On Combining Backward and Forward Chaining in Constraint Logic Programming

Rémy Haemmerlé [*]

Universidad Politécnica de Madrid & IMDEA Software Institute, Spain

Abstract

We address the problem of designing constraint logic languages that usefully combine backward and forward chaining in a sound and complete way. Following the approach of Constraint Logic Programming, we define a class of programming languages that generalize both Constraint Logic and Concurrent Constraint Programming. Syntactically, this class corresponds to Constraint Handling Rules with disjunctions, but differ operationally by featuring set-based semantics instead of multiset-based ones; i.e., conjunction and disjunction are idempotent. The assumption of program confluence is the crux on which both the committed choice strategy and the logical completeness of the languages rely.

Keywords CLP, backward & forward chaining, CHR, confluence.

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

1. Introduction

Broadly understood, Constraint Logic Programming (CLP) is a programming paradigm in which programs are viewed as sets of logical implications executed on a principle of automated inference guided by a constraint solver. One of the main challenges of this paradigm has always been the design of operationally efficient yet logically sound and complete inference mechanisms. This paradigm has given rise to a number of languages that can be classified according to the direction of the inference principle they work with.

On one hand, some languages use *backward chaining* (or *top-down reasoning*), an inference principle that works from consequents toward antecedents. Among these languages are found the original CLP languages introduced by Jaffar and Lassez [21]. Backward chaining languages are characterized by their operational locality: only one atomic object, usually called *goal*, is processed at a time. This locality property gives such languages a high inference rate (number of inference steps by unit of time). On the downside, these languages yield a high level of nondeterminism that must be handled by backtracking, often repeating the same inferences during a computation, and may explore possibly infinite parts of the search space where no solution exists.

On the other hand are the languages that use *forward chaining* (or *bottom-up reasoning*), an inference principle that works from antecedents toward consequents. The constraint databases [23] and the Concurrent Constraint (CC) languages [32] constitute prominent examples of forward chaining paradigms. Operationally, these languages differ from backward chaining languages in that they are global: they work in parallel with a set of objects, often called *store*. In the case of CC, those objects are suspensions whose possible wake-up must be considered each time a variable is further constrained. This operational globality provides forward chaining languages with better control over execution. It also limits backtracking, as inference steps become generally committed-choice. On the downside, their inference rate of those languages is low, and may even decrease as the store grows. Furthermore such language may produce inference that do not contribute in any way to the answer of the query.

It is natural to expect that the general efficiency of constraint logic systems could be improved by combining the two kinds of chaining. For instance, while carrying out a backward reasoning, it may be advantageous to use forward reasoning to detect and cut out those branches in the search tree which are trivially inconsistent.

Example 1. Consider the following set $\mathcal{P}_1$ of Horn clauses that defines the "strictly less" relation over natural numbers:

$$0 < \mathrm{s}(x) \leftarrow \top \qquad \mathrm{s}(x) < \mathrm{s}(y) \leftarrow x < y$$

When processed by standard backward chaining with respect to $\mathcal{P}$, a query such as $(x < y \wedge y < x)$ loops without encountering any solution. To avoid such pointless computation, the programmer may use the asymmetry of the "strictly less" relation. This property, which guarantees the above query has no solution, can be specified by the multiheaded implication:

$$x < y \wedge y < x \rightarrow \bot$$

To be effective, this rule must be handled by forward chaining interleaved with the main backward chaining computation. In particular, it must be applied as soon the antecedent is subsumed by the set of goals currently being processed.

Conversely, when processing a program forward, one may want to use backward reasoning to detect and discard from the store redundant or tautological objects.

[*] The research leading to the results presented in this paper has received funding from the Programme for Attracting Talent/young PHDs of the Montegancedo Campus of International Excellence *PICD*, the European Union 7th Framework Programme under grant agreement 318337 *ENTRA*, Spanish MINECO TIN'12-39391 *StrongSoft* and TIN'08-05624 *DOVES* projects, and Madrid TIC-1465 *PROMETIDOS-CM* project.

PPDP'14, September 8-10, 2014, Canterbury, UK.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2947-7…$15.00.
http://dx.doi.org/10.1145/2643135.2643144

Example 2. Consider the following clauses that enforce the properties of transitivity and asymmetry on the "strictly less" relation presented in the previous example:

$$x < y \wedge y < z \rightarrow y < z \qquad x < y \wedge y < x \rightarrow \bot$$

If handled by forward chaining, the first clause computes the transitive closure of the relation $<$ defined by the atoms of the store, while the second rule concurrently ensures that no two symmetric atoms are present at the same time. Thus, when processing a store that contains a chain of the form

$$0 < s(0) \wedge \ldots \wedge \mathrm{s}^{n-1}(0) < \mathrm{s}^{n}(0)$$

where $\mathrm{s}^{k}(x)$ stands for the k^{th} successor of x, the transitivity rule will add a quadratic number of atoms to the store (i.e., all the atoms $s^k(0) < s^l(0)$ s.t. $0 \leq k$, $k+1 < l$, and $l \leq n$).

To control this explosion of the atoms, the programmer might choose to use the definition of the relation $<$ given at Example 1 in a backward manner, which guarantees that such atoms are true in any model, and thus cannot contribute to any proof of inconsistency.

However, to the best of our knowledge no attempt to combine backward and forward chaining while preserving strict logical completeness has succeeded so far.

In this paper, we advance the thesis that the two forms of chaining can be combined as described above while preserving logical completeness. Our starting point is the class of CHR (Constraint Handling Rules) languages [13], and more precisely its disjunctive extension, CHR$^{\vee}$ [4].

CHR programs consist of sets of guarded rules that rewrite multisets of constrained atoms in a committed-choice manner. Declaratively, they can be understood as sets of logical equivalences executed on a particular forward inference principle, which can consume the heads of the rule while preserving the logical equivalence of the store. The main advantage of this inference mechanism over standard forward chaining is that it allows a non-monotonic evolution of the store, hence limiting the explosion of atoms in the store. On the downside, the language is logically incomplete in the general case. This problematic behavior is due to the multiset nature of CHR: The multiplicity of atoms matters for rule applications, while it obviously does not from the point of view of logical deduction.

CHR$^{\vee}$ extends CHR by allowing disjunctions in the body of rules. Originally, it was introduced as a general query language that allows for backward and forward chaining in a unified framework. Although CHR$^{\vee}$ opened promising perspectives, the formal study of its logical foundations and theoretical properties has been elusive. A noticeable exception is the work of Betz and Frühwirth [6], which shows that the multiset semantics of CHR$^{\vee}$ is actually better understood as pure forward chaining in linear logic. From the point of view of classical logic, however, CHR$^{\vee}$ still faces the same problems of incompleteness as its nondisjunctive ancestor.

In order to overcome these logical weaknesses, we propose in Section 2 an analytical set-based semantics (i.e., where conjunction and disjunction within states are idempotent) for CHR$^{\vee}$. This set-based approach differs form other implementation-oriented set-based semantics [11, 34] in the sense that rules may consume (or not) atoms from their head in a nondeterministic manner. Formally, these semantics are obtained through an extension with disjunction of the equivalence-based semantics of Raiser et al. [29] that enforces the impotency of both conjunction and disjunction. It appears these general operational semantics capture backward chaining when the rules are logically read backward (as implications from right to left) and forward chaining when the rules are logically read forward (from left to right).

To determine the completeness of this language, we divide the study of its logical properties in two. In Section 3, we focus on the backward logical reading of programs. In particular, by using standard CLP tools, we establish that the smallest Tarski model of this backward reading can be characterized as the smallest fixed point over the so-called immediate consequence operator. Operationally, we show that the set of answers with respect to the backward chaining part of the general operational semantics fully abstracts this fixed point. Conversely, in Section 4, we concentrate on the forward logical reading. Specifically, we show the biggest Tarski models of this forward reading can be captured using a greatest fixed point over an operator dual to the one used previously. We also establish that the set of nonfailures with respect to the forward chaining part of the general operational semantics fully abstract this greatest fixed point.

Then, in Section 5, we combine the results of the two previous sections. In particular, we show that the least model of full logical reading (when the rules are read as equivalences) of a confluent program coincides with the least of its backward readings, and that, conversely, its greatest models coincide with the greatest of its forward readings. Based on this, we establish the full completeness of the language with respect to answers and accessible constraints.

Finally, implementation is briefly discussed in Section 6 and related work is covered in Section 7.

2. The General Framework

In this section, we introduce the syntax, the declarative semantics, and a general operational semantics of our language. For this purpose, we will assume a language of *(built-in) constraints* containing the equality $=$, $\bot$, and $\top$ over some theory $\mathcal{T}$. We define *(user-defined) atoms* using a different set of predicate symbols.

In the following, variables will be denoted by lower case letters from the end of the alphabet, such as x, y, z, $\ldots$, while atoms and atomic constraints will be indicated by lowercase letters from the beginning of the alphabet, such as a, b, c, d, $\ldots$ Sets of constraints and atoms will be denoted by blackboard capital letters, such as $\mathbb{A}$, $\mathbb{B}$, $\mathbb{C}$, $\mathbb{D}$. By a slight abuse of notation, we will confuse conjunctions and sets of constraints and atoms, forget braces around such sets, and use commas for their unions. We will use $\mathrm{fv}(\phi)$ to denote the set of free variables of a formula ϕ.

2.1 States

A *branch* is a tuple $(\mathbb{A}|\mathbb{C})$, where $\mathbb{A}$ (the *user store*) is a finite set of user-defined atoms and $\mathbb{C}$ (the *built-in store*) is a finite conjunction of atomic built-in constraints. Sets of branches can be denoted by Greek letters, Δ or Γ. By a slight abuse of notation, we will confuse disjunctions and sets of branches, forget braces around such sets, and use semicolons for their unions. A *state* is a non-empty tuple of the form:

$$\langle (\mathbb{A}_1|\mathbb{C}_1)\,; \ldots ; (\mathbb{A}_n|\mathbb{C}_n) \rangle_{\bar{x}}$$

where the $(\mathbb{A}_i|\mathbb{C}_i)$ are branches, and $\bar{x}$ (the *global variables*) is a finite sequence of variables. Unsurprisingly, the *local variables* of a state are those variables of the state that are not global. When no confusion can occur, we will syntactically merge user atoms and built-in constraints within a branch. For the sake of conciseness, we will sometimes denote disjunctions of branches within a state as a finite family. For instance, the above state may be denoted by $\left\langle (\mathbb{A}_i|\mathbb{C}_i)_{i \in I} \right\rangle_{\bar{x}}$, where I stands for the set of indices $\{1, \ldots, n\}$. The set of all states will be denoted by Σ.

Branches may be logically understood as the conjunction of their atoms and constraints. States may be understood as the disjunction of their branches where local variables are implicitly existentially quantified. Formally, the *logical reading* of the above state is the quantified disjunctive normal form:

$$\exists_{\text{-}\bar{x}} (\mathbb{A}_1 \wedge \mathbb{C}_1 \vee \cdots \mathbb{A}_n \wedge \mathbb{C}_n)\ ^1$$

where the notation $\exists_{\text{-}\bar{x}}\psi$ denotes the existential closure of the formula ψ with the exception of the variables $\bar{x}$. The logical reading of a state S will be denoted by $S^\dagger$.

2.2 Programs

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

$$r \ @\ \mathbb{H} \Longleftrightarrow \mathbb{G} \mid (\mathbb{B}_1|\mathbb{C}_1); \ldots ; (\mathbb{B}_n|\mathbb{C}_n).$$

where $\mathbb{H}$ (the *head*) is a nonempty set of atoms, $\mathbb{G}$ (the *neck* or *guard*) is a conjunction of constraints, the $(\mathbb{B}_i|\mathbb{C}_i)$ are branches forming the so-called *body*, and r is an identifier assumed unique in the program. An empty guard $\top$ can be omitted with the symbol $|$. For the sake of simplicity, we will always assume the guard and the body of a same rule share only variables that appear in their head. Disjunctions of branches within the body of a rule may be denoted by a finite family of branches in a way similar to branches within a state. The *local variables* of a rule are those variables that appear free in the body but do not appear either in the head or in the guard.

