. By construction and by inductive hypothesis

$$\begin{aligned} & V_{loc}(t) = V_{loc}(t'),\ Fv(t) \subseteq Fv(t') \cup Fv(G) \text{ and} \\ & V_{loc}(\bar{\delta}) = V_{loc}(\delta') \subseteq V_{loc}(\delta'_1) \cup V_{loc}(\delta_2). \end{aligned} \tag{2.24}$$

Since t' is compatible with δ'_1 (Definition 2.4 point 1) and $\alpha(\delta'_1) \parallel \alpha(\delta_2)$ is defined, we can conclude that

$$V_{loc}(\delta'_1) \cap (Fv(t') \cup Fv(G)) = \emptyset. \tag{2.25}$$

According to points 3. and 4. of the hypothesis $Fv(K_1, c_1) \cap V_{loc}(\delta_2) = \emptyset$ and the definition of derivation $Fv(G) \cap V_{loc}(\delta_2) = \emptyset$. Moreover, through point 1. of the hypothesis we can say that $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined and therefore $(Fv(H) \cup V_{loc}(t')) \cap V_{loc}(\delta_2) = \emptyset$. Then by definition and in accordance with the first statement in (2.24)

$$Fv(t) \cap V_{loc}(\delta_2) = (Fv(c_1, H, G, K_1) \cup V_{loc}(t')) \cap V_{loc}(\delta_2) = \emptyset. \quad (2.26)$$

Then

$$\begin{array}{lll} V_{loc}(\bar{\delta}) \cap Fv(t) & \subseteq & \text{(according to the last statement in (2.24))} \\ (V_{loc}(\delta_1') \cup V_{loc}(\delta_2)) \cap Fv(t) & \subseteq & \text{(according to the second statement in (2.24) and (2.25))} \\ V_{loc}(\delta_2) \cap Fv(t) & = & \text{(according to (2.26))} \\ \emptyset. & & \end{array}$$

2. $V_{loc}(t) \cap V_{ass}(\bar{\delta}) = \emptyset$. The proof is immediate via point 3. of the hypothesis.
3. for $i \in [2, n]$, $V_{loc}(t) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j)$. The proof is immediate through construction and point 4. of the hypothesis.

($V_{loc}(\delta) \subseteq V_{loc}(\delta_1) \cup V_{loc}(\delta_2)$). Via construction

$$\begin{array}{lll} V_{loc}(\delta) & = & \text{(through construction)} \\ V_{loc}(t) \cup V_{loc}(\bar{\delta}) & = & \text{(through construction)} \\ V_{loc}(t') \cup V_{loc}(\delta') & \subseteq & \text{(through previous results)} \\ V_{loc}(t') \cup V_{loc}(\delta_1') \cup V_{loc}(\delta_2) & = & \text{(through construction)} \\ V_{loc}(\delta_1) \cup V_{loc}(\delta_2). & & \end{array}$$

($Inc(\delta) \subseteq Inc(\delta_1) \cup Inc(\delta_2)$)

$$\begin{array}{lll} Inc(\delta) & = & \text{(through construction)} \\ Inc(t) \cup Inc(\bar{\delta}) & = & \text{(through construction)} \\ Inc(t') \cup Inc(\delta') & \subseteq & \text{(through previous results)} \\ Inc(t') \cup Inc(\delta'_1) \cup Inc(\delta_2) & = & \text{(through construction)} \\ Inc(\delta_1) \cup Inc(\delta_2). & & \end{array}$$

($Ass(\delta) \subseteq Ass(\delta_1) \cup Ass(\delta_2)$)

$$\begin{array}{lll} Ass(\delta) & = & \text{(through construction)} \\ Ass(t) \cup Ass(\bar{\delta}) & = & \text{(through the definition of } Ass(t)\text{)} \\ (\tilde{L}_1 \setminus \tilde{L}) \cup Ass(\bar{\delta}) & \subseteq & \text{(through the definition of } \setminus\text{)} \\ \tilde{L}_1 \cup Ass(\bar{\delta}) & \subseteq & \text{(through the definition of } \bar{\delta}\text{)} \\ \tilde{L}_1 \cup Ass(\delta') & \subseteq & \text{(through previous results)} \\ \tilde{L}_1 \cup Ass(\delta'_1) \cup Ass(\delta_2) & = & \text{(through construction)} \\ Ass(\delta_1) \cup Ass(\delta_2). & & \end{array}$$

(there exists a renaming ρ such that $\sigma = \alpha(\delta)\rho$ (and therefore $\sigma \simeq \alpha(\delta)$) and $\rho(id) = id$ for each $id \leq n_1$). By inductive hypothesis there exists a renaming ρ' such that $\sigma' = \alpha(\delta')\rho'$ and $\rho'(id) = id$ for each $id \leq n_2$. Since by definition of the derivation, for each $j \in id(\tilde{L}')$, $j \leq n_2$ we can conclude that $\rho'(j) = j$ for each $j \in id(\tilde{L}')$. Then

$$\begin{array}{lll} \sigma' \setminus \tilde{L}' & = & \text{(since } \sigma' = \alpha(\delta')\rho'\text{)} \\ \alpha(\delta')\rho' \setminus \tilde{L}' & = & \text{(by previous observation)} \\ (\alpha(\delta') \setminus \tilde{L}')\rho' & = & \text{(through the definition of } \setminus \text{ and } \alpha\text{)} \\ (\alpha(\delta' \setminus \tilde{L}'))\rho' & = & \text{(through the definition of } \delta''\text{)} \\ (\alpha(\delta''))\rho'. & & \end{array}$$

Moreover, since for $i \in [1, o]$ and for $r \in [1, k]$, $id_i \leq n_2$ $id'_i \leq n_2$, $p_1 + r \leq n_2$ and $j_r \leq n_2$, we can say that

$$\begin{array}{l} (\alpha(\delta''))\rho'[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho_1 = \\ (\alpha(\delta''))[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho_1\rho' \end{array}$$

and therefore

$$
\begin{aligned}
&(\sigma' \setminus \tilde{L}')[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho_1 \quad = \\
&\qquad \text{(by a previous result)} \\
&(\alpha(\delta''))\rho'[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho_1 \quad = \\
&\qquad \text{(by previous observation)} \\
&(\alpha(\delta''))[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho_1\rho' \quad = \\
&\qquad \text{(according to the definition of } \bar{\delta}) \\
&(\alpha(\bar{\delta}))\rho'.
\end{aligned}
$$

Then, according to the definition of renaming

$$
(\sigma' \setminus \tilde{L}')[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o] \quad = \quad (\alpha(\bar{\delta}))\rho \tag{2.27}
$$

where $\rho = \rho'\rho_2$ and $\rho_2 = [j_1/p_1 + 1, \ldots, j_k/p_1 + k] = \rho_1^{-1}$.

By definition, we can say that ρ is a renaming and by construction $\rho(j) = j$ for each $j \leq n_1$.

According to the definition of δ, we can observe that

$$
\begin{aligned}
\alpha(\delta)\rho \quad &= \quad \text{(by virtue of the definition of } \alpha \text{ and } \delta) \\
\langle c_1, \tilde{K}'_1\rho, \tilde{W}'_1\rho, d_1\rangle \cdot \alpha(\bar{\delta})\rho \quad &= \quad \text{(via the definition of renaming and (2.27))} \\
\langle c_1, \tilde{K}'_1\rho, \tilde{W}'_1\rho, d_1\rangle \cdot (\sigma' \setminus \tilde{L}')&[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o].
\end{aligned} \tag{2.28}
$$

where $\tilde{W}'_1 = \tilde{B}_1 \cap \tilde{B}_2$, where $\tilde{B}_1$ and $\tilde{B}_2$ are the sets of atoms in $(\tilde{H}, \tilde{G})$ which are not rewritten by t and by $\bar{\delta}$, respectively. Now observe that since $\tilde{K}'_1 = \tilde{K}_1\rho_1$ and since $\rho'(j) = j$ for each $j \leq n_2$, we can mainain that

$$
\tilde{K}'_1\rho = \tilde{K}_1\rho_1\rho = \tilde{K}_1\rho_1\rho'\rho_2 = \tilde{K}_1\rho_1\rho_2 = \tilde{K}_1. \tag{2.29}
$$

Moreover, according to the construction $\tilde{B}_1 = ((\tilde{N}_1 \cup \tilde{M}_1) \setminus \tilde{L}')$ and (2.27) $\tilde{B}_2 = (\tilde{W}'_2 \setminus \tilde{L}')[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho^{-1}$, where $\tilde{W}'_2$ is the first stable set of σ'. Then

$$
\begin{aligned}
\tilde{W}'_1 \quad = \quad & ((\tilde{N}_1 \cup \tilde{M}_1) \setminus \tilde{L}') \cap \\
& (\tilde{W}'_2 \setminus \tilde{L}')[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]\rho^{-1}.
\end{aligned} \tag{2.30}
$$

Now, observe that $\rho(j) = j$ for each $j \leq n_1$ and for each $i \in id((\tilde{N}_1 \cup \tilde{M}_1) \setminus \tilde{L}')$, we can say that $i \leq n_1$. Then by (2.30)

$$\begin{aligned} \tilde{W}'_1 &= ((\tilde{N}_1 \cup \tilde{M}_1) \setminus \tilde{L}') \cap \\ &\quad (\tilde{W}'_2 \setminus \tilde{L}')[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o]. \end{aligned} \tag{2.31}$$

As a result of the construction for each $i \in [1, o]$, we can observe that $id_i > n_1$ and $id'_i \in id(\tilde{L}')$. Then by (2.31)

$$\tilde{W}'_1 = ((\tilde{N}_1 \cup \tilde{M}_1) \setminus \tilde{L}') \cap (\tilde{W}'_2 \setminus L').$$

As a result of the construction and via the definition of $\|$, $W'_2 = ((\tilde{N}_2 \cup \tilde{M}_1) \setminus \tilde{L}'')$, where $\tilde{N}_2$ is the set of stable atoms of $\alpha(\delta'_1)$, and therefore, as in the previous result

$$\tilde{W}'_1 = ((\tilde{N}_1 \cup \tilde{M}_1) \setminus \tilde{L}') \cap ((\tilde{N}_2 \cup \tilde{M}_1) \setminus \tilde{L}''). \tag{2.32}$$

Moreover, since for each $i \in [1, o]$, $id_i \in id(\tilde{L})$ we can observe that

$$\begin{aligned} &\tilde{W}_1 = \tilde{W}_1[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o] \text{ and} \\ &\tilde{K}_1 = \tilde{K}_1[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o] \end{aligned} \tag{2.33}$$

Then

$$\begin{aligned}
&\tilde{W}'_1\rho && = \text{(since } \rho(i) = i \text{ for each } i \leq n_1) \\
&\tilde{W}'_1 && = \text{(via (2.32))} \\
&((\tilde{N}_1 \cup \tilde{M}_1) \setminus \tilde{L}') \cap (((\tilde{N}_2 \cup \tilde{M}_1) \setminus \tilde{L}'') && = \text{(through the properties of} \\
& && \quad \text{set operators)} \\
&((\tilde{N}_1 \cup \tilde{M}_1) \cap (\tilde{N}_2 \cup \tilde{M}_1)) \setminus (\tilde{L}' \cup \tilde{L}'') && = \text{(since by definition } \tilde{N}_1 \subseteq \tilde{N}_2) \\
&(\tilde{N}_1 \cup \tilde{M}_1) \setminus (\tilde{L}' \cup \tilde{L}'') && = \text{(as a result of construction)} \\
&\tilde{W}_1 &&
\end{aligned}$$

and therefore

$$\begin{aligned}
&\alpha(\delta)\rho &=\\
&\quad\text{(through (2.28))}\\
&\langle c_1, \tilde{K}'_1\rho, \tilde{W}'_1\rho, d_1\rangle \cdot (\sigma' \setminus L')[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o] &=\\
&\quad\text{(through (2.29) and the previous result)}\\
&\langle c_1, \tilde{K}_1, \tilde{W}_1, d_1\rangle \cdot (\sigma' \setminus L')[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o] &=\\
&\quad\text{(through (2.33))}\\
&(\langle c_1, \tilde{K}_1, \tilde{W}_1, d_1\rangle \cdot (\sigma' \setminus L'))[g_1\#id_1/h_1\#id'_1, \ldots, g_o\#id_o/h_o\#id'_o] &=\\
&\quad\text{(by definition)}\\
&\sigma
\end{aligned}$$

and then the thesis.

□

Finally, the last immediate main lemma considers two sequences that differ only for identifiers, so the free variables of assumptions and local ones are the same in both sequences.

Lemma 2.11 *Let $\sigma, \sigma' \in \mathcal{D}$ such that $\sigma \simeq \sigma'$. Then $V_r(\sigma) = V_r(\sigma')$ holds, where $r \in \{ass, loc\}$.*

Now by using the above results we can prove the following theorem.

Theorem 2.1 (Compositionality) *Let P be a program and let H and G be two goals. Then*

$$\mathcal{S}_P(H,G) \simeq \mathcal{S}_P(H) \parallel \mathcal{S}_P(G).$$

Proof We prove the two inclusions separately.

($\mathcal{S}_P(H,G) \ll \mathcal{S}_P(H) \parallel \mathcal{S}_P(G)$). Let $\sigma \in \mathcal{S}_P(H,G)$. According to the definition of $\mathcal{S}_P$, there exists $\delta \in \mathcal{S}'_P(H,G)$ such that $\sigma = \alpha(\delta)$. According to Lemma 2.6 there

$\delta_1 \in \mathcal{S}'_P(H)$ and $\delta_2 \in \mathcal{S}'_P(G)$ exist such that for $i = 1, 2$, $V_{loc}(\delta_i) \subseteq V_{loc}(\delta)$, $\sigma' \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$ and $\sigma' \simeq \sigma$. Let

$$\delta = \langle(\tilde{H}, \tilde{G}), c_1, T_1, n_1, \tilde{K}_1, \tilde{B}_2, d_1, T_2, n'_1\rangle \cdots \\ \cdots \langle \tilde{B}_m, c_m, T_m, n_m, \emptyset, \tilde{B}_m, c_m, T_m, n_m\rangle$$

and let $\sigma' = \langle c_1, \tilde{K}_1, \tilde{W}_1, d_1\rangle \cdots \langle c_m, \tilde{W}_m, T_m\rangle$. According to Lemma 2.11 and since $\sigma \simeq \sigma'$, in order to prove the thesis we have only to show that

$(V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma) = \emptyset$ and
for $i \in [1, m]$, $(V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j)$,

which are the two conditions which fail to satisfy all those of Definition 2.9. First observe that through Lemma 2.1 and the hypothesis, we can say respectively that

$$V_{ass}(\sigma) = V_{ass}(\delta) \text{ and for } i \in \{1, 2\}, V_{loc}(\alpha(\delta_i)) = V_{loc}(\delta_i) \subseteq V_{loc}(\delta). \quad (2.34)$$

Then through the previous results and the properties of the derivations (point 2) of Definition 2.4 (Compatibility))

$$(V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma) \subseteq V_{loc}(\delta) \cap V_{ass}(\delta) = \emptyset.$$

Moreover, in keeping with the hypothesis and point 3) of Definition 2.4 (Compatibility), for $i \in [1, m]$,

$$(V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) \subseteq V_{loc}(\delta) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j)$$

holds and this completes the proof of the first inclusion.

($\mathcal{S}_P(H) \parallel \mathcal{S}_P(G) \ll \mathcal{S}_P(H, G)$). Let $\sigma \in \mathcal{S}_P(H) \parallel \mathcal{S}_P(G)$. According to the definition of $\mathcal{S}_P$ and of $\parallel$, $\delta_1 \in \mathcal{S}'_P(H)$ and $\delta_2 \in \mathcal{S}'_P(G)$ exist, such that $\sigma_1 = \alpha(\delta_1)$, $\sigma_2 = \alpha(\delta_2)$, $\sigma_1 \parallel \sigma_2$ is defined, $\sigma = \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1\rangle \cdots \langle c_m, \tilde{H}_m, T_m\rangle \in \eta(\sigma_1 \parallel \sigma_2)$, $(V_{loc}(\sigma_1) \cup V_{loc}(\sigma_2)) \cap V_{ass}(\sigma) = \emptyset$ and for $i \in [1, m]$, $(V_{loc}(\sigma_1) \cup V_{loc}(\sigma_2)) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j)$. The proof is then straightforward, by using Lemma 2.10.

