CLP Projection for Constraint Handling Rules

Rémy Haemmerlé
Technical University of Madrid

Pedro Lopez-Garcia
IMDEA Software Institute and
Spanish National Research Council

Manuel V. Hermenegildo
IMDEA Software Institute and
Technical University of Madrid

Abstract

This paper introduces and studies the notion of CLP projection for Constraint Handling Rules (CHR). The CLP projection consists of a naive translation of CHR programs into Constraint Logic Programs (CLP). We show that the CLP projection provides a safe operational and declarative approximation for CHR programs. We demonstrate moreover that a confluent CHR program has a least model, which is precisely equal to the least model of its CLP projection (closing hence a ten year-old conjecture by Abdenader et al.). Finally, we illustrate how the notion of CLP projection can be used in practice to apply CLP analyzers to CHR. In particular, we show results from applying AProVE to prove termination, and CiaoPP to infer both complexity upper bounds and types for CHR programs.

Keywords CHR, CLP, Declarative Semantics, Static Analysis.

Categories and Subject Descriptors F.3 [*Theory of Computation*]: Logic and Meaning of Programs

General Terms Theory

1. Introduction

Constraint Handling Rules (CHR) [7] is a concurrent, committed-choice, rule-based programming language introduced in the 1990s by Frühwirth. CHR was originally designed for the design and implementation of constraint solvers, initially in the context of Constraint Logic Programming (CLP) [15], but it has since come into use as a general-purpose concurrent programming language.

It is well-known that CLP can be encoded into CHR (see, for instance Section 6.3.1 in Frühwirth's Book [7]). Operationally the encoding is sound and complete. From the logical point of view the encoding is an under-approximation, since the CHR encoding in CLP corresponds to the Clark completion [3]. Conversely, CLP has been from the beginning an implementation vehicle for CHR programs [6, 13, 22], since, as mentioned before, one of the initial objectives of CHR was precisely to encode constraint solving algorithms meant to run within CLP systems. However these translations are really too low-level code, typically using attributed variables [12].

However, and perhaps surprisingly, few attempts can be found in the literature to perform a direct translation of CHR into (pure) CLP. Such an encoding can be interesting in order to relate the CLP and CHR theoretical foundations, and to be able to use the many tools available for the semantic analysis of CLP programs in the context of CHR.

With this objective in mind in this paper we introduce the notion CLP projection. CLP projection consists of a naive translation of CHR programs into (pure) CLP. We show that CLP projection provides a safe operational and declarative approximation for CHR programs. In particular, we show that:

- A CHR program is operationally simulated by its CLP projection.
- The logical models of a CHR program are under-approximated by the least model of its projection. We show moreover that the least model of a confluent CHR program is precisely the least model of its CLP projection (closing hence a ten year-old conjecture by Abdenader et al. [1]).
- The success set with respect a CHR program can be charaterized by the successes of its projection.

Finally, we also illustrate how the notion of CLP projection can be used in practice to apply CLP analyzers to CHR. In particular, we show results from applying the AProVE analyzer [8] to prove termination, and the Ciao preprocessor (CiaoPP) [11] to infer both complexity upper bounds and types for CHR programs.

To the best of our knowledge the only attempt to translate CHR programs into Prolog is the so-called transformational approach of Pilozzi's et al [18, 19]. It consists of a Prolog meta-interpreter that preserves store accessibility. As the CLP projection, it provides an over-approximation of the CHR operational semantics, and has been used to prove termination and to infer types for CHR programs. The meta-level nature of this approach has the main advantage of making the notion of user-defined store explicit, but it also makes the task for Prolog/CLP analyzers much more complex, since it is a well-known fact that high levels of meta-interpretation can result in loss of precision for analyzes based on approximations. Furthermore it seems more difficult to relate the declarative semantics using a meta-interpreted approach.

The rest of the paper is structured as follows: Section 2 recalls basic notation, definitions, and results for fixpoints, reduction, and first-order logic. Then, Section 3 presents the syntax and both the operational and the declarative semantics for CLP frameworks. Section 4 similarly presents the CHR framework. Then, Section 5 formally introduces the notion of CLP projection. In Section 6, we illustrate the relevance of the CLP projection approach for the static analysis of CHR programs through different applications ranging from termination proofs to type inference through complexity upper bounds. Finally, in Section 7 we present our conclusions.

2. Preliminaries

In this section, we recall the theoretical framework of CLP.

2.1 Notations

We assume as given a denumerable set $\mathcal{V}$ of variables (denoted by X, Y, Z ...), a denumerable set Σ_f of function and constant symbols, and a set of predicate symbols Σ_p (denoted by characters or words in teletype font, such as c or p). Symbols of both kinds are assumed given with their respective arity. The set of first order terms built from $\mathcal{V}$ and Σ_f will be denoted by T, its elements by $t, s, \ldots$. Sets (resp. sequences) of variables and terms will be distinguished by a bar (resp. arrow) above, as, for instance, $\bar{X}$ and $\bar{t}$ (resp. $\vec{X}$ and $\vec{t}$). Atomic propositions built from T and Σ_p are denoted by $a, b, c, d, \ldots$. By a slight abuse of notation we will use interchangeably conjunction and multiset of atomic propositions, forget braces around multisets, and use comma for multiset union. Conjunctions and multisets will be denoted by capital blackboard letters such that $\mathbb{A}$ or $\mathbb{C}$.

For an arbitrary formula ϕ, we use $\mathrm{fv}(\phi)$ to denote the set of free variables occurring in ϕ, and $\phi[\vec{X}\backslash\vec{t}]$ to represent ϕ in which the free occurrences of variables $\vec{X}$ have been replaced by terms $\vec{t}$ (with the usual renaming of bound variables, avoiding variable clashes). The notation $\exists_{\setminus\psi}\phi$ denotes the existential closure of ϕ with the exception of variables free in the formula ψ, which remain free.

In this paper, we assume that the set of predicate symbols Σ_p is partitioned into two: Σ_b, the set of *(built-in) constraint* symbols, Σ_a the set of *(user-defined) atom* symbols. Naturally, atomic propositions built from Σ_b will be called *(built-in) constraints* while atomic propositions built from Σ_b will be called *(user-defined) atoms*. For constraints we assume given a consistent (first order) axiomatic theory $\mathcal{C}$ describing their meaning.

2.2 Preliminaries on Fixpoints

Here, we recall some definitions and results about fixpoints in an arbitrary complete lattice $(\mathcal{L}, \supseteq, \cap, \cup, \top, \bot)$. We will say that a function $f : \mathcal{L} \to \mathcal{L}$ is *monotonic* if $f(\mathcal{X}) \supseteq f(\mathcal{Y})$ whenever $\mathcal{X} \supseteq \mathcal{Y}$. The *upward (ordinal) power* of a function $f : \mathcal{L} \to \mathcal{L}$ is defined by the transfinite induction:

- $f{\uparrow}0 = \bot$
- $f{\uparrow}\alpha = f(f{\uparrow}(\alpha - 1))$ if α is a successor ordinal,
- $f{\uparrow}\alpha = \bigcup\{f{\uparrow}\beta \mid \beta < \alpha\}$ if α is a limit ordinal.

An element $\mathcal{X} \in \mathcal{L}$ is a *fixpoint* for $f : \mathcal{L} \to \mathcal{L}$ if $f(\mathcal{X}) = \mathcal{X}$. $\mathcal{X}$ is a *least fixpoint* for f if it is a fixpoint and $\mathcal{Y} \supseteq \mathcal{X}$ whenever $\mathcal{Y}$ is a fixpoint for f. We use $\mu\mathcal{X}.f(\mathcal{X})$ to denote the least fixpoint.

Theorem 1 (Knaster–Tarski)**.** *If f is a monotonic function on $\mathcal{L}$, then f has a least fixpoint. Furthermore there exists a limit ordinal α such that:*

$$\mu X.f(X) = \bigcap\{X \in \mathcal{L} | X = f(X)\} = f{\uparrow}\alpha.$$

2.3 Preliminaries on Reductions

A *reduction* is a binary relation $\to$ defined over some given set $\mathcal{A}$. Let assume some reductions $\to$, $\twoheadrightarrow$ defined over the same set $\mathcal{A}$. We shall use the following notations and definitions:

- $\circ$ is the composition : $(\to \circ \twoheadrightarrow) = \{(a, b) \mid \exists c \in \mathcal{A}\ (a \to c \wedge c \twoheadrightarrow b)\}$;
- $\to^0 = \{a \to a \mid a \in \mathcal{A}\}$ and $\to^n = \to \circ \to^{n-1}$ for $n \geq 1$;
- $\to^* = \cup_{i \geq 0} \to^i$ is the transitive-reflexive closure of $\to$;
- $\to$ is *terminating* if there is no infinite sequence $e_0 \to e_1 \to \ldots$;
- $\to$ is *confluent* if for any element $a, b, c \in \mathcal{A}$ such that $a \to^* b$ and $a \to^* c$ there exists an element $d \in \mathcal{A}$ such that $b \to^* c$ and $c \to^* d$.

2.4 Preliminaries on First order logic

In this subsection we recall some basics about model theoretic semantics of first order logic.

2.4.1 First order models

Let $\mathcal{L}_p$ be the first order language built from the set T of first order terms and the set of predicate symbols Σ_p. An *interpretation* of $\mathcal{L}_p$ is a tuple $I = \langle D, [\,]\rangle$, composed of an *interpretation domain* D together with a semantics function $[\,]$, which associates to each function symbol $f \in \Sigma_f$ of arity m a function $[f] : D^m \to D$, and to each predicate symbol $p \in \Sigma_p$ of arity n a function $[p] : D^n \to \{\top, \bot\}$. For a given interpretation I, an *I-valuation* is a function $\rho : \mathcal{V} \to D$. An *I-instance* of a term t (resp. a formula ϕ) is the tuple $t\rho$ (resp. $\phi\rho$), where ρ is an I-valuation.

Let I be an interpretation of $\mathcal{L}_p$. The *assignment* (with respect I) of *I-instances* of terms and atomic propositions in $\mathcal{L}_p$ is the function $[\,]_I$ defined by structural induction as:

- $[X\rho]_I = \rho(X)$ if $X \in \mathcal{V}$;
- $[f(t_1, \ldots, t_n)\rho]_I = [f]([t_1\rho]_I, \ldots, [t_n\rho]_I)$ if $f \in \Sigma_f$;
- $[\mathtt{c}(t_1, \ldots, t_n)\rho]_I = [\mathtt{c}]([t_1\rho]_I, \ldots, [t_n\rho]_I)$ if $\mathtt{c} \in \Sigma_p$.

