TLP **11** (4–5): 593–609, 2011. © Cambridge University Press 2011
doi:10.1017/S1471068411000196

(Co-)Inductive semantics for Constraint Handling Rules

RÉMY HAEMMERLÉ
Technical University of Madrid, Madrid, Spain

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 the greatest fixpoint. Then, in order to take advantage of both types of rules without losing fixpoint characterization, we present a new operational semantics with persistent constraints.

We finally establish that this semantics can be characterized by two nested fixpoints, and we show that the resulting language is an elegant framework to program using coinductive reasoning.

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

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 *et al.* (2010)).

Soundness of operational semantics of CSR (i.e., to each derivation that corresponds a deduction) has been proved by Frühwirth (2009), while completeness (i.e., to each deduction corresponds a derivation) has been tackled by Abdennadher *et al.* (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. As any set of propagation rules trivial 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, 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 ω_h similar to the one recently proposed by Betz *et al.* (2010). Subsequently we demonstrate that this new semantics can be characterized by two nested fixpoints and can be implemented in a simple manner to provide the first logically complete system (w.r.t. failures) for a segment of CHR with propagations rules. We also show that this semantics 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, we present two fixpoint semantics for CHR. We show that these semantics, which are 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 introduced recently by Betz *et al.* (2010). We prove that 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.

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 for implementation purposes.

2.1 Syntax

The formalization of CHR assumes a language of *built-in constraints* containing the equality $=$, **false**, and **true** over some theory $\mathscr{C}$ 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, $\dots$, and (user-defined or built-in) constraints by lowercase letters $c, d, e\dots$ 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 $\mathrm{fv}(\phi)$, a 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 $\mathscr{C}$ 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 *et al.*'s (1991) *simple constraints system.*

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

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

where $\mathbb{K}$ (the *kept head*) and $\mathbb{H}$ (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 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:

$$r \ @\ \mathbb{K} \Longrightarrow \mathbb{G} \mid \mathbb{C}, \mathbb{B}.$$

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 a set of states and Σ_b a 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 $\mathscr{C}$ (i.e., there exists an interpretation of $\mathscr{C}$, which is a model of $\exists\mathbb{C}$), *inconsistent* otherwise.

2.2 Declarative semantics

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

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

$\mathscr{CP}$, the *logical reading* of a program $\mathscr{P}$ within a constraint theory $\mathscr{C}$, is a conjunction of logical readings of rules of $\mathscr{P}$ with constraint theory $\mathscr{C}$.

2.3 Equivalence-based operational semantics

Here we recall the *equivalence-based* operational semantics, ω_e, given by Raiser *et al.* (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. As 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 logical equivalent readings in the theory $\mathscr{C}$. The equivalence class of some state σ by $\equiv_i$ will be noted $[\![\sigma]\!]_i$. For a given program $\mathscr{P}$ and a sound equivalence $\equiv_i$, the $\equiv_i$*-transition relation*, noted $\overset{\mathscr{P}}{\rightarrow}_i$, is the least relation satisfying the following rules:

$$\frac{\left(r \ @ \ \mathbb{K}\backslash\mathbb{H} \Longleftrightarrow \mathbb{G}|\mathbb{C},\mathbb{B}\right) \in \mathscr{P}\rho \quad \mathrm{lv}(r) \cap \bar{X} = \emptyset}{\langle \mathbb{K},\mathbb{H},\mathbb{D};\mathbb{G}\wedge\mathbb{E};\bar{X}\rangle \overset{\mathscr{P}}{\rightarrow}_i \langle \mathbb{C},\mathbb{K},\mathbb{D};\mathbb{B}\wedge\mathbb{G}\wedge\mathbb{E};\bar{X}\rangle} \qquad \frac{\sigma_1 \equiv_i \sigma_1' \quad \sigma_1' \overset{\mathscr{P}}{\rightarrow}_i \sigma_2' \quad \sigma_2' \equiv_i \sigma_2}{\sigma_1 \overset{\mathscr{P}}{\rightarrow}_i \sigma_2},$$

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 state σ. The transitive closure of the relation $(\overset{\mathscr{P}}{\rightarrow}_i \cup \equiv_i)$ is denoted by $\overset{\mathscr{P}}{\rightarrow}_i^*$. A $\equiv_i$*-derivation* is a finite or infinite sequence of the form $\sigma_1 \overset{\mathscr{P}}{\rightarrow}_i \ldots \overset{\mathscr{P}}{\rightarrow}_i \sigma_n \overset{\mathscr{P}}{\rightarrow}_i \ldots$ If $\equiv_i$-derivation is *consistent*, so are all the states constituting it. A $\equiv_i$-transition is *confluent* if whenever $\sigma \overset{\mathscr{P}}{\rightarrow}_i^* \sigma_1$ and $\sigma \overset{\mathscr{P}}{\rightarrow}_i^* \sigma_2$ hold, there exists a state σ' such that $\sigma_1 \overset{\mathscr{P}}{\rightarrow}_i^* \sigma'$ and $\sigma_2 \overset{\mathscr{P}}{\rightarrow}_i^* \sigma'$ hold as well.

