THE SEMANTICS OF CONSTRAINT LOGIC PROGRAMS

JOXAN JAFFAR, MICHAEL MAHER, KIM MARRIOTT AND PETER STUCKEY

▷ The Constraint Logic Programming (CLP) Scheme was introduced by Jaffar and Lassez. The scheme gave a formal framework, based on constraints, for the basic operational, logical and algebraic semantics of an extended class of logic programs. This paper presents for the first time the semantic foundations of CLP in a self-contained and complete package. The main contributions are threefold. First, we extend the original conference paper by presenting definitions and basic semantic constructs from first principles, giving new and complete proofs for the main lemmas. Importantly, we clarify which theorems depend on conditions such as solution compactness, satisfaction completeness and independence of constraints. Second, we generalize the original results to allow for incompleteness of the constraint solver. This is important since almost all CLP systems use an incomplete solver. Third, we give conditions on the (possibly incomplete) solver which ensure that the operational semantics is confluent, that is, has independence of literal scheduling. ◁

1. INTRODUCTION

The Constraint Logic Programming (CLP) Scheme was introduced by Jaffar and Lassez [8]. The scheme gave a formal framework, based on constraints, for the basic operational, logical and algebraic semantics of an extended class of logic programs. This framework extended traditional logic programming in a natural way

Address correspondence to Joxan Jaffar, Department of Information Systems and Computer Science, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260 Email: joxan@iscs.nus.edu.sg
Michael Maher, School of Computing and Information Technology, Griffith University, Nathan, Queensland 4111, Australia. Email: m.maher@cit.gu.edu.au
Kim Marriott, Department of Computer Science, Monash University, Clayton Vic. 3168, Australia. Email: marriott@cs.monash.edu.au
Peter Stuckey, Department of Computer Science, University of Melbourne, Parkville 3052, Australia. Email: pjs@cs.mu.oz.au

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/94/$7.00

by generalizing the term equations of logic programming to constraints from any pre-defined computation domain. Different classes of constraints give rise to different programming languages with different areas of application. Since then there has been considerable interest in the semantics and implementation of CLP languages, in part because they have proven remarkably useful, for systems modeling and for solving complex combinatorial optimization problems [11, 20].

CLP languages have a rich semantic theory which generalizes earlier research into semantics for logic programs. In the context of logic programs, van Emden and Kowalski [4] gave a simple and elegant fixpoint and model theoretic semantics for definite clause logic programs based on the least Herbrand model of a program. Apt and van Emden [1] extended this work to establish the soundness and completeness of the operational semantics (SLD resolution) with respect to success and to characterize finite failure. Clark [2] introduced the program completion as a logical semantics for finite failure and proved soundness of the operational semantics with respect to the completion. Jaffar et al [9] proved completeness of the operational semantics with respect to the completion. Together these results provide an elegant algebraic, fixpoint and logical semantics for pure logic programs. The book of Lloyd [17] provides a detailed introduction to the semantics of logic programs.

One natural generalization of logic programs is to allow different unification mechanisms in the operational semantics. Such a generalization was welcomed since it promised the integration of the functional and logical programming paradigms. Jaffar et al [10] generalized the theory of pure logic programs to a logic programming scheme which was parametric in the underlying equality theory, and proved that the main semantic results continued to hold. However, the theory of logic programs with equality was still not powerful enough to handle logic languages which provided more than equations. In particular, Prolog II [3] provided inequations over the rational trees. Jaffar and Stuckey [13] showed that the standard semantic results still held for Prolog II in the presence of inequations. The CLP Scheme generalized these two strands of work to provide a scheme over arbitrary constraints which could be equations, inequations or whatever. Somewhat surprisingly, the key results for the logic programming semantics continue to hold in this much more general setting. Indeed, as we shall show, presenting the standard logic programming results in terms of CLP actually results in a more direct and elegant formalization and provides deeper insight into why the results hold for logic programming.

This paper presents for the first time the semantic foundations of CLP in a self-contained and complete package. The original presentation of the CLP scheme was in the form of an extended abstract [8], referring much of the technical details, including all formal proofs, to an unpublished report [7]. The conference paper of Maher [18] provided a stronger completeness result. Subsequent papers on CLP semantics have either been partial in the sense that they focus on certain aspects only, or they have been informal, being part of a tutorial or survey. Indeed, Jaffar and Maher's comprehensive survey of CLP [11] did not present the semantics in a formal way, nor include any important proofs. The main contributions of the present paper are:

- We extend the original conference papers by presenting definitions and basic semantic constructs from first principles, with motivating discussions and examples, and give new and complete proofs for the main lemmas. Importantly, we clarify which theorems depend on conditions such as solution compactness, satisfaction completeness and independence of constraints.

- We generalize the original results to allow for incompleteness of the constraint solver. This is important since almost all CLP systems use an incomplete solver.
- We give conditions on the (possibly incomplete) solver which ensure that the operational semantics is confluent, that is, has independence of literal scheduling.

A synopsis is as follows. In the next section we introduce the notions of constraints, solvers and constraint domains. In Section 3 the operational semantics of CLP is introduced, together with breadth-first derivations. In Section 4, soundness and completeness results for successful derivations are derived. Also, two fixpoint semantics are introduced. In Section 5 we give soundness and completeness results for finite failure. Section 6 summarizes our main results and relates them to the standard results for logic programming.

2. CONSTRAINTS

We assume that the reader is familiar with the basics of first-order logic. See for example [22]. We use the notation $\vec{s}$ to denote a sequence of terms or variables $s_1, \ldots, s_n$. In an abuse of notation we shall often write $\vec{s} = \vec{t}$, where $\vec{s}$ and $\vec{t}$ are vectors of length n, to denote the sequence (or conjunction) of equations $s_1 = t_1, \ldots, s_n = t_n$.

We let $\exists \vec{x} F$, where $\vec{x}$ is a vector of variables, denote the logical formula $\exists x_1 \exists x_2 \cdots \exists x_n F$ Similarly we let $\exists_W F$ denote the logical formula $\exists x_1 \exists x_2 \cdots \exists x_n F$ where variable set $W = \{x_1, \ldots, x_n\}$, and we let $\bar{\exists}_W F$ denote the restriction of the logical formula F to the variables in W. That is, $\bar{\exists}_W F$ is $\exists_{vars(F) \setminus W} F$, where the function $vars$ takes a syntactic object and returns the set of free variables occurring in it. We let $\tilde{\exists} F$ denote the existential closure of F and $\tilde{\forall} F$ denote the universal closure of F.

A *renaming* is a bijective mapping between variables. We naturally extend renamings to mappings between logical formulas, rules, and constraints. Syntactic objects s and s' are said to be *variants* if there is a renaming ρ such that $\rho(s) = s'$.

A *signature* defines a set of function and predicate symbols and associates an arity with each symbol. A Σ*-structure*, $\mathcal{D}$, is an interpretation of the symbols in the signature Σ. It consists of a set D and a mapping from the symbols in Σ to relations and functions over D which respects the arities of the symbols. A *first-order* Σ*-formula* is a first order logical formula built from variables and the function and predicate symbols in Σ in the usual way using the logical connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$ and the quantifiers $\exists$ and $\forall$. A Σ*-theory* is a possibly infinite set of closed Σ-formulas. A *solver* for a set $\mathcal{L}$ of Σ-formulas is a function which maps each formula to one of $true$, $false$ or $unknown$, indicating that the formula is satisfiable, unsatisfiable or it cannot tell.

CLP languages extend logic-based programming languages by allowing constraints with a pre-defined interpretation. The key insight of the CLP scheme is that for these languages the operational semantics, declarative semantics and the relationship between them can be parameterized by a choice of constraints, solver and an algebraic and logical semantics for the constraints.

More precisely, the scheme defines a class of languages, $CLP(\mathcal{C})$, which are parametric in the *constraint domain* $\mathcal{C}$. The constraint domain contains the following components:

- the *constraint domain signature*, $\Sigma_\mathcal{C}$;
- the class of *constraints*, $\mathcal{L}_\mathcal{C}$, which is some predefined subset of first-order Σ-formulas;
- the *domain of computation*, $\mathcal{D}_\mathcal{C}$, which is a Σ-structure that is the intended interpretation of the constraints;
- the *constraint theory*, $\mathcal{T}_\mathcal{C}$, which is a Σ-theory that describes the logical semantics of the constraints; and
- the *solver*, $solv_\mathcal{C}$, which is a solver for $\mathcal{L}_\mathcal{C}$.

We assume that:

- The binary predicate symbol "=" is in $\Sigma_\mathcal{C}$, that $=$ is interpreted as identity in $\mathcal{D}_\mathcal{C}$ and that $\mathcal{T}_\mathcal{C}$ contains the standard equality axioms for $=$.
- The class of constraints $\mathcal{L}_\mathcal{C}$ contains, among other formulas, all atoms constructed from $=$, the always satisfiable constraint *true* and the unsatisfiable constraint $false$ and is closed under variable renaming, existential quantification and conjunction.
- The solver does not take variable names into account, that is, for all renamings ρ, $solv_\mathcal{C}(c) = solv_\mathcal{C}(\rho(c))$.
- The domain of computation, solver and constraint theory *agree* in the sense that $\mathcal{D}_\mathcal{C}$ is a model of $\mathcal{T}_\mathcal{C}$ and that for any constraint $c \in \mathcal{L}_\mathcal{C}$, if $solv_\mathcal{C}(c) = false$ then $\mathcal{T}_\mathcal{C} \models \neg\,\tilde{\exists} c$, and if $solv_\mathcal{C}(c) = true$ then $\mathcal{T}_\mathcal{C} \models \tilde{\exists} c$.

For a particular constraint domain $\mathcal{C}$, we call an element of $\mathcal{L}_\mathcal{C}$ a *constraint* and an atomic constraint is called a *primitive constraint*.

In this paper we will make use of the following two example constraint domains.

Example 2.1. The constraint domain *Real* which has $\leq, \geq, <, >, =$ as the relation symbols, function symbols $+$, $-$, $*$ and $/$, and sequences of digits with an optional decimal point as constant symbols. The intended interpretation of *Real* has as its domain the set of real numbers, $\mathcal{R}$. The primitive constraints $\leq, \geq, <, >, =$ are interpreted as the obvious arithmetic relations over $\mathcal{R}$, and the function symbols $+$, $-$, $*$ and $/$, are the obvious arithmetic functions over $\mathcal{R}$. Constant symbols are interpreted as the decimal representation of elements of $\mathcal{R}$. The theory of the *real closed fields* is a theory for *Real* [22]. A possible implementation of a solver for *Real* is based on that of $CLP(\mathcal{R})$ [12]. It uses the Simplex algorithm and Gauss-Jordan elimination to handle linear constraints and delays non-linear constraints until they become linear.

Example 2.2. The constraint domain $Term$ has $=$ as the primitive constraint, and strings of alphanumeric characters as function symbols or as constant symbols. $CLP(Term)$ is, of course, the core of the programming language Prolog.

The intended interpretation of $Term$ is the set of finite trees, $Tree$. The interpretation of a constant a is a tree with a single node labeled with a. The interpretation of the n-ary function symbol f is the function $f_{Tree} : Tree^n \to Tree$ which maps the trees $T_1, \ldots, T_n$ to a new tree with root node labeled by f and with $T_1, \ldots, T_n$ as children. The interpretation of $=$ is the identity relation over $Tree$. The natural theory, $\mathcal{T}_{Term}$, was introduced in logic programming by Clark [2] (see also [19]) in which "=" is required to be syntactic equality on trees. The unification algorithm is a constraint solver for this domain.

Note that if the solver returns *unknown* this means the solver cannot determine satisfiability; it does not mean that the constraint theory does not imply satisfiability or unsatisfiability of the constraint. Thus the solver is allowed to be incomplete. Because of the agreement requirement, a solver for constraint domain $\mathcal{C}$ can only be as powerful as the constraint domain theory $\mathcal{T}_\mathcal{C}$. A solver with this property is *theory complete*. That is a, a solver is theory complete whenever

- $solv_\mathcal{C}(c) = false$ iff $\mathcal{T}_\mathcal{C} \models \neg\, \tilde{\exists} c$, and
- $solv_\mathcal{C}(c) = true$ iff $\mathcal{T}_\mathcal{C} \models \tilde{\exists} c$.

If the solver only ever returns $true$ or $false$ it is said to be *complete*. If the solver for constraint domain $\mathcal{C}$ is complete then we must have that the constraint theory $\mathcal{T}_\mathcal{C}$ is *satisfaction complete* [8], that is, for every constraint c, either $\mathcal{T}_\mathcal{C} \models \neg\, \tilde{\exists} c$ or $\mathcal{T}_\mathcal{C} \models \tilde{\exists} c$.

It is important to note that a theory for a constraint domain may have models which are very different to the intended model. If the solver is not complete, then constraints which are false in the domain of computation $\mathcal{D}_\mathcal{C}$ may be true in these models. If the solver is complete then all models must agree about whether a constraint is satisfiable or not. We call a model which is not the intended model a *non-standard* model.

Example 2.3. A well-known non-standard model of the real closed field (due to Abraham Robinson, see e.g. [21]) is the model $\mathcal{R}^\star$ which contains (1) "infinitesimals" which are not zero but smaller than every non-zero real number and (2) "infinite elements" which are larger than every real number.