The assignment (with respect I) is extended to logical formulas in $\mathcal{L}_p$ by applying the truth table of the logical connectors and the following rules for the quantifiers:

- $[(\forall X\phi)\rho]_I = \top$ if and only if for any element $d \in D$, $[\phi(\rho \circ [X\backslash d])]_I = \top$;
- $[(\exists X\phi)\rho]_I = \top$ if and only if there exists an element $d \in D$, $[\phi(\rho \circ [X\backslash d])]_I = \top$.

An interpretation I of $\mathcal{L}_p$ is a *model* for a formula $\phi \in \mathcal{L}_p$, if for all I-valuations ρ, $[\phi\rho]_I = \top$. Naturally, an interpretation I of $\mathcal{L}_p$ is a *model* of a theory $\mathcal{T}$ if I is a model of all of its axioms. A formula ϕ is *satisfiable within a theory $\mathcal{T}$* (or, more briefly, *$\mathcal{T}$-satisfiable*) if there is a model of $\mathcal{T}$ which is a model of $\exists\phi$ as well. In the following, we use the notation $\mathcal{T} \models \phi$ to mean that any model of a theory $\mathcal{T}$ is as well a model of the formula ϕ.

2.4.2 Model with respect a constraint theory

In this subsection, we introduce the classical notion of constrained atoms. Sets of constrained atoms will be called $\mathcal{C}$-interpretations. For a given interpretation limited to constraints, a $\mathcal{C}$-interpretation represents an interpretation for the whole set of propositions (including both constraints and atoms). $\mathcal{C}$-interpretations have the advantage with respect classical interpretations as presented in the previous section that they are sets of syntactic objects while classical interpretations are not (for instance the domain of real numbers contains elements which cannot be represented syntactically). Hence, it appears that in a large number of cases manipulating $\mathcal{C}$-interpretations is simpler than manipulating the classical ones.

Definition 2 ($\mathcal{C}$-base)**.** *A* constrained atom *is a pair $(a|\mathbb{C})$, where a is a user-defined atom and $\mathbb{C}$ is a conjunction of built-in constraints. The set of constrained atoms is called* $\mathcal{C}$-base *and is denoted by $\mathcal{B}_\mathcal{C}$.*

For the sake of simplicity, we will work always with sets of constrained atoms closed by the closure operator $\Uparrow^\mathcal{C}$ defined next. This operator returns the set of all atoms more constrained than the one given as its input.

Definition 3. *The closure operator* $\Uparrow^{\mathcal{C}} : 2^{\mathcal{B}_\mathcal{C}} \to 2^{\mathcal{B}_\mathcal{C}}$ *is defined as:*

$$\Uparrow^{\mathcal{C}}(\mathcal{Z}) = \{(a|\mathbb{C}) \in \mathcal{B}_\mathcal{C} \mid (b|\mathbb{D}) \in \mathcal{Z} \text{ and } \mathcal{C} \models \mathbb{C} \to \exists_{\text{-}a}(a = b \wedge \mathbb{D})\}$$

In the following, we use the notation $(a_1, \ldots, a_n|\mathbb{C})$ *for the set of constrained atoms* $\{(a_1|\mathbb{C}), \ldots (a_n|\mathbb{C})\}$.

Example 4. *Assume* $\mathcal{C}$ *defines the order* $<$ *on integers. Let* $\mathtt{p}$ *and* $\mathtt{q}$ *be two constrained symbols of arity* 1 *and let* $\mathcal{Z} = \Uparrow^{\mathcal{C}}(\mathtt{p}(X)|0<X \wedge X<4)$. *For instance, we have:*

- $(\mathtt{p}(1)|\top)$, $(\mathtt{p}(Y)|1<Y \wedge X=Y \wedge X<4)$, *and* $(\mathtt{q}(Y)|\bot)$ *are in* $\mathcal{Z}$.
- *neither* $(\mathtt{p}(5)|\top)$ *nor* $(\mathtt{p}(Y)|Y>5)$ *are in* $\mathcal{Z}$.

We define the $\mathcal{C}$-model of a formula ϕ, as a set of constrained atoms that validate ϕ without contradicting any model of $\mathcal{C}$.

Definition 5 ($\mathcal{C}$-model)**.** *For a given interpretation* I *of* $\mathcal{C}$*, the assignment of* I*-instances associated to a set* $\mathcal{Z}$ *of constrained atoms is defined as:*

$$[\![\mathcal{Z}]\!]_I(a\rho) = \begin{cases} [\mathbb{C}\rho]_I & \text{if } (a|\mathbb{C}) \in \mathcal{Z} \\ \bot & \text{otherwise} \end{cases}$$

A set $\mathcal{Z}$ *of constrained atoms is a* $\mathcal{C}$-model *of a first order formula* ϕ *in* $\mathcal{L}_p$*, if for any model* $I = \langle D, [\,]\rangle$ *of* $\mathcal{C}$*,* $\langle D, [\,] \cup [\![\mathcal{Z}]\!]_I\rangle$ *is a model of* ϕ.

Obviously, a formula is satisfiable in any interpretation of $\mathcal{C}$ if and only if it has a $\mathcal{C}$-model. The following technical lemma will be useful later to prove that a set of constrained atoms is a model of an implication.

Lemma 6. *Let* $\mathcal{Z}$ *be a set of constrained atoms,* $\mathbb{A}$ *and* $\mathbb{B}$ *be two conjunctions of user-defined atoms,* $\mathbb{C}$ *and* $\mathbb{D}$ *be two sets of built-in constraints, and* $\vec{X}$ *a sequence of variables not free in* $(\mathbb{A} \wedge \mathbb{C})$*. If for any conjunction* $\mathbb{E}$ *of built-in constraints satisfying* $\vec{X} \cap \text{fv}(\mathbb{E}) = \emptyset$*,* $(\mathbb{A}|\mathbb{C} \wedge \mathbb{E}) \in \mathcal{Z}$ *implies* $(\mathbb{B}|\mathbb{C} \wedge \mathbb{D} \wedge \mathbb{E}) \in \mathcal{Z}$ *together with* $\mathcal{C} \models (\mathbb{C} \wedge \mathbb{E}) \to \exists\vec{X}(\mathbb{C} \wedge \mathbb{D} \wedge \mathbb{E})$*, then* $\mathcal{Z}$ *is a* $\mathcal{C}$*-model for the implication* $(\mathbb{A} \wedge \mathbb{C}) \to \exists\vec{X}(\mathbb{B} \wedge \mathbb{D})$.

Proof. Let $I = \langle D, [\,]\rangle$ be a model of $\mathcal{C}$ and $[\,]_\mathcal{Z} = ([\,] \cup [\![\mathcal{Z}]\!]_I)$. We have to show that $\langle D, [\,]_\mathcal{Z}\rangle$ is a model for $(\mathbb{A} \wedge \mathbb{C}) \to \exists X.(\mathbb{B} \wedge \mathbb{D})$, that is, for any I-valuation ρ, if $[(\mathbb{A} \wedge \mathbb{C})\rho]_\mathcal{Z} = \top$ then there exists a sequence of terms $\vec{d} \in D$ such that $[(\mathbb{A} \wedge \mathbb{C})(\rho \circ [\vec{X}\backslash\vec{d}])]_\mathcal{Z} = \top$. Let us assume some I-valuation ρ satisfying $[(\mathbb{A} \wedge \mathbb{C})\rho]_\mathcal{Z} = \top$. We have:

$$[(\mathbb{A} \wedge \mathbb{C})\rho]_\mathcal{Z} = \top$$
$$\implies [\mathbb{A}\rho]_\mathcal{Z} = \top \text{ and } [\mathbb{C}\rho]_\mathcal{Z} = \top \quad (1)$$
$$\implies (\mathbb{A}|\mathbb{E}) \in \mathcal{Z},\ [\mathbb{E}\rho]_I = \top, \text{ and } [\mathbb{C}\rho]_I = \top \text{ for some } \mathbb{E} \quad (2)$$
$$\implies [(\mathbb{C} \wedge \mathbb{E})\rho]_I = \top \quad (3)$$
$$\implies (\mathbb{A}|\mathbb{C} \wedge \mathbb{E}) \in \mathcal{Z} \quad (4)$$
$$\implies (\mathbb{B}|\mathbb{C} \wedge \mathbb{E}) \in \mathcal{Z} \text{ and } \mathcal{C} \models (\mathbb{C} \wedge \mathbb{E}) \to \exists\vec{X}(\mathbb{D} \wedge \mathbb{E}) \quad (5)$$
$$\implies \text{there exist } \vec{d} \in D \text{ s.t. } \left[(\mathbb{D} \wedge \mathbb{E})\left(\rho \circ \left[\vec{X}\backslash\vec{d}\right]\right)\right]_I = \top \quad (6)$$
$$\implies \left[\mathbb{B}\left(\rho \circ \left[\vec{X}\backslash\vec{d}\right]\right)\right]_\mathcal{Z} \text{ and } \left[(\mathbb{D} \wedge \mathbb{E})\left(\rho \circ \left[\vec{X}\backslash\vec{d}\right]\right)\right]_\mathcal{Z} = \top \quad (7)$$
$$\implies \left[(\mathbb{B} \wedge \mathbb{F})\left(\rho \circ \left[\vec{X}\backslash\vec{d}\right]\right)\right]_\mathcal{Z} \quad (8)$$

(1) is by definition of $[\,]_\mathcal{Z}$. (2) is by definition of $[\![\mathcal{Z}]\!]_I$. (Without loss of generality we assume $\vec{X} \cap \text{fv}(\mathbb{E}) = \emptyset$.) (3) is by definition of $[\,]_I$. (4) is by definition of $[\![\mathcal{Z}]\!]_I$. (5) is by hypothesis. (6) is because I is a model of $\mathcal{C}$. (7) is by definition of $[\![\mathcal{Z}]\!]_I$. (8) is by definition of $[\,]_\mathcal{Z}$. □

3. Constraint Logic Programming

Here we recall basic definitions and results for CLP.

3.1 Syntax

In CLP, we distinguish two syntactical categories, the clauses that form the programs and the goals that are rewritten by the programs. A *(CLP) clause* is a logical formula of the form:

$$\forall\ (a \vee \neg a_1 \ldots \neg a_m \vee \neg c_1 \vee \ldots \neg c_n),$$

where the a and a_i's are atoms and the c_i's are constraints. This formula is noted in a simpler way as: $(a \leftarrow \mathbb{A} \mid \mathbb{C})$, where $\mathbb{A}$ is the multiset of the a_i's and $\mathbb{C}$ is the conjunction of the c_i's. Empty constraints (i.e., $\top$) can be omitted together with the symbol $|$. A *CLP program* is a finite set of clauses.

