

A compositional Semantics for CHR

Maurizio Gabbrielli

Università di Bologna

and

Maria Chiara Meo

Università “G. D’Annunzio” di Chieti-Pescara

Constraint Handling Rules (CHR) are a committed-choice declarative language which has been designed for writing constraint solvers. A CHR program consists of multi-headed guarded rules which allow one to rewrite constraints into simpler ones until a solved form is reached.

CHR has received a considerable attention, both from the practical and from the theoretical side. Nevertheless, due the use of multi-headed clauses, there are several aspects of the CHR semantics which have not been clarified yet. In particular, no compositional semantics for CHR has been defined so far.

In this paper we introduce a fix-point semantics which characterizes the input/output behavior of a CHR program and which is and-compositional, that is, which allows to retrieve the semantics of a conjunctive query from the semantics of its components. Such a semantics can be used as a basis to define incremental and modular analysis and verification tools.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*Semantics*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Constraints*

General Terms: Languages, Theory, Semantics

1. INTRODUCTION

Constraint Handling Rules (CHR) [Frühwirth 1991; 1998] are a committed-choice declarative language which has been specifically designed for writing constraint solvers. The first constraint logic languages used mainly built-in constraint solvers designed by following a “black box” approach. This made it hard to modify, debug, and analyze a specific solver. Moreover, it was very difficult to adapt an existing solver to the needs of some specific application, and this was soon recognized as a serious limitation since often practical applications involve application specific constraints.

By using CHR one can easily introduce specific user-defined constraints and the

Author’s address:

Maurizio Gabbrielli, Dipartimento di Scienze dell’Informazione, Mura A. Zamboni 7, 40127 Bologna, Italy. gabbri@cs.unibo.it.

Maria Chiara Meo, Dipartimento di Scienze, Viale Pindaro 42, 65127 Pescara, Italy. cmeo@unich.it.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 2007 ACM 1529-3785/07/1100-0001 \$5.00

related solver into a host language. In fact, a CHR program consists of (a set of) multi-headed guarded simplification and propagation rules which are specifically designed to implement the two most important operations involved in the constraint solving process: Simplification rules allow to replace constraints by simpler ones, while preserving their meaning. Propagation rules are used to add new redundant constraints which do not modify the meaning of the given constraint and which can be useful for further reductions. It is worth noting that the presence of multiple heads in CHR is an essential feature which is needed in order to define reasonably expressive constraint solvers (see the discussion in [Frühwirth 1998]). However, such a feature, which differentiates this proposal from many existing committed choice logic languages, complicates considerably the semantics of CHR, in particular it makes very difficult to obtain a compositional semantics, as we argue below.

This is unfortunate, as compositionality is an highly desirable property for a semantics. To be more precise and to set the ground for the content of this paper, given a language L assume that its syntax allows to use some operator, here generically denoted by op , to build programs (or processes, according to the concurrency terminology). Assume also that a semantics \mathcal{S} is given which associates to each “piece of program” P in L some suitable semantic object $\mathcal{S}(P)$. As well known, this can be done in a variety of ways, by using algebraic, logic, operational or denotational techniques. We say that \mathcal{S} is compositional w.r.t. the syntactic operator op of the language L if we can reconstruct $\mathcal{S}(P op Q)$ from $\mathcal{S}(P)$ and $\mathcal{S}(Q)$ by using a suitable semantic counterpart of op . In other words, \mathcal{S} is compositional w.r.t. op if $\mathcal{S}(P op Q) = \mathcal{S}(P) \tilde{op} \mathcal{S}(Q)$ holds for a suitable \tilde{op} . From the above definition it follows immediately that a compositional semantics induces an equivalence on programs which is a congruence: that is, if $\mathcal{S}(P) = \mathcal{S}(Q)$ then $\mathcal{S}(P op R) = \mathcal{S}(Q op R)$ holds, for any other piece of program R . From a theoretical perspective reasoning with congruences is much easier than using simple equivalences. Indeed, basically all the important equivalences used in process calculi are congruences. From a more practical point of view, compositionality (and congruences) provides the basis to define incremental and modular tools for software analysis and verification. These features are essential in order to deal with partially defined components and to reduce the complexity of verification of large systems by considering separately smaller components.

In this paper we introduce a fix-point semantics for CHR which characterizes the input/output behavior of a program and which is and-compositional, that is, which allows to retrieve the semantics of a conjunctive query from the semantics of its components.

In general, due to the presence of synchronization mechanisms, the input/output semantics is not compositional for committed choice logic languages and for most concurrent languages in general. Indeed, the need for more complicate semantic structures based on traces was recognized very early as a necessary condition to obtain a compositional model, first for dataflow languages [Jonsson 1985] and then in the case of many other paradigms, including imperative concurrent languages [Brookes 1993] and concurrent constraint and logic languages [de Boer and Palamidessi 1991].

When considering CHR this basic problem is further complicated: due to the

presence of multiple heads, the traces consisting of sequences of input/output pairs, analogous to those used in the above mentioned works, are not sufficient to obtain a compositional semantics.

Our solution to obtain a compositional model is to use an augmented semantics based on traces which includes at each step two “assumptions” on the external environment and two “outputs” of the current process.

Thus our model is based on sequences of quadruples, rather than of simple input/output pairs.

Our compositional semantics is obtained by a fixpoint construction which uses an enhanced transitions system implementing the rules for assumptions described above. We prove the correctness of the semantics w.r.t. a notion of observables which characterizes the input/output behavior of terminating computations where the original goal has been completely reduced to built-in constraints. We will discuss later the extensions needed in order to characterize different notions of results, such as the “qualified answers” used in [Frühwirth 1998].

The remaining of this paper is organized as follows. The next section introduces some preliminaries about CHR and its operational semantics. Section 3 contains the definition of the compositional semantics, while section 4 presents the compositionality and correctness results. Section 5 discusses related work while section 6 concludes by indicating directions for future work.

A preliminary, short version of this paper appeared in [Delzanno et al. 2005].

2. PRELIMINARIES

In this section we first introduce some preliminary notions and then define the CHR syntax and operational semantics. Even though we try to provide a self-contained exposition, some familiarity with constraint logic languages and first order logic could be useful.

2.1 CHR syntax

We first need to distinguish the constraints handled by an existing solver, called built-in (or predefined) constraints, from those defined by the CHR program, user-defined (or CHR) constraints. An atomic constraint is a first-order predicate (atomic formula). By assuming to use two disjoint sorts of predicate symbols we then distinguish built-in atomic constraints from CHR atomic constraints. A built-in constraint c is defined by

$$c ::= a \mid c \wedge c \mid \exists_x c$$

where a is an atomic built-in constraint¹. For built-in constraints we assume given a theory CT which describes their meaning.

On the other hand, according to the usual CHR syntax, we assume that a user-defined constraint is a conjunction of atomic user-defined constraints. We use c, d to denote built-in constraints, h, k to denote CHR constraints and a, b, f, g to denote both built-in and user-defined constraints (we will call these generically constraints).

¹We could consider more generally first order formulas as built-in constraints, as far as the results presented here are concerned.

We also denote by **false** any inconsistent (conjunction of) constraint(s) and with **true** any empty built-in constraint multiset. The capital versions of these notations will be used to denote multisets of constraints. Furthermore we denote by \mathcal{U} the set of user-defined constraints.

We will often use “,” rather than \wedge to denote conjunction and we will often consider a conjunction of atomic constraints as a multiset of atomic constraints. In particular, we will use this notation based on multisets in the syntax of CHR. 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 V which remain unquantified. $Fv(\phi)$ denotes the free variables appearing in ϕ and we denote by \cdot the concatenation of sequences and by ε the empty sequence. Given a set A , $\wp(A)$ denotes the set consisting of all subsets of A , while $\wp_m(A)$ denotes the set consisting of all the multisets over A . Moreover, if $\bar{t} = t_1, \dots, t_m$ and $\bar{t}' = t'_1, \dots, t'_m$ are sequences of terms then the notation $p(\bar{t}) = p'(\bar{t}')$ represents the set of equalities $t_1 = t'_1, \dots, t_m = t'_m$ if $p = p'$, and it is undefined otherwise. Analogously, if $H = h_1, \dots, h_k$ and $H' = h'_1, \dots, h'_k$ are sequences of constraints, the notation $H = H'$ represents the set of equalities $h_1 = h'_1, \dots, h_k = h'_k$. Finally, \uplus denotes the multiset union, while we consider \setminus as an overloaded operator used both for set and multiset difference (the meaning depends on the type of the arguments).

We are now ready to introduce the CHR syntax as defined in [Frühwirth 1998].

Definition 2.1. [Syntax] A CHR *simplification* rule has the form

$$r@H \Leftrightarrow C \mid B$$

while a CHR propagation rule has the form

$$r@H \Rightarrow C \mid B,$$

where r is a unique identifier of a rule, H (the head) is a (non-empty) multiset of user-defined constraints, C (the guard) is a multiset of built-in constraints and B is a possibly empty multiset of (built-in and user-defined) constraints². A CHR program is a finite set of CHR simplification and propagation rules.

In the following if the guard $C = \mathbf{true}$, then $\mathbf{true} \mid$ is omitted. We prefer to use multisets rather than sequences (as in the original CHR papers) since multisets appear to correspond more precisely to the nature of CHR rules. Moreover in this paper we will not use the identifiers of the rules, which will then be omitted.

A CHR goal is a multiset of (both user-defined and built-in) constraints. *Goals* is the set of all goals. An example can be useful to see what kind of programs we are considering here.

EXAMPLE 2.2. *The following CHR program, given a forest of finite trees, is able*

²Some papers consider also simplagation rules. Since these are abbreviations for propagation and simplification rules we do not need to introduce them.

Solve	$\frac{CT \models c \wedge d \leftrightarrow d' \text{ and } c \text{ is a built-in constraint}}{\langle (c, G), K, d \rangle \longrightarrow_s \langle G, K, d' \rangle}$
Introduce	$\frac{h \text{ is a user-defined constraint}}{\langle (h, G), K, d \rangle \longrightarrow_s \langle G, (h, K), d \rangle}$
Simplify	$\frac{H \Leftrightarrow C \mid B \in P \quad x = Fv(H) \quad CT \models d \rightarrow \exists_x((H = H') \wedge C)}{\langle G, H' \wedge K, d \rangle \longrightarrow_s \langle B \wedge G, K, H = H' \wedge d \rangle}$
Propagate	$\frac{H \Rightarrow C \mid B \in P \quad x = Fv(H) \quad CT \models d \rightarrow \exists_x((H = H') \wedge C)}{\langle G, H' \wedge K, d \rangle \longrightarrow_s \langle B \wedge G, H' \wedge K, H = H' \wedge d \rangle}$

Table I. The standard transition system for CHR

to recognize if two nodes belong to the same tree:

$$\begin{aligned}
\text{same}(X, Y) &\Leftrightarrow \text{find}(X, Z), \text{find}(Y, Z) \\
\text{find}(X, Y), \text{edge}(Z, V) &\Leftrightarrow X = V, \text{find}(Z, Y), \text{edge}(Z, V) \\
\text{find}(X, Y), \text{root}(X) &\Leftrightarrow X = Y, \text{root}(X) \\
\text{edge}(X, Y) &\Leftrightarrow \\
\text{root}(X) &\Leftrightarrow
\end{aligned}$$

2.2 Operational semantics

We describe now the operational semantics of CHR as provided by [Frühwirth 1998] by using a transition system $T_s = (Conf_s, \longrightarrow_s)$ (s here stands for “standard”, as opposed to the semantics we will use later). Configurations in $Conf_s$ are triples of the form $\langle G, K, d \rangle$ where G are the constraints that remain to be solved, K are the user-defined constraints that have been accumulated and d are the built-in constraints that have been simplified³.

An *initial configuration* has the form

$$\langle G, \emptyset, \emptyset \rangle$$

and consists of a goal G , an empty user-defined constraint and an empty built-in constraint.

A *final configuration* has either the form

$$\langle G, K, \mathbf{false} \rangle,$$

when it is *failed*, i.e. when it contains an inconsistent built-in constraint store represented by the unsatisfiable constraint \mathbf{false} , or has the form

$$\langle \emptyset, K, d \rangle$$

³In [Frühwirth 1998] triples of the form $\langle G, K, d \rangle_{\mathcal{V}}$ were used, where the annotation \mathcal{V} , which is not changed by the transition rules, is used to distinguish the variables appearing in the initial goal from the variables which are introduced by the rules. We can avoid such an indexing by explicitly referring to the original goal.

when it is successfully terminated because there are no applicable rules.

Given a program P , the transition relation $\longrightarrow_s \subseteq \text{Conf} \times \text{Conf}$ is the least relation satisfying the rules in Table I (for the sake of simplicity, we omit indexing the relation with the name of the program). The **Solve** transition allows to update the constraint store by taking into account a built-in constraint contained in the goal. Without loss of generality, we will assume that $Fv(d') \subseteq Fv(c) \cup Fv(d)$. The **Introduce** transition is used to move a user-defined constraint from the goal to the CHR constraint store, where it can be handled by applying CHR rules. The transitions **Simplify** and **Propagate** allow to rewrite user-defined constraints (which are in the CHR constraint store) by using rules from the program. As usual, in order to avoid variable names clashes, both these transitions assume that clauses from the program are renamed apart, that is assume that all variables appearing in a program clause are fresh ones. Both the **Simplify** and **Propagate** transitions are applicable when the current store (d) is strong enough to entail the guard of the rule (C), once the parameter passing has been performed (this is expressed by the equation $H = H'$). Note that, due to the existential quantification over the variables x appearing in H , in such a parameter passing the information flow is from the actual parameters (in H') to the formal parameters (in H), that is, it is required that the constraints H' which have to be rewritten are an instance of the head H . When applied, both these transitions add the body B of the rule to the current goal and the set of equations $H = H'$, expressing the parameter passing mechanism, to the built-in constraint store. The difference between **Simplify** and **Propagate** is in the fact that while the former transition removes the constraints H' which have been rewritten from the CHR constraint store, this is not the case for the latter.

Given a goal G , the operational semantics that we consider observes the final stores of computations terminating with an empty goal and an empty user-defined constraint. We call these observables data sufficient answers by following the terminology of [Frühwirth 1998].

Definition 2.3. [Data sufficient answers] Let P be a program and let G be a goal. The set $\mathcal{SA}_P(G)$ of data sufficient answers for the query G in the program P is defined as follows

$$\begin{aligned} \mathcal{SA}_P(G) = & \{ \exists_{-Fv(G)} d \mid \langle G, \emptyset, \emptyset \rangle \longrightarrow_s^* \langle \emptyset, \emptyset, d \rangle \not\rightarrow_s \} \\ & \cup \\ & \{ \mathbf{false} \mid \langle G, \emptyset, \emptyset \rangle \longrightarrow_s^* \langle G', K, \mathbf{false} \rangle \}. \end{aligned}$$

Thus data sufficient answers consider the results of terminated computations where all the user-defined constraints have been rewritten into built-in constraints.

In [Frühwirth 1998] it is also considered the following different notion of answer, obtained by computations terminating with a user-defined constraint which does not need to be empty.

Definition 2.4. [Qualified answers] Let P be a program and let G be a goal. The set $\mathcal{QA}_P(G)$ of qualified answers for the query G in the program P is defined as

follows

$$\mathcal{QA}_P(G) = \left\{ \exists_{-Fv(G)}(K \wedge d) \mid \langle G, \emptyset, \emptyset \rangle \longrightarrow_s^* \langle \emptyset, K, d \rangle \not\rightarrow_s \right\} \\ \cup \\ \{ \mathbf{false} \mid \langle G, \emptyset, \emptyset \rangle \longrightarrow_s^* \langle G', K, \mathbf{false} \rangle \}.$$

In this paper we consider data sufficient answers and we discuss in subsection 4.4 a possible extension in order to characterize also qualified answers. Note that both previous notions of observables characterize an input/output behavior, since the input constraint is implicitly considered in the goal.