In the following, we distinguish special cases of rules.

- The so-called *backward rules*, which are those rules with the head repeated as an independent branch in the body; that is to say, rules of the form:

  $$r \ @\ \mathbb{H} \Longleftrightarrow \mathbb{G} \mid (\mathbb{B}_1|\mathbb{C}_1)\,; \ldots ; (\mathbb{B}_n|\mathbb{C}_n)\,; (\mathbb{H}|\mathbb{G}\rho)\,.$$

  where ρ renames $(\mathrm{fv}(\mathbb{G}) \setminus \mathrm{fv}(\mathbb{H}))$ by fresh variables.
  Such a rule can be written with the alternative syntax:

  $$r \ @\ \mathbb{H} \Longleftarrow \mathbb{G} \mid (\mathbb{B}_1|\mathbb{C}_1)\,; \ldots ; (\mathbb{B}_n|\mathbb{C}_n)\,.$$

- The so-called *forward rules* (or propagation rules), which are those rules with the head repeated in each branch of the body; that is to say, rules of the form:

  $$r \ @\ \mathbb{H} \Longleftrightarrow \mathbb{G} \mid (\mathbb{H}, \mathbb{B}_1|\mathbb{G}\rho, \mathbb{C}_1)\,; \ldots ; (\mathbb{H}, \mathbb{B}_n|\mathbb{G}\rho, \mathbb{C}_n)\,.$$

  where ρ renames $(\mathrm{fv}(\mathbb{G}) \setminus \mathrm{fv}(\mathbb{H}))$ by fresh variables.
  Such a rule can be written with the alternative syntax:

  $$r \ @\ \mathbb{H} \Longrightarrow \mathbb{G} \mid (\mathbb{B}_1|\mathbb{C}_1)\,; \ldots ; (\mathbb{B}_n|\mathbb{C}_n)\,.$$

Before considering the logical reading of programs, we introduce our running example. This is an enhanced version of the classic CHR introductory example.

Example 3. Assume $\mathcal{T}$ is the equality theory as defined by Clark [10], and consider the following program $\mathcal{P}_3$:

$$\begin{array}{lll}
\textit{leq\_0} & @ & 0 \leq x \Longleftarrow \top. \\
\textit{leq\_s} & @ & \mathrm{s}(x) \leq \mathrm{s}(y) \Longleftarrow x \leq y. \\
\textit{transitivity} & @ & x \leq y, y \leq z \Longrightarrow x \leq z. \\
\textit{antisymmetry} & @ & x \leq y, y \leq x \Longleftrightarrow x = y.
\end{array}$$

The two first rules define the "less or equal" predicate, as could be done in classical logic programming. The two following rules define the three usual properties of the partial order. On one hand, the rule *transitivity* states that the relation is transitive. On the other hand, the rule *antisymmetry* captures the antisymmetry and reflexivity properties, reading from left to right and right to left, directions respectively.

Logically, a rule can be understood as a logical equivalence between the head and the body. Formally, the *backward (logical) reading* of the rule given at the beginning of this section is the logical implication:

$$\forall\,(\mathbb{G} \wedge (\mathbb{B}_1 \wedge \mathbb{C}_1 \vee \cdots \vee \mathbb{B}_n \wedge \mathbb{C}_n) \rightarrow \mathbb{H})$$

while its *forward (logical) reading* is the implication:

$$\forall\,(\mathbb{G} \wedge \mathbb{H} \rightarrow \exists_{\text{-}\mathbb{H}}\,(\mathbb{B}_1 \wedge \mathbb{C}_1 \vee \cdots \vee \mathbb{B}_n \wedge \mathbb{C}_n))$$

The *backward* (respectively, *forward*) *reading* of a program $\mathcal{P}$ is the conjunction of the backward (forward) readings of its rules, and is denoted by $\overleftarrow{\mathcal{P}}$ ($\overrightarrow{\mathcal{P}}$). The *logical reading* of a program $\mathcal{P}$, denoted by $\overleftrightarrow{\mathcal{P}}$, is the the conjunction of its backward and forward readings.

Note that the backward reading of a forward rule is of the form $\forall\,(\phi \wedge \psi_1 \vee \cdots \vee \phi \wedge \psi_m \rightarrow \phi)$, while the forward reading of a backward rule is of the form $\forall\,(\phi \rightarrow \psi \vee \phi)$. In other words, both are tautologies and can thus be safely ignored. It is also worth noting the backward reading of a rule is of the form

$$\forall(\phi_1 \vee \cdots \vee \phi_m \rightarrow a_1 \wedge \cdots \wedge a_n)$$

which is equivalent to the conjunction of the Horn clauses

$$\{\forall(\phi_i \rightarrow a_j) \mid i \in 1, \ldots, m \ \&\ j \in 1, \ldots, n\}$$

In other words, one may logically understand the backward reading of a program as a standard CLP program.

Example 4. The backward reading of the program $\mathcal{P}_3$ given at Example 3 is (after removing tautological clauses):

$$\begin{array}{l}
\forall x\,(\top \rightarrow 0 \leq x) \wedge \\
\forall xy\,(x \leq y \rightarrow \mathrm{s}(x) \leq \mathrm{s}(y)) \wedge \\
\forall xy\,(x = y \rightarrow x \leq y)
\end{array}$$

while its forward reading is:

$$\begin{array}{l}
\forall xyz\,(x \leq y \wedge y \leq z \rightarrow x \leq y \wedge y \leq z \wedge x \leq z) \wedge \\
\forall xy\,(x \leq y \wedge y \leq x \rightarrow x = y)
\end{array}$$

2.3 Analytical Operational semantics

The operational semantics of our language can be represented by a simple transition relation defined modulo a state equivalence.

2.3.1 State subsumption, state equivalence

The following state equivalence is an extension of Raiser et al.'s equivalence [29], which handles disjunctions and enforces the idempotence of both conjunction and disjunction. In particular, this equivalence is defined as the least symmetric relation containing a notion of state subsumption. In the rest of this paper, state subsumption will be an invaluable tool for characterizing some fundamental aspects of our semantics (in particular, over- and under-approximation).

Definition 5 (State Subsumption & State Equivalence). Given two states $\langle(\mathbb{A}_i|\mathbb{C}_i)_{i\in I}\rangle_{\bar{x}}$ and $\langle(\mathbb{B}_j|\mathbb{D}_j)_{j\in J}\rangle_{\bar{y}}$, we will say that the branch $(\mathbb{A}_i|\mathbb{C}_i)$ *covers* the branch $(\mathbb{B}_j|\mathbb{D}_j)$ ($i \in I$ and $j \in J$), if the following implication holds:

$$\mathcal{T} \vDash \mathbb{C}_i \rightarrow \exists_{\text{-}(\mathbb{A}_i,\bar{x})}\,(\mathbb{D}_j \wedge (\mathbb{A}_i \sqsupseteq \mathbb{B}_j))$$

with the side condition that $\mathrm{fv}(\mathbb{A}_i, \mathbb{C}_i) \cap \mathrm{fv}(\mathbb{B}_j, \mathbb{D}_j) \subseteq \bar{x} \cap \bar{y}$.

State subsumption is the least partial order $\sqsupseteq$ closed under alpha-renaming of local variables that satisfies $R \sqsupseteq S$ if every branch of R covers some branch of S. *State equivalence* is then equivalence induced by state subsumption, i.e., $R \equiv S$ iff $R \sqsupseteq S$ and $S \sqsupseteq R$.

The state subsumption can be roughly understood as a containment order where the conjunction is interpreted as a union and the

[1] Within this paper we use the usual precedence for logical connectives, that is $\exists$, $\forall$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$ in the order of decreasing precedence.

disjunction as an intersection. The main difference between state equivalence and the full logical equivalence rests on the fact that incomparable (in the sense of logical entailment) built-in stores cannot be merged into a common branch. (See the non-equivalence (3) in Example 6.) This property captures the fact that, in general, constraint solvers poorly deal with the implications involving disjunctive formulas. In the following $\Sigma_{\equiv}$ will denote the quotient of the set Σ by $\equiv$ and $[S]$ the equivalence class of the state S by $\equiv$.

Example 6. Let us assume $\mathcal{T}$ be an axiomatization for a finite domain solver [18], where a constraint of the form (x in $t_1..t_2$) forces the value of x to be an integer between t_1 and t_2.

Consider first the subsumption (1):

$$\langle(\mathrm{p}(x)|x \text{ in } 0..2)\,;(\mathrm{p}(x),\mathrm{q}(x)|x \text{ in } 0..3)\rangle \sqsupseteq \langle(\mathrm{p}(x)|x \text{ in } 0..3)\,;\Gamma\rangle$$

The relation holds because both branches $(\mathrm{p}(x)|x \text{ in } 0..2)$ and $(\mathrm{p}(x),\mathrm{q}(x)|x \text{ in } 0..3)$ cover $(\mathrm{p}(x)|x \text{ in } 0..3)$. We can see that a branch covers another branch if it contains more "information". This additional information can take the form of stronger built-in constraints (second branch case), i.e.,

$$\mathcal{T} \vDash x \text{ in } 0..2 \to x \text{ in } 0..3 \wedge (\{\mathrm{p}(x)\} \supseteq \{\mathrm{p}(x)\})$$

or additional user atoms (first branch case), i.e.,

$$\mathcal{T} \vDash x \text{ in } 0..3 \to x \text{ in } 0..3 \wedge (\{\mathrm{p}(x),\mathrm{q}(x)\} \supseteq \{\mathrm{p}(x)\})$$

Note that the subsumption holds for an arbitrary Γ, since the definition of $\sqsupseteq$ only requires that the right-hand side branches cover some (but not all) left-hand side branches.

Consider now the equivalence (2):

$$\langle(\mathrm{p}(x),\mathrm{q}(x)|x \text{ in } 0..3)\,;(\mathrm{p}(x)|x \text{ in } 0..2)\rangle \equiv \langle(\mathrm{p}(x)|x \text{ in } 0..2)\rangle$$

This latter holds because the branches $(\mathrm{p}(x),\mathrm{q}(x)|x \text{ in } 0..3)$ and $(\mathrm{p}(x)|x \text{ in } 0..2)$ both cover the branch $(\mathrm{p}(x)|x \text{ in } 0..2)$ (for the left-to-right subsumption) and because the branch $(\mathrm{p}(x)|x \text{ in } 0..2)$ covers the branch $(\mathrm{p}(x)|x \text{ in } 0..2)$ (for the reverse subsumption). This relation illustrates how state equivalence can prune (when used from left to right) or create (when used from right to left) branches that contain more atoms or are more constrained.

Now consider the nonequivalence (3):

$$\langle(\mathrm{p}(x)|x \text{ in } 0..3)\rangle \not\equiv \langle(\mathrm{p}(x)|x \text{ in } 0..1)\,;(\mathrm{p}(x)|x \text{ in } 2..3)\rangle$$

This illustrates that logically equivalent states may not be equivalent with respect to $\equiv$. Indeed, despite the fact that the two states have equivalent logical readings, the left-hand branch covers none of the branches in the right-hand state.

Lastly, consider the equivalences (4) and (5):

$$\langle(\emptyset|\top)\,;\Gamma\rangle_{\bar{x}} \equiv \langle(\emptyset|\top)\rangle_{\bar{x}} \quad \text{and} \quad \langle(\emptyset|\bot)\rangle_{\bar{x}} \equiv \left\langle(\mathbb{A}_i|\mathbb{C}_i)_{i\in I}\right\rangle_{\bar{x}}$$

The left-hand side equivalence holds for any set of branches Γ, i.e., any branch covers $(\emptyset|\top)$. However the right-hand size equivalence holds only if $\mathcal{T} \vDash \mathbb{C}_i \to \bot$ holds for any $i \in I$, i.e., only inconsistent branches cover $(\emptyset|\bot)$.

2.3.2 Analytical Operational Semantics

Once the state equivalence has been defined, the operational semantics can be stated by a single rule.