A *(CLP) goal* is a logical formula of the form:

$$\forall\ (\neg a_1 \ldots \neg a_n \vee \neg c_1 \vee \ldots \neg c_m),$$

where the a_i's are atoms and the c_i's are constraints. This formula is noted in a simpler way as: $\langle \mathbb{A}|\mathbb{C}\rangle$, where $\mathbb{A}$ is the multiset of the a_i's and $\mathbb{C}$ is the conjunction of the c_i's.

3.2 Operational semantics

The operational semantics of CLP is given by CSLD resolution, that we present briefly in the following.

For a given program $\mathcal{P}$, the transition relation $\xrightarrow{\mathcal{P}}$ over goals is defined as the least relation satisfying the following principle of resolution:

$$\frac{(b \leftarrow \mathbb{B} \mid \mathbb{D}) \in \mathcal{P}\theta \qquad \mathbb{D} \text{ is } \mathcal{C}\text{-satisfiable}}{\langle \mathbb{A}, a|\mathbb{C}\rangle \xrightarrow{\mathcal{P}} \langle \mathbb{A}, \mathbb{B}|\mathbb{C} \wedge a = b \wedge \mathbb{D}\rangle}$$

where θ is a renaming with fresh variables.

We state next a straightforward result about CLP transitions that we will use later in the paper:

Proposition 7.
Let $\langle \mathbb{A}|\mathbb{C}\rangle$*,* $\langle \mathbb{A}'|\mathbb{C}'\rangle$*, and* $\langle \mathbb{B}|\mathbb{D}\rangle$ *be CLP goals such that* $\text{fv}(\mathbb{A}', \mathbb{C}') \cap \text{fv}(\mathbb{B}, \mathbb{D}) \subseteq \text{fv}(\mathbb{A}, \mathbb{C})$.

If $\langle \mathbb{A}|\mathbb{C}\rangle \xrightarrow{\mathcal{P}} \langle \mathbb{A}'|\mathbb{C}'\rangle$ *and* $\mathbb{C} \wedge \mathbb{D}$ *is* $\mathcal{C}$*-satisfiable, then*

$$\langle \mathbb{A}, \mathbb{B}|\mathbb{C} \wedge \mathbb{D}\rangle \xrightarrow{\mathcal{P}} \langle \mathbb{A}', \mathbb{B}|\mathbb{C}' \wedge \mathbb{D}\rangle.$$

A *success* for a CLP program $\mathcal{P}$ is a goal that has a consistent answer with respect $\mathcal{P}$ (i.e., that can be derived by $\mathcal{P}$ to a goal of the form $\langle \emptyset|\mathbb{C}\rangle$ where $\mathbb{C}$ is $\mathcal{C}$-satisfiable).

3.3 Fixpoint semantics

In this section we recall the fixpoint semantics of CLP.

Definition 8 (Immediate Consequence Operator)**.** *For any CLP program* $\mathcal{P}$*, the* immediate consequence operator $T_\mathcal{P}^\mathcal{C} : 2^{\mathcal{B}_\mathcal{C}} \to 2^{\mathcal{B}_\mathcal{C}}$ *is defined as:*

$$T_\mathcal{P}^\mathcal{C}(\mathcal{X}) = \{(a|\mathbb{C} \wedge \mathbb{D}) \in \mathcal{B}_\mathcal{C} \mid (a \leftarrow \mathbb{A} \mid \mathbb{C}) \in \mathcal{P} \wedge (\mathbb{A}|\mathbb{D}) \in \mathcal{X}\}$$

Both $T_\mathcal{P}^\mathcal{C}$ and $\Uparrow^\mathcal{C}$ being obviously monotonous, Tarski's theorem ensures the function $\lambda\mathcal{X}.\Uparrow^\mathcal{C}\left(T_\mathcal{P}^\mathcal{C}(\mathcal{X})\right)$ has a least and greatest fixpoint.

Theorem 9 (Least $\mathcal{C}$-models [15])**.** *Let* $\{M_i\}_{i\in I}$ *be the set of all* $\mathcal{C}$*-models of a CLP programs* $\mathcal{P}$*.* $\mathcal{P}$ *has a least model, which satisfying:*

$$M_\mathcal{P}^\mathcal{C} = \bigcap_{i\in I} M_i = \mu\mathcal{X}.\Uparrow^\mathcal{C}\left(T_\mathcal{P}^\mathcal{C}(\mathcal{X})\right)$$

4. Constraint Handling Rules

In this section we introduce the syntax, the equivalence-based operational semantics ω_e, and the declarative semantics of CHR.

4.1 Syntax

We recall first the syntax of the language.

4.1.1 Programs

A *Constraint Handling Rule* (CHR) is a rule of the form:

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

where $\mathbb{K}$ and $\mathbb{H}$ are multisets of user-defined atoms, called *kept head* and *removed head* respectively, $\mathbb{G}$ is a conjunction of built-in constraints called *guard*, $\mathbb{C}$ are conjunctions of built-in constraints, $\mathbb{A}$ are multisets user-defined atoms, and r is an arbitrary identifier assumed unique in the program and called *rule name*. Rules where both heads are empty are prohibited. The empty guard $\top$ can be omitted together with the symbol $|$. The *local variables* of the rule are the variables occurring in the guard and in the body but not in the head (i.e., $\mathrm{lv}(r) = \mathrm{fv}(\mathbb{G}, \mathbb{C}, \mathbb{B}) \setminus \mathrm{fv}(\mathbb{K}, \mathbb{H})$).

CHR programs are finite sets of CHR rules. We use CHR_1 to denote the CHR language limited to single headed rules (i.e., rules with a single head atom).

Example 10. *Assume that the constraint* $\mathtt{a}(I, X)$ *represents the* I^{th} *cell of an array containing arbitrary data* X*. Now, consider* $\mathcal{P}_{10}$ *to be the following classical CHR program [7] consisting of the following single rule:*

$$\mathtt{a}(I, X), \mathtt{a}(J, Y) \iff I{>}J, X{<}Y \mid \mathtt{a}(I, Y), \mathtt{a}(J, X)$$

This rule sorts the array by swapping values at positions that are in the wrong order.

4.1.2 States

A *CHR state* is a tuple $\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle$ where $\mathbb{A}$ and $\mathbb{C}$ are multisets of atoms and constraints, respectively, and $\bar{X}$ is a set of variables, called *global variables*. The global variables represent the variable free in the initial goal.

Following Raiser et al. [21], we will consider CHR states modulo a structural equivalence. Formally, the *CHR state equivalence* is the least equivalence relation $\equiv_{\mathcal{C}}$ over states satisfying the following rules:

- $\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle \equiv_{\mathcal{C}} \langle \mathbb{A}; \mathbb{D}; \bar{X} \rangle$ if $\mathcal{C} \vDash \exists_{\text{-}(\mathbb{A}, \bar{X})}(\mathbb{C}) \leftrightarrow \exists_{\text{-}(\mathbb{A}, \bar{X})}(\mathbb{D})$
- $\langle \mathbb{A}; \bot; \bar{X} \rangle \equiv_{\mathcal{C}} \langle \mathbb{B}; \bot; \bar{X} \rangle$
- $\langle \mathbb{A}, c; \mathbb{C} \wedge c{=}d; \bar{X} \rangle \equiv_{\mathcal{C}} \langle \mathbb{A}, d; \mathbb{C} \wedge c{=}d; \bar{X} \rangle$
- $\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle \equiv_{\mathcal{C}} \langle \mathbb{A}; \mathbb{C}; \{Y\} \cup \bar{X} \rangle$ if $Y \notin \mathrm{fv}(\mathbb{A}, \mathbb{C})$.

will say that a state is *consistent* if its built-in store $\mathbb{C}$ is $\mathcal{C}$-satisfiable, and *inconsistent* otherwise.

To give some intuition about this structural equivalence we recall some examples by Fraiser et al.

Example 11 ([21])**.** *Let* $\mathcal{C}$ *be an arbitrary constraint theory and* $\mathtt{p}$ *a unary user-defined atom symbol. We have:*

$$\begin{aligned}
\langle \mathtt{p}(X); \top; \emptyset \rangle &\equiv_{\mathcal{C}} \langle \mathtt{p}(Y); \top; \emptyset \rangle \\
\langle \mathtt{p}(X); X{=}0; \{X\} \rangle &\equiv_{\mathcal{C}} \langle \mathtt{p}(0); X{=}0; \{X\} \rangle \\
\langle \emptyset; X \leq 0 \wedge X \geq 0 \wedge Y{=}0; \{X\} \rangle &\equiv_{\mathcal{C}} \langle \emptyset; X{=}0; \{X\} \rangle \\
\langle \mathtt{p}(0); \top; \{X\} \rangle &\equiv_{\mathcal{C}} \langle \mathtt{p}(0); \top; \emptyset \rangle \\
\langle \mathtt{p}(X); \top; \{X\} \rangle &\not\equiv_{\mathcal{C}} \langle \mathtt{p}(Y); \top; \{Y\} \rangle
\end{aligned}$$

We recall a useful result on structural equivalence.

Theorem 12 ([21])**.** *Let* $R = \langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle$ *and* $S = \langle \mathbb{B}; \mathbb{D}; \bar{Y} \rangle$ *be CHR states, such that* $(\mathrm{fv}(\mathbb{A}, \mathbb{C}) \cap \mathrm{fv}(\mathbb{B}, \mathbb{D})) \subseteq (\bar{X} \cap \bar{Y})$.

$$R \equiv_{\mathcal{C}} S \text{ if and only if } \begin{cases} \mathcal{C} \models \exists_{\text{-}\bar{X}}(\mathbb{C}) \leftrightarrow \exists_{\text{-}\bar{Y}}(\mathbb{D}), \\ \mathcal{C} \models \mathbb{C} \rightarrow \exists_{\text{-}\bar{Y}}(\mathbb{A}{=}\mathbb{B}), \text{ and} \\ \mathcal{C} \models \mathbb{D} \rightarrow \exists_{\text{-}\bar{X}}(\mathbb{A}{=}\mathbb{B}). \end{cases}$$

4.2 Operational semantics

Once state equivalence has been stated, the *equivalence-based* operational semantics ω_e of Raiser et al. [21] can be defined by a single inference rule. The resulting operational semantics is very similar to the *very abstract* semantics ω_a [7], the most general operational semantics of CHR. The equivalence-based style is preferred here, because the use of a notion of equivalence will simplify many formulations.