In the remaining of this paper we will consider only simplification rules since propagation rules can be mimicked by simplification rules, as far as the results contained in this paper are concerned.

Note that in presence of propagation rules the abstract operational semantics that we consider in this paper introduces redundant infinite computations: Since propagation rules do not remove user defined constraints (see rule Propagate in Table I), when a propagate rule is applied it introduces an infinite computation (obtained by subsequent applications of the same rule). Note however that this does not imply that in presence of an active propagation rule the semantics that we consider is empty. In fact, the application of a simplification rule after a propagation rule can cause the termination of the computation, by removing the atoms which are needed by the head of the propagation rule. It is also possible to define a different operational semantics (see [Abdennadher 1997] and [Duck et al. 2004]) which avoids these infinite computations by allowing to apply at most once a propagation rule to the same constraints. We will discuss further this issue in Section 5.

2.3 And-compositionality in CHR

As mentioned in the introduction, compositionality is an essential feature of a semantics since it allows to reconstruct the meaning of a compound construct from the meaning of its components. The basic construct which allows to structure CHR programs and goals is the and-composition of logic (atomic) formulae. It is therefore rather natural to consider in the context of CHR and-compositionality, namely the possibility of retrieving the semantics of a conjunctive query from the semantics of its components. This form of compositionality can be defined more formally as follows.

Definition 2.5. [And-compositional semantics] Let P be a program and let A and B be (atomic) formulae. A semantics \mathcal{S}_P is and-compositional if $\mathcal{S}_P(A, B) = \mathcal{C}(\mathcal{S}_P(A), \mathcal{S}_P(B))$ ⁴ for a suitable composition operator \mathcal{C} which does not depend on the program P .

Due to the presence of guards and multiple heads in CHR, the semantics which associates to a program P the denotation \mathcal{SA}_P is not and-compositional, since goals which have the same input/output behavior can behave differently when composed with other goals. The following example clarifies this point.

⁴Note that here, according to the syntax previously defined, A, B represents the logical conjunction $A \wedge B$.

EXAMPLE 2.6. Let P be the program consisting of the single rule

$$g, h \Leftrightarrow c$$

(where c is a built-in constraint). According to Definition 2.4 we have that $\mathcal{SA}_P(g) = \mathcal{SA}_P(k) = \emptyset$, while

$$\mathcal{SA}_P(g, h) = \{(\exists_{-Fv(g,h)}c)\} \neq \emptyset = \mathcal{SA}_P(k, h),$$

where k is a CHR constraint.

An analogous example can be made to show that also the semantics \mathcal{QA} is not and-compositional. Thus, in order to obtain a compositional characterization of the observables \mathcal{SA}_P some semantic structures richer than pair need to be used. As we will see in the next section, we will develop a compositional model based on traces.

3. A COMPOSITIONAL TRACE SEMANTICS

As shown by the previous example 2.6 the semantics \mathcal{SA} is not compositional. It is worth noting that the problem exemplified above is different from the classic problem of concurrent languages (see for example [Brookes 1993; de Boer and Palamidessi 1991]) where the interaction of non-determinism and synchronization makes the input/output observables non-compositional.

Intuitively the problem can be stated as follows. The CHR rule $r@ g, h \Leftrightarrow c$ cannot be used to rewrite a goal g , no matter how the variables are constrained (that is, for any input constraint), because the goal consists of a single atom g while the head of the rule contains two atoms g, h . Therefore, if we considered a semantics based on input/output traces, we would obtain the empty denotation for the goal g in the program consisting of the rule r . Clearly also the goal k (which is not defined) has an empty denotation. However, since the rule r can be used to rewrite the goal g, h the semantics of g, h is not empty, differently from the case of the semantics of k, h . This means that the semantics of g, h cannot be derived from the semantics of h and g , that is, the semantics is not compositional. It is worth noting that even restricting to a more simple notion of observable, such as the results of terminating computations, does not simplify this problem. In fact, differently from the case of ccp (concurrent constraint programming) languages, also the semantics based on these observables (usually called resting points) is not compositional for CHR.

In order to solve this problem we have then to use some additional information which allows us to describe the behavior of goals in any possible and-composition without, of course, considering explicitly all the possible and-compositions. The basic idea of our approach is then to collect in the semantics also 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 . In other words, h is an assumption which is made in the semantics describing the computation of g . When composing (by using a suitable notion of composition) such a semantics with that one of a goal which contains h we can verify that the “assumption” h is satisfied

and therefore obtain 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.

3.1 A new transition system

The idea sketched above is developed by defining a new transition system which implements this mechanism based on assumptions for dealing with the missing parts of heads. The new transition system allows one to generate the sequences appearing in the compositional model by using a standard fix-point construction. As a first step in our construction we modify the notion of configuration used before: Since we do not need to distinguish user-defined constraints which appear in the goal from the user-defined constraints which have been already considered for reduction, we merge the first and the second components of previous triples (similarly to what has been done in [Abdennadher and Frühwirth 2003]). Thus we do not need the **Introduce** rule anymore. On the other hand, we need the information on the new assumptions, which is added as a superscript label of the transitions. Moreover, similarly to the case of the models for ccp, we also maintain at each step an assumption on the constraints appearing in the guards of the rules, in order to ensure that these are satisfied and the computation can proceed. Finally, we memorize at each step a second output element, consisting of those atoms which are not rewritten in the current derivation and which could be used to satisfy some assumptions (of the second type) when composing sequences representing different computations.

Thus we define a transition system $T = (Conf, \longrightarrow_P)$ where configurations in $Conf$ are pairs: the first component is a multiset of indexed atoms (the goal) and the second one is a built-in constraint (the store). Indexes are associated to atoms in order to denote the point in the derivation where they have been introduced. Atoms in the original goal are indexed by 0, while atoms introduced at the i -th derivation step are indexed by i . Given a program P , the transition relation $\longrightarrow_P \subseteq Conf \times Conf \times \wp_m(\mathcal{U})$ is the least relation satisfying the rules in Table II. Note that we consider only **Solve** and **Simplify** rules, as the other rules as previously mentioned are redundant in this context. **Solve'** is the same rule as before, while the **Simplify'** rule is modified to consider assumptions: When reducing a goal G by using a rule having head H , the multiset of assumptions $K = H \setminus G$ (with $H \neq K$) is used to label the transition (\setminus denotes multiset difference). Indexes allow us to distinguish different occurrences of the same atom which have been introduced in different derivation steps. We will use the notation G^i to indicate that all the atoms in G are indexed by i .

When indexes are not needed we will simply omit them. As before, we assume that program rules to be used in the new **Simplify'** rule use fresh variables to avoid names clashes.

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

EXAMPLE 3.1. *Given the goal $\{same(X, c), root(c), edge(c, b), edge(c, d), edge(d, e)\}$ and the program of Example 2.2 by using the transition system of Table II we obtain*

<p>Solve' $\frac{CT \models c \wedge d \leftrightarrow d'}{\langle c \wedge G, d \rangle \xrightarrow{\emptyset} \langle G, d' \rangle}$</p> <p>Simplify' $\frac{H \Leftrightarrow C \mid B \in P \quad x = Fv(H) \quad G \neq \emptyset \quad CT \models d \rightarrow \exists_x((H = (G, K)) \wedge C)}{\langle G \wedge A, d \rangle \xrightarrow{K} \langle B^{i+1} \wedge A, d \wedge (H = (G, K)) \rangle}$</p> <p style="text-align: center;">where i is the maximal index occurring in the goal $G \wedge A$</p>
--

Table II. The transition system for the compositional semantics

the following derivation

$$\begin{aligned}
& \langle \{\mathbf{same}(\mathbf{X}, \mathbf{c}), \mathbf{root}(c), \mathbf{edge}(d, f), \mathbf{edge}(c, d), \mathbf{edge}(d, e)\}, \mathbf{true} \rangle && \xrightarrow{\emptyset} \\
& \langle \{\mathbf{find}(X1, Z1), \mathbf{find}(\mathbf{Y1}, \mathbf{Z1}), \mathbf{root}(c), \mathbf{edge}(d, f), \mathbf{edge}(c, d), \mathbf{edge}(d, e)\}, \{X = X1, c = Y1\} \rangle && \xrightarrow{\emptyset} \\
& \langle \{\mathbf{X2} = \mathbf{Y2}, \mathbf{root}(X2), \mathbf{find}(X1, Z1), \mathbf{edge}(d, f), \mathbf{edge}(c, d), \mathbf{edge}(d, e)\}, && \\
& \quad \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\} \rangle && \xrightarrow{\emptyset} \\
& \langle \{\mathbf{root}(X2), \mathbf{find}(\mathbf{X1}, \mathbf{Z1}), \mathbf{edge}(d, f), \mathbf{edge}(c, d), \mathbf{edge}(d, e)\}, && \\
& \quad \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\} \rangle && \xrightarrow{\emptyset} \\
& \langle \{\mathbf{X3} = \mathbf{V3}, \mathbf{find}(Z3, Y3), \mathbf{edge}(Z3, V3), \mathbf{root}(X2), \mathbf{edge}(d, f), \mathbf{edge}(d, e)\}, && \\
& \quad \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c, X1 = X3, Z1 = Y3, c = Z3, d = V3\} \rangle && \xrightarrow{\emptyset} \\
& \langle \{\mathbf{find}(\mathbf{Z3}, \mathbf{Y3}), \mathbf{edge}(Z3, V3), \mathbf{root}(\mathbf{X2}), \mathbf{edge}(d, f), \mathbf{edge}(d, e)\}, && \\
& \quad \{X = d, X1 = d, X2 = c, X3 = d, Y1 = c, Y2 = c, Y3 = c, Z1 = c, Z3 = c, V3 = d\} \rangle && \xrightarrow{\emptyset} \\
& \langle \{\mathbf{X4} = \mathbf{Y4}, \mathbf{root}(X4), \mathbf{edge}(Z3, V3), \mathbf{edge}(d, f), \mathbf{edge}(d, e)\}, && \\
& \quad \{X = d, X1 = d, X2 = c, X3 = d, Y1 = c, Y2 = c, Y3 = c, Z1 = c, Z3 = c, V3 = d, && \\
& \quad Z3 = X4, Y3 = Y4, X2 = X4\} \rangle && \xrightarrow{\emptyset} \\
& \langle \{\mathbf{root}(X4), \mathbf{edge}(Z3, V3), \mathbf{edge}(d, f), \mathbf{edge}(d, e)\}, && \\
& \quad \{X = d, X1 = d, X2 = c, X3 = d, X4 = c, Y1 = c, Y2 = c, Y3 = c, Y4 = c, && \\
& \quad Z1 = c, Z3 = c, V3 = d\} \rangle. &&
\end{aligned}$$

3.2 The semantic domain

The semantics domain of our compositional semantics is based on sequences which represent derivations obtained by the transition system in Table II. More precisely, we first consider concrete sequences consisting of tuples of the form $\langle G, c, K, G', d \rangle$: Such a tuple represents a derivation step $\langle G, c \rangle \xrightarrow{K} \langle G', d \rangle$. The sequences we consider are terminated by tuples of the form $\langle G, c, \emptyset, G', c \rangle$ (with either $c = \mathbf{false}$ or $G \in \wp_m(\mathcal{U})$), which represent a terminating step (see the precise definition below). Since a sequence represents a derivation, we assume that the “output” goal G' at step i is equal to the “input” goal G at step $i + 1$, that is, we assume that if

$$\dots \langle G_i, c_i, K_i, G'_i, d_i \rangle \langle G_{i+1}, c_{i+1}, K_{i+1}, G'_{i+1}, d_{i+1} \rangle \dots$$

appears in a sequence, then $G'_i = G_{i+1}$ holds.

On the other hand, the input store c_{i+1} can be different from the output store d_i produced at the 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. However, we assume that if

$$\dots \langle G_i, c_i, K_i, G'_i, d_i \rangle \langle G_{i+1}, c_{i+1}, K_{i+1}, G'_{i+1}, d_{i+1} \rangle \dots$$

appears in a sequence then $CT \models c_{i+1} \rightarrow d_i$ holds: This means that the assumption made on the external environment cannot be weaker than the constraint store produced at the previous step. This reflects the monotonic nature of computations,

where information can be added to the constraint store and cannot be deleted from it. 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.

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

Definition 3.2. [Concrete sequences] The set Seq containing all the possible (concrete) sequences is defined as the set

$$Seq = \{ \langle G_1, c_1, K_1, G_2, d_1 \rangle \langle G_2, c_2, K_2, G_3, d_2 \rangle \cdots \langle G_n, c_n, \emptyset, G_n, c_n \rangle \mid \\ \text{for each } j, 1 \leq j \leq n \text{ and for each } i, 1 \leq i \leq n-1, \\ G_j \text{ are CHR goals, } K_i \text{ are multisets of CHR constraints,} \\ c_j, d_i \text{ are built-in constraints such that} \\ CT \models d_i \rightarrow c_i, CT \models c_{i+1} \rightarrow d_i \text{ and} \\ \text{either } c_n = \mathbf{false} \text{ or } G_n \in \wp_m(\mathcal{U}) \}.$$

From these concrete sequences we extract *abstract sequences* which are the objects of our semantic domain: From each tuple $\langle G, c, K, G', d \rangle$ in a sequence $\delta \in Seq$ we extract a tuple of the form $\langle c, K, H, d \rangle$ where we consider as before the input and output store (c and d , respectively) and the assumptions (K), while we do not consider anymore the output goal G' . Furthermore, we restrict the input goal G to that part H consisting of all user-defined constraints which will not be rewritten in the (derivation represented by the) sequence δ . Intuitively H contains those atoms which are available for satisfying assumptions of other goals, when composing two different sequences (representing two derivations of different goals). We also assume that if

$$\langle c_i, K_i, H_i, d_i \rangle \langle c_{i+1}, K_{i+1}, H_{i+1}, d_{i+1} \rangle$$

is in a sequence then $H_i \subseteq H_{i+1}$ holds, since these atoms which will not be rewritten in the derivation can only augment. Finally, indexes are not used in the abstract sequences (they are only needed to define stable atoms, see Definition 3.4).

We then define formally the semantic domain as follows.

Definition 3.3. [Abstract sequences] The semantic domain \mathcal{D} containing all the possible (abstract) sequences is defined as the set

$$\mathcal{D} = \{ \langle c_1, K_1, H_1, d_1 \rangle \cdots \langle c_n, \emptyset, H_n, c_n \rangle \mid \\ \text{for each } j, 1 \leq j \leq n \text{ and for each } i, 1 \leq i \leq n-1, \\ H_j \text{ and } K_i \text{ are multisets of CHR (non indexed) constraints,} \\ c_j, d_i \text{ are built-in constraints and } CT \models d_i \rightarrow c_i, \\ H_i \subseteq H_{i+1} \text{ and } CT \models c_{i+1} \rightarrow d_i \text{ holds } \}.$$

3.3 The compositional semantics

In order to define our semantics we need three more notions. First, we define an abstraction operator α which extracts from the concrete sequences in Seq (representing exactly derivation steps) the abstract sequences used in our semantic domain.

Definition 3.4. [Abstraction and Stable atoms] Let

$$\delta = \langle G_1, c_1, K_1, G_2, d_1 \rangle \dots \langle G_n, c_n, \emptyset, G_n, c_n \rangle$$

be a sequence of derivation steps where we assume that atoms are indexed as previously specified. We say that an indexed atom a^j is stable in δ if a^j appears in G_i , for each $1 \leq i \leq n$. The abstraction operator $\alpha : Seq \rightarrow \mathcal{D}$ is then defined inductively as

$$\begin{aligned} \alpha(\varepsilon) &= \varepsilon \\ \alpha(\langle G, c, K, G', d \rangle \cdot \delta') &= \beta(\langle c, K, H, d \rangle) \cdot \alpha(\delta') \end{aligned}$$

where H is the multiset consisting of all the user-defined atoms in G which are stable in $\langle G, c, K, G', d \rangle \cdot \delta'$ and the function β simply removes the indexes from the atoms in H .