2.3 Correctness

In order to show the correctness of the semantics $\mathcal{S}_P$ with respect to the (input/output) observables $\mathcal{SA}_P$, we must first introduce a different characterization of $\mathcal{SA}_P$, obtained by using the new transition system as defined in Table 2.1.

Definition 2.12 *Let P be a program and let G be a goal and let $\longrightarrow_P$ be (the least relation) defined by the rules in Table 2.1. We define*

$$\begin{array}{rl} \mathcal{S}A'_P(G) = & \{\exists_{-Fv(G)}c \mid \langle \tilde{G}, \mathtt{true}, \emptyset\rangle_{n_1} \longrightarrow^{\emptyset}_P \cdots \longrightarrow^{\emptyset}_P \langle \emptyset, c, T_m\rangle_{n_m} \not\longrightarrow^{K}_P\} \\ & \cup \\ & \{\mathtt{false} \mid \ \langle \tilde{G}, \mathtt{true}, \emptyset\rangle_{n_1} \longrightarrow^{\emptyset}_P \cdots \longrightarrow^{\emptyset}_P \langle \tilde{G}', c, T\rangle_{n_m} \textit{ and} \\ & \qquad CT \models c \leftrightarrow \mathtt{false}\}. \end{array}$$

The correspondence of $\mathcal{SA}'$ with the original notion $\mathcal{SA}$ is stated by the following proposition, whose proof is immediate.

Proposition 2.1 *Let P be a program and let G be a goal. Then $\mathcal{SA}_P(G) = \mathcal{SA}'_P(G)$.*

The observables $\mathcal{SA}'_P$, and therefore $\mathcal{SA}_P$, describing answers of successful computations can be obtained from $\mathcal{S}_P$, by considering suitable sequences, namely those sequences which do not perform assumptions either on CHR constraints or on built-in constraints. The first condition means that the second component of tuples (of the sequences $\ldots\langle c, \tilde{K}, \tilde{H}, d\rangle\ldots$) of our compositional semantics must be empty, while the second one means that the assumed constraint at step i must be equal to the produced constraint of steps $i-1$. We call "connected" those sequences which satisfy these requirements.

Definition 2.13 (Connected sequences) *Let $\sigma = \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1\rangle \ldots \langle c_m, \tilde{H}_m, T\rangle \in \mathcal{D}$. We say that σ is connected if for each j, $1 \leq j \leq m-1$, $\tilde{K}_j = \emptyset$ and $d_j = c_{j+1}$.*

The proof of the following result derives from the definition of the connected sequence and an easy inductive argument. If $\sigma = \langle c_1, \tilde{K}_1, \tilde{H}_1, d_1\rangle \ldots \langle c_m, \tilde{H}_m, T\rangle$ is a sequence, we denote by $instore(\sigma)$ and $store(\sigma)$ the built-in constraints c_1 and c_m, respectively and by $lastg(\sigma)$ the goal $\tilde{H}_m$.

Proposition 2.2 *Let P be a program and let G be a goal. Then*

$$\begin{array}{rcll} \mathcal{SA}'_P(G) & = & \{\exists_{-Fv(G)}c \mid & \textit{there exists } \sigma \in \mathcal{S}_P(G) \textit{ such that } instore(\sigma) = \emptyset, \\ & & & \sigma \textit{ is connected, } lastg(\sigma) = \emptyset \textit{ and } c = store(\sigma)\} \\ & & & \cup \\ & & \{\texttt{false} \mid & \textit{there exists } \sigma \in \mathcal{S}_P(G) \textit{ such that } instore(\sigma) = \emptyset, \sigma \textit{ is} \\ & & & \textit{connected and } CT \models store(\sigma) \leftrightarrow \texttt{false}\}. \end{array}$$

The following corollary follows immediately from Proposition 2.1.

Corollary 2.1 (Correctness) *Let P be a program and let G be a goal. Then*

$$\begin{array}{rcll} \mathcal{SA}_P(G) & = & \{\exists_{-Fv(G)}c \mid & \textit{there exists } \sigma \in \mathcal{S}_P(G) \textit{ such that } instore(\sigma) = \emptyset, \\ & & & \sigma \textit{ is connected, } lastg(\sigma) = \emptyset \textit{ and } c = store(\sigma)\} \\ & & & \cup \\ & & \{\texttt{false} \mid & \textit{there exists } \sigma \in \mathcal{S}_P(G) \textit{ such that } instore(\sigma) = \emptyset, \sigma \textit{ is} \\ & & & \textit{connected and } CT \models store(\sigma) \leftrightarrow \texttt{false}\}. \end{array}$$

Chapter 3

Program Transformation: Unfolding in CHR

Program transformation was initially developed to assist in writing correct and efficient programs. This could be done in two phases: The first one consisted in writing a possibly inefficient program whose correctness was simple to prove. The second phase was the sound transformation of the considered program, following a safe methodology to increase efficiency. Nowadays, program transformation is also used to prove properties like non-termination and deadlock-freeness.

Burstall and Darlington [BD77] advocated the program transformation approach. Said approach did not usually consist in a single transformation step. In fact, many intermediate steps were usually performed until the final one is reached. So, let P_0 be the initial program, a sequence of equivalent programs $\langle P_0, \ldots, P_n \rangle$ is created. Such equivalence is of a semantic nature. This means that, given a fixed input, the same output is found when P_i is applied with $0 \leq i \leq n$. Said property is represented by $Sem(P_j) = Sem(P_{j+1})$ with $0 \leq j < n - 1$ where Sem is the function that associates its semantics with every program. Figure 3.1 depicts a sequence of program transformations.

The original target of the transformation depicted above (Figure 3.1) would be an

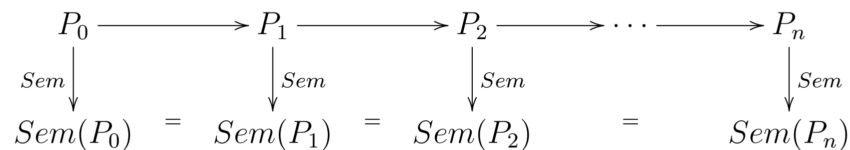

Figure 3.1: Equivalent program transformation

improvement of the considered program, in terms of space or time complexity. This can also be seen in the following Example 3.1.

Example 3.1 Various implementations of the Fibonacci sequence give an example of program transformation. Let us recall the mathematical definition as introduced in [BPS04] before proceeding further.

$$\begin{cases} F_{k+2} = F_{k+1} + F_k \text{ with } (k \geq 1) \\ F_0 = 1 \\ F_1 = 1 \end{cases}$$

```
function FiboR(N: LongInt): LongInt;
begin
   if(N < 2) then
      FiboR := 1
   else
      begin
         FiboR := FiboR(N-1) + FiboR(N-2);
      end;
end;
```

```
function FiboIter(N: LongInt): LongInt;
Var I, A, B, C: LongInt;
begin
   if(N < 2) then
      FiboIter:=1
   else
      begin
         A:=1;
         B:=1;
         for I:=3 to N do
             begin
                C:=A+B;
                A:=B;
                B:=C;
             end;
         FiboIter:=C;
      end;
end;
```

The recursive version `FiboR` has time complexity $O(2^n)$ and space complexity $O(n)$ while the iterative version `FiboIter` has time complexity $O(n)$ and space complexity $O(1)$. □

An example of how the performances can be improved using program transformation is now proposed.

Example 3.2 Let us suppose that a program $P = \{r_1, r_2\}$ is given where

$$r_1 @ f(X,Y), f(Y,Z) \Leftrightarrow g(X,Z)$$
$$r_2 @ g(X,Z) \Leftrightarrow gs(Z,X)$$

and where the constraint $f(X, Y)$ represents the relation "X is the father of Y", afterwards the constraint $g(X, Z)$ means "X is the grandfather of Z" and finally the constraint $gs(Z, X)$ stands for "Z is the grandson of X".

We can observe that rule r_1 can be substituted by the rule

$$r_1'@f(X, Y), f(Y, Z) \Leftrightarrow gs(Z, X)$$

where the body of r_1' is substituted by that of r_2. This can happen because only rule r_2 can be applied after the application of rule r_1, so, the intermediate CHR constraint $g(X, Z)$ can be directly replaced by the final one with respect to the computation, that is $gs(Z, X)$.

Naturally, the application of only one instead of two rules could lead to an improvement in performance.

□

The experience of the scientific community has shown that program transformation is a very valuable methodology, especially for the task of "programming in the small". This can be achieved by subdividing a big program into small modules that will cooperate together to obtain the task of the main program. Such small tasks can be efficiently optimized [PP96].

Naturally, the first experiences in program transformation considered the imperative programming paradigm but subsequently also transformation rules for declarative ones were defined.

Tamaki and Sato [TS84] was the first paper that proposed a general framework for the fold/unfold transformations of programs, written using a Declarative Logic Language. Pettorossi and Proietti [PP96] suggested further transformation for Logic Programs (LP). Such transformations can be found with some adaptation also in the more recent languages that can be considered a derivation of LP like Constraint Logic Programming (CLP) [JM94] and Concurrent Constraint Programming (CCP) [Sar93]. Examples of transformation techniques for the above mentioned languages can be found in [FPP04] for CLP and in [EGM01] for CCP.

Only a few papers, notably [FH03, Frü04, SSD05b], consider source to source transformation of CHR programs. This is not surprising, since program transformation is in

general very difficult for (logic) concurrent languages and in the case of CHR it is even more complicated, as we shall discuss later.

While [Frü04] focuses on specialization of a program for a given goal, here we consider unfolding. This is a basic operation of any source to source transformation (and specialization) system and essentially consists in the replacement of a procedure-call by its definition. Said call consists in a constraint with which a CLP rule can unify and considering CHR it is a conjunction of CHR constraints with which a CHR rule can match. While this operation can be performed rather easily for sequential languages, and indeed in the field of logic programming, it was first investigated by Tamaki and Sato more than twenty years ago [TS84], whereby considering logic concurrent languages it becomes quite difficult to define reasonable conditions which ensure its correctness. This is mainly due to the presence of guards in the rules: Intuitively, when unfolding a rule r by using a rule v (i.e. when replacing in the body of r a "call" of a procedure by its definition v) it could happen that some guard in v is not satisfied statically (i.e. when we perform the unfold), even though it could become satisfied later when the unfolded rule is actually used. If we move the guard of v in the unfolded version of r we can then lose some computations (because the guard is anticipated). This means that if we want to preserve the meaning of a program we cannot replace the rule r by its unfolded version, and we have to keep both the rules. Another source of difficulties consists in the matching substitution mechanism. Only the variables in the atoms of the head of a rule r can be instantiated to become equal to the goal terms following the previous mechanism. On the other hand, the unification mechanism also permits instantiation of the variables in the atoms of the goal. Considering the matching substitution, the deletion of r, when a rule v could be used to unfold r if strong enough hypotheses were considered, they could cause computation loss also if r were unfolded by another rule v'. Finally, the situation for CHR is further complicated by the presence of multiple heads in the rules. In fact, as was recalled in Section 1.1.3, unlike CLP, the head of CHR rule can be composed of more than one constraint or atom. Due to this particular characteristic, we say that CHR rules have multiple heads. Let B be the body of a rule r and let H be the (multiple) head of a rule v, which can be used to unfold r, we cannot be sure that at run-time all the atoms in H will be used

to rewrite B, since in general B could be in a conjunction with other atoms even though the guards are satisfied. This technical point, that one could currently legitimately find obscure, will be further clarified in Chapter 3.4.

Despite these technical problems, the study of unfolding techniques for concurrent languages, and for CHR in particular, is important as it could lead to significant improvements in the efficiency and in non-termination analysis of programs.

In this chapter we will define an unfolding rule for CHR programs and show that it preserves the semantics of the program in terms of qualified answers (a notion already defined in the literature and previously introduced in Definition 1.3). We also provide a syntactic condition which allows us to replace a rule by its unfolded version in a program while preserving qualified answers. Even though the idea of the unfolding is straightforward, its technical development is complicated by the presence of guards, multiple heads and matching substitution, as previously mentioned. In particular, conditions which allow us to replace the original rule by its unfolded version, are not immediately identified. Moreover, a further complication arises from the fact that we consider the reference semantics (called ω_t), as defined in [DSGdlBH04] which avoids trivial non-termination by using a, so called, token store or history (see Subsection 1.5.2). Due to the presence of this token store, in order to define the unfolding correctly we have to slightly modify the syntax of CHR programs by adding a local token store to each rule. The resulting programs are called annotated and we define their semantics by providing a (slightly) modified version of the semantics ω_t, which is proven to preserve the qualified answers. Finally, the maintenance of termination and confluence of property between the original and the ones, which are modified following the above techniques, is proven.

Some of the results where already published in [TMG07].

3.1 CHR annotated syntax

As introduced in 1.5.2, a *simpagation* rule can simulate both simplification and propagation rule by considering, respectively, either H_1 or H_2 empty (with $(H_1, H_2) \neq \emptyset$). In the following pages we will go on to consider in the formal treatment only simpagation rules.

When considering unfolding we need to consider a slightly different syntax, where rule identifiers are not necessarily unique, atoms in the body are associated with an identifier, that is unique to the rule, and where each rule is associated with a local token store T.

Definition 3.1 CHR ANNOTATED SYNTAX. *Let us define a token as an object of the form $r@i_1, \ldots, i_l$, where r is the name of a rule and $i_1, \ldots, i_l$ is a sequence of identifiers. A token store (or history) is a set of tokens (Subsection 1.5.2).*
*An **annotated** rule has then the following form:*

$$r@H_1 \setminus H_2 \Leftrightarrow C \mid \tilde{B}; T$$

where r is an identifier, H_1 and H_2 are sequences of user-defined constraints, $\tilde{B}$ is a sequence of (not-identified) built-in and identified CHR constraints such that different (occurrences of) CHR constraints have different identifiers, and T is a token store, called the local token store of rule r. An annotated CHR program consists of a finite set of annotated CHR rules.

Having introduced the annotated syntax for CHR, some definitions, previously introduced in Section 1.3, are now reproposed. The extensions of said definitions, which are needed in this chapter, are now given. In the following sections, the functions *chr(h#i)=h* and the overloaded function *id(h#i)=i*, [and $id(r@i_1, \ldots, i_l) = \{i_1, \ldots, i_l\}$] possibly extended to sets and sequences of identified CHR constraints [or tokens] in the obvious way will be used. Given a goal G, we denote by $\tilde{G}$ one of the possible identified versions of G. $Goals$ is the set of all (possibly identified) goals.

Intuitively, identifiers are used to distinguish different occurrences of the same atom in a rule. The identified atoms can be obtained by using a suitable function which associates a (unique) integer with each atom. More precisely, let B be a goal which contains m CHR-constraints. We assume that the function $I_n^{n+m}(B)$ identifies each CHR constraint in B by associating a unique integer in $[n+1, m+n]$ with it according to the lexicographic order.

On the other hand, the token store allows us to memorize some tokens, where each token describes which (propagation) rule has been used for reducing which identified atoms. As we previously discussed in Section 1.5, the use of this information was originally proposed in [Abd97] and then further elaborated upon in the semantics defined in [DSGdlBH04], in order to avoid trivial non-termination, arising from the repeated application of the same propagation rule to the same constraints. Here we simply incorporate this information in the syntax, since we will need to manipulate it in our unfolding rule.

Given a CHR program P, by using the function $I_n^{n+m}(B)$ and an initially empty local token store, we can construct its annotated version as follows.

Definition 3.2 *Let P be a CHR program. Then its annotated version is defined as follows:*

$$\begin{aligned} Ann(P) = \{ \quad & r@H_1 \setminus H_2 \Leftrightarrow C \mid I_0^m(B); \emptyset \mid \\ & r@H_1 \setminus H_2 \Leftrightarrow C \mid B \in P \textit{ and} \\ & m \textit{ is the number of CHR-constraints in } B \quad \}. \end{aligned}$$

Notation

In the following examples, given a (possibly annotated) rule

$$r@H_1 \setminus H_2 \Leftrightarrow C \mid B(;T),$$

we write it as

$$r@H_2 \Leftrightarrow C \mid B(;T),$$

if H_1 is empty and we write it as

$$r@H_1 \Rightarrow C \mid B(;T),$$

if H_2 is empty.