Note that from the above definition we can easily define a constraint domain $\mathcal{C}$ given a signature $\Sigma_\mathcal{C}$, language of constraints $\mathcal{L}_\mathcal{C}$ and a solver $solv_\mathcal{C}$ and either a domain of computation or a constraint theory that agrees with $solv_\mathcal{C}$. Given a domain of computation $\mathcal{D}_\mathcal{C}$, then a suitable constraint theory $\mathcal{T}_\mathcal{C}$ is just the theory of $\mathcal{D}_\mathcal{C}$, that is all first order formulae true in $\mathcal{D}_\mathcal{C}$. Alternatively given a constraint theory $\mathcal{T}_\mathcal{C}$ we can take $\mathcal{D}_\mathcal{C}$ to be an arbitrary model of the theory.

A constraint domain provides three different semantics for the constraints: an operational semantics given by the solver, an algebraic semantics given by the intended interpretation, and a logical semantics given by the theory. One of the nicest properties of the CLP languages is that it is possible to also give an operational, algebraic and logical semantics to the user defined predicates, that is programs. We now do so.

3. OPERATIONAL SEMANTICS

In this section we define an abstract operational semantics for constraint logic programs based on top-down derivations and investigate when the semantics is confluent, that is when the results are independent from the literal selection strategy. We also introduce a canonical form of operational semantics, breadth-first derivations, which will prove a useful bridge to the algebraic semantics.

3.1. Constraint logic programs and their operational semantics

As described in the last section, a constraint logic programming language is *parameterized* by the underlying *constraint domain* $\mathcal{C}$. The constraint domain determines the constraints and the set of function and constant symbols from which terms in the program may be constructed, as well as a solver $solv_{\mathcal{C}}$. The solver determines when (or if) to prune a branch in the derivation tree. Different choices of constraint domain and solver give rise to different programming languages. For a particular constraint domain $\mathcal{C}$, we let $CLP(\mathcal{C})$ be the constraint programming language based on $\mathcal{C}$.

A *constraint logic program* (CLP), or *program*, is a finite set of rules. A *rule* is of the form H :- B where H, the *head*, is an atom and B, the *body*, is a finite, non-empty sequence of literals. We let $\Box$ denote the empty sequence. We shall write rules of the form H :- $\Box$ simply as H. A *literal* is either an atom or a primitive constraint. An *atom* has the form $p(t_1, ..., t_n)$ where p is a user-defined predicate symbol and the t_i are terms from the constraint domain.

Our examples will make use of the language $CLP(Real)$ which is based on the constraint domain *Real* and the language $CLP(Term)$ which is based on the constraint domain $Term$.

The *definition of an atom* $p(t_1, ..., t_n)$ *in program* P, $defn_P(p(t_1, ..., t_n))$, is the set of rules in P such that the head of each rule has form $p(s_1, ..., s_n)$. To side-step renaming issues, we assume that each time $defn_P$ is called it returns variants with distinct new variables.

The operational semantics is given in terms of the "derivations" from goals. Derivations are sequences of reductions between "states", where a *state* is a tuple $\langle G \mid c\rangle$ which contains the current literal sequence or "goal" G and the current constraint c. At each reduction step, a literal in the goal is *selected* according to some fixed *selection rule*, which is often left-to-right. If the literal is a primitive constraint, and it is consistent with the current constraint, then it is added to it. If it is inconsistent then the derivation "fails". If the literal is an atom, it is reduced using one of the rules in its definition.

A state $\langle L_1, ..., L_m \mid c\rangle$ can be *reduced* as follows: Select a literal L_i then:

1. If L_i is a primitive constraint and $solv(c \wedge L_i) \neq false$, it is reduced to $\langle L_1, ..., L_{i-1}, L_{i+1}, ..., L_m \mid c \wedge L_i\rangle$.

2. If L_i is a primitive constraint and $solv(c \wedge L_i) = false$, it is reduced to $\langle \Box \mid false\rangle$.

3. If L_i is an atom, then it is reduced to

$$\langle L_1, ..., L_{i-1}, s_1 = t_1, ..., s_n = t_n, B, L_{i+1}, ..., L_m \mid c\rangle$$

for some $(A \texttt{ :- } B) \in defn_P(L_i)$ where L_i is of form $p(s_1, ..., s_n)$ and A is of form $p(t_1, ..., t_n)$.

4. If L_i is an atom and $defn_P(L_i) = \emptyset$, it is reduced to $\langle \Box \mid false \rangle$.

A *derivation* from a state S in a program P is a finite or infinite sequence of states $S_0 \Rightarrow S_1 \Rightarrow \cdots \Rightarrow S_n \Rightarrow \cdots$ where S_0 is S and there is a reduction from each S_{i-1} to S_i, using rules in P. A *derivation* from a goal G in a program P is a derivation from $\langle G \mid true \rangle$. The *length* of a (finite) derivation of the form $S_0 \Rightarrow S_1 \Rightarrow \cdots \Rightarrow S_n$ is n. A derivation is *finished* if the last goal cannot be reduced. The last state in a finished derivation from G must have the form $\langle \Box \mid c \rangle$. If c is $false$ the derivation is said to be *failed*. Otherwise the derivation is *successful*. The *answers* of a goal G for program P are the constraints $\bar{\exists}_{vars(G)} c$ where there is a successful derivation from G to final state with constraint c. Note that in the operational semantics the answer is treated syntactically.

In many implementations of CLP languages the answer is simplified into a logically equivalent constraint, perhaps by removing existentially quantified variables, before being shown to the user. For simplicity we will ignore such a simplification step although our results continue to hold modulo logical equivalence with respect to the theory.

Example 3.1. Consider the following simple $CLP(Real)$ program to compute the factorial of a number:

$(R1)$ $fac(0, 1)$.
$(R2)$ $fac(N, N * F) \texttt{ :- } N \geq 1,\ fac(N - 1, F)$.

A successful derivation from the goal $fac(1, X)$ is:

$$
\begin{array}{c}
\langle \underline{fac(1,X)} \mid true \rangle \\
\Downarrow R2 \\
\langle \underline{1 = N}, X = N \times F, N \geq 1, fac(N-1,F) \mid true \rangle \\
\Downarrow \\
\langle \underline{X = N \times F}, N \geq 1, fac(N-1,F) \mid 1 = N \rangle \\
\Downarrow \\
\langle \underline{N \geq 1}, fac(N-1,F) \mid 1 = N \wedge X = N \times F \rangle \\
\Downarrow \\
\langle \underline{fac(N-1,F)} \mid 1 = N \wedge X = N \times F \wedge N \geq 1 \rangle \\
\Downarrow R1 \\
\langle \underline{N - 1 = 0}, F = 1 \mid 1 = N \wedge X = N \times F \wedge N \geq 1 \rangle \\
\Downarrow \\
\langle \underline{F = 1} \mid 1 = N \wedge X = N \times F \wedge N \geq 1 \wedge N - 1 = 0 \rangle \\
\Downarrow \\
\langle \Box \mid 1 = N \wedge X = N \times F \wedge N \geq 1 \wedge N - 1 = 0 \wedge F = 1 \rangle
\end{array}
$$

In each step the selected literal is underlined, and if an atom is rewritten, the rule used is written beside the arrow. Since the intermediate variables are not of interest, they are quantified away to give the answer

$$\exists N \exists F(1 = N \wedge X = N \times F \wedge N \geq 1 \wedge N - 1 = 0 \wedge F = 1)$$

which is logically equivalent to $X = 1$.

Example 3.2. Consider the factorial program again. One failed derivation from the goal $fac(2, X)$ is:

$$\begin{array}{c} \langle \underline{fac(2, X)} \mid true \rangle \\ \Downarrow R1 \\ \langle \underline{2 = 0}, X = 1 \mid true \rangle \\ \Downarrow \\ \langle \Box \mid false \rangle \end{array}$$

Note that because the solver can be incomplete, a successful derivation may give an answer which is unsatisfiable since the solver may not be powerful enough to recognize that the constraint is unsatisfiable.

Example 3.3. For example using the solver of $CLP(\mathcal{R})$, the following derivation is possible:

$$\begin{array}{c} \langle \underline{Y = X \times X}, Y < 0 \mid true \rangle \\ \Downarrow \\ \langle \underline{Y < 0} \mid Y = X \times X \rangle \\ \Downarrow \\ \langle \Box \mid Y = X \times X \wedge Y < 0 \rangle \end{array}$$

Definition 3.1. An answer c to a goal G for program P is *satisfiable* if $\mathcal{T}_{\mathcal{C}} \models \tilde{\exists} c$. Otherwise c is a *pseudo-answer* for G.

3.2. Confluence of the operational semantics

In the definition of derivation, there are three sources of non-determinism. The first is the choice of which rule to use when rewriting an atom. The second is the choice of how to rename the rule. The third is the choice of the selected literal. Different choices for which rule to rewrite with lead to different answers, and so for completeness an implementation must consider all choices. However, in this subsection we give simple conditions on the solver which ensure that the choice of the selected literal and choice of the renaming do not effect the outcome. This allows an implementation to use fixed rules for renaming and for selecting the literal with a guarantee that it will still find all of the answers. This is important for the efficient implementation of constraint logic programs systems.

The results of this section generalize those given in Lloyd [17] for logic programs. The primary difference from the logic programming case is that not considering substitutions makes the results much easier to obtain. One technical difference is the need to consider incomplete solvers.

In general, the strategy used to rename rules does not affect the derivations of a goal or its answers in any significant way. This is because the names of the local variables do not affect the validity of the derivation as the solver does not take names of variables into account.

We now show that the results of evaluation are "essentially" independent from the choice of literal selection. We will first define precisely what we mean by a literal selection strategy (called a "computation rule" in Lloyd [17]).

Definition 3.2. A *literal selection strategy* $\mathcal{S}$ is a function which given a derivation returns a literal L in the last goal in the derivation.

A derivation is *via* a selection rule $\mathcal{S}$ if all choices of the selected atoms in the derivation are performed according to $\mathcal{S}$. That is, if the derivation is

$$\langle G_1 \mid c_1 \rangle \Rightarrow \langle G_2 \mid c_2 \rangle \Rightarrow \cdots \Rightarrow \langle G_n \mid c_n \rangle \Rightarrow \cdots$$

then for each $i \geq 1$, the literal selected from state $\langle G_i \mid c_i \rangle$ is

$$\mathcal{S}(\langle G_1 \mid c_1 \rangle \Rightarrow \cdots \Rightarrow \langle G_i \mid c_i \rangle).$$

Note that a literal selection strategy is free to select different literals in the same goal if it occurs more than once in the derivation.

Unfortunately, answers are not independent of the literal selection strategy for all solvers. The first problem is that different selection strategies can collect the constraints in different orders, and the solver may take the order of the primitive constraints into account when determining satisfiability.

Example 3.4. Consider the goal $p(X)$ and the program

$$p(Y) \text{ :- } Y = 1, Y = 2.$$

Imagine that the solver, *solv*, is defined so that it does not consider the last primitive constraint occurring in its argument. That is,

$$\begin{array}{ll}
solv(X = Y) & = unknown \\
solv(X = Y \wedge Y = 1) & = unknown \\
solv(X = Y \wedge Y = 1 \wedge Y = 2) & = unknown \\
solv(Y = 2) & = unknown \\
solv(Y = 2 \wedge Y = 1) & = unknown \\
solv(Y = 2 \wedge Y = 1 \wedge X = Y) & = false
\end{array}$$

Using a left-to-right literal selection strategy with this solver, the answer $\exists Y(X = Y \wedge Y = 1 \wedge Y = 2)$ is obtained. However, with a right-to-left selection strategy the goal has a single failed derivation.

The second problem is shown in the following example.

Example 3.5. Consider the goal and the program from the preceding example. Imagine that the solver, *solv*, is now defined so that it is complete for all constraints with only two primitives and returns *unknown* for larger constraints. That is,

$$\begin{array}{ll} solv(X = Y) & = true \\ solv(X = Y \land Y = 1) & = true \\ solv(X = Y \land Y = 1 \land Y = 2) & = unknown \\ solv(Y = 2) & = true \\ solv(Y = 2 \land Y = 1) & = false \\ solv(Y = 2 \land Y = 1 \land X = Y) & = unknown \end{array}$$

Using a left-to-right literal selection strategy with this solver, the answer $\exists Y(X = Y \land Y = 1 \land Y = 2)$ is obtained. However, with a right-to-left selection strategy the goal has a single failed derivation. The problem is that the solver is not "monotonic".

Fortunately, most real world solvers do not exhibit such pathological behavior. They are well-behaved in the following sense.

Definition 3.3. A constraint solver *solv* for constraint domain $\mathcal{C}$ is *well-behaved* if for any constraints c_1 and c_2 from $\mathcal{C}$:

logical: $solv(c_1) = solv(c_2)$ whenever $\models c_1 \leftrightarrow c_2$. That is, if c_1 and c_2 are logically equivalent *using no information* about the constraint domain, then the solver answers the same for both.

monotonic: if $solv(c_1) = false$ then $solv(c_2) = false$ whenever $\models c_1 \leftarrow \bar{\exists}_{vars(c_1)} c_2$. That is, if the solver fails c_1 then, whenever c_2 contains "more constraints" than c_1, the solver also fails c_2.

The solvers in the above two examples are not well-behaved. The solver in the first example is not logical, while that of the second example is not monotonic. Note that the above definitions do not use information from the constraint domain and so do not assume that equality is modeled by identity. For instance, a monotonic solver for *Real* is allowed to map $solv(1 = 0)$ to $false$ and $solv(X * Y = 1 \land X * Y = 0)$ to *unknown*. We note that any complete solver is well-behaved.