The *CHR transition* $\xrightarrow{\mathcal{P}}$ is the least relation satisfying the two following rules

$$\frac{(r \ @\ \mathbb{K} \backslash \mathbb{H} \iff \mathbb{G} | \mathbb{B}, \mathbb{C}) \in \mathcal{P}\theta \quad (\mathbb{G} \wedge \mathbb{D}) \text{ is } \mathcal{C}\text{-satisfiable}}{\langle \mathbb{H}, \mathbb{K}, \mathbb{A}; \mathbb{G} \wedge \mathbb{D}; \bar{X} \rangle \xrightarrow{\mathcal{P}} \langle \mathbb{B}, \mathbb{K}, \mathbb{A}; \mathbb{G} \wedge \mathbb{C} \wedge \mathbb{D}; \bar{X} \rangle}$$

where θ is a renaming with fresh variables.

We will say a state is *data-sufficient* if it has a computation ending with a state of the form $\langle \emptyset; \mathbb{C}; \bar{X} \rangle$. Similarly to CLP, a *success* for a CHR program $\mathcal{P}$ is a state that has a consistent answer with respect $\mathcal{P}$ (i.e., that can be derived by $\mathcal{P}$ to a goal of the form $\langle \emptyset; \mathbb{C}; \bar{X} \rangle$ where $\mathbb{C}$ is $\mathcal{C}$-satisfiable).

The following straightforward technical lemma about CHR transitions will be used later in the paper.

Lemma 13. *If* $\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}}{}^{*} \langle \mathbb{B}; \mathbb{D}; \bar{Y} \rangle$ *then* $\mathcal{C} \vDash \exists_{\text{-}Y}(\mathbb{D}) \rightarrow \exists_{\text{-}X}(\mathbb{C})$.

Proof. By induction on the length of $\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}}{}^{*} \langle \mathbb{B}; \mathbb{D}; \bar{Y} \rangle$. □

4.3 Declarative semantics

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

$$(\mathbb{K} \backslash \mathbb{H} \iff \mathbb{G} \mid \mathbb{B}, \mathbb{C})$$

is the following guarded equivalence:

$$\forall\big((\mathbb{K} \wedge \mathbb{G}) \rightarrow \big(\mathbb{H} \leftrightarrow \exists_{\text{-}(\mathbb{K}, \mathbb{H})}(\mathbb{G} \wedge \mathbb{B} \wedge \mathbb{C})\big)\big)$$

or, equivalently, the conjunction of implications:

$$\begin{aligned}
&\forall\big((\mathbb{K} \wedge \mathbb{H} \wedge \mathbb{G}) \rightarrow \exists_{\text{-}(\mathbb{K}, \mathbb{H})}(\mathbb{G} \wedge \mathbb{B} \wedge \mathbb{C})\big) \wedge \\
&\forall\big((\mathbb{K} \wedge \mathbb{G} \wedge \mathbb{B} \wedge \mathbb{C}) \rightarrow \mathbb{H}\big)
\end{aligned}$$

The *logical reading* of a program $\mathcal{P}$ within a constraint theory $\mathcal{C}$ is the theory $\mathcal{C}$ extended with the logical readings of the rules of $\mathcal{P}$. The *logical reading* of a state $\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle$ is the first order formula: $\exists_{\text{-}\bar{X}}(\mathbb{A} \wedge \mathbb{C})$.

Example 14. *The logical reading of the program* $\mathcal{P}_{10}$ *given in Example 10 is equivalent to the conjunction of the two following implications:*

$$\begin{aligned}
&\forall(I{>}J \wedge X{>}Y \wedge \mathtt{a}(I, X), \mathtt{a}(J, Y)) \rightarrow (\mathtt{a}(I, Y), \mathtt{a}(J, X)) \\
&\forall(I{>}J \wedge X{>}Y \wedge \mathtt{a}(I, Y), \mathtt{a}(J, X)) \rightarrow (\mathtt{a}(I, X), \mathtt{a}(J, Y))
\end{aligned}$$

The theorem we give next summarizes basic results about soundness and completeness of *data-sufficient states* with respect the declarative meaning of a program.

Theorem 15. *Let* $\mathcal{P}$ *be a CHR program with a consistent logical reading, and* $\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}}{}^{*} \langle \emptyset; \mathbb{D}; Y \rangle$ *be a valid CHR derivation.* $(\mathbb{A} \wedge \mathbb{C})$ *is satisfiable with respect the logical reading of* $\mathcal{P}$ *if and only if* $\mathcal{P}, \mathcal{C} \vDash \exists_{\text{-}\bar{X}}(\mathbb{A} \wedge \mathbb{C}) \leftrightarrow \exists_{\text{-}\bar{X}}(\mathbb{D})$.

Proof. Direct by the Soundness and Stronger Completeness of failed computations Theorems (refer to Theorems 3.21 and 3.25 in Frühwirth's book [7]). □

5. CLP Projection

In this section, we introduce and formally study the CLP projection for CHR programs. In the next section, we will see some direct applications.

5.1 Definition

As explained in Section 4.3, the logical reading of a simplification rule is an equivalence. The basis of the CLP projection is to ignore the right-to-left implication part of the equivalence and consider only the left-to-right part. Indeed, one can consider 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 $(c_1 \leftarrow \exists Z(\mathbb{K} \wedge \mathbb{G} \wedge \mathbb{B})), \ldots, (c_n \leftarrow \exists Z(\mathbb{G} \wedge \mathbb{D} \wedge \mathbb{B} \wedge \mathbb{K}))$. Formally the CLP projection can be defined as follows:

Definition 16 (CLP Projection)**.** *The* (CLP) projection *of a CHR program $\mathcal{P}$ is the set $\pi(\mathcal{P})$ of CLP clauses defined as:*

$$\pi(\mathcal{P}) = \{\, (a \leftarrow \mathbb{K}, \mathbb{B} \mid \mathbb{G} \wedge \mathbb{C}) \mid (\mathbb{K} \backslash \mathbb{H} \Longleftrightarrow \mathbb{G} \mid \mathbb{B}, \mathbb{C}) \in \mathcal{P} \text{ and } a \in (\mathbb{H}, \mathbb{K}) \}$$

The (CLP) projection *of a CHR state is defined in a straightforward way, i.e.:*

$$\pi(\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle) = \langle \mathbb{A} | \mathbb{C} \rangle$$

We illustrate now the result of a CLP projection.

Example 17. *Consider the program $\mathcal{P}_{10}$ given in Example 10. The CLP projection of $\mathcal{P}_{10}$ consists of the following CLP clauses:*

$$\mathtt{a}(I, X) \leftarrow \mathtt{a}(I, Y), \mathtt{a}(J, X) \mid I{>}J \wedge X{<}Y$$
$$\mathtt{a}(J, Y) \leftarrow \mathtt{a}(I, Y), \mathtt{a}(J, X) \mid I{>}J \wedge X{<}Y$$

5.2 Approximating CHR operational semantics

In this section, we relate the operational behavior of a CHR program and its CLP projection. To state the result of the section, we first introduce two relations of state subsumption.

Definition 18 (State subsumption relations)**.** *The relations $\sqsubseteq_{\mathcal{C}}$ and $\subseteq_{\mathcal{C}}$ are defined as the least transitive relations containing $\equiv_{\mathcal{C}}$ and satisfying respectively*

$$\langle \mathbb{A}; \mathbb{C} \wedge \mathbb{D}; \bar{X} \rangle \sqsubseteq_{\mathcal{C}} \langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle$$
$$\langle \mathbb{A}; \mathbb{C} \wedge \mathbb{D}; \bar{X} \cup \bar{Y} \rangle \subseteq_{\mathcal{C}} \langle \mathbb{A}, \mathbb{B}; \mathbb{C}; \bar{X} \rangle$$

where $\mathbb{B}$ stands for an arbitrary multiset of atoms, $\mathbb{D}$ stands for an arbitrary conjunction of constraints, and $\bar{Y}$ stand for a arbitrary set of variables.

Note that both relations mean that the $\mathcal{C}$-interpretation of the left-hand side state is more constrained than that of the right-hand side state. $\sqsubseteq_{\mathcal{C}}$ differs from $\subseteq_{\mathcal{C}}$ in that $\sqsubseteq_{\mathcal{C}}$ preserves precise information about the multiplicity of user-defined atoms while $\subseteq_{\mathcal{C}}$ does not.

The theorem we give next states that the operational semantics of a CHR program can be simulated by its projection. We will use this theorem, which establishes that the termination of $\pi(\mathcal{P})$ implies the termination of $\mathcal{P}$, in Section 6.1.

Theorem 19. *For any CHR program $\mathcal{P}$, we have:*

For all states R, R', S if $R \xrightarrow{\mathcal{P}} R'$ and $R \subseteq_{\mathcal{C}} S$, then

there exists a state S' such that $\pi(S) \xrightarrow{\pi(\mathcal{P})} \pi(S')$ and $R' \subseteq_{\mathcal{C}} S'$.

The theorem is graphically represented by the diagram of Figure 1. Following standard diagrammatic notation, solid edges stand for universally-quantified relations (i.e., the premise), and the dashed edges stand for existentially-quantified relations (i.e., the conclusions). CHR states are nodes in the upper side and CLP goals are nodes in the lower side.

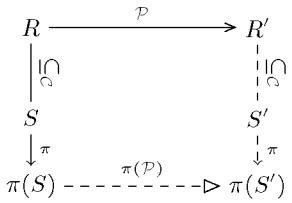

Figure 1. Simulation diagram for a CHR program $\mathcal{P}$.

Before formally proving the theorem, we illustrate the theorem on our running example.

