Toward a Logically Complete Fixpoint Semantics for Constraint Handling Rules

March 2011

facultad de informática

universidad politécnica de madrid

Rémy Haemmerlé

TR Number CLIP3/2011.0

Technical Report Number: CLIP3/2011.0
March, 2011

Authors

Rémy Haemmerlé
Technical University of Madrid

Keywords

CHR, coinduction, fixpoint, declarative semantics, persistent constraints.

Abstract

In this paper, we address the problem of defining a fixpoint semantics for Constraint Handling Rules (CHR) that captures the behavior of both simplification and propagation rules in a sound and complete way with respect to their declarative semantics. Firstly, we show that the logical reading of states with respect to a set of simplification rules can be characterized by a least fixpoint over the transition system generated by the abstract operational semantics of CHR. Similarly, we demonstrate that the logical reading of states with respect to a set of propagation rules can be characterized by a greatest fixpoint, providing along the way the first completeness result for CPR. Then, in order to take advantage of both types of rules without losing fixpoint characterization, we present an operational semantics with persistent constraints closely related to the one recently proposed by Betz, Raiser, and Frühwirth We finally establish that this semantics can be characterized by two nested fixpoints, and we show the resulting language is an elegant framework to program using coinductive reasoning.

Contents

1 Introduction

Owing to its origins in the tradition of Constraint Logic Programming (CLP) (Jaffar and Lassez 1987), Constraint Handling Rules (CHR) (Frühwirth 1998) feature declarative semantics through direct interpretation in classical logic. However, no attempt to provide fixpoint semantics to the whole language, sound and complete with respect to this declarative semantics has succeeded so far. This is particularly surprising considering that the fixpoint semantics is an important foundation of the declarative semantics of CLP. It is perhaps because CHR is the combination of two inherently distinct kinds of rules that this formulation is not so simple. On the one hand, the so-called Constraint Simplification Rules (CSR) replace constraints by simpler ones while preserving their meaning. On the other hand, the so-called Constraint Propagation Rules (CPR) add redundant constraints in a monotonic way. Even though in the declarative semantics the two notions merge, one of the main interests of the language comes from the explicit distinction between the two. Indeed, it is well known that propagation rules are useful in practice but have to be managed in a different way from simplification rules to avoid trivial non-termination. (See for instance explanations by Frühwirth (2009) or Betz, Raiser, and Frühwirth (2010).)

Soundness of the operational semantics of CSR (i.e. to each derivation corresponds a deduction) have been proved by Frühwirth (2009), while completeness (i.e. to each deduction corresponds a derivation) has been tackled by Abdennadher, Frühwirth, and Meuss (1999). However, it is worth noticing that the completeness result is limited to terminating programs. On the other hand, the accuracy of CPR with respect to its classical logic semantics is only given through its naive translation into CSR. Since any set of propagation rules trivially loops when seen as simplification rules, the completeness result does not apply to CPR.

It is well known that termination captures least fixpoint (l.f.p.) of states of a transition system. Quite naturally, non-termination captures the greatest fixpoint (g.f.p.). Starting from this observation, we show in this paper that if they are considered independently, CSR and CPR can be characterized by a l.f.p. (or inductive) and g.f.p. (or coinductive) semantics respectively. By doing so, we give the first completeness result for CPR. Then, in order to take advantage of both types of rules without losing fixpoint characterization, we present an operational semantics ω_h similar to the one recently proposed by Betz, Raiser, and Frühwirth (2010). Subsequently we demonstrate that this new semantics can be characterized by two nested fixpoints. We show as well that it can be implemented in a simple manner and yields an elegant framework for programming with coinductive reasoning on infinite (or non-well founded) objects (Barwise and Moss 1996).

The remainder of this paper is structured as follows: Section 2 states the syntax of CHR and summarizes several semantics. In Section 3, after summarizing some classical notations and results about fixpoints, we present two fixpoint semantics for CHR. We show these semantics, which built over the transition system induced by the abstract operational semantics of CHR, offer a characterization of logical reading of queries with respect to CSR and CPR, respectively. In Section 4, we define semantics with persistent constraints related to the one recently introduced by Betz, Raiser, and Frühwirth (2010). We prove this new operational semantics can be characterized by a l.f.p. nested within a g.f.p., and give an implementation via a source-to-source transformation. Finally, in Section 5, we illustrate the power of the language before concluding in Section 6. Proofs omitted in the body of the paper can be found in Appendix C.

2 Preliminaries on CHR

In this section, we introduce the syntax, the declarative semantics and two different operational semantics for CHR. In the next sections, both operational semantics will be used, the former as theoretical foundation for our different fixpoint semantics, and the latter as a target to implementation purposes.

2.1 Syntax

The formalization of CHR assumes a language of *built-in constraints* containing the equality $=$, $\bot$, and $\top$ over some theory $\mathcal{C}$ and defines *user-defined constraints* using a different set of predicate symbols. In the following, we will denote variables by upper case letters, X, Y, Z, ..., and (user-defined or built-in) constraints by lowercase letters $c, d, e \ldots$ By a slight abuse of notation, we will confuse conjunction and multiset of constraints, forget braces around multisets and use comma for multiset union. We note $\mathrm{fv}(\phi)$ the set of free variables of any formula ϕ. The notation $\exists_{\text{-}\bar{X}}\phi$ denotes the existential closure of ϕ with the exception of variables in $\bar{X}$, which remain free. In this paper, we require the non-logical axioms of $\mathcal{C}$ to be *coherent formula* (i.e. formulas of the form $\forall(\mathbb{C} \rightarrow \exists\bar{Z}.\mathbb{D})$, where both $\mathbb{C}$ and $\mathbb{D}$ stand for possibly empty conjunctions of built-in constraints). Constraint theories verifying such requirements correspond to Saraswat's *simple constraints systems* (1991).

A *CHR program* is a finite set of eponymous rules of the form:

$$r \mathbin{@} \mathbb{K}\backslash\mathbb{H} \iff \mathbb{G} \mid \mathbb{C}, \mathbb{B}$$

where $\mathbb{K}$ (the *kept head*), $\mathbb{H}$ (the *removed head*) are multisets of user-defined constraints respectively, $\mathbb{G}$ (the *guard*) is a conjunction of built-in constraints, $\mathbb{C}$ is a conjunction of built-in constraints, $\mathbb{B}$ is a multiset of user-defined constraints and, r (the *rule name*) is an arbitrary identifier assumed unique in the program. Rules, where both heads are empty, are prohibited. Empty kept-head can be omitted together with the symbol $\backslash$. The *local variables* of rule are the variables occurring in the guard and in the body but not in the head that is $\mathrm{lv}(r) = \mathrm{fv}(\mathbb{G}, \mathbb{C}, \mathbb{B}) \setminus \mathrm{fv}(\mathbb{K}, \mathbb{H})$. CHR rules are divided into two classes: *simplification rules* if the removed head is non empty and *propagation rules* otherwise. Propagation rules can be written using the alternative syntax: $r \mathbin{@} \mathbb{K} \Longrightarrow \mathbb{G} \mid \mathbb{C}, \mathbb{B}$.

A *CHR state* is a tuple $\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle$, where $\mathbb{C}$ (the *CHR store*) is a multiset of CHR constraints, $\mathbb{E}$ (the *built-in store*) is a conjunction of built-in constraints, and X (the *global variables*) is a set of variables. In the following, Σ will denote the set of states and Σ_b the set of *answers* (i.e. states of the form $\langle \emptyset; \mathbb{C}; \bar{X} \rangle$). A state is *consistent* if its built-in store is satisfiable within $\mathcal{C}$ (i.e. there exists an interpretation of $\mathcal{C}$ which is a model of $\exists\mathbb{C}$), *inconsistent* otherwise.

2.2 Declarative semantics

We state now the declarative semantics of CHR. The *logical reading* of a rule and a state is defined as follows:

- Rule: $(\mathbb{K}\backslash\mathbb{H} \iff \mathbb{G} \mid \mathbb{C}, \mathbb{B})^\dagger = \forall\big((\mathbb{K} \wedge \mathbb{G}) \rightarrow \big(\mathbb{H} \leftrightarrow \exists_{\text{-}\mathrm{fv}(\mathbb{K},\mathbb{H})}(\mathbb{G} \wedge \mathbb{C} \wedge \mathbb{B})\big)\big)$;
- State: $\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle^\dagger = \exists_{\text{-}\bar{X}}(\mathbb{C} \wedge \mathbb{E})$.

For the sake of simplicity, in proofs within this paper, we will use the alternative but equivalent reading $\forall\big((\mathbb{K}\wedge\mathbb{H}\wedge\mathbb{G})\rightarrow\exists_{\text{-fv}(\mathbb{K},\mathbb{H})}(\mathbb{G}\wedge\mathbb{C}\wedge\mathbb{B})\big)\wedge\forall((\mathbb{K}\wedge\mathbb{G}\wedge\mathbb{C}\wedge\mathbb{B})\rightarrow\mathbb{H})$ for the rule $(\mathbb{K}\backslash\mathbb{H}\iff\mathbb{G}\mid\mathbb{C},\mathbb{B})$. $\mathcal{CP}$, the *logical reading* of a program $\mathcal{P}$ within a constraint theory $\mathcal{C}$ is the conjunction of the logical readings of the rules of $\mathcal{P}$ with the constraint theory $\mathcal{C}$.

2.3 Equivalence-base operational semantics

Here, we recall the *equivalence-based* operational semantics ω_e of Raiser, Betz, and Frühwirth (2009). It is similar to the *very abstract* semantics ω_a of Frühwirth (2009), the most general operational semantics of CHR. We prefer the former because it includes an explicit notion of equivalence, that will simplify many formulations. Because this is the most abstract operational semantics we consider in this paper, we will refer to it as the *abstract (operational) semantics*. For the sake of generality, we present it in a parametric form, according to some sound equivalence relation.

We will say that an equivalence relation $\equiv_i$ is *(logically) sound* if two states equivalent with respect to $\equiv_i$ have logically equivalent readings in the theory $\mathcal{C}$. The equivalence class of some state σ by $\equiv_i$ will be noted $[\![\sigma]\!]_i$. For a given program $\mathcal{P}$ and a sound equivalence $\equiv_i$, the *$\equiv_i$-transition relation*, noted $\xrightarrow{\mathcal{P}}_i$, is the least relation satisfying the following rules:

$$\frac{(r\ @\ \mathbb{K}\backslash\mathbb{H}\iff\mathbb{G}|\mathbb{C},\mathbb{B})\in\mathcal{P}\rho\quad \text{lv}(r)\cap\bar{X}=\emptyset}{\langle\mathbb{K},\mathbb{H},\mathbb{D};\mathbb{G}\wedge\mathbb{E};\bar{X}\rangle\xrightarrow{\mathcal{P}}_i\langle\mathbb{C},\mathbb{K},\mathbb{D};\mathbb{B}\wedge\mathbb{G}\wedge\mathbb{E};\bar{X}\rangle}\qquad\frac{\sigma_1\equiv_i\sigma_1'\quad\sigma_1'\xrightarrow{\mathcal{P}}_i\sigma_2'\quad\sigma_2'\equiv_i\sigma_2}{\sigma_1\xrightarrow{\mathcal{P}}_i\sigma_2}$$

where ρ is a renaming. If such transition is possible with $\mathbb{H}=\emptyset$, we will say that the tuple $r@\langle\mathbb{K};\mathbb{G}\wedge\mathbb{E};\bar{X}\rangle$ is a *propagation redex* for the state σ. The transitive closure of the relation $(\xrightarrow{\mathcal{P}}_i\cup\equiv_i)$ is denoted by $\xrightarrow{\mathcal{P}}_i^*$. A *$\equiv_i$-derivation* is a finite or infinite sequence of the form $\sigma_1\xrightarrow{\mathcal{P}}_i\ldots\xrightarrow{\mathcal{P}}_i\sigma_n\xrightarrow{\mathcal{P}}_i\ldots$ A $\equiv_i$-derivation is *consistent* if so are the states constituting it. A $\equiv_i$-transition is *confluent* if whenever $\sigma\xrightarrow{\mathcal{P}}_i^*\sigma_1$ and $\sigma\xrightarrow{\mathcal{P}}_i^*\sigma_2$ hold, there exists a state σ' such that $\sigma_1\xrightarrow{\mathcal{P}}_i^*\sigma'$ and $\sigma_2\xrightarrow{\mathcal{P}}_i^*\sigma'$ hold as well.

The *abstract equivalence* is the least equivalence $\equiv_a$ defined over Σ verifying:

1. $\langle\mathbb{C};Y=t\wedge\mathbb{D};\bar{X}\rangle\equiv_a\langle\mathbb{G}[y\backslash t];Y=t\wedge\mathbb{D};\bar{X}\rangle$
2. $\langle\mathbb{C};\bot;\bar{X}\rangle\equiv_a\langle\mathbb{D};\bot;\bar{X}\rangle$
3. $\langle\mathbb{C};\mathbb{D};\bar{X}\rangle\equiv_a\langle\mathbb{C};\mathbb{E};\bar{X}\rangle$ if $\mathcal{C}\vDash\exists_{\text{-fv}(\mathbb{C},\bar{X})}(\mathbb{D})\leftrightarrow\exists_{\text{-fv}(\mathbb{C},\bar{X})}(\mathbb{E})$
4. $\langle\mathbb{C};\mathbb{E};\bar{X}\rangle\equiv_a\langle\mathbb{C};\mathbb{E};\{Y\}\cup\bar{X}\rangle$ where $Y\notin\text{fv}(\mathbb{C},\mathbb{E})$.

Note that this equivalence is logically sound (Refer to case 1 of Lemma 3 in (Raiser, Betz, and Frühwirth 2009)). For a given program $\mathcal{P}$, the *abstract transition systems* is defined as the tuple $(\Sigma,\xrightarrow{\mathcal{P}}_a)$.

2.4 Concrete operational semantics

This section presents the operational semantics ω_p of de Koninck, Schrijvers, and Demoen (2007). In this framework rules are annotated with explicit priorities that reduce the non-determinism in the choice of the rule to apply. As initially proposed by Abdennadher (1997),

this semantics includes a partial control that prevents the trivial looping of propagation rules by restricting their firing only once on same instances. By opposition to the abstract semantics we have just defined, we will call it *concrete (operational) semantics*.

An *identified constraint* is a pair noted $c\#i$, associating a CHR constraint c with an integer i. For any identified constraints, we define the functions $\mathrm{chr}(c\#i) = c$ and $\mathrm{id}(c\#i) = i$, and extend them to sequences and sets of identified constraints. A *token* is a tuple $(r, \bar{\imath})$, where r is a rule name and $\bar{\imath}$ is a sequence of integers. A *concrete CHR state* is a tuple of the form $\langle\!\langle \mathbb{C}; \mathbb{D}; \mathbb{E}; T \rangle\!\rangle_n^{\bar{X}}$ where $\mathbb{C}$ is a multiset of CHR and built-in constraints, $\mathbb{D}$ is a multiset of identified constraints, $\mathbb{E}$ is a conjunction of built-in constraints, T is a set of tokens and n is an integer. We assume moreover that the identifier of each identified constraints in the CHR store is unique and smaller than n. For any program $\mathcal{P}$, the *concrete transition* relation, $\xrightarrow{\mathcal{P}}_c$, is defined as following:

Solve $\langle\!\langle c, \mathbb{C}; \mathbb{D}; \mathbb{E}; T \rangle\!\rangle_n^{\bar{X}} \xrightarrow{\mathcal{P}}_c \langle\!\langle \mathbb{C}; \mathbb{D}; c \wedge \mathbb{E}; T \rangle\!\rangle_n^{\bar{X}}$
if c is a built-in constraint and $\mathcal{C} \vDash \forall((c \wedge \mathbb{B}) \leftrightarrow \mathbb{B}')$.