That is, we also maintain the notation previously introduced for simplification and propagation rules. Moreover, if $C = \texttt{true}$, then $\texttt{true} \mid$ is omitted. Finally, if in an annotated rule the token store is empty, we simply omit it.

3.2 The modified semantics ω_t'

This section introduces a slightly different operational semantics with respect to the one proposed in Subsection 1.5.2, called ω_t', which considers annotated programs and which will be used to prove the correctness of our unfolding rules (via a specific form of equivalence between ω_t' and ω_t).

We now define the semantics ω_t' which considers annotated rules. This semantics differs from ω_t in two aspects.

First, in ω_t' the goal store and the CHR store are fused in a unique generic *store*, where CHR constraints are immediately labelled. As a consequence, we do not need the Introduce rule anymore and every CHR constraint in the body of an applied rule is immediately usable for rewriting.

The second difference concerns the shape of the rules. In fact, each annotated rule r has a local token store (which can be empty) that is associated with it and which is used to keep trace of the propagation rules that are used to unfold the body of r. Note also that here, unlike the case of the propagation history in ω_t, the token store associated with the real computation can be updated, by adding more tokens at once (because an unfolded rule with many tokens in its local token store has already been used).

In order to define ω_t' formally we need a function $inst$ which updates the formal identifiers of a rule to the actual computation ones and which is defined as follows:

Definition 3.3 *Let $Token$ be the set of all possible token set and let $\mathbb{N}$ be the set of natural numbers. We denote by $inst : Goals \times Token \times \mathbb{N} \to Goals \times Token \times \mathbb{N}$ the function such that $inst(\tilde{B}, T, n) = (\tilde{B}', T', m)$, where*

- *$\tilde{B}$ is an identified CHR goal,*
- *$(\tilde{B}', T')$ is obtained from $(\tilde{B}, T)$ by incrementing each identifier in $(\tilde{B}, T)$ with n and*
- *m is the highest identifier in $(\tilde{B}', T')$.*

We now describe the operational semantics ω'_t for annotated CHR programs by using, as usual, a transition system

$$T_{\omega'_t} = (Conf'_t, \longrightarrow_{\omega'_t}).$$

Configurations in $Conf'_t$ are tuples of the form $\langle \tilde{S}, C, T\rangle_n$ with the following meaning. $\tilde{S}$ is the set of identified CHR constraints that can be matched with rules in the program P and built-in constraints. The built-in constraint store C is a conjunction of built-in constraints and T is a set of tokens, while the counter n represents the last integer which was used to number the CHR constraints in $\tilde{S}$.

Given a goal G, the *initial configuration* has the form

$$\langle I_0^m(G), \mathtt{true}, \emptyset\rangle_m,$$

where m is the number of CHR constraints in G. A *final configuration* has either the form $\langle \tilde{S}, \mathtt{false}, T\rangle_n$ when it has *failed* or it has the form $\langle \tilde{S}, C, T\rangle_n$ when it represents a successful termination, since there are no more applicable rules.

The relation $\longrightarrow_{\omega'_t}$ (of the transition system of operational semantics ω'_t) is defined by the rules in Table 3.1. Let us now briefly discuss the rules.

Solve' moves a built-in constraint from the store to the built-in constraint store;

Apply' uses the rule $r@H'_1\backslash H'_2 \Leftrightarrow D \,|\, \tilde{B}; T_r$ provided that there exists a matching substitution θ such that $chr(\tilde{H}_1, \tilde{H}_2) = (H'_1, H'_2)\theta$, D is entailed by the built-in constraint store of the computation (θ is considered) and $r@id(\tilde{H}_1, \tilde{H}_2) \notin T$; $\tilde{H}_2$ is replaced by $\tilde{B}$, where the identifier are suitably incremented by $inst$ function and $chr(\tilde{H}_1, \tilde{H}_2) = (H'_1, H'_2)$ is added to built-in constraint store.

In order to show the equivalence of the semantics ω_t and ω'_t we will use the notion of observables called "qualified answers" which is introduced in 1.5.2.

Analogously, we can define the qualified answer of an annotated program.

Definition 3.4 (QUALIFIED ANSWERS FOR ANNOTATED PROGRAMS). *Let P be an annotated CHR program and let G be a goal with m CHR constraints. The set $\mathcal{QA}'_P(G)$ of*

Solve' $\dfrac{CT \models c \wedge C \leftrightarrow C' \text{ and } c \text{ is a built-in constraint}}{\langle \{c\} \cup \tilde{G}, C, T\rangle_n \longrightarrow_{\omega_t'} \langle \tilde{G}, C', T\rangle_n}$

Apply' $\dfrac{\begin{array}{c} r@H_1' \backslash H_2' \Leftrightarrow D \mid \tilde{B}; T_r \in P, \quad x = Fv(H_1', H_2') \\ CT \models C \rightarrow (\exists_x (chr(\tilde{H}_1, \tilde{H}_2) = (H_1', H_2')) \wedge D), \quad r@id(\tilde{H}_1, \tilde{H}_2) \notin T \end{array}}{\langle \tilde{H}_1 \cup \tilde{H}_2 \cup \tilde{G}, C, T\rangle_n \longrightarrow_{\omega_t'} \langle \tilde{B}' \cup \tilde{H}_1 \cup \tilde{G}, (chr(\tilde{H}_1, \tilde{H}_2) = (H_1', H_2') \wedge C, T'\rangle_m}$

where $(\tilde{B}', T_r', m) = inst(\tilde{B}, T_r, n)$ and $T' = T \cup \{r@id(\tilde{H}_1, \tilde{H}_2)\} \cup T_r'$ if $\tilde{H}_2 = \emptyset$ otherwise $T' = T \cup T_r'$.

Table 3.1: The transition system $T_{\omega_t'}$ for ω_t' semantics

qualified answers for the query G in the annotated program P is defined as follows:

$$\begin{array}{l} \mathcal{QA}'_P(G) = \\ \quad \{\exists_{-Fv(G)} K \wedge d \mid \langle I_0^m(G), \mathtt{true}, \emptyset\rangle_m \rightarrow^*_{\omega_t'} \langle \tilde{K}, d, T\rangle_n \not\rightarrow_{\omega_t'}\} \\ \quad \cup \\ \quad \{\mathtt{false} \mid \langle I_0^m(G), \mathtt{true}, \emptyset\rangle_m \rightarrow^*_{\omega_t'} \langle \tilde{G}', \mathtt{false}, T\rangle_n\}. \end{array}$$

The following definition is introduced to describe the equivalence of two intermediate states and it is only used in the proofs. We consider two states equivalent when they are identical with the possible exception of the renaming of local variables and renaming of identifiers and logical equivalence of built-in constraints.

Definition 3.5 (INTER-SEMANTICS STATE EQUIVALENCE) *Let $\sigma = \langle (H_1, C), \tilde{H}_2, D, T\rangle_n$ be a state in the transition system ω_t and let $\sigma' = \langle (\tilde{K}, C), D, T'\rangle_m$ be a state in the transition system ω_t'.*

States σ and σ' are equivalent *and we write $\sigma \equiv \sigma'$ if:*

1. *$\tilde{K}_1$ and $\tilde{K}_2$ exist, whereby $\tilde{K} = \tilde{K}_1 \cup \tilde{K}_2$, $H_1 = chr(\tilde{K}_1)$ and $chr(\tilde{H}_2) = chr(\tilde{K}_2)$,*
2. *for each $l \in id(\tilde{K}_1)$, l does not occur in T',*
3. *there exists a renaming of identifier ρ s.t. $T\rho = T'$ and $\tilde{H}_2\rho = \tilde{K}_2$.*

The following result shows the equivalence of the two introduced semantics proving the equivalence (with respect to Definition 3.5) of intermediate states. The proof is easy by definition of ω_t and ω_t'.

Proposition 3.1 *Let P and $Ann(P)$ be respectively a CHR program and its annotated version. Then, for every goal G,*

$$\mathcal{QA}_P(G) = \mathcal{QA}'_{Ann(P)}(G)$$

holds.

PROOF. By definition of $\mathcal{QA}$ and of $\mathcal{QA}'$, the initial states of the two transition systems are equivalent. Then, we have only to show that any transition step from any state in one system can be imitated from an equivalent transition step, originating in an equivalent state in the other system, in order to achieve an equivalent state.

Let $\sigma = \langle (H_1, C), \tilde{H}_2, D, T\rangle_n$ be a state in the transition system ω_t and let $\sigma' = \langle (\tilde{K}, C), D, T'\rangle_m$ be a state in the transition system ω_t', such that $\sigma \equiv \sigma'$

Solve and Solve': they move a built-in constraint from the Goal store or the Store respectively to the built-in constraint store. In this case let $C = C' \cup \{c\}$. According to the definition of the two transition systems

$$\sigma \longrightarrow_{\omega_t}^{Solve} \langle (H_1, C'), \tilde{H}_2, D \wedge c, T\rangle_n \text{ and } \sigma' \longrightarrow_{\omega_t'}^{Solve'} \langle (\tilde{K}, C'), D \wedge c, T'\rangle_m.$$

According to the definition of $\equiv$, it is easy to check that $\langle (H_1, C'), \tilde{H}_2, D \wedge c, T\rangle_n \equiv \langle (\tilde{K}, C'), D \wedge c, T'\rangle_m$.

Introduce: this kind of transition exists only in ω_t semantics and its application labels a CHR constraint into the goal store and moves it into the CHR store. In this case, let $H_1 = H_1' \uplus \{h\}$ and

$$\sigma \longrightarrow_{\omega_t}^{Introduce} \langle (H_1', C), \tilde{H}_2 \cup \{h\#n\}, D, T\rangle_{n+1}.$$

Let us denote $\tilde{H}_2 \cup \{h\#n\}$ by $\tilde{H}_2'$. As in the definition of $\equiv$, $\tilde{K}_1$ and $\tilde{K}_2$ exist whereby $\tilde{K} = \tilde{K}_1 \cup \tilde{K}_2$, $H_1 = chr(\tilde{K}_1)$ and $chr(\tilde{H}_2) = chr(\tilde{K}_2)$. Therefore, an

identified atom $h\#m \in \tilde{K}_1$ exists. Let $n' = \rho(n)$ (where $n' = n$ if n is not in the domain of ρ). Through construction and hypothesis, $\tilde{K}'_1 = \tilde{K}_1 \setminus \{h\#m\}$ and $\tilde{K}'_2 = \tilde{K}_2 \setminus \{h\#m\}$ are such that $\tilde{K} = \tilde{K}'_1 \cup \tilde{K}'_2$, $H'_1 = chr(\tilde{K}'_1)$ and $chr(\tilde{H}'_2) = chr(\tilde{K}'_2)$.

Moreover, according to the definition of $\equiv$, for each $l \in id(\tilde{K}_1)$, l does not occur in T'. Therefore, since by construction $\tilde{K}'_1 \subseteq \tilde{K}_1$, we can say that for each $l \in id(\tilde{K}'_1)$, l does not occur in T'.

Now, to prove that $\sigma' \equiv \langle (H'_1, C), \tilde{H}'_2, D, T\rangle_{n+1}$, we have only to prove that there exists a renaming ρ', such that $T\rho' = T'$ and $\tilde{H}'_2\rho' = \tilde{K}'_2$.

We can consider the new renaming $\rho' = \rho \circ \{n'/m, m/n'\}$. By definition ρ' is a renaming of identifiers. Since by construction, $m \notin id(\tilde{K}_2)$, we can observe that if there exists $m'/m \in \rho$, then $m' \notin id(\tilde{H}_2)$. Moreover, since $m \notin id(\tilde{K}_2)$, if there is no $m/m' \in \rho$ then $m \notin id(\tilde{H}_2)$. As a result of the previous observations, we can conclude that $\tilde{H}'_2\rho' = \tilde{H}_2\rho \cup \{h\#n\}\{n/m\} = \tilde{K}'_2$. Finally, since n does not occur in T, we can say that $T\rho' = T\rho\{m/n'\} = T'\{m/n'\}$, where the last equality follows by hypothesis. Moreover, since $m \in id(\tilde{K}_1)$, we can conclude that m does not occur in T'. Therefore, $T'\{m/n'\} = T'$ and then the thesis follows.

Apply: Let $r@F' \backslash F'' \Leftrightarrow D_1 \,|\, B, C_1 \in P$ and $r@F' \backslash F'' \Leftrightarrow D_1 \,|\, \tilde{B}, C_1 \in Ann(P)$ be its annotated version which can be applied to the considered state $\sigma' = \langle (\tilde{K}, C), D, T'\rangle_m$, in particular F', F'' match respectively with $\tilde{P}_1$ and $\tilde{P}_2$. Without loss of generality, by using a suitable number of Introduce steps, we can assume that $r@F' \backslash F'' \Leftrightarrow D_1 \,|\, B, C_1 \in P$ can be applied to $\sigma = \langle (H_1, C), \tilde{H}_2, D, T\rangle_n$. In particular, we can assume for $i = 1, 2$, there exists $\tilde{Q}_i \subseteq \tilde{H}_2$ such that $\tilde{Q}_i\rho = \tilde{P}_i$ and F', F'' match respectively with $\tilde{Q}_1$ and $\tilde{Q}_2$.

According to the definition of $\equiv$, $\tilde{P}_3$ and $\tilde{Q}_3$ exist such that $\tilde{Q}_3\rho = \tilde{P}_3$, $\tilde{K}_2 = \tilde{P}_1 \cup \tilde{P}_2 \cup \tilde{P}_3$, $\tilde{H}_2 = \tilde{Q}_1 \cup \tilde{Q}_2 \cup \tilde{Q}_3$ and let $x = Fv(\tilde{P}_1, \tilde{P}_2) = Fv(\tilde{Q}_1, \tilde{Q}_2)$.

Through construction, since $T\rho = T'$ and $(\tilde{P}_1, \tilde{P}_2) = (\tilde{Q}_1, \tilde{Q}_2)\rho$, we have that

- $r@id(\tilde{P}_1, \tilde{P}_2) \notin T'$ if and only if $r@id(\tilde{Q}_1, \tilde{Q}_2) \notin T$ and

- $CT \models D \rightarrow \exists_x(((F', F'') = chr(\tilde{P}_1, \tilde{P}_2)) \wedge D_1)$ if and only if $CT \models D \rightarrow \exists_x(((F', F'') = chr(\tilde{Q}_1, \tilde{Q}_2)) \wedge D_1)$.

Therefore, in accordance with the definition of **Apply** and of **Apply'**

$$\sigma \longrightarrow_{\omega_t}^{Apply} \langle \{H_1, C\} \uplus \{B, C_1\}, (\tilde{Q}_1, \tilde{Q}_3), ((F', F'') = chr(\tilde{Q}_1, \tilde{Q}_2)) \wedge D, T_1 \rangle_n$$

if and only if

$$\sigma' \rightarrow_{\omega_t'}^{Apply'} \langle (\tilde{K}_1, \tilde{P}_1, \tilde{P}_3, C, \tilde{B}', C_1), ((F', F'') = chr(\tilde{P}_1, \tilde{P}_2)) \wedge D, T_1' \rangle_o$$

where

- $T' = T \cup \{r@id(\tilde{Q}_1)\}$ if $\tilde{Q}_2 = \emptyset$, otherwise $T_1 = T$,
- $(\tilde{B}', \emptyset, o) = inst(\tilde{B}, \emptyset, m)$ and
- $T_1' = T' \cup \{r@id(\tilde{P}_1)\}$ if $\tilde{Q}_2 = \emptyset$, otherwise $T_1' = T'$.

Let $\sigma_1 = \langle \{H_1, C\} \uplus \{B, C_1\}, (\tilde{Q}_1, \tilde{Q}_3), ((F', F'') = chr(\tilde{Q}_1, \tilde{Q}_2)) \wedge D, T_1 \rangle_n$ and $\sigma_1' = \langle (\tilde{K}_1, \tilde{P}_1, \tilde{P}_3, \tilde{B}', C, C_1), ((F', F'') = chr(\tilde{P}_1, \tilde{P}_2)) \wedge D, T_1' \rangle_o$.