Example 20. *Consider the program $\mathcal{P}_{10}$ given in Example 10. Applying the rule* sort*, we infer the possible transition:*

$$R = \langle \mathtt{a}(0,7), \mathtt{a}(1,5); \top; \emptyset \rangle \xrightarrow{\mathcal{P}_{10}} \langle \mathtt{a}(0,5), \mathtt{a}(1,7); \top; \emptyset \rangle = R'$$

It is straightforward to verify that using the projection of $\mathcal{P}_{10}$ (given explicitly in Example 17) that the following derivation is valid with respect $\pi(\mathcal{P}_{10})$:

$$\langle \mathtt{a}(0,7), \mathtt{a}(1,5) | \top \rangle \xrightarrow{\pi(\mathcal{P}_{10})}$$
$$\langle \mathtt{a}(I,7), \mathtt{a}(0,X), \mathtt{a}(1,5) | I{>}0 \wedge X{<}7 \rangle$$

Note that it holds that R' is included (with respect $\subseteq_{\mathcal{C}}$) in the state:

$$\langle \mathtt{a}(I,7), \mathtt{a}(0,X), \mathtt{a}(1,5); I{>}0 \wedge X{<}7; \emptyset \rangle$$

Proof. Assume that r is of the form

$$\mathbb{K} \backslash \mathbb{H} \Longleftrightarrow \mathbb{G} | \mathbb{B}, \mathbb{C}$$

Without loss of generality we can assume that the states R, R' and S satisfy:

$$R \equiv_{\mathcal{C}} \langle \mathbb{K}', \mathbb{H}', c, \mathbb{A}; \mathbb{G} \wedge \mathbb{D} \wedge \mathbb{D}'; \bar{X} \cup \bar{Y} \rangle$$
$$S \equiv_{\mathcal{C}} \langle \mathbb{K}', \mathbb{H}', c, \mathbb{A}, \mathbb{A}'; \mathbb{D}'; \bar{X} \rangle$$
$$R' \equiv_{\mathcal{C}} \langle \mathbb{B}, \mathbb{K}, \mathbb{A}; \mathbb{C} \wedge \mathbb{G} \wedge \mathbb{D} \wedge \mathbb{D}'; \bar{X} \cup \bar{Y} \rangle$$

with $(\mathbb{K}, \mathbb{H}) = (\mathbb{K}', \mathbb{H}', c)$.

Let θ be a renaming with fresh variables. Hence, we have:

$$\pi(S) \xrightarrow{\pi(\mathcal{P})} \big(\mathbb{K}\theta, \mathbb{B}\theta, \mathbb{K}', \mathbb{H}', \mathbb{A}, \mathbb{A}' | \mathbb{D}' \wedge c{=}(c\theta) \wedge \mathbb{G}\theta \wedge \mathbb{C}\theta\big)$$

To conclude it is sufficient to notice that:

$$R' \subseteq_{\mathcal{C}} \langle \mathbb{K}\theta, \mathbb{B}\theta, \mathbb{K}', \mathbb{H}', \mathbb{A}, \mathbb{A}'; \mathbb{D}' \wedge c{=}(c\theta) \wedge \mathbb{G}\theta \wedge \mathbb{C}\theta; \bar{X} \rangle$$

□

In fact, one can prove a more precise correspondence in the case of single headed simplifications.

Theorem 21. *For any CHR$_1$ program $\mathcal{P}$, we have:*

For all states R, R', S if $R \xrightarrow{\mathcal{P}} R'$ and $R \sqsubseteq_{\mathcal{C}} S$, then

there exists a state S' such that $\pi(S) \xrightarrow{\pi(\mathcal{P})} \pi(S')$ and $R' \sqsubseteq_{\mathcal{C}} S'$.

This second theorem is graphically represented by the diagram of Figure 2.

Proof. Assume r is of the form:

$$c \Longleftrightarrow \mathbb{G} | \mathbb{B}, \mathbb{C}$$

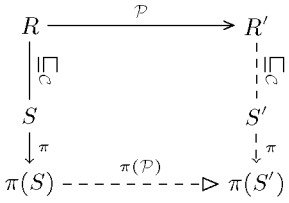

Figure 2. Strong simulation diagram for a CHR_1 program $\mathcal{P}$.

Without loss of generality we can assume that the states R, R' and S satisfy:

$$R \equiv_{\mathcal{C}} \langle c, \mathbb{A}; \mathbb{G} \wedge \mathbb{D} \wedge \mathbb{D}'; \bar{X} \cup \bar{Y} \rangle$$
$$S \equiv_{\mathcal{C}} \langle c, \mathbb{A}, \mathbb{A}'; \mathbb{D}'; \bar{X} \rangle$$
$$R' \equiv_{\mathcal{C}} \langle \mathbb{B}, \mathbb{A}; \mathbb{C} \wedge \mathbb{G} \wedge \mathbb{D} \wedge \mathbb{D}'; \bar{X} \cup \bar{Y} \rangle$$

Let θ be a renaming with fresh variables. Hence, we have:

$$\pi(S) \xrightarrow{\pi(\mathcal{P})} \big(\mathbb{K}\theta, \mathbb{B}\theta, \mathbb{K}', \mathbb{H}', \mathbb{A}, \mathbb{A}' | \mathbb{D}' \wedge c = (c\theta) \wedge \mathbb{G}\theta \wedge \mathbb{C}\theta\big)$$

To conclude it is sufficient to notice that:

$$R' \sqsubseteq_{\mathcal{C}} \langle \mathbb{K}\theta, \mathbb{B}\theta, \mathbb{K}', \mathbb{H}', \mathbb{A}, \mathbb{A}'; \mathbb{D}' \wedge c = (c\theta) \wedge \mathbb{G}\theta \wedge \mathbb{C}\theta; \bar{X} \rangle$$

□

In general the theorems do not hold in the reverse direction. Indeed as shown through the following counter-example that a program does not simulate is projection.

Example 22. *Consider the program $\mathcal{P}_{10}$ given in Example 10 and its projection $\pi(\mathcal{P}_{10})$ given explicitly at Example 17. The CLP transition:*

$$\langle \mathtt{a}(0,5) | \top \rangle \xrightarrow{\pi(\mathcal{P}_{10})} \langle \mathtt{a}(I,5), \mathtt{a}(0,X), | I{>}0 \wedge X{<}5 \rangle$$

is valid with respect the CLP projection of $\mathcal{P}_{10}$. Nevertheless $\langle \mathtt{a}(0,5); \top; \emptyset \rangle$ cannot be derived by $\mathcal{P}_{10}$. The reason for this dichotomy is that $\mathcal{P}_{10}$ is a terminating program while $\pi(\mathcal{P}_{10})$ is not – in fact, $\pi(\mathcal{P}_{10})$ has no terminating derivations.

5.3 Approximating CHR declarative semantics

In this section, we relate the model of a CHR program with that of its projection. Indeed, the following proposition states that the logical model of a CHR program is bounded by the least model of its projections in the lattice of $\mathcal{C}$-interpretations.

Proposition 23. *Let $\mathcal{P}$ be a CHR program. For any $\mathcal{C}$-model $\mathcal{M}$ of $\mathcal{P}$, we have:*

$$\mu\mathcal{X}.\Uparrow^{\mathcal{C}} \left(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\right) \subseteq \mathcal{M}$$

Proof. If $\mathcal{P}$ has no $\mathcal{C}$-model, then the result is direct. Otherwise, let $\mathcal{M}$ be a $\mathcal{C}$-model of $\mathcal{P}$. Obviously $\pi(\mathcal{P})$ is a logical consequence of $\mathcal{P}$ (i.e., $\mathcal{C} \models \mathcal{P} \rightarrow \pi(\mathcal{P})$), hence $\mathcal{M}$ is a $\mathcal{C}$-model for $\pi(\mathcal{P})$. The conclusion is then direct using Theorem 9. □

As shown by the following examples this approximation is in general quite imprecise.

Example 24. *Consider the program $\mathcal{P}_{24}$ comprising the following rules:*

$$\mathtt{c} \Longleftrightarrow \top \qquad \mathtt{c} \Longleftrightarrow \mathtt{d} \qquad \mathtt{e} \Longleftrightarrow \mathtt{e}$$

The projection of $\mathcal{P}_{24}$ consists of the following clauses:

$$\mathtt{c} \leftarrow \top \qquad \mathtt{c} \leftarrow \mathtt{d} \qquad \mathtt{e} \leftarrow \mathtt{e}$$

The least $\mathcal{C}$-model of $\pi(\mathcal{P}_{24})$ is $\Uparrow^{\mathcal{C}}((\mathtt{c}|\top))$, but the least $\mathcal{C}$-model of $\mathcal{P}_{24}$ is $\Uparrow^{\mathcal{C}}((\mathtt{c},\mathtt{d}|\top))$.

We show now that in the case that $\mathcal{P}$ is confluent, the least fixpoint of $T^{\mathcal{C}}_{\pi(\mathcal{P})}$ provides a logical model for $\mathcal{P}$.

Theorem 25. *Let $\mathcal{P}$ be a confluent program. $\mu\mathcal{X}.\Uparrow^{\mathcal{C}}\big(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\big)$ is the least $\mathcal{C}$-model of $\mathcal{P}$.*

The proof of the proposition relies on two main lemmas. The first one states that if a state R can be derived to a consistent state S which has a projection included in a fixpoint of $\lambda\mathcal{X}.\Uparrow^{\mathcal{C}}\big(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\big)$, then there is a consistent state R' more constrained than R which is the same fixpoint.

Lemma 26. *Let $\mathcal{P}$ be a program, $\mathcal{Y}$ a set of constrained atoms, and $\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}}{}^{*} \langle \mathbb{B}; \mathbb{D}; \bar{X} \rangle$ a valid derivation such that $\text{fv}(\mathbb{A}, \mathbb{C}) \subseteq \bar{X}$. If $\mathcal{Y}$ is a fixpoint of $\lambda\mathcal{X}.\Uparrow^{\mathcal{C}}\big(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\big)$ and $(\mathbb{B}|\mathbb{D}) \subseteq \mathcal{Y}$, then $(\mathbb{A}|\mathbb{C} \wedge \mathbb{D}) \subseteq \mathcal{Y}$.*

Proof. We prove only the case of the one-step derivation, the general case will follow directly by reflexivity and transitivity of the inclusion. Let $r@(\mathbb{K} \backslash \mathbb{H} \Longleftrightarrow \mathbb{G} | \mathbb{F}, \mathbb{E})$ be the rule of $\mathcal{P}$ used for the transition. We have for some $\mathbb{A}'$, $\mathbb{C}'$, and $\bar{X}'$:

$$\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle \equiv_{\mathcal{C}} \langle \mathbb{H}, \mathbb{K}, \mathbb{A}'; \mathbb{G} \wedge \mathbb{C}'; \bar{X} \rangle \tag{1}$$
$$\langle \mathbb{F}, \mathbb{K}, \mathbb{A}'; \mathbb{E} \wedge \mathbb{G} \wedge \mathbb{C}'; \bar{X} \rangle \equiv_{\mathcal{C}} \langle \mathbb{B}; \mathbb{D}; \bar{Y} \rangle \tag{2}$$
$$\Longrightarrow \big(\mathbb{F}, \mathbb{A}', \mathbb{K} | \mathbb{E} \wedge \mathbb{G} \wedge \mathbb{C}'\big) \subseteq \mathcal{Z} \tag{3}$$
$$\Longrightarrow \big(\mathbb{H} | \mathbb{E} \wedge \mathbb{G} \wedge \mathbb{C}'\big) \subseteq T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{Z}) \tag{4}$$
$$\Longrightarrow \big(\mathbb{H}, \mathbb{K}, \mathbb{A}' | \mathbb{E} \wedge \mathbb{G} \wedge \mathbb{C}'\big) \subseteq \mathcal{Z} \tag{5}$$

(1) and (2) are by the definition of $\xrightarrow{\mathcal{P}}$. (3) is by Theorem 12 and idempotence of a closure operator. (4) is by the definition of $T^{\mathcal{C}}_{\pi(\mathcal{P})}$. (5) combines (4) with (3) and uses the fact that $\mathcal{Z}$ is a fixpoint of $\lambda\mathcal{X}.\Uparrow^{\mathcal{C}}\big(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\big)$.