Introduce $\langle\!\langle c, \mathbb{C}; \mathbb{D}; \mathbb{E}; T \rangle\!\rangle_n^{\bar{X}} \xrightarrow{\mathcal{P}}_c \langle\!\langle \mathbb{C}; c\#n, \mathbb{D}; \mathbb{E}; T \rangle\!\rangle_{n+1}^{\bar{X}}$
if c is a CHR constraint.

Apply $\langle\!\langle \emptyset; \mathbb{K}, \mathbb{H}, \mathbb{E}; \mathbb{C}; T \rangle\!\rangle_n^{\bar{X}} \xrightarrow{\mathcal{P}}_c \langle\!\langle \mathbb{B}, \mathbb{G}; \mathbb{K}, \mathbb{D}; \mathbb{C} \wedge \theta; t \cup T \rangle\!\rangle_n^{\bar{X}}$
if $(p :: r \mathbin{@} \mathbb{K}'\backslash\mathbb{H}' \Leftrightarrow \mathbb{G} \mid \mathbb{B})$ is a rule in $\mathcal{P}$ of priority p renamed with fresh variables, θ is a substitution such that $\mathrm{chr}(\mathbb{K}) = \mathbb{K}'\theta$, $\mathrm{chr}(\mathbb{H}) = \mathbb{H}'\theta$, $t = (r, \mathrm{id}(\mathbb{K}, \mathbb{H}))$, $t \notin T$, $\mathbb{C}$ is satisfiable within $\mathcal{C}$, and $\mathcal{C} \vDash \forall(\mathbb{C} \rightarrow \exists(\theta \wedge G))$. Furthermore, no rule of priority bigger than p exists for which the above conditions hold.

3 Transition system semantics for pure CSR and CPR

In this section, we propose a fixpoint semantics for both CSR and CPR programs. We call it transition system semantics because it is defined as a fixpoint over the abstract transition system, built in a way similar to μ-calculus formula (Clarke, Grumberg, and Peled 2000).

Before formally introducing the semantics, we recall some standard notations and results about fixpoints[1].

3.1 Fixpoints

Let assume some arbitrary complete lattice $(\mathcal{L}, \supset, \cap, \cup, \top, \bot)$. An element $\mathcal{X} \in \mathcal{L}$ is an *upper bound* for a set $\mathcal{S} \in 2^{\mathcal{L}}$ if $\mathcal{X} \supset Y$ for all $Y \in \mathcal{L}$. $\mathcal{X}$ is a *least upper bound* for $\mathcal{S}$, noted $\mathrm{lub}(\mathcal{S})$ if $\mathcal{X}$ is an upper bound for $\mathcal{S}$ and $\mathcal{Y} \supset X$ for any upper bound $\mathcal{Y}$ of $\mathcal{S}$. A function $f : \mathcal{L} \rightarrow \mathcal{L}$ is *monotonic* if $f(\mathcal{X}) \supset f(\mathcal{Y})$ whenever $\mathcal{X} \supset \mathcal{Y}$. A function f is *continuous* if for any set $\mathcal{S}$, $f(\mathrm{lub}(\mathcal{S})) = \mathrm{lub}(f(\mathcal{S}))$. The *upward (ordinal) power* of a function $f : \mathcal{L} \rightarrow \mathcal{L}$ is defined by the transfinite induction : $f \uparrow 0 = \bot$, $f \uparrow \alpha = f(f \uparrow (\alpha - 1))$ if α is a successor ordinal, and $f \uparrow \alpha = \mathrm{lub}(\{f \uparrow \beta \mid \beta < \alpha\})$ if α is a limit ordinal. An element $\mathcal{X} \in \mathcal{L}$ is a *fixpoint* for $f : \mathcal{L} \rightarrow \mathcal{L}$ if $f(\mathcal{X}) = \mathcal{X}$. $\mathcal{X}$ is a *least fixpoint* for f if it is a fixpoint and $\mathcal{Y} \supset \mathcal{X}$ whenever $\mathcal{Y}$ is a fixpoint for f. For any function f and any set $\mathcal{S}$, *lower bounds for* $\mathcal{S}$, *greatest lower bounds for* $\mathcal{S}$ ($\mathrm{glb}(\mathcal{S})$), the *co-continuity* of f, the *backward (ordinal) power* of f ($f \downarrow \alpha$) and *greatest fixpoints* of f are defined by duality.

[1] For more details about fixpoints, one can refer, for example, to Lloyd's Book (1987).

The following classical results ensure respectively that the l.f.p. and the g.f.p. of a monotonic function exist and can be computed by the ordinal closures limited to ω if the function is furthermore (co)continuous.

Theorem 3.1 (Knaster–Tarski)**.** *Let f be a monotonic function on sets. Then F has a l.f.p. $\mu X.f(X)$ and a greatest fixpoint $\nu X.f(X)$.*

Proposition 3.2. *If $f : \mathcal{L} \to \mathcal{L}$ is a continuous (resp. co-continuous) function then $\mu X.f(X) = f \uparrow \omega$ (resp. $\nu X.f(X) = f \downarrow \omega$).*

3.2 Inductive semantics for CSR

In this section we give a first fixpoint semantics limited to CSR. We call it inductive, since it is defined as a least fixpoint.

Definition 3.3 (Inductive transition system semantics for CSR)**.** *For a given program $\mathcal{P}$ and an given sound equivalence relation $\equiv_i$, the existential immediate cause operator $\langle \mathcal{P} \rangle_i : 2^\Sigma \to 2^\Sigma$ is defined as:*

$$\langle \mathcal{P} \rangle_i (\mathcal{X}) = \{\sigma \in \Sigma \mid \text{there exists } \sigma' \in \Sigma \text{ such that } \sigma \xrightarrow{\mathcal{P}}_i \sigma' \text{ and } \sigma' \in \mathcal{X}\}$$

The inductive (transition system) semantics of a CSR program $\mathcal{Q}$ is the set:

$$\mathcal{F}^{\mathcal{C}}(\mathcal{Q}) = \mu\mathcal{X}.(\langle \mathcal{Q} \rangle_a (\mathcal{X}) \cup (\Sigma_b \setminus [\![\langle \emptyset; \bot; \emptyset \rangle]\!]_a))$$

The existential immediate cause operator being clearly monotonic, Tarski's theorem ensures the inductive semantics of a CSR program is well defined. For a given program $\mathcal{P}$, $\mathcal{F}^{\mathcal{C}}(\mathcal{P})$ is exactly the set of states that can be rewritten by $\mathcal{P}$ to a consistent answer. Remark that since answers cannot be rewritten by any program, any state in $\mathcal{F}^{\mathcal{C}}(\mathcal{P})$ has at least one terminating derivation.

Example 3.4. *Consider the program $\mathcal{P}_1$ consisting in the two following rules:*

$$a \iff \top \qquad\qquad b \iff b$$

We can pick the following sets of consistent states: $\mathcal{X}_1$ of the form $\langle a^i; \mathbb{C}; \bar{X} \rangle$, $\mathcal{X}_2$ of the form $\langle a^i, b; \mathbb{C}; \bar{X} \rangle$, and $\mathcal{X}_3$ of the form $\langle a^i, b^j; \mathbb{C}; \bar{X} \rangle$, where c^n denotes n copies of the constraint c. These three are fixpoints of $\lambda\mathcal{X}.(\langle \mathcal{Q} \rangle_a (\mathcal{X}) \cup (\Sigma_b \setminus [\![\langle \emptyset; \bot; \emptyset \rangle]\!]_a))$, but only $\mathcal{X}_1$ is the least fixpoint.

We next present a theorem that uses fixpoints semantics to reformulate results on soundness and completeness of CSR (Frühwirth 1998; Abdennadher, Frühwirth, and Meuss 1999). It says that if a state has at least one answer then this state is in the inductive semantics of a confluent CSR program if and only if its logical reading is satisfiable within the theory $\mathcal{CP}$. Notice that because in the context of this paper, we do not require $\mathcal{C}$ to be ground complete, we have to content ourselves with satisfiability instead of validity. Nonetheless, we will see in Section 5, that satisfiability is particularly useful to express coinductive definitions such as bisimulation.

Theorem 3.5. *Let $\mathcal{P}$ be program such that $\xrightarrow{\mathcal{P}}_a$ is confluent. Let $\langle \mathbb{D}; \mathbb{E}; \bar{X} \rangle$ be a state having at least one answer. We have*

$$\langle \mathbb{D}; \mathbb{E}; \bar{X} \rangle \in \mathcal{F}^{\mathcal{C}}(\mathcal{P}) \text{ if and only if } \exists(\mathbb{D} \wedge \mathbb{E}) \text{ is satisfiable within } \mathcal{CP}.$$

Proof. The theorem is corollary of the more general Theorem 4.10. □

Our inductive semantics for CSR has strong connections with the fixpoint semantics of Gabbrielli and Meo (2009). In contrast to ours, this semantics focuses on input/output behaviour and is not formally related to logical semantics, although it is constructed in similar way as a l.f.p. over the abstract transition system. However, because it does not distinguish propagation from simplification rules, this semantics cannot characterize reasonable programs using propagations. Indeed, it has been later extended to handle propagation rules by adding into the states an explicit token store *à la* Abdennadher (1997) in order to remember the propagation history (Gabbrielli, Meo, and Tacchella 2008). Nonetheless, such an extension leads to a quite complicated model which is moreover incomplete with respect to logical semantics.

Gabbrielli and Levi (1992) have defined a fixpoint semantics for non-deterministic concurrent constraint programming in the typical logic programming style. Nevertheless this semantics is based on infinite unfoldings which seems difficult to transpose to standard CHR, which allows multiple head and forbid disjunctions. It seems that any other formulations of fixpoint semantics for concurrent constraint programming (see for example (de Boer and Palamidessi 1991; Saraswat, Rinard, and Panangaden 1991; Nyström and Jonsson 1993)) are more removed from typical logic programming style and are, in any case, limited to single head rules.

3.3 Coinductive semantics for CPR

We continue by giving a similar characterization for CPR. This semantics is defined by the g.f.p. of a universal version of the cause operator presented in Definition 3.3. Hence, we call it coinductive semantics.

Definition 3.6 (Coinductive transition system semantics for CPR)**.** *For a given program $\mathcal{P}$ and an given sound equivalence relation $\equiv_i$, the universal (immediate) cause operator $[\mathcal{P}]_i : 2^\Sigma \to 2^\Sigma$ is defined as:*

$$[\mathcal{P}]_i(\mathcal{X}) = \{\sigma \in \Sigma \mid \textit{for all } \sigma' \in \Sigma, \sigma \xrightarrow{\mathcal{P}}_i \sigma' \textit{ implies } \sigma' \in \mathcal{X}\}$$

The coinductive (transition system) semantics of a CPR program $\mathcal{Q}$ is the set:

$$\mathcal{F}^{\mathcal{C}}_{co}(\mathcal{Q}) = \nu\mathcal{X}.([Q]_a(\mathcal{X}) \cap (\Sigma \setminus [\![\langle \emptyset; \bot; \emptyset\rangle]\!]_a))$$

As it is the case for $\langle\mathcal{Q}\rangle_i$, $[\mathcal{Q}]_i$ is obviously monotonic. Our semantics is therefore well defined. Notice, that $\mathcal{F}^{\mathcal{C}}_{\text{co}}(\mathcal{Q})$ is precisely the set of states that cannot be rewritten to an inconsistent states. As illustrated by the following example, states belonging to $\mathcal{F}^{\mathcal{C}}_{\text{co}}(\mathcal{Q})$ have in general only non-terminating derivations with respect to the abstract operational semantics.

Example 3.7. *Let $\mathcal{C}$ be a constraints theory providing usual constraints over integers. Consider the program $\mathcal{P}_2$ consisting in the two following rules:*

$$q(X) \Longrightarrow q(X+1) \qquad\qquad q(0) \Longrightarrow \bot$$

The greatest fixpoint of $\lambda\mathcal{X}.([Q]_a(\mathcal{X}) \cap (\Sigma \setminus [\![\langle \emptyset; \bot; \emptyset\rangle]\!]_a))$ is the set of consistent states that do not contain a CHR constraint $p(X)$ where X is negative or null. Note the empty set is also a fixpoint but not the greatest. Note that states such as $\langle q(1); \top; \emptyset\rangle$, which are in the greatest fixpoint, have only infinite derivations.

We give next a theorem that states the accuracy of the coinductive semantics with respect to the logical reading of CPR. Remark that for the completeness direction, we have to ensure that a sufficient number of constraints is provided for launching each rule of the derivation. To state the theorem we assume the following notation ($n \cdot \mathbb{B}$ stand for the scalar product of the multiset $\mathbb{B}$ by n):

$$\mathcal{P}^n = \{(r \ @ \ \mathbb{K} \Longrightarrow \mathbb{G} \mid \mathbb{C}, n \cdot \mathbb{B}) \mid (r \ @ \ \mathbb{K} \Longrightarrow \mathbb{G} \mid \mathbb{C}, \mathbb{B}) \in \mathcal{P}\}$$

Theorem 3.8 (Soundness and completeness of coinductive semantics for CPR)**.** *Let $\mathcal{P}$ be a CPR program and n be some integer greater than the maximal number of constraints occurring in the head of any rule of $\mathcal{P}$. We have:*

$$\left\langle n \cdot \mathbb{E}; \mathbb{C}; \bar{X} \right\rangle \in \mathcal{F}^{\mathcal{C}}_{co}(\mathcal{P}^n) \text{ if and only if } (\mathbb{E} \wedge \mathbb{C}) \text{ is satisfiable within } \mathcal{CP}.$$

Proof. The theorem is corollary of the more general Theorem 4.10 together with Proposition 4.2. □

The coinductive semantics for CPR, has strong similarities with the fixpoint semantics of CLP (Jaffar and Lassez 1987). Both are defined by fixpoint of somehow dual operators and fully abstract the logical meaning of programs. Nonetheless the coinductive semantics of CPR is not compositional. That is not a particular drawback of our semantics, since the logical semantics we characterize is neither compositional. Indeed, if the logical readings of two states are independently consistent, then one cannot ensure that so is their conjunction. It should be noticed that this non-compositionality prevents the immediate cause operators to be defined over the $\mathcal{C}$-base (i.e. the cross product of the set of CHR constraints and the set of conjunctions of built-in constraints) as it is done for CLP, and requires a definition over set of states.