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

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. $\langle \mathbb{C}; \mathbf{false}; \bar{X}\rangle \equiv_a \langle \mathbb{D}; \mathbf{false}; \bar{X}\rangle$.
3. $\langle \mathbb{C}; \mathbb{D}; \bar{X}\rangle \equiv_a \langle \mathbb{C}; \mathbb{E}; \bar{X}\rangle$ if $\mathscr{C} \vDash \exists_{\text{-fv}(\mathbb{C},\bar{X})}(\mathbb{D}) \leftrightarrow \exists_{\text{-fv}(\mathbb{C},\bar{X})}(\mathbb{E})$.
4. $\langle \mathbb{C}; \mathbb{E}; \bar{X}\rangle \equiv_a \langle \mathbb{C}; \mathbb{E}; \{Y\} \cup \bar{X}\rangle$ where $Y \notin \mathrm{fv}(\mathbb{C},\mathbb{E})$.

Note that this equivalence is logically sound (Raiser *et al.* 2009). For a given program $\mathscr{P}$, the *abstract transition system* is defined as the tuple $(\Sigma, \overset{\mathscr{P}}{\rightarrow}_a)$.

2.4 Concrete operational semantics

This section presents the operational semantics, ω_p, given by de Koninck *et al.* (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 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 constraint, 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. Moreover, we assume that the identifier of each identified constraints in the CHR store is unique and smaller than n. For any program $\mathscr{P}$, the *concrete transition* relation, $\overset{\mathscr{P}}{\rightarrow}_c$, is defined as follows:

Solve $\langle\!\langle c, \mathbb{C}; \mathbb{D}; \mathbb{E}; T\rangle\!\rangle_n^{\bar{X}} \overset{\mathscr{P}}{\rightarrow}_c \langle\!\langle \mathbb{C}; \mathbb{D}; c \wedge \mathbb{E}; T\rangle\!\rangle_n^{\bar{X}}$
if c is a built-in constraint and $\mathscr{C} \models \forall((c \wedge \mathbb{B}) \leftrightarrow \mathbb{B}')$.
Introduce $\langle\!\langle c, \mathbb{C}; \mathbb{D}; \mathbb{E}; T\rangle\!\rangle_n^{\bar{X}} \overset{\mathscr{P}}{\rightarrow}_c \langle\!\langle \mathbb{C}; c\#n, \mathbb{D}; \mathbb{E}; T\rangle\!\rangle_{n+1}^{\bar{X}}$
if c is a CHR constraint.
Apply $\langle\!\langle \emptyset; \mathbb{K}, \mathbb{H}, \mathbb{E}; \mathbb{C}; T\rangle\!\rangle_n^{\bar{X}} \overset{\mathscr{P}}{\rightarrow}_c \langle\!\langle \mathbb{B}, \mathbb{G}; \mathbb{K}, \mathbb{D}; \mathbb{C} \wedge \theta; t \cup T\rangle\!\rangle_n^{\bar{X}}$
if $(p :: r @ \mathbb{K}' \backslash \mathbb{H}' \Leftrightarrow \mathbb{G} \mid \mathbb{B})$ is a rule in $\mathscr{P}$ 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 $\mathscr{C}$, and $\mathscr{C} \models \forall(\mathbb{C} \rightarrow \exists(\theta \wedge G))$. 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 *et al.* 2000). The proofs of this section are only sketched. Detailed versions can be found in a technical report (Haemmerle *et al.* to appear).

Before formally introducing the semantics, we recall some standard notations and results about fixpoints in an arbitrary complete lattice $(\mathscr{L}, \supset, \cap, \cup, \top, \bot)$. [1] A function $f : \mathscr{L} \rightarrow \mathscr{L}$ is *monotonic* if $f(\mathscr{X}) \supset f(\mathscr{Y})$ whenever $\mathscr{X} \supset \mathscr{Y}$. An element $\mathscr{X} \in \mathscr{L}$ is a *fixpoint* for $f : \mathscr{L} \rightarrow \mathscr{L}$ if $f(\mathscr{X}) = \mathscr{X}$. The *least fixpoint* (respectively, *the greastest fixpoint*) of f is its fixpoint $\mathscr{X}$ satisfying $\mathscr{Y} \supset \mathscr{X}$ (respectively, $\mathscr{Y} \subset \mathscr{X}$) whenever $\mathscr{Y}$ is a fixpoint for f. It is denoted by $\mu\mathscr{X}.f(\mathscr{X})$ (respectively, $\nu\mathscr{X}.f(\mathscr{X})$). Tarski's (1995) celebrated fixpoint theorem ensures that monotonic functions have both an l.f.p. and a g.f.p.