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

EXAMPLE 3.5. *The application of the function α to the concrete sequence representing the derivation of Example 3.1 gives the following abstract sequence:*

$\langle \text{true}, \emptyset, \{edge(d, f), edge(d, e)\}, \{X = X1, c = Y1\} \rangle$
 $\langle \{X = X1, c = Y1\}, \emptyset, \{edge(d, f), edge(d, e)\}, \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\} \rangle$
 $\langle \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\}, \emptyset, \{edge(d, f), edge(d, e)\}, \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\} \rangle$
 $\langle \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\}, \emptyset, \{edge(d, f), edge(d, e)\}, c_1 \rangle$
 $\langle c_1, \emptyset, \{edge(Z3, V3), edge(d, f), edge(d, e)\}, c_2 \rangle$
 $\langle c_2, \emptyset, \{edge(Z3, V3), edge(d, f), edge(d, e)\}, c_3 \rangle$
 $\langle c_3, \emptyset, \{root(X4), edge(Z3, V3), edge(d, f), edge(d, e)\}, c_4 \rangle$
 $\langle c_4, \emptyset, \{root(X4), edge(Z3, V3), edge(d, f), edge(d, e)\}, c_4 \rangle$

where

$$\begin{aligned} c_1 &= \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c, X1 = X3, Z1 = Y3, c = Z3, d = V3\}, \\ c_2 &= \{X = d, X1 = d, X2 = c, X3 = d, Y1 = c, Y2 = c, Y3 = c, Z1 = c, Z3 = c, V3 = d\}, \\ c_3 &= \{X = d, X1 = d, X2 = c, X3 = d, Y1 = c, Y2 = c, Y3 = c, Z1 = c, Z3 = c, V3 = d, \\ &\quad Z3 = X4, Y3 = Y4, X2 = X4\} \text{ and} \\ c_4 &= \{X = d, X1 = d, X2 = c, X3 = d, X4 = c, Y1 = c, Y2 = c, Y3 = c, Y4 = c, Z1 = c, \\ &\quad Z3 = c, V3 = d\} \end{aligned}$$

Then we need the notion of “compatibility” of a tuple w.r.t. a sequence. To this aim we first provide some further notation: Given a sequence δ of derivation steps

$$\langle G_1, c_1, K_1, G_2, d_1 \rangle \langle G_2, c_2, K_2, G_3, d_2 \rangle \dots \langle G_n, c_n, \emptyset, G_n, c_n \rangle$$

we denote by $length(\delta)$ the length of the derivation δ (i.e. the number of tuples in the sequence). Moreover using t as a shorthand for the tuple $\langle G, c, K, G', d \rangle$ we define

- $V_{loc}(t) = Fv(G', d) \setminus Fv(G, c, K)$ (the local variables of t),
- $V_{ass}(\delta) = \bigcup_{i=1}^{n-1} Fv(K_i)$ (the free variables in the assumptions of δ),
- $V_{stable}(\delta) = Fv(H)$, where $H \subseteq G_n$ is the set consisting of all the user-defined atoms in G_n (the free variables in all the stable multisets of δ) and
- $V_{oc}(\delta) = \bigcup_{i=1}^{n-1} Fv(G_{i+1}, d_i) \setminus Fv(G_i, c_i, K_i)$ (the local variables of δ , namely the free variables in the clauses used in the derivation δ).

We then define the notion of compatibility as follows.

Definition 3.6. [Compatibility] Let $t = \langle G_1, c_1, K_1, G_2, d_1 \rangle$ a tuple representing a derivation step for the goal G_1 and let $\delta = \langle G_2, c_2, K_2, G_3, d_2 \rangle \dots \langle G_n, c_n, \emptyset, G_n, c_n \rangle$ be a sequence of derivation steps for G_2 . We say that t is compatible with δ if the following hold:

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

The three conditions of Definition 3.6 reflect the following facts respectively: the clauses in a derivation are renamed apart; the variables in the assumptions are disjoint from the variables in the clauses used in a derivation and each of the local variable appearing in an input constraint either has already appeared in an output constraint or is a variable appearing in the stable multisets of δ . These conditions ensure that, by using the notation of the definition above, if $t \cdot \delta \in Seq$ and 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 4.6). We can now define the compositional semantics.

Definition 3.7. [Compositional semantics] Let P be a program and let G be a goal. The compositional semantics of G in the program P , $\mathcal{S}_P : Goals \rightarrow \wp(\mathcal{D})$, is defined as

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

where α is the pointwise extension to sets of the operator given in Definition 3.4 and $\mathcal{S}'_P : Goals \rightarrow \wp(Seq)$ is defined as follows:

$$\begin{aligned} \mathcal{S}'_P(G) = & \{ \langle G, c, K, G', d \rangle \cdot \delta \in Seq \mid CT \not\models c \leftrightarrow \mathbf{false}, \langle G, c \rangle \xrightarrow{K}_P \langle G', d \rangle \\ & \text{and } \delta \in \mathcal{S}'_P(G') \text{ for some } \delta \text{ such that} \\ & \langle G, c, K, G', d \rangle \text{ is compatible with } \delta \} \\ & \cup \\ & \{ \langle G, c, \emptyset, G, c \rangle \in Seq \}. \end{aligned}$$

It can be observed that $\mathcal{S}'_P(G)$ is also the least fixed-point of the corresponding operator $\Phi \in (Goals \rightarrow \wp(Seq)) \rightarrow Goals \rightarrow \wp(Seq)$ defined by

$$\begin{aligned} \Phi(I)(G) = & \{ \langle G, c, K, G', d \rangle \cdot \delta \in Seq \mid CT \not\models c \leftrightarrow \mathbf{false}, \langle G, c \rangle \xrightarrow{K}_P \langle G', d \rangle \\ & \text{and } \delta \in I(G') \text{ for some } \delta \text{ such that} \\ & \langle G, c, K, G', d \rangle \text{ is compatible with } \delta \} \\ & \cup \\ & \{ \langle G, c, \emptyset, G, c \rangle \in Seq \}. \end{aligned}$$

In the above definition, $I : Goals \rightarrow \wp(Seq)$ stands for a generic interpretation assigning to a goal a set of sequences, and the ordering on the set of interpretations $Goals \rightarrow \wp(Seq)$ 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 [Davey and Priestley 1990]).

4. COMPOSITIONALITY AND CORRECTNESS

In this section we prove that the semantics defined above is and-compositional and correct w.r.t. the observables \mathcal{SAP} .

4.1 Composition operators

In order to prove the compositionality result we first need to define how two sequences describing a computation of two goals A and B can be composed in order to obtain a computation of the conjunction A, B . Such a composition is defined by the (semantic) operator \parallel which first performs an interleaving of the actions described by the two sequences and then eliminates the assumptions which are satisfied in the resulting sequence. For technical reasons, rather than modifying the existing sequences, the elimination of satisfied assumptions is performed on new sequences which are generated by a closure operator η defined as follows.

Definition 4.1. [η operator] Let W be a multiset of indexed atoms, σ be a sequence in \mathcal{D} of the form

$$\langle c_1, K_1, H_1, d_1 \rangle \langle c_2, K_2, H_2, d_2 \rangle \dots \langle c_n, K_n, H_n, d_n \rangle$$

and let us define

$$\tilde{H}_1 = H_1^1 \text{ and, for } i \in [2, n], \tilde{H}_i = \tilde{H}_{i-1} \uplus (H_i \setminus H_{i-1})^i \quad (1)$$

where we use the notation H^i to indicate that all the atoms in H are indexed by i (\setminus denotes the multisets difference).

Furthermore, let us denote by $\sigma \setminus W$ the sequence

$$\beta(\langle c_1, K_1, \tilde{H}_1 \setminus W, d_1 \rangle \langle c_2, K_2, \tilde{H}_2 \setminus W, d_2 \rangle \dots \langle c_n, K_n, \tilde{H}_n \setminus W, d_n \rangle)$$

where the multisets difference $\tilde{H}_i \setminus W$ considers indexes and, as in Definition 3.4, the function β simply removes the indexes from the stable atoms.

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

- (1) $S \subseteq \eta(S)$;
- (2) if $\sigma' \cdot \langle c, K, H, d \rangle \cdot \sigma'' \in \eta(S)$ then $(\sigma' \cdot \langle c, K \setminus K', H, d \rangle \cdot \sigma'') \setminus W \in \eta(S)$

where $K' = \{k_1, \dots, k_n\} \subseteq K$ is a multiset such that there exists a multiset of indexed atoms $W = \{h_1^{j_1}, \dots, h_n^{j_n}\} \subseteq \tilde{H}$, with \tilde{H} defined according to (1) and $CT \models c \wedge k_l \leftrightarrow c \wedge h_l$, for each $l \in [1, n]$.

A few explanations about the previous definition are in order. The operator η is an upper closure operator⁵ which saturates a set of sequences S by adding new sequences where redundant assumptions can be removed: an assumption k (in K_i) can be removed if h^j appears as a stable atom (in \tilde{H}_i) and $CT \models c_i \wedge k \leftrightarrow c_i \wedge h$. Once a stable atom is “consumed” for satisfying an assumption it is removed from (the multiset of stable atoms of) all the tuples appearing in the sequence, to avoid multiple uses of the same atom. Note that stable atoms are considered without the

⁵ $S \subseteq \eta(S)$ holds by definition, and it is easy to see that $\eta(\eta(S)) = \eta(S)$ holds and that $S \subseteq S'$ implies $\eta(S) \subseteq \eta(S')$.

index in the condition $CT \models c \wedge k_l \leftrightarrow c \wedge h_l$, while they are considered as indexed atoms in the removal operation $\tilde{H}_i \setminus W$. The reason for this slight complication is explained by the following example.

EXAMPLE 4.2. *Let S be the set consisting of the only sequence*

$$\langle c, \emptyset, \{k\}, d \rangle \langle c', \{k\}, \{k, k\}, d' \rangle \langle c'', \emptyset, \{k, k\}, c'' \rangle.$$

From this sequence, we construct a new one, where the stable atoms are indexed as follows:

$$\langle c, \emptyset, \{k^1\}, d \rangle \langle c', \{k\}, \{k^1, k^2\}, d' \rangle \langle c'', \emptyset, \{k^1, k^2\}, c'' \rangle.$$

Such a new sequence indicates that at the second step we have an assumption k , while both at the first and at the second step we have produced a stable atom k , which has been indexed by 1 and 2, respectively. In order to satisfy the assumption k we can use either k^1 or k^2 .

However, depending on what indexed atom we use, we obtain two different simplified sequences in $\eta(S)$, namely

$\langle c, \emptyset, \emptyset, d \rangle \langle c', \emptyset, \{k\}, d' \rangle \langle c'', \emptyset, \{k\}, c'' \rangle$ and $\langle c, \emptyset, \{k\}, d \rangle \langle c', \emptyset, \{k\}, d' \rangle \langle c'', \emptyset, \{k\}, c'' \rangle$, which describe correctly the two different situations.

It is also worth noting that it is possible to disregard indexes in the result of the normalization operator.

EXAMPLE 4.3. *We show now an example of derivation steps and abstract sequence for the goals $\{\text{same}(X, c), \text{edge}(c, d), \text{edge}(d, e)\}$ and $\{\text{root}(c), \text{edge}(c, b)\}$ that is a possible goal subdivision of the one in Example 3.1 using the program in Example 2.2. The goal $\{\text{same}(X, c), \text{edge}(c, d), \text{edge}(d, e)\}$ has the following sequence of derivation steps*

$$\begin{aligned} & \langle \{\mathbf{same}(X, c), \text{edge}(d, f), \text{edge}(d, e)\}, \mathbf{true} \rangle && \rightarrow^{\emptyset} \\ & \langle \{\text{find}(X1, Z1), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, \{X = X1, c = Y1\} \rangle \\ & \langle \{\mathbf{find}(X1, Z1), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, && \\ & \quad \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\} \rangle && \rightarrow^{\text{edge}(c, d)} \\ & \langle \{\mathbf{X3} = \mathbf{V3}, \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, && \\ & \quad \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c, X1 = X3, Z1 = Y3, c = Z3, d = V3\} \rangle && \rightarrow^{\emptyset} \\ & \langle \{\text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, && \\ & \quad \{X = d, X1 = d, X2 = c, X3 = d, Y1 = c, Y2 = c, Y3 = c, Z1 = c, Z3 = c, V3 = d\} \rangle \end{aligned}$$

and denoting by δ the concrete sequence arising from such derivation steps, we obtain $\delta =$

$$\begin{aligned} & \langle \{\text{same}(X, c), \text{edge}(d, f), \text{edge}(d, e)\}, \mathbf{true}, \emptyset, \\ & \quad \{\text{find}(X1, Z1), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, \{X = X1, c = Y1\} \rangle \\ & \langle \{\text{find}(X1, Z1), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\}, \\ & \quad \{\text{edge}(c, d)\}, \{\mathbf{X3} = \mathbf{V3}, \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_1 \rangle \\ & \langle \{X3 = V3, \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_1, \emptyset, \\ & \quad \{\text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_2 \rangle \\ & \langle \{\text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_4, \emptyset, \\ & \quad \{\text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_4 \rangle \end{aligned}$$

and then the abstract sequence $\alpha(\delta) =$

$$\begin{aligned} & \langle \{\mathbf{true}, \emptyset, \{\text{edge}(d, f), \text{edge}(d, e)\}, \{X = X1, c = Y1\} \rangle && (a) \\ & \langle \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\}, && \\ & \quad \{\text{edge}(c, d)\}, \{\text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_1 \rangle && (b) \\ & \langle c_1, \emptyset, \{\text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_2 \rangle && (c) \\ & \langle c_4, \emptyset, \{\text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_4 \rangle && (d) \end{aligned}$$

where

$$\begin{aligned} c_1 &= \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c, X1 = X3, Z1 = Y3, c = Z3, d = V3\} \\ c_2 &= \{X = d, X1 = d, X2 = c, X3 = d, Y1 = c, Y2 = c, Y3 = c, Z1 = c, Z3 = c, V3 = d\} \text{ and} \\ c_4 &= \{X = d, X1 = d, X2 = c, X3 = d, X4 = c, Y1 = c, Y2 = c, Y3 = c, Y4 = c, \\ &\quad Z1 = c, Z3 = c, V3 = d\} \end{aligned}$$