4 Transition system semantics for CHR with persistent constraints

In this section, we aim at obtaining a fixpoint semantics for the whole language. Nonetheless, one has to notice that the completeness result for CSR needs, among other things, the termination of $\xrightarrow{\mathcal{P}}_a$, while the equivalent result for CPR is based on the monotonic evolution of the constraints store along derivations. Hence combining naively CSR and CPR will break both properties, leading consequently to an incomplete model. In order to provide an accurate fixpoint semantics to programs combining both kinds of rules (meanwhile removing unsatisfactory scalar product in the wording of CPR completeness), we introduce a notion of *persistent constraints*, following ideas of Betz, Raiser, and Frühwirth (2010) for their semantics $\omega_!$. Persistent constraints are special CHR constraints acting as classical logic statements (i.e. they are not consumable and their multiplicity does not matter). Since they act as linear logic statements (i.e. they are consumable and their multiplicity matters), usual CHR constraints are called *linear*. Because it combines persistent and linear constraints in a slightly less transparent way that $\omega_!$, we call this semantics *hybrid*, and note it ω_h.

4.1 Hybrid operational semantics ω_h

On the contrary of $\omega_!$, the kind of a constraint (linear or persistent) in ω_h, is not dynamically determined according the type of rules from which it was produced, but statically fixed. Hence,

we will assume the set of CHR constraints symbols is divided in two: the *linear* symbols and the *persistent* symbols. Naturally, CHR constraints built from the linear (resp. persistent) symbols are called linear (resp. persistent) constraints. A *hybrid rule* is a CHR rule where the kept head contains only persistent constraints and the removed head contains only linear constraints. We will denote by Σ_p, the set of *purely persistent* states (i.e. states of the form $\langle \mathbb{P}; \mathbb{C}; \bar{X} \rangle$ where $\mathbb{P}$ is a set of persistent constraints). $\mathcal{P}^s$ will refer to the set of simplification rules of a hybrid program $\mathcal{P}$, respectively.

The hybrid programs are programs where propagation rules "commute" with simplification rules in the sense of abstract rewriting systems (Terese 2003)(cf. Lemma C.3). In other words, derivations can be permuted so that simplification rules are fired first, and propagation rules fire only when simplification rule firings are exhausted. Indeed, the syntactical restriction prevents the propagation head constraints to be consumed by simplification rules, hence once a propagation rule is applicable, then it will be so for ever. Of course, number of CHR programs such as the classic "less-or-equal" (refer to Section 2.4.2 in the Frühwirth's book) do not respect the hybrid syntax and therefore cannot be run in our framework. Nonetheless, what we loose with this restriction, we compensate by pioneering a logically complete approach to solve the problem of trivial non-termination (see next Theorems 4.15 and 4.16).

The hybrid semantics is expressed as a particular instance of the equivalence based semantics presented in Section 2.3. It uses the abstract state equivalence extended by a *contraction* rule enforcing the impotency of persistent constraints.

Definition 4.1 (Hybrid operational transition)**.** *The hybrid equivalence is the smallest relation, $\equiv_h$, over states containing $\equiv_a$ satisfying the following rule:*

$$\langle c, c, \mathbb{C}; \mathbb{D}; \bar{X} \rangle \equiv_h \langle c, \mathbb{C}; \mathbb{D}; \bar{X} \rangle \text{ if } c \text{ is a persistent constraints}$$

The hybrid transition system is defined as the tuple $(\Sigma, \xrightarrow{\mathcal{P}}_h)$.

Here we relate the hybrid operational semantics with the abstract one. The following theorem states that a hybrid state can be derived if and only if a corresponding derivation in the abstract semantics exists. Notice that for the completeness direction, we have to ensure that the persistent constraints are provide in a sufficient quantity to engage the derivation. To state the theorem we extend to hybrid programs a notation introduced in previous section for CPR programs:

$$\mathcal{P}^n = \{(r \ @ \ \mathbb{K} \backslash \mathbb{H} \iff \mathbb{G} \mid \mathbb{B}, \mathbb{L}, n \cdot \mathbb{P}) \mid (r \ @ \ \mathbb{K} \backslash \mathbb{H} \iff \mathbb{G} \mid \mathbb{B}, \mathbb{L}, \mathbb{P}) \in \mathcal{P}\}$$

Proposition 4.2. *Let $\mathcal{P}$ be a hybrid program and n be an integer bigger or equal than the maximal number of persistent constraints occurring in the head of a rule of $\mathcal{P}$.*

(i). If $\langle \mathbb{L}, \mathbb{P}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}}_h \langle \mathbb{L}', \mathbb{P}'; \mathbb{C}'; \bar{X} \rangle$, then there is a transition $\langle \mathbb{L}, n \cdot \mathbb{P}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}^n}_a \langle \mathbb{L}'', n \cdot \mathbb{P}''; \mathbb{C}''; \bar{X} \rangle$ such that $\langle \mathbb{L}', \mathbb{P}'; \mathbb{C}'; \bar{X} \rangle \equiv_h \langle \mathbb{L}'', \mathbb{P}''; \mathbb{C}''; \bar{X} \rangle$

(ii). If $\langle \mathbb{L}, n \cdot \mathbb{P}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}^n}_a \langle \mathbb{D}'; \mathbb{C}'; \bar{X} \rangle$, then there exists a transition $\langle \mathbb{L}, \mathbb{P}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}}_h \langle \mathbb{L}'', \mathbb{P}''; \mathbb{C}''; \bar{X} \rangle$ such that $\langle \mathbb{D}'; \mathbb{C}'; \bar{X} \rangle \equiv_a \langle \mathbb{L}'', \mathbb{P}''; \mathbb{C}''; \bar{X} \rangle$.

We enunciate now some useful properties about hybrid transition. The following monotonicity property means that adding (CHR or built) constraints to a state cannot previous the applicability of a rule. This classical property (cf. Lemma 4.1 in the Frühwirth's book) is here adapted to hybrid semantics.

Proposition 4.3 (Transition monotonicity). *Let* $\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle$ *and* $\langle \mathbb{C}'; \mathbb{E}'; X' \rangle$ *be two states. Let* $\mathbb{D}$*, be a multiset of CHR constraints,* $\mathbb{F}$ *be a multiset of built-in constraints, and* Y *be a set of variables such that* $\mathrm{fv}(\mathbb{D}, \mathbb{F}) \cap \mathrm{fv}(\mathbb{C}, \mathbb{E}, \mathbb{C}', \mathbb{E}') \subset (X \cup X')$*. If* $\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle \xrightarrow{\mathcal{P}}{}^{*}_{h} \langle \mathbb{C}'; \mathbb{E}'; X' \rangle$ *is a valid derivation, so is* $\langle \mathbb{C}, \mathbb{D}; \mathbb{E} \wedge \mathbb{F}; X \cap Y \rangle \xrightarrow{\mathcal{P}}{}^{*}_{h} \langle \mathbb{C}', \mathbb{D}; \mathbb{E}' \wedge \mathbb{F}; X' \cap Y \rangle$*.*

The following strong monotonicity states that the application of rules cannot consumes information about built-in store and persistent constraints.

Proposition 4.4 (Strong monotonicity of persistent and built-in stores). *Let* $\langle \mathbb{L}, \mathbb{P}; \mathbb{E}; \bar{X} \rangle$ *and* $\langle \mathbb{D}; \mathbb{F}; \bar{Y} \rangle$ *be two states, such that* $\mathbb{L}$ *is a multiset of persistent constraints and* $\mathbb{P}$ *is a multiset of persistent constraints. Let* ρ *be a renaming of the local variables of* $\langle \mathbb{D}; \mathbb{F}; \bar{Y} \rangle$ *with fresh variables. If* $\langle \mathbb{L}, \mathbb{P}; \mathbb{E}; \bar{X} \rangle \xrightarrow{\mathcal{P}}{}^{*}_{h} \langle \mathbb{D}; \mathbb{F}; \bar{Y} \rangle$ *is a valid, then so is* $\langle \mathbb{D}; \mathbb{F}; \bar{Y} \rangle \equiv_h \langle \mathbb{D}\rho, \mathbb{P}; \mathbb{F}\rho \wedge \mathbb{E}; \bar{X} \rangle$*.*

We transpose now the soundness of classical CHR transition to hybrid semantics (cf. Lemma 3.20 in Frühwirth's Book (2009)).

Proposition 4.5 (Soundness of hybrid derivations). *Let* $\mathcal{P}$ *be a hybrid proposition, and* σ *and* σ' *be two states. We have :*

$$\textit{If } \sigma \xrightarrow{\mathcal{P}}{}^{*}_{h} \sigma', \textit{ then } \mathcal{CP} \vDash \forall(\sigma^{\dagger} \leftrightarrow \sigma'^{\dagger}).$$

The proposition we give next says the composition of a confluent program together with a set of propagation rules is confluent as well.

Proposition 4.6. *Let* $\mathcal{P}$ *and* $\mathcal{Q}$ *be hybrid programs. If* $\mathcal{P}$ *is an arbitrary set of propagation rules and and* $\mathcal{Q}$ *is confluent then* $\xrightarrow{\mathcal{P} \cup Q}{}^{*}_{h}$ *is confluent.*

4.2 Hybrid transition system semantics

We present the transition system semantics for hybrid programs. This semantics is expressed by fixpoints over the hybrid transition system. It is built using the immediate cause operators we have defined in the previous section. Together with Theorem 3.1 and Proposition 3.2, the following lemma ensures that fixpoints of the cause operators exist and can be computed by ordinal power limited to ω.

Lemma 4.7. *For any programs* $\mathcal{P}$*,* $\langle \mathcal{P} \rangle_h$ *is monotonic and continuous, and* $[\mathcal{P}]_h$ *is monotonic and cocontinuous.*

Definition 4.8 (Hybrid transition system semantics). *The hybrid (transition system) semantics of a hybrid program* $\mathcal{P}$ *is defined as:*

$$\mathcal{F}^{\mathcal{C}}_{h}(\mathcal{P}) = \nu\mathcal{X}.([\mathcal{P}]_h(\mathcal{X}) \cap \mu\mathcal{Y}.(\langle \mathcal{P}^s \rangle_h(\mathcal{Y}) \cup (\Sigma_p \setminus [\![\langle \emptyset; \bot; \emptyset \rangle]\!]_h)))$$

The theorem we give next states soundness and completeness of the hybrid transition system semantics of confluent programs, provided the states respect a data-sufficiency property. In this paper, we do not address the problem of proving confluence of hybrid programs, but claim it can be tackled by extending straightforwardly the work of Abdennadher, Frühwirth, and Meuss (1999) or by adequately instanstiating the notion of abstract critical pair we proposed in a previous work (Haemmerlé and Fages 2007).

Definition 4.9 (Data-sufficient state)**.** *A hybrid state σ is data-sufficient with respect to a hybrid program $\mathcal{P}$ if any state σ' accessible form σ can be simplified (i.e. rewritten by $\mathcal{P}^s$) into a purely persistent sate. Formally the set of data-sufficient states can be defined as:*

$$DS(\mathcal{P}) = \nu\mathcal{X}.([\mathcal{P}]_h(\mathcal{X}) \cap \mu\mathcal{Y}.(\langle\mathcal{P}^s\rangle_h(\mathcal{Y}) \cup \Sigma_p))$$

This property ensures there is at least one computation where propagation rules are applied only once all linear constraints have been completely simplified. It is a natural extension of the eponymous property for CSR (Abdennadher, Frühwirth, and Meuss 1999).

Theorem 4.10 ((Soundness and completeness of hybrid transition system semantics)**.** *Let $\mathcal{P}$ be a hybrid program such that $\mathcal{P}^s$ is confluent. Let $\langle \mathbb{L}, \mathbb{P}; \mathbb{E}; \bar{X}\rangle$ be a data-sufficient state with respect to $\mathcal{P}$. We have:*

$$\langle \mathbb{L}, \mathbb{P}; \mathbb{E}; \bar{X}\rangle \in \mathcal{F}_h^{\mathcal{C}}(\mathcal{P}) \text{ if and only if } (\mathbb{L} \wedge \mathbb{P} \wedge \mathbb{E}) \text{ is satisfiable within } \mathcal{CP}$$

The proof of the theorem is decomposed into two lemmas.

Lemma 4.11. *Let $\mathcal{P}$ be a hybrid program such that $\mathcal{P}^s$ is confluent, let $\mathcal{Z}$ be a set of irreducible state and let σ a data-sufficient state.*

$$\sigma \in \mathcal{F}_h^{\mathcal{C}}(\mathcal{P}) \text{ if and only if } \sigma \overset{\mathcal{P}}{\not\rightarrow}{}_h^* \langle \emptyset; \bot; \emptyset\rangle$$

Lemma 4.12. *Let $\mathcal{P}$ be a hybrid program, such that $\mathcal{P}^s$ is confluent. Let σ be a data-sufficient state. If $\mathcal{CP} \vDash \exists\sigma^\dagger \rightarrow \bot$ then $\sigma \overset{\mathcal{P}}{\rightarrow}{}_h^* \langle \emptyset; \bot; \emptyset\rangle$.*

Proof of Theorem 4.10. We prove the contrapositive that is:

$$\mathcal{CP} \vDash \exists(\mathbb{L}, \mathbb{P} \wedge \mathbb{E}) \rightarrow \bot \text{ if and only if } \left\langle \mathbb{L}, \mathbb{P}; \mathbb{E}; \bar{X}\right\rangle \notin \mathcal{F}_{\mathrm{h}}^{\mathcal{C}}(\mathcal{P})$$

By Lemma 4.11 it is equivalent to show:

$$\mathcal{CP} \vDash \exists(\mathbb{L} \wedge \mathbb{P} \wedge \mathbb{E}) \rightarrow \bot \text{ if and only if } \left\langle \mathbb{L}, \mathbb{P}; \mathbb{E}; \bar{X}\right\rangle \overset{\mathcal{P}}{\rightarrow}{}_a^* \langle \emptyset; \bot; \emptyset\rangle$$

The "if" direction is corollary of soundness of hybrid transition (Proposition 4.5), while the "only if" direction is corollary of Lemma 4.12. □

The following proposition states that it is sufficient to consider only one fair derivation. This result is fundamental to allow the hybrid semantics to be efficiently implemented.