We can prove that for well-behaved solvers the answers are independent of the selection strategy. The core of the proof of this result is contained in the following lemma:

Lemma 3.1. (Switching Lemma)
Let S be a state and L, L' be literals in the goal of S. Let solv be a well-behaved solver and let $S \Rightarrow S_1 \Rightarrow S'$ be a non-failed derivation constructed using solv with L selected first, followed by L'. There is a derivation $S \Rightarrow S_2 \Rightarrow S''$ also constructed using solv in which L' is selected first, followed by L, and S' and S'' are identical up to reordering of their constraint components.

Proof. There are four ways by which S can be reduced to S'. For simplicity we will assume that S is the state $\langle L, L' \mid c\rangle$. This clarifies the argument by removing the need to keep track of other literals in the goal which are unaffected by the reductions.

1. In the first case both L and L' are constraints. In this case S_1 is $\langle L' \mid c \land L\rangle$ and S' is $\langle \Box \mid c \land L \land L'\rangle$. If we choose S_2 to be $\langle L \mid c \land L'\rangle$ and S'' to

be $\langle \Box \mid c \wedge L' \wedge L \rangle$ then $S \Rightarrow S_2 \Rightarrow S''$ is a valid derivation as we know that $solv(c \wedge L \wedge L') \neq false$ and so from well-behavedness of the constraint solver, $solv(c \wedge L') \neq false$ and $solv(c \wedge L' \wedge L) \neq false$.

2. The second case is when L and L' are both atoms. Assume that L is of form $p(t_1, \ldots, t_m)$ and was reduced using the rule renaming of form $p(s_1, \ldots, s_m)$ `:-` B and that L' is of form $q(t'_1, \ldots, t'_{m'})$ and was reduced using the rule renaming of form $q(s'_1, \ldots, s'_{m'})$ `:-` B' . Then S_1 is

$$\langle t_1 = s_1, \ldots, t_m = s_m, B, L' \mid c \rangle$$

and S' is

$$\langle t_1 = s_1, \ldots, t_m = s_m, B, t'_1 = s'_1, \ldots, t'_{m'} = s'_{m'}, B' \mid c \rangle.$$

In this case we choose S_2 to be

$$\langle L, t'_1 = s'_1, \ldots, t'_{m'} = s'_{m'}, B' \mid c \rangle$$

and S'' to be S'. Clearly $S \Rightarrow S_2 \Rightarrow S'$ is a valid derivation since the rule renamings are still disjoint from each other.

3. In the second case L is a constraint and L' is an atom. This case is a simple combination of the above two cases.

4. In the third case L' is a constraint and L is an atom. It is symmetric to the previous case. $\Box$

We can now prove that for well-behaved solvers the operational semantics is confluent, that is independent of the literal selection strategy.

Theorem 3.1. (Independence of the Literal Selection Strategy)
Assume that the underlying constraint solver is well-behaved and let P be a program and G a goal. Suppose that there is derivation from G with answer c. Then, for any literal selection strategy $\mathcal{S}$, there is a derivation of the same length from G via $\mathcal{S}$ with an answer which is a reordering of c.

PROOF. The induction hypothesis is that if there is a successful derivation D of length N from a state S to state $\langle \Box \mid c \rangle$ then for $\mathcal{S}$, there is a derivation of the same length from S using $\mathcal{S}$ to $\langle \Box \mid c' \rangle$ where c' is a reordering of c. The proof is by induction on the length of D. In the base case when the length N is 0, S is simply $\langle \Box \mid c \rangle$ and the result clearly holds.

We now prove the induction step. Consider the derivation D of length $N+1$,

$$S \Rightarrow S_1 \Rightarrow \cdots \Rightarrow S_N \Rightarrow \langle \Box \mid c \rangle.$$

Assume that $\mathcal{S}$ selects literal L in the (singleton state) derivation S. As D is a successful derivation, every literal in D must be selected at some stage. Thus L must be selected at some point, say when reducing S_i to S_{i+1}. By applying Lemma 3.1 i times we can reorder D to obtain a derivation E of form

$$S \Rightarrow S'_1 \Rightarrow \cdots \Rightarrow S'_N \Rightarrow \langle \Box \mid c'' \rangle$$

in which L is selected in state S and c'' is a reordering of c. From the induction hypothesis there is a derivation E' of length N using $\mathcal{S}'$ from S'_1 to $\langle \Box \mid c' \rangle$ where $\mathcal{S}'$ is a literal selection strategy which picks the same literal in E' as is picked by $\mathcal{S}$ in $S \Rightarrow E'$ and c' is reordering of c'' and hence of c. Thus the derivation $S \Rightarrow E'$ is the required derivation.

The proof follows by induction. $\Box$

Even for solvers which are not well-behaved, it is possible to show a weaker confluence result – namely that the answers which are satisfiable are the same. To show this, we first need a lemma which relates the "power" of the constraint solver to the answers.

Definition 3.4. Let *solv* and *solv'* be constraint solvers for the same constraint domain. Solver *solv* is *more powerful* than *solv'* if for all constraints c, $solv(c) = unknown$ implies $solv(c') = unknown$.

A more powerful constraint solver limits the size of derivations and the number of successful derivations since unsatisfiable constraints are detected earlier in the construction of the derivation and so derivations leading to pseudo-answers may fail. Successful derivations which have an answer which is satisfiable are, of course, not pruned.

Lemma 3.2. Let S be a state and solv and solv' be constraint solvers such that solv is more powerful than solv'.

(a) *Each derivation from S using solv is also a derivation from S using solv'.*

(b) *Each successful derivation from S using solv' with a satisfiable answer is also a derivation from S using solv.*

PROOF. Part (a) follows by induction on the length of the derivation and the definition of more powerful.

The proof of part (b) relies on the observation that if a successful derivation has an answer which is satisfiable then the constraint component of each state in the derivation must be satisfiable in the constraint theory. Thus *solv* cannot prune this derivation. $\Box$

We can now show that the successful derivations with satisfiable answers are independent of the solver used and of the literal selection strategy.

Theorem 3.2. (Weak Independence of the Literal Selection Strategy and Solver) Let P be a $CLP(\mathcal{C})$ program and G a goal. Suppose there is a successful derivation, D, from G with answer c and that c is satisfiable. Then for any literal selection strategy $\mathcal{S}$ and constraint solver solv for $\mathcal{C}$, there is a successful derivation from G via $\mathcal{S}$ using solv of the same length as D and which gives an answer which is a reordering of c.

PROOF. Let *usolv* be the solver for $\mathcal{C}$ which always returns *unknown*. Clearly any solver for $\mathcal{C}$ is more powerful than *usolv*. Thus it follows from Lemma 3.2 that D is also a successful derivation from S using *usolv*. Now *usolv* is well-behaved. Thus, from Theorem 3.1, there is a successful derivation D' from S via $\mathcal{S}$ using *usolv* of the same length as D and with an answer c' which is a reordering of c. Since c and hence c' is satisfiable, it follows from Lemma 3.2 that D' is also a derivation from S via $\mathcal{S}$ using *solv*. $\Box$

3.3. Derivation trees and finite failure

Independence of the literal selection strategy means that the implementation is free to use a single selection strategy since all answers will be found. The derivations from a goal for a single literal selection strategy can be conveniently collected together to form a "derivation tree". This is a tree such that each path from the top of the tree is a derivation. Branches occur in the tree when there is a choice of rule to reduce an atom with. In a CLP system, execution of a goal may be viewed as a traversal of the derivation tree.

Definition 3.5. A *derivation tree* for a goal G, program P and literal selection strategy $\mathcal{S}$ is a tree with states as nodes and constructed as follows. The root node of the tree is the state $\langle G \mid true \rangle$, and the children of a node in the tree are the states it can reduce to where the selected literal is chosen with $\mathcal{S}$.

A derivation tree represents all of the derivations from a goal for a fixed literal selection strategy. A derivation tree is unique up to variable renaming. A successful derivation is represented in a derivation tree by a path from the root to a leaf node with the empty goal and a constraint which is not $false$. A failed derivation is represented in a derivation tree by a path from the root to a leaf node with the empty goal and the constraint $false$.

Apart from returning answers to a goal, execution of a constraint logic program may also return the special answer *no* indicating that the goal has "failed" in the sense that all derivations of the goal are failed for a particular literal selection strategy.

Definition 3.6. If a state or goal G has a finite derivation tree for literal selection strategy $\mathcal{S}$ and all derivations in the tree are failed, G is said to *finitely fail* for $\mathcal{S}$.

Example 3.6. Recall the definition of the factorial predicate from before. The derivation tree for the goal $fac(0,2)$ constructed with a left-to-right literal selection strategy is shown in Figure 3.1. From the derivation tree we see that, with a left-to-right literal selection strategy, the goal $fac(0,2)$ finitely fails.

We have seen that the answers obtained from a goal are independent of the literal selection strategy used as long as the solver is well-behaved. However a goal may also finitely fail. It is therefore natural to ask when finite failure is independent of the literal selection strategy.

We first note that finite failure is not independent of the literal strategy if the solver is not well-behaved. For instance consider the solvers from Examples 3.4 and 3.5. For both solvers the goal $p(X)$ for the program in Example 3.4 finitely fails with a right-to-left literal selection strategy but does not finitely fail with a left-to-right literal selection strategy.

However, for independence we need more than just a well-behaved solver.

Example 3.7. Consider the program

$$p \text{ :- } p.$$

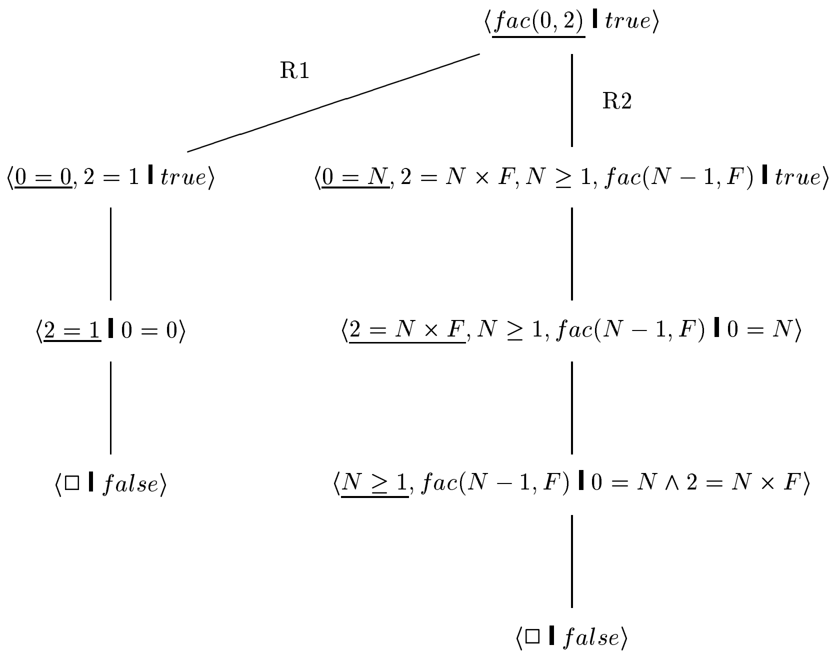

FIGURE 3.1. Derivation tree for $fac(0, 2)$

and the goal $(p, 1 = 2)$. With a left-to-right selection rule this goal has a single infinite derivation, in which p is repeatedly rewritten to itself. With a right-to-left selection rule however, this goal has a single failed derivation, so the goal finitely fails.

The reason independence does not hold for finite failure in this example is that in an infinite derivation, a literal which will cause failure may never be selected. To overcome this problem we require the literal selection strategy to be "fair" [16]:

Definition 3.7. A literal selection strategy $\mathcal{S}$ is *fair* if in every infinite derivation via $\mathcal{S}$ each literal in the derivation is selected.

A left-to-right literal selection strategy is not fair. A strategy in which literals that have been in the goal longest are selected in preference to newer literals in the goal is fair.

For fair literal selection strategies, finite failure is independent of the selection strategy whenever the underlying constraint solver is well-behaved.

Lemma 3.3. Let the underlying constraint solver be well-behaved. Let P be a program and G a goal. Suppose that G has a derivation of infinite length via a fair literal selection strategy $\mathcal{S}$. Then, G has a derivation of infinite length via any literal selection strategy $\mathcal{S}'$.

PROOF. Let D be a derivation of infinite length via $\mathcal{S}$. We inductively define a sequence of infinite fair derivations $D_0, D_1, D_2, \ldots$ such that for each N, if D_N is

$$S_0 \Rightarrow S_1 \Rightarrow \cdots \Rightarrow S_N \Rightarrow \cdots$$

then the derivation prefix,

$$S_0 \Rightarrow S_1 \Rightarrow \cdots \Rightarrow S_N,$$

is a derivation from G via $\mathcal{S}'$. The limit of this sequence is an infinite derivation from G via $\mathcal{S}'$.

For the base case $N = 0$, the derivation is just D itself. Now assume that D_N is

$$S_0 \Rightarrow S_1 \Rightarrow \cdots \Rightarrow S_N \Rightarrow S_{N+1} \Rightarrow S_{N+2} \Rightarrow \cdots$$

Let the literal L be selected by $\mathcal{S}'$ in S_N. As D_N is fair, L must also be selected at some stage in D_N, say at S_{N+i} where $i \geq 0$. By applying Lemma 3.1 i times we can reorder D_N to obtain a derivation D_{N+1} of the form

$$S_0 \Rightarrow S_1 \Rightarrow \cdots \Rightarrow S_N \Rightarrow S'_{N+1} \Rightarrow S'_{N+2} \Rightarrow \cdots$$

in which L is selected in state S_N. By construction

$$S_0 \Rightarrow S_1 \Rightarrow \cdots \Rightarrow S_N \Rightarrow S'_{N+1}$$

is a derivation from G via $\mathcal{S}'$. Also D_{N+1} is fair as it has only reordered a literal selection in the fair derivation D_N. $\Box$

Theorem 3.3. Assume that the underlying solver is well-behaved. Let P be a program and G a goal. Suppose that G finitely fails via literal selection strategy $\mathcal{S}$. Then, G will finitely fail via any fair literal selection strategy.