On the other hand we have:

$$\mathcal{C} \vDash \mathbb{D} \rightarrow \mathbb{C} \text{ and } \mathcal{C} \vDash \mathbb{D} \rightarrow \exists_{\text{-}\bar{X}} \big(\mathbb{E} \wedge \mathbb{G} \wedge \mathbb{C}'\big) \tag{6}$$
$$\mathcal{C} \vDash \mathbb{D} \rightarrow \mathbb{C} \text{ and } \big(\mathbb{H}, \mathbb{K}, \mathbb{A}' | \mathbb{D}\big) \subseteq \mathcal{Z} \tag{7}$$
$$\mathcal{C} \vDash \mathbb{D} \rightarrow \mathbb{C} \text{ and } (\mathbb{A}|\mathbb{D}) \subseteq \mathcal{Z} \tag{8}$$
$$(\mathbb{A}|\mathbb{C} \wedge \mathbb{D}) \subseteq \mathcal{Z} \tag{9}$$

The left hand side of (6) is by Lemma 13, the right hand side is inferred form (2) by Theorem 12. (7) is inferred from (5) because $\mathcal{Z}$ is closed by $\Uparrow^{\mathcal{C}}$. Finally, 8 combines the left and right hand sides in a straightforward way. □

The second lemma says that any state derived from a consistent state which has a CLP projection in a fixpoint of $\lambda\mathcal{X}.\Uparrow^{\mathcal{C}}\big(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\big)$, has a CLP projection in the same fixpoint. Contrary to the previous lemma, the core of the proof relies on the confluence of the considered CHR program.

Lemma 27. *Let $\mathcal{P}$ be a confluent program. For any ordinal α, if $\langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle \xrightarrow{\mathcal{P}}{}^{*} \langle \mathbb{A}'; \mathbb{C}'; \bar{X} \rangle$, $\text{fv}(\mathbb{A}) \subseteq X$, and $(\mathbb{A}|\mathbb{C}) \subseteq \lambda\mathcal{X}.\Uparrow^{\mathcal{C}}\big(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\big) \uparrow \alpha$, then $(\mathbb{A}'|\mathbb{C}') \subseteq \mu\mathcal{X}.\Uparrow^{\mathcal{C}}\big(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\big)$ and $\mathcal{C} \vDash \mathbb{C} \rightarrow \exists_{\text{-}X} \mathbb{C}'$.*

Proof. Let F denote the function $\lambda\mathcal{X}.\Uparrow^{\mathcal{C}}\big(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\big)$. The proof is by transfinite induction on α:

- The base case, $\alpha = 0$ is trivial.

- For a successor ordinal, we have:

$$(\mathbb{A}|\mathbb{C}) \subseteq F \uparrow (\alpha{+}1) = \Uparrow^{\mathcal{C}}\left(T^{\mathcal{C}}_{\pi(\mathcal{P})}(F \uparrow \alpha)\right) \quad (1)$$

$$\Longrightarrow \mathbb{A} \text{ is of the form } \{a_1, \ldots, a_n\} \text{ with for } i \in 1, \ldots, n$$

$$r_i @ (\mathbb{K}_i \backslash a_i, \mathbb{H}_i \Longleftrightarrow \mathbb{G}_i \mid \mathbb{B}_i, \mathbb{C}_i) \in \rho_i(\mathcal{P}), \quad (2)$$

$$(\mathbb{K}_i, \mathbb{B}_i|\mathbb{C}_i) \subseteq F \uparrow \alpha, \text{ and} \quad (3)$$

$$\mathcal{C} \vDash \mathbb{C} \rightarrow \exists_{\text{-}c_i}(\mathbb{G}_i \wedge \mathbb{B}_i \wedge \mathbb{D}_i) \quad (4)$$

(1) is by definition of upward power. (2) to (4) are by definition of $T^{\mathcal{C}}_{\pi(\mathcal{P})}$ and $\Uparrow^{\mathcal{C}}$.

Assuming $\vec{A}$ is an abbreviation for any sequence of constraints of the form $A_i, \ldots, A_n$ and $Y = X \cup \text{fv}\left(\vec{\mathbb{K}}, \vec{\mathbb{H}}\right)$, we have:

$$\Longrightarrow \langle \mathbb{A}, \bar{\mathbb{H}}, \bar{\mathbb{K}}; \mathbb{C} \wedge \bar{\mathbb{C}} \wedge \bar{\mathbb{G}}; \bar{Y} \rangle \xrightarrow{\mathcal{P}}{}^{*} \langle \mathbb{A}', \bar{\mathbb{H}}, \bar{\mathbb{K}}; \mathbb{C}' \wedge \bar{\mathbb{C}} \wedge \bar{\mathbb{G}}; \bar{Y} \rangle \quad (5)$$

$$\langle \mathbb{A}, \bar{\mathbb{H}}, \bar{\mathbb{K}}; \mathbb{C} \wedge \bar{\mathbb{C}} \wedge \bar{\mathbb{G}}; \bar{Y} \rangle \xrightarrow{\mathcal{P}}{}^{*} \langle \bar{\mathbb{B}}, \bar{\mathbb{K}}; \mathbb{C} \wedge \bar{\mathbb{C}} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{D}}; \bar{Y} \rangle \quad (6)$$

$$\Longrightarrow \text{there is } \langle \mathbb{B}'; \mathbb{D}'; \bar{Y} \rangle \text{ s.t. :}$$

$$\langle \mathbb{A}', \bar{\mathbb{H}}, \bar{\mathbb{K}}; \mathbb{C}' \wedge \bar{\mathbb{C}} \wedge \bar{\mathbb{G}}; \bar{Y} \rangle \xrightarrow{\mathcal{P}}{}^{*} \langle \mathbb{B}'; \mathbb{D}'; \bar{Y} \rangle \text{ and} \quad (7)$$

$$\langle \bar{\mathbb{B}}, \bar{\mathbb{K}}; \mathbb{C} \wedge \bar{\mathbb{C}} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{D}}; \bar{Y} \rangle \xrightarrow{\mathcal{P}}{}^{*} \langle \mathbb{B}'; \mathbb{D}'; \bar{Y} \rangle \quad (8)$$

$$\Longrightarrow \left(\mathbb{B}'|\mathbb{D}'\right) \subseteq \mu\mathcal{X}.F(\mathcal{X}) \text{ with}$$

$$\mathcal{C} \models \left(\mathbb{C} \wedge \bar{\mathbb{C}} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{D}}\right) \rightarrow \exists_{\text{-}Y}\mathbb{D}' \quad (9)$$

$$\Longrightarrow \left(\mathbb{A}'|\mathbb{C} \wedge \bar{\mathbb{C}} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{D}}\right) \subseteq \mu\mathcal{X}.F(\mathcal{X}) \quad (10)$$

(5) is by monotonicity of CHR derivation (Lemma 4.2 in [7]). (6) is by applying the r_i rules. (7) and (8) are by confluence of $\xrightarrow{\mathcal{P}}$. (9) is inferred from (3), (8), induction hypothesis, and the fact that $F \uparrow \alpha$ is closed by $\Uparrow^{\mathcal{C}}$. (10) is by Lemma 26.

On the other hand we have:

$$\Longrightarrow \mathcal{C} \models \mathbb{D}' \rightarrow \exists_{\text{-}\bar{Y}}\mathbb{C}' \quad (11)$$

$$\mathcal{C} \models \mathbb{C} \rightarrow \exists_{\text{-}\bar{Y}}\left(\mathbb{C} \wedge \bar{\mathbb{C}} \wedge \bar{\mathbb{G}} \wedge \bar{\mathbb{D}}\right) \quad (12)$$

$$\Longrightarrow \mathcal{C} \models \mathbb{C} \rightarrow \exists_{\text{-}\bar{Y}}\mathbb{C}' \quad (13)$$

Thanks to Lemma 13, (11) is inferred from (8). (12) is straightforward from (4). Finally (13) is obtained from (11), (12) and (9) using transitivity of the implication.

- For a limit ordinal, we have $T^{\mathcal{C}}_{\pi(\mathcal{P})} \uparrow \alpha = \bigcup_{\beta<\alpha} F \uparrow \beta$. Using monotonicity of $\lambda\mathcal{X}.(\mathcal{Y} \cup T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X}))$ and the fact that $\mathbb{D}$ is finite, there exists obviously an ordinal $\beta < \alpha$ such that $\mathbb{D}\rho \in \lambda\mathcal{X}.(\mathcal{Y} \cup T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})) \uparrow \beta$. The conclusion is then direct using induction hypothesis.