Definition 4.13 (Propagation fair derivation)**.** *A derivation $\sigma_0 \overset{\mathcal{P}}{\rightarrow}_h \sigma_1 \overset{\mathcal{P}}{\rightarrow}_h \ldots$ is propagation fair if for any state σ_i in the derivation and any propagation redex $r@\langle \mathbb{C}; \mathbb{E}; \bar{X}\rangle$ of σ_i, there exist states σ_j, σ_{j+1} s.t. the transition from σ_j to σ_{j+1} is a propagation application where the reduced redex is identical or stronger to $r@\langle \mathbb{C}; \mathbb{E}; \bar{X}\rangle$ (i.e. the reduced redex is of the form $r@\langle \mathbb{D}; \mathbb{F}; \bar{Y}\rangle$ with $\mathcal{C} \vDash \exists_{\bar{Y}}\mathbb{F} \rightarrow \exists_{\bar{X}}\mathbb{E}$).*

Proposition 4.14 (Soundness and completeness of propagation fair derivations)**.** *Let $\mathcal{P}$ be a hybrid program such that $\mathcal{P}^s$ is confluent and terminating. Let σ be a data-sufficient state. $\sigma \in \mathcal{F}_h^{\mathcal{C}}(\mathcal{P})$ holds if and only if there is a consistent propagation fair derivation starting from σ.*

Let $\mathcal{P}^\diamond$ the program $\mathcal{P}$ where simplification and propagation rules are given with the priorities 3 and 4 respectively. Apply the following steps:

step 1 . Apply the following rules until convergence :
If $(p :: c, d, \mathbb{K}\backslash\mathbb{H} \Longleftrightarrow \mathbb{G} \mid \mathbb{B}, \mathbb{L}, \mathbb{P}))$ is in $\mathcal{P}^\diamond$, with $\mathcal{C} \vDash \exists(c{=}d)$
then add the rule $(p :: c, \mathbb{K}\backslash\mathbb{H} \Longleftrightarrow c{=}d \wedge \mathbb{G} \mid \mathbb{B}, \mathbb{L}, \mathbb{P})$ to $\mathcal{P}^\diamond$

step 2 Substitute any rule $(p :: c_1, \ldots, c_m\backslash\mathbb{H} \Longleftrightarrow \mathbb{G} \mid \mathbb{B}, \mathbb{L}, d_1, \ldots d_n)$ by
$(p :: a(X_1, c_1), \ldots, a(X_m, c_m)\backslash\mathbb{H} \Longleftrightarrow \mathbb{G} \mid \mathbb{B}_l, \mathbb{L}, f(d_1), \ldots, f(d_n))$
where $x_1, \ldots x_m$ are pairwise distinct variables

step 3 Add to $\mathcal{P}^\diamond$ the rules:

$1 ::$	*stamp* @	$f(X), c_f(Y) \Longleftrightarrow f(Y, X), c_f(Y+1)$
$2 ::$	*set* @	$a(Y, X) \setminus a(Z, X) \Longleftrightarrow Y < Z \mid \top$
$5 ::$	*unfreeze* @	$f(Y, X), c_a(Y) \Longleftrightarrow a(Y, X), c_a(Y+1)$

Figure 1: Source-to-source translation for hybrid programs

4.3 Implementation of the hybrid semantics

We continue by addressing the question of implementing the hybrid semantics in a sound and complete way. For this purpose, we assume without loss of generality that the constraint symbols $f/1$, $f/2$, $a/2$, $c_f/1$, and $c_a/1$ are fresh with respect to the program $\mathcal{P}$ we consider. The implementation of a hybrid program $\mathcal{P}$ consists in a source-to-source translation $\mathcal{P}^\diamond$ intended to be executed in the concrete semantics ω_p. This transformation is given in detail in Figure 1. In order to be executed an hybrid state σ has to be translated into a concrete state $\sigma^\diamond$ as follows: if $\mathbb{L}$ and $(c_1, \ldots, c_n)$ are multisets of linear and persistent constraints respectively, then $\langle \mathbb{L}, d_1, \ldots d_n; \mathbb{D}; \mathbb{V} \rangle^\diamond = \langle\!\langle \mathbb{L}, f(d_1), \ldots, f(d_n), c_f(0), c_a(0); \emptyset; \mathbb{D}; \emptyset \rangle\!\rangle_0^{\mathbb{V}}$

Before going further, let us give some intuition about the behaviour of the translation. If a rule needs two occurrences of the same persistent constraint, step 1 will insert an equivalent rule which needs only one occurrence of the constraint. In the translation each persistent constraint can be applied in three different successive states: *fresh*, indicated by $f/1$, *frozen*, indicated by $f/2$, and *alive*, indicated by $a/2$. Step 2 ensures, on the one hand, that only alive constraints can be used to launch a rule, and on the other hand, that the persistent constraints of the right-hand side are inserted as fresh. Each frozen and alive constraint is associated to a time stamp indicating the order in which it has been asserted. The fresh constraints are time stamped and marked as frozen as soon as possible by *stamp*, the rule of highest priority (the constraint $c_f/1$ indicating the next available time stamp). Only if no other rule can be applied, the *unfreeze* rule turns the oldest frozen constraint into an alive constraint while preserving its time stamp (the constraint $c_a/1$ indicating the next constraint to be unfrozen). Rule *set* prevents trivial loops, by removing the youngest occurrence of two identical persistent constraints. From a proof point of view, the application of this last rule corresponds to the detection of a cycle in a coinduction proof, the persistent constraints representing coinduction hypothesises (Barwise and Moss 1996).

The two following theorems state that our implementation is sound and complete with respect to failure. Theorem 4.15 shows furthermore that the implementation we propose here is sound with respect to finite success. It is worth noting that it is hopeless to look for a complete implementation with respect to to success, since the problem to know if a data-sufficient state

is in the coinductive semantics is undecidable. The intuition behind this claim is that otherwise it would possible to solve the halting problem.

Theorem 4.15 (Soundness with respect to success and failure)**.** *Let $\mathcal{P}$ be a hybrid program such that $\mathcal{P}^s$ is confluent, and let $\left\langle \mathbb{C}; \mathbb{E}; \bar{X} \right\rangle^\diamond \rightarrow^*_{\mathcal{P}^\diamond} \left\langle\!\left\langle \emptyset; \mathbb{D}; \mathbb{F}; T \right\rangle\!\right\rangle_i^{\bar{Y}} \not\rightarrow_{\mathcal{P}^\diamond}$ be a terminating derivation. $\left\langle \mathbb{C}; \mathbb{E}; \bar{X} \right\rangle \in \mathcal{F}_h^{\mathcal{C}}(\mathcal{P})$ holds if and only if $\mathbb{F}$ is satisfiable within $\mathcal{C}$.*

Theorem 4.16 (Completeness with respect to failure)**.** *Let $\mathcal{P}$ be a hybrid program such that $\mathcal{P}^s$ is confluent and terminating. Let $\left\langle \mathbb{C}; \mathbb{E}; \bar{X} \right\rangle$ be a data-sufficient state. If $\left\langle \mathbb{C}; \mathbb{D}; \bar{X} \right\rangle \notin \mathcal{F}_h^{\mathcal{C}}(\mathcal{P})$, then any concrete derivation starting form $\left\langle \mathbb{C}; \mathbb{D}; \bar{X} \right\rangle^\diamond$ finitely fails.*

The implementation we have proposed has strong connections with the co-SLD, an implementation of the g.f.p. semantics of Logic Programming proposed by Simon et al. (2006). Both are based on a dynamic synthesizing of coinductive hypothesises and a detection of cycle in proof. But because it is limited to rational recursion, the co-SLD is logically incomplete with respect to both successes and failures (i.e. there are queries true and false with respect to the logical reading of a program that cause the interpreter to loop). That contrasts with CHR, where any coherent constraint system can be used without loosing logical completeness with respect to failures.

Simon et al. (2007), have latter generalizes the co-SLD in order to combining both induction with co-induction in LP. The way they achieve this marriage is somehow similar to the way the hybrid semantic put together induction on simplification rules with coinduction on propagation rules. Nonetheless as we have just argued the resulting language cannot be implemented in a complete way with respect to the logical reading of programs.

5 Applications

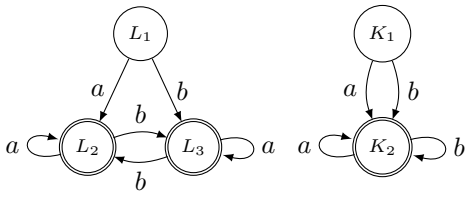
In this section, we illustrate the power of CHR for coinductive reasoning when it is provided with its fixpoint semantics. In particular we show it yields an elegant framework to realize coinductive equality proofs for languages and regular expressions as presented by Rutten (1998).

5.1 Coinductive language equality proof

Firstly, let us introduce the classical notion of binary automaton in a slightly different way from usual. A *binary automaton* is a pair $(\mathcal{L}, f)$ where $\mathcal{L}$ is a possibly infinite set of *states* and $f : \mathcal{L} \rightarrow \{0,1\} \times \mathcal{L} \times \mathcal{L}$ is a function called *destructor*. Let us assume some automaton $(\mathcal{L}, f)$. For any state $L \in \mathcal{L}$ such that $f(L) = (T, L_a, L_b)$, we write $L \xrightarrow{a} L_a$, and $L \xrightarrow{b} L_b$, and $t(L) = T$. $\mathcal{L}(L) = \{a_1 \dots a_n | L \xrightarrow{a_1} L_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} L_n \wedge t(L_n) = 1\}$ is the language accepted by a state L. A *bisimulation* between states is a relation $\mathcal{R} \subset \mathcal{L} \times \mathcal{L}$ verifying:

$$\text{If } K \,\mathcal{R}\, L \text{ then } \begin{cases} t(K) = t(L), \\ K \xrightarrow{a} K_a,\ L \xrightarrow{a} L_a,\ K_a \,\mathcal{R}\, L_a, \text{ and} \\ K \xrightarrow{b} K_b,\ L \xrightarrow{b} L_b,\ K_b \,\mathcal{R}\, L_b, \end{cases}$$

Contrary to the standard definition, in the present setting, an automaton does not have an initial state and may have an infinite number of states. As represented here, an automaton is a



$$\begin{aligned}\mathbb{D} = (&f(L_1,(0,L_2,L_3)),\\ &f(L_2,(1,L_2,L_3)),\\ &f(L_3,(1,L_3,L_2)),\\ &f(K_1,(0,K_2,K_2)),\\ &f(K_2,(1,K_2,K_2)))\end{aligned}$$

Figure 2: A binary automaton and its CPR representation.

particular coalgebra (Barwise and Moss 1996). Due to the space limitations, we will not enter in details in the topic of coalgebra[2], but only state the following *Coinductive Proof Principle* (Rutten 1998; Barwise and Moss 1996) which gives rise to the representation of automata as coalgebra:

> In order to prove the equality of the languages recognized by two states K and L, it is sufficient to establish the existence of a bisimulation relation in $\mathcal{L}$ that includes the pair (K, L).

A nice application of CPR consists in the possibility to directly represent coalgebra and prove bisimulation between states. For instance, one can easily represent a finite automaton using variables for states and binary user-defined constraints (of main symbol $f/2$) for the destructor function. Figure 2 gives an example of an automaton and its representation as a multiset $\mathbb{D}$ of CHR constraints. Once the automaton representation is fixed, one can translate the definition of bisimulation into a single propagation rule:

$$f(L,(L_t,L_a,L_b)),\ f(K,(K_t,K_a,K_b)),\ L{\sim}K \implies L_t{=}K_t,\ L_a{\sim}K_a,\ L_b{\sim}K_b$$

Using coinductive proof principle and Theorem 3.8, it is simple to prove two states of the coalgebra represented by $\mathbb{D}$ accept or not the same language. For example, to conclude that L_1 and K_1 recognize the same language while L_1 and K_2 do not, one can prove the execution of $\langle 3 \cdot (\mathbb{D}, L_1 \sim K_1); \top; \emptyset\rangle$ never reaches inconsistent states, while there are inconsistent derivations starting from $\langle 3 \cdot (\mathbb{D}, L_1 \sim K_2); \top; \emptyset\rangle$.

5.2 Coinductive solver for regular expressions

We have just shown that CPR yields a nice framework for coinductive reasoning about coalgebra. Nonetheless the explicit representation of an automaton by user defined constraints (as in Figure 2) would limit ourselves to finite state automata. One simple idea to circumvent this limitation is to implictly represent infinite states automata. For instance, one can represent states using regular expressions and implement the computation of the destructor function using derivatives (Rutten 1998).

Let us assume the following syntax for regular expressions:

$$E ::= L \mid a \mid b \mid E, E \mid E^* \qquad L ::= [] \mid [E|L]$$

where a and b are characters, $(^*)$ and $(,)$ stand for the Kleene star and the concatenation operators respectively, and a list corresponds to the alternation of its elements. Here follows a

[2]We invite unfamiliar readers to refer to the gentle introduction of Rutten (1998).

possible implementation of the destructor function[3]:

$$
\begin{array}{l}
f([],R) \iff R=(0,[],[]).\\
f(a,R) \iff R=(0,[[]^*],[]).\\
f(b,R) \iff R=(0,[],[[]^*]).\\
f([E|L],R) \iff R=(T,A,B), f(E,(E_t,E_a,E_b)), f(L,(L_t,L_a,L_b)),\\
\qquad or(E_t,L_t,T), merge(E_a,L_a,A), merge(E_b,L_b,B).\\
f(E^*,R) \iff R=(1,[(E_a,[E^*])],[(E_b,[E^*])]), f(E,(\_,E_a,E_b)).\\
f((E,F),R) \iff f(E,(E_t,E_a,E_b)), f_{\mathrm{conc}}(E_t,E_a,E_b,F,R).\\
f_{\mathrm{conc}}(0,E_a,E_b,F,R) \iff R=(0,[(E_a,F)],[(E_b,F)]).\\
f_{\mathrm{conc}}(1,E_a,E_b,F,R) \iff R=(T,A,B), f(F,F_a,F_b)),\\
\qquad merge([(E_a,F)],F_a,A), merge([(E_b,F)],F_b,B).
\end{array}
$$

where $or/3$ unifies its third argument with the Boolean disjunction of its two first elements and $merge/3$ unifies its last argument with the ordered union of the lists given as first arguments. Now one can adapt the encoding of bisimulation given in the previous subsection as follows:

$$L{\sim}K \Longrightarrow nonvar(L),nonvar(K)|f(L,(T,L_a,L_b)),f(K,(T,K_a,K_b)),L_a{\sim}K_a,L_b{\sim}K_b.$$

We are now able to prove equality of regular expression using the implementation of CHR hybrid semantics provided in Section 4.3. For example, the following state leads to an irreducible consistent state, proving thanks to the Coinductive Proof Principle, whether Theorem 4.10 and Theorem 4.15 that the two regular expressions recognize the same language:

$$\langle((b^*,a)^*,(a,b^*))^* \sim [[]^*,(a,[a,b]^*),([a,b]^*,(a,(a,[a,b]^*)))];\top;\emptyset\rangle^{\diamond}$$

It should be underlined that the use of simplification rules instead of propagation rules for encoding the destructor function is essential here in order to avoid rapid saturation of the memory by useless constraints. Notice that on the one hand, the confluence of the set of simplification rules needed by the theorems of Section 4 can be easily inferred, since the program is deterministic. On the other hand, termination of the the set of simplification rules, which is required by Theorem 4.16 can be established using standard ranking techniques (See for example section 5.1 in the Frühwirth's book).

Of course, no one should be surprised that equivalence of regular expressions is decidable. The interesting point here is that the notion of coalgebra and bisimulation can be casted naturally in CHR. Moreover, it is worth noticing the program given has the properties required by a constraint solver. Firstly the program is effective, i.e. it can actually prove or disprove if that two expressions are equal. The first part of the claim can be proved using Kleene theorem (Rutten 1998) and the idempotency and commutativity of the alternation, enforced here by the *merge/3* predicate. The second part is direct by the completeness with respect to failures. Secondly, the program is incremental: it can deal with partially instanciated expressions by freezing some computations provided without enough information. Last, but not least, one can easily add to the system new expressions (as for instance ϵ, $E?$, or E^+). For this purpose it is just necessary to provide a new simplification rule for computing the result of the corresponding destructor function. For example, we can add to the program the following rule and prove as previously that a^+ and (a,a^*) recognize the same language while a^+ and a^* do not:

$$f(K^+,R) \iff R=(T,\,[K_a,(K_a,K^+)],\,[K_b,(K_b,K^+)]),\; f(K,\,(T,\,K_a,\,K_b)).$$

[3] A complete version of the program can be found in Appendix A.