Definition 7 (General operational semantics). Formally, the *analytical (operational) semantics*, is the least binary relation $\xrightarrow{\mathcal{P}}$ on $\Sigma_{\equiv}$ satisfying for a given program $\mathcal{P}$ the following rule:

$$\frac{\left(r \mathbin{@} \mathbb{H} \Longleftrightarrow \mathbb{G} \mid (\mathbb{B}_i|\mathbb{C}_i)_{i\in I}\right) \in \mathcal{P}\rho}{\left[\langle(\mathbb{H},\mathbb{A}|\mathbb{G},\mathbb{D})\,;\Gamma\rangle_{\bar{x}}\right] \xrightarrow{\mathcal{P}} \left[\left\langle(\mathbb{B}_i,\mathbb{A}|\mathbb{G},\mathbb{C}_i,\mathbb{D})_{i\in I}\,;\Gamma\right\rangle_{\bar{x}}\right]}$$

where ρ renames the local variables of $\mathcal{P}$ with fresh variables. $\xrightarrow{\mathcal{P}}^*$ will denote the reflexive transitive closure of $\xrightarrow{\mathcal{P}}$. For all states S and S' we may write $S \xrightarrow{\mathcal{P}} S'$ instead of $[S] \xrightarrow{\mathcal{P}} [S']$.

Example 8. Consider the rule *antisymmetry* of the program $\mathcal{P}_3$ given at Example 3. The analytical operational semantics can handle nondeterministically such a rule in three different ways, depending how state equivalence is used. First, they can apply the rule in a "combined mode", replacing the head of a rule by its body in a committed choice way, exactly as classical CHR would do.

$$\langle(x \leq y, y \leq x)\rangle_{xy} \xrightarrow{\mathcal{P}_3} \langle(x = y)\rangle_{xy}$$

Alternatively, they can process the rule in a "backward mode". In this case, a new branch is first created by duplicating and further constraining an existing branch in such a way that the rule becomes applicable. Then the rule proceeds by applying to this new branch. Such a creation of branches captures the "don't know" nondeterminism of backward chaining.

$$\begin{aligned}\langle(x \leq y)\rangle_{xy} &\equiv \langle(x \leq y, y \leq x)\,;(x \leq y)\rangle_{xy}\\ &\xrightarrow{\mathcal{P}_3} \langle(x = y)\,;(x \leq y)\rangle_{xy}\end{aligned}$$

Finally, the semantics can treat the rule in a "forward mode". In this case, the hypotheses of the rule within a branch are duplicated by state equivalence before the rule is applied. In this way, none of the atoms of the original branch will be consumed by the transition. The fact that this application strategy does not create more branches than the body of a rule imposes gives an account of the "don't care" nondeterminism of forward chaining.

$$\begin{aligned}\langle(x \leq y, y \leq x)\rangle_{xy} &\equiv \langle(x \leq y, x \leq y, y \leq x, y \leq x)\rangle_{xy}\\ \xrightarrow{\mathcal{P}_3} \langle(x \leq y, y \leq x, x = y)\rangle_{xy} &\equiv \langle(x \leq x, x = y)\rangle_{xy}\end{aligned}$$

The precise operational semantics depend on the choice of observable. We shall consider *successes*, *nonfailures*, *answers*, and *accessible constraints*.

Definition 9 (Observables). Let $\mathcal{P}$ be a program.

- A *success* for $\mathcal{P}$ is a state S such that $S \xrightarrow{\mathcal{P}}^* \langle(\emptyset|\top)\rangle$.
- A *nonfailure* for $\mathcal{P}$ is a state S such that $S \not\xrightarrow{\mathcal{P}}^* \langle(\emptyset|\bot)\rangle$.
- An *answer* for a state S w.r.t. $\mathcal{P}$ is a quantified conjunction of constraints $\exists_{\text{-}\bar{x}}\mathbb{C}$ s.t. $S \xrightarrow{\mathcal{P}}^* \langle(\emptyset|\mathbb{C})\,;\Gamma\rangle_{\bar{x}}$.
- An *accessible constraint* for a state S w.r.t. $\mathcal{P}$ is a quantified conjunction of constraints $\exists_{\text{-}\bar{x}}\mathbb{C}$ s.t. $S \xrightarrow{\mathcal{P}}^* \left\langle(\mathbb{B}_i|\mathbb{C},\mathbb{D}_i)_{i\in I}\right\rangle_{\bar{x}}$.

Note that unlink within CLP, in our language the set of answers for a state is necessarily closed by subsumption. Indeed as explained in Example 6 if $\mathcal{T} \vDash \mathbb{D} \to \exists\bar{x}\mathbb{C}$ then $\langle(\emptyset|\mathbb{C})\,;\Gamma\rangle_{\bar{x}} \equiv \langle(\emptyset|\mathbb{D})\,;(\emptyset|\mathbb{C})\,;\Gamma\rangle_{\bar{x}}$, in other words if $\exists_{\text{-}\bar{x}}\mathbb{C}$ is answer for some state S, then the stronger constraint $\exists_{\text{-}\bar{x}}\mathbb{D}$ is also answer for S.

As the name suggests, analytic semantics are specially designed to be as close as possible to the logical meaning of programs, but not to be easily implementable. In particular, it may seem that whenever the program is not empty, it will never terminate. The reason for this is illustrated by the following example: it is always possible to convert a given state into an equivalent one that contains the head of an arbitrary rule, hence making the rule applicable. In particular the answers are not terminating in our framework. Nevertheless it is worth noting that application of any rule on an answer is idempotent: if $\langle(\emptyset|\mathbb{C})\rangle_{\bar{x}} \xrightarrow{\mathcal{P}} S$ then $S \equiv \langle(\emptyset|\mathbb{C})\rangle_{\bar{x}}$. The issue of non-termination is discussed later in Example 6.

Example 10. Consider the rule *antisymmetry* of the program $\mathcal{P}_3$ given at Example 3 and a consistent state of the form $\langle(\mathbb{A}|\mathbb{C})\rangle_{\bar{x}}$

that does not contain any atom built with the $\leq$ symbols.

$$\begin{aligned}&\langle(\mathbb{A}|\mathbb{C})\rangle_{\bar{x}} \equiv \langle(x \leq y, y \leq x, \mathbb{A}|\mathbb{C})\,;(\mathbb{A}|\mathbb{C})\rangle_{\bar{x}}\\ \xrightarrow{\mathcal{P}_3}\ &\langle(x=y, \mathbb{A}|\mathbb{C})\,;(\mathbb{A}|\mathbb{C})\rangle_{\bar{x}} \equiv \langle(\mathbb{A}|\mathbb{C})\rangle_{\bar{x}}\end{aligned}$$

2.3.3 Logical Soundness

As in the original CHR, our language features a strong logical soundness. Basically, transitions preserve the logical meaning of states with respect to the constraint theory and the program.

THEOREM 11 (Soundness).

$$\textit{If } R \xrightarrow{\mathcal{P}}{}^{*} S\textit{, then } \mathcal{T}\overleftrightarrow{\mathcal{P}} \vDash R^{\dagger} \leftrightarrow S^{\dagger}$$

Unsurprisingly, completeness results are more difficult to establish. The three following sections will be dedicated to tackling this problem.

3. Backward Models

This section is dedicated to relating backward models of any program $\mathcal{P}$ (i.e. the Tarski's models of the backward reading of the program) to the backward chaining processing of the program. For this purpose, we introduce a restriction of the general operational semantics, the so-called *minimal semantics*, which process every rule as a backward rule, similarly to how CLP processes constrained Horn clauses.

3.1 Minimal Operational Semantics

The so-called minimal operational semantics prevent the rules from consuming any atoms in a branch: the result of each rule's application is stored in a new branch, while the original branches are kept intact. We argue that they capture the pure backward chaining transitions of the general semantics.

Definition 12 (Minimal Semantics). The *minimal (operational) semantics* of a program $\mathcal{P}$ are given by the least relation $\overset{\mathcal{P}}{\rightharpoondown}$ on $\Sigma_{\equiv}$ satisfying the following rule:

$$\frac{\left(r \ @\ \mathbb{H} \Longleftrightarrow \mathbb{G} \mid (\mathbb{B}_i|\mathbb{C}_i)_{i\in I}\right) \in \mathcal{P}\rho}{\left[\langle(\mathbb{H},\mathbb{A}|\mathbb{G},\mathbb{D});\Gamma\rangle_{\bar{x}}\right] \overset{\mathcal{P}}{\rightharpoondown} \left[\left\langle(\mathbb{B}_i,\mathbb{A}|\mathbb{G},\mathbb{C}_i,\mathbb{D})_{i\in I}\,;(\mathbb{H},\mathbb{A}|\mathbb{G},\mathbb{D});\Gamma\right\rangle_{\bar{x}}\right]}$$

where ρ renames the local variables of $\mathcal{P}$ with fresh variables. $\overset{\mathcal{P}}{\rightharpoondown}{}^{*}$ will denote the reflexive transitive closure of $\overset{\mathcal{P}}{\rightharpoondown}$. For all states S and S' we may write $S \overset{\mathcal{P}}{\rightharpoondown} S'$ instead of $[S] \overset{\mathcal{P}}{\rightharpoondown} [S']$.

The following example show how the minimal semantics apply on our running example.

Example 13. The following derivation step is obtained by applying the minimal semantics on the rule *antisymmetry* of the program $\mathcal{P}_3$ from Example 3. One may observe such step corresponds to the "backward mode" described at Example 8.

$$\begin{aligned}&\langle(x \leq y, y \leq z)\rangle_{xyz}\\ \equiv\ &\langle(x \leq y, y \leq x, x=z)\,;(x \leq y, x \leq z)\rangle_{xyz}\\ \overset{\mathcal{P}_3}{\rightharpoondown}\ &\langle(x=y, x=z)\,;(x \leq y, x \leq z)\rangle_{xyz}\end{aligned}$$

Note that any minimal derivation step corresponds to some possibly multi-step CLP derivation defined with respect to the backward reading of the rule that have been applied. In the present case the backward reading of *antisymmetry* is the clause $(x \leq y\ \leftarrow\ x=y)$ and the corresponding derivation is:

$$(x \leq y, y \leq z) \rightarrow_{\text{CLP}} (x=y, y \leq z) \rightarrow_{\text{CLP}} (x=y, x=z)$$

Inversely to any CLP derivation step corresponds a minimal step. For instance the first CLP step in the derivation above corresponds to the following minimal step:

$$\begin{aligned}&\langle(x \leq y, y \leq z)\rangle_{xyz}\\ \equiv\ &\langle(x \leq y, y \leq x, y \leq x)\,;(x \leq y, x \leq z)\rangle_{xyz}\\ \overset{\mathcal{P}_3}{\rightharpoondown}\ &\langle(x=y, y \leq z)\,;(x \leq y, x \leq z)\rangle_{xyz}\end{aligned}$$

One may remark that the minimal semantics are idempotent when applied to forward rules. For instance, consider the following step, obtained by applying the rule *transitivity* of $\mathcal{P}_3$:

$$\begin{aligned}&\langle(x \leq y, y \leq z)\rangle_{xyz}\\ \overset{\mathcal{P}_3}{\rightharpoondown}\ &\langle(x \leq y, y \leq z, x \leq z)\,;(x \leq y, y \leq z)\rangle_{xyz}\\ \equiv\ &\langle(x \leq y, y \leq z)\rangle_{xyz}\end{aligned}$$

The following proposition justifies the name of the semantics. It basically states that the minimal semantics underapproximate (in the sense of state subsumption) the general semantics.

PROPOSITION 14. *For any program $\mathcal{P}$ and all states S, S', and R the following propositions hold:* (Soundness) *If $S \overset{\mathcal{P}}{\rightharpoondown} S'$ then $S \xrightarrow{\mathcal{P}} S'$.* (Completeness) *If $S \xrightarrow{\mathcal{P}} S'$ and $S \sqsupseteq R$, then there exists R', such that $R \overset{\mathcal{P}}{\rightharpoondown}{}^{*} R'$ and $S' \sqsupseteq R'$.*

3.2 Inductive Semantics

In order to relate the logical model of the backward reading of a program and the minimal operational semantics, we borrow CLP theory's fixed point semantics. In fact, the present subsection simply transposes well-known results from the CLP state of the art [22] to our language.