Now, to prove the thesis, we have to prove that $\sigma_1 \equiv \sigma_1'$.

The following holds.

1. $\tilde{K}_1' = (\tilde{K}_1, \tilde{B}')$ and $\tilde{K}_2' = (\tilde{P}_1, \tilde{P}_3)$ exist, whereby $(\tilde{K}_1, \tilde{P}_1, \tilde{P}_3, \tilde{B}') = \tilde{K}_1' \cup \tilde{K}_2'$, $H_1 \uplus B = chr(\tilde{K}_1')$ and $chr(\tilde{Q}_1, \tilde{Q}_3) = chr(\tilde{K}_2')$.
2. Since for each $l \in id(\tilde{K}_1)$, l does not occur in T', $\tilde{P}_1 \subseteq \tilde{K}_2$ and by definition of Apply' transition, we can conclude that for each $l \in id(\tilde{K}_1') = id(\tilde{K}_1, \tilde{B}')$, l does not occur in T_1',
3. Through construction and since $T\rho = T'$, we can further conclude that $T_1\rho = T_1'$. Moreover, as a result of construction $(\tilde{Q}_1, \tilde{Q}_3)\rho = (\tilde{P}_1, \tilde{P}_3) = \tilde{K}_2'$.

As per the definition, we arrive at the following conclusion, namely that $\sigma_1 \equiv \sigma_1'$ and then the thesis.

□

3.3 The unfolding rule

In this section we define the *unfold operation* for CHR simpagation rules. As a particular case we also obtain unfolding for simplification and propagation rules, as these can be seen as particular cases of the former.

The unfolding allows us to replace a conjunction S of constraints (which can be seen as a procedure-call) in the body of a rule r by the body of a rule v, provided that the head of v matches with S. More precisely, assume that the head H of v, instantiated by a substitution θ, matches with the conjunction S (in the body of r). Then the unfolded rule is obtained from r, by performing the following steps: 1) the new guard in the unfolded rule is the conjunction of the guard of r with the guard of v, the latter instantiated by θ and without those constraints that are entailed by the built-in constraints which are in r, 2) the body of v and the equality $H = S$ are added to the body of r (equality here is interpreted as syntactic equality), 3) the conjunction of constraints S can be removed, partially removed or left in the body of the unfolded rule, depending on the fact that v is a simplification, a simpagation or a propagation rule, respectively, 4) as for the local token store T_r associated to every rule r, this is updated consistently during the unfolding operations, in order to avoid that a propagation rule is used twice to unfold the same sequence of constraints.

Before formally defining the unfolding we need to define the function

$$clean : Goals \times Token \rightarrow Token,$$

as follows: $clean(\tilde{B}, T)$ deletes from T all the tokens for which at least one identifier is not present in the identified goal $\tilde{B}$. More formally

$$clean(\tilde{B}, T) = \{t \in T \mid t = r@i_1, \ldots, i_k \text{ and } i_j \in id(\tilde{B}), \text{ for each } j \in [1, k]\}.$$

Let us also recall at this juncture that we defined *chr(h#i)=h*.

Definition 3.6 (UNFOLD). *Let P be an annotated CHR program and let $r, sp \in P$ be two annotated rules such that:*

$$\begin{aligned} r@H_1 \backslash H_2 &\Leftrightarrow D \mid \tilde{K}, \tilde{S}_1, \tilde{S}_2, C; T \textit{ and} \\ sp@H_1' \backslash H_2' &\Leftrightarrow D' \mid \tilde{B}; T' \textit{ is a simpagation rule} \end{aligned}$$

where $chr(\tilde{S}_1, \tilde{S}_2)$ *is identical to* $(H'_1, H'_2)\theta$*, that is, the constraints* H'_1 *in the head of rule* sp *match with* $chr(\tilde{S}_1)$ *and* H'_2 *matches with* $chr(\tilde{S}_2)$ *by using the substitution* θ*. Furthermore, assume that* C *is the conjunction of all the built-in constraints in the body of* r*, that* m *is the highest identifier which appears in the rule* r *and that* $(\tilde{B}_1, T_1, m_1) = inst(\tilde{B}, T', m)$*. Then the* unfolded *rule is:*

$$r@H_1 \backslash H_2 \Leftrightarrow D, (D''\theta) \,|\, \tilde{K}, \tilde{S}_1, \tilde{B}_1, C, chr(\tilde{S}_1, \tilde{S}_2) = (H'_1, H'_2); T''$$

where $sp@id(\tilde{S}_1, \tilde{S}_2) \notin T$*,* $D'' = D' \backslash V$*,* $V \subseteq D'$ *where* V *is the biggest possible set such that* $CT \models C \wedge D \rightarrow V\theta$*, the constraint* $(D, (D''\theta))$ *is satisfiable and*

- *if* $H'_2 = \emptyset$ *then* $T'' = clean((\tilde{K}, \tilde{S}_1), T) \cup T_1 \cup \{sp@id(\tilde{S}_1)\}$
- *if* $H'_2 \neq \emptyset$ *then* $T'' = clean((\tilde{K}, \tilde{S}_1), T) \cup T_1$.

The reader can notice that, after the application of the previous unfolding definition, a new rule labelled with r is created. Said rule is usually introduced into the CHR program which also contains the initial rule r. This means that, after unfolding, the rule identifiers are usually not unique. We point out that the function $inst$ (defined in Definition 3.3) is used in order to increment the value of the identifiers associated with atoms in the unfolded rule. This allows us to distinguish the new identifiers introduced in the unfolded rule from the old ones. We should also note that the condition of the token store is needed to obtain a correct rule. Consider for example a ground annotated program $P = \{r_1@h \Leftrightarrow \tilde{k},\, r_2@k \Rightarrow \tilde{s},\, r_3@s, s \Leftrightarrow \tilde{B}\}$ and let h be the start goal. In this case the unfolding could change the semantics if the token store were not used. In fact, according to the semantics proposed in Table 1.3 or 3.1, we have the following computation: $\tilde{h} \rightarrow^{(r_1)} \tilde{k} \rightarrow^{(r_2)} \tilde{k}, \tilde{s} \not\rightarrow_{\omega_t}$. On the other hand, considering an unfolding without the update of the token store one would have $r_1@h \Leftrightarrow \tilde{k} \overset{\text{unfold using } r_2}{\longrightarrow} r_1@h \Leftrightarrow \tilde{k}, \tilde{s} \overset{\text{unfold using } r_2}{\longrightarrow} \underline{r_1@h \Leftrightarrow \tilde{k}, \tilde{s}, \tilde{s}} \overset{\text{unfold using } r_3}{\longrightarrow} r_1@h \Leftrightarrow \tilde{k}, \tilde{B}$ so, starting from the constraint h we could arrive at constraint k, B, that is not possible in the original program (the clause obtained after the wrongly applied unfolding rule is underlined).

As previously mentioned, the unfolding rules for simplification and propagation can be obtained as particular cases of Definition 3.6, by setting $H_1' = \emptyset$ and $H_2' = \emptyset$, respectively, and by considering the resulting unfolded rule accordingly. In the following examples we will use $\odot$ to denote both $\Leftrightarrow$ and $\Rightarrow$. The rule names will use the overlined notation to point out a close resemblance between the rules themselves.

Example 3.3 The following program $P = \{r_1, r_2, \bar{r}_2\}$ deduces information about genealogy. Predicate f is considered as father, g as grandfather, gs as grandson and gg as great-grandfather. The following rules are such that we can unfold some constraints in the body of r_1 using the rule r_2 $[\bar{r}_2]$.

$$r_1 @ f(X,Y), f(Y,Z), f(Z,W) \odot g(X,Z)\#1, f(Z,W)\#2, gs(Z,X)\#3.$$
$$r_2 @ g(X,Y), f(Y,Z) \odot gg(X,Z)\#1.$$
$$\bar{r}_2 @ g(X,Y) \backslash f(Y,Z) \Leftrightarrow gg(X,Z)\#1.$$

Now we unfold the body of rule r_1 by using the rule r_2 where we assume $\odot = \Leftrightarrow$ (so we have a simplification rule). We use $inst(gg(X,Z)\#1, \emptyset, 3) = (gg(X,Z)\#4, \emptyset, 4)$ and a renamed version of r_2

$$r_2 @ g(X',Y'), f(Y',Z') \Leftrightarrow gg(X',Z')\#1.$$

in order to avoid variable clashes. So the new unfolded rule is:

$$r_1 @ f(X,Y), f(Y,Z), f(Z,W) \odot gg(X',Z')\#4, gs(Z,X)\#3, X' = X, Y' = Z, Z' = W.$$

Now, we unfold the body of rule r_1 by using the simpagation rule $\bar{r}_2$. As before,

$$inst(gg(X,Z)\#1, \emptyset, 3) = (gg(X,Z)\#4, \emptyset, 4)$$

and a renamed version of $\bar{r}_2$

$$\bar{r}_2 @ g(X',Y') \backslash f(Y',Z') \Leftrightarrow gg(X',Z')\#1.$$

is used to avoid variable clashes. The new unfolded rule is:

$$r_1 @ f(X,Y), f(Y,Z), f(Z,W) \odot g(X,Z)\#1,$$
$$gg(X',Z')\#4, gs(Z,X)\#3, X' = X, Y' = Z, Z' = W.$$

Finally, we unfold the body of r_1, by using the r_2 rule where $\odot = \Rightarrow$ is assumed (so we have a propagation rule). As usual, $inst(gg(X,Z)\#1, \emptyset, 3) = (gg(X,Z)\#4, \emptyset, 4)$ and a renamed version of r_2 is used to avoid variable clashes such as:

$$r_2@g(X',Y'), f(Y',Z') \Rightarrow gg(X',Z')\#1.$$

and so the new unfolded rule is:

$$\begin{aligned} r_1@f(X,Y), f(Y,Z), f(Z,W) \odot g(X,Z)\#1, \\ f(Z,W)\#2, gs(Z,X)\#3, gg(X',Z')\#4, X' = X, Y' = Z, Z' = W; \{r_2@1,2\}. \end{aligned}$$

□

The following example considers more specialized rules with guards which are not `true`.

Example 3.4 The following program $P = \{r_1, r_2, \bar{r}_2\}$ specializes in the rules introduced in Example 3.3 to the genealogy of Adam. So here we remember that Adam was the father of Seth, Seth was the father of Enosh, Enosh was the father of Kenan. As before, we consider the predicate f as father, g as grandfather, gs as grandson and gg as great-grandfather.

$$\begin{aligned} &r_1@f(X,Y), f(Y,Z) f(Z,W) \odot X = Adam, Y = Seth \,| \\ &\qquad g(X,Z)\#1, f(Z,W)\#2, gs(Z,X)\#3, Z = Enosh. \\ &r_2@g(X,Y), f(Y,Z) \odot X = Adam, Y = Enosh \,|\, gg(X,Z)\#1, Z = Kenan. \\ &\bar{r}_2@g(X,Y) \backslash f(Y,Z) \Leftrightarrow X = Adam, Y = Enosh \,|\, gg(X,Z)\#1, Z = Kenan. \end{aligned}$$

If we unfold r_1, by using (a suitably renamed version of) r_2, where we assume $\odot = \Leftrightarrow$, we obtain:

$$\begin{aligned} &r_1@f(X,Y), f(Y,Z) f(Z,W) \odot X = Adam, Y = Seth \,|\, gg(X',Z')\#4, Z' = Kenan, \\ &\qquad gs(Z,X)\#3, Z = Enosh, X' = X, Y' = Z, Z' = W. \end{aligned}$$

When $\bar{r}_2$ is considered to unfold r_1 we have

$$\begin{aligned} &r_1@f(X,Y), f(Y,Z) f(Z,W) \odot X = Adam, Y = Seth \,|\, g(X,Z)\#1, gg(X',Z')\#4, \\ &\qquad Z' = Kenan, gs(Z,X)\#3, Z = Enosh, X' = X, Y' = Z, Z' = W. \end{aligned}$$

Finally if we assume $\odot \Longrightarrow$ in r_2 from the unfolding we obtain

$$\begin{aligned} r_1@f(X,Y), f(Y,Z), f(Z,W) \odot X = Adam, Y = Seth \,|\, g(X,Z)\#1, f(Z,W)\#2, \\ gs(Z,X)\#3, gg(X',Z')\#4, Z' = Kenan, Z = Enosh, X' = X, Y' = Z, \\ Z' = W; \{r_2@1,2\}. \end{aligned}$$

Note that $X' = Adam, Y' = Enosh$ are not added to the guard of the unfolded rule because $X' = Adam$ is entailed by the guard of r_1 and $Y' = Enosh$ is entailed by the built-in constraints in the body of r_1. □

We will now prove the correctness of our unfolding definition. Let r' be the rule obtained by the unfolding of rule r, using rule v. Let the state $\sigma^t_{r'}$ be obtained after the application of the rule r' and let the state σ^f_v be obtained, using the rules r and v. The proof of the following proposition is done by showing the equivalence between the states $\sigma^t_{r'}$ and σ^f_v. Before the introduction of the proposition, three new definitions are given. The first one presents the concept of the built-in free state. Said state has no built-in constraints in the store.

Definition 3.7 (BUILT-IN FREE STATE) *Let $\sigma = \langle \tilde{G}, D, T \rangle_o$ be a state. The state σ is built-in free, if either $\tilde{G}$ is a multiset of CHR-constraints or $D =$ `false`.*

The second definition introduces the state equivalence between states, obtained using our ω'_t modified semantics transition system which was introduced in Table 3.1. Note that in such a definition, the equivalence operator is represented by the symbol $\simeq$.

Definition 3.8 (STATE EQUIVALENCE) *Let $\sigma = \langle \tilde{G}, D, T \rangle_o$ and $\sigma' = \langle \tilde{G}', D', T' \rangle_o$ be built-in free states. σ and σ' are equivalent and we write $\sigma \simeq \sigma'$ if one of the following facts hold.*

- *either $D =$ `false` and $D' =$ `false`*
- *or $\tilde{G} = \tilde{G}'$, $CT \models D \leftrightarrow D'$ and $clean(\tilde{G}, T) = clean(\tilde{G}', T)$.*

The above definition, could be generalized. In fact, given a state, different fresh variables can be chosen to rename the variables of a rule. For our purposes, we suppose that two states, have the same new variables, where possible.

Finally, the third definition presents the normal derivation. A derivation is called normal if no other $Solve'$ transition is possible when an $Apply'$ one happens.

Definition 3.9 (NORMAL DERIVATION) *Let P be an annotated CHR program and let $\delta = \langle \tilde{G}, c, T\rangle_m \rightarrow^*_{\omega'_t} \langle \tilde{K}, d, T'\rangle_n \not\rightarrow_{\omega'_t}$ be a derivation in P: We say that δ is normal if it uses a transition Solve' as soon as possible, namely it is possible to use a transition Apply' on a state σ only if σ is built-in free.*

Note that, by definition, given an annotated program P, $\mathcal{QA}'(P)$ can be calculated by considering only normal derivations. Analogously for an annotated CHR program P'. The proof of the following proposition, which is about the equality bethween the [prime] qualified answer, obtained using a general derivation and the ones obtained using only normal derivation, is straightforward and hence it is omitted.