6 Conclusion

We have defined a l.f.p. semantics for CHR simplification rules and a g.f.p. semantics for CHR propagations rules, and proved both to be accurate with respect to the logical reading of the rules. By using a hybrid operational semantics with persistent constraints similar to the one of Betz et al., we were able to characterize CHR programs combining both simplification and propagation rules by a fixpoint semantics without losing completeness with respect to to logical semantics. In doing so, we have improved noticeably results about logical semantics of CHR. Subsequently we proposed an implementation of this hybrid semantics and showed it yields an elegant framework for programming with coinductive reasoning.

The observation that non-termination of all derivations starting from a given state ensures this latter to be in the coinductive semantics of an hybrid program, suggests that the statics analysis of universal non-termination of a CHR program might be worth investigating. The comparison of CHR to other coinductive programming frameworks such that the circular coinductive rewriting of Goguen, Lin, and Rosu (2000) may suggest it should be possible to improve completeness with respect to success of the implementation proposed here.

References

Abdennadher, Slim. 1997. "Operational Semantics and Confluence of Constraint Propagation Rules." *CP*, Volume 1330 of *LNCS*. Springer, 252–266.

Abdennadher, Slim, Thom Frühwirth, and Holger Meuss. 1999. "Confluence and Semantics of Constraint Simplification Rules." *Constraints* 4 (2): 133–165.

Barwise, Jon, and Larry Moss. 1996. *Vicious circles*. CSLI Publications.

Betz, Hariolf, Frank Raiser, and Thom Frühwirth. 2010. "A Complete and Terminating Execution Model for Constraint Handling Rules." *TPLP* 10 (Special Issue 4-6 (ICLP)): 597–610.

Bezem, Marc, and Thierry Coquand. 2005. "Automating Coherent Logic." *LPAR*, Volume 3835 of *LNCS*. Springer, 246–260.

Bueno, F., D. Cabeza, M. Carro, M. V. Hermenegildo, P. Lopez-Garca, and G. Puebla. 1997–2010. "The Ciao Prolog system. Reference manual." Technical Report, University of Madrid. System and on-line version of the manual available at `http://www.ciaohome.org`.

Clarke, Edmund M., Orna Grumberg, and Doron A. Peled. 2000. *Model Checking*. MIT Press.

de Boer, Frank S., and Catuscia Palamidessi. 1991. "A Fully Abstract Model for Concurrent Constraint Programming." *Theory and Practice of Software Development*, Volume 493 of *LNCS*. Springer, 296–319.

de Koninck, Lesli, Tom Schrijvers, and Bart Demoen. 2007. "User-definable rule priorities for CHR." *PPDP*. ACM, 25–36.

Frühwirth, Thom. 1998. "Theory and Practice of Constraint Handling Rules." *J. Log. Program.* 37 (1-3): 95–138.

———. 2009. *Constraint Handling Rules*. Cambridge University Press.

Gabbrielli, Maurizio, and Giorgio Levi. 1992. "Unfolding and Fixpoint Semantics of Concurrent Constraint Logic Programs." *Theor. Comput. Sci.* 105 (1): 85–128.

Gabbrielli, Maurizio, and Maria Meo. 2009. "A compositional semantics for CHR." *ACM Trans. Comput. Log.* 10, no. 2.

Gabbrielli, Maurizio, Maria Meo, and Paolo Tacchella. 2008. "A Compositional Semantics for CHR with Propagation Rules." In *Constraint Handling Rules: Current Research Topics*, Volume 5388 of *LNAI*, 119–160. Springer.

Goguen, Joseph A., Kai Lin, and Grigore Rosu. 2000. "Circular Coinductive Rewriting." *Automated Software Engineering*. 123–132.

Haemmerlé, Rémy, and François Fages. 2007. "Abstract Critical Pairs and Confluence of Arbitrary Binary Relations." *RTA*, LNCS no. 4533. Springer, 214–228.

Huet, Gérard. 1980. "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems: Abstract Properties and Applications to Term Rewriting Systems." *Journal of the ACM* 27 (4): 797–821 (October).

Jaffar, Joxan, and Jean-Louis Lassez. 1987. "Constraint Logic Programming." *POPL*. ACM, 111–119.

Lloyd, John. 1987. *Foundations of Logic Programming*. Springer.

Nyström, Sven-Olof, and Bengt Jonsson. 1993. "Indeterminate Concurrent Constraint Programming: A Fixpoint Semantics for Non-Terminating Computations." *ILPS*. 335–352.

Raiser, Frank, Hariolf Betz, and Thom Frühwirth. 2009. "Equivalence of CHR States Revisited." *CHR*, Report CW 555. Kath. Univ. Leuven, 34–48.

Rutten, Jan. 1998. "Automata and Coinduction (An Exercise in Coalgebra)." *CONCUR*, Volume 1466 of *LNCS*. Springer, 194–218.

Saraswat, Vijay A., Martin C. Rinard, and Prakash Panangaden. 1991. "Semantic foundations of concurrent constraint programming." *POPL*. ACM.

Simon, Luke, Ajay Bansal, Ajay Mallya, and Gopal Gupta. 2007. "Co-Logic Programming: Extending Logic Programming with Coinduction." *ICALP*, LNCS 4596:472–483.

Simon, Luke, Ajay Mallya, Ajay Bansal, and Gopal Gupta. 2006. "Coinductive Logic Programming." *ICLP*, Volume 4079 of *LNCS*. 330–345.

Terese. 2003. *Term Rewriting Systems*. Volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

A Coinductive solver for regular expressions

Here follows a complete version of the program presented in Section 5.2. The declarations `linear/1` and `persistent/1` are used to indicate linear and persistent constraint symbols, respectively. A prototype of the implementation described in Section 4.3 and used to execute this program has been implementated as Ciao Prolog package (Bueno et al. 2010).

```
:- op(200, yf, *).
:- op(200, yf, +).
:- op(200, yf, ?),
:- op(700, xfx, ~).

:- linear f/2, fconc/5.
:- persistent ~/2.
```

```
:- linear mergesort/2, split/3, merge_sorted/3, merge/3.
:- linear or/3.

%%%%%%%%%%%%%%%%%%%%
% Main constraints %
%%%%%%%%%%%%%%%%%%%%

%%% Destructor for essential regular expressions %%%

% We assume that list are always sorted w.r.t. @<

f([], R) <=> R = (0, [], []).
f([X|E], R) <=> R = (T, A, B),
    f(X, (Xt, Xa, Xb)), f(E, (Et, Ea, Eb)),
    or(Xt, Et, T), merge_sorted(Xa, Ea, A), merge_sorted(Xb, Eb, B).
f(a, R) <=> R = (0, [[]*], []).
f(b, R) <=> R = (0, [], [[]*]).
f(E*, R) <=> R = (1, [(Ea,[E*])], [(Eb,[E*])]),
    f(E, (_, Ea, Eb)).
f((E, F), R) <=>  f(E, (Et, Ea, Eb)),
    fconc(Et, Ea, Eb, F, R).
fconc(0, Ea, Eb, F, R) <=> R = (0, [(Ea,F)], [(Eb,F)]).
fconc(1, Ea, Eb, F, R) <=> R = (T, A, B),
    f(F, (T, Fa, Fb)),
    merge_sorted([(Ea, F)], Fa, A), merge_sorted([(Eb, F)], Fb, B).

%%% Destructor for redundant regular expressions %%%

f(eps, R) <=> R = (1, [], []).
f(empty, R) <=> R = (0, [], []).
f((E+F), R) <=> R = (T, A, B),
    f(E, (Et, Ea, Eb)), f(F, (Ft, Fa, Fb)),
    or(Et, Ft, T), merge(Ea, Fa, A), merge(Eb, Fb, B).
f(E+, R) <=> R = (T, [(Ea,[E*])], [(Eb,[E*])]),
    f(E, (T, Ea, Eb)).
f((E ?), R) <=>  R = (1, Ea, Eb),
    f(E, (_, Ea, Eb)).

%%% Bisimulation %%%

E~F ==> nonvar(E), nonvar(F) |
     f(E, (T, Ea, Eb)), f(F, (T, Fa, Fb)), Ea ~ Fa, Eb ~ Fb.

%%%%%%%%%%%%%%%%%%%%%%%%
% Utility constraints %
%%%%%%%%%%%%%%%%%%%%%%%%

mergesort([], S) <=> S = [].
mergesort([X], S) <=> S = [X].
mergesort([X, Y| T], S) <=> split(T, L1, L2),
   mergesort([X| L1], S1), mergesort([Y| L2], S2),
   merge_sorted(S1, S2, S).

split([], L1, L2) <=> L1 = [], L2 = [].
```

```
split([X], L1, L2) <=> L1 = [X], L2 = [].
split([X, Y| T], L1, L2) <=> L1 = [X| T1], L2 = [Y| T2],
    split(T, T1, T2).

merge_sorted(L, [], M) <=> M = L.
merge_sorted([], L, M) <=> M = L.
merge_sorted([X| T1], [Y| T2], M) <=> X @< Y | M = [X| T],
    merge_sorted(T1, [Y| T2], T).
merge_sorted([X| T1], [Y| T2], M) <=> X = Y  | M = [X| T],
    merge_sorted(T1, T2, T).
merge_sorted([X| T1], [Y| T2], M) <=> X @> Y | M = [Y| T],
    merge_sorted([X| T1], T2, T).

merge(L1, L2, M) <=>
    mergesort(L1, S1), mergesort(L2, S2),
    merge_sorted(S1, S2, M).

or(1, _, R) <=> R = 1.
or(0, 1, R) <=> R = 1.
or(0, 0, R) <=> R = 0.

%%%%%%%%
% Tests %
%%%%%%%%

expr('1a', a ? ).
expr('1b', eps + a).

expr('2a', a+ ).
expr('2b', (a, a*)).

expr('3a', ((b*, a)*,(a, b*))*).
expr('3b', eps + (a, (a+b)*) + ((a + b)*, (a, (a, (a + b)*)))).
expr('3c', [[]*, (a, [a,b]*), ([a,b]*,(a,(a,[a,b]*)))]).

expr('4', eps + (a, (a+b)*) + ((a + b)*, (a, (a, (a + b)*))) + b*).

test(I, J, R):-
    expr(I, E1), expr(J, E2),
    ( E1 ~ E2 -> R = equal; R = different).

test(I, J):-
    once(test(I, J, R)),
    format("expr ~w and ~w are ~w.~n", [I, J, R, X]),
    fail; true.

test:-
    test('1a', '1b'),  test('2a', '2b'),
    test('3a', '3b'),  test('3b', '3c'),  test('3c', '3a'),
    test('3a', '4'),   test('3b', '4').
```

B Logical model for hybrid programs

In this section, we construct a interpretation for confluent programs in the pure CLP style and demonstrate it provides a model for confluent programs.

A *constrained atom* is a pair $(c|\mathbb{C})$ where c is a CHR constrained and $\mathbb{C}$ is a conjunction of built-in atoms. The set of constrained atoms is called $\mathcal{C}$*-base* and denoted by $\mathcal{B}_\mathcal{C}$. For a set $\mathcal{Z}$ of constrained atoms, let us note $\Downarrow_\mathcal{C}(\mathcal{Z}) = \{(c|\mathbb{C}) \in \mathcal{B}_\mathcal{C} \mid (d|\mathbb{D}) \in \mathcal{Z} \text{ and } \mathcal{C} \vDash \mathbb{C} \rightarrow \exists_{\text{-}c}(c{=}d \wedge \mathbb{D})\}$. For a given interpretation I of $\mathcal{C}$, the set of closed I-instances of a set $\mathcal{Z}$ of constrained atoms, is defined as:

$$[\![\mathcal{Z}]\!]_I = \{c\rho \mid (c|\mathbb{C}) \in Z, I \vDash \mathbb{C}\rho\}$$

A set $\mathcal{Z}$ of constrained atoms is a $\mathcal{C}$*-model* for a program $\mathcal{P}$, if for any interpretation I of $\mathcal{C}$, $I \cup [\![\mathcal{Z}]\!]_I$ is a model of $\mathcal{P}$. By a slight abuse of notation, we will identify the state $\langle c_1, \ldots, c_n; \mathbb{C}; \emptyset\rangle$ with the set of constrained atoms $\{(c_1|\mathbb{C}), \ldots (c_1|\mathbb{C})\}$, when no confusion is possible.

Lemma B.1. *Let σ and σ' be two states equivalent with respect to $\equiv_h$. For any set $\mathcal{Z}$ of constrained atoms, if $\sigma \subset \mathcal{Z}$ then $\sigma' \subset \Downarrow_\mathcal{C}(\mathcal{Z})$.*

Proof. By induction on the definition of $\equiv_h$. □

Definition B.2 (Immediate consequence operator)**.** *For any hybrid program $\mathcal{P}$, the immediate consequence operator $T^\mathcal{C}_\mathcal{P} : 2^{\mathcal{B}_\mathcal{C}} \rightarrow 2^{\mathcal{B}_\mathcal{C}}$ is defined as:*

$$\begin{aligned} T^\mathcal{C}_\mathcal{P}(\mathcal{X}) = \{(c|\mathbb{C}) \in \mathcal{B}_\mathcal{C} \mid\ & \textit{their exists a renamed rule } (\mathbb{K}\backslash\mathbb{H} \Longleftrightarrow \mathbb{G}|d_i, \ldots, d_n, \mathbb{B}_b) \textit{ in } \mathcal{P} \\ & \textit{such that } ((d_1|\mathbb{C}_1), \ldots (d_n|\mathbb{C}_n)) \subset \mathcal{X},\ d \in \mathbb{H} \textit{ and,} \\ & \mathbb{C} = (\mathbb{G} \wedge \mathbb{B}_b \wedge \mathbb{C}_1 \wedge \cdots \wedge \mathbb{C}_n)\} \end{aligned}$$

It is worth noting that $T^\mathcal{C}_\mathcal{P}$ is nothing more than the immediate consequence operator of the CLP program derived form $\mathcal{P}$ when simplification rules are read as backward implications (i.e. implications from the right hand side to the left hand side). Indeed, one can considered the implication $c_1 \wedge \cdots \wedge c_n \leftarrow \exists Z(\mathbb{K} \wedge \mathbb{G} \wedge \mathbb{B})$ as the conjunction of the n implications $(c_1 \leftarrow \exists Z(\mathbb{K} \wedge \mathbb{G} \wedge \mathbb{B}))$, …, $(c_n \leftarrow \exists Z(\mathbb{G} \wedge \mathbb{D} \wedge \mathbb{B} \wedge \mathbb{K}))$. $T^\mathcal{C}_\mathcal{P}$ being obviously monotonic, Theorem 3.1 ensures it has a l.f.p.

Proposition B.3. *Let $\mathcal{P}$ be a set of hybrid simplifications such that $\xrightarrow{\mathcal{P}}{}^*_h$ is confluent, and $\mathcal{Y}$ be a set of valued persistent constraints. $\mu\mathcal{X}.\Downarrow_\mathcal{C}\big(\mathcal{Y} \cup T^\mathcal{C}_\mathcal{P}(\mathcal{X})\big)$ is a $\mathcal{C}$-model of $\mathcal{P}^\dagger$.*

The proof of the proposition uses two main lemmas.