3.2.1 Inductive Semantics w.r.t. Constraint Models

Let $\mathcal{S}$ be an interpretation of the constraint language. An $\mathcal{S}$-*interpretation* is an interpretation that agrees with $\mathcal{S}$ on the constraint symbols. The $\mathcal{S}$-*base*, denoted by $\mathcal{B}_{\mathcal{S}}$, is the set of atoms valued in $\mathcal{S}$. An $\mathcal{S}$-interpretation can thus be identified with a subset of $\mathcal{B}_{\mathcal{S}}$ formed of the atoms that are true in the interpretation. An $\mathcal{S}$-*model* for a formula ϕ is an $\mathcal{S}$-interpretation that models ϕ. A *backward $\mathcal{S}$-model* of a program $\mathcal{P}$ is an $\mathcal{S}$-model for $\overleftarrow{\mathcal{P}}$. Clearly, $\mathcal{B}_{\mathcal{S}}$ is the largest backward $\mathcal{S}$-model of every program. We will show now the existence of a unique least one.

Definition 15 (Inductive Semantics w.r.t. Constraint Models). Assume $\mathcal{S}$ is an interpretation of the constraint language. The *immediate (backward) consequences operator* with respect to a program $\mathcal{P}$ and $\mathcal{S}$ is the function $\langle\mathcal{P}\rangle^{\mathcal{S}} : 2^{\mathcal{B}_{\mathcal{S}}} \to 2^{\mathcal{B}_{\mathcal{S}}}$, defined as:

$$\begin{aligned}\langle\mathcal{P}\rangle^{\mathcal{S}}(X) \overset{\text{def}}{=} \{\ a\rho \mid &\left(\mathbb{H} \Longleftrightarrow \mathbb{G} |\ (\mathbb{B}_i|\mathbb{D}_i)_{i\in I}\right) \text{ is a rule of } \mathcal{P},\\ &\rho \text{ is an } \mathcal{S}\text{-valuation, and } j \text{ is an index in } I \text{ s.t.}\\ &\mathcal{S} \vDash (\mathbb{G} \wedge \mathbb{D}_j)\rho,\ \mathbb{B}_j\rho \subseteq X \text{ and } a \in \mathbb{H}\ \}\end{aligned}$$

The *inductive semantics* of a program $\mathcal{P}$ with respect to $\mathcal{S}$ is the set inductively defined by the immediate consequence operator associated with $\mathcal{P}$ and $\mathcal{S}$, i.e., the intersection of the prefixed points of $\langle\mathcal{P}\rangle^{\mathcal{S}}$.

$$\mathrm{I}^{\mathcal{S}}_{\mathcal{P}} \overset{\text{def}}{=} \bigcap \left\{X \mid \langle\mathcal{P}\rangle^{\mathcal{S}}(X) \subseteq X\right\}$$

The inductive semantics provides a least logical model for the backward reading of a program.

THEOREM 16 (Least backward $\mathcal{S}$-model). *Let $\mathcal{S}$ be an model for $\mathcal{T}$, and $\mathcal{P}$ be a program. $\overleftarrow{\mathcal{P}}$ has a least $\mathcal{S}$-model. This latter is equal to $\mathrm{I}^{\mathcal{S}}_{\mathcal{P}}$.*

3.2.2 Inductive Semantics w.r.t. the Constraint Theory

The denotational semantics can alternatively be defined on the lattice of constrained atoms. A *constrained atom* is a tuple of the form $(a|\mathbb{C})$, where a is an atom and $\mathbb{C}$ a conjunction of constraints. In the following, we will confuse a branch $(a_1, \ldots, a_n|\mathbb{C})$ with the set of constrained atoms $\{(a_1|\mathbb{C}) \ldots, (a_n|\mathbb{C})\}$. The power set of the constrained atoms form a complete lattice called the $\mathcal{T}$*-base* and denoted by $\mathcal{B}_\mathcal{T}$. A $\mathcal{T}$*-interpretation* is a subset of the $\mathcal{T}$-base.

Definition 17 (Inductive Semantics w.r.t. the Constraint Theory). The *immediate (backward) consequence operator* with respect to a program $\mathcal{P}$ and the theory $\mathcal{T}$ is the function $\langle\mathcal{P}\rangle^\mathcal{T} : 2^{\mathcal{B}_\mathcal{T}} \to 2^{\mathcal{B}_\mathcal{T}}$, defined as:

$$\langle\mathcal{P}\rangle^\mathcal{T}(X) \overset{\text{def}}{=} \{ (a|\mathbb{C}) \mid \left(\mathbb{H} \Longleftrightarrow \mathbb{G}| \left(\mathbb{B}_i|\mathbb{D}_i\right)_{i\in I}\right) \text{ is a rule of } \mathcal{P} \text{ renamed apart and } j \text{ is an index in } I \text{ s.t. } (\mathbb{B}_j|\mathbb{E}) \subseteq X\text{, and } \mathcal{T} \models \mathbb{C} \to (a \in \mathbb{H}) \wedge \mathbb{G} \wedge \mathbb{D}_j \wedge \mathbb{E} \}$$

We defined the *inductive semantics of a program* $\mathcal{P}$ *with respect to* $\mathcal{T}$ as the set inductively defined by the immediate consequence operator associated with $\mathcal{P}$ and $\mathcal{T}$, that is:

$$\mathrm{I}_\mathcal{P}^\mathcal{T} \overset{\text{def}}{=} \bigcap \left\{ X \mid \langle\mathcal{P}\rangle^\mathcal{T}(X) \subseteq X \right\}$$

The inductive semantics provide accurate denotational semantics for the minimal operational semantics. Indeed, the inductive semantic precisely describe the answers of a program with respect to the minimal semantics.

THEOREM 18 (Full Abstraction of Answers).

$$(\mathbb{A}|\mathbb{C}) \subseteq \mathrm{I}_\mathcal{P}^\mathcal{T} \textit{ iff } \langle(\mathbb{A}|\top)\rangle_{\bar{x}} \overset{\mathcal{P}}{\nearrow}{}^{*} \langle(\emptyset|\mathbb{C})\,;\Gamma\rangle_{\bar{x}}$$

3.2.3 Relating Inductive Semantics

There is complete adequacy between the two kinds of inductive semantics. For a given interpretation $\mathcal{S}$ for the constraint theory, the set of $\mathcal{S}$*-instances* associated to a $\mathcal{T}$-interpretation X is defined as:

$$[\![X]\!]_\mathcal{S} = \{a\rho \in \mathcal{B}_\mathcal{S} \mid (a|\mathbb{C}) \in X \ \& \ \mathcal{S} \models \mathbb{C}\rho\}$$

PROPOSITION 19. *For any interpretation* $\mathcal{S}$ *of* $\mathcal{T}$ *and any program* $\mathcal{P}$*,* $[\![\mathrm{I}_\mathcal{P}^\mathcal{T}]\!]_\mathcal{S} = \mathrm{I}_\mathcal{P}^\mathcal{S}$

The results of the present section would allow us to establish the strong completeness of minimal semantics with respect to the backward reading of a program. However, we delay the statement of a more general version of those results to Section 5.

4. Forward Models

This section will relate the models of the forward forward models of a program (i.e., the logical models its forward reading) to the forward chaining processing of the program. For this purpose, we first introduce operational semantics dual to the minimal ones. These semantics, which we call *maximal semantics*, are defined such that all the rules are processed as forward rules.

4.1 Maximal Operational Semantics

The so-called *maximal semantics* prevent rules from consume their heads. Furthermore, they never create more branches than the body of a rule imposes. We argue that this captures the pure forward chaining transitions of the general semantics.

Definition 20 (Maximal Semantics). The *maximal (operational) semantics* of a program $\mathcal{P}$ are given by the least relation $\overset{\mathcal{P}}{\rightharpoondown}$ on $\Sigma_\equiv$ satisfying the following rule:

$$\frac{\left(r \ @ \ \mathbb{H} \Longleftrightarrow \mathbb{G} \mid \left(\mathbb{B}_i|\mathbb{C}_i\right)_{i\in I}\right) \in \mathcal{P}\rho}{\left[\langle(\mathbb{H}, \mathbb{A}|\mathbb{G}, \mathbb{D}); \Gamma\rangle_{\bar{x}}\right] \overset{\mathcal{P}}{\rightharpoondown} \left[\left\langle\left(\mathbb{H}, \mathbb{B}_i, \mathbb{A}|\mathbb{G}, \mathbb{C}_i, \mathbb{D}\right)_{i\in I}; \Gamma\right\rangle_{\bar{x}}\right]}$$

where ρ renames the local variables of $\mathcal{P}$ with fresh variables. $\overset{\mathcal{P}}{\rightharpoondown}{}^{*}$ will denote the reflexive transitive closure of $\overset{\mathcal{P}}{\rightharpoondown}$. For all states S and S' we may write $S \overset{\mathcal{P}}{\rightharpoondown} S'$ instead of $[S] \overset{\mathcal{P}}{\rightharpoondown} [S']$.

The following example shows how the maximal semantics apply to our running example.

Example 21. The following step is obtained by applying the maximal semantics on the rule *antisymmetry* of the program $\mathcal{P}_3$ given at Example 3. It may be observed that such a step corresponds to the "forward mode" described at Example 8.

$$\begin{aligned} &\langle(x \leq y, x \leq y)\rangle_{xy} \\ \overset{\mathcal{P}_3}{\rightharpoondown} &\langle(x \leq y, x \leq y, x = y)\rangle_{xy} \equiv \langle(x \leq x, x = y)\rangle_{xy} \end{aligned}$$

It is also worth noting that the maximal semantics is idempotent when applied on the backward rule. For instance, consider the following derivation obtained by applying the rule *leq_s* of $\mathcal{P}_3$:

$$\begin{aligned} &\langle(\mathrm{s}(x) \leq \mathrm{s}(y))\rangle_{xy} \\ \overset{\mathcal{P}_3}{\rightharpoondown} &\langle(\mathrm{s}(x) \leq \mathrm{s}(y)); (\mathrm{s}(x) \leq \mathrm{s}(y), x \leq y)\rangle_{xy} \equiv \langle(\mathrm{s}(x) \leq \mathrm{s}(y))\rangle_{xy} \end{aligned}$$

The following proposition states that the maximal semantics overapproximate the general semantics.

PROPOSITION 22. *For any program* $\mathcal{P}$ *and all states* S*,* S'*, and* R *the following propositions hold:* (Soundness) *If* $S \overset{\mathcal{P}}{\rightharpoondown} S'$ *then* $S \overset{\mathcal{P}}{\mapsto} S'$. (Completeness) *If* $S \overset{\mathcal{P}}{\mapsto} S'$ *and* $R \sqsupseteq S$ *then there exists* R' *such that* $R \overset{\mathcal{P}}{\rightharpoondown}{}^{*} R'$ *and* $R' \sqsupseteq S'$.

4.2 Coinductive Semantics

To relate the forward model to the maximal operational semantics, we introduce fixed point semantics dual to those presented in Section 3.2. Unlike the inductive semantics, which are directly borrowed from CLP theory, the so-called coinductive semantics have been developed for the context of this paper. However we will see those semantics have close relationship with the greatest fixed points of the immediate consequence operator in CLP theory.

4.2.1 Coinductive Semantics w.r.t. Constraint Models

In Section 3.2.1, we have seen that, for a fixed interpretation $\mathcal{S}$ of the constraint theory, there exists a unique greatest and a unique least $\mathcal{S}$-model for the backward reading of a program. On one hand, the greatest $\mathcal{S}$-interpretation is clearly a model for backward reading. One the other hand, we were able to characterize the least model as the fixed point of some monotonic operator over $\mathcal{S}$-interpretations.