Moreover we have the following sequence of derivation steps for $\{\text{root}(c), \text{edge}(c, b)\}$

$$\begin{aligned} &\langle \{\text{root}(c), \text{edge}(c, d)\}, \{X = X1, c = Y1\} \rangle && \xrightarrow{\text{find}(Y1, Z1)} \\ &\langle \{\mathbf{X2} = \mathbf{Y2}, \text{root}(X2), \text{edge}(c, d)\}, \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\} \rangle && \xrightarrow{\emptyset} \\ &\langle \{\text{root}(X2), \text{edge}(c, d)\}, \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\} \rangle && \\ &\langle \{\text{root}(\mathbf{X2}), \text{edge}(c, d)\}, c_2 \rangle && \xrightarrow{\text{find}(Z3, Y3)} \\ &\langle \{\mathbf{X4} = \mathbf{Y4}, \text{root}(X4), \text{edge}(c, d)\}, c_3 \rangle && \xrightarrow{\emptyset} \\ &\langle \{\text{root}(X4), \text{edge}(c, d)\}, c_4 \rangle && \end{aligned}$$

and therefore we have that $\gamma =$

$$\begin{aligned} &\langle \{\text{root}(c), \text{edge}(c, d)\}, \{X = X1, c = Y1\}, \{\text{find}(Y1, Z1)\}, \\ &\quad \{X2 = Y2, \text{root}(X2), \text{edge}(c, d)\}, \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\} \rangle \\ &\langle \{X2 = Y2, \text{root}(X2), \text{edge}(c, d)\}, \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\}, \emptyset, \\ &\quad \{\text{root}(X2), \text{edge}(c, d)\}, \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\} \rangle \\ &\langle \{\text{root}(X2), \text{edge}(c, d)\}, c_2, \{\text{find}(Z3, Y3)\}, \\ &\quad \{X4 = Y4, \text{root}(X4), \text{edge}(c, d)\}, c_3 \rangle \\ &\langle \{X4 = Y4, \text{root}(X4), \text{edge}(c, d)\}, c_3, \emptyset, \{\text{root}(X4), \text{edge}(c, d)\}, c_4 \rangle \\ &\langle \{\text{root}(X4), \text{edge}(c, d)\}, c_4, \emptyset, \{\text{root}(X4), \text{edge}(c, d)\}, c_4 \rangle \end{aligned}$$

is a concrete sequence for $\{\text{root}(c), \text{edge}(c, b)\}$. Then $\alpha(\gamma)$ follows:

$$\begin{aligned} &\langle \{X = X1, c = Y1\}, \{\text{find}(Y1, Z1)\}, \{\text{edge}(c, d)\}, \\ &\quad \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\} \rangle && (e) \\ &\langle \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\}, \emptyset, \{\text{edge}(c, d)\}, \\ &\quad \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\} \rangle && (f) \\ &\langle c_2, \{\text{find}(Z3, Y3)\}, \{\text{edge}(c, d)\}, c_3 \rangle && (g) \\ &\langle c_3, \emptyset, \{\text{root}(X4), \text{edge}(c, d)\}, c_4 \rangle && (h) \\ &\langle c_4, \emptyset, \{\text{root}(X4), \text{edge}(c, d)\}, c_4 \rangle && (i) \end{aligned}$$

where

$$c_3 = \{X = d, X1 = d, X2 = c, X3 = d, Y1 = c, Y2 = c, Y3 = c, Z1 = c, Z3 = c, V3 = d, \\ Z3 = X4, Y3 = Y4, X2 = X4\}.$$

Before defining the composition operator \parallel on sequences we need a notation for the sequences in \mathcal{D} analogous to that one introduced for sequences of derivation steps:

Let $\sigma = \langle c_1, K_1, H_1, d_1 \rangle \langle c_2, K_2, H_2, d_2 \rangle \cdots \langle c_n, \emptyset, H_n, d_n \rangle \in \mathcal{D}$ be a sequence for the goal G . We define

$$\begin{aligned} &\cdot V_{\text{ass}}(\sigma) = \bigcup_{i=1}^{n-1} Fv(K_i) \text{ (the free variables in the assumptions of } \sigma), \\ &\cdot V_{\text{stable}}(\sigma) = Fv(H_n) = \bigcup_{i=1}^n Fv(H_i) \text{ (the free variables in the stable multisets of } \sigma), \\ &\cdot V_{\text{constr}}(\sigma) = \bigcup_{i=1}^{n-1} Fv(d_i) \setminus Fv(c_i) \text{ (the free variables in the output constraints of } \sigma \text{ which are not in the corresponding input constraints),} \\ &\cdot V_{\text{loc}}(\sigma) = (V_{\text{constr}}(\sigma) \cup V_{\text{stable}}(\sigma)) \setminus (V_{\text{ass}}(\sigma) \cup Fv(G)) \text{ (by using Definition 3.6 and by Lemma 4.6, the local variables of a sequence } \sigma \text{ are the local variables of the derivations } \delta \text{ such } \alpha(\delta) = \sigma). \end{aligned}$$

We can now define the composition operator \parallel on sequences. To simplify the notation we denote by \parallel both the operator acting on sequences and that one acting on sets of sequences.

Definition 4.4. [Composition operator \parallel] The operator $\parallel: \mathcal{D} \times \mathcal{D} \rightarrow \wp(\mathcal{D})$ is defined inductively as follows. Assume that $\sigma_1 = \langle c_1, K_1, H_1, d_1 \rangle \cdot \sigma'_1$ and $\sigma_2 = \langle c_2, K_2, H_2, d_2 \rangle \cdot \sigma'_2$ are sequences for the goals G_1 and G_2 , respectively. If

$$(V_{loc}(\sigma_1) \cup Fv(G_1)) \cap (V_{loc}(\sigma_2) \cup Fv(G_2)) = Fv(G_1) \cap Fv(G_2) \quad (2)$$

then $\sigma_1 \parallel \sigma_2$ is defined by cases as follows:

- (1) If both σ_1 and σ_2 have length 1 and have the same store, say $\sigma_1 = \langle c, \emptyset, H_1, c \rangle$ and $\sigma_2 = \langle c, \emptyset, H_2, c \rangle$, then

$$\sigma_1 \parallel \sigma_2 = \{ \langle c, \emptyset, H_1 \uplus H_2, c \rangle \}.$$

- (2) If σ_2 has length 1 and σ_1 has length > 1 then

$$\sigma_1 \parallel \sigma_2 = \{ \langle c_1, K_1, H_1 \uplus H_2, d_1 \rangle \cdot \sigma \in \mathcal{D} \mid \sigma \in \sigma'_1 \parallel \sigma_2 \}.$$

The symmetric case is analogous and therefore omitted.

- (3) If both σ_1 and σ_2 have length > 1 then

$$\begin{aligned} \sigma_1 \parallel \sigma_2 = & \{ \langle c_1, K_1, H_1 \uplus H_2, d_1 \rangle \cdot \sigma \in \mathcal{D} \mid \sigma \in \sigma'_1 \parallel \sigma_2 \} \\ & \cup \\ & \{ \langle c_2, K_2, H_1 \uplus H_2, d_2 \rangle \cdot \sigma \in \mathcal{D} \mid \sigma \in \sigma_1 \parallel \sigma'_2 \} \end{aligned}$$

Finally the composition of sets of sequences $\parallel: \wp(\mathcal{D}) \times \wp(\mathcal{D}) \rightarrow \wp(\mathcal{D})$ is defined by

$$\begin{aligned} S_1 \parallel S_2 = & \{ \sigma \in \mathcal{D} \mid \text{there exist } \sigma_1 \in S_1 \text{ and } \sigma_2 \in S_2 \text{ such that} \\ & \sigma = \langle c_1, K_1, H_1, d_1 \rangle \cdots \langle c_n, \emptyset, H_n, c_n \rangle \in \eta(\sigma_1 \parallel \sigma_2), \\ & (V_{loc}(\sigma_1) \cup V_{loc}(\sigma_2)) \cap V_{ass}(\sigma) = \emptyset \text{ and for } i \in [1, n] \\ & (V_{loc}(\sigma_1) \cup V_{loc}(\sigma_2)) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j) \cup Fv(H_i) \}. \end{aligned}$$

Let us briefly illustrate some points in the previous definition.

Condition (2) ensures that the rules used to construct the (derivations abstracted by the) sequences σ_1 and σ_2 have been renamed apart (that is, they do not share variables). Moreover, the local variables of each sequence are different from those which appear in the initial goal for the other sequence.

Moreover, in the definition of the composition of sets of sequences $\parallel: \wp(\mathcal{D}) \times \wp(\mathcal{D}) \rightarrow \wp(\mathcal{D})$, the first condition ensures that the variables appearing in the rules used to construct the sequences σ_1 and σ_2 are distinct from the variables appearing in the assumptions. The second condition is needed to ensure that σ is the abstraction of a sequence satisfying condition (3) in Definition 3.6 (Compatibility).

EXAMPLE 4.5. *We consider now an application of Definition 4.4 and of Definition 4.1 by using the two abstract sequences of Example 4.3, showing that this composition give us the sequence in Example 3.5. First of all we compose the abstract sequences $\alpha(\delta)$ and $\alpha(\gamma)$ of Example 4.3 by using Definition 4.4 and obtain the following interleaved sequence:*

$$\begin{aligned}
& \langle \{\text{true}, \emptyset, \{\text{edge}(c, d), \text{edge}(d, f), \text{edge}(d, e)\}, \{X = X1, c = Y1\}\} & (a(e)) \\
& \langle \{X = X1, c = Y1\}, \{\text{find}(Y1, Z1)\}, \{\text{edge}(c, d), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, & \\
& \langle \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\} \rangle & (e(b)) \\
& \langle \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\}, \emptyset, & \\
& \langle \{\text{edge}(c, d), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\} \rangle & (f(b)) \\
& \langle \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\}, \{\text{edge}(c, d)\}, & \\
& \langle \{\text{edge}(c, d), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_1 \rangle & (b(g)) \\
& \langle c_1, \emptyset, \{\text{edge}(c, d), \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_2 \rangle & (c(g)) \\
& \langle c_2, \{\text{find}(Z3, Y3)\}, & \\
& \langle \{\text{edge}(c, d), \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_3 \rangle & (g(d)) \\
& \langle c_3, \emptyset, \{\text{root}(X4), \text{edge}(c, d), \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_4 \rangle & (h(d)) \\
& \langle c_4, \emptyset, \{\text{root}(X4), \text{edge}(c, d), \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{find}(Y1, Z1), \text{edge}(d, f), \text{edge}(d, e)\}, c_4 \rangle & (i \text{ and } d)
\end{aligned}$$

where c_1 , c_2 , c_3 and c_4 is defined as in Example 4.3 and (a(e)) means that is used the tuple a and the stable atoms of tuple e (and analogously for the other steps) until the last step of interleaving, that uses d and h , closes the composition.

The application of Definition 4.1 substitutes the assumptions $\text{find}(Y1, Z1)$ in (e(b)) with the same constraint in the corresponding stable multiset.

$$\begin{aligned}
& \langle \{\text{true}, \emptyset, \{\text{edge}(c, d), \text{edge}(d, f), \text{edge}(d, e)\}, \{X = X1, c = Y1\}\} & (a(e)) \\
& \langle \{X = X1, c = Y1\}, \emptyset, \{\text{edge}(c, d), \text{edge}(d, f), \text{edge}(d, e)\}, & \\
& \langle \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\} \rangle & (e(b)) \\
& \langle \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\}, \emptyset, & \\
& \langle \{\text{edge}(c, d), \text{edge}(d, f), \text{edge}(d, e)\}, \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\} \rangle & (f(b)) \\
& \langle \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\}, \{\text{edge}(c, d)\}, & \\
& \langle \{\text{edge}(c, d), \text{edge}(d, f), \text{edge}(d, e)\}, c_1 \rangle & (b(g)) \\
& \langle c_1, \emptyset, \{\text{edge}(c, d), \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{edge}(d, f), \text{edge}(d, e)\}, c_2 \rangle & (c(g)) \\
& \langle c_2, \{\text{find}(Z3, Y3)\}, & \\
& \langle \{\text{edge}(c, d), \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{edge}(d, f), \text{edge}(d, e)\}, c_3 \rangle & (g(d)) \\
& \langle c_3, \emptyset, \{\text{root}(X4), \text{edge}(c, d), \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{edge}(d, f), \text{edge}(d, e)\}, c_4 \rangle & (h(d)) \\
& \langle c_4, \emptyset, \{\text{root}(X4), \text{edge}(c, d), \text{find}(Z3, Y3), \text{edge}(Z3, V3), \text{edge}(d, f), \text{edge}(d, e)\}, c_4 \rangle & (i \text{ and } d)
\end{aligned}$$

Note that another two applications of Definition 4.1 are possible by satisfying the assumptions $\text{edge}(c, d)$ in (b(g)) and $\text{find}(Z3, Y3)$ in (g(d)) with the same constraints in the corresponding stable multisets, and then the following sequence is produced:

$$\begin{aligned}
& \langle \{\text{true}, \emptyset, \{\text{edge}(d, f), \text{edge}(d, e)\}, \{X = X1, c = Y1\}\} & (a(e)) \\
& \langle \{X = X1, c = Y1\}, \emptyset, \{\text{edge}(d, f), \text{edge}(d, e)\}, & \\
& \langle \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\} \rangle & (e(b)) \\
& \langle \{X = X1, c = Y1, Y1 = X2, Z1 = Y2, c = X2\}, \emptyset, & \\
& \langle \{\text{edge}(d, f), \text{edge}(d, e)\}, \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\} \rangle & (f(b)) \\
& \langle \{X = X1, X2 = c, Y1 = c, Y2 = c, Z1 = c\}, \emptyset, & \\
& \langle \{\text{edge}(d, f), \text{edge}(d, e)\}, c_1 \rangle & (b(g)) \\
& \langle c_1, \emptyset, \{\text{edge}(Z3, V3), \text{edge}(d, f), \text{edge}(d, e)\}, c_2 \rangle & (c(g)) \\
& \langle c_2, \emptyset, \{\text{edge}(Z3, V3), \text{edge}(d, f), \text{edge}(d, e)\}, c_3 \rangle & (g(d)) \\
& \langle c_3, \emptyset, \{\text{root}(X4), \text{edge}(Z3, V3), \text{edge}(d, f), \text{edge}(d, e)\}, c_4 \rangle & (h(d)) \\
& \langle c_4, \emptyset, \{\text{root}(X4), \text{edge}(Z3, V3), \text{edge}(d, f), \text{edge}(d, e)\}, c_4 \rangle & (i \text{ and } d)
\end{aligned}$$

which is equal to the one of Example 3.5.

4.2 Compositionality

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 lemmas.

In the following, given a sequence γ , where $\gamma \in \text{Seq} \cup \mathcal{D}$, we will denote by $\text{instore}(\gamma)$ and by $\text{Inc}(\gamma)$ the first input constraint and the set of input constraints

of γ , respectively. Moreover, we will denote by $Ass(\gamma)$ and $Stable(\gamma)$ the set (corresponding to the multiset) of assumptions of γ and the set (corresponding to the multiset) of CHR atoms in the last goal of γ , respectively.

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

LEMMA 4.6. *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 $r \in \{ass, stable, loc\}$.*

PROOF. If $r \in \{ass, stable\}$ then the proof is straightforward by definition of α and of V_r . Then we have only to prove that $V_{loc}(\delta) = V_{loc}(\sigma)$. The proof is by induction on $n = length(\delta)$.

($n = 1$). In this case $\delta = \langle G, c, \emptyset, G, c \rangle$, $\sigma = \langle c, \emptyset, H, c \rangle$, where either $H = G$ or $c_n = \mathbf{false}$ and H is the multiset consisting of all the user-defined atoms in G . Therefore, by definition $V_{loc}(\delta) = V_{loc}(\sigma) = \emptyset$.

($n \geq 1$). Let $\delta = \langle G_1, c_1, K_1, G_2, d_1 \rangle \langle G_2, c_2, K_2, G_3, d_3 \rangle \cdots \langle G_n, c_n, \emptyset, G_n, c_n \rangle$, where $G = G_1$.

By definition of $\mathcal{S}'_P(G)$, there exists $\delta' \in \mathcal{S}'_P(G_2)$ such that $t = \langle G_1, c_1, K_1, G_2, d_1 \rangle$ is compatible with δ' and $\delta = t \cdot \delta' \in Seq$.

By inductive hypothesis, we have that $V_{loc}(\delta') = V_{loc}(\sigma')$, where $\sigma' = \alpha(\delta')$. Moreover, by definition of α , $\sigma = \langle c_1, K_1, H_1, d_1 \rangle \cdot \sigma'$, where H_1 is the multiset consisting of all the atoms in G_1 which are stable in δ .