Lemma B.4. *Let $\mathcal{P}$ be a program, $\mathcal{Y}$ and $\mathcal{Z}$ be a two set of constraints atoms, and $\langle \mathbb{E}; \mathbb{C}; X\rangle \xrightarrow{\mathcal{P}}{}^*_h \langle \mathbb{F}; \mathbb{D}; X\rangle$ be a valid derivation such that $\mathrm{fv}(\mathbb{E}, \mathbb{C}) \subset X$. If $\langle \mathbb{F}; \mathbb{D}; X\rangle \subset \mathcal{Z}$ and $\mathcal{Z}$ is a fixpoint of $\lambda\mathcal{X}.\Downarrow_\mathcal{C}\big(\mathcal{Y} \cup T^\mathcal{C}_\mathcal{P}(\mathcal{X})\big)$ then $\langle \mathbb{E}; \mathbb{C} \wedge \mathbb{D}; X\rangle \subset \mathcal{Z}$.*

Proof. We prove only the case where the derivation is a one step transition, the general case following directly by reflexivity and transitivity of the inclusion. Let $r@(\mathbb{K}\backslash\mathbb{H} \Longleftrightarrow \mathbb{G}|\mathbb{B}_c, \mathbb{B}_b)$ be the rule of $\mathcal{P}$ used for the transition. We have $\langle \mathbb{E}; \mathbb{C}; X\rangle \equiv_h \langle \mathbb{K}, \mathbb{H}, \mathbb{E}'; \mathbb{G}{\wedge}\mathbb{C}'; X\rangle$ and $\langle \mathbb{F}; \mathbb{D}; X\rangle \equiv_h \langle \mathbb{B}_c, \mathbb{K}, \mathbb{E}'; \mathbb{B}_b{\wedge}\mathbb{G}{\wedge}\mathbb{C}'; X\rangle$ with $\mathrm{lv}(r)$ for some $\mathbb{E}'$ and $\mathbb{C}'$. By Lemma B.1, we get $\langle \mathbb{B}_c, \mathbb{K}, \mathbb{E}'; \mathbb{B}_b{\wedge}\mathbb{G}{\wedge}\mathbb{C}'; X\rangle \subset \Downarrow_\mathcal{C}(\mathcal{Z})$, that is by idempotency of the closure operator $\Downarrow_\mathcal{C}$, $\langle \mathbb{B}_c, \mathbb{K}, \mathbb{E}'; \mathbb{B}_b \wedge \mathbb{G} \wedge \mathbb{C}'; X\rangle \subset$

$\mathcal{Z}$. By definition of $T^{\mathcal{C}}_{\mathcal{P}}$, we infer $\langle \mathbb{H}; \mathbb{B}_b \wedge \mathbb{G} \wedge \mathbb{C}'; X\rangle \subset \mathcal{Z}$, i.e. $\langle \mathbb{K}, \mathbb{H}, \mathbb{E}'; \mathbb{B}_b \wedge \mathbb{G} \wedge \mathbb{C}'; X\rangle \subset \mathcal{Z}$. On the other hand, we have $\mathcal{C} \vDash \exists_{\text{-}X}(\mathbb{B}_b \wedge \mathbb{G} \wedge \mathbb{C}') \leftrightarrow \exists_{\text{-}X}(\mathbb{B}_b \wedge \mathbb{G} \wedge \mathbb{C}') \wedge \exists_{\text{-}X}(\mathbb{G} \wedge \mathbb{C}')$, that is by soundness of $\equiv_h$, $\mathcal{C} \vDash \exists_{\text{-}X}(\mathbb{B}_b \wedge \mathbb{G} \wedge \mathbb{C}') \leftrightarrow \exists_{\text{-}X}\mathbb{D} \wedge \exists_{\text{-}X}\mathbb{C}$. Let $\mathcal{Z} = \text{fv}(\mathbb{D}) \setminus X$ and ρ a renamming of $\mathcal{Z}$ by fresh variables. We infer easily $\mathcal{C} \vDash \exists_{\text{-}X}(\mathbb{B}_b \wedge \mathbb{G} \wedge \mathbb{C}') \leftrightarrow \exists_{\text{-}X}(\mathbb{D}\rho \wedge \mathbb{C})$, that is by definition of $\equiv_h$, $\langle \mathbb{K}, \mathbb{H}, \mathbb{E}'; \mathbb{B}_b \wedge \mathbb{G} \wedge \mathbb{C}'; X\rangle \equiv_h \langle \mathbb{E}; \mathbb{C} \wedge \mathbb{D}\rho; X\rangle$. By Lemma B.1 and idempotency of $\Downarrow_{\mathcal{C}}$, we get $\langle \mathbb{E}; \mathbb{C} \wedge \mathbb{D}\rho; X\rangle \subset \mathcal{Z}$. We conclude using the fact that clearly $\mathcal{C} \vDash \mathbb{C} \wedge \mathbb{D} \rightarrow \exists_{\text{-}c}(\mathbb{C} \wedge \mathbb{D}\rho)$ and that $\mathcal{Z}$ is closed by $\Downarrow_{\mathcal{C}}$. □

Lemma B.5. *Let $\mathcal{P}$ be a confluent set of hybrid simplifications, $\mathcal{Y}$ be a set of constraints persistent constraints,$\mathbb{P}$ and $\mathbb{L}$ be two multisets of persistent and linear constraints, respectively, $\mathbb{E}$ be two multisets of CHR constraints, $\mathbb{C}$ and $\mathbb{D}$ be two conjunctions of built-in constraints, and X a set of variables such that that* $\text{fv}(\mathbb{L}, \mathbb{P}, \mathbb{C}) \subset X$. *For any ordinal α, if $\langle \mathbb{L}, \mathbb{P}; \mathbb{C}; X\rangle \xrightarrow{\mathcal{P}}{}^*_h \langle \mathbb{E}; \mathbb{D}; X\rangle$ is a valid derivation and $\langle \mathbb{L}, \mathbb{P}; \mathbb{C}; X\rangle \subset \lambda\mathcal{X}.\Downarrow_{\mathcal{C}}\big(\mathcal{Y} \cup T^{\mathcal{C}}_{\mathcal{P}}(\mathcal{X})\big) \uparrow \alpha$, then $\langle \mathbb{E}; \mathbb{D}; X\rangle \subset \mu\mathcal{X}.\Downarrow_{\mathcal{C}}\big(\mathcal{Y} \cup T^{\mathcal{C}}_{\mathcal{P}}(\mathcal{X})\big)$ and $\mathcal{C} \vDash \mathbb{C} \rightarrow \exists_{\text{-}X}\mathbb{D}$.*

Proof. Let note $F = \lambda\mathcal{X}.\Downarrow_{\mathcal{C}}\big(\mathcal{Y} \cup T^{\mathcal{C}}_{\mathcal{P}}(\mathcal{X})\big)$. The proof is by transfinite induction on α.

The base case i $\alpha = 0$ is trivial.

For a successor ordinal, we have $F \uparrow (\alpha{+}1) = \Downarrow_{\mathcal{C}}\big(\mathcal{Y} \cup T^{\mathcal{C}}_{\mathcal{P}}(F \uparrow \alpha)\big)$. That is $\langle \mathbb{P}; \mathbb{C}; X\rangle \subset \Downarrow_{\mathcal{C}}(\mathcal{Y})$ and $\langle \mathbb{L}; \mathbb{C}; X\rangle \subset \Downarrow_{\mathcal{C}}\big(T^{\mathcal{C}}_{\mathcal{P}}(F \uparrow \alpha)\big)$. By definition of $T^{\mathcal{C}}_{\mathcal{P}}$ and $\Downarrow_{\mathcal{C}}$, we infer $\mathbb{L}$ is of the form $c_1, \ldots, c_n$ such that for any $i \in 1, \ldots, n$ theire exists a renammed rules $r_i@(\mathbb{K}_i \backslash c_i, \mathbb{H}_i \iff \mathbb{G}_i \mid \mathbb{F}_i, \mathbb{B}_i)$ with $\langle \mathbb{K}_i, \mathbb{F}_i; \mathbb{C}; Y\rangle \subset F \uparrow \alpha$ and $\mathcal{C} \vDash \mathbb{C} \rightarrow \exists_{\text{-}c_i}(\mathbb{G}_i \wedge \mathbb{B}_i)$ (i.e. $\mathcal{C} \vDash \mathbb{C} \rightarrow \exists_{\text{-}X}\big(\mathbb{C} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}}\big)$). Let $Y = X \cup \text{fv}\big(\mathbb{E}, \bar{\mathbb{K}}, \bar{\mathbb{H}}, \bar{\mathbb{G}}, \bar{\mathbb{B}}\big)$ anf $\bar{A}$ be an abbreviation for any sequence of constraints of the form $A_i, \ldots, A_n$. On the one hand, using rules r_i's, one can infer $\big\langle \mathbb{P}, \mathbb{L}, \bar{\mathbb{K}}, \bar{\mathbb{H}}; \mathbb{C} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}}; Y\big\rangle \xrightarrow{\mathcal{P}}{}^*_a \big\langle \mathbb{P}, \bar{\mathbb{K}}, \bar{\mathbb{F}}; \mathbb{C} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}}; Y\big\rangle$ and on the other hand, using monotony of $\xrightarrow{\mathcal{P}}{}^*_a$ (Proposition 4.3), one get $\big\langle \mathbb{L}, \bar{\mathbb{K}}, \bar{\mathbb{H}}; \mathbb{C} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}}; Y\big\rangle \xrightarrow{\mathcal{P}}{}^*_a \big\langle \mathbb{E}, \bar{\mathbb{K}}, \bar{\mathbb{H}}; \mathbb{D} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}}; Y\big\rangle$. Since $\xrightarrow{\mathcal{P}}{}_a$ is confluent their exists a state $\langle \mathbb{F}'; \mathbb{D}'; Y\rangle$ such that $\big\langle \mathbb{P}, \bar{\mathbb{K}}, \bar{\mathbb{E}}; \mathbb{C} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}}; Y\big\rangle \xrightarrow{\mathcal{P}}{}^*_h \langle \mathbb{F}; \mathbb{D}; Y\rangle$ and $\big\langle \mathbb{E}, \bar{\mathbb{K}}, \bar{\mathbb{H}}; \mathbb{D} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}}; Y\big\rangle \xrightarrow{\mathcal{P}}{}^*_h \langle \mathbb{F}'; \mathbb{D}'; Y\rangle$. By induction hypothesis, we have $\langle \mathbb{F}'; \mathbb{D}'; Y\rangle \in \mu\mathcal{X}.\Downarrow_{\mathcal{C}}\big(\mathcal{Y} \cup T^{\mathcal{C}}_{\mathcal{P}}(\mathcal{X})\big)$ with $\mathcal{C} \models \mathbb{C} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}} \rightarrow \exists_{\text{-}Y}\mathbb{D}'$. By Lemma B.4, one get $\big\langle \mathbb{E}, \bar{\mathbb{K}}, \bar{\mathbb{H}}; \mathbb{D} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}} \wedge \mathbb{D}'; Y\big\rangle \subset \mu\mathcal{X}.\Downarrow_{\mathcal{C}}\big(\mathcal{Y} \cup T^{\mathcal{C}}_{\mathcal{P}}(\mathcal{X})\big)$. By strong monotonicity of built-in store, we have $\mathcal{C} \vDash \exists_{\text{-}Y}\big(\mathbb{D} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}}\big) \rightarrow \exists_{\text{-}Y}\mathbb{C}$, i.e. $\mathcal{C} \vDash \exists_{\text{-}X}(\mathbb{D}) \rightarrow \exists_{\text{-}X}\big(\mathbb{D} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{B}} \wedge \mathbb{D}'\big)$; hence by definition of $\equiv_h$, we get $\langle \mathbb{E}; \mathbb{D}; X\rangle \subset \mu\mathcal{X}.\Downarrow_{\mathcal{C}}\big(\mathcal{Y} \cup T^{\mathcal{C}}_{\mathcal{P}}(\mathcal{X})\big)$. We conclude by noticing that clearly $\mathcal{C} \vDash \exists_{\text{-}X}(\mathbb{D}') \rightarrow \exists_{\text{-}X}\mathbb{D}$, i.e. $\mathcal{C} \vDash \mathbb{C} \rightarrow \exists_{\text{-}X}\mathbb{D}$. □

Proof of Proposition B.3. If $\mathcal{C}$ has no interpretations, the result is trivial, otherwise let I be one of them. Let $\mathcal{Z} = \mu\mathcal{X}.(\mathcal{Y} \cup T^{\mathcal{C}}_{\mathcal{P}}(\mathcal{X}))$. Let $\mathbb{K} \backslash \mathbb{H} \iff \mathbb{G} \mid \mathbb{L}, \mathbb{P}, \mathbb{C}$ be a rule form $\mathcal{P}$ with local variables Z. (we note $X = \text{fv}(\mathbb{H}, \mathbb{K}, \mathbb{G})$). We have to show that $I \cup [\![\mathcal{Z}]\!]_I \cup \vDash \forall\big((\mathbb{K} \wedge \mathbb{H} \wedge \mathbb{G}) \rightarrow \exists_{\text{-}\text{fv}(\mathbb{K},\mathbb{H})}(\mathbb{G} \wedge \mathbb{L} \wedge \mathbb{P} \wedge \mathbb{C})\big) \wedge \forall((\mathbb{K} \wedge \mathbb{G} \wedge \mathbb{L} \wedge \mathbb{P} \wedge \mathbb{C}) \rightarrow \mathbb{H})$, i.e. for any valuation ρ (1) if $(\mathbb{K}, \mathbb{H}, \mathbb{G})\rho \subset (I \cup [\![\mathcal{Z}]\!]_I)$ then there exist a valuation ρ' of domain Z such that $(\mathbb{G}, \mathbb{C}, \mathbb{B})\rho'\rho \subset (I \cup [\![\mathcal{Z}]\!]_I)$ and (2) if $(\mathbb{K}, \mathbb{G}, \mathbb{C}, \mathbb{B})\rho \subset (I \cup [\![\mathcal{Z}]\!]_I)$ then $\mathbb{H}\rho \subset (I \cup [\![\mathcal{Z}]\!]_I)$. It is straightforward to show that for $\langle \mathbb{K}, \mathbb{H}; \mathbb{G}; \text{fv}(\mathbb{K}, \mathbb{H})\rangle \xrightarrow{\mathcal{P}}{}_h \langle \mathbb{K}, \mathbb{B}; \mathbb{G}, \mathbb{C}; \text{fv}(\mathbb{K}, \mathbb{H})\rangle$ is a valid transition; hence one can use Lemma B.5 to infer (1) and Lemma B.4 to infer (2). □

C Proofs

C.1 Proofs of Section 4.1. (Hybrid operational semantics ω_h)

Proof of Proposition 4.2. For *(i)* just notice that since n is bigger or equal than the maximal number of heads in $\mathcal{P}$, that is there is enough persistent constraints in $\langle \mathbb{L}, n \cdot \mathbb{P}; \mathbb{C}; \bar{X} \rangle$ to launch the rule in $\mathcal{P}^n$ that corresponds to the one used for $\langle \mathbb{L}, \mathbb{P}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}}_h \langle \mathbb{L}', \mathbb{P}'; \mathbb{C}'; \bar{X} \rangle$. *(ii)* is trivial. □

Proof of Proposition 4.3. By induction on the length of the derivation $\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle \xrightarrow{\mathcal{P}}_h^* \langle \mathbb{C}'; \mathbb{E}'; X' \rangle$. □