3.1 Inductive semantics for CSR

In this section we give a first fixpoint semantics limited to CSR. It is called inductive, as it is defined as an l.f.p.

Definition 3.1 (*Inductive transition system semantics for CSR*)
For a given program $\mathscr{P}$ and a given sound equivalence relation $\equiv_i$, the *existential immediate cause operator* $\langle\mathscr{P}\rangle_i : 2^\Sigma \rightarrow 2^\Sigma$ is defined as

$$\langle\mathscr{P}\rangle_i(\mathscr{X}) = \{\sigma \in \Sigma \mid \text{there exists } \sigma' \in \Sigma \text{ such that } \sigma \overset{\mathscr{P}}{\rightarrow}_i \sigma' \text{ and } \sigma' \in \mathscr{X}\}.$$

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

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

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

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

Example 3.2
Consider the program $\mathscr{P}_1$ consisting of the following two rules:

$$a, a \Longleftrightarrow \textbf{true} \qquad\qquad b \Longleftrightarrow b \qquad\qquad c \Longleftrightarrow \textbf{false}$$

We can pick sets of consistent states of respective forms $\langle a^{2i}; \mathbb{C}; \bar{X}\rangle$ and $\langle a^{2i}, b^j; \mathbb{C}; \bar{X}\rangle$, where d^n denotes n copies of a constraint d. Both sets are fixpoints of $\lambda\mathscr{X}.\big(\langle\mathscr{P}_1\rangle_a(\mathscr{X}) \cup \big(\Sigma_b \setminus [\![\langle\emptyset; \textbf{false}; \emptyset\rangle]\!]_a\big)\big)$, but only the former is an l.f.p. Note that no state of the form $\langle a^{2i+1}, \mathbb{E}; \mathbb{C}; \bar{X}\rangle$ or $\langle c, \mathbb{E}; \mathbb{C}; \bar{X}\rangle$ is in such fixpoints.

We next present a theorem that uses fixpoints semantics to reformulate results on CSR logical semantics (Frühwirth 1998; Abdennadher *et al.* 1999). It says that a state that has at least one answer is in the inductive semantics of a confluent CSR program if and only if its logical reading is satisfiable within the theory $\mathscr{C}\mathscr{P}$. Note that because in the context of this paper, we do not require $\mathscr{C}$ 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 3.3
Let $\mathscr{P}$ be program such that $\overset{\mathscr{P}}{\rightarrow}_a$ is confluent. Let $\langle\mathbb{D}; \mathbb{E}; \bar{X}\rangle$ be a state having at least one answer. We have

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

Proof sketch
As we have already said, $\mathscr{F}^{\mathscr{C}}(\mathscr{P})$ is the set of states that can be rewritten to a consistent answer. Hence, it is sufficient to prove:

$$\big\langle\mathbb{E}; \mathbb{C}; \bar{X}\big\rangle \overset{\mathscr{P}}{\rightarrow}_a^{*} \big\langle\emptyset; \mathbb{D}; \bar{Y}\big\rangle \text{ with } \mathscr{C}\mathscr{P} \nvDash \neg\exists(\mathbb{D}) \text{ if and only if } \mathscr{C}\mathscr{P} \nvDash \neg\exists(\mathbb{E} \wedge \mathbb{C})$$

or equivalently, the contrapositive:

$$\mathscr{C}\mathscr{P} \vDash \neg\exists(\mathbb{E} \wedge \mathbb{C}) \text{ if and only if } \big\langle\mathbb{E}; \mathbb{C}; \bar{X}\big\rangle \overset{\mathscr{P}}{\rightarrow}_a^{*} \langle\emptyset; \textbf{false}; \emptyset\rangle.$$

"If" and "only if" directions are respective corollaries of soundness and completeness for CHR (Lemma 3.20 and Theorem 3.25 in Frühwirth's (2009)book). □

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 behavior and is not formally related to logical semantics, although it is constructed in similar way as an 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 *et al.* 2008). Nonetheless, such an extension leads to a quite complicated model, which is moreover incomplete with respect to logical semantics.

3.2 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 cause operator presented in Definition 3.1. Hence, we call it coinductive.