By definition of V_{loc} and by inductive hypothesis

$$\begin{aligned} V_{loc}(\delta) &= \bigcup_{i=1}^{n-1} Fv(G_{i+1}, d_i) \setminus Fv(G_i, c_i, K_i) \\ &= V_{loc}(\delta') \cup (Fv(G_2, d_1) \setminus Fv(G_1, c_1, K_1)) \\ &= V_{loc}(\sigma') \cup (Fv(G_2, d_1) \setminus Fv(G_1, c_1, K_1)). \end{aligned} \quad (3)$$

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

$$V_{loc}(\sigma') = (V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma') \cup Fv(G_2)). \quad (4)$$

Therefore by (3), by properties of \cup and since $Fv(G_2) \cap Fv(G_1, c_1, K_1) \subseteq Fv(G_2) \cap Fv(G_1)$, we have that

$$\begin{aligned} V_{loc}(\delta) &= ((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma') \cup Fv(G_2))) \cup \\ &\quad (Fv(G_2) \setminus Fv(G_1)) \cup (Fv(d_1) \setminus Fv(G_1, c_1, K_1)). \end{aligned} \quad (5)$$

Now, let $x \in Fv(K_1) \cap (V_{constr}(\sigma') \cup V_{stable}(\sigma))$. By definition $x \in Fv(t)$, since t is compatible with δ' and by condition (1) of Definition 3.6 (Compatibility), we have that $x \notin V_{loc}(\delta') = V_{loc}(\sigma')$ and therefore by (4) $x \in V_{ass}(\sigma') \cup Fv(G_2)$. Then by (5)

$$\begin{aligned} V_{loc}(\delta) &= ((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup Fv(G_2))) \\ &\quad \cup (Fv(G_2) \setminus Fv(G_1)) \cup (Fv(d_1) \setminus Fv(G_1, c_1, K_1)). \end{aligned} \quad (6)$$

By properties of \cup , we have that

$$\begin{aligned}
& ((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus V_{ass}(\sigma) \cup Fv(G_2)) \cup \\
& (Fv(G_2) \setminus Fv(G_1)) = \\
& ((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup (Fv(G_2) \cap Fv(G_1)))) \cup \\
& (Fv(G_2) \setminus Fv(G_1)). \tag{7}
\end{aligned}$$

Now let $x \in Fv(G_1) \setminus Fv(G_2)$ and let us assume that $x \in V_{constr}(\sigma') \cup V_{stable}(\sigma)$. By definition $x \in Fv(t)$, since t is compatible with δ' and by condition (1) of Definition 3.6 (Compatibility), we have that $x \notin V_{loc}(\delta') = V_{loc}(\sigma')$ (where the last equality follows by inductive hypothesis). Then since $x \notin Fv(G_2)$ and by (4) we have that $x \in V_{ass}(\sigma')$. Therefore, by the previous results and by (6) and (7), we have that

$$\begin{aligned}
V_{loc}(\delta) &= ((V_{constr}(\sigma') \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup Fv(G_1))) \cup \\
& (Fv(G_2) \setminus Fv(G_1)) \cup (Fv(d_1) \setminus Fv(G_1, c_1, K_1)). \tag{8}
\end{aligned}$$

Now let $x \in (Fv(d_1) \setminus Fv(c_1)) \cap V_{ass}(\sigma')$. Since by point (2) of Definition 3.6 (Compatibility) $V_{loc}(t) \cap V_{ass}(\sigma') = \emptyset$, we have that $x \in Fv(G_1, K_1)$. Then

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

Then by (8),

$$\begin{aligned}
V_{loc}(\delta) &= ((V_{constr}(\sigma) \cup V_{stable}(\sigma)) \setminus (V_{ass}(\sigma) \cup Fv(G_1))) \cup \\
& (Fv(G_2) \setminus Fv(G_1)). \tag{9}
\end{aligned}$$

Finally let $x \in Fv(G_2) \setminus Fv(G_1)$. We prove that $x \in ((V_{constr}(\sigma) \cup V_{stable}(\sigma)) \setminus V_{ass}(\sigma))$. First of all, observe that $x \in V_{loc}(t)$ and therefore, by definition of compatibility, $x \notin V_{ass}(\sigma)$. Now, let $\{a_1, \dots, a_l\} \subseteq G_2$ the set of atoms in G_2 such that $x \in Fv(a_j)$, for each $j \in [1, l]$. We have two cases.

—there exists $v \in [1, l]$ such that a_v is a CHR constraint and let us assume that $a_v \notin Stable(\sigma) = Stable(\delta)$. Then, by definition of derivation, there exists $j \in [1, n-1]$ such that $x \in Fv(d_j)$. Let h the least index $j \in [1, n-1]$ such that $x \in Fv(d_h)$. By condition (3) of Definition 3.6 (Compatibility), we have that $x \notin Fv(c_h)$ and then $x \in Fv(d_k) \setminus Fv(c_k) \subseteq V_{constr}(\sigma)$. Then by (9), by the previous result and by definition of V_{loc} ,

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

and then the thesis holds

- for each $v \in [1, l]$, a_v is a built-in constraint. Now, we have two further cases.
 - c_n is satisfiable. In this case, by Definition 3.2 (Concrete sequences), we have that a_v is evaluated in δ , for each $v \in [1, l]$. Analogously to the previous case, by condition (3) of Definition 3.6 (Compatibility), we have that $x \in V_{constr}(\sigma)$.
 - $c_n = \mathbf{false}$. Then by definition of the operational semantics, without loss of generality, we can assume that δ evaluates at least a constraint in $\{a_1, \dots, a_l\}$. Therefore, as before, $x \in V_{constr}(\sigma)$.

Now the proof is the same as that one of the previous case.

□

In the following, given a sequence of derivation steps

$$\delta = \langle B_1, c_1, K_1, B_2, d_1 \rangle \dots \langle B_n, c_n, \emptyset, B_n, c_n \rangle$$

and a goal W , we denote by $\delta \oplus W$ the sequence

$$\langle (B_1, W), c_1, K_1, (B_2, W), d_1 \rangle \dots \langle (B_n, W), c_n, \emptyset, (B_n, W), c_n \rangle$$

and by $\delta \ominus W$ the sequence

$$\langle B_1 \setminus W, c_1, K_1, B_2 \setminus W, d_1 \rangle \dots \langle B_n \setminus W, c_n, \emptyset, B_n \setminus W, c_n \rangle.$$

The proof of the following two lemma is straightforward by definition of derivation.

LEMMA 4.7. *Let F, G be goals and let $\delta \in \mathcal{S}'_P(F, G)$ such that*

$$\delta = \langle (F, G), c_1, K_1, R_2, d_1 \rangle \langle R_2, c_2, K_2, R_3, d_2 \rangle \dots \langle R_n, c_n, \emptyset, R_n, c_n \rangle$$

where $F = (F', F'')$, $F'' \neq \emptyset$ and the first tuple of the sequence δ represents a derivation step s , which uses the **Apply'** rule and rewrites only and all the atoms in (F'', G) . Then there exists a derivation $\delta' \in \mathcal{S}'_P(F)$ such that

$$\delta' = \langle F, c_1, K_1 \uplus G, R_2, d_1 \rangle \langle R_2, c_2, K_2, R_3, d_2 \rangle \dots \langle R_n, c_n, \emptyset, R_n, c_n \rangle.$$

LEMMA 4.8. *Let G be a goal, W be a multiset of atoms and let $\delta \in \mathcal{S}'_P(G)$ such that $Fv(W) \cap V_{loc}(\delta) = \emptyset$. Then $\delta \oplus W \in \mathcal{S}'_P(G, W)$.*

The following lemma proves that we can obtain the same concrete semantics both from a goal and from one part of it. It is used in Lemma 4.10.

LEMMA 4.9. *Let P be a program and let F and G be two goals such that there exists a derivation step*

$$s = \langle (F, G), c_1 \rangle \xrightarrow{K_1} \langle (B, G), d_1 \rangle,$$

where only the atoms in F are rewritten in s .

Assume that there exists $\delta \in \mathcal{S}'_P(F, G)$ such that $\delta = t \cdot \delta'$, where

$$t = \langle (F, G), c_1, K_1, (B, G), d_1 \rangle,$$

$\delta' \in \mathcal{S}'_P(B, G)$ 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) for $i = 1, 2$, $V_{loc}(\delta'_i) \subseteq V_{loc}(\delta')$ and $Inc(\delta'_i) \subseteq Inc(\delta')$.
- (2) $Ass(\delta'_1) \subseteq Ass(\delta') \cup Stable(\delta'_2)$ and $Ass(\delta'_2) \subseteq Ass(\delta') \cup Stable(\delta'_1)$,
- (3) $\alpha(\delta'_1) \parallel \alpha(\delta'_2)$ is defined and $\alpha(\delta') \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta'_2))$.

Then $\delta_1 = t' \cdot \delta'_1 \in \mathcal{S}'_P(F)$, where $t' = \langle F, c_1, K_1, B, d_1 \rangle$, $\alpha(\delta_1) \parallel \alpha(\delta'_2)$ is defined and $\alpha(\delta) \in \eta(\alpha(\delta_1) \parallel \alpha(\delta'_2))$.

PROOF. In the following, assume that

$$\begin{aligned}\delta'_1 &= \langle B_1, e_1, M_1, B_2, f_1 \rangle \langle B_2, e_2, M_2, B_3, f_2 \rangle \cdots \langle B_l, e_l, \emptyset, B_l, e_l, \rangle \\ \delta'_2 &= \langle G_1, r_1, N_1, G_2, s_1 \rangle \langle G_2, r_2, N_2, G_3, s_2 \rangle \cdots \langle G_p, r_p, \emptyset, G_p, r_p \rangle \\ \delta' &= \langle R_1, c_2, K_2, R_2, d_2 \rangle \langle R_2, c_3, K_3, R_3, d_3 \rangle \cdots \langle R_{n-1}, c_n, \emptyset, R_{n-1}, c_n \rangle,\end{aligned}$$

where $B_1 = B$, $G_1 = G$, $R_1 = (B, G)$ and $e_l = r_p = c_n$. The following holds.

(a) t' represents the derivation step $s' = \langle F, c_1 \rangle \xrightarrow{K_1} \langle B, d_1 \rangle$.

The proof is straightforward by observing that t represents the derivation step $s = \langle (F, G), c_1 \rangle \xrightarrow{K_1} \langle (B, G), d_1 \rangle$, which uses only atoms in F .

(b) $\delta_1 \in \mathcal{S}'_P(F)$.

By considering the previous point, by hypothesis and by definition of $\mathcal{S}'_P(F)$, we have to prove that $\delta_1 \in \text{Seq}$ and that t' is compatible with δ'_1 . By hypothesis $\text{Inc}(\delta'_1) \subseteq \text{Inc}(\delta')$ and then $CT \models \text{instore}(\delta'_1) \rightarrow \text{instore}(\delta')$. Moreover since t is compatible with δ' , we have that $CT \models \text{instore}(\delta') \rightarrow d_1$ and therefore $CT \models \text{instore}(\delta'_1) \rightarrow d_1$ by transitivity. Then we have only to prove that t' is compatible with δ'_1 and so that the three conditions of Definition 3.6 hold. The following holds.

(1) By hypothesis $V_{loc}(\delta'_1) \subseteq V_{loc}(\delta')$ and by 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) First of all observe that given a derivation $\tilde{\delta}$, we have that

$$V_{\text{Stable}}(\tilde{\delta}) \subseteq Fv(\tilde{G}) \cup V_{loc}(\tilde{\delta}), \quad (10)$$

where \tilde{G} is the initial goal of the derivation $\tilde{\delta}$. Then have that

$$\begin{aligned}V_{loc}(t') \cap V_{ass}(\delta'_1) &\subseteq \\ &(\text{since } V_{loc}(t') = V_{loc}(t) \text{ and since by hypothesis} \\ &\quad \text{Ass}(\delta'_1) \subseteq \text{Ass}(\delta') \cup \text{Stable}(\delta'_2)) \\ V_{loc}(t) \cap (V_{ass}(\delta') \cup V_{\text{Stable}}(\delta'_2)) &\subseteq \\ &(\text{by (10)}) \\ V_{loc}(t) \cap (V_{ass}(\delta') \cup Fv(G) \cup V_{loc}(\delta'_2)) &\subseteq \\ &(\text{since by hypothesis } V_{loc}(\delta'_2) \subseteq V_{loc}(\delta')) \\ V_{loc}(t) \cap (V_{ass}(\delta') \cup Fv(G) \cup V_{loc}(\delta')) &= \\ &(\text{since } t \text{ is compatible with } \delta' \text{ and by definition of } V_{loc}) \\ \emptyset &\end{aligned}$$

(3) We have to prove that for $i \in [1, l]$, $V_{loc}(t') \cap Fv(e_i) \subseteq \bigcup_{j=1}^{i-1} Fv(f_j) \cup Fv(d_1) \cup V_{\text{Stable}}(\delta'_1)$. Let $i \in [1, l]$ and let $x \in V_{loc}(t') \cap Fv(e_i)$.

Since by inductive hypothesis $\text{Inc}(\delta'_1) \subseteq \text{Inc}(\delta')$, there exists a least index $h \in [2, n]$ such that $e_i = c_h$. Therefore, since $V_{loc}(t') = V_{loc}(t)$ and t is compatible with δ' , we have that

$$x \in \bigcup_{j=1}^{h-1} Fv(d_j) \cup V_{\text{Stable}}(\delta'). \quad (11)$$

Moreover, since $x \in V_{loc}(t') = V_{loc}(t)$, t is compatible with δ' and by hypothesis $V_{loc}(\delta'_2) \subseteq V_{loc}(\delta')$

$$x \notin Fv(G) \cup V_{loc}(\delta'_2). \quad (12)$$

Now, observe that

$$\begin{aligned} V_{stable}(\delta') &\subseteq (\text{by definition of } \parallel \text{ and since by hypothesis} \\ &\quad \alpha(\delta') \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta'_2))) \\ V_{stable}(\delta'_1) \cup V_{stable}(\delta'_2) &\subseteq (\text{by (10)}) \\ V_{stable}(\delta'_1) \cup Fv(G) \cup V_{loc}(\delta'_2). & \end{aligned}$$

Then by (11) and (12), we have that $x \in \bigcup_{j=1}^{h-1} Fv(d_j) \cup V_{stable}(\delta'_1)$. Then to prove the thesis, we have to prove that

if $x \in \bigcup_{j=1}^{h-1} Fv(d_j) \cup V_{stable}(\delta'_1)$ then $x \in \bigcup_{j=1}^{i-1} Fv(f_j) \cup Fv(d_1) \cup V_{stable}(\delta'_1)$.

Let us assume that $x \in \bigcup_{j=2}^{h-1} Fv(d_j)$ and let k the least index $j \in [2, h-1]$ such that $x \in Fv(d_j)$.

If d_k is an output constraint of δ'_1 , i.e. there exists $j \in [1, i-1]$ such that $d_k = f_j$, the proof is terminated.

Now assume that d_k is an output constraint of δ'_2 , i.e. there exists $w \in [1, m]$ such that $d_k = s_w$ and for each $j \in [1, w-1]$, we have that $x \notin Fv(s_j)$. Since k is the least index j such that $x \in Fv(d_j)$ and since t is compatible with δ' , we have that $x \notin Fv(c_k)$ and therefore $x \notin Fv(r_w)$.

Moreover, since by (12), $x \notin Fv(G) \cup V_{loc}(\delta'_2)$, we have that $x \notin Fv(G_w)$. Then by definition of derivation step, since $x \in Fv(s_w) \setminus (Fv(r_w) \cup Fv(G_w))$, we have that $x \in Fv(N_w)$ and therefore $x \in V_{ass}(\delta'_2)$. By hypothesis $x \in V_{ass}(\delta') \cup V_{stable}(\delta'_1)$. Then since t is compatible with δ' and $x \in V_{loc}(t)$, we have that $x \notin V_{ass}(\delta')$ and therefore $x \in V_{stable}(\delta'_1)$ and then the proof.