Here, the situation is dual. Indeed, the least $\mathcal{S}$-interpretation (i.e., the one that maps no atom to true) obviously models the forward reading of every program. However such a reading may have no unique greatest models. For instance, $\{a\}$ and $\{b\}$ are the two greatest incomparable models for the multi-headed rule $(a, b \Longrightarrow \bot)$. It appears that the notion of an $\mathcal{S}$-interpretation is not convenient for characterizing the possibly infinite number of such models. To circumvent this problem, we introduce a new construct that will be able to capture a possibly infinite number of models as a single object.

Let $\mathcal{S}$ be an interpretation of the constraint language. The $\mathcal{S}$*-cobase*, denoted by $\bar{\mathcal{B}}^\mathcal{S}$, is the set of finite $\mathcal{S}$-interpretations (i.e., the set of finite sets of atoms valued in $\mathcal{S}$). An $\mathcal{S}$*-cointerpretation* is a subset of $\bar{\mathcal{B}}^\mathcal{S}$. We say that an $\mathcal{S}$-cointerpretation $\mathcal{Y}$ is an $\mathcal{S}$*-comodel* for a formula ϕ, if for any $X \in \mathcal{Y}$ there exists an $\mathcal{S}$-model for ϕ that contains X. Thus, an $\mathcal{S}$*-comodel* for a formula ϕ can be understood as a set of finite subsets of models of ϕ. A *forward*

$\mathcal{S}$-comodel for a program $\mathcal{P}$ is an $\mathcal{S}$*-comodel* for $\overrightarrow{\mathcal{P}}$. Obviously the empty $\mathcal{S}$*-cointerpretation* is the least forward $\mathcal{S}$-comodel of every program. Furthermore, since $\mathcal{S}$-comodels are clearly stable by union, we know that any program has a greatest forward $\mathcal{S}$-comodel.

Before introducing the so-called immediate cause operator as the dual of the immediate consequence operator, we introduce a notion of breadth-first coverage inspired by previous works of Bezem and Coquand on coherent logic [7].

Definition 23 ($\mathcal{S}$-Coverage). Let $\mathcal{S}$ be a model for the constraint theory, and assume $\mathcal{P}$ is of the form:

$$\{(\mathbb{H}_1 \Longleftrightarrow \mathbb{G}_1 \mid \Gamma_1)\,,\,\ldots\,,\,(\mathbb{H}_m \Longleftrightarrow \mathbb{G}_m \mid \Gamma_m)\}$$

and X is a finite $\mathcal{S}$-interpretation. Now, consider all closed instances $\mathbb{H}_i\rho$ of the rule heads of $\mathcal{P}$ which are in X and satisfy $\mathcal{S} \vDash \mathbb{G}_i\rho$. There exist at most finitely many such instances and we may enumerate them all by $\mathbb{H}_{i_1}\rho_1, \ldots, \mathbb{H}_{i_n}\rho_n$. We will say that a finite $\mathcal{S}$-interpretation X' is a *(breadth-first forward)* $\mathcal{S}$*-coverage* for X with respect to $\mathcal{P}$ if:

$$X' = X \cup \mathbb{B}_0[\vec{y}\backslash\bar{d}_0]\rho_o \cup \cdots \cup \mathbb{B}_n[\vec{y}\backslash\vec{d_n}]\rho_n$$

for some branch $(\mathbb{B}_j|\mathbb{D}_j) \in \Gamma_j$ and some $\mathcal{S}$-values $\bar{d}_j$, such that $\mathcal{S} \vDash (\mathbb{G}_{i_j} \wedge \mathbb{D}_j)[\vec{y}\backslash\vec{d_j}]\rho_j$ (for $j \in 1, \ldots, m$). In such a case, we write $X \vartriangleright^{\mathcal{S}}_{\mathcal{P}} X'$.

Intuitively we can understand the notion of $\mathcal{S}$-coverage with respect to a program $\mathcal{P}$ as a kind of coinductive evidence for satisfiability with respect to $\overrightarrow{\mathcal{P}}$. Indeed, on one hand, it is easy to verify that a set of valued atoms which has no coverage is not included in any model of $\overrightarrow{\mathcal{P}}$. Conversely, as the following proposition shows, a set X_0 is satisfiable if it has a coverage X_1, which in turn has a coverage X_2, and so on infinitely.

PROPOSITION 24. *Let $\mathcal{S}$ be a model of the constraint theory and $\mathcal{P}$ a program. Let $\{X_i\}_{i\in\mathbb{N}}$ be a family of finite $\mathcal{S}$-interpretations, such that $X_i \vartriangleright^{\mathcal{S}}_{\mathcal{P}} X_{i+1}$ (for any $i \in \mathbb{N}$). The union of the X_i is an $\mathcal{S}$-model for $\overrightarrow{\mathcal{P}}$.*

Based on this notion of coverage, we define an algebraic semantics for the forward reading of a program as the greatest fixed point over an operator dual to the immediate consequence operator. Basically, the fixed point computation iteratively removes from the $\mathcal{S}$-cobase all those sets with no coverage.

Definition 25 (Coinductive Semantics w.r.t. Constraint Models). Let $\mathcal{S}$ be a model for the constraint theory. For a given program $\mathcal{P}$, the *immediate (forward) cause operator* with respect to a program $\mathcal{P}$ and $\mathcal{S}$, $[\mathcal{P}]^{\mathcal{S}} : 2^{\bar{\mathcal{B}}^{\mathcal{S}}} \rightarrow 2^{\bar{\mathcal{B}}^{\mathcal{S}}}$ is defined as:

$$[\mathcal{P}]^{\mathcal{S}}(\mathcal{Y}) \overset{\text{def}}{=} \left\{X \in \bar{\mathcal{B}}^{\mathcal{S}} \mid \text{ there is } X' \in \mathcal{Y} \text{ such that } X \vartriangleright^{\mathcal{S}}_{\mathcal{P}} X'\right\}$$

The *coinductive semantics* of a program $\mathcal{P}$ with respect to $\mathcal{S}$ is defined as the set coinductively defined by the immediate cause operator associated with $\mathcal{P}$ and $\mathcal{S}$, i.e., the union of the postfixed points of $[\mathcal{P}]^{\mathcal{S}}$:

$$\mathrm{C}^{\mathcal{S}}_{\mathcal{P}} \overset{\text{def}}{=} \bigcup \left\{\mathcal{Y} \in 2^{\bar{\mathcal{B}}^{\mathcal{S}}} \mid \mathcal{Y} \subseteq [\mathcal{P}]^{\mathcal{S}}(\mathcal{Y})\right\}$$

The coinductive semantics capture all backward models of a program.

THEOREM 26. (GREATEST BACKWARD $\mathcal{S}$-COMODEL) *Let $\mathcal{S}$ be a model of the constraint theory $\mathcal{T}$, and $\mathcal{P}$ be a program. $\mathrm{C}^{\mathcal{S}}_{\mathcal{P}}$ is the greatest $\mathcal{S}$-comodel for $\overrightarrow{\mathcal{P}}$.*

4.2.2 Coinductive Semantics w.r.t. the Constraint Theory

As with the immediate consequence operator, we will extend the immediate cause operator for a given particular model to the constraint theory itself. Since the immediate cause operator is based on the notion of coverage on a valued set of atoms, we need first to extend this notion to a constrained set of atoms. The power set of the consistent branches forms a complete lattice, which we call the $\mathcal{T}$*-cobase* and denote by $\bar{\mathcal{B}}^{\mathcal{T}}$. A $\mathcal{T}$*-cointerpretation* is a subset of the $\mathcal{T}$-cobase.

Definition 27 ($\mathcal{T}$-Coverage). Assume given a program $\mathcal{P}$ and a branch $(\mathbb{A}|\mathbb{C})$ that do not share any variables. Now consider all pairs $(\mathbb{A}', (\mathbb{H} \Longleftrightarrow \mathbb{G} \mid \Gamma))$ such that $\mathbb{A}'$ is a subset of $\mathbb{A}$ and $(\mathbb{H} \Longleftrightarrow \mathbb{G} \mid \Gamma)$ is a rule of $\mathcal{P}$ that satisfies the implication $\mathcal{T} \vDash \mathbb{C} \rightarrow \exists_{\text{-}\mathbb{A}}(\mathbb{A}' = \mathbb{H} \wedge \mathbb{G})$. Then there exist at most finitely many such pairs, which we may enumerate thus:

$$(\mathbb{A}_1, (\mathbb{H}_1 \Longleftrightarrow \mathbb{G}_1 \mid \Gamma_1))\,, \ldots, (\mathbb{A}_n, (\mathbb{H}_n \Longleftrightarrow \mathbb{G}_n \mid \Gamma_n))$$

where the variables of each rule have been previously renamed apart. We will say that the branch $(\mathbb{E}|\mathbb{F})$, which equals

$$(\mathbb{A}, \mathbb{B}_1, \ldots, \mathbb{B}_n|\mathbb{C}, \mathbb{A}_1 = \mathbb{H}_1, \mathbb{D}_1, \mathbb{G}_1, \ldots, \mathbb{A}_n = \mathbb{H}_n, \mathbb{D}_n, \mathbb{G}_n)$$

is a *(breadth-first forward)* $\mathcal{T}$*-coverage* for $(\mathbb{A}|\mathbb{C})$ if for any $j \in 1, \ldots, n$, $(\mathbb{B}_j|\mathbb{D}_j) \in \Gamma_j$. In this case, we write $(\mathbb{A}|\mathbb{B}) \vartriangleright^{\mathcal{T}}_{\mathcal{P}} (\mathbb{E}|\mathbb{F})$.

We understood $\mathcal{T}$-coverage as an evidence for nonfailure. Indeed, in a coverage relationship, the right-hand branch is always more constrained than the left-hand branch. Therefore, an inconsistent branch has no consistent coverage. More interestingly, the following proposition establishes that a branch $(\mathbb{A}|\mathbb{B})$ covered by a branch $(\mathbb{E}|\mathbb{F})$ cannot produce with one step branches more constrained than $(\mathbb{E}|\mathbb{F})$. Consequently, a branch covered by a consistent branch cannot be made inconsistent in at most one derivation step. As previously, this notion of evidence is coinductive, in the sense that nonfailure is guaranteed by an infinite chain of consistent evidence.

PROPOSITION 28. *Let $\langle(\mathbb{A}|\mathbb{C})\rangle_{\bar{x}} \xrightarrow{\mathcal{P}} \langle(\mathbb{B}_i|\mathbb{D}_i)_{i\in I}\rangle_{\bar{x}}$ be a valid derivation and $(\mathbb{E}|\mathbb{F})$ be a breadth-first forward $\mathcal{T}$-coverage of $(\mathbb{A}|\mathbb{C})$. If for any $i \in I$ $\mathcal{T} \vDash \mathbb{D}_i \rightarrow \exists_{\text{-}\bar{x},\mathbb{B}_i}((\mathbb{E} \subseteq \mathbb{B}_i) \wedge \mathbb{F})$, then there exists $j \in I$ such that $\mathcal{T} \vDash \mathbb{F} \rightarrow \exists_{\text{-}\bar{x},\mathbb{B}_j}((\mathbb{B}_j \subseteq \mathbb{E}) \wedge \mathbb{D}_ij)$.*

We extend the coinductive semantics to the theory by substituting the coverage relation. As previously, the fixed point computation removes those branches that have no consistent evidence. Here, it is worth remembering the cobase $\bar{\mathcal{B}}^{\mathcal{T}}$ is the set of consistent branches.