Definition 3.4 (*Coinductive transition system semantics for CPR*)
For a given program $\mathscr{P}$ and a given sound equivalence relation $\equiv_i$, the *universal (immediate) cause operator* $[\mathscr{P}]_i : 2^\Sigma \to 2^\Sigma$ is defined as

$$[\mathscr{P}]_i(\mathscr{X}) = \{\sigma \in \Sigma \mid \text{for all } \sigma' \in \Sigma, \sigma \overset{\mathscr{P}}{\rightarrow}_i \sigma' \text{ implies } \sigma' \in \mathscr{X}\}.$$

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

$$\mathscr{F}^{\mathscr{C}}_{\mathrm{co}}(\mathscr{Q}) = \nu\mathscr{X}.\big([Q]_a(\mathscr{X}) \cap \big(\Sigma \setminus [\![\langle\emptyset;\mathbf{false};\emptyset\rangle]\!]_a\big)\big).$$

Note that contrary to the inductive semantics, the coinductive one is not just a reformulation of the existing semantics. Indeed, the universal essence of the operator $[\mathscr{Q}]_i$ conveys that the meaning we give to CPR states relies on all of its derivations, whereas the existential essence of the operator $\langle\mathscr{Q}\rangle_i$ makes explicit the fact that the classical meaning states are dependent on the existence of a successful derivation. This semantic subtlety is fundamental for CPR completeness (Lemma 3.7).

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

Example 3.5
Let $\mathscr{C}$ be the usual constraints theory over integers and $\mathscr{P}2$ be the program:

$$q(X) \Longrightarrow q(X+1) \qquad\qquad q(0) \Longrightarrow \mathbf{false}$$

The g.f.p. of $\lambda\mathscr{X}.\big([P_2]_a(\mathscr{X}) \cap \big(\Sigma \setminus [\![\langle\emptyset;\mathbf{false};\emptyset\rangle]\!]_a\big)\big)$ is the set of consistent states that does not contain a CHR constraint $p(X)$, where X is negative or null. Note that the empty set is also a fixpoint but not the greatest. Note that states such as $\langle q(1);\mathbf{true};\emptyset\rangle$, which are in the g.f.p., have only infinite derivations.

We next give a theorem that states the accuracy of 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 are provided for launching each rule of derivation. To state the theorem we assume the following notation ($n \cdot \mathbb{B}$ stands for the scalar product of the multiset $\mathbb{B}$ by n):

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

Theorem 3.6 (*Soundness and completeness of coinductive semantics for CPR*)
Let $\mathscr{P}$ be a CPR program and n be some integer greater than the maximal number of constraints occurring in the head of any rule of $\mathscr{P}$. We have:

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

The proof of the theorem is based on the following completeness lemma that ensures that to each intuitionistic deduction in the theory $\mathscr{C}\mathscr{P}$ corresponds a CPR derivation with respect to the program $\mathscr{P}$. In the following we use the notation $\mathscr{T} \models \mathbb{E} \rightarrow \exists X.\mathbb{F}$ and $\mathbb{E} \vdash_{\mathscr{T}} \exists X.\mathbb{F}$ to emphasize that a deduction within a theory $\mathscr{T}$ is done in the classical logic model framework and the intuitionistic proof framework, respectively. Fortunately, both lines of reasoning coincide in our setting. This remarkable property is due to the fact we are reasoning in a fragment of classical logic, known as *coherent logic*, where classical provability coincides with intuitionistic provability.

Lemma 3.7 (*Intuitionistic completeness of CPR*)
Let $\mathscr{P}$ be a CPR program and n be some integer greater than the maximal number of constraints occurring in the heads of $\mathscr{P}$. Let $\mathbb{E}$ and $\mathbb{F}$ (respectively $\mathbb{C}$ and $\mathbb{D}$) be two multiset of CHR (respectively built-in) constraints. If $\mathbb{E} \wedge \mathbb{C} \vdash_{\mathscr{C}\mathscr{P}} \exists \bar{X}.(\mathbb{F} \wedge \mathbb{D})$, then there exist $\mathbb{F}'$ and $\mathbb{D}'$ such that $\langle n \cdot \mathbb{E}; \mathbb{C}; \text{fv}(\mathbb{F}, \mathbb{C})\rangle \overset{Pn}{\longrightarrow}{}^{*}_{a} \left\langle n \cdot \mathbb{F}'; \mathbb{D}'; \bar{Y} \right\rangle$ and $\mathbb{F}' \wedge \mathbb{D}' \vdash_{\mathscr{C}} \exists \bar{X}.(\mathbb{F} \wedge \mathbb{C})$.