Proof. We prove the contrapositive, namely that if G does not finitely fail via a fair literal selection strategy $\mathcal{S}'$ then G cannot finitely fail via any other strategy, say $\mathcal{S}$. If G does not finitely fail with $\mathcal{S}'$, then the derivation tree D for G constructed with $\mathcal{S}'$ must have either a successful derivation or be infinite in size. If D contains a successful derivation then from Theorem 3.1 there will also be a successful derivation via $\mathcal{S}$, so G does not finitely fail with $\mathcal{S}$. Otherwise if D has no successful derivations but is infinite, then it must have a derivation of infinite length by Koenig's Lemma. By Lemma 3.3 there must be an infinite derivation from G via $\mathcal{S}$. But this means that G does not have a finite derivation tree with $\mathcal{S}$ and so does not finitely fail with $\mathcal{S}$. $\Box$

3.4. Breadth-first derivations

It will prove useful in subsequent sections to introduce a type of canonical top-down evaluation strategy. In this strategy all literals are reduced at each step in a derivation. For obvious reasons, such a derivation is called "breadth first." Breadth-first derivations were first introduced for logic programs in [24].

Definition 3.8. A *breadth-first derivation step* from $\langle G_0 \mid c_0 \rangle$ to $\langle G_1 \mid c_1 \rangle$ using program P, written $\langle G_0 \mid c_0 \rangle \Rightarrow_{BF(P)} \langle G_1 \mid c_1 \rangle$, is defined as follows. Let G_0 consist of the atoms $A_1, \ldots, A_m$ and the primitive constraints $c'_1, \ldots, c'_n$.

1. If $\mathcal{T}_{\mathcal{C}} \models \neg\tilde{\exists}\ (c_0 \wedge \bigwedge_{i=1}^{n} c'_i)$ or for some A_j in G_0, $defn_P(A_j) = \emptyset$, then G_1 is the empty goal and c_1 is $false$.
2. Otherwise, c_1 is $c_0 \wedge \bigwedge_{i=1}^{n} c'_i$ and G_1 is $B_1 \wedge \cdots \wedge B_m$ where each B_j is a reduction of A_j by some rule in the program using a renaming such that all rules are variable-disjoint.

A *breadth-first derivation (or BF-derivation)* from a state $\langle G_0 \mid c_0\rangle$ for program P is a sequence of states

$$\langle G_0 \mid c_0\rangle \Rightarrow_{BF(P)} \langle G_1 \mid c_1\rangle \Rightarrow_{BF(P)} \cdots \Rightarrow_{BF(P)} \langle G_i \mid c_i\rangle \Rightarrow_{BF(P)} \cdots$$

such that for each $i \geq 0$, there is a breadth-first derivation step from $\langle G_i \mid c_i\rangle$ to $\langle G_{i+1} \mid c_{i+1}\rangle$. When the program P is fixed we will use the notation $\Rightarrow_{BF}$ rather than $\Rightarrow_{BF(P)}$.

For our purposes we have defined the consistency check for breadth-first derivations in terms of satisfiability in the constraint theory. In effect the solver is restricted to be theory complete. However, one can also generalize this check to use an arbitrary constraint solver.

We extend the definition of answer, successful derivation, failed derivation, derivation tree and finite failure to the case of BF-derivations in the obvious way.

Example 3.8. Recall the factorial program and goal $fac(1, X)$ from Example 3.1. A successful BF-derivation from this goal is:

$$\begin{gathered}\langle fac(1,X) \mid true\rangle \\ \Downarrow_{BF} \\ \langle 1 = N, X = N \times F, N \geq 1, fac(N-1,F) \mid true\rangle \\ \Downarrow_{BF} \\ \langle N-1 = 0, F = 1 \mid 1 = N \wedge X = N \times F \wedge N \geq 1\rangle \\ \Downarrow_{BF} \\ \langle \Box \mid 1 = N \wedge X = N \times F \wedge N \geq 1 \wedge N-1 = 0 \wedge F = 1\rangle\end{gathered}$$

We now relate BF-derivations to the more standard operational definition. We can mimic the construction of a BF-derivation by choosing a literal selection strategy in which the "oldest" literals are selected first.

Definition 3.9. The *index* of a literal in a derivation is the tuple $\langle i, j\rangle$ where i is the index of the first state in the derivation in which the literal occurs and j is the index of its position in this state.

The *index-respecting* literal selection strategy is to always choose the literal with the smallest index where indices are ordered lexicographically.

Note that the index-respecting literal selection strategy is fair.

Definition 3.10. Let D be a derivation and D_{BF} a breadth-first derivation from the same state. Let D_{BF} be of the form

$$\langle G_0 \mid c_0\rangle \Rightarrow_{BF} \langle G_1 \mid c_1\rangle \Rightarrow_{BF} \cdots \Rightarrow_{BF} \langle G_i \mid c_i\rangle \Rightarrow_{BF} \cdots$$

D and D_{BF} *correspond* if D has the form

$$\langle G_0 \,\mathbf{I}\, c_0 \rangle \Rightarrow \cdots \Rightarrow \langle G_1 \,\mathbf{I}\, c_1 \rangle \Rightarrow \cdots \Rightarrow \langle G_i \,\mathbf{I}\, c_i \rangle \Rightarrow \cdots$$

and D and D_{BF} are both infinite or both have the same last state.

For instance the BF-derivation of Example 3.8 corresponds to the derivation of Example 3.1.

It is straightforward to show that:

Lemma 3.4. Let P be a $CLP(\mathcal{C})$ program and G a goal.

1. *Every finished derivation D from G for program P via the index-respecting literal selection strategy and using a theory complete solver has a corresponding breadth-first derivation D_{BF} from G for P.*
2. *Every breadth-first derivation D_{BF} from G for program P has a corresponding derivation D from a goal G via the index-respecting literal selection strategy and using a theory complete solver.* □

We can now relate BF-derivations to usual derivations. The result for successful derivations follows immediately from the above lemma and Theorem 3.2

Theorem 3.4. Let P be a $CLP(\mathcal{C})$ program and G a goal.

1. *For every successful derivation from G with satisfiable answer c, there is a successful BF-derivation which gives an answer which is a reordering of c.*
2. *For every successful BF-derivation from G with answer c and for any literal selection strategy $\mathcal{S}$ and constraint solver solv for $\mathcal{C}$ there is a successful derivation from G via $\mathcal{S}$ using solv that gives an answer which is a reordering of c.* □

The correspondence for finitely failed goals requires a little more justification.

Theorem 3.5. Let P be a program and G a goal. G finitely fails using BF-derivations iff there exists a well-behaved solver solv and selection strategy $\mathcal{S}$ such that G finitely fails using (usual) derivations.

PROOF. From Lemma 3.4, G finitely fails using BF-derivations iff G finitely fails with the index-respecting literal selection strategy when using a theory complete solver. We must now prove that if G finitely fails with some solver *solv* and some literal selection strategy, $\mathcal{S}$ say, then G finitely fails with the index-respecting literal selection strategy when using a theory complete solver. From Theorem 3.3 and since the index-respecting literal selection strategy is fair, if G finitely fails with $\mathcal{S}$ and with solver *solv* then G finitely fails with the index-respecting literal selection strategy when using *solv*. Thus from Lemma 3.2, G finitely fails with the index-respecting literal selection strategy when using a theory complete solver since this is more powerful than *solv*. □

4. THE SEMANTICS OF SUCCESS

In this section we give an algebraic and logical semantics for the answers to a CLP program and show that these semantics accord with the operational semantics.

4.1. Logical semantics

We first look at a logical semantics for a $CLP(\mathcal{C})$ program. We can view each rule in a CLP program, say

$$A \texttt{ :- } L_1, \ldots, L_n$$

as representing the formula

$$\tilde{\forall}(A \leftarrow L_1 \wedge \ldots \wedge L_n)$$

and the program is understood to represent the conjunction of its rules.

The *logical semantics* of a $CLP(\mathcal{C})$ program P is the theory obtained by adding the rules of P to a theory of the constraint domain $\mathcal{C}$.

The first result we need to show for any semantics is that the operational semantics is sound with respect to the semantics. For the logical semantics soundness means that any answer returned by the operational semantics, logically implies the initial goal. Thus the answer c to a goal G is logically read as: if c holds, then so does G.

Lemma 4.1. Let P be a $CLP(\mathcal{C})$program. If $\langle G \mid c\rangle$ is reduced to $\langle G' \mid c'\rangle$,

$$P, \mathcal{T_C} \models (G' \wedge c') \rightarrow (G \wedge c).$$

PROOF. Let G be of the form $L_1, \ldots, L_n$ where L_i is the selected literal. There are four cases to consider.

The first case is when L_i is a primitive constraint and $solv(c \wedge L_i) \neq false$. In this case G' is $L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$ and c' is $c \wedge L_i$. Thus $G' \wedge c'$ is $L_1 \wedge \cdots \wedge L_{i-1} \wedge L_{i+1} \wedge \cdots \wedge L_n \wedge c \wedge L_i$ which is logically equivalent to $G \wedge c$. Thus, $P, \mathcal{T_C} \models (G' \wedge c') \rightarrow (G \wedge c)$.

The second case is when L_i is a primitive constraint and $solv(c \wedge L_i) = false$. In this case G' is $\Box$ and c' is $false$. Trivially $P, \mathcal{T_C} \models (G' \wedge c') \rightarrow (G \wedge c)$ because $(G' \wedge c')$ is equivalent to $false$.

The third case is when L_i is a user defined constraint. Let L_i be of the form $p(s_1, \ldots, s_m)$. In this case, there is a renaming,

$$p(t_1, \ldots, t_n) \texttt{ :- } B$$

of a rule in P such that G' is $L_1, \ldots, L_{i-1}, s_1 = t_1, \ldots, s_m = t_m, B, L_{i+1}, \ldots, L_n$ and c' is c. Then, clearly

$$P \models B \rightarrow p(t_1, \ldots, t_n)$$

Hence, since $\mathcal{T_C}$ treats $=$ as identity,

$$\mathcal{T_C} \models s_1 = t_1, \ldots, s_m = t_m \rightarrow p(s_1, \ldots, s_n) \leftrightarrow p(t_1, \ldots, t_n)$$

and so from the above two statements

$$P, \mathcal{T_C} \models B \wedge s_1 = t_1, \ldots, s_m = t_m \rightarrow p(s_1, \ldots, s_n)$$

Hence from the above and since the remaining parts are unchanged.

$$P, \mathcal{T_C} \models (G' \wedge c') \rightarrow (G \wedge c)$$

The fourth case is when L_i is a user defined constraint for which $defn_P(L_i)$ is empty. In this case G' is $\Box$ and c' is $false$. As in the second case above, trivially $P, \mathcal{T_C} \models (G' \wedge c') \rightarrow (G \wedge c)$ because $(G' \wedge c')$ is equivalent to $false$. $\Box$

The above lemma straightforwardly gives us the soundness of success.

Theorem 4.1. (Logical Soundness of Success)
Let $\mathcal{T}_{\mathcal{C}}$ be a theory for constraint domain $\mathcal{C}$ and P be a $CLP(\mathcal{C})$ program. If goal G has answer c, then

$$P, \mathcal{T}_{\mathcal{C}} \models c \rightarrow G.$$

PROOF. Let c be the answer. Then there must be a finite derivation

$$\langle G_0 \mid c_0 \rangle \Rightarrow \ldots \Rightarrow \langle G_n \mid c_n \rangle$$

where G_0 is G, c_0 is $true$, G_n is $\Box$ and c is $\bar{\exists}_{vars(G)} c_n$. By repeated use of Lemma 4.1, we have that $P, \mathcal{T}_{\mathcal{C}} \models (G_n \wedge c_n) \rightarrow (G_0 \wedge c_0)$. Thus $P, \mathcal{T}_{\mathcal{C}} \models c_n \rightarrow G$ and so $P, \mathcal{T}_{\mathcal{C}} \models \bar{\exists}_{vars(G)} c_n \rightarrow G$. $\Box$

4.2. Algebraic semantics

We now turn our attention to the algebraic semantics. Such a semantics depends on us finding a model for the program which is the "intended" interpretation of the program. For logic programs this model is the least Herbrand model. In the context of constraint logic programs we must generalize this approach to take into account the intended interpretation of the primitive constraints. Clearly the intended interpretation of a CLP program should not change the interpretation of the primitive constraints or function symbols. All it can do is extend the intended interpretation so as to provide an interpretation for each user-defined predicate symbol in P.

Definition 4.1. A *$\mathcal{C}$-interpretation* for a $CLP(\mathcal{C})$ program P is an interpretation which agrees with $D_{\mathcal{C}}$ on the interpretation of the symbols in $\mathcal{C}$.

Since the meaning of the primitive constraints is fixed by $\mathcal{C}$, we may represent each $\mathcal{C}$-interpretation I simply by a subset of the *$\mathcal{C}$-base* of P, written *$\mathcal{C}$-base$_P$*, which is the set

$$\{p(d_1, \ldots, d_n) \mid \begin{array}{l} p \text{ is an } n\text{-ary user-defined predicate in } P \\ \text{and each } d_i \text{ is a domain element of } D_{\mathcal{C}} \end{array}\}.$$

Note that the set of all possible $\mathcal{C}$-interpretations for P is just the set of all subsets of *$\mathcal{C}$-base$_P$*, $\mathcal{P}$(*$\mathcal{C}$-base$_P$*). Also note that *$\mathcal{C}$-base$_P$* itself is the $\mathcal{C}$-interpretation in which each user-defined predicate is mapped to the set of all tuples, that is, in which everything is considered true.