Definition 29 (Coinductive Semantics w.r.t. the Constraint Theory). For a given program $\mathcal{P}$, the *immediate (forward) cause operator* with respect to a program $\mathcal{P}$ and the theory $\mathcal{T}$, $[\mathcal{P}]^{\mathcal{T}} : 2^{\bar{\mathcal{B}}^{\mathcal{T}}} \rightarrow 2^{\bar{\mathcal{B}}^{\mathcal{T}}}$ is defined as:

$$[\mathcal{P}]^{\mathcal{T}}(\mathcal{Y}) \overset{\text{def}}{=} \{(\mathbb{A}|\mathbb{B}) \in \bar{\mathcal{B}}^{\mathcal{T}} \mid \text{ there is } (\mathbb{E}|\mathbb{F}) \in \mathcal{Y} \text{ such that } (\mathbb{A}|\mathbb{B}) \vartriangleright^{\mathcal{T}}_{\mathcal{P}} (\mathbb{E}|\mathbb{F})\}$$

The *coinductive semantics* of a program $\mathcal{P}$ with respect to the theory $\mathcal{T}$ is defined as the set coinductively defined by the immediate cause operator associated with $\mathcal{P}$ and $\mathcal{T}$:

$$\mathrm{C}^{\mathcal{T}}_{\mathcal{P}} \overset{\text{def}}{=} \bigcup \left\{\mathcal{Y} \in 2^{\bar{\mathcal{B}}^{\mathcal{T}}} \mid \mathcal{Y} \subseteq [\mathcal{P}]^{\mathcal{T}}(\mathcal{Y})\right\}$$

Similar to the case of inductive semantics, the coinductive semantics provide an accurate denotation for the maximal operational semantics. In fact, the coinductive semantics precisely describe nonfailures.

THEOREM 30. (FULL ABSTRACTION OF NONFAILURES)

$$(\mathbb{A}|\mathbb{C}) \in \mathrm{C}^{\mathcal{T}}_{\mathcal{P}} \textit{ iff } \langle(\mathbb{A}|\mathbb{C})\rangle_{\bar{x}} \not\xrightarrow{\mathcal{P}}{}^{*} \langle(\emptyset|\bot)\rangle$$

It is worth noting that the coinductive semantics for CHR programs have a strong relationship with the greatest fixed point of the immediate consequence operator, which abstracts non-failures of CLP programs [22]. However, such a characterization is simpler in the particular context of CLP, since it can be done within the standard $\mathcal{T}$-base, while the equivalent characterization in CHRmust be done within the $\mathcal{T}$-cobase. This relative simplicity comes from the fact that CLP rules are monoheaded.

4.2.3 Relating Coinductive Semantics

In the general case, we can show that the semantics defined with respect to the theory abstract the ones defined with respect to a model of the theory. For this purpose, we define an abstraction that maps any $\mathcal{S}$-cointerpretation to a $\mathcal{T}$-cointerpretation. Formally, for a given interpretation $\mathcal{S}$ of the constraint language $\mathcal{L}_c$, we define the *abstraction* of an S-cointerpretation $\mathcal{Y}$ as:

$$[\![\mathcal{Y}]\!]_{\mathcal{S}^{-1}} \stackrel{\text{def}}{=} \left\{(\mathbb{A}|\mathbb{C}) \in \vec{\mathcal{B}}^{\mathcal{T}} \mid \mathbb{A}\rho \in \mathcal{Y} \;\&\; \mathcal{S} \models \mathbb{C}\rho\right\}$$

PROPOSITION 31. *For any model $\mathcal{S}$ of $\mathcal{T}$, $[\![\mathrm{C}^{\mathcal{S}}_{\mathcal{P}}]\!]_{\mathcal{S}^{-1}} \subseteq \mathrm{C}^{\mathcal{T}}_{\mathcal{P}}$.*

However, unlike in the case of inductive semantics, it is not possible in the general case to establish full adequacy between the two kinds of conductive semantics. Indeed, as shown by the following example, the coinductive semantics with respect to $\mathcal{T}$ may contain nonground branches to which there correspond no instances in the coinductive semantics defined with respect to any model of $\mathcal{T}$. In other words, there may exist nonground branches in $\mathrm{C}^{\mathcal{T}}_{\mathcal{P}}$ that are satisfiable with respect to no model of $\overrightarrow{\mathcal{P}}$.

Example 32. Assume the constraint theory contains only the axiom $\forall x(\mathrm{c}(x) \vee \mathrm{d}(x))$ and consider the program $\mathcal{P}_{32}$ consisting of the following rules:

$$\mathrm{p}(x) \Longrightarrow \mathrm{c}(x) \mid \bot \qquad \mathrm{p}(x) \Longrightarrow \mathrm{d}(x) \mid \bot$$

While $\mathrm{C}^{\mathcal{T}}_{\mathcal{P}_{32}}$ contains branches such as $\langle(p(x,y))\rangle$, for any $\mathcal{S}$ of $\mathcal{T}$, $\mathrm{C}^{\mathcal{S}}_{\mathcal{P}_{32}}$ is empty.

To circumvent this problem, we restrict the form of the the constraint theory. We will say that a constraint theory is *simple* if its nonlogical axioms are of the form:

$$\mathbb{C} \rightarrow \exists \bar{x} \mathbb{D}$$

where $\mathbb{C}$ and $\mathbb{D}$ are conjunctions of built-in constraints. Such an assumption is standard in concurrent constraint programming [12, 33]. It is worth noting that a number of constraint theories, such as Clark's equality theory, are simple theories.

PROPOSITION 33. *If $\mathcal{T}$ is a simple and consistent constraint theory, then there exists a model $\mathcal{S}$ of $\mathcal{T}$ such that $[\![\mathrm{C}^{\mathcal{S}}_{\mathcal{P}}]\!]_{\mathcal{S}^{-1}} \supseteq \mathrm{C}^{\mathcal{T}}_{\mathcal{P}}$.*

In general, the models for which the proposition holds are not the intended ones. For instance, in the case of Clark's equality theory, the domain of such models is typically the complete Herbrand Universe [24] (i.e., the set of finite and infinite terms). Nevertheless, we will see later that this result is sufficient to establish the completeness of our language for failures and accessible constraints with respect to the constraint theory.

The results of this section would also allow us to establish that the maximal semantics is complete for failures and accessible constraints. Nonetheless, we delay the statement of this result until the more general context of the next section. (Refer to Theorem 40.) The completeness result extends the previous work of Bezem and Coquand [7] by showing that the geometric logic extended with a simple constraint theory (in particular native equational logic) is complete with respect to Tarskian truth.

5. Combining Chainings

In this section, we aim to combine the results of the previous sections to obtain strict completeness results for the answers and accessible constraints with respect to the whole logical reading of a given program, including both backward and forward readings. The crucial property we will use here is the notion of confluence. Confluence is a fundamental property that guarantees that the computations are not dependent on rule application order.

5.1 Confluence

Before getting into the substance of the completeness results, we formalize the notion of confluence and completion. Basically, this property refers to the fact that two finite computations starting from a common state can always be prolonged so as to eventually meet in a common state again. This fundamental property justifies the committed choice policy of rule application.

Definition 34 (Confluence). Formally a program $\mathcal{P}$ is *confluent* if for all states S, S_1, and S_2 such that $S \xrightarrow{\mathcal{P}}{}^{*} S_1$ and $S \xrightarrow{\mathcal{P}}{}^{*} S_2$, there exists a state S' such that $S_1 \xrightarrow{\mathcal{P}}{}^{*} S'$ and $S_2 \xrightarrow{\mathcal{P}}{}^{*} S'$.

Example 35 presents examples of confluent and non-confluent program.

Example 35. Consider our running example, $\mathcal{P}_3$. Several rules match the state $\langle x \leq 0, 0 \leq y\rangle$ and may then produce incomparable states, e.g.:

$$\begin{array}{lll} \langle(x \leq 0, 0 \leq x)\rangle_x & \xrightarrow{\textit{antisymmetry}} & \langle(x = 0)\rangle_x \\ \langle(x \leq 0, 0 \leq x)\rangle_x & \xrightarrow{\textit{leq\_0}} & \langle(x \leq 0)\,;\,(x \leq 0, 0 \leq x)\rangle_x \equiv \langle x \leq 0\rangle_x \end{array}$$

One can verify that the two resulting states cannot be rewritten to a common state: $\mathcal{P}_3$ is not confluent. Nevertheless $\mathcal{P}_3$ can be completed [1], that this to say that one can add rule to $\mathcal{P}_3$ in order to make it confluent. For instance consider the program $\mathcal{P}_{35}$ built by adding to $\mathcal{P}_3$ the following rule. It is interesting noting this rule corresponds to the Clark's completion of the backward reading of $\mathcal{P}_3$:

$$\textit{completion} \;@\; x \leq y \implies x = 0; x = \mathrm{s}(x'), y = \mathrm{s}(y'), x' \leq y'$$

In $\mathcal{P}_{35}$ the two problematic states can be joined as follows:

$$\begin{array}{rl} & \langle x \leq 0\rangle_x \\ \xrightarrow{\textit{completion}} & \big\langle(x \leq 0, x = 0)\,;\,\big(x \leq 0, x = \mathrm{s}(x'), 0 = \mathrm{s}(y'), x' \leq y'\big)\big\rangle_x \\ \equiv & \langle(0 \leq 0, x = 0)\,;\,(\bot)\rangle_x \\ \xrightarrow{\textit{leq\_0}} & \langle(x = 0)\,;\,(0 \leq 0, x = 0)\,;\,(\bot)\rangle_x \\ \equiv & \langle(x = 0)\rangle_x \end{array}$$

Because of space limitations, we will not further discuss how the confluence of programs can be demonstrated, but leave this problem open for future work.

We will, however, refer to a technical report [15], in which we establish that (perhaps surprisingly) confluence with respect to the classical multiset-based semantics implies confluence with respect to the set-based semantics presented here. This suggests that the confluence proof techniques for classical CHR—such as those based on local confluence [2], strong confluence [16], or decreasing diagrams [14]—can be extended to deal with the semantics presented here.

5.2 Completeness of Answers

The following theorem shows that the inductive semantics of a confluent program coincide with the least model of its whole logical reading. Note that this does not require $\mathcal{T}$ to be simple.

THEOREM 36. (LEAST $\mathcal{S}$-MODEL) *Let $\mathcal{S}$ be a model of the constraint theory $\mathcal{T}$. If a program $\mathcal{P}$ is confluent, then its logical reading $\overleftrightarrow{\mathcal{P}}$ has a least $\mathcal{S}$-model. This latter is equal to $\mathrm{I}^{\mathcal{S}}_{\mathcal{P}}$.*

As a corollary, we see that the logical reading of a confluent program is consistent. This idea can be traced back to the pioneering works about CHR confluence [2], but the constructive characterization of the least model for confluent programs is more recent [17]. The proof of this characterization is, however, more involved in the original settings of CHR because of the multiset nature of atoms. To the best of our knowledge, the characterization of the biggest models are new.

Combined with Theorem 18 and Proposition 19 this theorem allows us to establish the completeness of our language with respect to answers.

THEOREM 37. (COMPLETENESS OF ANSWERS) *Let $\mathcal{P}$ be a confluent program and $R \xrightarrow{\mathcal{P}}{}^* S$ be a valid derivation. If the implication $\mathcal{T}\overleftrightarrow{\mathcal{P}} \vDash \mathbb{C} \rightarrow R^\dagger$ holds, then there is a derivation of the form $S \overset{\mathcal{P}}{\rightharpoondown}{}^* \langle(\emptyset|\mathbb{D}_i)_{i\in I} \,; \Gamma\rangle_{\bar{x}}$ such that $\mathcal{T} \vDash \mathbb{C} \rightarrow \exists_{\text{-}\bar{x}} \left(\bigvee_{i\in I} \mathbb{D}_i\right)$.*

By contraposition, we obtain that if a state does not reduce to the trivial state $\langle(\emptyset|\top)\rangle$, then its logical reading is not valid with respect to the logical reading of the program. Of course, there is no effective way to determine when a state lacks consistent answers—the problem being clearly undecidable—but we can identify some particular cases. Given a program $\mathcal{P}$, we will say that a state R is *minimally saturated* with respect to $\mathcal{P}$ if for any state S, $R \overset{\mathcal{P}}{\rightharpoondown} S$ implies $R \sqsubseteq S$. Because the number of ways a rule may apply on a given state is finite, minimal saturation is decidable whenever the constraint theory is.

COROLLARY 38. (SOUNDNESS OF MINIMAL SATURATION) *Let $\mathcal{P}$ be a confluent program. If a state S derives to a minimally saturated state R, then $\mathcal{T}\overleftrightarrow{\mathcal{P}} \vDash S^\dagger$ iff $R \equiv \langle(\emptyset|\top)\rangle$.*

5.3 Completeness of Accessible Constraints

Conversely, the following theorem states that the coinductive semantics coincide with the greatest $\mathcal{S}$-comodel of the whole program.