Proof sketch
By structural induction on the proof tree π of $\mathbb{E} \wedge \mathbb{C} \vdash_{\mathscr{C}\mathscr{P}} \exists \bar{X}.(\mathbb{F} \wedge \mathbb{D})$. For the case where π is a logical axiom, we use the reflexivity of $\overset{Pn}{\longrightarrow}{}^{*}_{a}$. For the case where π is a non-logical axiom from $\mathscr{C}$, we use the definition of $\equiv_a$. For the case where π is a non-logical axiom corresponding to a propagation rule $\mathbb{K} \Longrightarrow \mathbb{G} \mid \mathbb{B}_c, \mathbb{B}_b$, we choose $\mathbb{F}' = (\mathbb{E}, \mathbb{B}_c)$ and $\mathbb{D}' = (\mathbb{C} \wedge \mathbb{G} \wedge \mathbb{B}_b)$, and apply r. For the case where π ends with a cut or a right introduction of a conjunction, we use induction hypothesis and the fact that constraints are never consumed along a CPR derivation. Other cases are more straightforward. □

Proof sketch of Theorem 3.6
As we have noted previously, $\mathscr{F}^{\mathscr{C}}_{\text{co}}(\mathscr{P}^n)$ is the set of states that cannot be rewritten to an inconsistent states. Hence, it is sufficient to prove

$$\left\langle n \cdot \mathbb{E}; \mathbb{C}; \bar{X} \right\rangle \overset{\mathscr{P}^n}{\not\longrightarrow}{}^{*}_{a} \langle \emptyset; \textbf{false}; \emptyset \rangle \text{ if and only if } \mathscr{C}\mathscr{P} \not\models \neg\exists(\mathbb{E} \wedge \mathbb{C})$$

or equivalently the contrapositive

$$\mathscr{C}\mathscr{P} \models \forall((\mathbb{E} \wedge \mathbb{C}) \rightarrow \textbf{false}) \text{ if and only if } \left\langle n \cdot \mathbb{E}; \mathbb{C}; \bar{X} \right\rangle \overset{\mathscr{P}^n}{\longrightarrow}{}^{*}_{a} \langle \emptyset; \textbf{false}; \emptyset \rangle.$$

The "if" direction is direct by soundness of CHR. For the "only if" direction, as $\mathscr{C}\mathscr{P}$ is a *coherent logic theory* (i.e., a set of formulas of the form $\forall\big(\mathbb{F} \rightarrow \exists \bar{X}.\mathbb{F}'\big)$, where both $\mathbb{F}$ and $\mathbb{F}'$ are conjunctions of atomic propositions), it can be assumed without loss of generality that $\mathbb{E} \wedge \mathbb{C} \vdash_{\mathscr{C}\mathscr{P}} \textbf{false}$ (see Bezem and Coquand's (2005) work about coherent logic). The result is then directby Lemma 3.7. □

The coinductive semantics for CPR has strong similarities with the fixpoint semantics of CLP (Jaffar and Lassez 1987). Both are defined by fixpoints of some 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, as the logical semantics we characterize is not compositional. Indeed, if the logical readings of two states are independently consistent, one cannot ensure that their conjunctions are also independently consistent. It should be noticed that this non-compositionality prevents the immediate cause operators to be defined over the $\mathscr{C}$-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 $\overset{\mathscr{P}}{\to}_a$, 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 *et al.* (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). As they act as linear logic statements (i.e., they are consumable and their multiplicity matters), usual CHR constraints are called *linear*. As it combines persistent and linear constraints in a slightly less transparent way than $\omega_!$, we call this semantics *hybrid*, and note it ω_h. Due to the space limitations, the (nontrivial) proofs of the section are omitted, but can be found in the extended version of this paper.

4.1 Hybrid operational semantics ω_h

On the contrary of $\omega_!$, the kind of constraint (linear or persistent) in ω_h is not dynamically determined according to the type of rules from which it was produced, but statically fixed. Hence, we will assume that the set of CHR constraint symbols is divided in two: the *linear* symbols and the *persistent* symbols. Naturally, CHR constraints built from the linear (respectively persistent) symbols are called linear (respectively 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 the set of *purely persistent* states by Σ_p (i.e., states of the form $\langle \mathbb{P}; \mathbb{C}; \bar{X} \rangle$, where $\mathbb{P}$ is a set of persistent constraints). $\mathscr{P}^s$ will refer to the set of simplification rules of a hybrid program $\mathscr{P}$.

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:

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

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

The hybrid programs are programs where propagation rules "commute" with simplification rules in the sense of abstract rewriting systems (Terese 2003). 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, it will be so for ever. Of course, numbers of CHR programs 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 Theorem 4.8).

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 that we have defined in the previous section.

Definition 4.2 (*Hybrid transition system semantics*)
The hybrid (transition system) semantics of a hybrid program $\mathscr{P}$ is defined as

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

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 *et al.* (1999) or by adequately instanstiating the notion of abstract critical pair that we have proposed in a previous work (Haemmerlé and Fages 2007).