The intended interpretation of a CLP program P will be a "$\mathcal{C}$-model" of P.

Definition 4.2. A *$\mathcal{C}$-model* of a $CLP(\mathcal{C})$ program P is a $\mathcal{C}$-interpretation which is a model of P.

Every program has a least $\mathcal{C}$-model which is usually regarded as the intended interpretation of the program since it is the most conservative $\mathcal{C}$-model. This result is analogous to that for logic programs in which the algebraic semantics of a logic program is given by its least Herbrand model. The proof of existence of the least model is essentially identical to that for logic programs. The proof makes use of the following obvious result:

Lemma 4.2. Let P be a $CLP(\mathcal{C})$ program, L a literal and M and M' be $\mathcal{C}$-models of P, where $M \subseteq M'$. Then for any valuation σ, $M \models_\sigma L$ implies $M' \models_\sigma L$. $\Box$

Theorem 4.2. (Model Intersection Property)
Let $\mathbf{M}$ be a set of $\mathcal{C}$-models of a $CLP(\mathcal{C})$ program P. Then $\bigcap \mathbf{M}$ is a $\mathcal{C}$-model of P.

PROOF. Suppose to the contrary $\bigcap \mathbf{M}$ is not a model of P. Then there exists a rule A :- $L_1, \ldots, L_n$ and valuation σ where $\bigcap \mathbf{M} \models_\sigma L_1 \wedge \cdots \wedge L_n$ but $\bigcap \mathbf{M} \not\models_\sigma A$. By n uses of Lemma 4.2 for each model $M \in \mathbf{M}$

$$M \models_\sigma L_1 \wedge \cdots \wedge L_n$$

and since M is a model of P, $M \models_\sigma A$. Hence $\sigma(A) \in M$ and hence $\sigma(A) \in \bigcap \mathbf{M}$, which is a contradiction. $\Box$
If we let $\mathbf{M}$ be the set of all $\mathcal{C}$-models of P in the above theorem we arrive at:

Corollary 4.1. Every $CLP(\mathcal{C})$ program has a least $\mathcal{C}$-model. $\Box$

Definition 4.3. We denote the least $\mathcal{C}$-model of a $CLP(\mathcal{C})$ program P by $lm(P, \mathcal{C})$.

Example 4.1. Recall the factorial program from Example 3.1

$fac(0, 1).$
$fac(N, N * F) :- N \geq 1,\ fac(N-1, F).$

It has an infinite number of *Real*-models, including

$$\{fac(n, n!) \mid n \in \{0, 1, 2, \ldots\}\} \cup \{fac(n, 0) \mid n \in \{0, 1, 2, \ldots\}\}.$$

and

$$\{fac(r, r') \mid r, r' \in \mathcal{R}\}.$$

As one might hope, the least *Real*-model is

$$\{fac(n, n!) \mid n \in \{0, 1, 2, \ldots\}\}.$$

As one would hope, if a goal is satisfiable in the least $\mathcal{C}$-model then it holds in all $\mathcal{C}$-models. Hence we have the following theorem:

Theorem 4.3. Let P be a $CLP(\mathcal{C})$ program, G a goal and σ a valuation. Then $P, \mathcal{D}_\mathcal{C} \models_\sigma G$ iff $lm(P, \mathcal{C}) \models_\sigma G$.

PROOF. The "if" direction follows from the fact that G is a conjunction of literals and Lemma 4.2 above. The "only if" direction follows from the argument behind Theorem 4.2. $\Box$

Corollary 4.2. Let P be a $CLP(\mathcal{C})$ program and G a goal. Then $P, \mathcal{D}_\mathcal{C} \models \tilde{\exists} G$ iff $lm(P, \mathcal{C}) \models \tilde{\exists} G$. $\Box$

The next theorem shows that the operational semantics is sound for the least model. This follows immediately from Theorem 4.1.

Theorem 4.4. (Algebraic Soundness of Success)
Let P be a $CLP(\mathcal{C})$ program. If goal G has answer c, then $lm(P,\mathcal{C}) \models c \rightarrow G$. $\square$

4.3. Fixpoint semantics

Soundness of the logical and algebraic semantics ensures that the operational semantics only returns answers which are solutions to the goal. However, we would also like to be sure that the operational semantics will return all solutions to the goal. This is called *completeness.*

To prove completeness it is necessary to introduce yet another semantics for CLP programs which bridges the gap between the algebraic and the operational semantics. This semantics, actually two semantics, are called fixpoint semantics and generalize the T_P semantics for logic programs.

The fixpoint semantics is based on the "immediate consequence operator" which maps the set of "facts" in a $\mathcal{C}$-interpretation to the set of facts which are implied by the rules in the program. In a sense, this operator captures the Modus Ponens rule of inference. The T_P^{Term} operator is due to van Emden and Kowalski [4] (who called it T). Apt and van Emden [1] later used the name T_P which has become standard.

Definition 4.4. Let P be a $CLP(\mathcal{C})$ program. The *immediate consequence function* for P is the function $T_P^{\mathcal{C}} : \mathcal{P}(\mathcal{C}\text{-}base_P) \rightarrow \mathcal{P}(\mathcal{C}\text{-}base_P)$. Let I be a $\mathcal{C}$-interpretation, and let σ range over valuations for $\mathcal{C}$. Then $T_P^{\mathcal{C}}(I)$ is defined as

$$\{\sigma(A) \mid A \texttt{ :- } L_1, \ldots, L_n \text{ is a rule in } P \text{ for which } I \models_\sigma L_1 \wedge \ldots \wedge L_n\}$$

This is quite a compact definition. It is best understood by noting that

$$I \models_\sigma p_1(\vec{t_1}) \wedge \cdots \wedge p_1(\vec{t_n})$$

iff for each literal $p_i(\vec{t_i})$ either p_i is a primitive constraint and $\mathcal{D}_\mathcal{C} \models_\sigma p_i(\vec{t_i})$ or p_i is a user-defined predicate and $p_i(\sigma(\vec{t_i})) \in I$.

Note that $\mathcal{P}(\mathcal{C}\text{-}base_P)$ is a complete lattice ordered by the subset relation on $\mathcal{C}$-interpretations (viewed as sets). It is not too hard to show [1] that:

Theorem 4.5. Let P be a $CLP(\mathcal{C})$ program. Then $T_P^{\mathcal{C}}$ is continuous. $\square$

Recall the definition of the *ordinal powers* of a function F over a complete lattice X:

$$F \uparrow \alpha = \begin{cases} \bigsqcup\{F \uparrow \alpha' \mid \alpha' < \alpha\} & \text{if } \alpha \text{ is a limit ordinal} \\ F\ (F \uparrow (\alpha - 1)) & \text{if } \alpha \text{ is a successor ordinal} \end{cases}$$

and dually,

$$F \downarrow \alpha = \begin{cases} \bigsqcap\{F \downarrow \alpha' \mid \alpha' < \alpha\} & \text{if } \alpha \text{ is a limit ordinal} \\ F\ (F \downarrow (\alpha - 1)) & \text{if } \alpha \text{ is a successor ordinal} \end{cases}$$

Since the first limit ordinal is 0, it follows that in particular, $F \uparrow 0 = \bot_X$ (the bottom element of the lattice X) and $F \downarrow 0 = \top_X$ (the top element).

From Kleene's fixpoint theorem we know that the the least fixpoint of any continuous operator is reached at the first infinite ordinal ω. Hence, *Corollary 4.3.* $\mathit{lfp}(T_P^{\mathcal{C}}) = T_P^{\mathcal{C}} \uparrow \omega$. $\square$

Example 4.2. Let P be the factorial program from Example 4.1. Then

$$\begin{array}{lll}
T_P^{Real} \uparrow 0 = \bot & & = \emptyset \\
T_P^{Real} \uparrow 1 = T_P^{Real}(T_P^{Real} \uparrow 0) & & = \{fac(0,1)\} \\
T_P^{Real} \uparrow 2 = T_P^{Real}(T_P^{Real} \uparrow 1) & & = \{fac(0,1), fac(1,1)\} \\
T_P^{Real} \uparrow 3 = T_P^{Real}(T_P^{Real} \uparrow 2) & & = \{fac(0,1), fac(1,1), fac(2,2)\} \\
\qquad\qquad \vdots & & \\
T_P^{Real} \uparrow k = T_P^{Real}(T_P^{Real} \uparrow (k-1)) & & = \{fac(n,n!) \mid n \in \{0,1,2,\ldots,k-1\}\} \\
\qquad\qquad \vdots & & \\
T_P^{Real} \uparrow \omega = \bigcup_{k \geq 0} T_P^{Real} \uparrow k & & = \{fac(n,n!) \mid n \in \{0,1,2,\ldots,\}\}.
\end{array}$$

Thus $\mathit{lfp}(T_P^{Real}) = \{fac(n,n!) \mid n \in \{0,1,2,\ldots\}\}$. It also useful to consider the greatest fixpoint of $T_P^{\mathcal{C}}$. We have that

$$\begin{array}{lll}
T_P^{Real} \downarrow 0 & = Real\text{-base}_P & = \{fac(r,r') \mid r,r' \in \mathcal{R}\} \\
T_P^{Real} \downarrow 1 & = T_P^{Real}(T_P^{Real} \downarrow 0) & = \{fac(0,1)\} \cup \{fac(r,r') \mid r \geq 1 \text{ and } r,r' \in \mathcal{R}\} \\
T_P^{Real} \downarrow 2 & = T_P^{Real}(T_P^{Real} \downarrow 1) & = \{fac(0,1), fac(1,1)\} \cup \\
 & & \quad \{fac(r,r') \mid r \geq 2 \text{ and } r,r' \in \mathcal{R}\} \\
 & \quad \vdots & \\
T_P^{Real} \downarrow k & = T_P^{Real}(T_P^{Real} \downarrow (k-1)) & = \{fac(n,n!) \mid n \in \{0,1,2,\ldots,k-1\}\} \\
 & & \quad \cup \ \{fac(r,r') \mid r \geq k \text{ and } r,r' \in \mathcal{R}\} \\
 & \quad \vdots & \\
T_P^{Real} \downarrow \omega & = \bigcap_{k \geq 0} T_P^{Real} \downarrow k & = \{fac(n,n!) \mid n \in \{0,1,2,\ldots\}\} \\
T_P^{Real} \downarrow \omega + 1 & = T_P^{Real}(T_P^{Real} \downarrow \omega) & = \{fac(n,n!) \mid n \in \{0,1,2,\ldots\}\}.
\end{array}$$

Thus $\mathit{gfp}(T_P^{Real}) = \{fac(n,n!) \mid n \in \{0,1,2,\ldots\}\}$. As this is the same as the least fixpoint, this is the unique fixpoint of the program P defining the fac predicate.

In general, the immediate consequence function of a program may have many fixpoints, and the greatest fixpoint may not be reached by step ω in the descending Kleene sequence. This is also the case for logic programs.

Example 4.3. Consider the $CLP(Term)$ program P:

$$\begin{array}{lll}
q(a) & \text{:-} & p(X) \\
p(f(X)) & \text{:-} & p(X)
\end{array}$$

The downward powers of T_P^{Term} are

$$\begin{array}{lll}
T_P^{Term} \downarrow 0 & = Term\text{-base}_P & = \{q(r) \mid r = f^i(a), 0 \leq i\} \cup \{p(r) \mid r = f^i(a), 0 \leq i\} \\
T_P^{Term} \downarrow 1 & = T_P^{Term}(T_P^{Term} \downarrow 0) & = \{q(a)\} \cup \{p(r) \mid r = f^i(a), 1 \leq i\} \\
 & \quad \vdots & \\
T_P^{Term} \downarrow k & = T_P^{Term}(T_P^{Term} \downarrow (k-1)) & = \{q(a)\} \cup \{p(r) \mid r = f^i(a), k \leq i\} \\
 & \quad \vdots & \\
T_P^{Term} \downarrow \omega & = & = \{q(a)\} \\
T_P^{Term} \downarrow \omega + 1 & = T_P^{Term}(T_P^{Term} \downarrow \omega) & = \emptyset.
\end{array}$$

The greatest fixpoint of T_P^{Term} is $T_P^{Term} \downarrow \omega + 1$.

There is a simple relationship between the $\mathcal{C}$-models of a program and the $T_P^{\mathcal{C}}$ operator: the $\mathcal{C}$ models are exactly the pre-fixpoints of $T_P^{\mathcal{C}}$. The following result for the $Term$ constraint domain was proven in [4], the proof below is essentially identical.

Lemma 4.3. Let P be a $CLP(\mathcal{C})$ program. Then M is a $\mathcal{C}$-model of P iff M is a pre-fixpoint of $T_P^{\mathcal{C}}$, that is $T_P^{\mathcal{C}}(M) \subseteq M$.