(c) $\alpha(\delta_1) \parallel \alpha(\delta'_2)$ is defined.

We have to prove that

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

By Lemma 4.6

$$V_{loc}(\alpha(\delta_1)) = V_{loc}(\alpha(\delta'_1)) \cup V_{loc}(t') \quad (13)$$

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

$$V_{loc}(\alpha(\delta'_1)) \cap (V_{loc}(\alpha(\delta'_2)) \cup Fv(G)) = \emptyset. \quad (14)$$

Now observe that, since t is compatible with δ' , $V_{loc}(t') = V_{loc}(t)$ and by Lemma 4.6, we have that $V_{loc}(t') \cap V_{loc}(\alpha(\delta')) = \emptyset$. Moreover, by hypothesis for $V_{loc}(\alpha(\delta'_2)) \subseteq V_{loc}(\alpha(\delta'))$ and by definition of t , we have that $Fv(G) \cap V_{loc}(t') = Fv(G) \cap V_{loc}(t) = \emptyset$. Then

$$\begin{aligned} V_{loc}(\alpha(\delta_1)) \cap (V_{loc}(\alpha(\delta'_2)) \cup Fv(G)) &= \\ (V_{loc}(\alpha(\delta'_1)) \cup V_{loc}(t')) \cap (V_{loc}(\alpha(\delta'_2)) \cup Fv(G)) &= \emptyset. \end{aligned}$$

Moreover, since t is compatible with δ' , $Fv(F) \subseteq Fv(t)$ and by hypothesis $V_{loc}(\alpha(\delta'_2)) \subseteq V_{loc}(\alpha(\delta'))$

$$Fv(F) \cap V_{loc}(\alpha(\delta'_2)) \subseteq Fv(F) \cap V_{loc}(\alpha(\delta')) = \emptyset$$

and then the thesis holds.

(d) $\alpha(\delta) \in \eta(\alpha(\delta_1) \parallel \alpha(\delta'_2))$.

By hypothesis $\alpha(\delta') \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta'_2))$, $\alpha(\delta) = \langle c_1, K_1, W_1, d_1 \rangle \cdot \alpha(\delta')$ and $\alpha(\delta_1) =$

$\langle c_1, K_1, J_1, d_1 \rangle \cdot \alpha(\delta'_1)$, where W_1 is the multiset of atoms in (F, G) which are not rewritten in δ and J_1 is the multiset of atoms in F which are not rewritten in δ_1 .

Moreover let us denote by

- J_2 the set of atoms in B which are not rewritten in δ'_1 , by
- Y_1 the set of atoms in G which are not rewritten in δ'_2 and by
- W_2 the set of atoms in (B, G) which are not rewritten in δ' .

Since $\alpha(\delta') \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta'_2))$ there exists $\sigma' \in \mathcal{D}$ such that

$$\sigma' \in \alpha(\delta'_1) \parallel \alpha(\delta'_2) \text{ and } \alpha(\delta') \in \eta(\{\sigma'\}).$$

By our assumptions, $\sigma' = \langle c_2, H_1, J_2 \uplus Y_1, d_2 \rangle \cdot \sigma''$ and by definition of \parallel ,

$$\sigma = \langle c_1, K_1, J_1 \uplus Y_1, d_1 \rangle \cdot \sigma' \in \alpha(\delta_1) \parallel \alpha(\delta'_2).$$

By definition of η and since $\alpha(\delta') \in \eta(\{\sigma'\})$,

$$\langle c_1, K_1, (J_1 \uplus Y_1) \setminus S, d_1 \rangle \cdot \alpha(\delta') \in \eta(\alpha(\delta_1) \parallel \alpha(\delta'_2)), \quad (15)$$

where the multisets difference $(J_1 \uplus Y_1) \setminus S$ considers indexes and S is such that $(J_2 \uplus Y_1) \setminus S = W_2$. Then we can choose S in such a way that S restricted to the atoms with index equal to 1 is the set of (non-indexed) atoms $(J_1 \uplus Y_1) \setminus W_1$ and S restricted to the atoms with index equal to 2 is the set of (non-indexed) atoms $(J_2 \setminus J_1) \setminus (W_2 \setminus W_1)$. It is easy to check that S satisfies the condition $(J_2 \uplus Y_1) \setminus S = W_2$. Moreover, by construction $(J_1 \uplus Y_1) \setminus S = W_1$. Therefore by (15)

$$\alpha(\delta) = \langle c_1, K_1, W_1, d_1 \rangle \cdot \alpha(\delta') \in \eta(\alpha(\delta_1) \parallel \alpha(\delta'_2))$$

and this completes the proof.

□

Following lemma states that given a concrete sequence derived by the goal (F, G) , there exist two concrete sequences that are derived from F and G , for which exists an abstract composition that is equal to the abstraction of the given sequence.

LEMMA 4.10. *Let P be a program, F and G be two goals and assume that $\delta \in \mathcal{S}'_P(F, G)$. Then there exists $\delta_1 \in \mathcal{S}'_P(F)$ and $\delta_2 \in \mathcal{S}'_P(G)$, such that $\alpha(\delta) \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$.*

PROOF. We construct, by induction on the $l = \text{length}(\delta)$ two sequences $\delta \uparrow_{(F,G)} = (\delta_1, \delta_2)$, where

- (1) for $i = 1, 2$, $V_{loc}(\delta_i) \subseteq V_{loc}(\delta)$ and $Inc(\delta_i) \subseteq Inc(\delta)$ (and therefore $CT \models \text{instore}(\delta_i) \rightarrow \text{instore}(\delta)$).
- (2) $Ass(\delta_1) \subseteq Ass(\delta) \cup Stable(\delta_2)$ and $Ass(\delta_2) \subseteq Ass(\delta) \cup Stable(\delta_1)$,
- (3) $\delta_1 \in \mathcal{S}'_P(F)$, $\delta_2 \in \mathcal{S}'_P(G)$, $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined and $\alpha(\delta) \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$.

($l = 1$). In this case $\delta = \langle (F, G), c, \emptyset, (F, G), c \rangle$. We define

$$\delta \uparrow_{(F,G)} = (\langle (F, c, \emptyset, F, c) \rangle, \langle (G, c, \emptyset, G, c) \rangle) = (\delta_1, \delta_2),$$

where $\delta_1 \in \mathcal{S}'_P(F)$ and $\delta_2 \in \mathcal{S}'_P(G)$. By definition for $i = 1, 2$, $V_{loc}(\delta_i) = \emptyset$, $Inc(\delta_i) = \{c\} = Inc(\delta)$ and $Ass(\delta_i) = \emptyset$.

Moreover $\alpha(\delta_1) = \langle c, \emptyset, F, c \rangle$ and $\alpha(\delta_2) = \langle c, \emptyset, G, c \rangle$ and then $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined. Now the proof is straightforward by definition of \parallel .

($l > 1$). Assume that $\delta \in \mathcal{S}'_P(F, G)$. By definition

$$\delta = \langle (F, G), c_1, K_1, B_2, d_1 \rangle \cdot \delta',$$

where $\delta' \in \mathcal{S}'_P(B_2)$ and $t = \langle (F, G), c_1, K_1, B_2, d_1 \rangle$ is compatible with δ' . Recall that, by definition, the tuple t represents a derivation step

$$s = \langle (F, G), c_1 \rangle \xrightarrow{K_1} \langle B_2, d_1 \rangle.$$

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

—In the derivation step s , we use the **Solve'** rule. In this case, without loss of generality, we can assume that $F = (c, F')$,

$$s = \langle (F, G), c_1 \rangle \xrightarrow{\emptyset} \langle (F', G), d_1 \rangle,$$

$CT \models c_1 \wedge c \leftrightarrow d_1$, $t = \langle (F, G), c_1, \emptyset, (F', G), d_1 \rangle$ and $\delta' \in \mathcal{S}'_P(F', G)$. Moreover $\alpha(\delta) = \langle c_1, \emptyset, W, d_1 \rangle \cdot \alpha(\delta')$, where W is the first stable multiset of $\alpha(\delta')$.

By inductive hypothesis there exist $\delta'_1 \in \mathcal{S}'_P(F')$ and $\delta_2 \in \mathcal{S}'_P(G)$ such that $\delta' \uparrow_{(F', G)} = (\delta'_1, \delta_2)$, $\alpha(\delta'_1) \parallel \alpha(\delta_2)$ is defined and $\alpha(\delta') \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta_2))$. Then, we define

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

By definition $\langle F, c_1 \rangle \xrightarrow{\emptyset} \langle F', d_1 \rangle$, $t' = \langle F, c_1, \emptyset, F', d_1 \rangle$ represents a derivation step for F , $Fv(d_1) \subseteq Fv(F) \cup Fv(c_1)$ and therefore $V_{loc}(t') = \emptyset$. Then the following holds.

- (1) Let $i \in [1, 2]$. By the inductive hypothesis, by construction and by the previous observation $V_{loc}(\delta_i) \subseteq V_{loc}(\delta') = V_{loc}(\delta)$ and $Inc(\delta_i) \subseteq Inc(\delta') \cup \{c_1\} = Inc(\delta)$.
- (2) By inductive hypothesis and by construction, $Ass(\delta_1) = Ass(\delta'_1) \subseteq Ass(\delta') \cup Stable(\delta_2) = Ass(\delta) \cup Stable(\delta_2)$ and $Ass(\delta_2) \subseteq Ass(\delta') \cup Stable(\delta'_1) = Ass(\delta) \cup Stable(\delta_1)$.
- (3) By inductive hypothesis $\delta_2 \in \mathcal{S}'_P(G)$. The proof of the other statements follows by Lemma 4.9 and by inductive hypothesis.

—In the derivation step s , we use the **Simplify'** rule and let us assume that in the derivation step s atoms deriving from F only are rewritten.

In this case, we can assume that $s = \langle (F, G), c_1 \rangle \xrightarrow{K_1} \langle (B, G), d_1 \rangle$, $\delta' \in \mathcal{S}'_P(B, G)$ and $t = \langle (F, G), c_1, K_1, (B, G), d_1 \rangle$. By inductive hypothesis there exist $\delta'_1 \in \mathcal{S}'_P(B)$ and $\delta_2 \in \mathcal{S}'_P(G)$ such that $\delta' \uparrow_{(B, G)} = (\delta'_1, \delta_2)$, $\alpha(\delta'_1) \parallel \alpha(\delta_2)$ is defined and $\alpha(\delta') \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta_2))$. Then, we define

$$\delta \uparrow_{(F, G)} = (\delta_1, \delta_2) \text{ where } \delta_1 = \langle F, c_1, K_1, B, d_1 \rangle \cdot \delta'_1.$$

By definition $\langle F, c_1 \rangle \xrightarrow{K_1} \langle B, d_1 \rangle$, $t' = \langle F, c_1, K_1, B, d_1 \rangle$ represents a derivation step for F and $V_{loc}(t') = V_{loc}(t)$.

Now the following holds.

- (1) Let $i \in [1, 2]$. By the inductive hypothesis, by construction and by the previous observation $V_{loc}(\delta_i) \subseteq V_{loc}(\delta') \cup V_{loc}(t) = V_{loc}(\delta)$ and $Inc(\delta_i) \subseteq Inc(\delta') \cup \{c_1\} = Inc(\delta)$.

(2) By inductive hypothesis and by construction,

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

and

$$Ass(\delta_2) \subseteq Ass(\delta') \cup Stable(\delta'_1) \subseteq Ass(\delta) \cup Stable(\delta_1).$$

(3) By inductive hypothesis $\delta_2 \in \mathcal{S}'_P(G)$. The proof of the other statements follows by Lemma 4.9 and by inductive hypothesis.

—In the derivation step s , we use the **Simplify'** rule and let us assume that in the derivation step s atoms deriving both from F and G are rewritten.

In this case, we can assume that $F = (F', F'')$, $G = (G', G'')$, $F'' \neq \emptyset$, $G'' \neq \emptyset$, $s = \langle (F, G), c_1 \rangle \xrightarrow{K_1} \langle (F', G', B), d_1 \rangle$, $\delta' \in \mathcal{S}'_P(F', G', B)$ and $t = \langle (F, G), c_1, K_1, (F', G', B), d_1 \rangle$.

By using the same arguments as those of the previous point there exist $\delta'_1 \in \mathcal{S}'_P(F, G'')$ and $\delta'_2 \in \mathcal{S}'_P(G'')$ such that $\delta \uparrow_{((F, G''), G')} = (\delta'_1, \delta'_2)$.

Now, observe that, by Lemma 4.7 and by definition of \uparrow , there exists $\delta_1 \in \mathcal{S}'_P(F)$ such that $Ass(\delta_1) = Ass(\delta'_1) \cup \{G''\}$, $\alpha(\delta'_1) = \langle c_1, K_1, W_1, d_1 \rangle \cdot \sigma_1$, $\alpha(\delta_1) = \langle c_1, K_1 \uplus \{G''\}, W_1, d_1 \rangle \cdot \sigma_1$ and $V(\delta_1) = V(\delta'_1)$ for $V \in \{V_{loc}, Inc, Stable\}$.

Moreover, since $\delta \in \mathcal{S}'_P(F, G)$ and $V_{loc}(\delta'_2) \subseteq V_{loc}(\delta)$, we have that $Fv(G'') \cap V_{loc}(\delta'_2) = \emptyset$. Then by Lemma 4.8, we have that $\delta_2 = \delta'_2 \oplus \tilde{G}'' \in \mathcal{S}'_P(G)$. By construction $Stable(\delta_2) = Stable(\delta'_2) \cup \{G''\}$ and $V(\delta_2) = V(\delta'_2)$ for $V \in \{V_{loc}, Inc, Ass\}$.

Then, we define

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

Now the following holds.

- (1) Let $i \in [1, 2]$. By definition of \uparrow and by the previous observation $V_{loc}(\delta_i) = V_{loc}(\delta'_i) \subseteq V_{loc}(\delta)$ and $Inc(\delta_i) = Inc(\delta'_i) \subseteq Inc(\delta)$.
- (2) By definition of \uparrow and by construction $Ass(\delta_1) = Ass(\delta'_1) \cup \{G''\} \subseteq Ass(\delta) \cup Stable(\delta'_2) \cup \{G''\} = Ass(\delta) \cup Stable(\delta_2)$ and $Ass(\delta_2) = Ass(\delta'_2) \subseteq Ass(\delta) \cup Stable(\delta'_1) = Ass(\delta) \cup Stable(\delta_1)$.
- (3) 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 defined. Finally, the proof that $\alpha(\delta) \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$ follows by observing that by definition of \uparrow , $\alpha(\delta) \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta'_2))$ and by construction $\eta(\alpha(\delta'_1) \parallel \alpha(\delta'_2)) \subseteq \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$.

□

The following Lemma states that given two concrete sequences, that are derived from the goals F and G , there exists a concrete sequence, that is derived from (F, G) , which abstraction is equal to the abstraction of composition of the given two sequences.

LEMMA 4.11. *Let P be a program, let F and G be two goals and assume that $\delta_1 \in \mathcal{S}'_P(F)$ and $\delta_2 \in \mathcal{S}'_P(G)$ are two sequences such that the following hold:*

- (1) $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined,

- (2) $\sigma = \langle c_1, K_1, W_1, d_1 \rangle \cdots \langle c_n, \emptyset, W_n, c_n \rangle \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$,
 (3) $(V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma) = \emptyset$,
 (4) for $i \in [1, n]$, $(V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j) \cup Fv(W_i)$.