Definition 4.3 (*Data-sufficient state*)
A hybrid state σ is *data-sufficient* with respect to a hybrid program $\mathscr{P}$ if any state σ' accessible from σ can be simplified (i.e., rewritten by $\mathscr{P}^s$) into a purely persistent sate (i.e., for any state $\sigma' \in \Sigma$, if $\sigma \xrightarrow{\mathscr{P}}_h^* \sigma'$, then there exists a state $\sigma'' \in \Sigma_p$ s.t. $\sigma' \xrightarrow{\mathscr{P}^s}_h^* \sigma''$).

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

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 $\mathcal{C} \models \exists(c{=}d)$,
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 $(p{::}\ c_1, \ldots, c_m\backslash\mathbb{H}{\Longleftrightarrow}\mathbb{G}|\mathbb{B}{,}\mathbb{L}, d_1, \ldots d_n)$ by
$(p :: a(X_1,\ \ _1), \ldots, a(X_m, c_m)\cap\mathbb{H} \Longleftrightarrow \mathbb{G} \mid \mathbb{B}_l, \mathbb{L}, f(d_1), \ldots, f(d_n))$,
where $x_1, \ldots x_m$ are pairwise distinct variables.

Step 3. Add to $\mathcal{P}^{\diamond}$ the rules:
$$\begin{array}{rcrcl}
1 & :: & \textit{stamp} & @ & f(X), c_f(Y) \Longleftrightarrow f(Y, X), c_f(Y+1). \\
2 & :: & \textit{set} & @ & a(Y, X) \setminus a(Z, X) \Longleftrightarrow Y < Z \mid \top\cdot \\
5 & :: & \textit{unfreeze} & @ & f(Y, X), c_a(Y) \Longleftrightarrow a(Y, X), c_a(Y+1).
\end{array}$$

Fig. 1. Source-to-source translation for hybrid programs.

Theorem 4.4 (*Soundness and completeness of hybrid transition system semantics*)
Let $\mathscr{P}$ be a hybrid program such that $\mathscr{P}^s$ is confluent. Let $\langle \mathbb{L}, \mathbb{P}; \mathbb{E}; \bar{X}\rangle$ be a data-sufficient state with respect to $\mathscr{P}$. We have

$$\langle \mathbb{L}, \mathbb{P}; \mathbb{E}; \bar{X}\rangle \in \mathscr{F}^{\mathscr{C}}_{\mathrm{h}}(\mathscr{P}) \text{ if and only if } (\mathbb{L} \wedge \mathbb{P} \wedge \mathbb{E}) \text{ is satisfiable within } \mathscr{C}\mathscr{P}.$$

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 implemented efficiently.

Definition 4.5 (*Propagation fair derivation*)
A derivation $\sigma_0 \overset{\mathscr{P}}{\rightarrow}_h \sigma_1 \overset{\mathscr{P}}{\rightarrow}_h \ldots$ is propagation fair if for any propagation redex $r@\langle \mathbb{K}; \mathbb{E}; \bar{X}\rangle$ of a state s_i in the derivation, there exist two states s_j and s_{j+1} such the transition from σ_j to σ_{j+1} is a propagation application where the reduced redex is identical or stronger to $r@\langle \mathbb{K}; \mathbb{E}; \bar{X}\rangle$ (i.e., the reduced redex is of the form $r@\langle \mathbb{K}; \mathbb{F}; \bar{Y}\rangle$ with $\mathscr{C} \models \exists_{\text{-}\bar{Y}}\mathbb{F} \rightarrow \exists_{\text{-}\bar{X}}\mathbb{E}$).

Proposition 4.6 (*Soundness and completeness of propagation fair derivations*)
Let $\mathscr{P}$ be a hybrid program such that $\mathscr{P}^s$ is confluent and terminating. Let σ be a data-sufficient state. $\sigma \in \mathscr{F}^{\mathscr{C}}_{\mathrm{h}}(\mathscr{P})$ holds if and only if there is a consistent propagation fair derivation starting from σ.

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 $\mathscr{P}$ that we consider. The implementation of a hybrid program $\mathscr{P}$ consists in a source-to-source translation $\mathscr{P}^{\diamond}$ intended to be executed in 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 $\langle \mathbb{L}, d_1, \ldots d_n; \mathbb{D}; \mathbb{V}\rangle^{\diamond} = \langle\!\langle \mathbb{L}, f(d_1), \ldots, f(d_n), c_f(0), c_a(0); \emptyset; \mathbb{D}; \emptyset\rangle\!\rangle^{\mathbb{V}}_0$.