Proposition 3.2 *Let P be CHR program and let P' an annotated CHR program. Then*

$$\begin{array}{rcl}\mathcal{QA}_P(G) & = & \{\exists_{-Fv(G)}K \wedge d \mid \delta = \langle G, \emptyset, \mathtt{true}, \emptyset\rangle_1 \rightarrow^*_{\omega_t} \langle \emptyset, \tilde{K}, d, T\rangle_n \not\rightarrow_{\omega_t} \\ & & \quad \textit{and } \delta \textit{ is normal}\} \\ & & \cup \\ & & \{\mathtt{false} \mid \delta = \langle G, \emptyset, \mathtt{true}, \emptyset\rangle_1 \rightarrow^*_{\omega_t} \langle G', \tilde{K}, \mathtt{false}, T\rangle_n \\ & & \quad \textit{and } \delta \textit{ is normal}\}\end{array}$$

and

$$\begin{array}{rcl}\mathcal{QA}'_P(G) & = & \{\exists_{-Fv(G)}K \wedge d \mid \delta = \langle I_0^m(G), \mathtt{true}, \emptyset\rangle_m \rightarrow^*_{\omega'_t} \langle \tilde{K}, d, T\rangle_n \not\rightarrow_{\omega'_t} \\ & & \quad \textit{and } \delta \textit{ is normal}\} \\ & & \cup \\ & & \{\mathtt{false} \mid \delta = \langle I_0^m(G), \mathtt{true}, \emptyset\rangle_m \rightarrow^*_{\omega'_t} \langle \tilde{G}', \mathtt{false}, T\rangle_n \\ & & \quad \textit{and } \delta \textit{ is normal}\}.\end{array}$$

Proposition 3.3 *Let r, v be annotated CHR rules and r' be the result of the unfolding of r with respect to v. Let σ be a generic built-in free state such that we can use the transition $Apply'$ with the clause r', obtaining the state $\sigma_{r'}$ and then the built-in free state $\sigma^f_{r'}$. Then we can construct a derivation which uses at most the clauses r and v and obtain a built-in free state σ^f such that $\sigma^f_{r'} \simeq \sigma^f$.*

PROOF. Assume that

$$\begin{array}{ll} \sigma & \longrightarrow^{r'} \sigma_{r'} \longrightarrow^{Solve^*} \sigma^f_{r'} \\ & \searrow_r \sigma_r \longrightarrow^{Solve^*} \sigma^f_r (\longrightarrow^{v} \sigma_v \longrightarrow^{Solve^*} \sigma^f_v) \end{array}$$

The labeled arrow $\longrightarrow^{Solve^*}$ means that only solve transitions are applied. Moreover,

- if σ^f_r has the form $\langle \tilde{G}, \mathtt{false}, T\rangle$ then the derivation between the parenthesis is not present and $\sigma^f = \sigma^f_r$.
- the derivation between the parenthesis is present and $\sigma^f = \sigma^f_v$, otherwise.

Preliminaries: Let $\sigma = \langle(\tilde{H}_1, \tilde{H}_2, \tilde{H}_3), C, T\rangle_j$ be a built-in free state and let $r@H'_1\backslash H'_2 \Leftrightarrow D_r \mid \tilde{K}, \tilde{S}_1, \tilde{S}_2, C_r; T_r$ and $v@S'_1\backslash S'_2 \Leftrightarrow D_v \mid \tilde{P}, C_v; T_v$ where $chr(\tilde{S}_1, \tilde{S}_2)$ is identical to $(S'_1, S'_2)\theta$. Furthermore, assume that m is the greatest identifier which appears in the rule r and that $inst(\tilde{P}, T_v, m) = (\tilde{P}_1, T_1, m_1)$. Then the *unfolded* rule is:

$$r'@H'_1\backslash H'_2 \Leftrightarrow D_r, (D'_v\theta) \mid \tilde{K}, \tilde{S}_1, \tilde{P}_1, C_r, C_v, chr(\tilde{S}_1, \tilde{S}_2) = (S'_1, S'_2); T_2$$

where $v@id(\tilde{S}_1, \tilde{S}_2) \notin T_r$, $D'_v = D_v\backslash V$, V is the biggest set such that $V \subseteq D_v$ and

$$CT \models C_r \wedge D_r \rightarrow V\theta, \tag{3.1}$$

the constraint $(D_r, (D'_v\theta))$ is satisfiable and

- if $S'_2 = \emptyset$ then $T_2 = clean((\tilde{K}, \tilde{S}_1), T_r) \cup T_1 \cup \{v@id(\tilde{S}_1)\}$
- if $S'_2 \neq \emptyset$ then $T_2 = clean((\tilde{K}, \tilde{S}_1), T_r) \cup T_1$.

The proof: By definition of the transition $Apply'$, we can say that

$$CT \models C \rightarrow \exists_x((chr(\tilde{H}_1, \tilde{H}_2) = (H'_1, H'_2)) \wedge D_r \wedge (D'_v\theta)), \quad (3.2)$$

where $x = Fv(H'_1, H'_2)$ and

$$\sigma_{r'} = \langle(\tilde{Q}, C_r, C_v, chr(\tilde{S}_1, \tilde{S}_2) = (S'_1, S'_2)), chr(\tilde{H}_1, \tilde{H}_2) = (H'_1, H'_2) \wedge C, T_3\rangle_{j+m_1},$$

where $\tilde{Q} = (\tilde{H}_1, \tilde{H}_3, \tilde{Q}_1)$, with $inst((\tilde{K}, \tilde{S}_1, \tilde{P}_1), T_2, j) = (\tilde{Q}_1, T'_2, j + m_1)$ and

- if $H'_2 = \emptyset$ then $T_3 = T \cup T'_2 \cup \{r@id(\tilde{H}_1)\}$
- if $H'_2 \neq \emptyset$ then $T_3 = T \cup T'_2$.

Therefore, by definition

$$\sigma^f_{r'} = \langle\tilde{Q}, C^f_{r'}, T_3\rangle_{j+m_1}.$$

where

$$CT \models C^f_{r'} \leftrightarrow C_r \wedge C_v \wedge chr(\tilde{S}_1, \tilde{S}_2) = (S'_1, S'_2) \wedge chr(\tilde{H}_1, \tilde{H}_2) = (H'_1, H'_2) \wedge C.$$

On the other hand, since through (3.2),

$$CT \models C \rightarrow \exists_x((chr(\tilde{H}_1, \tilde{H}_2) = (H'_1, H'_2)) \wedge D_r)$$

by definition of the transition $Apply'$, we can observe that

$$\sigma_r = \langle(\tilde{Q}_2, C_r), chr(\tilde{H}_1, \tilde{H}_2) = (H'_1, H'_2) \wedge C, T_4\rangle_{j+m},$$

where $\tilde{Q}_2 = (\tilde{H}_1, \tilde{H}_3, \tilde{K}'', \tilde{S}''_1, \tilde{S}''_2)$,
$((\tilde{K}'', \tilde{S}''_1, \tilde{S}''_2), T'_r, j + m) = inst((\tilde{K}, \tilde{S}_1, \tilde{S}_2), T_r, j)$ and

- if $H'_2 = \emptyset$ then $T_4 = T \cup T'_r \cup \{r@id(\tilde{H}_1)\}$
- if $H'_2 \neq \emptyset$ then $T_4 = T \cup T'_r$.

Therefore, as per the definition

$$\sigma^f_r = \langle\tilde{Q}_2, C^f_r, T_4\rangle_{j+m}.$$

where

$$CT \models C^f_r \leftrightarrow C_r \wedge chr(\tilde{H}_1, \tilde{H}_2) = (H'_1, H'_2) \wedge C.$$

We now have two possibilities

($C_r^f = \texttt{false}$). In this case, through construction we can conclude that $C_{r'}^f = \texttt{false}$. Therefore $\sigma_{r'}^f \simeq \sigma_r^f$ and then the thesis is proven.

($C_r^f \neq \texttt{false}$). According to the definition, (3.1), (3.2) and since $chr(\tilde{S}_1, \tilde{S}_2) = chr(\tilde{S}_1'', \tilde{S}_2'')$ is identical to $(S_1', S_2')\theta$, we can further conclude that

$$\begin{aligned} CT \models \;& (C_r \wedge chr(\tilde{H}_1, \tilde{H}_2) = (H_1', H_2') \wedge C) \rightarrow \\ & (\exists_y((chr(\tilde{S}_1, \tilde{S}_2) = (S_1', S_2')) \wedge D_v)), \end{aligned}$$

where $y = Fv(S_1', S_2')$ and

$$\begin{aligned} \sigma_v = \;& \langle (Q_3, C_v), chr(\tilde{S}_1, \tilde{S}_2) = (S_1', S_2') \wedge C_r \wedge \\ & chr(\tilde{H}_1, \tilde{H}_2) = (H_1', H_2') \wedge C,\; T_5 \rangle_{j+m_1}, \end{aligned}$$

where $\tilde{Q}_3 = (\tilde{H}_1, \tilde{H}_3, \tilde{K}'', \tilde{S}_1'', \tilde{P}_2)$, with $inst(\tilde{P}, T_v, j+m) = (\tilde{P}_2, T_v', j+m_1)$ and

- if $S_2' = \emptyset$ then $T_5 = T_4 \cup T_v' \cup \{v@id(\tilde{S}_1'')\}$
- if $S_2' \neq \emptyset$ then $T_5 = T_4 \cup T_v'$.

Finally, according to the definition, we can conclude that

$$\sigma_v^f = \langle \tilde{Q}_3, C_v^f,\; T_5 \rangle_{j+m_1},$$

whereby

$$C_v^f \leftrightarrow C_v \wedge chr(\tilde{S}_1, \tilde{S}_2) = (S_1', S_2') \wedge C_r \wedge chr(\tilde{H}_1, \tilde{H}_2) = (H_1', H_2') \wedge C.$$

If $C_v^f = \texttt{false}$ then the proof is analogous to the previous case and hence it is omitted. Otherwise, observe that through construction, $\tilde{Q} = (\tilde{H}_1, \tilde{H}_3, \tilde{Q}_1)$, where $\tilde{Q}_1$ is obtained from $(\tilde{K}, \tilde{S}_1, \tilde{P}_1)$, by adding the natural j to each identifier in $(\tilde{K}, \tilde{S}_1)$ and by adding the natural $j+m$ to each identifier in $\tilde{P}$. Analogously, through construction, $\tilde{Q}_3 = (\tilde{H}_1, \tilde{H}_3, \tilde{K}'', \tilde{S}_1'', \tilde{P}_2)$, where $(\tilde{K}'', \tilde{S}_1'')$ are obtained from $(\tilde{K}, \tilde{S}_1)$, by adding the natural j to each identifier in $(\tilde{K}, \tilde{S}_1)$ and $\tilde{P}_2$ is obtained from $\tilde{P}$, by adding the natural $j+m$ to each identifier in $\tilde{P}$.

Therefore $\tilde{Q} = \tilde{Q}_3$.

Now, to prove the thesis, we have only to prove that

$$clean(T_3, \tilde{Q}) = clean(T_5, \tilde{Q}_3).$$

Let us introduce the function $up : \{Token\} \times \mathbb{N} \longrightarrow \mathbb{N}$ as the function that adds to each identifier of every token of the set $\{Token\}$ the value $\mathbb{N}$. So

$$\begin{aligned} T_3 &= T \cup T_2'[\cup\{r@id(\tilde{H}_1)\}] \\ &= T \cup up(T_2, j)[\cup\{r@id(\tilde{H}_1)\}] \\ &= T \cup up((clean((\tilde{K}, \tilde{S}_1), T_r) \cup T_1 \cup [\{v@id(\tilde{S}_1)\}]), j)[\cup\{r@id(\tilde{H}_1)\}] \\ &= T \cup up((clean((\tilde{K}, \tilde{S}_1), T_r) \cup up(T_v, m) \cup [\{v@id(\tilde{S}_1)\}]), j) \\ &\quad [\cup\{r@id(\tilde{H}_1)\}] \end{aligned}$$

$$\begin{aligned} T_5 &= T_4 \cup T_v'[\cup\{v@id(S_1'')\}] \\ &= T \cup T_r'[\cup\{r@id(\tilde{H}_1)\}] \cup up(T_v, j+m)[\cup\{v@id(S_1'')\}] \\ &= T \cup up(T_r, j)[\cup\{r@id(\tilde{H}_1)\}] \cup up(T_v, j+m)[\cup\{up(\{v@id(\tilde{S}_1)\}, j)\}] \end{aligned}$$

The proof follows immediately when we consider that $\tilde{Q} = \tilde{Q}_3$ and the result of the application of function $clean$.

□

We will now go on to prove the correctness of our unfolding rule.

Proposition 3.4 *Let P be an annotated CHR program with $r, v \in P$. Let r' be the result of the unfolding of r with respect to v and let P' be the program obtained from P by adding rule r'. Then, for every goal G, $\mathcal{QA}'_{P'}(G) = \mathcal{QA}'_P(G)$ holds.*

PROOF. We have only to prove that $\mathcal{QA}'_{P'}(G) \subseteq \mathcal{QA}'_P(G)$. The proof of the other inclusion is an obvious consequence of the operational semantics of CHR, since in a computation step one may apply any applicable rule.

The proof follows from Propositions 3.2 and 3.3 and by a straightforward inductive argument.

□

The proof of the following result is obtained immediately from the previously mentioned proposition and Proposition 3.1.

Corollary 3.1 (Correctness) *Let P be CHR program and let $Ann(P)$ be its annotated version (as previously defined). Let P' be the program obtained from $Ann(P)$ by adding a rule which is obtained, by unfolding a rule in $Ann(P)$. Then, for every G, $\mathcal{QA}'_{P'}(G) = \mathcal{QA}_P(G)$ holds.*

PROOF. Proposition 3.1 proves that $\mathcal{QA}_P(G) = \mathcal{QA}'_{Ann(P)}(G)$ where $Ann(P) = P''$ and P'' is a program with no unfolded rules. Proposition 3.4 proves that $\mathcal{QA}'_{P''}(G) = \mathcal{QA}'_{P'}(G)$ where P'' is an annotated program, that could also contain no unfolded rules and P' is an annotated program with at least one unfolded rule. And hence the proof.
□

3.4 Safe rule replacement

The previously mentioned corollary shows that we can safely add to a program P a rule resulting from the unfolding, while preserving the semantics of P (in terms of qualified answers). However, when a rule r in program P has been unfolded producing the new rule r', in some cases we would also like to replace r by r' in P, since this could improve the efficiency of the resulting program. Performing such a replacement while preserving the semantics is in general a very difficult task for three reasons.

First of all, anticipating the guard of v in the guard of r (as we do in the unfold operation) could lead to a loss of some computations when the unfolded rule r' is used, rather than the original rule r. This is shown by the following example.

Example 3.5 Consider the program

$$\begin{array}{lll} P = & \{ & r_1@g(X,Y) \Leftrightarrow s(Z,X), f(Z,Y), X = Adam. \\ & & r_2@s(M,N) \Leftrightarrow M = Seth \,|\, . \\ & & r_3@f(O,P) \Leftrightarrow O = Seth, P = Enosh. \qquad \} \end{array}$$

where we do not consider the identifiers (and the local token store) in the body of rules, because we do not have propagation rules in P.

It is clear that the goal $g(Q, R)$ has a successful derivation which computes $Q = X, X = Adam, R = Y, Y = P, P = Enosh, Z = O, O = Seth, Z = M, X = N$, by using the above program. On the other hand, this is not the case if one uses the program

$$
\begin{aligned}
P' = \quad \{ \quad & r'_1 g(X, Y) \Leftrightarrow M = Seth \,|\, f(Z, Y), X = Adam, Z = M, Y = N. \\
& r_2 @ s(M, N) \Leftrightarrow M = Seth \,|\, . \\
& r_3 @ f(O, P) \Leftrightarrow O = Seth, P = Enosh. \qquad \}
\end{aligned}
$$

where the rule r'_1 is obtained, by unfolding the rule r_1, using the rule r_2. For this reason deleting the unfolded rule in general is not safe. □

The second problem is related to multiple heads. In fact, the unfolding that we have defined assumes that the head of a rule matches completely with the body of another one, whereas in general, during a CHR computation, a rule can match with constraints produced by more than one rule and/or introduced by the initial goal. The following example illustrates this point.

Example 3.6 Let us consider the program

$$
\begin{aligned}
P = \{ \quad & r @ g(X, Y) \Leftrightarrow f(X, Z), s(Y, Z). \\
& r' @ f(V, W), s(W, M) \Leftrightarrow V = M, U = V. \quad \}
\end{aligned}
$$

where we do not consider the identifiers and the token store in the body of rules, because we do not have propagation rules in P.