PROOF. Now M is a $\mathcal{C}$-model of P iff for each rule A :- $L_1, ..., L_n$ in P, $M \models \tilde{\forall} A \leftarrow L_1 \wedge \cdots \wedge L_n$. Thus, M is a $\mathcal{C}$-model of P iff for each rule A :- $L_1, ..., L_n$ in P and for each valuation σ, $M \models_\sigma A \leftarrow L_1 \wedge \cdots \wedge L_n$. Thus, M is a $\mathcal{C}$-model of P iff for each rule A :- $L_1, ..., L_n$ in P and for each valuation σ, if $M \models_\sigma L_1 \wedge \cdots \wedge L_n$ then $M \models_\sigma A$. Hence by the definition of $T_P^{\mathcal{C}}$, M is a $\mathcal{C}$-model of P iff $T_P^{\mathcal{C}}(M) \subseteq M$. $\square$

Given this relationship, it is straightforward to show that the least model of a program P is also the least fixpoint of $T_P^{\mathcal{C}}$. This will (eventually) allow us to relate the algebraic semantics to the fixpoint semantics and to the operational semantics.

Theorem 4.6. Let P be a $CLP(\mathcal{C})$ program. Then $lm(P, \mathcal{C}) = lfp(T_P^{\mathcal{C}}) = T_P^{\mathcal{C}} \uparrow \omega$.

PROOF.

$$\begin{aligned} lm(P,\mathcal{C}) &= \textstyle\bigsqcap\{M | M \text{ is a } \mathcal{C}\text{-model of } P\} \\ &= \textstyle\bigsqcap\{M | M \text{ is a pre-fixpoint of } P\} \text{ From Lemma 4.3} \\ &= lfp(T_P^{\mathcal{C}}) \text{ By the Knaster-Tarski Fixpoint Theorem [15].} \end{aligned}$$

It follows from Corollary 4.3 that $lfp(T_P^{\mathcal{C}}) = T_P^{\mathcal{C}} \uparrow \omega$. $\square$

We now introduce another fixpoint semantics which is a modification of the immediate consequence function which works on the syntactic level of constraints rather than the semantic level of valuations. It will be used to bridge the gap between the immediate consequence function and the operational semantics. It works on "facts" which are CLP rules in which the body contains only a single existentially quantified constraint.

Definition 4.5. A *fact* is of the form A :- c where A is an atom and c is a constraint.

Definition 4.6. Given a rule R of the form A :- G, and a set of facts F, we can define an *immediate consequence* of F using R, as the fact

A :- c

where there exists a successful BF-derivation

$$\langle G \mid true \rangle \Rightarrow^*_{BF(F)} \langle \square \mid c \rangle$$

That is, there is a breadth-first derivation for G using the set of facts F as the program, that has last state $\langle \square \mid c \rangle$.

Note that because of the form of F any BF derivation can be at most two steps long, because the bodies of rules in F do not contain atoms. For example if c_0 is the conjunction of primitive constraints appearing in G a derivation for G has the form

$$\langle G \,\mathbf{|}\, true\rangle \Rightarrow_{BF(F)} \langle c_1 \,\mathbf{|}\, c_0\rangle \Rightarrow_{BF(F)} \langle \Box \,\mathbf{|}\, c_0 \wedge c_1\rangle.$$

Let $S_{\{R\}}(F)$ denote the set of all immediate consequences of F using R.

The *immediate consequences* of a set F of facts using a program P, denoted $S_P(F)$, is defined by

$$S_P(F) = \bigcup_{R \in P} S_{\{R\}}(F)$$

The function S_P was introduced by Gabrielli and Levi [6], inspired by related functions defined in [8] and the S-semantics [5]. We are using a different, though equivalent, formulation than [6].

Example 4.4. Let $F_1 = \{fac(0,1)\}$, and $R = (fac(N, N*F)$:- $N \geq 1, fac(N-1, F))$. There is a single successful BF derivation

$$\begin{aligned} \langle N \geq 1, fac(N-1,F) \,\mathbf{|}\, true\rangle &\Rightarrow_{BF(F_1)} \langle N-1=0, F=1 \,\mathbf{|}\, N \geq 1\rangle \\ &\Rightarrow_{BF(F_1)} \langle \Box \,\mathbf{|}\, N \geq 1 \wedge N-1=0 \wedge F=1\rangle \end{aligned}$$

Hence $S_{\{R\}}(F_1)$ is $\{fac(N, N*F)$:- $N \geq 1, N-1=0, F=1\}$.

Let P be the factorial program from Example 3.1 . Since S_P is a map over a complete lattice, the set of all facts for predicates defined in the original program, the ordinal powers of S_P can be defined in the usual way. Then

$$\begin{aligned} S_P \uparrow 0 &= \{\} \\ S_P(\emptyset) = S_P \uparrow 1 &= \{fac(0,1)\} \\ S_P(\{fac(0,1)\}) = S_P \uparrow 2 &= \{fac(0,1), (fac(N,N*F)\text{:-}N \geq 1, N-1=0, F=1.)\} \\ S_P \uparrow 3 &= \{fac(0,1), (fac(N,N*F)\text{:-}N \geq 1, N-1=0, F=1.), \\ &\quad (fac(N,N*F)\text{:-}N \geq 1,\ N-1=N', \\ &\qquad F=N'*F',\ N' \geq 1, \\ &\qquad N'-1=0,\ F'=1.\ ) \qquad \} \\ &\ \vdots \end{aligned}$$

As one would expect, the S_P operator is also continuous. The proof is analogous to the proof for T_P.

Theorem 4.7. Let P be a $CLP(\mathcal{C})$ program. Then S_P is continuous. $\Box$

Corollary 4.4. Let P be a $CLP(\mathcal{C})$ program. Then

$$lfp(S_P) = S_P \uparrow \omega = \cup_{i=0}^{\infty} S_P \uparrow i. \quad \Box$$

As one would expect, there is a very strong relationship between both fixpoint semantics. To formalize this correspondence, we need to translate facts to elements in the $\mathcal{C}$-base. This is done by means of "grounding":

Definition 4.7. Let $\mathcal{C}$ be a constraint domain. Let F be the fact A :- c. We define

$$[F]_{\mathcal{C}} = \{\sigma(A) \mid \mathcal{D}_{\mathcal{C}} \models_\sigma c\}.$$

This is lifted to sets of facts in the obvious way: let S be a set of facts, then

$$[S]_{\mathcal{C}} = \bigcup\{[F]_{\mathcal{C}} \mid F \in S\}.$$

For example,

$$[p(X,Y) \texttt{ :- } X = Y]_{Real} = \{p(r,r) \mid r \in \mathcal{R}\}.$$

and

$$[fac(N, N*F) \texttt{ :- } N \geq 1, N-1 = N', F = N' * F', N' \geq 1, N'-1 = 0, F' = 1]_{Real} \\ = \{fac(2,2)\}$$

Clearly variable names do not affect grounding, hence:

Lemma 4.4. Let ρ be a renaming and F a fact. Then $[F]_{\mathcal{C}} = [\rho(F)]_{\mathcal{C}}$. □

Now we can show how the application of S_P and $T_P^{\mathcal{C}}$ correspond.

Lemma 4.5. Let P be a $CLP(\mathcal{C})$ program and F a set of facts. Then,

$$[S_P(F)]_{\mathcal{C}} = T_P^{\mathcal{C}}([F]_{\mathcal{C}}).$$

Proof. We first show that $T_P^{\mathcal{C}}([F]_{\mathcal{C}}) \subseteq [S_P(F)]_{\mathcal{C}}$.

Now, if $y \in T_P^{\mathcal{C}}([F]_{\mathcal{C}})$, there is a rule $A \texttt{ :- } G$ in P and a valuation σ such that y is $\sigma(A)$ and

$$[F]_{\mathcal{C}} \models_\sigma G. \tag{4.1}$$

Let G contain atoms $p_1(\vec{s}_1), ..., p_n(\vec{s}_n)$ and let c' be the conjunction of primitive constraints which appear in G. From (4.1),

$$\mathcal{D}_{\mathcal{C}} \models_\sigma c' \tag{4.2}$$

and for each $p_i(\vec{s}_i)$ there is a fact $(p_i(\vec{t}_i) \texttt{ :- } c_i)$ in F, such that $\sigma(p_i(\vec{s}_i)) \in [(p_i(\vec{t}_i) \texttt{ :- } c_i)]_{\mathcal{C}}$. From Lemma 4.4, we can assume that these facts have been renamed so that the variables in each $p_i(\vec{t}_i) \texttt{ :- } c_i$ are disjoint from each other and from those in $A \texttt{ :- } G$.

Now $\sigma(p_i(\vec{s}_i)) \in [p_i(\vec{t}_i) \texttt{ :- } c_i]_{\mathcal{C}}$ implies that there is a valuation σ_i such that $\sigma(p_i(\vec{s}_i)) = \sigma_i(p_i(\vec{t}_i))$ and $\mathcal{D}_{\mathcal{C}} \models_{\sigma_i} c_i$. From the disjointedness assumption, the valuation σ' defined by

$$\sigma'(x) = \begin{cases} \sigma_i(x) \text{ when } x \in vars(p_i(\vec{t}_i) \texttt{ :- } c_i) \\ \sigma(x) \text{ otherwise.} \end{cases}$$

is well defined. Furthermore, for each i,

$$\mathcal{D}_{\mathcal{C}} \models_{\sigma'} \vec{s}_i = \vec{t}_i \wedge c_i$$

and from (4.2),

$$\mathcal{D}_{\mathcal{C}} \models_{\sigma'} c'.$$

Let c be the constraint

$$c' \wedge \vec{s}_1 = \vec{t}_1 \wedge c_1 \wedge \cdots \wedge \vec{s}_n = \vec{t}_n \wedge c_n.$$

Then c is satisfiable, because $\mathcal{D}_\mathcal{C} \models_{\sigma'} c$.

By construction, there is a BF derivation using the program F:

$$\begin{aligned}\langle G \mid true\rangle &\Rightarrow_{BF(F)} \langle \vec{s}_1 = \vec{t}_1 \wedge c_1 \wedge \cdots \wedge \vec{s}_n = \vec{t}_n \wedge c_n \mid c'\rangle \\ &\Rightarrow_{BF(F)} \langle \Box \mid c' \wedge \vec{s}_1 = \vec{t}_1 \wedge c_1 \wedge \cdots \wedge \vec{s}_n = \vec{t}_n \wedge c_n\rangle\end{aligned}$$

Hence $(A \ \texttt{:-}\ \exists_{vars(A)} c) \in S_P(F)$. By construction $\sigma'(A) \in [A \ \texttt{:-}\ \exists_{vars(A)} c]_\mathcal{C}$. But $\sigma'(A) = \sigma(A) = y$, so $y \in [S_P(F)]_\mathcal{C}$.

We must now show that $[S_P(F)]_\mathcal{C} \subseteq T_P^\mathcal{C}([F]_\mathcal{C})$. This can be done by reversing the implications in the above proof. $\Box$

Theorem 4.8. Let P be a $CLP(\mathcal{C})$ program. Then,

$$[lfp(S_P)]_\mathcal{C} = lfp(T_P^\mathcal{C}).$$

PROOF. We first prove by transfinite induction that for all ordinals α,

$$[S_P \uparrow \alpha]_\mathcal{C} = T_P^\mathcal{C} \uparrow \alpha.$$

There are two cases to consider. The first is when β is a successor ordinal. We have that

$$\begin{aligned}T_P^\mathcal{C} \uparrow \beta &= T_P^\mathcal{C}(T_P^\mathcal{C} \uparrow \beta - 1) && \text{By definition of the ordinal power}\\ &= T_P^\mathcal{C}([S_P \uparrow \beta - 1]_\mathcal{C}) && \text{By assumption}\\ &= [S_P(S_P \uparrow \beta - 1)]_\mathcal{C} && \text{From Lemma 4.5}\\ &= [S_P \uparrow \beta]_\mathcal{C} && \text{By definition of the ordinal power.}\end{aligned}$$

The second case is when β is a limit ordinal. We have that

$$\begin{aligned}T_P^\mathcal{C} \uparrow \beta &= \bigcup\{T_P^\mathcal{C} \uparrow \gamma | \gamma < \alpha\} && \text{By definition of the ordinal power}\\ &= \bigcup\{[S_P \uparrow \gamma]_\mathcal{C} | \gamma < \alpha\} && \text{By assumption}\\ &= [\bigcup\{S_P \uparrow \gamma | \gamma < \alpha\}]_\mathcal{C} && \text{From definition of grounding}\\ &= [S_P \uparrow \beta]_\mathcal{C} && \text{By definition of the ordinal power.}\end{aligned}$$

Thus, by transfinite induction, for all ordinals α,

$$[S_P \uparrow \alpha]_\mathcal{C} = T_P^\mathcal{C} \uparrow \alpha.$$

It follows from Corollaries 4.3 and 4.4 that $[lfp(S_P)]_\mathcal{C} = lfp(T_P^\mathcal{C})$. $\Box$

4.4. Correspondence between fixpoint and operational semantics

At first sight the two fixpoint semantics are quite different from the operational semantics, but in fact the ordinal powers of the S_P operator are strongly related to BF-derivations, as shown in the following lemma. Recall that BF-derivations are defined with respect to the theory, or, equivalently, they always make use of a theory-complete solver.

Lemma 4.6. For a $CLP(\mathcal{C})$ program P and goal G, there is a successful BF derivation of length less than or equal to $n+1$ for state $\langle G_0 \mid c_0\rangle$ in P with answer c iff there is a successful BF derivation for $\langle G_0 \mid c_0\rangle$ in $S_P \uparrow n$ with answer c' such that $\mathcal{T}_\mathcal{C} \models \bar{\exists}_{vars(G_0,c_0)} c \leftrightarrow \bar{\exists}_{vars(G_0,c_0)} c'$.

PROOF. We give the "then" direction, the "if" direction is proved analogously.