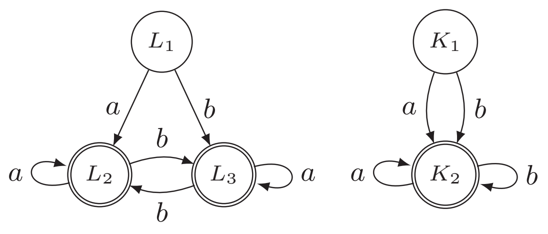
Before going further, let us give some intuition about the behavior of translation. If a rule needs two occurrences of the same persistent constraint, Step 1 will insert an equivalent rule that needs only one occurrence of the constraint. In the translation each persistent constraint can be applied in three different successive states: *fresh*, indicated by $f/1$, *frozen*, indicated by $f/2$, and *alive*, indicated by $a/2$. On the one hand, Step 2 ensures that only alive constraints can be used to launch a rule, and on the other hand the persistent constraints on 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 the 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 following two theorems state that our implementation is sound and complete with respect to failure. Furthermore, Theorem 4.7 shows 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, as 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 4.7 (*Soundness w.r.t. success and failure*)
Let $\mathscr{P}$ be a hybrid program such that $\mathscr{P}^s$ is confluent, and let $\langle \mathbb{C}; \mathbb{E}; \bar{X}\rangle^\diamond \rightarrow^*_{\mathscr{P}^\diamond} \langle\!\langle \emptyset; \mathbb{D}; \mathbb{F}; T\rangle\!\rangle_i^{\bar{Y}} \not\rightarrow_{\mathscr{P}^\diamond}$ be a terminating derivation. $\langle \mathbb{C}; \mathbb{E}; \bar{X}\rangle \in \mathscr{F}_{\mathrm{h}}^{\mathscr{C}}(\mathscr{P})$ holds if and only if $\mathbb{F}$ is satisfiable within $\mathscr{C}$.

Theorem 4.8 (*Completeness w.r.t. failure*)
Let $\mathscr{P}$ be a hybrid program such that $\mathscr{P}^s$ is confluent and terminating. Let $\langle \mathbb{C}; \mathbb{E}; \bar{X}\rangle$ be a data-sufficient state. If $\langle \mathbb{C}; \mathbb{D}; \bar{X}\rangle \notin \mathscr{F}_{\mathrm{h}}^{\mathscr{C}}(\mathscr{P})$, then any concrete derivation starting from $\langle \mathbb{C}; \mathbb{D}; \bar{X}\rangle^\diamond$ fails finitely.

The implementation that we propose has strong connections with the co-Selective Linear Definite (SLD), an implementation of the g.f.p. semantics of Logic Programming proposed by Simon *et al.* (2006). Both are based on a dynamic synthesis of coinductive hypothesises and a cycle detection in proofs. But, as 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 w.r.t. the logical reading of a program that causes the interpreter to loop). That contrasts with CHR, where any coherent constraint system can be used without loosing logical completeness with respect to failures.



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

Fig. 2. A binary automaton and its CPR representation.

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

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

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

> In order to prove the equality of languages recognized by two states K and L, it is sufficient to establish the existence of a bisimulation relation in $\mathscr{L}$ 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

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

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 3.6, it is simple to prove that the two states of coalgebra represented by $\mathbb{D}$ accept or reject 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 us 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 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 possible implementation of the destructor function[3]:

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

where $or/3$ unifies its third argument with the Boolean disjunction of its first two 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:

$$L\sim K \implies nonvar(L), nonvar(K) \mid f(L,(T,L_a,L_b)), f(K,(T,K_a,K_b)), L_a\sim K_a, L_b\sim K_b.$$

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

[3] A complete version of program can be found in technical report version of the paper.

leads to an irreducible consistent state. This implies thanks to the Coinductive Proof Principle, together with Theorems 4.4 and 4.7 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 in order to avoid rapid saturation of 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, as the program is deterministic; on the other hand, termination of the set of simplification rules, which is required by Theorem 4.8, can be easily established by using, for instance, techniques we have recently proposed for single-headed programs (Haemmerlé *et al.* 2011).

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 naturally casted in CHR. Moreover, it is worth noticing that the program given has the properties required by a constraint solver. Firstly, the program is effective, i.e., it can actually prove or disprove that two expressions are equal. The first part of the claim can be proved by using the Kleene theorem (Rutten 1998) and the idempotency and commutativity of the alternation, enforced here by the *merge/3* predicate. The second part is directed 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 done previously that a^+ and (a, a^*) recognize the same language while a^+ and a^* do not:

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

6 Conclusion

We have defined an 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 given by Betz et al. (2010), we were able to characterize CHR programs combining both simplification and propagation rules by a fixpoint semantics without losing completeness with respect 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 that 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 *et al.* (2000) may suggest it should be possible to improve completeness with respect to success of the implementation proposed here.