THEOREM 39. (GREATEST $\mathcal{S}$-COMODEL) *Let $\mathcal{T}$ be a simple and consistent constraint system. If a program $\mathcal{P}$ is confluent, then there exists a model $\mathcal{S}$ of $\mathcal{T}$ such that its greatest $\mathcal{S}$-comodel is $\mathrm{C}^{\mathcal{S}}_{\mathcal{P}}$.*

By combining this theorem with Theorem 30 and Propositions 31 and 33, we obtain a strong completeness result with respect to accessible constraints.

THEOREM 40. (COMPLETENESS OF ACCESSIBLE CONSTRAINTS) *Let $\mathcal{T}$ be a simple constraint theory, $\mathcal{P}$ be a confluent program, and $R \xrightarrow{\mathcal{P}}{}^* S$ be a valid derivation. If the implication $\mathcal{T}\overleftrightarrow{\mathcal{P}} \vDash R^\dagger \rightarrow \exists\bar{x}\mathbb{C}$ holds, then there exists a derivation of the form $S \overset{\mathcal{P}}{\searrow}{}^* \langle(\mathbb{B}_i|\mathbb{D}_i)_{i\in I}\rangle_{\bar{y}}$ such that $\mathcal{T} \vDash \left(\bigvee_{i\in I} \mathbb{D}_i\right) \rightarrow \exists\bar{x}\mathbb{C}$.*

It is worth noting that this theorem generalizes the completeness of the so-called negation-as-failure for CLP programs [21, 22]. This property states that if a query is false with respect to the Clark completion of a program, then its standard CLP evaluation finitely fails. Indeed the previous theorem can be applied to Clark completed CLP programs as they are trivially confluent. (Just notice that a failure is a state that has $\bot$ as accessible constraint.)

By contraposition of Theorem 40, we see that if a state does not reduced to the inconsistent state $\langle(\emptyset|\bot)\rangle$, then is logical reading is satisfiable with respect to the program. We identify special cases of such states as duals of minimally saturated states. We will say that a state R is *maximally saturated* with respect to a program $\mathcal{P}$ if, for any valid derivation of the form $R \overset{\mathcal{P}}{\searrow} S$, $R \sqsupseteq S$ holds. As for minimal saturation, maximal one is decidable whenever the constraint theory is.

COROLLARY 41. (SOUNDNESS OF MAXIMAL SATURATION) *Let us assume the constraint theory $\mathcal{T}$ is simple, and $\mathcal{P}$ is a confluent program. If a state S reduces to a maximally saturated state R, then $\mathcal{T}\overleftrightarrow{\mathcal{P}} \vDash S^\dagger \rightarrow \bot$ iff $R \equiv \langle(\emptyset|\bot)\rangle$.*

6. Toward Implementation

As explained previously, we designed the analytical semantics to be as close as possible to the classical logic meaning of a program, without too much concern about their implementability. In this section, we do not study the details of possible implementations of the whole language in full, as this is beyond the scope of the present paper. Instead, we briefly discuss how existing CHR frameworks can benefit from the theoretical results of this paper. For this purpose, we divide the discussion into two, according to the principal objective of the inference that a user may run. Given a program $\mathcal{P}$ and a state, this main objective may be to find the solutions for the state (i.e., those constraints that make the state valid with respect to the logical reading of the program) or, conversely, to prove that the state is (un)satisfiable with respect to the program.

6.1 Solution-Oriented Implementation

In this section, we are interested in finding all the solutions of a given confluent program. For this purpose, we propose a simple source-to-source translation intended to be executed by a standard (i.e., multiset-based) CHR$^\vee$ implementation.

The translation $\mathcal{P}^\diamond$ of a program $\mathcal{P}$ is defined as follows:

step 1 First, for each user-defined symbol p, add in $\mathcal{P}^\diamond$, a simpagation rule of the form $(p(\vec{x}) \setminus p(\vec{x}) \Longleftrightarrow \top)$. Such a rule, which particularly makes sense in a muliset-based setting, launches only if two occurrences of $p(\vec{x})$ are present in the store. It then removes the older of the two occurrences. From a logical point of view, this rule is read as a tautology.

step 2 Next, the forward reading of $\mathcal{P}$ is naively translated as forward rules, and added to $\mathcal{P}^\diamond$.

step 3 Finally, the backward reading of $\mathcal{P}$ is first translated to a CLP program (following the remark at the end of Section 2.2), and then translated back to CHR$^\vee$ using Clark's completion [10] (as described by Abdennadher and Schütz [4]). It is then added to $\mathcal{P}^\diamond$. Repeated and tautological CLP clauses are eliminated during the process.

Example 42. The translation $\mathcal{P}^\diamond_3$ of the running example is as follows:

$$\begin{aligned}
\text{set} \ @ \ & x \leq y \setminus x \leq y \Longleftrightarrow \top.\\
\text{trans}' \ @ \ & x \leq y \ , \ y \leq z \Longrightarrow x \leq y.\\
\text{anti}' \ @ \ & x \leq y \ , \ y \leq x \Longrightarrow x = y.\\
\text{back} \ @ \ & x \leq y \Longleftrightarrow x = 0; x = \mathrm{s}(x'), y = \mathrm{s}(y'), x' \leq y';\\
& \qquad\qquad\qquad\qquad x = 0, y = 0; x = y.
\end{aligned}$$

The rules trans$'$ and anti$'$ correspond to the forward reading of the transitivity and antisymmetry rules, respectively. On the other side, the back rule corresponds to Clark's completion of the backward reading.

We argue that, if evaluated by the standard CHR$^\vee$ system, this translation is sound and complete for the solutions of the original program. Formally, a constraint c is a solution of a state w.r.t. a program $\mathcal{P}$ iff and only if there is a finite set of answers $d_1, \dots, d_n$ for $\mathcal{P}^\diamond$ when processed by the standard multiset semantics, such that $\mathcal{T} \vDash d_1 \vee \dots \vee d_n \rightarrow c$. The proof can be sketched as follows:

By Theorem 37, we know that (1) the minimal semantics can find any solution of a confluent program $\mathcal{P}$ without loss of generality. On the other hand, as explained in Section 3, we also know that (2) the minimal semantics correspond exactly to the CLP processing of the backward reading of $\mathcal{P}$. Furthermore, Abdennadher and Schütz [4] showed that (3) the classical operational semantics of a CLP program are equivalent to the CHR$^\vee$ multiset semantics operating on its Clark's completion [4]. By combining (1), (2), and (3), we infer that a constraint is a solution of $\mathcal{P}$ iff it is an answer obtained by processing only the rules resulting from step 3 of the translation with multiset semantics.

We still need to show that the whole program $\mathcal{P}^\diamond$ has no more solutions than $\mathcal{P}$. Since a CLP program and its completion have the same least models, so do $\mathcal{P}$ and $\mathcal{P}^\diamond$. Hence, for any state, $\mathcal{P} \vDash c \rightarrow S$ holds iff $\mathcal{P}^\diamond \vDash c \rightarrow S$. We conclude using soundness of CHR.

We now conclude the section with some remarks. First, note that, while the set-based evaluation $\mathcal{P}$ never terminates, the multiset $\mathcal{P}^\diamond$ evaluation may terminate. In particular, if the latter fails, we know the former has no solutions. Secondly, we remark that the multiset interpretation of $\mathcal{P}$ may have no answers, whereas the same interpretation of $\mathcal{P}^\diamond$ is complete for answers.

6.2 Satisfiability-Oriented Implementation

In this section, we are interested in proofs of the (un)satisfiability of a state with respect to a program. In this context, we discuss how results from Section 5.2 can be used in the context of the set-based semantics of Sarna-Starosta and C.R. Ramakrishnan [34], and those of Duck [11]. These semantics, which we will call naive set semantics, have been developed for the purpose of implementation, but to the best of our knowledge their mathematical properties have not been formally studied. They differ from the semantics we have proposed, in the sense that they force the consummation of all the atoms of the head of a rule in a state. In order to better understand the different classes of semantics, we propose to simplify the formalizations of these classes.

In order to better understand the different classes of semantics we propose formalizations of each of them. For the sake of simplicity, we limit our discussion to the propositional case. Hence, the propositional version of the classical multiset semantics $\rightarrowtail_{\mathrm{m}}$, the analytical semantics $\rightarrowtail_\alpha$ presented at Section 2, and the naive set semantics $\rightarrowtail_{\mathrm{s}}$ can be defined by the following rules:

$$\frac{\mathbb{H} \Longleftrightarrow \mathbb{B} \in \mathcal{P}}{\langle(\mathbb{A} \uplus \mathbb{H})\rangle \rightarrowtail_{\mathrm{m}} \langle(\mathbb{A} \uplus \mathbb{B})\rangle}$$

$$\frac{\mathbb{H} \Longleftrightarrow \mathbb{B} \in \mathcal{P}}{\langle(\mathbb{A} \cup \mathbb{H})\rangle \rightarrowtail_\alpha \langle((\mathbb{A} \cup \mathbb{B})\rangle}$$

$$\frac{\mathbb{H} \subseteq \mathbb{A} \quad \mathbb{H} \Longleftrightarrow \mathbb{B} \in \mathcal{P} \quad (\mathbb{A} \setminus \mathbb{H}) \cup \mathbb{B} \neq \mathbb{A}}{\langle(\mathbb{A})\rangle \rightarrowtail_{\mathrm{s}} \langle((\mathbb{A} \setminus \mathbb{H}) \cup \mathbb{B})\rangle}$$

Based on these formalization, we can make several remarks. First, naive set semantics break the *monotonicity* of transitions—one of the most important properties of CHR (See Chapter 4 in Früwirth's book [13].). Indeed, we have

$$\begin{array}{lll} \langle(\mathbb{A})\rangle \rightarrowtail_{\mathrm{m}} \langle(\mathbb{A}')\rangle & \text{implies} & \langle(\mathbb{A} \uplus \mathbb{B})\rangle \rightarrowtail_{\mathrm{m}} \langle(\mathbb{A}' \uplus \mathbb{B})\rangle, \text{ and} \\ \langle(\mathbb{A})\rangle \rightarrowtail_\alpha \langle(\mathbb{A}')\rangle & \text{implies} & \langle(\mathbb{A} \cup \mathbb{B})\rangle \rightarrowtail_\alpha \langle(\mathbb{A}' \cup \mathbb{B})\rangle \end{array}$$

but

$$\langle(\mathbb{A})\rangle \rightarrowtail_{\mathrm{s}} \langle(\mathbb{A}')\rangle \quad \text{does not imply} \quad \langle(\mathbb{A} \cup \mathbb{B})\rangle \rightarrowtail_{\mathrm{s}} \langle(\mathbb{A}' \cup \mathbb{B})\rangle$$

(take $\mathbb{B} = \mathbb{A}$ as a counterexample). Note that such behavior should in principle complicate any confluence proofs at the naive set level, since all the CHR confluence proof techniques we are aware of heavily rely on transition monotonicity.

Secondly, the naive set semantics, just like the multiset semantics, are trivially incomplete with respect to both success and failure. For instance, consider the program

$$\mathrm{p} \Longleftrightarrow \mathrm{q}. \qquad \mathrm{p} \Longleftrightarrow \mathrm{q}. \qquad \mathrm{p}, \mathrm{q} \Longleftrightarrow \mathrm{c}.$$

where c stands for $\top$ or $\bot$. Clearly, we have $\mathcal{T}\overleftrightarrow{\mathcal{P}} \vDash p \leftrightarrow c$ but $\langle(p)\rangle \not\rightarrowtail_{\mathrm{m}} \langle(c)\rangle$ and $\langle(p)\rangle \not\rightarrowtail_{\mathrm{s}} \langle(c)\rangle$.

The naive semantics have, however, some interesting properties when compared with the analytical semantics: The naive set semantics are *analytically sound* (i.e., $R \rightarrowtail_{\mathrm{s}} S$ implies $R \rightarrowtail_\alpha S$), and if $\mathcal{P}$ is a set of forward rules, they are *analytically complete* (i.e., $R \rightarrowtail_\alpha S$ implies $R \rightarrowtail_{\mathrm{s}}^{\equiv} S$).