The unfolding of r by using r' returns the new rule

$$r @ g(X, Y) \Leftrightarrow V = X, W = Z, W = Y, M = Z, V = M, U = V.$$

Now the program

$$
\begin{aligned}
P' = \{ \quad & r @ g(X, Y) \Leftrightarrow V = X, W = Z, W = Y, M = Z, V = M, U = V. \\
& r' @ f(V, W), s(W, M) \Leftrightarrow V = M. \qquad \}
\end{aligned}
$$

where we substitute the original rule by its unfolded version is not semantically equivalent to P. In fact, given the goal $G = (g(B,C), f(Seth, Enosh), s(Seth, Adam))$, when the program P is considered, the following derivation can be obtained

$$\langle(\underline{g(B,C)}, f(Seth, Enosh), s(Seth, Adam)), \emptyset\rangle \rightarrow^{r} \rightarrow^{Solve^*}$$
$$\langle(f(X,Z), \underline{s(Y,Z)}, \underline{f(Seth, Enosh)}, s(Seth, Adam)), (X = B, Y = C)\rangle \rightarrow^{r'} \rightarrow^{Solve^*}$$
$$\langle(\underline{f(X,Z)}, \underline{s(Seth, Adam)}), (X = B, Y = C, V = Y, W = Z, U = Seth,$$
$$M = Enosh, V = M, U = W)\rangle \rightarrow^{r'} \rightarrow^{Solve^*}$$
$$\langle(\emptyset, (X = B, Y = C, V = Y, W = Z, U = Seth, M = Enosh, V = M, U = W,$$
$$V' = X, W' = Z, U' = Seth, M' = Adam, V' = M', U' = W')\rangle.$$

On the other hand, when P' is considered, only rule r can be applied in order not to obtain an inconsistent computation:

$$\langle(\underline{g(B,C)}, f(Seth, Enosh), s(Seth, Adam)), \emptyset\rangle \rightarrow^{r} \rightarrow^{Solve^*}$$
$$\langle(f(Seth, Enosh), s(Seth, Adam)), (B = X, C = Y, V = M, U = W, X = V,$$
$$Z = W, Y = V, Z = M)\rangle.$$

The underscored CHR constraints indicates the constraint considered for the application of the chosen CHR rule.

Now, we can finally maintain that $\mathcal{QA}'_{P'}(G) \neq \mathcal{QA}'_P(G)$. □

The final problem is related to the matching substitution. In fact, following Definition 3.6, there are some matchings that could become possible only at run time, and not at compile time, because a more powerful built-in constraint store would be needed. Also in this case, a rule elimination could lead to a loss of possible answers, as illustrated in the following example.

Example 3.7 Let P be a program

$$\begin{aligned} P \ = \{ \ & r_1@g(X,Y) \Leftrightarrow f(X,Z), s(Y,Z). \\ & r_2@f(Adam, W) \Leftrightarrow W = Enosh. \\ & r_3@f(T,J) \Leftrightarrow s(J,T). \qquad \} \end{aligned}$$

where we do not consider the identifiers and the token store in the body of rules, because we do not have propagation rules in P. Let P' be the program where the rule r_1, that is unfolded, using r_3 in P, substitutes the original r_1 (note that other unfoldings are not possible, in particular the rule r_2 can not be used to unfold r_1)

$$\begin{array}{lll} P' & = \{ & r_1@g(X,Y) \Leftrightarrow s(J,T), s(Y,Z), X = T, Z = J. \\ & & r_2@f(Adam, W) \Leftrightarrow W = Enosh. \\ & & r_3@f(T,J) \Leftrightarrow s(J,T). \qquad \} \end{array}$$

Let be $G = g(Adam, R)$ the goal, we can see that the state $\sigma = \langle s(Y,Z), (X = Adam, R = Y, W = Z, W = Enosh,)\rangle$ can be reached when program P is considered. On the other hand a state σ' such that $\sigma \simeq \sigma'$ can not be reached from program P' because, with the considered goal (and consequently the considered built-in constraint store) r_2 can fire in P but can not fire in P'. Naturally $\mathcal{QA}'_{P'}(G) \neq \mathcal{QA}'_P(G)$. □

We have found a case in which we can safely replace the original rule r by its unfolded version, while maintaining the qualified answers semantics. Intuitively, this holds when: 1) the constraints of the body of r can be rewritten only by CHR rules with a single-head, 2) no rule v exists which has a multiple head H such that a part of H can match with a part of the constraints introduced in the body of r (that is, there exists no rule v which can be fired, by using a part of the constraints introduced in the body of r plus some other constraints) and 3) all the rules, that can be applied at run time to the body of the original rule r, can also be applied at transformation time (so unfolding avoidance for built-in constraint store and guard-anticipation problems are solved).

Before formally defining these conditions we need some further notations. First of all, given a rule $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$, we define two sets. The first one contains a set of pairs, whose first component is a rule that can be used to unfold $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$, while the second one is the sequence of the identifiers of the atoms in the body of r, which are used in the unfolding.

The second set contains all the rules that can be used for the *partial unfolding* of $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$, that is the set of rules that can fire by using at least an atom in the body $\tilde{A}$ of the rule and some others CHR and built-in constraints. Furthermore, it contains

the rules that can fire if an opportune built-in constraint store is given by the computation but that can not be unfolded according to Definition 3.6.

Definition 3.10 *Let P be an annotated CHR program and let*

$$r@H_1\backslash H_2 \;\Leftrightarrow\; D \,|\, \tilde{A}; T \text{ and}$$
$$r'@H_1'\backslash H_2' \;\Leftrightarrow\; D' \,|\, \tilde{B}; T'$$

be two annotated rules, such that $r, r' \in P$ and r' is renamed separately with respect to r. We define U_P^+ and $U_P^{\#}$ as follows:

1. *$(r'@H_1'\backslash H_2' \Leftrightarrow D' \,|\, \tilde{B}; T', (i_1, \ldots, i_n)) \in U_P^+(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T)$ if and only if $r@H_1\backslash H_2 \;\Leftrightarrow\; D \,|\, \tilde{A}; T$ can be unfolded with $r'@H_1'\backslash H_2' \;\Leftrightarrow\; D' \,|\, \tilde{B}; T'$ (by Definition 3.6), by using the sequence of the identified atoms in $\tilde{A}$ with identifiers $(i_1, \ldots, i_n)$.*

2. *$r'@H_1'\backslash H_2' \;\Leftrightarrow\; D' \,|\, \tilde{B}; T' \in U_P^{\#}(r@H_1\backslash H_2 \;\Leftrightarrow\; D \,|\, \tilde{A}; T)$ if and only if one of the following holds:*

   (a) either there exists $\tilde{A}' = (\tilde{A}_1, \tilde{A}_2) \subseteq \tilde{A}$ and a built in constraint C' such that $Fv(C') \cap Fv(r') = \emptyset$, the constraint $D \wedge C'$ is satisfiable, $CT \models (D \wedge C') \rightarrow \exists_x((chr(\tilde{A}_1, \tilde{A}_2) = (H_1', H_2')) \wedge D')$, $r'@id(\tilde{A}_1, \tilde{A}_2) \notin T$ with $x \in Fv(r')$ and $(r'@H_1'\backslash H_2' \Leftrightarrow D' \,|\, \tilde{B}; T', id(\tilde{A}_1, \tilde{A}_2)) \notin U_P^+(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T)$

   (b) or there exist $\tilde{A}' \subseteq \tilde{A}$, a multiset of CHR constraints $H' \neq \emptyset$ and a built in constraint C' such that $\tilde{A}' \neq \emptyset$, $Fv(C') \cap Fv(r') = \emptyset$, the constraint $D \wedge C'$ is satisfiable, $\{chr(A'), H'\} = \{K_1, K_2\}$ and $CT \models (D \wedge C') \rightarrow \exists_x(((K_1, K_2) = (H_1', H_2')) \wedge D')$ with $x \in Fv(r')$.

Some explanations are in order here.

The set U^+ contains all the tuples composed by rules, that can be used to unfold a fixed rule r, and the identifiers of the constraints considered in the unfolding, introduced in Definition 3.6.

Let us now consider the set $U^{\#}$. The conjunction of built-in constraints C' represents a generic set of built-in constraints (said set naturally can be equal to every possible built-in constraint store that can be generated by a real computation before the application of rule r'). The condition $Fv(C') \cap Fv(r') = \emptyset$ is required to avoid free variable capture. It represents the fresh variable renaming of a rule r' with respect to the computation before the use of the r' itself in an Apply transition. The condition $r'@id(\tilde{A}_1, \tilde{A}_2) \notin T$ grants the propagation rules trivial non-termination avoidance. The conditions $CT \models (D \wedge C') \rightarrow \exists_x((chr(\tilde{A}_1, \tilde{A}_2) = (H_1', H_2')) \wedge D')$ and $CT \models (D \wedge C') \rightarrow \exists_x(((K_1, K_2) = (H_1', H_2')) \wedge D')$ ensure that a strong enough built-in constraint is possessed by the computation, before the application of rule r'. The conditions $A_1' \neq \emptyset$ and $H' \neq \emptyset$ assure respectively that at least one constraint in the body of rule r and that at least one constraint form the initial goal or introduced by the body of other rules are unfolded. Finally, the following condition $(r'@H_1' \backslash H_2' \Leftrightarrow D' | \tilde{B}; T', id(\tilde{A}_1, \tilde{A}_2)) \notin U_P^+(r@H_1 \backslash H_2 \Leftrightarrow D \mid \tilde{A}; T)$ is required to avoid the consideration of the rules that can be correctly unfolded in the body of r. There are two kinds of rules that are added to $U^{\#}$. The first one, introduced by Example 3.7, points out the matching substitution problem (Condition 2a of Definition 3.10). The second kind, introduced by Example 3.6, points out the multiple heads problem: the rule r' can match with the body of r but also can match with other constraints introduced by the initial goal or generated by other rules (Condition 2b of Definition 3.10).

Note also that if $U_P^+(r@H_1 \backslash H_2 \Leftrightarrow D \mid \tilde{A}; T)$ contains a pair, whose first component is not a rule with a single atom in the head, then by definition, $U_P^{\#}(r@H_1 \backslash H_2 \Leftrightarrow D \mid \tilde{A}; T) \neq \emptyset$.

Finally, given an annotated CHR program P and an annotated rule $r@H_1 \backslash H_2 \Leftrightarrow D \mid \tilde{A}; T$, we can define

$$Unf_P(r@H_1 \backslash H_2 \Leftrightarrow D \mid \tilde{A}; T)$$

as the set of all annotated rules obtained by unfolding the rule $r@H_1 \backslash H_2 \Leftrightarrow D \mid \tilde{A}; T$ with a rule in P, by using Definition 3.6.

We now can give the central definition of this section.

Definition 3.11 (SAFE RULE REPLACEMENT) *Let P be an annotated CHR program and let $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T \in P$, such that the following holds*

i) $U_P^{\#}((r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T) = \emptyset$ *and*

ii) $U_P^{+}(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T) \neq \emptyset$ *and*

iii) for each

$$r@H_1\backslash H_2 \Leftrightarrow D' \,|\, \tilde{A}'; T' \in Unf_P(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T)$$

we can say that $CT \models D \leftrightarrow D'$.

Then we can conclude that the rule $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$ can be safely replaced (by its unfolded version) in P.

Some further explanations are in order here.

Condition i) of the above mentioned definition implies that $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$ can be safely deleted from P only if:

- $U_P^{+}(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T)$ contains only pairs, whose first component is a rule with a single atom in the head.
- a sequence of identified atoms of body of the rule r can be used to fire a rule r' only if r can be unfolded with r', by using the same sequence of the identified atoms.

Condition **ii)** states that at least one rule exists which unfolds the rule $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}$.

Condition **iii)** states that each annotated clause obtained by the unfolding of r in P must have a guard equivalent to that of r: In fact, the condition $CT \models D \leftrightarrow D'$ in **iii)** avoids the problems discussed in Example 3.5, thus allowing the anticipation of the guard in the unfolded rule.

We can now provide the result which shows the correctness of the safe rule replacement condition.

Theorem 3.1 *Let P be an annotated program, $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$ be a rule in P such that $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$ can be safely replaced in P, according to Definition 3.11. Assume also that*

$$P' = (P \setminus \{(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T)\}) \cup Unf_P(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T).$$

Then $\mathcal{QA'}_P(G) = \mathcal{QA'}_{P'}(G)$ for any arbitrary goal G.

PROOF. By using a straightforward inductive argument and according to Proposition 3.4, we can observe that $\mathcal{QA'}_P(G) = \mathcal{QA'}_{P''}(G)$ where

$$P'' = P \cup Unf_P(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T),$$

for any arbitrary goal G.

Then to prove the thesis, we have only to prove that

$$\mathcal{QA'}_{P'}(G) \supseteq \mathcal{QA'}_{P''}(G).$$

Since by hypothesis $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$ can be safely replaced in P, following Definition 3.11 (Safe rule replacement), we can observe that $\tilde{A} = C, \tilde{K}$, where $\tilde{K}$ is a set of (identified) CHR constraints and

$$\tilde{K} \neq \emptyset. \tag{3.3}$$

Now, let σ be a generic built-in free state such that we can use the transition $Apply'$ with the clause $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$, obtain the state σ_r and then the built-in free state σ_r^f.

$$\sigma \quad \longrightarrow_{P''}^{r} \sigma_r \longrightarrow^{Solve^*} \sigma_r^f = \langle \tilde{K}_r, C_r^f, T_r^f \rangle_m,$$

where $chr(\tilde{K}) \subseteq chr(\tilde{K}_r)$ ($\subseteq$ denotes the inclusion in the multisets) and $C_r^f \rightarrow C$. The labelled arrow $\longrightarrow^{Solve^*}$ means that only solve transitions are applied.

We now have two possibilities

($\sigma_r^f \not\longrightarrow_{P''}$) In this case, as per (3.3) and since by definition

$$U_P^+(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T) \neq \emptyset,$$

we can say that

$$CT \models C_r^f \leftrightarrow \texttt{false}. \tag{3.4}$$

Moreover, since $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$ can be safely replaced there exists a clause

$$r@H_1\backslash H_2 \Leftrightarrow D' \,|C, C', \tilde{B}; T' \in Unf_P(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T)$$

and $CT \models D \leftrightarrow D'$. Since by hypothesis, σ is a built-in free state such that we can use the transition $Apply'$, with the clause $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$, we can further say that for the state σ we can use the transition $Apply'$, with the clause $r@H_1\backslash H_2 \Leftrightarrow D' \,|C, C', \tilde{B}; T'$ and then the state σ'_r is obtained and then the built-in free state ${\sigma'}_r^f$ where ${\sigma'}_r^f = \langle \tilde{K}'_r, {C'}_r^f, {T'}_r^f \rangle_{m'}$ and $CT \models {C'}_r^f \leftarrow C_r^f$ Therefore, as per (3.4), $CT \models {C'}_r^f \leftrightarrow \texttt{false}$ and hence the thesis is proven.

($\sigma_r^f \longrightarrow_{P''}^v \sigma'_v \longrightarrow^{Solve^*} {\sigma'}_v^f$) In this case, there exists a clause v in P'', such that we can use the transition $Apply'$, with the clause v, in order to rewrite the state σ_r^f to the state σ'_v. As per the definition of qualified answers and since $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$ (with $\tilde{A} = (C, \tilde{K})$) can be safely replaced, we can assume, without loss of generality, that v is a clause that rewrites at least an atom introduced by the rule $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$ (this holds, since by definition the guard of the clause v is implied by $D \wedge C$). Moreover, since $r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T$ can be safely replaced we can say that v exactly rewrites an atom $\tilde{k} \in \tilde{K}$ by condition i) of Definition 3.11.

Let v be the clause $v@\backslash k' \Leftrightarrow D_v \,|\, \tilde{A}_v; T_v$ (or let v be the propagation clause $[v@k'\backslash \Leftrightarrow D_v \,|\, \tilde{A}_v; T_v]$). By definition and though construction

$$\begin{array}{ll} (r'@H_1\backslash H_2 \Leftrightarrow D \,|\, [\tilde{k}], \tilde{K}, C, \tilde{A}'_v; T'_v \cup T[\cup\{v@id(\tilde{k})\}], id(\tilde{k})) & \in \\ U_P^+(r@H_1\backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T) & \subseteq \\ P' & \end{array}$$

Now, to prove the thesis, we have to prove that

$$\sigma \longrightarrow^{r'} \sigma_{r'} \longrightarrow^{Solve^*} \sigma_{r'}^f \text{ and } {\sigma'}^f \equiv \sigma_{r'}^f.$$

The above mentioned result immediately follows from the definition of unfolding of r, using v and Proposition 3.4.