The proof is by induction on n. For the base case, the only one step successful BF derivations are where G_0 is entirely made up of constraints. In this case the derivation

$$\langle G_0 \mid c_0 \rangle \Rightarrow_{BF(Q)} \langle \Box \mid c_0 \wedge G_0 \rangle$$

exists regardless of the program Q, and clearly the same derivation is a successful derivation in the empty program $S_P \uparrow 0$.

Consider a successful BF derivation in P of the form

$$\langle G_0 \mid c_0 \rangle \Rightarrow_{BF(P)} \langle G_1 \mid c_1 \rangle \Rightarrow_{BF(P)} \cdots \Rightarrow_{BF(P)} \langle \Box \mid c_{n+1} \rangle$$

Consider the BF derivation step

$$\langle G_0 \mid c_0 \rangle \Rightarrow_{BF(P)} \langle G_1 \mid c_1 \rangle$$

Then c_1 is $c_0 \wedge c_0'$ where c_0' are the constraints in G_0. Let $p_i(\vec{s}_i), 1 \leq i \leq n$ be the atoms in G_0.

The derivation step uses renamed apart program rules $p_i(\vec{t}_i)$ `:-` B_i for each atom $p_i(\vec{s}_i)$ to obtain

$$G_1 \ \equiv \ \vec{s}_1 = \vec{t}_1, B_1, \vec{s}_2 = \vec{t}_2, B_2, \ldots, \vec{s}_n = \vec{t}_n, B_n$$

Let V_1 be $vars(\langle G_1 \mid c_1 \rangle)$. By the induction hypothesis there is a successful BF derivation for $\langle G_1 \mid c_1 \rangle$ with final state $\langle \Box \mid x' \rangle$ where $\mathcal{T}_{\mathcal{C}} \models \bar{\exists}_{V_1} c_{n+1} \leftrightarrow \bar{\exists}_{V_1} x'$. It must take the form

$$\langle G_1 \mid c_0 \wedge c_0' \rangle \Rightarrow_{BF(S_P \uparrow (n-1))} \langle x \mid c_0 \wedge c_0' \wedge c_1' \rangle \Rightarrow_{BF(S_P \uparrow (n-1))} \langle \Box \mid x' \rangle$$

where c_1' is the constraints in G_1, x is the constraints that result from replacing the atoms in G and x' is $c_0 \wedge c_0' \wedge c_1' \wedge x$.

Let x_i be the constraints in B_i. Then c_1' is $\vec{s}_1 = \vec{t}_1 \wedge x_1 \wedge \cdots \wedge \vec{s}_n = \vec{t}_n \wedge x_n$.

Let $q_{ij}(\vec{u}_{ij}), 1 \leq j \leq m_i$ be the atoms in B_i. For each q_{ij} there exists a renamed apart copy of a fact in $S_P \uparrow (n-1)$,

$$q_{ij}(\vec{v}_{ij}) \ \texttt{:-} \ B_{ij}$$

used in the BF derivation step. Hence

$$x \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m_i} \vec{u}_{ij} = \vec{v}_{ij} \wedge B_{ij}$$

Because x' is satisfiable, each of the constraints in the following BF-derivations are satisfiable. We have a successful BF-derivation for each B_i.

$$\begin{aligned} \langle B_i \mid true \rangle &\Rightarrow_{BF(S_P \uparrow (n-1))} \langle \bigwedge_{j=1}^{m_i} \vec{u}_{ij} = \vec{v}_{ij} \wedge B_{ij} \mid x_i \rangle \\ &\Rightarrow_{BF(S_P \uparrow (n-1))} \langle \Box \mid x_i \wedge \bigwedge_{j=1}^{m_i} \vec{u}_{ij} = \vec{v}_{ij} \wedge B_{ij} \rangle \end{aligned}$$

Let C_i be $\bar{\exists}_{vars(\vec{t}_i)} x_i \wedge \bigwedge_{j=1}^{m_i} \vec{u}_{ij} = \vec{v}_{ij} \wedge B_{ij}$. Hence an (appropriately renamed) copy of each of the facts

$$p_i(\vec{t}_i) \ \texttt{:-} \ C_i$$

exists in $S_P \uparrow n$ by the definition of S_P.

We can now construct a successful derivation for $\langle G_0 \mathbin{\rule{0.1em}{0.6em}} c_0\rangle$ in $S_P \uparrow n$, using some renamed apart versions of the above facts $\rho_i(p_i(\vec{t_i})$:- $C_i)$, to rewrite each atom $p_i(\vec{s_i})$.

$$\langle G_0 \mathbin{\rule{0.1em}{0.6em}} c_0\rangle \Rightarrow_{BF(S_P\uparrow n)} \langle \vec{s}_1 = \rho_1(\vec{t}_1), \rho_1(C_1), \ldots, \vec{s}_n = \rho_n(\vec{t}_n), \rho_n(C_n) \mathbin{\rule{0.1em}{0.6em}} c_0 \wedge c_0'\rangle$$
$$\Rightarrow_{BF(S_P\uparrow n)} \langle \Box \mathbin{\rule{0.1em}{0.6em}} c_0 \wedge c_0' \wedge \vec{s}_1 = \rho_1(\vec{t}_1) \wedge \rho_1(C_1) \wedge \cdots \wedge \vec{s}_n = \rho_n(\vec{t}_n) \wedge \rho_n(C_n)\rangle$$

Let c' be $c_0 \wedge c_0' \wedge \vec{s}_1 = \rho_1(\vec{t}_1) \wedge \rho_1(C_1) \wedge \cdots \wedge \vec{s}_n = \rho_n(\vec{t}_n) \wedge \rho_n(C_n)$ and let $V_0 = vars(\langle G_0 \mathbin{\rule{0.1em}{0.6em}} c_0\rangle)$. Then

$\bar{\exists}_{V_0} c' \leftrightarrow c_0' \wedge c_0 \wedge \bar{\exists}_{V_0} \bigwedge_{i=1}^n \vec{s}_i = \rho_i(\vec{t}_i) \wedge \rho_i(C_1)$
since c' and c_0' only involve variables in V_0
$\leftrightarrow c_0 \wedge c_0' \wedge \bigwedge_{i=1}^n (\bar{\exists}_{V_0} \vec{s}_i = \rho_i(\vec{t}_i) \wedge \rho_i(C_i))$
since each expression $\rho_i(\vec{t}_i) \wedge \rho_i(C_i)$ does not share variables
$\leftrightarrow c_0 \wedge c_0' \wedge \bigwedge_{i=1}^n (\bar{\exists}_{V_0} \vec{s}_i = \vec{t}_i \wedge C_i)$
since variables in $\vec{t}_i$ and C_i do not intersect those in G_0 and c_0
$\leftrightarrow c_0 \wedge c_0' \wedge \bigwedge_{i=1}^n (\bar{\exists}_{V_0} \vec{s}_i = \vec{t}_i \wedge x_i \wedge \bigwedge_{j=1}^{m_i} \vec{u}_{ij} = \vec{v}_{ij} \wedge B_{ij})$
by definition of C_i
$\leftrightarrow \bar{\exists}_{V_0} c_0 \wedge c_0' \wedge \bigwedge_{i=1}^n (\vec{s}_i = \vec{t}_i \wedge x_i \wedge \bigwedge_{j=1}^{m_i} \vec{u}_{ij} = \vec{v}_{ij} \wedge B_{ij})$
since by construction the terms do not share variables
$\leftrightarrow \bar{\exists}_{V_0} c_0 \wedge c_0' \wedge \vec{s}_1 = \vec{t}_1 \wedge \cdots \vec{s}_n = \vec{t}_n \wedge x_1 \wedge \cdots \wedge x_n \wedge \bigwedge_{i=1}^n \bigwedge_{j=1}^{m_i} \vec{u}_{ij} = \vec{v}_{ij} \wedge B_{ij}$
rearranging terms
$\leftrightarrow \bar{\exists}_{V_0} c_0 \wedge c_0' \wedge c_1' \wedge x$ by the definition of c_1' and x
$\leftrightarrow \bar{\exists}_{V_0} x'$ by definition of x'
$\leftrightarrow \bar{\exists}_{V_0} c_{n+1}$ because $V_0 \subseteq V_1$ and $\bar{\exists}_{V_1} c_{n+1} \leftrightarrow \bar{\exists}_{V_1} x'$

This completes the proof of the induction step. $\Box$

Using the above lemma and Corollary 4.4 it is easy to show the following:

Lemma 4.7. Let P be a $CLP(\mathcal{C})$ program. Goal G has an successful BF derivation with answer c for program P iff there exists some integer n such that G has a successful BF-derivation for program $S_P \uparrow n$ with answer c' such that $\mathcal{T}_{\mathcal{C}} \models c \leftrightarrow c'$. $\Box$

Now we are in a position to relate the S_P operator to the standard top-down semantics.

Theorem 4.9. Let P be a $CLP(\mathcal{C})$ program. Goal G has an answer c for program P iff G has a successful derivation for program $lfp(S_P)$ with answer c' such that $\mathcal{T}_{\mathcal{C}} \models c \leftrightarrow c'$.

PROOF. Since a successful BF-derivation is finite, G has a successful BF-derivation for program $lfp(S_P)$ iff there exists some integer n such that G has a successful BF-derivation for program $S_P \uparrow n$. Using this observation, the result is an immediate consequence of Theorem 3.4 and Lemma 4.7. $\Box$

The results in this subsection were first presented in [6].

4.5. Completeness

We are now in a position to prove that the operational semantics is complete for the algebraic semantics.

Theorem 4.10. (Algebraic Completeness of Success)
Let P be a $CLP(\mathcal{C})$ program, G be a goal and θ a valuation. If

$$lm(P, \mathcal{C}) \models_\theta G.$$

then G has an answer c such that $\mathcal{D}_\mathcal{C} \models_\theta c$.

PROOF. If

$$lm(P, \mathcal{C}) \models_\theta G, \tag{4.3}$$

then, from Theorem 4.6,

$$lfp(T_P^\mathcal{C}) \models_\theta G.$$

From Theorem 4.8,

$$[lfp(S_P)]_\mathcal{C} \models_\theta G.$$

Let c_0 be the conjunction of constraints in G, and $p_i(\vec{s}_i), 1 \leq i \leq n$ be the atoms in G. For each $p_i(\vec{s}_i)$ there exist renamed apart versions of facts in $lfp(S_P)$ $(p_i(\vec{t}_i)\ \texttt{:-}\ c_i)$ and valuations θ_i such that $\theta(p_i(\vec{s}_i)) = \theta_i(p_i(\vec{t}_i))$ and $\mathcal{D}_\mathcal{C} \models_{\theta_i} c_i$. From the definition of $lfp(S_P)$ there also exists k such that each $(p_i(\vec{t}_i)\ \texttt{:-}\ c_i)$ is in $S_P \uparrow k$. From the disjointedness assumption, the valuation θ' defined by

$$\theta'(x) = \begin{cases} \theta_i(x) \text{ when } x \in vars(p_i(\vec{t}_i)\ \texttt{:-}\ c_i) \\ \theta(x) \text{ otherwise.} \end{cases}$$

is well defined. Furthermore, for each i,

$$\mathcal{D}_\mathcal{C} \models_{\theta'} \vec{s}_i = \vec{t}_i \wedge c'_i$$

and from (4.3),

$$\mathcal{D}_\mathcal{C} \models_{\theta'} c_0.$$

Let $c' \equiv c_0 \wedge \vec{s}_1 = \vec{t}_1 \wedge c_1 \wedge \cdots \wedge \vec{s}_n = \vec{t}_n \wedge c_n$. Then

$$\mathcal{D}_\mathcal{C} \models_{\theta'} c'.$$

Hence there is a successful BF derivation for program $S_P \uparrow k$:

$$\langle G \mid true \rangle \Rightarrow_{BF(S_P \uparrow k))} \langle \vec{s}_1 = \vec{t}_1 \wedge c_1 \wedge \cdots \wedge \vec{s}_n = \vec{t}_n \wedge c_n \mid c_0 \rangle \Rightarrow_{BF(S_P \uparrow k)} \langle \Box \mid c' \rangle$$

By Theorem 4.9 there exists a successful derivation for G in P with answer c such that $\mathcal{T}_\mathcal{C} \models \bar{\exists}_{vars(G)} c' \leftrightarrow c$. Hence, since $\mathcal{D}_\mathcal{C}$ models $\mathcal{T}_\mathcal{C}$,

$$\mathcal{D}_\mathcal{C} \models_{\theta'} c$$

and since θ and θ' are the same on the variables of G, $\mathcal{D}_\mathcal{C} \models_\theta c$. $\Box$

We can rephrase Theorem 4.4 and Theorem 4.10 to succinctly capture that the solutions to the goal in the minimal model are exactly the solutions to the constraints the operational semantics returns as goals. The "if" direction follows from Theorem 4.10 and the "only if" from Theorem 4.4.

Theorem 4.11. Let P be a $CLP(\mathcal{C})$ program and G be a goal with answers $c_1, c_2, \ldots$. Then

$$lm(P, \mathcal{C}) \models G \leftrightarrow \bigvee_{i=1}^{\infty} c_i. \quad \Box$$

The second result we need to show is that the operational semantics is complete with respect to the logical semantics. For the logical semantics, completeness is understood as that the answers returned by the operational semantics cover all of the constraints which imply the goal.