We can obtain a naive implementation complete for unsatisfiability with respect to an (analytically) confluent program by simply running the forward reading according to naive set-based semantics. The proof can be sketched as follows: Theorem 39 ensures that if a state is unsatisfiable, then there is a maximal derivation that leads this state to the inconsistent state $\top$. Since the maximal semantics ignore backward rules, such a derivation can be obtained by considering only the forward reading. We conclude with the analytical completeness of the naive set semantics for forward rules. This shows that an unsatisfiability-complete implementation of confluent program can be reached. This solution is, however, not satisfactory, as it does not allow any form of combination of the two kinds of chainings. We leave the question of how to incorporate some form of backward chaining, while preserving completeness, for future investigations.

Before concluding, we can underline that Corollary 41 ensures that naive set semantics is sound for stability. In other words, assuming that the program is consistent, if a state derives to a terminal and consistent state (different from $\langle(\emptyset|\bot)\rangle$), then the state is satisfiable with respect to the program. This result rests on the analytical completeness for forward rules, which implies that states terminal with respect to the naive set semantics are maximally saturated with respect to the analytical semantics. In the same context, the analytical semantics can be also helpful in proving the consistency of program processing, since analytical confluence (which, as explained previously, is simpler to establish than confluence with respect to other semantics) ensures the consistency of the program (by Theorem 36).

7. Related Work

Although not explicit, the procedural interpretation of CLP clauses as presented by Jaffar and Lassez [21] already combines backward and forward reasoning. On one hand, the definite clauses handle the general control of the program, while the constraint theory tackles native data-type operation manipulations. The processing of the clauses is of course based on a backward chaining, but the constraint solver, seen as a black box, can be implemented using any kind of reasoning, and in particular can employ forward chaining that well accommodates solver incrementality—a key property for efficient implementations of CLP [28]. Examples of forward reasoning within constraint solvers are the first order term unification [31][2], and the local propagation mechanism of finite domain solvers [18].

The present work gives a partial account of this hierarchical phenomenon in a one-level formalism. Indeed, one can understand the CLP clauses as a set of backward rules and the constraint solver as an arbitrary confluent program, with both sets of rules using disjoint alphabets of predicate symbols for their heads. Thanks to the modularity of confluence [13, 14] (i.e., the union of nonoverlapping confluent programs is confluent), the results presented here ensure the soundness and completeness of a general operational semantics for CLP, where the implementation of the solver would be made explicit.

ALPS is a class of flat committed-choice logic languages proposed by Maher as a concurrent framework suitable for algorithmic programming [26]. Roughly, ALPS programs are sets of monoheaded backward rules implicitly completed by Clark's completion. Operationally, a rule applies either when its guard is entailed by the current store (in a forward-like manner) or when it is the only one satisfiable among all the rules with same head (in a backward-like manner). For deterministic programs (i.e., programs whose guards are mutually exclusive) such derivation steps correspond to applications of the analytical semantics that do not create branches. Abdennadher et al. have already identified the strong relationship between deterministic ALPS programs and confluent CHR programs [2], but only the general framework presented in this paper captures both types of rule application. Indeed, in standard CHR, rules apply if and only if their guards are implied by the built-in store. In fact, the soundness and completeness theorems presented here thoroughly generalize all similar results proved by Maher for ALPS.

In the context of (constraint) deductive databases, Magic Sets and their extensions [5, 30] introduce some elements of backward chaining into the purely forward processing of the deductive database. In those approaches, the program is first processed to collect the so-called predicate adornments—i.e., the calling patterns involved in the backward processing of a query. Then, based on these adornments, the program is transformed, before being evaluated by the standard fixed-point forward processing. As a result, this approach does not generate more facts than the standard backward processing would , but better preserves termination properties. On the downside, it is typically inefficient when one is looking for only one solution. Compare this to the approach proposed in this paper, where the combination of backward and forward chaining is dynamic. In contrast, the combination in the Magic Sets approach is completely static. The Magic Sets approach also differs from ours in the sense that the clauses within a program are all treated in the same way, while in our framework, logical implications are processed differently, depending on the way they are written.

In the field of linear logic programming, the LolliMon [25] system extends the natural backward chaining operational semantics of the Lolli [19] system with forward chaining inside a monad. In this approach, the monad plays a similar role to the commutation property we assume here, to prevent logical interference between to the two forms of computation. The focusing strategy for the inverse method of Chaudhuri et al. [9] is an other way for combining the different forms of chainings within the intuitionistic linear logic. This approach differs from ours in the sense that the discrimination of behaviour is based on atoms (i.e., some atoms are processed backward, others forward), while in our framework, it is based on implications (i.e., some implications are handled backward, others forward).

[2] While the original presentation of the unification by Robinson is purely algorithmic, Huet later showed it can be viewed as a forward reasoning [20].

In the context of theorem proving, the Prolog implementation of the model generator SATCHMO [27] allows the combination of the two forms of reasoning into a logic more restricted than ours. By default, SATCHMO rules are processed on a forward principal, but on the other hand, rules in the scope of Horn Logic can be specified as ordinary Prolog. The basic algorithm is to incrementally search for forward rules that have a head satisfied by the current state of the model being constructed, and to incrementally add to the model the corresponding bodies. Specifically, the system builds the model by asserting ground facts into the Prolog internal database, and tests for satisfiability by means of Prolog evaluation over the program, which consists of the Prolog rules together with the asserted facts.

The model generation algorithm of SATCHMO is not directly related to the operational semantics presented here, but looks more like the program completion. Indeed, Theorem 36 ensures that completable programs have a Tarski model. Further investigations are required to better understand the relationship between completion and model generation.

In the context of pure forward chaining provers, the CPUHR tableaux [3] are an extension with the constraints of the PUHR tableaux [8], the underlying theoretical foundation of SATCHMO. In this formalism, the rules correspond to the forward rules without existential quantification, and are processed in a way similar to the maximal semantics where the guard entailment check is replaced by the so-called $\exists$-unification. $\exists$-unification differs from entailment checks in the sense that rather than just answering the question of whether the built-in store entails a guard, it computes a disjunction of constraints that represents the set of instances of the store that imply this guard. Hence, CPUHR tableaux can deal with non-simple constraint system, but are in general incomplete with respect to failure.

8. Conclusions

In this article, we have aimed to combine backward and forward chaining in a CLP framework in a logically complete way to take advantage of the strengths of each type of chaining. Specifically, by enforcing the idempotence of both conjunction and disjunction within states, we have derived from CHR$^\vee$ a new class of constraint logic programming languages. We argue that these new languages capture both backward chaining, when the programs are read from right to left, and forward chaining, when the programs are read from left to right. We have also demonstrated that the least and the greatest Tarski models of the confluent programs can be captured by a least and a greatest fixed point, respectively. Based on this, we have established the completeness of the languages with respect to answers and accessible constraints.

Acknowledgments

We would like to thank the reviewers for their useful comments.

References

[1] S. Abdennadher and T. W. Frühwirth. On completion of Constraint Handling Rules. In *CP*, volume 1520 of *LNCS*, pages 25–39. Springer, 1998.

[2] S. Abdennadher, T. W. Frühwirth, and H. Meuss. Confluence and semantics of Constraint Simplification Rules. *Constraints*, 4(2):133–165, 1999.

[3] S. Abdennadher and H. Schütz. Model generation with existentially quantified variables and constraints. In *ALP/HOA*, volume 1298 of *LNCS*, pages 256–272. Springer, 1997.

[4] S. Abdennadher and H. Schütz. Chrv: A flexible query language. In *FQAS*, volume 1495 of *LNCS*, pages 1–14. Springer, 1998.

[5] C. Beeri and R. Ramakrishnan. On the power of magic. In *PODS*, pages 269–284. ACM, 1987.

[6] H. Betz and T. W. Frühwirth. Linear-logic based analysis of Constraint Handling Rules with disjunction. *ACM Trans. Comput. Log.*, 14(1):1, 2013.

[7] M. Bezem and T. Coquand. Automating coherent logic. In *LPAR*, volume 3835 of *LNCS*, pages 246–260. Springer, 2005.

[8] F. Bry and A. H. Yahya. Positive unit hyperresolution tableaux and their application to minimal model generation. *J. Autom. Reasoning*, 25(1):35–82, 2000.

[9] K. Chaudhuri, F. Pfenning, and G. Price. A logical characterization of forward and backward chaining in the inverse method. *J. Autom. Reasoning*, 40(2-3):133–177, 2008.

[10] K. L. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322, 1977.

[11] G. J. Duck. SMCHR: Satisfiability modulo Constraint Handling Rules. *TPLP (ICLP'12 Special Issue)*, 12(4-5):601–618, 2012.

[12] F. Fages, P. Ruet, and S. Soliman. Linear concurrent constraint programming: Operational and phase semantics. *Inf. Comput.*, 165(1):14–41, 2001.

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

[14] R. Haemmerlé. Diagrammatic confluence for Constraint Handling Rules. *TPLP (ICLP'12 Special Issue)*, 12(4-5):737–753, 2012.

[15] R. Haemmerlé. On the confluence of analytical semantics of CHR. Technical Report CLIP-2/2014.0, CLIP Lab, 2014. `http://cliplab.org/papers/HaemmerleCLIP14.pdf`

[16] R. Haemmerlé and F. Fages. Abstract critical pairs and confluence of arbitrary binary relations. In *RTA*, volume 4533 of *LNCS*, pages 214–228. Springer, 2007.

[17] R. Haemmerlé, P. López-García, and M. V. Hermenegildo. CLP projection for Constraint Handling Rules. In *PPDP*, pages 137–148. ACM, 2011.

[18] P. V. Hentenryck, V. A. Saraswat, and Y. Deville. Design, implementation, and evaluation of the constraint language cc(fd). *J. Log. Program.*, 37(1-3):139–164, 1998.

[19] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Inf. Comput.*, 110(2):327–365, 1994.

[20] G. Huet. *Résolution d'équations dans des langages d'ordre* $1, 2, \ldots, \omega$. PhD thesis, Université Paris VII, 1976.

[21] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *POPL*, pages 111–119. ACM Press, 1987.

[22] J. Jaffar, M. J. Maher, K. Marriott, and P. J. Stuckey. The semantics of constraint logic programs. *J. Log. Program.*, 37(1-3):1–46, 1998.

[23] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *J. Comput. Syst. Sci.*, 51(1):26–52, 1995.

[24] J. Lloyd. *Foundations of Logic Programming*. springer, 1987.

[25] P. López, F. Pfenning, J. Polakow, and K. Watkins. Monadic concurrent linear logic programming. In *PPDP*, pages 35–46. ACM, 2005.

[26] M. J. Maher. Logic semantics for a class of committed-choice programs. In *ICLP*, pages 858–876. MIT Press, 1987.

[27] R. Manthey and F. Bry. Satchmo: A theorem prover implemented in prolog. In *CADE*, volume 310 of *LNCS*, pages 415–434. Springer, 1988.

[28] K. Marriott and P. J. Stuckey. *Programming with Constraints: an Introduction*. MIT Press, 1998.

[29] F. Raiser, H. Betz, and T. W. Frühwirth. Equivalence of CHR states revisited. In *CHR*, Report CW 555, pages 34–48. Kath. Univ. Leuven, 2009.

[30] R. Ramakrishnan. Magic templates: A spellbinding approach to logic programs. *J. Log. Program.*, 11(3&4):189–216, 1991.

[31] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[32] V. A. Saraswat and M. C. Rinard. Concurrent constraint programming. In *POPL*, pages 232–245. ACM Press, 1990.

[33] V. A. Saraswat, M. C. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *POPL*, pages 333–352. ACM Press, 1991.

[34] B. Sarna-Starosta and C. R. Ramakrishnan. Compiling Constraint Handling Rules for efficient tabled evaluation. In *PADL*, volume 4354 of *LNCS*, pages 170–184. Springer, 2007.