□

Of course, the result of the previous theorem can be applied to a sequence of program transformations. Let us define such a sequence as follows.

Definition 3.12 (U-sequence) *Let P be an annotated CHR program. An* U-sequence *of programs, starting from P is a sequence of annotated CHR programs $P_0, \ldots, P_n$, such that*

$$\begin{array}{rcl} P_0 & = & P \textit{ and} \\ P_{i+1} & = & (P_i \setminus \{(r@H_1 \backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T)\}) \cup \\ & & Unf_{P_i}(r@H_1 \backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T), \end{array}$$

where $i \in [0, n-1]$, $(r@H_1 \backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T) \in P_i$ and $(r@H_1 \backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T)$ is safety deleting from $P_{i+1} \cup \{r@H_1 \backslash H_2 \Leftrightarrow D \,|\, \tilde{A}; T\}$.

Then from Theorem 3.1 and Proposition 3.1 we immediately reach the following corollary.

Corollary 3.2 *Let P be a program and let $P_0, \ldots, P_n$ be an U-sequence starting from $Ann(P)$. Then $\mathcal{QA}_P(G) = \mathcal{QA'}_{P_n}(G)$ for any arbitrary goal G.*

PROOF. Proposition 3.1 proves that the qualified answer for a program P and its annotated version $P_0 = Ann(P)$, having fixed a start goal, is the same. Theorem 3.1 proves that, having fixed a goal, the qualified answer for program P_0 is equal to the qualified answer for a program P_1 obtained by P_0 adding all the possible unfoldings of a rule $r \in P_0$ and deleting r, supposing that the safe rule replacement condition holds for r. The proof of this Corollary follows by transitivity of equality in fact $\mathcal{QA}_P(G) = \mathcal{QA}'_{P_0}(G)$ with $Ann(P) = P_0$ for Proposition 3.1 and $\mathcal{QA}'_{P_i}(G) = \mathcal{QA}'_{P_{i+1}}(G)$ with $0 \leq i \leq n-1$ for Theorem 3.1.

□

3.5 Confluence and termination

In this section the confluence and termination preservation between an original program and the ones derived using the previous introduced methodology will be analysed.

Before starting to analyse the above mentioned topics, we only want to investigate the termination characteristic of the process, as introduced in Definition 3.12. A simple example can prove that the considered process is generally non-terminant. In fact, let $P = \{r@p \Leftrightarrow p, q\}$ be a program where at least a rule contains in its body a conjunction of constraints that can match with the head of the rule. The rule r can be unfolded ad infinitum. At other times, rules which generate a non-terminating computation can reach a fixpoint if the transfomation proposed in Definition 3.12 is applied as depicted in the following example. Let $P = \{r_1@p \Leftrightarrow q, r_2@q \Leftrightarrow p\}$ be a CHR program, its' fixpoint is the program $P' = \{r_1@p \Leftrightarrow q, r_2@q \Leftrightarrow p, r_1@p \Leftrightarrow p, r_2@q \Leftrightarrow q\}$. In fact, further unfolding would only generate rules that have already been introduced.

Let us now prove that our unfolding preserves confluence and termination.

The formal definition of termination from [Frü04] is introduced and adapted to our ω_t' semantics.

Definition 3.13 (Termination) *A CHR program P is called* terminating*, if there are no infinite computations.*

Definition 3.14 (Normal Termination) *A (possibly annotated) CHR program P is called* normal terminating*, if there are no infinite normal computations.*

Proposition 3.5 (Normal Termination) *Let P be a CHR program and let $P_0, \ldots, P_n$ be an U-sequence, starting from $Ann(P)$. P satisfies normal termination if and only if P_n satisfies normal termination.*

PROOF. First of all observe that by using the same arguments of Proposition 3.1, we can say that P is normal terminating if and only if $Ann(P)$ is normal terminating. Moreover, from Proposition 3.3 and by using a straightforward inductive argument, we can observe that for each $i = 0, \ldots, n-1$, if P_i satisfies normal termination if and only if P_{i+1} satisfies

the normal termination too and then the thesis.

□

When (standard) termination is considered instead of normal termination, program transformation, as defined in Definition 3.12 (U-sequence), can introduce problems connected to the guard elimination process of Definition 3.6 (Unfold), as shown in the following example.

Example 3.8 Let us consider the following program:

$$\begin{aligned} P \quad = \{ \quad & r_1@p(X) \Leftrightarrow \mid X = a, q(X). \\ & r_2@q(Y) \Leftrightarrow Y = a \mid r(Y). \\ & r_3@r(Z) \Leftrightarrow Z = d \mid p(Z). \quad \} \end{aligned}$$

where we do not consider the identifiers and the token store in the body of rules, because we do not have propagation rules in P. Then the following possible unfolded program P' is given. In said program, the previous r_1 is unfolded using r_2 (following Definition 3.6) and the (original clause) $r_1 \in P$ is deleted because safe rule replacement holds, so that the results of Theorem 3.1 can be applied as follows:

$$\begin{aligned} P' \quad = \{ \quad & r_1@p(X) \Leftrightarrow \mid X = a, X = Y, r(Y). \\ & r_2@q(Y) \Leftrightarrow Y = a \mid r(Y). \\ & r_3@r(Z) \Leftrightarrow Z = d \mid p(Z). \qquad \} \end{aligned}$$

It is easy to check that the program P satisfies the (standard) termination. If instead the program P' and the start goal $(V = d, p(V))$ are considered, the following state can be reached

$$\langle (X = a, p(Z)\#3), (V = d, V = X, X = Y, Y = Z), \emptyset \rangle_4$$

where r_1, r_3 (in the order) can be applied infinitely if the built-in constraint $X = a$ is not moved by $Solve'$ rule into the built-in store, where it would be evaluated. This can happen because of the non-determinism in rule application of ω_t' semantics. □

The confluence property guarantees that any computation for a goal results in the same final state, no matter which of the applicable rules are applied [AF04]. This means that

$\mathcal{QA}_P(G)$ has cardinality one for each goal G. The formal definition of confluence from [Frü04] is introduced and adapted to our ω_t' semantics. Confluence is only considered for normal terminating programs. In the following $\mapsto^*$ means either $\longrightarrow_{\omega_t}$ or $\longrightarrow_{\omega_t'}$.

Definition 3.15 (Confluence) *A CHR [annotated] program is* confluent *if for all states $\sigma, \sigma_1, \sigma_2$: if $\sigma \mapsto^* \sigma_1$ and $\sigma \mapsto^* \sigma_2$ then states σ_f' and σ_f'' exist such that $\sigma_1 \mapsto^* \sigma_f'$ and $\sigma_2 \mapsto^* \sigma_f''$ and σ_f' and σ_f'' are identical with the possible exception of renaming of local variables, identifiers and logical equivalence of built-in constraints.*

From the previous definition it follows that σ_f' and σ_f'' are equivalent, so $\sigma_f \simeq \sigma_f'$.

Corollary 3.3 (Confluence) *Let P be a normal terminating CHR program and let $P_0, \ldots, P_n$ be an U-sequence, starting from $Ann(P)$. P satisfies confluence if and only if P_n also satisfies confluence.*

PROOF. We prove only that if P is normal terminating and confluent, then P_n is also confluent. The proof of the converse is similar and hence it is omitted. First of all, observe, that by hypothesis and by Proposition 3.5, we can say that P_n is normal terminating.

Let us assume by contrary that P_n does not satisfy confluence. This means that, there exists a state $\sigma = \langle (\tilde{K}, D), C, T \rangle_o$ such that

$$\sigma \longrightarrow_{\omega_t'}^* \sigma_1 \text{ and } \sigma \longrightarrow_{\omega_t'}^* \sigma_2$$

in P_n and two states σ_1' and σ_2' that are identical with the possibile exception of renaming of local variables, identifiers and logical equivalence of built-in constraints for which $\sigma_1 \longrightarrow_{\omega_t'}^* \sigma_1'$ and $\sigma_2 \longrightarrow_{\omega_t'}^* \sigma_2'$ in P_n do not exist.

Since P_n is normal terminating, there are two normal derivations

$$\sigma_1 \longrightarrow_{\omega_t'}^* \sigma_1^f \not\longrightarrow_{\omega_t'} \text{ and } \sigma_2 \longrightarrow_{\omega_t'}^* \sigma_2^f \not\longrightarrow_{\omega_t'}$$

in P_n, where σ_1^f and σ_2^f are built-in free states.

Then, by using arguments similar to that given in Proposition 3.1 and Proposition 3.3, we can conclude that two normal derivations exists

$$\sigma' \longrightarrow_{\omega_t}^* \sigma_f' \not\longrightarrow_{\omega_t} \text{ and } \sigma' \longrightarrow_{\omega_t}^* \sigma_f'' \not\longrightarrow_{\omega_t}$$

in P, where $\sigma' = \langle D, \tilde{K}, C, T \rangle_{o+1}$, $\sigma'_f \equiv \sigma_1^f$ and $\sigma''_f \equiv \sigma_2^f$. Then, since by hypothesis P is confluent, we can further state that σ'_f and σ''_f are identical with the possible exception of the renaming of local variables, identifiers and logical equivalence of built-in constraints. Therefore, according to the definition of $\equiv$, we obtain a contradiction to the assumption that two states σ'_1 and σ'_2 as previously defined do not exist.

□

Finally, we conjecture that some relation can be found between confluence and qualified answer maintenance. In particular we suppose that if P is a CHR program confluent, Theorem 3.1 further holds without part 2a of Definition 3.10.

Chapter 4

Related work and conclusions

This thesis has introduced both a compositional semantics and a program transformation for CHR programs.

The above mentioned compositional semantics is based on sequences and it is compositional with respect to the and-composition and it is correct as regards to "success answers". The proposed semantics takes into account the theoretical operational semantics called ω_t, as defined in [DSGdlBH04]. The ω_t semantics is refined with respect to the original one introduced by [Frü98]. In fact, token store and univoque identifiers are added respectively to states and CHR constraints. Such ω_t semantics admits propagation rules (simulated by a simpagation one with an empty simplification part) and uses a token store and a unique identifier for every CHR constraint, in order to avoid trivial non-termination.

The need to model within our semantics the token store as well as the identifiers (by means of the identification function $I_n^{n+m}(G)$) constitutes the main technical difference between this thesis and [DGM05]. In fact, [DGM05] simulates a propagation rule using a simplification one. This result can be obtained by creating a simplification rule which has as a head, the head of the propagation one, and as a body, all the constraints of the body of the simplification rule joined with the constraints in the head of the propagation one. In this way the application ad infinitum of a such transformed propagation rule is impossible to avoid if no other simplification rule is applied to the constraints which are both in the body and in the head of the transformed rule. The introduction of fairness hypothesis is worth nothing for such modified propagation rules. This means that propa-

gation rules were not treated in a satisfactory way since the original operational semantics of CHR was considered, thus allowing any propagation rules to introduce trivial infinite computations. On the other hand, the theoretical semantics introduced in [DSGdlBH04] inspires the compositional semantics introduced in this thesis. In fact, a propagation and a simplification rule are transformed into a simpagation one where the right hand side and the left hand side of the head with respect to the "\" symbol are respectively empty. Said compositional semantics, associating an unique identifier to each CHR constraint and maintaining the history of the constraints, to which a propagation like rule is applied, permits trivial non-termination avoidance, without further considering the fairness hypothesis [GMT06].

The presence of multiple heads, as well as the need to model the token store and the related management of the identifiers associated with the CHR constraints, leads to a semantic model which is technically involved, even though the basic idea is simple. However, it is difficult to avoid this complication if one wants to model precisely the observables we are interested in. A simpler model could be obtained by considering a more abstract semantics which characterizes a superset of the observables in question. Such an abstract semantics could be useful for program analysis, along the lines of the abstract interpretation theory. Of course, it would be desirable to introduce within the semantic model the minimum amount of information needed to obtain compositionality, whilst at the same time preserving correctness. In other words, it would be desirable to obtain a fully abstract semantics (for the success answers notion of observables). This will be left to future studies. Even though techniques similar to those used in [dBP91, dBGM97] for CCP could be considered, in the case of CHR, the possibility of removing information considerably complicates (unlike CCP) the problem. Moreover, the fully abstract model would probably retain all the complication of the semantics presented here, further augmented by the need to introduce suitable abstraction operators. Finally, it would be desirable to also obtain a compositional characterization for "qualified answers", by modifying Definition 1.2, in order to also consider computations terminating with a non-empty user-defined constraint. In spite of the obvious interest from a theoretical and practical point of view of a such kind of compositional semantics, there is a hurdle to overcome.

The compositional semantics, as presented in this thesis, is not refined enough to obtain the desired result. In fact, the acceptance of a non-empty final store in the concrete semantics means a non-empty stable atom set in the abstract one. This in general can permit the interleaving with other abstract sequences which possibly present constraints in the assumption set which can be satisfied, using the ones in the stable atom set. This fact introduces a clear difficulty in the determination of when an abstract sequence is terminated. Let us consider as an example of what has been previously said the following one. Let $P = \{r@p, q \Leftrightarrow m\}$ be a CHR program. The sequences of one element $\langle \emptyset, p, \emptyset \rangle \in \mathcal{QA}_P(p)$ and $\langle \emptyset, q, \emptyset \rangle \in \mathcal{QA}_P(q)$ are possible elements of qualified answer set, while their composition $\langle \emptyset, (p, q), \emptyset \rangle$ is not a element of $\mathcal{QA}_P(p, q)$. The introduction of a set, which contains the names of rules that can not be applied from a certain point onwards, could be a possible solution to this problem.

The other topic considered in this thesis concerns the definition of an unfold operation for CHR which preserves the qualified answers of a program. This was obtained by transforming a CHR program into an annotated one which is then unfolded. The equivalence of the unfolded program and the original (non-annotated) one is proven (Corollary 3.1), by using a slightly modified operational semantics for annotated programs (as defined in Section 3.2). We then provided a condition that could be used to safely replace a rule with its unfolded version, whilst simultaneously preserving qualified answers, for a restricted class of rules. Confluence and termination maintenance of the program modified in the previous way with respect to the orginal one are proven.

There are only a few other papers that consider source to source transformation of CHR programs. They will now be introduced in order from the most relevant to the less relevant. Finally, the differences between the program transformation proposed for two logic concurrent languages as GHC and CCP will be expounded.

The most relevant paper on the transformation topic introduced in the thesis is [TMG07]. Said paper presents preliminary results with respect to the ones that are given here.

Another interesting paper about CHR program transformation is [Frü04]. Rather than considering a generic transformation system, it focuses on the specialization of rules regarding a specific goal, analogously to what happens in partial evaluation. In particular,

it defines how to specialize a simplification and a propagation rule with respect to a goal with which the considered rule overlaps, which means that the head of the rule and the goal must have at least one CHR constraint in common. Naturally, a form of correctness is given. In fact, the specialized rules obtained by the given definitions are proven to be redundant and their addition or deletion is also proven to preserve the termination. A rule is said to be redundant for a program if and only if for all possible states, only identical states with the possible exception of the renaming of logical variables and logical equivalence of built-in constraints, can be reached. Said paper does not consider the trivial non-termination problem so identifiers associated with CHR constraints and token store are not introduced. At the end of the paper a section with program transformation examples is introduced. This section is the most relevant for our work. In fact, some examples about unfolding are given. Naturally, the trivial non-termination problem is not considered at all.

In [FH03], CHR rules are transformed in a relational normal form to create smaller and streamlined programs. Said form is obtained by the introduction of special CHR constraints for each component of a rule, such as head, guard and body instead of the rule itself. After that, the transformation is executed by CHR programs till a fixpoint is reached, acting on the previously introduced normal CHR constraints form. Then, the final relational normal form obtained in the previous computation step is translated back into CHR rules. This process is called source-to-source program transformation. Performance improvement can be considered as a similarity between the paper under discussion and our thesis. In fact, unfolding can increase the performance of a program. Some differences between the two works can be however pointed out. First of all, the correctness of such proposed transformation methodology was not given. As in [Frü04], also in this paper the trivial non-termination problem is not considered at all. Finally, the main difference between our program transformation and the one introduced in the paper under discussion consists in the use of the relational normal form, which this thesis does not use.