Proof of Proposition 4.4. By induction on the length of the derivation $\langle \mathbb{L}, \mathbb{P}; \mathbb{E}; \bar{X} \rangle \xrightarrow{\mathcal{P}}_h^* \langle \mathbb{D}; \mathbb{F}; \bar{Y} \rangle$ □

Proof of Proposition 4.5. Let n be a an integer bigger or equal than the maximal number of persistent constraints occurring in the head of the rules of $\mathcal{P}$ and let assume $\sigma = \langle \mathbb{L}, \mathbb{P}; \mathbb{C}; \bar{X} \rangle$. By Proposition 4.2, we know that $\langle \mathbb{L}, n \cdot \mathbb{P}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}}{}^* \sigma''$ such that $\sigma'' \equiv_h \sigma'$. By soundness of abstract semantics (cf. Lemma 3.20 in Frühwirth's Book (2009)), $\mathcal{CP} \vDash \forall(\langle \mathbb{L}, n \cdot \mathbb{P}; \mathbb{C}; \bar{X} \rangle^\dagger \leftrightarrow \sigma''^\dagger)$. We conclusion using the soundness of $\equiv_h$. □

Lemma C.1 (Huet (1980))**.** *Let E be an arbitrary set, and let $\rightarrow \subset E \times E$ be a reduction. If $\rightarrow$ is strongly confluent (i.e. for all elements e, e_1 and e_2, such that $e \rightarrow e_2$ and $e \rightarrow e_2$, there exists an elements e' such that $e_1 \rightarrow^\epsilon e$ and $e \rightarrow^\epsilon e_2$), then it is confluent.*

Lemma C.2 (Hindley-Rosen (Terese 2003))**.** *Let E be an arbitrary set, and let $\rightarrow_1 \subset E \times E$ and $\rightarrow_2 \subset E \times E$ be two confluent reductions. If $\rightarrow_1$ commutes with $\rightarrow_2$ (i.e. for all elements e, e_1 and e_2, such that $e \rightarrow_1 e_2$ and $e \rightarrow_2 e_2$, there exists an elements e' such that $e_1 \rightarrow_1 e$ and $e \rightarrow_2 e_2$), then $(\rightarrow_1 \cup \rightarrow_2)$ is confluent.*

Lemma C.3 (Commutation of propagations)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be hybrid programs. If $\mathcal{P}$ does not contain simplification rules then $\xrightarrow{\mathcal{P}}_h^*$ and $\xrightarrow{\mathcal{Q}}_h^*$ commutes.*

Proof. The lemma is corollary of Proposition 4.3 and Proposition 4.4. □

Proof of Proposition 4.6. By Lemma C.3, $\mathcal{P}$ commutes with itself, it is then strongly confluent, or equivalently thanks to Huet's lemma, confluent. Furthermore, by Lemma C.3, $\mathcal{P}$ commutes with $\mathcal{Q}$ as well ; hence by Hindley-Rosen's lemma $\xrightarrow{\mathcal{P}\cup\mathcal{Q}}_h^* = \xrightarrow{\mathcal{P}}_h^* \cup \xrightarrow{\mathcal{Q}}_h^*$ is confluent. □

C.2 Proofs of Section 4.2. (Hybrid transition system semantics)

[. Proof of Lemma 4.7] We show each property independently:

- For the *monotonicity of* $\langle \mathcal{P} \rangle_h$ let assume $\mathcal{X} \subset \mathcal{Y}$.
  - if $\sigma \in \langle \mathcal{P} \rangle_h(\mathcal{X})$ holds,

  - then there exists $\sigma' \in \mathcal{X}$ such that $\sigma \xrightarrow{\mathcal{P}}_h \sigma'$ holds.
  - then there exists $\sigma' \in \mathcal{Y}$ such that $\sigma \xrightarrow{\mathcal{P}}_h \sigma'$ holds.
  - then $\sigma \in \langle\mathcal{P}\rangle_h(\mathcal{Y})$ holds.

- For the *continuity of* $\langle\mathcal{P}\rangle_h$ let us assume a increasing chain $\{\mathcal{X}_i\}_{i\in\mathbb{N}}$.
  - $\sigma \in \langle\mathcal{P}\rangle_h\big(\sup\big(\{\mathcal{X}_i\}_{i\in\mathbb{N}}\big)\big)$ holds,
  - iff there exists $\sigma' \in \sup\big(\{\mathcal{X}_i\}_{i\in\mathbb{N}}\big)$ such that $\sigma \xrightarrow{\mathcal{P}}_a \sigma'$ holds,
  - iff for some $i \in \mathbb{N}$ there exist $\sigma' \in \mathcal{X}_i$ such that $\sigma \xrightarrow{\mathcal{P}}_a \sigma'$ holds,
  - iff for some $i \in \mathbb{N}$, $\sigma \in \langle\mathcal{P}\rangle_h(\mathcal{X}_i)$ holds.
  - iff $\sigma \in \sup\left(\{\langle\mathcal{P}\rangle_h(\mathcal{X}_i)\}_{i\in\mathbb{N}}\right)$ holds.

- For the *monotonicity of* $[\mathcal{P}]_h$ let assume $\mathcal{X} \subset \mathcal{Y}$.
  - if $\sigma \in \langle\mathcal{P}\rangle_h(\mathcal{X})$ holds,
  - then for any σ' such that $\sigma \xrightarrow{\mathcal{P}}_a \sigma'$, $\sigma' \in \mathcal{X}$ holds,
  - then for any σ' such that $\sigma \xrightarrow{\mathcal{P}}_a \sigma'$, $\sigma' \in \mathcal{Y}$ holds,
  - if $\sigma \in [\mathcal{P}]_h(\mathcal{Y})$

- For the *cocontinuity* let us assume a decreasing chain $\{\mathcal{X}_i\}_{i\in\mathbb{N}}$.
  - $\sigma \in [\mathcal{P}]_h\big(\inf\big(\{\mathcal{X}_i\}_{i\in\mathbb{N}}\big)\big)$ holds ,
  - iff for any σ' such that $\sigma \xrightarrow{\mathcal{P}}_a \sigma'$, $\sigma' \in \inf\big(\{\mathcal{X}_i\}_{i\in\mathbb{N}}\big)$ holds,
  - iff for any $i \in \mathbb{N}$ and for any σ' such that $\sigma \xrightarrow{\mathcal{P}}_a \sigma'$, $\sigma' \in \{\mathcal{X}_i\}$ holds,
  - iff for any $i \in \mathbb{N}$, $\sigma \in [\mathcal{P}]_h(\{\mathcal{X}_i\})$ holds,
  - iff $\sigma \in \inf\left\{([\mathcal{P}]_h(\{\mathcal{X}_i\}))\right\}_{i\in\mathbb{N}}$. □

The third line is true because for a given set finite set of rules (i.e. a program), the set $\{\sigma' \mid \sigma \xrightarrow{\mathcal{P}}_h \sigma'\}$ is a finite up to $\equiv_h$. □

Lemma C.4. *Let $\mathcal{P}$ and $\mathcal{Q}$ be two hybrid programs such that $\mathcal{Q} \subset \mathcal{P}$, and $\mathcal{Z}$ a set of states. Let assume the following functions:*

- $F_1 = \lambda\mathcal{Y}.(\langle\mathcal{Q}\rangle_h(\mathcal{Y}) \cup (\mathcal{Z} \setminus [\![\langle\emptyset; \bot; \emptyset\rangle]\!]_h))$
- $G_1 = \lambda\mathcal{Y}.(\langle\mathcal{Q}\rangle_h(\mathcal{Y}) \cup \mathcal{Z})$
- $F_2 = \lambda\mathcal{X}.([\mathcal{P}]_h(\mathcal{X}) \cap \mu\mathcal{Y}.F_1(\mathcal{Y}))$

- $G_2 = \lambda\mathcal{X}.([\mathcal{P}]_h(\mathcal{X}) \cap \mu\mathcal{Y}.G_1(\mathcal{Y}))$

For any ordinal α, we have:

(i) If $\sigma \in (G_1 \uparrow \alpha)$ and $\sigma \notin \mu\mathcal{Y}.F_1(\mathcal{Y})$ then $\sigma \xrightarrow{\mathcal{Q}}{}_h^* \langle\emptyset; \bot; \emptyset\rangle$.

(i) If $\sigma \in \nu\mathcal{X}.G_2(\mathcal{X})$ and $\sigma \notin (F_2 \downarrow \alpha)$ then $\sigma \xrightarrow{\mathcal{P}}{}_h^* \langle\emptyset; \bot; \emptyset\rangle$.

B. oth cases are by transfinite induction on α:

(i) The base case, $\alpha = 0$, is immediate.

For the inductive case, where α is a successor ordinal (i.e. $\alpha = \beta{+}1$), we have $\sigma \in (\langle\mathcal{Q}\rangle_h(G_1 \uparrow \beta) \cup \mathcal{Z})$. There are two subcases according σ:

- $\sigma \in (\langle\mathcal{Q}\rangle_h(G_1 \uparrow \beta))$. By definition of $\langle\mathcal{Q}\rangle_h$, one can infer there exists a valide transition $\sigma \xrightarrow{\mathcal{Q}}{}_h \sigma'$ such that $\sigma' \in (G_1 \uparrow \beta)$. Clearly $\sigma' \notin \mu\mathcal{Y}.F_1(\mathcal{Y})$, otherwise by definition of $\langle\mathcal{P}^s\rangle_h$, $\sigma \in \mu\mathcal{Y}.F_1(\mathcal{Y})$ would hold. Hence by induction hypothesis, $\sigma' \in [\![\langle\emptyset; \bot; \emptyset\rangle]\!]_h$ or $\sigma \xrightarrow{\mathcal{Q}}{}_h^* \langle\emptyset; \bot; \emptyset\rangle$, that is $\sigma \xrightarrow{\mathcal{P}}{}_h^* \langle\emptyset; \bot; \emptyset\rangle$.
- $\sigma \in \mathcal{Z}$. Since $\sigma \notin \mu\mathcal{Y}.F_1(\mathcal{Y})$, we by definition of a fixpoint $\sigma \notin \langle\mathcal{Q}\rangle_h(\mu\mathcal{Y}.F_1(\mathcal{Y})) \cup (\mathcal{Z} \setminus [\![\langle\emptyset; \bot; \emptyset\rangle]\!]_h)$, that is $\sigma \notin (\mathcal{Z} \setminus [\![\langle\emptyset; \bot; \emptyset\rangle]\!]_h)$. Hence $\sigma \in [\![\langle\emptyset; \bot; \emptyset\rangle]\!]_h$.

For the inductive case, where α is a limit ordinal, we have $\sigma \in \bigcup_{\beta<\alpha}(G_1 \uparrow \beta)$, that is $\sigma \in (G_1 \uparrow \beta)$ for some $\beta < \alpha$. By induction hypothesis, $\sigma \in [\![\langle\emptyset; \bot; \emptyset\rangle]\!]_h$ or $\sigma \xrightarrow{\mathcal{P}}{}_h^* \langle\emptyset; \bot; \emptyset\rangle$.

(i) The base case, $\alpha = 0$, is immediate.

For the inductive case, where α is a successor ordinal (i.e. $\alpha = \beta{+}1$), we have $\sigma \notin ([\mathcal{P}]_h(F_2 \downarrow \beta) \cap \mu\mathcal{Y}.F_1(Y))$. There are two subcases according σ:

- $\sigma \notin ([\mathcal{P}]_h(F_2 \downarrow \beta))$. By definition of $[\mathcal{P}]_h$, one can infer there exists a valid transition $\sigma \xrightarrow{\mathcal{P}}{}_h \sigma'$ such that $\sigma' \notin (F_2 \downarrow \beta)$. Hence by induction hypothesis, $\sigma' \in [\![\langle\emptyset; \bot; \emptyset\rangle]\!]_h$ or $\sigma \xrightarrow{\mathcal{P}}{}_h^* \langle\emptyset; \bot; \emptyset\rangle$, that is $\sigma \xrightarrow{\mathcal{P}}{}_h^* \langle\emptyset; \bot; \emptyset\rangle$.
- $\sigma \notin \mu\mathcal{Y}.F_1(Y)$. Since $\sigma \in \nu\mathcal{X}.G_2(\mathcal{X})$ (i.e. $\sigma \in ([\mathcal{P}]_h(\mathcal{X}) \cap \mu\mathcal{Y}.G_1(Y))$), we get by monotonicity of $\cap$ that $\sigma \in \mu\mathcal{Y}.G_1(\mathcal{Y})$. By continuity of $[\mathcal{P}]_h$ and $\cup$, Proposition 3.2, and case *(i)*, we get $\sigma \in [\![\langle\emptyset; \bot; \emptyset\rangle]\!]_h$ or $\sigma \xrightarrow{\mathcal{P}}{}_h^* \langle\emptyset; \bot; \emptyset\rangle$.

For the inductive case, where α is a limit ordinal, we have $\sigma \notin \bigcap_{\beta<\alpha}(F_2 \downarrow \beta)$, that is $\sigma \in (F_2 \downarrow \beta)$ for some $\beta < \alpha$. By induction hypothesis, $\sigma \in [\![\langle\emptyset; \bot; \emptyset\rangle]\!]_h$ or $\sigma \xrightarrow{\mathcal{P}}{}_h^* \langle\emptyset; \bot; \emptyset\rangle$. □

□

Lemma C.5. *Let $\mathcal{P}$ be a confluent hybrid program, $\mathcal{Y}$ be a set of irreducible and consistent states, let $F = \lambda\mathcal{X}.(\langle\mathcal{P}\rangle_h(\mathcal{X}) \cup \mathcal{Y})$, and let $\sigma \xrightarrow{\mathcal{P}}{}_h^* \sigma'$ be a valid transition. For any ordinal α, $\sigma \in (F \uparrow \alpha)$ then σ' is consistent.*

Proof. By transfinite induction on α. The bases case, $\alpha = 0$ is immediate. For the inductive case, where α is a successor ordinal (i.e. $\alpha = \beta + 1$), we have $\sigma \in \langle\mathcal{P}\rangle_h(F \uparrow \beta) \cup \mathcal{Y}$. There are two subcases:

- $\sigma \in \langle\mathcal{P}\rangle_h(F \uparrow \beta)$. By definition of $\langle\mathcal{P}\rangle_h$, on can infer there exists a valid transition $\sigma \xrightarrow{\mathcal{P}}_h \sigma''$ such that $\sigma'' \in \langle\mathcal{P}\rangle_h(F \uparrow \beta)$. By confluence of $\xrightarrow{\mathcal{P}}_h$, there exists a states σ''' such that $\sigma' \xrightarrow{\mathcal{P}}_h^* \sigma'''$ and $\sigma'' \xrightarrow{\mathcal{P}}_h^* \sigma'''$. By induction hypothesis, σ''' is consistent ; hence by monotonicity of $\xrightarrow{\mathcal{P}}_h$, σ' is consistent as well.
- $\sigma \in \mathcal{Y}$. Since $\mathcal{Y}$ is irreducible, $\sigma' = \sigma'$; hence by hypothesis on $\mathcal{Y}$, σ' is consistent.