□

of Theorem 25. Assume that the following rule is in $\mathcal{P}$:

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

Let θ be a renaming of $\text{fv}(\mathbb{G}, \mathbb{B}, \mathbb{C}) \setminus \text{fv}(\mathbb{K}, \mathbb{H})$ with fresh variables (in particular $(\mathbb{K}, \mathbb{H})\theta = (\mathbb{K}, \mathbb{H})$) and let $\bar{Y} = \text{fv}(\mathbb{G}\theta, \mathbb{B}\theta, \mathbb{C}\theta)$. The logical reading of r is logically equivalent to the two following implications:

$$\forall((\mathbb{K} \wedge \mathbb{H} \wedge \mathbb{G}) \rightarrow \exists_{\text{-}\bar{Y}}(\mathbb{G}\theta \wedge \mathbb{B}\theta \wedge \mathbb{C}\theta))$$

$$\forall((\mathbb{K} \wedge \mathbb{G} \wedge \mathbb{B} \wedge \mathbb{C}) \rightarrow \mathbb{H})$$

Thanks to Lemma 6, in order to prove $\mathcal{Z} = \mu\mathcal{X}.\Uparrow^{\mathcal{C}}\left(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\right)$ is a model for $\mathcal{P}$, we know it is sufficient to show that:

(i) For any conjunction of constraints $\mathbb{E}$ such that $\text{fv}(\mathbb{E}) \cap \bar{Y} = \emptyset$, $(\mathbb{K}, \mathbb{H}|\mathbb{C} \wedge \mathbb{E}) \in \mathcal{Z}$ implies $(\mathbb{K}, \mathbb{B}|\mathbb{G}\theta \wedge \mathbb{B}\theta \wedge \mathbb{C}\theta) \in \mathcal{Z}$ and $\mathcal{C} \vDash (\mathbb{C} \wedge \mathbb{E}) \rightarrow \exists\bar{Y}(\mathbb{G}\theta \wedge \mathbb{C}\theta \wedge \mathbb{E})$

(ii) For any conjunction of constraints $\mathbb{E}$, $(\mathbb{K}, \mathbb{B}|\mathbb{G} \wedge \mathbb{C} \wedge \mathbb{E}) \in \mathcal{Z}$ implies $(\mathbb{H}|\mathbb{G} \wedge \mathbb{C} \wedge \mathbb{E}) \in \mathcal{Z}$

Since we have obviously

$$\langle \mathbb{K}, \mathbb{H}; \mathbb{G} \wedge \mathbb{E}; \bar{X} \rangle \xrightarrow{\mathcal{P}} \langle \mathbb{K}, \mathbb{B}\theta; \mathbb{G} \wedge \mathbb{E} \wedge \mathbb{G}\theta \wedge \mathbb{C}\theta; \bar{X} \rangle$$

(i) follows by Lemma 26 and (ii) by Lemma 27 together with Knaster–Tarski Theorem. Hence, we know that $\mu\mathcal{X}.\Uparrow^{\mathcal{C}}\left(T^{\mathcal{C}}_{\pi(\mathcal{P})}(\mathcal{X})\right)$ is a model of $\mathcal{P}$, its minimality being guaranteed by Proposition 23. □

As a direct corollary, we get that a confluent program is consistent. We close hence a conjecture of Abdenader et al. about consistency of general confluent CHR program [1], the original proof being limited to range restricted programs (i.e., programs without local variables). Note furthermore that our theorem does not assume that the constraint theory is ground complete. Consequently, it is possible to strengthen existing results about CHR declarative semantics, especially the completeness of operational semantics with respect failure where both conditions of range restriction and ground completeness of the constraint system can be dropped (refer to Corollary 5.19 in Früwirth's book). This improvement is important, since we identified in a recent publication a class of confluent programs (the so-called coinductive solvers) which by construction are not range restricted [10].

The following example illustrates that a confluent CHR program may not have an unique greatest $\mathcal{C}$-model. This comes from the non-compositionality of the declarative semantics of CHR (i.e., if the logical readings of two states are independently consistent, then one cannot ensure that so is their conjunction).

Example 28. *Let $\mathcal{P}_{28}$ be the program consisting of the following rules:*

$$p, q \Longleftrightarrow \bot$$

$\Uparrow^{\mathcal{C}}(\{p|\top\})$ and $\Uparrow^{\mathcal{C}}(\{q|\top\})$ are two greatest incomparable $\mathcal{C}$-models for $\mathcal{P}_{28}$.

An interesting consequence is that the logical semantics of both formalisms coincide on data-sufficient state.

Theorem 29. *A data-sufficient state is a success for a confluent CHR program $\mathcal{P}$ if and only if it is a success for $\pi(\mathcal{P})$.*

Proof. Let $S = \langle \mathbb{A}; \mathbb{C}; \bar{X} \rangle$ be a data-sufficient state with respect P. By Theorem 15, S is a success of $\mathcal{P}, \mathcal{C}$ if and only if $\mathcal{P}, \mathcal{C} \vdash \exists(\mathbb{A} \wedge \mathbb{C}) \rightarrow \exists\mathbb{D}$ for some $\mathcal{C}$-satisfiable conjunction $\mathbb{D}$. Since $\mathcal{P}$ and $\mathcal{P}, \mathcal{C}$ have the same least model S is a success of $\mathcal{P}, \mathcal{C}$ if and only if $\mathcal{P}, \mathcal{C} \vdash \exists(\mathbb{A} \wedge \mathbb{C}) \rightarrow \exists_{\text{-}\bar{X}}(\mathbb{D})$ for some $\mathcal{C}$-satisfiable conjunction $\mathbb{D}$. Then, by soundness and completeness of the CSCLD resolution [15] S is a success of $\mathcal{P}, \mathcal{C}$ if and only if $\pi(\mathcal{S})$ is a success for $\pi(\mathcal{P})$. □

Note that this result does not contradict Di Giusto et al.'s results [9] about the greater expressiveness of multi-headed programs with respect single-headed programs. Indeed, even though any multi-headed program has the same CLP projection as some single-headed program, the two CHR programs do not have the same set of data-sufficient states.

6. Applications

6.1 Termination proofs of CHR$_1$ programs

Theorem 19 ensures that if the CLP projection of a CHR program $\mathcal{P}$ terminates (with respect the CLP operational semantics), then $\mathcal{P}$ terminates (with respect the CHR operational semantics). Hence, in order to prove the termination of a CHR program, it is sufficient to prove the termination of its projection.

benchmark	trans.	proj.	benchmark	trans.	proj.
ackerman	−	+	modulo	+	+
average	+	+	oddeven	+	+
binlog	+	+	power	+	+
booland	+	+	revlist	+	+
convert	−	+	toyama	+	+
diff	+	+	weight	−	+
factorial	+	+			

Table 1. CHR_1 termination comparison.

Nonetheless, as pointed out by Example 22 there exist terminating CHR programs that have non-terminating projection. The reason for this dichotomy resides in two possible weaknesses of the projection:

- Information about multiplicity of linear atoms is lost.
- The guard conditions are ignored.

Consequently, the CLP projection cannot be used to prove termination of programs when they rely on the multiplicity of atoms in the store (as illustrated by Example 22) or on the non-entailment of guard conditions (as illustrated by the following example).

Example 30. *Consider the program $\mathcal{P}_{30}$ consisting of the single rule:*

$$\mathtt{f}(0) \Longleftrightarrow \mathtt{f}(Y)$$

Its projection, $\pi(\mathcal{P}_{30})$, is made up of the following rule:

$$\mathtt{f}(0) \leftarrow \mathtt{f}(Y)$$

It is straightforward to verify that any CHR state has only a finite derivation with respect $\mathcal{P}_{30}$. For instance, the following derivation cannot be extended:

$$\langle \mathtt{f}(0); \top; \emptyset\rangle \xrightarrow{\mathcal{P}_{30}} \langle \mathtt{f}(X_0); \top; \emptyset\rangle \xrightarrow{\mathcal{P}_{30}} \not\longrightarrow$$

but unconstrained CLP goals $\langle \mathtt{f}(X_0)|\top\rangle$ have an infinite derivation with respect the projection of $\mathcal{P}_{30}$:

$$\langle \mathtt{f}(X_0)|\top\rangle \xrightarrow{\pi(\mathcal{P}_{30})} \cdots \langle \mathtt{f}(X_i)|X_0{=}0 \wedge \cdots \wedge X_{i\text{-}1} = 0\rangle \xrightarrow{\pi(\mathcal{P}_{30})} \cdots$$

In spite of its weaknesses in treating guarded multi-headed programs – indeed the termination of most of multi-headed programs relies on the multiplicity of atoms in the constraints store – the CLP projection is a powerful notion for tackling the termination analysis of single-headed programs. For instance, Table 1 compares the termination of single-headed programs as inferred by the AProVE system [8] using Pilozzi et al.'s transformation [18] (column trans.) and using the CLP projection (column proj.). In the table, '+' indicates a positive termination inference, while '−' stands for a negative one. All the results in the *trans.* column are reported as given by Pilozzi et al. [18]. Out of a list of 24 programs [18], the CLP projection-based approach was able to prove termination of all 13 CHR_1 programs[1].

In fact, transforming CHR rules into Prolog clauses has the advantage with respect the meta-interpreter approach of Pilozzi et al. that user-defined atoms are converted to predicate names, and thus become control points. This allows using techniques that reduce the problem of global termination to several local termination problems [4] for which it is simpler to synthesize a ranking function. For instance, it is not clear what is the global ranking function for the Ackerman program, while the termination of the Prolog program can be proven easily by systems such as AProVE or TerminWeb [4].

[1] These favorable results suggest that widely-used benchmarks tend not to include programs as the one of Example 30 whose termination relies on non-entailment of guard conditions.

It goes without saying there exist today ad-hoc CHR analyzers that provide better results than Pilozzi et al.'s transformation-based approach. For instance, Pilozzi's CHRisTA system [17] can prove the termination of *convert* and *weight*. Nevertheless, to the best of our knowledge, the CLP projection together with AProVE provides the first automatic termination proof for the CHR implementation of *ackerman*.

6.2 Type analysis of CHR programs

In Section 5.3, we have shown that the success set of a confluent program $\mathcal{P}$ can be characterized by the success of the projection of $\mathcal{P}$. Consequently, any safe approximation of properties about the success set of a Prolog program inferred via static analysis is also a safe approximation of the projection of a confluent program.

As an illustration, we analyze the CLP projection of some confluent programs using CiaoPP [11]. Since the CLP projection has been implemented as a Ciao *package* [2], it is possible to transparently analyze CHR programs using CiaoPP.

CiaoPP can infer properties on the (values of) variables in the computation of predicates, i.e., *state properties*, as well as global properties of such computations (such as, e.g., the number of execution steps, determinacy, or the usage of some other resource). In CiaoPP *state properties* can be expressed by predicates. A particular case of *state properties* are *regular types* [5]. Regular types can be defined in libraries, defined by the user, or automatically inferred by the system.