A specific form of transformation for CHR is considered in [FDPW02]. In particular, the PCHR (Probabilistic CHR) is introduced here, that is a CHR where every rule is asso-

ciated with a nonnegative number. The higher said number, the higher is the probability that the considered rule fires, when alternative rules can also fire. This paper, like the thesis, considers the confluence concept but, differently from the thesis itself. It extends the considered concept to PCHR, introducing the probabilistic confluent notion. The source-to-source program transformation is used for the implementation of PCHR. In fact, the relational normal form, previously discussed for [FH03] was already introduced in this paper.

Various ways to improve complier performances are proposed in [SSD05b]. Said compiler optimizations are a major step towards allowing CHR programmers to write more readable and declarative programs, without sacrificing efficieny. In particular, we are interesting in two of the three improvement mechanism proposed, that are called guard optimization and types and modes. In fact, the other one, called occurrence subsumption, does not lead to a real rule rewriting. As a matter of fact, it only permits us to make considerations about particular rules, where the head and guards constraints are symmetric. Said considerations are enough to deduce that the considered rule can not be applied if only a partial rule application analysis fails. Guard optimization permits the simplification of (some) guard constraints, by considering that all recent CHR compiler/interpreter implementations use a specific operational semantics called refined operational semantics and represented by the ω_r symbol, where the rules are tried in textual order. This usually allows for the deletion of the constraints in the guard, if they are a direct consequence of the guards of the previous introduced rules, from the textual point of view. Types and modes mechanism consists in moving the charactrization of a variable such as a list, from the head to the guard of a rule, after which a type declaration is given by the user. Said transformation permits us to benefit from analysis like the never-stored one [HGdlBSD05]. This paper gives the statement of the equivalence between the ω_r semantics and the new semantics proposed. Unlike this paper, in our thesis the reference operational semantics is ω_t. In spite of this difference, a form of guard semplification is also introduced in our work. In fact, the guard constraints of the unfolded rule are a subset of conjunction or the guard constraints of the rule which would be unfolded and the rule which would be used to perform unfolding. As a matter of fact, the constraints of the rule that would be used

to do unfolding, that are entailed by those of the rule that would be unfolded, are deleted. Our thesis does not consider the never-stored analysis because such as analysis is more indicated for compiler optimization than for a source-to-source transformation.

The introduction of aggregates in CHR [SWS07] implies another form of source-to-source CHR program transfomation. In fact, the standard CHR is enriched by special CHR constraints like *sum*, *count*, *findall* and *min* which are managed by a preprocessor. Said preprocessor transforms a CHR program with aggregates into a standard CHR program without aggregates. Our thesis only considers standard CHR and not such an extension. Naturally, aggregate and unfolding techniques can work together to increase the readability and performance of a CHR program.

Both the general and the goal specific approaches are important in order to define practical transformation systems for CHR. In fact, on the one hand of course one needs some general unfold rule. On the other hand, given the difficulties in removing rules from the transformed program, some goal specific techniques can help to improve the efficiency of the transformed program for specific classes of goals. A method for deleting redundant CHR rules is considered in [AF04]. However, it is based on a semantic check and it is not clear whether it can be transformed into a specific syntactic program transformation rule.

When considering more generally the field of concurrent logic languages, we find a few papers which address the issue of program transformation. Notable examples include [EGM01] that deals with the transformation of concurrent constraint programming (CCP) and [UF88] that considers Guarded Horn Clauses (GHC). In [EGM01] several kinds of program transformation are defined, including which folding and unfolding. On the other hand, in [UF88] only the fold and unfold transformation are considered. The main difference between CCP and GHC consists in the underlying constraint solver. In fact, the sover of GHC is only able to manage equality as a built-in constraint, while the one used by CCP, similarly to CHR, can be a more complex built-in constraint solver. But the main problem in extending the results obtained in these papers to CHR consists in the possibility, for a CHR head to be composed of multiple constraints. In fact, when a CHR multiple head rule can be used to unfold another CHR rule, we can not guarantee that during the real computation exactly the same constraint will be used in the body. Another difference

between GHC and CCP with respect to CHR consists in the algorithm that evaluates if a rule can be applied to a (conjunction) of constraints. In fact, the first two above mentioned languages use unification algorithms unlike CHR which uses matching substitution. This causes difficulties especially when rule deletion is considered.

The third chapter of the current thesis can be considered as a first step in the direction of defining a transformation system for CHR programs, based on unfolding. This step could be improved in several directions. First of all, the unfolding operation could be extended to also take into consideration the constraints in the propagation part of the head of a rule instead of those in the body. In addition, the condition that we have provided for safely replacing a rule could be generalized to include more cases. Also, we could extend to CHR some of the other transformations, notably folding, which have been defined in [EGM01] for CCP. Finally, we would like to investigate from a practical perspective to what extent program transformation can improve the performances of the CHR solver. Clearly, the application of an unfolded rule avoids some computational steps, assuming of course that unfolding is done at the time of compilation, even though the increase in the number of rules could eliminate this improvement when the original rule cannot be removed. Here it would probably be important to consider some unfolding strategy, in order to decide which rules have to be unfolded.

References

[Abd97] Slim Abdennadher. Operational semantics and confluence of constraint propagation rules. In Gert Smolka, editor, *Third International Confonference on Principles and Practice of Constraint Programming (CP 1997)*, volume 1330 of *Lecture Notes in Computer Science*, pages 252–266. Springer-Verlag, October, November 1997.

[AF04] Slim Abdennadher and Thom Frühwirth. Integration and organization of rule-based constraint solvers. In Maurice Bruynooghe, editor, *Logic Based Program Synthesis and Transformation (LOPSTR 2003)*, volume 3018/2004 of *Lecture Notes in Computer Science*, pages 198–231, Berlin / Heidelberg, August 2004. Springer-Verlag.

[AFM99] Slim Abdennadher, Thom Frühwirth, and Holger Meuss. Confluence and Semantics of Constraint Simplification Rules. *Constraints*, 4(2):133–165, 1999. Special Issue on the Second International Conference on Principles and Practice of Constraint Programming.

[AK99] Hassan Aït-Kaci. *Warren's Abstract Machine - Tutorial Reconstruction*. MIT Press, Cambridge, U.S.A., 1999.

[AKSS02] Slim Abdennadher, Ekkerhard Krämer, Matthias Saft, and Matthias Schmauss. JACK: A Java Constraint Kit. *Electronic Notes in Theoretical Computer Science*, 64, 2002.

[BD77] Rod M. Burstall and John Darlington. A Transformation System for De-

veloping Recursive Programs. *Journal of the ACM (JACM)*, 24(1):44–67, 1977.

[BPS04] Marco Bramanti, Carlo Domenico Pagani, and Sandro Salsa. *Matematica. Calcolo infinitesimale e algebra lineare*. Zanichelli, Bologna, Italy, second edition, September 2004.

[Bro93] Stephen D. Brookes. Full abstraction for a shared variable parallel language. In *LICS '93: Proceedings of Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 98–109. IEEE Computer Society Press, June 1993.

[dBGM97] Frank S. de Boer, Maurizio Gabbrielli, and Maria C. Meo. Semantics and expressive power of a timed concurrent constraint language. In Gert Smolka, editor, *Proceedings of Third International Conference on Principles and Practice of Constraint Programming (CP 97)*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[dBP91] Frank S. de Boer and Catiuscia Palamidessi. A fully abstract model for concurrent constraint programming. In S. Abramsky and T.S.E. Maibaum, editors, *Proc. of TAPSOFT/CAAP*, volume 493 of *LNCS*, pages 296–319. Springer-Verlag, 1991.

[DGM05] Giorgio Delzanno, Maurizio Gabbrielli, and Maria C. Meo. A Compositional Semantics for CHR. In *PPDP '05: Proceedings of the 7th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 209–217, New York, NY, USA, July 2005. ACM.

[DP90] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[DSGdlBH04] Gregory J. Duck, Peter J. Stuckey, Maria J. García de la Banda, and

Christian Holzbaur. The Refined Operational Semantics of Constraint Handling Rules. In *ICLP*, pages 90–104, September 2004.

[Dum73] Michael Dummett. *Frege. Philosophy of language*. Duckworth, London, 1973.

[EGM01] Sandro Etalle, Maurizio Gabbrielli, and Maria C. Meo. Transformations of CCP programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(3):304–395, 2001.

[FA03] Thom Frühwirth and Slim Abdennadher. *Essentials of Constraint Programming*. Springer, April 2003.

[FDPW02] Thom Frühwirth, Alessandra Di Pierro, and Herbert Wiklicky. Probabilistic Constraint Handling Rules. In Marco Comini and Moreno Falaschi, editors, *11th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2002)*, volume 76 of *Electronic Notes in Theoretical Computer Science (ENTCS)*, 2002. Selected Papers.

[FH03] Thom Frühwirth and Christian Holzbaur. Source-to-Source Tranformation for a Class of Expressive Rules. In Francesco Buccafurri, editor, *APPIA-GULP-PRODE*, pages 386–397, September 2003.

[FPP04] Fabio Fioravanti, Alberto Pettorossi, and Maurizio Proietti. *Transformation Rules for Locally Stratified Constraint Logic Programs*, volume 3049/2004 of *Lecture Notes in Computer Science*, pages 291–339. Springer Berlin / Heidelberg, June, 23 2004.

[Fre84] Gottlob Frege. *Die Grundlagen der Arithmetik. Eine logisch-mathematische Untersuchung über den Begriff der Zahl*. Breslau, 1884.

[Fre23] Gottlob Frege. Logische Untersuchungen. Dritter Teil: Gedankengefüge. *Beiträge zur Philosophie des deutschen Idealismus*, III:36–51, 1923.

[Frü98] Thom Frühwirth. Theory and Practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1-3):95–138, October 1998. Special Issue on Constraint Logic Programming.

[Frü04] Thom Frühwirth. Specialization of Concurrent Guarded Multi-set Transformation Rules. In Sandro Etalle, editor, *Logic Based Program Synthesis and Transformation*, Lecture Notes in Computer Science, pages 133 – 148, Verona, August, 26 – 28 2004. 14th International Symposium, LOPSTR.

[Frü06] Thom Frühwirth. Constraint Handling Rules: The Story So Far. In *PPDP '06: Proceedings of the 8th ACM SIGPLAN symposium on Principles and practice of declarative programming*, pages 13–14, New York, NY, USA, 2006. ACM. Invited Tutorial.

[GdlBDMS02] Maria Garcia de la Banda, Bart. Demoen, Kim Mariott, and Peter J. Stuckey. To the Gates of HAL: A HAL Tutorial. In *Proceedings of the International Symposium on Functional and Logic Programming – FLOPS 2002*, pages 47 – 66, Japan, 2002. LNCS.

[GM06] Maurizio Gabbrielli and Simone Martini. *Linguaggi di programmazione: principi e paradigmi*. Collana di istruzione scientifica: serie di informatica. McGraw-Hill, Via Ripamonti, 89 - 20139 Milano, 2006.

[GMT06] Maurizio Gabbrielli, Maria C. Meo, and Paolo Tacchella. A compositional semantics for CHR with propagation rules. In *Proceedings of the Third Workshop on Constraint Handling Rules*, number 425 in Report CW, pages 99–107. Katholieke Universiteit Leuven, June 2006.

[Han06] Michael Hanus. Adding Constraint Handling Rules to Curry. In *Proceeding of the 20th Workshop on Logic Programming (WLP 2006)*, pages 81–90. INFSYS Research Report 1843-06-02 (TU Wien), 2006.

[HF00] Christian. Holzbaur and Thom Früwirth. A prolog constraint handling rules compiler and runtime system. *Journal of Applied Artificial Intelligence*, 14(4):369–388, April 2000.

[HGdlBSD05] Christian Holzbaur, Maria Garcia de la Banda, Peter J. Stuckey, and Gregory J. Duck. Optimizing Compilation of Contraint Handling Rules in HAL. *Theory and Practice of Logic Programming (TPLP)*, 5(1 & 2):503–531, March 2005.

[HP90] John L. Hennessy and David A. Patterson. *Computer architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

[Jan97] Theo M.V. Janssen. Compositionality. In Johan F. A. K. van Benthem and Alice G. B ter Meulen, editors, *Handbook of Logic and Language*, chapter 7, pages 417 –475. MIT Press, Cambridge, U.S.A., 1997.

[JM94] Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: a Survey. *Journal of Logic Programming*, 19/20:503–582, May/July 1994. Special 10th Anniversary Issue.

[Jon85] Bengt Jonsson. A model and a proof system for asynchronous networks. In *Proceedings of the 4th ACM Symposium on Principles of Distributed Computing*, pages 49–58. ACM Press, 1985.

[Mah87] Michael J. Maher. Logic Semantics for a Class of Committed-Choice Programs. In *Proceedings of the 4th International Conference on Logic Programming*, pages 858–876, New York, NY, USA, 1987. ACM Press.

[Pop76] Karl Popper. *Unended Quest. An intellectual autobiography*. Fontana, 1976.

[PP96] Alberto Pettorossi and Maurizio Proietti. Rules and Strategies for Transforming Functional and Logic Programs. *ACM Computing Surveys*, 28(2):360–414, 1996.

[Sar93] Vijay Saraswat. *Concurrent Constraint Programming*. MIT Press, Cambridge, U.S.A., 1993.

[Sch05] Tom Schrijvers. *Analyses, Optimizations and Extensions of Constraint Handling Rules*. PhD thesis, Katholieke Universiteit Leuven, Celestijneanlaan 200 A - B - 3001 Leuven, June 2005.

[SL07] Martin Sulzmann and Edmund S. L. Lam. Compiling Constraint Handling Rules with Lazy and Concurrent Search Techniques. In Khalil Djelloul, Gregory J. Duck, and Martin Sulzmann, editors, *Proceeding of Fourth Workshop on Constraint Handling Rules*, pages 139–149, September 2007.

[SSD05a] Jon Sneyers, Tom Schrijvers, and Bart Demoen. The computational power and complexity of constraint handling rules. In Tom Schrijvers and Thom Frühwirth, editors, *Proceedings of CHR 2005, Second Workshop on Constraint Handling Rules*, pages 3–17, Celestijnenlaan 200A – B-3001 Heverlee (Belgium), August 2005. Department of Computer Science, K.U. Leuven. Report CW421.

[SSD05b] Jon Sneyers, Tom Schrijvers, and Bart Demoen. Guard and continuation optimization for occurrence representations of CHR. In Maurizio Gabbrielli and Gopal Gupta, editors, *21st International Conference, ICLP 2005*, volume 3668 of *Lecture Notes in Computer Science*, pages 83–97, 2005.

[SWS07] Jon Sneyers, Peter V. Weert, and Tom Schrijvers. Aggregates for Constraint Handling Rules. In Khalil Djelloul, Gregory J. Duck, and Martin Sulzmann, editors, *Proceeding of Fourth Workshop on Constraint Handling Rules*, pages 91–105, September 2007.

[TMG07] Paolo Tacchella, Maria C. Meo, and Maurizio Gabbrielli. Unfolding in CHR. In *PPDP '07: Proceedings of the 9th ACM SIGPLAN interna-*

tional symposium on Principles and Practice of Declarative Programming, pages 179–186, New York, NY, USA, 2007. ACM.

[TS84] Hisao Tamaki and Taisuke Sato. Unfold/Fold transformations of logic programs. In *Proceedings of International Conference on Logic Programming*, pages 127–138, 1984.

[UF88] Kazunori Ueda and Koichi Furukawa. Transformation rules for GHC programs. In *Proceedings of International Conference on Fifth Generation Computer Systems 1988 (FGCS'88)*, pages 582–591. ICOT Press, 1988.

[VWSD05] Peter Van Weert, Tom Schrijvers, and Bart Demoen. K.U.Leuven JCHR: a user-friendly, flexible and efficient CHR system for Java. In Tom Schrijvers and Thom Frühwirth, editors, *Proceedings of Second Workshop on Constraint Handling Rules*, pages 47–62, September 2005.

[WSD07] Pieter Wuille, Tom Schrijvers, and Bart Demoen. CCHR: the fastest CHR Implementation, in C. In Khalil Djelloul, Gregory J. Duck, and Martin Sulzmann, editors, *Proceeding of Fourth Workshop on Constraint Handling Rules*, pages 123–137, September 2007.