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

PROOF. In the following, given two derivations $\delta_1 \in \mathcal{S}'_P(F)$ and $\delta_2 \in \mathcal{S}'_P(G)$, which verify the previous conditions, we construct by induction on the $l = \text{length}(\sigma)$ a derivation $\delta \in \mathcal{S}'_P(F, G)$ such that $V_{loc}(\delta) \subseteq V_{loc}(\delta_1) \cup V_{loc}(\delta_2)$ and $\sigma = \alpha(\delta)$.

($l = 1$). In this case $\delta_1 = \langle F, c, \emptyset, F, c \rangle$, $\delta_2 = \langle G, c, \emptyset, G, c \rangle$, $\alpha(\delta_1) = \langle c, \emptyset, F, c \rangle$, $\alpha(\delta_2) = \langle c, \emptyset, G, c \rangle$, $\sigma = \langle c, \emptyset, (F, G), c \rangle$ and $\delta = \langle (F, G), c, \emptyset, (F, G), c \rangle$.

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

$$\begin{aligned} \delta_1 &= t' \cdot \delta'_1, \quad \delta_2 = \langle G, e_1, J_1, G_2, f_1 \rangle \cdot \delta'_2, \\ \sigma_1 &= \alpha(\delta_1) = \langle c_1, L_1, N_1, d_1 \rangle \cdot \alpha(\delta'_1) \text{ and} \\ \sigma_2 &= \alpha(\delta_2) = \langle e_1, J_1, M_1, f_1 \rangle \cdot \sigma'_2, \end{aligned}$$

where $t' = \langle F, c_1, L_1, F_2, d_1 \rangle$, $\delta'_1 \in \mathcal{S}'_P(F_2)$, $\sigma \in \eta(\langle c_1, L_1, N_1 \uplus M_1, d_1 \rangle \cdot \bar{\sigma})$ and $\bar{\sigma} \in \alpha(\delta'_1) \parallel \sigma_2$.

By definition of η , there exist the multisets of atoms L', \bar{L}, L and the sequence σ' such that

$$\sigma = \langle c_1, L_1 \setminus L, ((N_1 \uplus M_1) \setminus \bar{L}) \setminus L', d_1 \rangle \cdot (\sigma' \setminus L'),$$

where $\sigma' \in \eta(\bar{\sigma}) \subseteq \eta(\alpha(\delta'_1) \parallel \sigma_2)$, $K_1 = L_1 \setminus L$ and $W_1 = ((N_1 \uplus M_1) \setminus \bar{L}) \setminus L'$. Now the following holds

- (1) $\alpha(\delta'_1) \parallel \alpha(\delta_2)$ is defined. By definition, we have to prove that

$$(V_{loc}(\alpha(\delta'_1)) \cup Fv(F_2)) \cap (V_{loc}(\alpha(\delta_2)) \cup Fv(G)) = Fv(F_2) \cap Fv(G).$$

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 have that $V_{loc}(\alpha(\delta'_1)) \cap (V_{loc}(\alpha(\delta_2)) \cup Fv(G)) = \emptyset$ and $(Fv(F) \cup V_{loc}(\alpha(\delta_1)) \cap (V_{loc}(\alpha(\delta_2)) = \emptyset$.

Now, observe that by definition of derivation, $Fv(F_2) \subseteq Fv(F) \cup V_{loc}(\alpha(\delta_1))$. Therefore, by the previous observations, $Fv(F_2) \cap V_{loc}(\alpha(\delta_2)) = \emptyset$ and then the thesis.

- (2) $\sigma' = \langle c_2, K_2, W_2 \uplus L', d_2 \rangle \cdots \langle c_n, \emptyset, W_n \uplus L', c_n \rangle \in \eta(\alpha(\delta'_1) \parallel \alpha(\delta_2))$. The proof is straightforward, by definition of \parallel .
 (3) By definition, by the hypothesis and by Lemma 4.6, we have that

$$\begin{aligned} (V_{loc}(\alpha(\delta'_1)) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma') &\subseteq \\ (V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma) &= \emptyset. \end{aligned}$$

- (4) For $i \in [2, n]$,

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

To prove this statement observe that by hypothesis and by Lemma 4.6, for $i \in [2, n]$,

$$\begin{aligned} (V_{loc}(\alpha(\delta'_1)) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) &\subseteq \\ (V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) &\subseteq \end{aligned}$$

$$\bigcup_{j=1}^{i-1} Fv(d_j) \cup Fv(W_i). \quad (16)$$

Let $i \in [2, n]$, such that there exists $x \in (V_{loc}(\alpha(\delta'_1)) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) \cap Fv(d_1)$. We have to prove that $x \in Fv(W_i)$ and then the thesis.

First of all, observe that since $x \in Fv(d_1)$, by definition of derivation, we have that $x \notin V_{loc}(\alpha(\delta'_1))$ and therefore $x \in V_{loc}(\alpha(\delta_2)) \cap Fv(c_i) \cap Fv(d_1)$.

Moreover, since by hypothesis $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined, we have that $x \notin Fv(F) \cup V_{loc}(t')$. Therefore, since $x \in Fv(d_1)$ and by definition of derivation, we have that $x \in Fv(L_1) \cup Fv(c_1)$. Now we have two possibilities

— $x \in Fv(c_1)$. In this case, since $x \in V_{loc}(\alpha(\delta_2))$ and by point 4 of the hypothesis, we have that $x \in Fv(W_i)$.

— $x \in Fv(L_1)$. In this case there exists $h \in L_1$ such that $x \in Fv(h)$. Since by hypothesis $(V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma) = \emptyset$, we have that $h \notin Ass(\sigma)$ (i.e. $h \notin K_1$) and therefore, by definition of \parallel , there exists $h' \in G$ such that $CT \models c_1 \wedge h \leftrightarrow c_1 \wedge h'$. Note that, since $x \in V_{loc}(\alpha(\delta_2))$, we have that $x \notin Fv(G) \supseteq Fv(h')$. Then $x \in Fv(c_1)$ and then analogously to the previous case, $x \in Fv(W_i)$.

Then, by (16),

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

and then the thesis.

By the previous results and by inductive hypothesis, we have that there exists $\bar{\delta} \in \mathcal{S}'_P(F_2, G)$ such that $V_{loc}(\bar{\delta}) \subseteq V_{loc}(\delta'_1) \cup V_{loc}(\delta_2)$ and $\sigma' = \alpha(\bar{\delta})$. Moreover by definition of η , $L' \subseteq (F_2, G)$ is a multiset of atoms which are stable in $\bar{\delta}$. Then $\delta' = \bar{\delta} \ominus L' \in \mathcal{S}'_P(B)$, where the goal B is obtained from the goal (F_2, G) by deleting the atoms in L' . By construction

$$V_{loc}(\delta') = V_{loc}(\bar{\delta}) \text{ and } V_{ass}(\delta') = V_{ass}(\bar{\delta}). \quad (17)$$

Now observe that since $t' = \langle F, c_1, L_1, F_2, d_1 \rangle$ represents a derivation step for F , we have that $t = \langle (F, G), c_1, K_1, B, d_1 \rangle$ represents a derivation step for (F, G) . Let us denote by δ the sequence $t \cdot \delta'$.

Then, to prove the thesis, we have to prove that $V_{loc}(\delta) \subseteq V_{loc}(\delta_1) \cup V_{loc}(\delta_2)$, $t \cdot \delta' \in Seq$, t is compatible with δ' (and therefore $\delta \in \mathcal{S}'_P(F, G)$) and $\sigma = \alpha(\delta)$.

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

$$\begin{aligned} V_{loc}(\delta) &= \text{by construction} \\ V_{loc}(t) \cup V_{loc}(\delta') &= \text{by (17)} \\ V_{loc}(t') \cup V_{loc}(\bar{\delta}) &\subseteq \text{by inductive hypothesis} \\ V_{loc}(t') \cup V_{loc}(\delta'_1) \cup V_{loc}(\delta_2) &= \text{by construction} \\ V_{loc}(\delta_1) \cup V_{loc}(\delta_2) & \end{aligned}$$

and then the thesis.

$(t \cdot \delta' \in Seq)$. By construction, we have only to prove that $CT \models instore(\delta') \rightarrow d_1$. The proof is straightforward, since by construction either $instore(\delta') = instore(\delta'_1)$ or $instore(\delta') = instore(\delta_2)$.

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

- (1) $V_{loc}(\delta') \cap Fv(t) = \emptyset$. By construction, (17) and by inductive hypothesis

$$\begin{aligned} V_{loc}(t) &= V_{loc}(t'), \quad Fv(t) = Fv(t') \cup Fv(G) \text{ and} \\ V_{loc}(\delta') &\subseteq V_{loc}(\delta'_1) \cup V_{loc}(\delta_2). \end{aligned} \quad (18)$$

By definition of derivation and since $\alpha(\delta'_1) \parallel \alpha(\delta_2)$ is defined, we have that $V_{loc}(\delta'_1) \cap (Fv(t') \cup Fv(G)) = \emptyset$ and therefore by the second statement in (18)

$$V_{loc}(\delta'_1) \cap Fv(t) = \emptyset. \quad (19)$$

By point 3 of the hypothesis $Fv(K_1) \cap V_{loc}(\delta_2) = \emptyset$. Moreover, since by definition of α and \parallel , $W_1 \subseteq (F, G)$, we have that

$$\begin{aligned} Fv(c_1) \cap V_{loc}(\delta_2) &\subseteq (\text{by point 4 of the hypothesis}) \\ Fv(W_1) \cap V_{loc}(\delta_2) &\subseteq (\text{by the previous observation}) \\ Fv(F, G) \cap V_{loc}(\delta_2) &= (\text{by definition of derivation and} \\ &\quad \text{since } \alpha(\delta_1) \parallel \alpha(\delta_2) \text{ is defined}) \\ \emptyset & \end{aligned}$$

Finally, since $\alpha(\delta_1) \parallel \alpha(\delta_2)$ is defined we have that $(Fv(F) \cup V_{loc}(t')) \cap V_{loc}(\delta_2) = \emptyset$. Then by definition and by (18)

$$Fv(t) \cap V_{loc}(\delta_2) = (Fv(c_1, F, K_1) \cup V_{loc}(t')) \cap V_{loc}(\delta_2) = \emptyset. \quad (20)$$

Then

$$\begin{aligned} V_{loc}(\delta') \cap Fv(t) &\subseteq (\text{by the last statement in (18)}) \\ (V_{loc}(\delta'_1) \cup V_{loc}(\delta_2)) \cap Fv(t) &\subseteq (\text{by (19)}) \\ V_{loc}(\delta_2) \cap Fv(t) &= (\text{by (20)}) \\ \emptyset. & \end{aligned}$$

- (2) $V_{loc}(t) \cap V_{ass}(\delta') = \emptyset$. The proof is immediate by the second statement of (17), since $\sigma' = \alpha(\bar{\delta})$, $V_{ass}(\sigma') \subseteq V_{ass}(\sigma)$, by the first statement in (18), since $V_{loc}(t') \subseteq V_{loc}(\delta_1)$ and by point 3 of the hypothesis.

- (3) for $i \in [2, n]$, $V_{loc}(t) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j) \cup V_{stable}(\delta')$. By construction, since $\delta' = \bar{\delta} \ominus L'$, $\sigma' = \alpha(\bar{\delta})$ and $Stable(\sigma') = W_n \uplus L'$, we have that $Stable(\delta') = W_n$.

Then the proof is immediate by observing that $V_{loc}(t) = V_{loc}(t') \subseteq V_{loc}(\delta_1)$, for $i \in [2, n]$, $W_i \subseteq W_n$ and by point 4 of the hypothesis.

($\sigma = \alpha(\delta)$).. By inductive hypothesis $\sigma' = \alpha(\bar{\delta})$ and then by construction $\sigma' \setminus L' = \alpha(\delta')$. Then

$$\sigma = \langle c_1, K_1, W_1, d_1 \rangle \cdot (\sigma' \setminus L') = \langle c_1, K_1, W_1, d_1 \rangle \cdot \alpha(\delta') = \alpha(\delta),$$

where the last equality follows by observing that $\delta = t \cdot \delta'$, where

$$t = \langle (F, G), c_1, K_1, B, d_1 \rangle$$

and W_1 is the multiset of all the atoms in (F, G) , which are stable in δ .

□

By using the above results we can finally prove the following theorem.

THEOREM 4.12. [*Compositionality*] *Let P be a program and let F and G be two goals. Then*

$$\mathcal{S}_P(F, G) = \mathcal{S}_P(F) \parallel \mathcal{S}_P(G)$$

holds.

PROOF. We prove the two inclusions separately.

($\mathcal{S}_P(F, G) \subseteq \mathcal{S}_P(F) \parallel \mathcal{S}_P(G)$). Let $\sigma \in \mathcal{S}_P(F, G)$. By definition of \mathcal{S}_P , there exists $\delta \in \mathcal{S}'_P(F, G)$ such that $\sigma = \alpha(\delta)$. By Lemma 4.10 there exist $\delta_1 \in \mathcal{S}'_P(F)$ and $\delta_2 \in \mathcal{S}'_P(G)$ such that for $i = 1, 2$, $V_{loc}(\delta_i) \subseteq V_{loc}(\delta)$ and $\sigma \in \eta(\alpha(\delta_1) \parallel \alpha(\delta_2))$. Let

$$\delta = \langle (F, G), c_1, K_1, B_2, d_1 \rangle \cdots \langle B_n, c_n, \emptyset, B_n, c_n \rangle$$

and let $\sigma = \langle c_1, K_1, F_1, d_1 \rangle \cdots \langle c_n, \emptyset, F_n, c_n \rangle$, where $F_n = B_n$. Then in order to prove the thesis we have only to show that

$$\begin{aligned} (V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap V_{ass}(\sigma) &= \emptyset \text{ and for } i \in [1, n], \\ (V_{loc}(\alpha(\delta_1)) \cup V_{loc}(\alpha(\delta_2))) \cap Fv(c_i) &\subseteq \bigcup_{j=1}^{i-1} Fv(d_j) \cup Fv(F_i). \end{aligned}$$

First observe that by Lemma 4.6 and by hypothesis, we have that $V_{ass}(\sigma) = V_{ass}(\delta)$ and for $i = 1, 2$, $V_{loc}(\alpha(\delta_i)) = V_{loc}(\delta_i) \subseteq V_{loc}(\delta)$. Then by the previous results and by the properties of the derivations

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

Moreover by condition (3) of Definition 3.6 (Compatibility), for $i \in [1, m]$,

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

holds. Now, observe that if $x \in V_{loc}(\delta) \cap Fv(c_i) \cap V_{stable}(\delta)$, then $x \in \bigcup_{j=1}^i V_{loc}(\delta) \cap Fv(B_j) \cap V_{stable}(\delta)$ and then $x \in Fv(F_i)$ and this completes the proof of the first inclusion.

($\mathcal{S}_P(F, G) \supseteq \mathcal{S}_P(F) \parallel \mathcal{S}_P(G)$). Let $\sigma \in \mathcal{S}_P(F) \parallel \mathcal{S}_P(G)$. By definition of \mathcal{S}_P and of \parallel there exist $\delta_1 \in \mathcal{S}'_P(F)$ and $\delta_2 \in \mathcal{S}'_P(G)$, such that $\sigma_1 = \alpha(\delta_1)$, $\sigma_2 = \alpha(\delta_2)$, $\sigma_1 \parallel \sigma_2$ is defined, $\sigma = \langle c_1, K_1, F_1, d_1 \rangle \cdots \langle c_n, \emptyset, F_n, c_n \rangle \in \eta(\sigma_1 \parallel \sigma_2)$, $(V_{loc}(\sigma_1) \cup V_{loc}(\sigma_2)) \cap V_{ass}(\sigma) = \emptyset$ and for $i \in [1, n]$, $(V_{loc}(\sigma_1) \cup V_{loc}(\sigma_2)) \cap Fv(c_i) \subseteq \bigcup_{j=1}^{i-1} Fv(d_j) \cup Fv(F_i)$. The proof is then straightforward by using Lemma 4.11. \square

4.3 Correctness

In order to show the correctness of the semantics \mathcal{S}_P w.r.t. the (input/output) observables \mathcal{SA}_P , we first introduce a different characterization of \mathcal{SA}_P obtained by using the new transition system defined in Table II.