For the inductive case, where α is a limit ordinal, we have $\sigma \in \bigcup_{\beta<\alpha}(F \uparrow \beta)$, that is $\sigma \in (F \uparrow \beta)$ for some $\beta < \alpha$. By induction hypothesis, σ' is consistent. □

Proof of Lemma 4.11. The "if" direction is consequence of Lemma C.4, cocontinuity of both $[\mathcal{P}]_h$ and $\cup$, and Proposition 3.2. For the "only if" direction let assume $\sigma \in \mathcal{F}_{\mathrm{h}}^{\mathcal{C}}(\mathcal{P})$. By definition of a fixpoint $\sigma \in [\mathcal{P}]_h\big(\mathcal{F}_{\mathrm{h}}^{\mathcal{C}}(\mathcal{P})\big) \cap \mu\mathcal{Y}.(\langle\mathcal{P}^s\rangle_h(\mathcal{P}) \cap (\Sigma_p \setminus [\![\langle\emptyset;\bot;\emptyset\rangle]\!]_h))$, that is $\sigma \in \mu\mathcal{Y}.(\langle\mathcal{P}^s\rangle_h(\mathcal{P}) \cap (\Sigma_p \setminus [\![\langle\emptyset;\bot;\emptyset\rangle]\!]_h))$. Clearly $(\Sigma_p \setminus [\![\langle\emptyset;\bot;\emptyset\rangle]\!]_h)$ is a set of states irreducible by $\mathcal{P}^s$, hence by Lemma C.5, continuity of both $\langle\mathcal{P}^s\rangle_h$ and $\cup$, and Proposition 3.2, $\sigma \not\xrightarrow{\mathcal{P}}_h^* \bot$. □

Lemma C.6. *Let $\mathcal{P}$ be a confluent and terminating set of hybrid simplification rules, $\mathbb{P}$ and $\mathbb{Q}$ be two multisets of persistent constraints, $\mathbb{C}$ and $\mathbb{D}$ two conjunctions of built-in constraints and X a set of variables. If $\mathcal{CP} \vDash \mathbb{P} \wedge \mathbb{C} \rightarrow \exists X(\mathbb{Q} \wedge \mathbb{D})$ then $\mathcal{C} \vDash \mathbb{P} \wedge \mathbb{C} \rightarrow \exists X(\mathbb{Q} \wedge \mathbb{D})$.*

Proof. Let assume $\mathcal{CP} \vDash \mathbb{P} \wedge \mathbb{C} \rightarrow \exists X(\mathbb{Q} \wedge \mathbb{D})$. Clearly c and d are logically equivalent to the declarative reading of the states $\langle\mathbb{P}_c;\mathbb{C}_c;X_c\rangle$ and $\langle\mathbb{P}_d;\mathbb{C}_d;X_d\rangle$ respectively (we assume without loss of generality $X_c \cap \mathrm{fv}(\mathbb{P}_d,\mathbb{C}_d) = \emptyset$).

To show that $\mathcal{C} \vDash c \rightarrow d$, it is sufficient to show that any set $\mathcal{Z}$ of valued constraints is a $\mathcal{C}$-model for $c \rightarrow d$. Let $\mathcal{Z}$ be a set of valued constraints and let $\mathcal{Y}$ be the set of persistent constraints contained in $\mathcal{Z}$. By Proposition B.3, $\mu\mathcal{X}.(\mathcal{Y} \cup T_{\mathcal{P}}^{\mathcal{C}}(\mathcal{X}))$ is a $\mathcal{C}$-model for $\mathcal{P}$ and then by hypothesis it is a $\mathcal{C}$-model for $\sigma^\dagger \rightarrow c$. By definition of a fixpoint $\mu\mathcal{X}.(\mathcal{Y} \cup T_{\mathcal{P}}^{\mathcal{C}}(\mathcal{X})) = \mathcal{Y} \cup T_{\mathcal{P}}^{\mathcal{C}}(\mu\mathcal{X}.(\mathcal{Y} \cup T_{\mathcal{P}}^{\mathcal{C}}(\mathcal{X}))$. By definition of $T_{\mathcal{P}}^{\mathcal{C}}$ and an hybrid program, for any set of constraints $\mathcal{X}$, $T_{\mathcal{P}}^{\mathcal{C}}(\mathcal{X})$ is a set of linear constraints. Hence the set of persistent constraints contained in $\mu\mathcal{X}.(\mathcal{Y} \cup T_{\mathcal{P}}^{\mathcal{C}}(\mathcal{X}))$ is precisely $\mathcal{Y}$. To conclude, one has to notice that since c and d are built from built-in and persistent constraints only, any set of valued CHR constraints is a $\mathcal{C}$-model for $c \rightarrow d$ if and only so is the set of its persistent constraints. □

In the following, we will assume the notion of *formal constraint* which is defined as quantified conjunction of (built-in or CHR) constraints (i.e. a logical formula of the form $\exists X.\mathbb{C}$). We defined furthermore $\Vdash_{\mathcal{C}}$, $\Vdash_{\mathcal{P}}$, $\Vdash_{=}$ as the least binary relations on formal constraints verifying:

- $\mathbb{C} \Vdash_{\mathcal{C}} \exists Y\mathbb{C}'$ if $\forall(\mathbb{C} \rightarrow \exists Y\mathbb{C}')$ is a non-logical axiom of $\mathcal{C}$;
- $\mathbb{K} \wedge \mathbb{H} \wedge \mathbb{G} \Vdash_{\mathcal{P}} \exists Z(\mathbb{C} \wedge \mathbb{G} \wedge \mathbb{B})$ and $\mathbb{K} \wedge \mathbb{C} \wedge \mathbb{G} \wedge \mathbb{B} \Vdash_{\mathcal{P}} \mathbb{H}$, if $r \mathbin{@} (\mathbb{K} \backslash \mathbb{H} \iff \mathbb{G} | \mathbb{C}, \mathbb{B}) \in \mathcal{P}$ and Z are the local variables of r;
- $p(x_1,\ldots,x_n) \wedge x_1 = y_1 \wedge \cdots \wedge x_n = y_n \Vdash_{=} p(y_1,\ldots,y_n)$ if p is a CHR constraints symbol of arity n.

Finally we define $\vdash_{\mathcal{P}^c}$ as the least relation containing $\Vdash_{\mathcal{C}}$, $\Vdash_{\mathcal{P}}$ and $\Vdash_=$, and closed by the rules of intuitionistic logic

$$c \vdash c \qquad \frac{\Gamma \vdash d \quad \Gamma, d \vdash c}{\Gamma \vdash d} \qquad \Gamma \vdash \top \qquad \Gamma, \bot \vdash c \qquad \frac{\Gamma, d, d \vdash c}{\Gamma, d \vdash c} \qquad \frac{\Gamma \vdash c}{\Gamma, d \vdash c}$$
$$\frac{\Gamma \vdash c \quad \Gamma \vdash d}{\Gamma \vdash c \wedge d} \qquad \frac{\Gamma, c_1, c_2 \vdash d}{\Gamma, c_1 \wedge c_2 \vdash d} \qquad \frac{\Gamma \vdash c}{\Gamma \vdash \exists x c} \qquad \frac{\Gamma, c \vdash d}{\Gamma, \exists x c \vdash d}\, x \notin \mathrm{fv}(\Gamma, d)$$

In the reaming of the section, we use $\models$ and $\vdash$ to emphasize that a reasoning is done in the classical logic model theory and intuitionistic proof theory, respectively. Fortunately, as shown by the following lemma, both lines of reasoning coincide in our framework. This remarkable property is due to the fact we are reasoning in a fragment of classical logic, know as *coherent logic*, where classical provability coincide with intuitionistic provability (Bezem and Coquand 2005).

Lemma C.7. *Let $\mathbb{C}$ and $\mathbb{D}$ be conjunction of constraints, and X be a set of variables. We have:*

$$\mathcal{CP} \models \mathbb{C} \to \exists X.\mathbb{D} \text{ if and only if } \mathbb{C} \vdash_{\mathcal{P}^c} \exists X.\mathbb{D}$$

Proof. Clearly $\mathbb{C} \vdash_{\mathcal{P}^c} \exists X.\mathbb{D}$ if and only if $\mathcal{CP} \vdash \forall(\mathbb{C} \to \exists X.\mathbb{D})$. Hence, one has just to notice that the axioms of $\mathcal{CP}$ and the formula $\mathbb{C} \to \exists X.\mathbb{D}$ are coherent logic formulas (i.e. formulas of the form $\forall(\mathbb{E} \to \exists X.\mathbb{F})$, where both $\mathbb{E}$ and $\mathbb{F}$ are conjunction of atomic propositions) and use the fact that classical provability and intuitionistic provability coincide in the coherent logic (See for instance Bezem and Coquand's work (2005)). □

We define an alternative logical reading of a states σ:

$$\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle^{\ddagger} = \begin{cases} \mathbb{C}, \mathbb{E} & \text{if } \mathrm{lv}\big(\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle\big) = \emptyset \\ \langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle & \text{otherwise} \end{cases}$$

Lemma C.8. *Let $\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle$ and $\langle \mathbb{D}; \mathbb{F}; \bar{Y} \rangle$ be purely persistent states. If $\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle^{\ddagger} \vdash_{\mathcal{C}} \langle \mathbb{D}; \mathbb{F}; \bar{Y} \rangle^{\dagger}$ then $\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle \equiv_h \langle \mathbb{C}, \mathbb{D}\rho; \mathbb{E} \wedge \mathbb{F}\rho; \bar{X} \rangle$ where ρ is a renaming of local variables of $\langle \mathbb{D}; \mathbb{F}; \bar{Y} \rangle$.*

Proof. By induction on the proof tree π of the sequent $\langle \mathbb{C}; \mathbb{E}; \bar{X} \rangle^{\ddagger} \vdash_{\mathcal{C}} \langle \mathbb{D}; \mathbb{F}; \bar{Y} \rangle^{\dagger}$. □

Lemma C.9. *Let $\mathcal{P}$ be a hybrid program such that $\mathcal{P}^s$ is confluent. Let σ be a data-sufficient and purely persistent states, and Γ a multiset of formal constraints.*

If $\sigma^{\dagger} \vdash_{\mathcal{P}^{lc}} \bigwedge \Gamma^{\dagger}$ and $\Gamma \vdash_{\mathcal{P}^c} c$, then $\sigma \xrightarrow{\mathcal{P}}{}^*_h \sigma'$ and $\sigma'^{\dagger} \vdash_{\mathcal{P}^{lc}} c$.

B. y induction on the proof tree of $\Gamma \vdash_{\mathcal{P}^c} c$. We focus only on the non trivial cases:

- π is a non-logical axiom of $\mathcal{C}$ or the equality axiom. The result is direct by applying rule 3 and 1 of the definition of abstract equivalence.
- π is a non-logical axiom corresponding to the logical reading of a propagation rules:

$$\mathbb{K} \wedge \mathbb{G} \Vdash_{\mathcal{P}^p} \exists Z(\mathbb{G} \wedge \mathbb{B}_c \wedge \mathbb{B}_b) \qquad \text{with } r \,@\, (\mathbb{K} \backslash \mathbb{H} \Longleftrightarrow \mathbb{G} | \mathbb{B}_c, \mathbb{B}_p) \in \mathcal{P}$$

By Lemma C.6 and Lemma C.7, one infer $\sigma^\dagger \vdash_{\mathcal{C}} \mathbb{K} \wedge \mathbb{G}$. By Lemma C.8, σ is equivalent to some state of the form $\langle \mathbb{C}, \mathbb{K}; \mathbb{E} \wedge \mathbb{G}; \bar{X} \rangle$, hence $\sigma \xrightarrow{\mathcal{P}}{}^*_h \langle \mathbb{C}, \mathbb{K}, \mathbb{B}_c; \mathbb{E} \wedge \mathbb{G} \wedge \mathbb{B}_c; \bar{X} \rangle$. Since σ is data-sufficient, we infer the existence of a purely persistent state σ' such that $\langle \mathbb{C}, \mathbb{K}, \mathbb{B}_c; \mathbb{E} \wedge \mathbb{G} \wedge \mathbb{B}_c; \bar{X} \rangle \xrightarrow{\mathcal{P}^s}{}^*_h \sigma'$. By soundness of hybrid transition, $\mathcal{P}^{s\mathcal{C}} \vDash \sigma' \leftrightarrow \langle \mathbb{C}, \mathbb{K}, \mathbb{B}_c; \mathbb{E} \wedge \mathbb{G} \wedge \mathbb{B}_c; \bar{X} \rangle^\dagger$, that implies by Lemma C.7 $\sigma' \vdash_{\mathcal{P}^{lc}} \exists Z(\mathbb{G} \wedge \mathbb{B}_c \wedge \mathbb{B}_b)$.

- π ends with a cut:

$$\frac{\Gamma \vdash_{\mathcal{P}^c} d \qquad \Gamma, d \vdash_{\mathcal{P}^c} c}{\Gamma \vdash_{\mathcal{P}^c} c}$$

By induction hypothesis, there exists a purely persistent state σ'' such that $\sigma \xrightarrow{\mathcal{P}}{}^*_h \sigma''$ and $\sigma''^\dagger \vdash_{\mathcal{P}^{lc}} d$. By strong monotonicity of built-in and persistent stores (Proposition 4.4) one can infer effortless $\sigma''^\dagger \vdash \sigma^\dagger$, that is $\sigma''^\dagger \vdash_{\mathcal{P}^{lc}} d \wedge \bigwedge \Gamma$. By induction hypothesis, there exists a purely persistent state σ' with $\sigma'' \xrightarrow{\mathcal{P}}{}^*_h \sigma'$ and $\sigma'^\dagger \vdash_{\mathcal{P}^{lc}} \exists X.c$. The conclusion of the case follows by transitivity of $\xrightarrow{\mathcal{P}}{}^*_h$.

- π ends with a right introduction of $\wedge$:

$$\frac{\Gamma \vdash_{\mathcal{P}^c} c \qquad \Gamma \vdash_{\mathcal{P}^c} d}{\Gamma \vdash_{\mathcal{P}^c} c \wedge d}$$

By induction hypothesis, there exists two purely persistent states σ_1 and σ_2 such that $\sigma \xrightarrow{\mathcal{P}}{}^*_h \sigma_1$, $\sigma \xrightarrow{\mathcal{P}}{}^*_h \sigma_2$, $\sigma_1^\dagger \vdash_{\mathcal{P}^{lc}} c$, and $\sigma_2^\dagger \xrightarrow{\mathcal{P}}{}^*_h d$. By Lemma C.3 we infer $\xrightarrow{\mathcal{P}}{}^*_h$ is confluent, then there exists a state σ' such that $\sigma_1 \xrightarrow{\mathcal{P}}{}^*_h \sigma'$ and $\sigma_2 \xrightarrow{\mathcal{P}}{}^*_h \sigma'$. By strong monotony of built-in and persistent stores one can infer $\sigma'^\dagger \vdash \sigma_1^\dagger$ and $\sigma'^\dagger \vdash \sigma_2^\dagger$ that is $\sigma \vdash \sigma_1^\dagger \wedge \sigma_2^\dagger$. Hence $\sigma' \vdash_{\mathcal{P}^{lc}} c \wedge d$, with by transitivity $\sigma \xrightarrow{\mathcal{P}}{}^*_h \sigma'$. □ □

Proof of Lemma 4.12. The lemma is corollary of Lemma C.9. □