For instance, consider the following module implementing the *oddeven* program:

```
:- module(oddeven, [oddeven/2], [clp_projection]).

oddeven(0,B) <=> B=even.
oddeven(1,B) <=> B=odd.
oddeven(A,B) <=> A > 2 |
    App is A - 2, oddeven(App,B).
```

The `[clp_projection]` argument in the module declaration states that the `clp_projection` package should be used. This package applies the CLP projection transformation so that if the CHR program above is fed to CiaoPP, CiaoPP sees the CLP projection of the program above. The result of applying CiaoPP's type analysis is then expressed by using assertions [20] as follows:

```
:- true success oddeven(A,B)
       => ( arithexpression(A), rt5(B) ).
:- regtype rt5/1.
rt5(even).
rt5(odd).
```

The first assertion expresses that on success, the first argument of `oddeven/2` is an arithmetic expression, while the second one is of type `rt5/1` (i.e., is either `even` or `odd`). The prefix `true` in this assertion expresses that it is a safe approximation automatically inferred by the analysis. In fact, it over-approximates the success set of predicate `oddeven/2`.

The *assertion* "`:- regtype rt5/1`" indicates that the `rt5/1` predicate is a regular type. The regular type `arithexpression/1` is defined in a system library and expresses that its argument is an ISO Prolog arithmetic expression [14]. However, the definition of the regular type `rt5/1` has been inferred by CiaoPP's `eterms` type (shape) analyzer, which is based on abstract interpretation and a regular type abstraction with widening [23].

As another example, consider the *weight* module:

```
:- module(weight, [weight/2], [clp_projection])

 weight([A,B|C], E) <=>
     sumlist([A,B|C],S), weight([S|C],E).
```

```
weight([C], D) <=> D is C.
sumlist([A|C], S) <=>
    sumlist(C, T), S is A + T.
sumlist([], S) <=> S is 0.
```

Similarly to the present example, the CiaoPP system infers the following assertions:

```
:- true success weight(X,Y)
        => ( rt1(X), num(Y) ).
:- true success sumlist(_1,S)
        => ( list(_1,arithexpression), num(S) ).
:- regtype rt1/1.
rt1([A|B]) :-
        arithexpression(A),
        list(B,arithexpression).
```

Here `list/1` refers to the regular type for standard lists (defined in the system libraries), and `num` refers to the ISO numbers (i.e., floating point or integer numbers).

CiaoPP is also able to analyze confluent multi-head programs. For instance, consider the following rules, forming part of an arc consistent finite domain (see section 8.2.3 in Früwirth's book [7]).

```
inconsistency @ X in A:B <=> A > B | false.
intersection  @ X in A:B, X in C:D <=>
    X in max(A, C):min(B, D).
instantiation @ X in A:A <=> X is A.
```

CiaoPP inferred the expected assertions:

```
:- true pred X in _2
        => ( number(X), rt0(_2) ).

:- regtype rt0/1.
rt0(A:B) :-
        arithexpression(A),
        arithexpression(B).
```

Note that the type analyses we can perform on CLP projections are complementary to the ones we could perform on the Pilozzi et al.'s transformation. Indeed, while the CLP projection preserves the success set, Pilozzi et al's transformation preserves the call set but not the set of successes [18].

6.3 Upper bound complexity analysis for CHR_1

Theorem 19 ensures that the least upper bound complexity for a CLP projection provides a safe upper bound for the projected CHR program. Theorem 21 goes further guaranteeing that this upper bound is accurate as far as states that are data-sufficient with respect single-head programs are concerned. Consequently, we can infer precise complexity upper-bound for CHR_1 program from its projection. Although this approach is limited to single-head programs, it provides the first automatic tool for obtaining complexity upper bounds for CHR.

Once again, we can use the CiaoPP system, which is able to infer such bounds for CLP programs [16]. For example, consider the *oddeven* module given in the previous section. To infer proper bounds, the system needs an *entry* declaration specifying the way in which the external calls to an atom will occur, i.e., how the atoms will be called form outside. For instance, the following declaration states that oddeven will be called with a number as the first argument and a variable as the second one:

```
:- entry oddeven(X,Y)
        : ( num(X), var(B) ).
```

Once such information has been added to the original program file, the system infers the following assertion:

```
:- true pred oddeven(X,Y)
  : ( num(X), var(B) )
 => ( num(X), rt5(Y),
      size_ub(X,int(X)), size_ub(Y,1.0) )
  + steps_ub(0.25*exp(-1.0,int(X))+0.5*int(X)+0.75).
```

This assertion includes a lot of information: the second line after the colon (`:`) contains the *preconditions*, and states that the condition specified by the `entry` declaration (`num(X), var(B)`) also holds for the recursive calls to `oddeven/2`'. The third and fourth lines, after the double arrow (`=>`), show the *postconditions* including the type of the arguments as inferred in previous subsection together with a size upper bound for the arguments on success (`int(X)` stands for the integer value of `X`). Finally, the field in the fifth line (after the `+`) shows the inferred complexity upper bound (in number of CSLD steps). Thanks to Theorem 21 we know that this upper bound provides a safe upper bound for the longest derivation with respect the original, which is as precise as it is for the projection.

Similarly, we can analyze the *weight* module and obtain:

```
:- true pred weight(A,B)
  : ( list(A,arithexpression), var(B) )
 => ( rt5(A), num(B),
      size_ub(A,length(A)), size_ub(B,bot) )
  + steps_ub(0.5*exp(length(A),2)+2.5*length(A)-2.0).
:- true pred sumlist(_1,S)
  : ( list(_1,arithexpression), var(S) )
 => ( list(_1,arithexpression), num(S),
      size_ub(_1,length(_1)), size_ub(S,bot) )
  + steps_ub(length(_1)+1).
```

where `length(A)` stands for the length of the list `A`.

Note that this result underlines once again the advantage of the direct translation of CHR_1 programs into CLP with respect the meta-interpreter approach of Pilozzi. Indeed, we were able to infer a bound for the *weight* program from the CLP projection, while Figure 1 illustrates it is already difficult to prove its termination using Pilozzi's translation.

7. Conclusions

We have introduced and studied the notion of CLP projection for Constraint Handling Rules (CHR). We have shown that the CLP projection provides a safe operational and declarative approximation for CHR programs. We have also shown that the least fixpoint of a confluent program is the same as that of its projection (and in doing so we have made some contributions to the logical foundations of CHR).

We have hopefully demonstrated that the CLP projection is a promising theoretical (and also practical) tool for the study and analysis of CHR programs. The CLP projection provides a good semantic approximation that is complementary to previous work. In particular, existing analyzers are good for the analysis of termination properties when the latter rely on multiplicity of atoms in the store. On the other hand the use of the CLP projection for termination proofs appears advantageous when termination does not relies on multiplicity of atoms in the store. Furthermore, our approach provides the first method (to the best of our knowledge) for providing cost bounds for CHR programs.

As future work it seems interesting to explore, within the CLP projection approach, the possibility of adding information about the multiplicity of atoms in store to be able to prove termination of multi-headed programs.

References

[1] S. Abdennadher, T. Frühwirth, and H. Meuss. Confluence and semantics of constraint simplification rules. *Constraints*, 4(2): 133–165, 1999.

[2] D. Cabeza and M. Hermenegildo. A new module system for Prolog. In *International Conference on Computational Logic*, number 1861 in Lecture Notes in Artificial Intelligence, pages 131–148. Springer, 2000.

[3] K. L. Clark. Negation as failure. In *Logic and Data Bases*. Plenum, 1978.

[4] M. Codish and C. Taboch. A semantic basis for the termination analysis of logic programs. *J. Log. Program.*, 41(1): 103–123, 1999.

[5] P. Dart and J. Zobel. A regular type language for logic programs. In *Types in Logic Programming*, pages 157–187. MIT Press, 1992.

[6] G. J. Duck. *Compilation of Constraint Handling Rules*. PhD thesis, University of Melburne, 2005.

[7] T. Frühwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.

[8] J. Giesl, P. Schneider-Kamp, and R. Thiemann. Aprove 1.2: Automatic termination proofs in the dependency pair framework. In *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 281–286. Springer, 2006.

[9] C. D. Giusto, M. Gabbrielli, and M. Meo. On the expressive power of multiple heads in CHR. *To appear in ACM Transactions on Computational Logic*.

[10] R. Haemmerlé. (Co)-inductive semantics of Constraint Handling Rules. *To appear in Theory and Practice of Logic Programming, (ICLP'11 Special Issue)*, 2011.

[11] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor). *Science of Computer Programming*, 58(1–2), 2005.

[12] C. Holzbaur. Metastructures vs. Attributed Variables in the Context of Extensible Unification. In *International Symposium on Programming Language Implementation and Logic Programming (PLIP)*, volume 631 of *Lecture Notes in Computer Science*, pages 260–268. Springer, August 1992.

[13] C. Holzbaur and T. W. Frühwirth. Compiling constraint handling rules into prolog with attributed variables. In *International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)*, volume 1702 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 1999.

[14] *Information technology – Programming languages – Prolog – Part 1: General core*. International Organization for Standardiztion, 1995. ISO/IEC 13211-1.

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

[16] J. Navas, E. Mera, P. López-García, and M. Hermenegildo. User-definable resource bounds analysis for logic programs. In *International Conference on Logc-Programming (ICLP)*, volume 4670 of *Lecture Notes in Computer Science*. Springer, 2007.

[17] P. Pilozzi and D. D. Schreye. Proving termination by invariance relations. In *International Conference on Logc-Programming (ICLP)*, volume 5649 of *Lecture Notes in Computer Science*, pages 499–503. Springer, 2009.

[18] P. Pilozzi, T. Schrijvers, and D. De Schreye. Proving termination of CHR in Prolog: A transformational approach. In *International Workshop on Termination (WST)*, June 2007.

[19] P. Pilozzi, T. Schrijvers, and M. Bruynooghe. A transformational approach for proving properties of the chr constraint store. In *International Symposium on Logic Program Synthesis and Transformation (LOPSTR)*, volume 3037 of *Lecture Notes in Computer Science*, pages 22–36. Springer, 2009.

[20] G. Puebla, F. Bueno, and M. Hermenegildo. An assertion language for constraint logic programs. In *Analysis and Visualization Tools for Constraint Programming*, number 1870 in Lecture Notes in Computer Science, pages 23–61. Springer, 2000.

[21] F. Raiser, H. Betz, and T. Frühwirth. Equivalence of CHR states revisited. In *Workshop on Constraint Handling Rules*, Report CW 555:34–48. Kath. Univ. Leuven, 2009.

[22] T. Schrijvers and B. Demoen. The k.u.leuven chr system: Implementation and application. In *Workshop on Constraint Handling Rules*, pages 1–5, 2004.

[23] C. Vaucheret and F. Bueno. More precise yet efficient type inference for logic programs. In *International Static Analysis Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 102–116. Springer, September 2002.