Definition 4.13. 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 II. We define

$$\begin{aligned} \mathcal{SA}'_P(G) &= \{ \exists _Fv(G)c \mid \langle G, \emptyset \rangle \xrightarrow{\emptyset}_P \cdots \xrightarrow{\emptyset}_P \langle \emptyset, c \rangle \not\xrightarrow{K}_P \} \\ &\cup \\ &\{ \mathbf{false} \mid \langle \tilde{G}, \emptyset \rangle \xrightarrow{\emptyset}_P \cdots \xrightarrow{\emptyset}_P \langle G', \mathbf{false} \rangle \}. \end{aligned}$$

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

PROPOSITION 4.14. *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 data sufficient computations can be obtained from \mathcal{S} by considering suitable sequences, namely those sequences which do not perform assumptions neither on CHR constraints nor on built-in constraints. The first condition means that the second components of tuples must be empty, while the second one means that the assumed constraint at step i must be equal to the produced constraint at step $i-1$. We call “connected” those sequences which satisfy these requirements:

Definition 4.15. [Connected sequences] Assume that

$$\sigma = \langle c_1, K_1, H_1, d_1 \rangle \dots \langle c_n, K_n, H_n, c_n \rangle$$

is a sequence in \mathcal{D} . We say that σ is connected if

- (1) $K_i = \emptyset$ for each i , $1 \leq i \leq n$,
- (2) $d_j = c_{j+1}$ for each j , $1 \leq j \leq n-1$ and
- (3) either $H_n = \emptyset$ or $c_n = \mathbf{false}$.

The proof of the following result derives from the definition of connected sequence and an easy inductive argument.

Given a sequence $\sigma = \langle c_1, K_1, H_1, d_1 \rangle \dots \langle c_n, K_n, H_n, d_n \rangle$, $\mathit{instore}(\sigma)$ and $\mathit{store}(\sigma)$ are the built-in constraint c_1 and the built-in constraint d_n , respectively.

PROPOSITION 4.16. *Let P be a program and let G be a goal. Then*

$$\mathcal{SA}'_P(G) = \{ \exists_{-Fv(G)} c \mid \text{there exists } \sigma \in \mathcal{S}_P(G) \text{ such that } \mathit{instore}(\sigma) = \emptyset \\ \sigma \text{ is connected and } c = \mathit{store}(\sigma) \}.$$

The following corollary is immediate from Proposition 4.14.

COROLLARY 4.17. [*Correctness*] *Let P be a program and let G be a goal. Then*

$$\mathcal{SA}_P(G) = \{ \exists_{-Fv(G)} c \mid \text{there exists } \sigma \in \mathcal{S}_P(G) \text{ such that } \mathit{instore}(\sigma) = \emptyset \\ \sigma \text{ is connected and } c = \mathit{store}(\sigma) \}.$$

4.4 A possible extension

The previous corollary proves that our semantics is correct w.r.t. a notion of observable which considers data sufficient answers. As shown by Definition 2.3 these are obtained by considering the results of terminated computations where all the user-defined constraints have been rewritten into built-in constraints. It could be desirable to obtain a compositional characterization also for qualified answers: these are obtained by considering computations terminating with a user-defined constraint which does not need to be empty (Definition 2.4).

Such a characterization could be obtained by extending our model to include termination modes. The problem here is that, given a tuple $\langle G, c, K, G', d \rangle$, in order

to reconstruct correctly the qualified answers we need to know whether the configuration $\langle G', d \rangle$ is terminating or not (that is, we need to know whether $\langle G', d \rangle \not\rightarrow_P^{K'}$ holds). This information could be provided by introducing in our semantics suitable termination modes. Hence a correct compositional semantics could be obtained also for qualified answers, at the price of a further complication of the traces.

5. RELATED WORK

In this paper we have considered the original abstract operational semantics of CHR given in [Frühwirth 1998] which, as previously discussed, can introduce trivial infinite computations. A semantics which avoids these computations was first defined in [Abdennadher 1997] and then refined in several other works. Essentially, the idea of [Abdennadher 1997] was to add a suitable *token store* to the global state in order to memorize the information about which propagation rules can be applied to the user-defined constraints. In this way, one can control that a propagation rule is applied at most once to the same sequence of constraints, thus avoiding trivial infinite computations. In [Gabbrielli et al. 2006] the present work has been extended to consider the classic operational semantics based on the token store. Such an extension is mainly technical and leads to a more complicated model which, however, uses the same approach presented here. In fact, the main difference of the construction in [Gabbrielli et al. 2006] w.r.t. the present one is that tuples of the form $\langle G, c, T, K, G', T', d \rangle$ are considered in the traces, where T and T' are sets modeling the token store.

A *refined semantics* of CHR has been defined in [Duck et al. 2004] in order to describe precisely the operational semantics implicitly used by (Prolog) implementations of CHR. This is obtained by considering explicitly (via suitable indexes) the (top down) order in which rules are applied. In principle a compositional characterization of the semantics in [Duck et al. 2004] could be given along the lines of the approach presented in this paper. However, this would result in a really complicated model, since one should model the fact that some traces representing the computations of two goals, say A and B , cannot be composed to obtain a trace for A, B , due to the order of application of the rules.

Recently the semantics of CHR has been analyzed more deeply from a logical perspective in [Betz and Frühwirth 2005] and [Meister et al. 2007] by using linear logic and transaction logic, respectively. The problem addressed by these papers is that the usual first order reading of CHR rules is not adequate when considering CHR as a general purpose concurrent language (rather than a language for constraint handling). In fact, such a reading may lead to a logical meaning which is inconsistent with the intended meaning of a program (the coin example in [Betz and Frühwirth 2005] and [Meister et al. 2007] describes this point very clearly). In order to describe more precisely the operational semantics of CHR more expressive logics should be then used, such as those used in the above mentioned papers. The aim of both [Betz and Frühwirth 2005] and [Meister et al. 2007] is therefore different from ours, even though these different approaches could be integrated in order to obtain a logical characterization of a compositional semantics. More precisely, since transaction logic allows to describe logically CHR computations (see Theorem 2 in [Meister et al. 2007]) in principle one could think of a logical characterization

of the traces that we use in our compositional semantics. Such a characterization however is not immediate, since our traces use assumptions and therefore are different from those arising from the “normal” CHR computations. It is worth noting that a logical characterization of traces containing assumptions was proposed in [de Boer et al. 2004] by using suitable (logical) modalities which provide a kind of assumption/commitment style of specification of a process. However, the traces in [de Boer et al. 2004] were used to model the behavior of (timed) concurrent constraint programming processes, hence they were much more simpler than those used here and it is not clear whether their logical characterization could be extended to the traces of this paper.

When considering more generally concurrent languages other than CHR one can find several compositional models based on traces. In fact, starting with the semantics for dataflow languages of [Jonsson 1985], one can find trace models for imperative concurrent languages [Brookes 1993], for concurrent constraint programming [de Boer and Palamidessi 1991], for Linda-like languages [de Boer et al. 2004] and also several trace semantics for more theoretical process calculi. However, as previously argued, the presence of multiple heads in CHR rules makes this language quite peculiar and rather different from the other concurrent languages previously mentioned. Hence our approach is substantially different from those of the above mentioned papers.

Finally we should mention that the problem of a compositional characterizations of CHR rules was also addressed by [Maher 2002], but only for a limited subset of CHR.

6. CONCLUSIONS

In this paper we have introduced the first semantics for CHR which is compositional w.r.t. the and-composition of goals and which is correct w.r.t. data sufficient answers. Compositionality is an important feature for a semantics, especially in the field of concurrent languages. Indeed, most of the (many) semantics for concurrency which have been investigated are compositional w.r.t. the syntactic operators of the language. Compositionality facilitates the semantic reasoning on processes, as one has to consider congruences rather than equivalences. Also, having a reference semantics which is compositional sets the ground for defining modular verification and analysis tools, which are important from a practical point of view. In fact the ability to decompose a proof or an analysis according to the syntactic components of a program, possible with compositional tools, allows one to manage the complexity of large systems by controlling the state explosion problem (typical of many model checking tools) and allows to verify and analyze partially defined software components. Of course, compositional verification and analysis tools can be defined directly, following one of the many techniques available (typically some form of assume-guarantee reasoning, see for example [Henzinger et al. 1998]). However, it is worth noting that starting from an existing compositional semantics can simplify the definition of these tools. Consider for example a proof system *à la* Hoare which allows to prove program properties. Such a proof system clearly needs to be compositional, since we want to reduce the proof of a property for a compound statement to the proofs available for its components (this is usually reflected by

the rules of the proof system, which are given in the natural deduction style). The definition of such a compositional proof system can be based on a compositional semantics, as done in [de Boer et al. 1997; de Boer et al. 2004] for (timed) ccp languages. Analogously, static analysis based on abstract interpretation can be done in a compositional way, if one starts from a compositional semantics (see, for example, [Codish et al. 1993; Falaschi et al. 1993]).

Our work could be extended along several different lines.

As previously mentioned, a more complicated model (including termination modes) could be defined in order to characterize in a compositional way also qualified answers.

A second, more interesting, possible extension is the investigation of the full abstraction issue. For obvious reasons it would be desirable to introduce in the semantics the minimum amount of information needed to obtain compositionality, while preserving correctness. In other terms, one would like to obtain a result of this kind: $\mathcal{S}_P(G) = \mathcal{S}_P(G')$ if and only if, for any F , $\mathcal{SAP}(G, F) = \mathcal{SAP}(G', F)$ (our Corollary 4.17 only ensures that the “only if” part holds). Such a full abstraction result is difficult to achieve: techniques similar to those used in [de Boer and Palamidessi 1991; de Boer et al. 2004] for analogous results in the context of ccp could be considered, even though they cannot be directly used, due the multiple heads of CHR rules.

It would be interesting also to study further notions of compositionality, for example that one which considers union of program rules rather than conjunctions of goals, analogously to what has been done in [Bossi et al. 1992] for logic programming. However, due to the presence of synchronization, the simple model based on clauses defined in [Bossi et al. 1992] cannot be used for CHR.

ACKNOWLEDGMENTS

We thank Michael Maher for having initially suggested the problem of compositionality for CHR semantics. We also thank the anonymous reviewers for their precise comments which helped us to improve the paper.

REFERENCES

- ABDENNADHER, S. 1997. Operational semantics and confluence of constraint propagation rules. In G. Smolka, editor, *Proc. Third Int'l Conf. on Principles and Practice of Constraint Programming (CP 97)*, Lecture Notes in Computer Science 1330. Springer-Verlag.
- ABDENNADHER, S. AND FRÜHWIRTH, T. 2003. *Essentials of Constraint Programming*. Springer-Verlag.
- BETZ, H. AND FRÜHWIRTH, T. 2005. A Linear-Logic Semantics for Constraint Handling Rules. In *Proc. 11th Int'l Conf. on Principles and Practice of Constraint Programming (CP 05)*, Lecture Notes in Computer Science 3709. Springer-Verlag.
- DE BOER, F. S., GABBRIELLI, M., MARCHIORI, E., AND PALAMIDESSI, C. 1997. Proving concurrent constraint programs correct. *Transactions on Programming Languages and Systems (TOPLAS)*, 19(5): 685–725. ACM Press.
- DE BOER, F. S., GABBRIELLI, M., AND MEO, M. C. 2000. A timed concurrent constraint language. *Information and Computation*, 161(1): 45–83.
- DE BOER, F. S., GABBRIELLI, M., AND MEO, M. C. 2004. Proving correctness of Timed Concurrent Constraint Programs. *ACM Transactions on Computational Logic* 5(4): 706–731.
- DE BOER, F. S., GABBRIELLI, M., AND MEO, M. C. 2004. A timed linda language and its denotational semantics. *Fundamenta Informaticae* 63(4): 309 – 330.

- DE BOER, F. S., KOK, J. N., PALAMIDESSI, C., AND RUTTEN, J. J. M. M. 1991. The failure of failures in a paradigm for asynchronous communication. In J. C. M. Baeten and J. F. Groote, editors, *Proc. of CONCUR'91*, Lecture Notes in Computer Science, 527: 111–126. Springer-Verlag.
- DE BOER, F. S. AND PALAMIDESSI, C. 1991. A fully abstract model for concurrent constraint programming. In S. Abramsky and T. S. E. Maibaum, editors, *Proc. of TAPSOFT/CAAP*, Lecture Notes in Computer Science, 493: 296–319. Springer-Verlag.
- BOSSI, A., GABBRIELLI, M., LEVI, G., AND MEO, M. C. 1992. A Compositional Semantics for Logic Programs. *Theoretical Computer Science* 122(1-2): 3–47.
- BROOKES, S. 1993. A fully abstract semantics of a shared variable parallel language. In *Proc. Eighth IEEE Symposium on Logic In Computer Science*. IEEE Computer Society Press.
- CODISH, M., DEBRAY, S., AND GIACOBAZZI, R. 1993. Compositional analysis of modular logic programs. In *Proc. 20th ACM Symp. on Principles of Programming Languages*, 451–464. ACM Press.
- DAVEY, B. A. AND PRIESTLEY, H. A. 1990. Introduction to Lattices and Order. Cambridge University Press.
- DELZANNO, G., GABBRIELLI, M., AND MEO, M. C. 2005. A Compositional Semantics for CHR. In *PPDP'05 Lisbon, Portugal*. ACM Press.
- DUCK, G. J., DE LA BANDA, M. G., AND STUCKEY, P. J. 2004. The Refined Operational Semantics of Constraint Handling Rules. In *Proc. of the 20th International Conference on Logic Programming, (ICLP'04)*.
- FALASCHI, M., GABBRIELLI, M., MARRIOTT, K., AND PALAMIDESSI, C. 1993. Compositional analysis of concurrent constraint languages. In *Proc. Eighth IEEE Symposium on Logic In Computer Science*. IEEE Computer Society Press.
- FRÜHWIRTH, T. 1991. Introducing simplification rules. TR ECRC-LP-63, ECRC Munich. October 1991.
- FRÜHWIRTH, T. 1998. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1-3): 95–138.
- GABBRIELLI, M., MEO, M. C., AND TACCHELLA, P. 2006. A Compositional Semantics for CHR with Propagation Rules. In *Proceedings of the Third Workshop on Constraint Handling Rules*. Report CW 425, June 2006, Katholieke Universiteit Leuven.
- HENZINGER, T. A., QADEER, S., AND RAJAMANI, S. K. 1998. You assume we guarantee: Methodology and case studies. In *Proceedings of the CAV 98: Computed-aided verification*. Lecture Notes in Computer Science, pp. 440–451. Springer-Verlag.
- JONSSON, B. 1985. A model and a proof system for asynchronous processes. In *Proc. of the 4th ACM Symp. on Principles of Distributed Computing*: 49–58. ACM Press.
- MAHER, M. 2002. Propagation Completeness of Reactive Constraints. In *Proc. International Conference on Logic Programming (ICLP)*: 148–162.
- MEISTER, M., DJELLOUL, K., AND ROBIN, J. 2007. A unified semantics for CHR in transaction logic. In *Proc. of 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007)*. Lecture Notes in Artificial Intelligence, 4483: 201–213. Springer-Verlag.

Received March 2006; revised May 2007; accepted November 2007