Acknowledgements

The research leading to these results has received funding from the Madrid Regional Government under the CM project P2009/TIC/1465 (PROMETIDOS), the Spanish Ministry of Science under the MEC project TIN-2008-05624 *DOVES*, and the European Seventh Framework Programme FP/2007-2013 under grant agreement 215483 (S-CUBE).

We are grateful to César Sánchez for interesting discussions about coinduction, and to Santiago Zanella Béguelin for referring us to Bezem and Coquand's work. At the end, we would like to thank reviewers for their helpful and constructive comments.

References

Abdennadher, S. 1997. Operational semantics and confluence of Constraint Propagation Rules. In *Third International Conference on Principles and Practice of Constraint Programming (CP)*, LNCS, vol. 1330. Springer, Berlin, 252–266.

Abdennadher, S., Frühwirth, T. and Meuss, H. 1999. Confluence and semantics of Constraint Simplification Rules. *Constraints 4*, 2, 133–165.

Barwise, J. and Moss, L. 1996. *Vicious Circles*. CSLI, Stanford, CA.

Betz, H., Raiser, F. and Frühwirth, T. 2010. A complete and terminating execution model for Constraint Handling Rules. *Theory and Practice of Logic Programming 10*, special issues 4–6 (ICLP), 597–610.

Bezem, M. and Coquand, T. 2005. Automating coherent logic. In *International Conferences on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, LNCS, vol. 3835. Springer, Berlin, 246–260.

Clarke, E. M., Grumberg, O. and Peled, D. A. 2000. *Model Checking*. MIT Press, Cambridge, MA.

de Koninck, L., Schrijvers, T. and Demoen, B. 2007. User-definable rule priorities for CHR. In *9th International Conference on Principles and Practice of Declarative Programming*, July 14–16, Wroclaw, Poland(*PPDP*). ACM, New York, 25–36.

Frühwirth, T. 1998. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming 37*, 1–3, 95–138.

Frühwirth, T. 2009. *Constraint Handling Rules*. Cambridge University Press, Cambridge, UK.

Gabbrielli, M. and Meo, M. 2009. A compositional semantics for CHR. *ACM Transactions Computer Logic 10*, 2.

Gabbrielli, M., Meo, M. and Tacchella, P. 2008. A compositional semantics for CHR with propagation rules. In *Constraint Handling Rules: Current Research Topics*. LNAI, vol. 5388. Springer, Berlin, 119–160.

Goguen, J. A., Lin, K. and Rosu, G. 2000. Circular coinductive rewriting. In *Proceedings of the 15th IEEE International Conference on Automated Software Engineering*. ASE, Washington, DC, 123–132.

Haemmerlé, R. 2011. *Toward Logically Complete Fixpoint Semantics for Constraint Hangling Rules*. Technical Report CLIP3/2011, Technical University of Madrid, Madrid, Spain.

Haemmerlé, R. and Fages, F. 2007. Abstract critical pairs and confluence of arbitrary binary relations. In *Conference on Rewriting Techniques and Applications*, LNCS, vol. 4533. Springer, New York, 214–228.

Haemmerlé, R., Lopez-Garcia, P. and Hemenegildo, M. V. To appear. CLP projection for constraint handling rules. In *Conference on Principles and Practice of Declarative Programming(PPDP)*. ACM, New York.

Jaffar, J. and Lassez, J.-L. 1987. Constraint logic programming. In *Symposium on Principles of Programming Languages* (*POPL*). ACM, New York, 111–119.

Lloyd, J. 1987. *Foundations of Logic Programming*. Springer, Berlin.

Raiser, F., Betz, H. and Frühwirth, T. 2009. *Equivalence of CHR States Revisited*. CHR Report CW 555. Katholieke University, Leuven, Belgium, 34–48.

Rutten, J. 1998. Automata and coinduction (an exercise in coalgebra). In *Proceedings of the Ninth International Conference on Concurrency Theory* (*CONCUR*), LNCS, vol. 1466. Springer, New York, 194–218.

Saraswat, V. A., Rinard, M. C. and Panangaden, P. 1991. Semantic foundations of concurrent constraint programming. In *Proceedings of Principles of Programming Languages* (*POPL*). ACM, New York, 333–352.

Simon, L., Mallya, A., Bansal, A. and Gupta, G. 2006. Coinductive logic programming. In *International Conference on Logic Programming* (*ICLP*), LNCS, vol. 4079. Springer, Berlin, 330–345.

Tarski, A. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics 5*(2), 285–309.

Terese. 2003. *Term Rewriting Systems*, Vol. 55. Cambridge University Press, Cambridge, UK.