Theorem 4.12. (Logical Completeness of Success)
Let $\mathcal{T}_\mathcal{C}$ be a theory for constraint domain $\mathcal{C}$ and P be a $CLP(\mathcal{C})$ program. Let G be a goal and c a constraint. If $P, \mathcal{T}_\mathcal{C} \models c \rightarrow G$ then G has answers $c_1, \ldots, c_n$ such that

$$\mathcal{T}_\mathcal{C} \models c \rightarrow (c_1 \vee \ldots \vee c_n).$$

PROOF. We first prove that if

$$P, \mathcal{T}_\mathcal{C} \models c \rightarrow G.$$

then

$$\mathcal{T}_\mathcal{C} \models c \rightarrow \bigvee_{i=1}^{\infty} c_i.$$

where $c_1, c_2, \ldots$ are the answers to G.

Given that

$$P, \mathcal{T}_\mathcal{C} \models c \rightarrow G \tag{4.4}$$

we show that for each model I of $\mathcal{T}_\mathcal{C}$

$$I \models c \rightarrow \bigvee_{i=1}^{\infty} c_i. \tag{4.5}$$

We can consider the models of P which are based on I. Because $lm(P, I)$ is a model of $P, \mathcal{T}_\mathcal{C}$, by (4.4), we have that

$$lm(P, I) \models c \rightarrow G. \tag{4.6}$$

By Theorem 4.11, we have that

$$lm(P, I) \models G \leftrightarrow (\bigvee_{i=1}^{\infty} c_i).$$

Thus by (4.6),

$$lm(P, I) \models c \rightarrow (\bigvee_{i=1}^{\infty} c_i).$$

And this means

$$I \models c \rightarrow \bigvee_{i=1}^{\infty} c_i.$$

The theorem now follows from the Compactness Theorem (see for example [21]).
□

This is a very strong result. It is worth pointing out, that in general, n can be greater than 1.

Example 4.5. Consider the $CLP(Real)$ program P:

$$\begin{array}{lll} p(X) & \texttt{:-} & X \geq 2. \\ p(X) & \texttt{:-} & X \leq 2. \end{array}$$

Then

$$P, \mathcal{T}_{Real} \models true \rightarrow p(X).$$

and the answers to $p(X)$ are $X \geq 2$ and $X \leq 2$. Both answers are needed to cover $true$:

$$\mathcal{T}_{Real} \models true \rightarrow (X \geq 2 \vee X \leq 2)$$

However, for some constraint domains, the number of answers which need to be considered is just one. The following definition captures such cases.

Definition 4.8. A theory T for a constraint domain has *independence of constraints* if for all constraints $c, c_1, \ldots, c_n$,

$$T \models c \leftrightarrow (\bar{\exists}_{vars(c)} c_1 \vee \ldots \vee \bar{\exists}_{vars(c)} c_n).$$

implies that for some i, $T \models c \leftrightarrow \bar{\exists}_{vars(c)} c_i$.

The following is a corollary of Theorem 4.12.

Corollary 4.5. Let P be a $CLP(\mathcal{C})$ program, G be a goal and let $\mathcal{T}_{\mathcal{C}}$ have independence of constraints. If $P, \mathcal{T}_{\mathcal{C}} \models c \rightarrow G$ for constraint c, then G has an answer A such that $P, \mathcal{T}_{\mathcal{C}} \models c \rightarrow A$. $\square$

The constraint theory $\mathcal{T}_{Real}$ does not have independence of constraints, witness Example 4.5. The constraint theory $\mathcal{T}_{Term}$ does have independence of constraints as long as there are an infinite number of function symbols. This explains the stronger logical completeness result for logic programs, for which any logical answer will be covered by a single qualified answer.

Finally, we can recast the results of this section in terms of the program's "success set." This set essentially contains the answers that the program will give to single atom queries.

Definition 4.9. The *success set* of a program P, SS_P, is the set of facts

$$\{A \texttt{ :- } c \mid c \text{ is an answer to } A \text{ for } P \text{ for some atom } A\}.$$

Theorem 4.13. Let P be a $CLP(\mathcal{C})$ program. The following are equivalent:

- $[SS_P]_{\mathcal{C}}$,
- $[lfp(S_P)]_{\mathcal{C}}$,
- $lfp(T_P^{\mathcal{C}})$,
- $lm(P, \mathcal{C})$.

PROOF. The first equivalence follows from Theorem 4.9, the second from Theorem 4.8, and the third from Theorem 4.6. □

5. SEMANTICS FOR FINITE FAILURE

We have seen that in the operational semantics for CLP programs, goals can also finitely fail. We now give an algebraic and a logical semantics for finite failure for CLP languages. Our first step is to define the Clark completion of a program.

5.1. The Clark completion

The algebraic and logical semantics we gave in the last section for successful goals does not fit well with finite failure, since there is at least one $\mathcal{C}$-model, namely the $\mathcal{C}$-base, in which any goal is satisfiable. The problem is that there are too many models for a program. This is possible because a rule is only read as an "if" definition for its head.

When dealing with finite failure, a constraint logic program must be understood as representing its "Clark completion". The Clark completion captures the reasonable assumption that the programmer really wants the rules defining a predicate to be an "if and only if" definition—the rules should cover all of the cases which make the predicate true. Clark's original definition, for logic programs, also included the theory of $Term$ [2].

Definition 5.1. The *definition* of n-ary predicate symbol p in the program P, is the formula

$$\forall X_1 \ldots \forall X_n \; p(X_1, \ldots, X_n) \leftrightarrow B_1 \vee \ldots \vee B_m$$

where each B_i corresponds to a rule in P of the form

$$p(t_1, \ldots, t_n) \texttt{ :- } L_1, \ldots, L_k$$

and B_i is

$$\exists Y_1 \ldots \exists Y_j \; (X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge L_1 \ldots \wedge L_k)$$

where $Y_1, \ldots, Y_j$ are the variables in the original rule and $X_1, \ldots, X_n$ are variables that do not appear in any rule. Note that if there is no rule with head p, then the definition of p is simply

$$\forall X_1 \ldots \forall X_n \; p(X_1, \ldots, X_n) \leftrightarrow false$$

as $\bigvee \emptyset$ is naturally considered to be $false$.

The *(Clark) completion*, $P^\star$, of a constraint logic program P is the conjunction of the definitions of the user-defined predicates in P.

Example 5.1. The completion of the factorial program is

$$\begin{aligned} fac(X_1, X_2) \leftrightarrow{} & (X_1 = 0 \wedge X_2 = 1) \vee \\ & \exists N \, \exists F \; (X_1 = N \wedge X_2 = N * F \wedge N \geq 1 \wedge fac(N-1, F)). \end{aligned}$$

If we take a program's completion as the logical formula which captures the *true* meaning of the program then the intended interpretation of the program should be a $\mathcal{C}$-interpretation which is a model for the completion.

Definition 5.2. Let P be a $CLP(\mathcal{C})$ program. A $\mathcal{C}$*-model* for $P^\star$ is a $\mathcal{C}$-interpretation which is a model for $P^\star$.

Example 5.2. Recall the factorial program and its completion from Example 3.1. The only *Real*-model for the completion is:

$$\{fac(n, n!) \mid n \in \{0, 1, 2, ...\}\}.$$

Other *Real*-interpretations, such as

$$\{fac(n, n!) \mid n \in \{0, 1, 2, ...\}\} \cup \{fac(n, 0) \mid n \in \{0, 1, 2, ...\}\}$$

or

$$\{fac(r, r') \mid r, r' \in \mathcal{R}\}$$

which are models of the original program are not models of the completion.

Of course there may still be more than one $\mathcal{C}$-model for a program's completion, witness the $CLP(Real)$ program

```
p(X) :- p(X).
```

The models of the completion have a very natural relationship with the fixpoints of the immediate consequence function: the $\mathcal{C}$-models are exactly the fixpoints of $T_P^{\mathcal{C}}$.

Lemma 5.1. Let P be a $CLP(\mathcal{C})$ program. A $\mathcal{C}$-interpretation I is a model of $P^\star$ iff I is a fixpoint of $T_P^{\mathcal{C}}$. $\square$

Given this relationship, it is clear that the completion of a program has a least and greatest $\mathcal{C}$-model which are the least and greatest fixpoints of $T_P^{\mathcal{C}}$.

Definition 5.3. Let P be a $CLP(\mathcal{C})$ program. We denote the least $\mathcal{C}$-model of $P^\star$ by $lm(P^\star, \mathcal{C})$ and the greatest $\mathcal{C}$-model of $P^\star$ by $gm(P^\star, \mathcal{C})$.

This allows us to relate the algebraic semantics of the program completion to the fixpoint semantics

Theorem 5.1. Let P be a $CLP(\mathcal{C})$ program.

- $lm(P^\star, \mathcal{C}) = lfp(T_P^{\mathcal{C}}) = T_P^{\mathcal{C}} \uparrow \omega = lm(P, \mathcal{C})$.
- $gm(P^\star, \mathcal{C}) = gfp(T_P^{\mathcal{C}})$. $\square$

There is a very natural notion of failure if the semantics of a program P is regarded as the models of its completion. Namely, G should fail iff $\tilde{\forall} \neg\, G$ holds in all $\mathcal{C}$-models of $P^\star$. This is symmetric with our notion of success, as can be seen from the following result.

Theorem 5.2. Let P be a $CLP(\mathcal{C})$ program and G a goal.

- $P^\star, \mathcal{D}_\mathcal{C} \models \tilde{\exists} G$ *iff* $lm(P^\star, \mathcal{C}) \models \tilde{\exists} G$.
- $P^\star, \mathcal{D}_\mathcal{C} \models \neg\, \tilde{\exists} G$ *iff* $gm(P^\star, \mathcal{C}) \models \neg\, \tilde{\exists} G$.

PROOF. Since $lm(P^\star, \mathcal{C})$ is a $\mathcal{C}$-model of $P^\star$, if $P^\star, \mathcal{D}_\mathcal{C} \models \tilde{\exists} G$ then $lm(P^\star, \mathcal{C}) \models \tilde{\exists} G$. To prove the other direction, suppose $lm(P^\star, \mathcal{C}) \models \tilde{\exists} G$. Then $lm(P^\star, \mathcal{C}) \models_\sigma G$, for some valuation σ. For every $\mathcal{C}$-model M of $P^\star$, we have $lm(P^\star, \mathcal{C}) \subseteq M$, so that, by Lemma 4.2, $M \models_\sigma G$. Thus $P^\star, \mathcal{D}_\mathcal{C} \models \tilde{\exists} G$.

The second item is proved as follows. Now $P^\star, \mathcal{D}_\mathcal{C} \models \neg\, \tilde{\exists} G$ implies $gm(P^\star, \mathcal{C}) \models \neg\, \tilde{\exists} G$, as $gm(P^\star, \mathcal{C})$ is a $\mathcal{C}$-model of $P^\star$. Now we prove the other direction. The proof is by contradiction. Assume that $gm(P^\star, \mathcal{C}) \models \neg\, \tilde{\exists} G$, but that for some $\mathcal{C}$-model of $P^\star$, M say, and valuation σ, $M \models_\sigma G$. But $gm(P^\star, \mathcal{C}) \supseteq M$. Hence by Lemma 4.2, $gm(P^\star, \mathcal{C}) \models_\sigma G$, a contradiction. □

Having related the previously developed logical and algebraic semantics to the Clark completion, we now turn to the operational semantics.

We first prove that the results for success given in the last section continue to hold if a program P is replaced by its completion $P^\star$. We can then prove the operational semantics for success is sound with respect to the program completion. This depends on the following proposition.

Proposition 5.1. Let P be a $CLP(\mathcal{C})$ program. Then, $\mathcal{T}_\mathcal{C} \models P^\star \rightarrow P$.

PROOF. Straightforward from the definition of $P^\star$. □

Corollary 5.1. Let P be a $CLP(\mathcal{C})$ program. If $P, \mathcal{T}_\mathcal{C} \models c \rightarrow G$ then

$$P^\star, \mathcal{T}_\mathcal{C} \models c \rightarrow G. \quad \square$$

Theorem 5.3. Let P be a $CLP(\mathcal{C})$ program. If goal G has answer c, then

$$P^\star, \mathcal{T}_\mathcal{C} \models c \rightarrow G.$$

PROOF. If G has answer c, then from Theorem 4.1

$$P, \mathcal{T}_\mathcal{C} \models c \rightarrow G.$$

From Corollary 5.1,

$$P^\star, \mathcal{T}_\mathcal{C} \models c \rightarrow G. \quad \square$$

The second result we need to show is that the operational semantics is complete with respect to the completion semantics. We do this by proving the converse of Corollary 5.1.

Lemma 5.2. Let P be a $CLP(\mathcal{C})$ program. If $P^\star, \mathcal{T}_\mathcal{C} \models c \rightarrow G$ then

$$P, \mathcal{T}_\mathcal{C} \models c \rightarrow G.$$

PROOF. Let I be any model of $\mathcal{T}_\mathcal{C}$. From the hypothesis, $lm(P^\star, I) \models c \rightarrow G$. By Theorem 5.1, $lm(P, I) \models c \rightarrow G$. For any valuation σ that satisfies c we have $lm(P, I) \models_\sigma G$ and so, by Theorem 4.3, $P, I \models_\sigma G$. Since this applies to all valuations satisfying c, $P, I \models c \rightarrow G$. Since I was arbitrary, $P, \mathcal{T}_\mathcal{C} \models c \rightarrow G$. □

Theorem 5.4. Let P be a $CLP(\mathcal{C})$ program. Let G be a goal and c a constraint. If $P^\star, \mathcal{T}_\mathcal{C} \models c \rightarrow G$ then G has answers $c_1, \ldots, c_n$ such that

$$\mathcal{T}_\mathcal{C} \models c \rightarrow (c_1 \vee \ldots \vee c_n).$$

PROOF. If $P^\star, \mathcal{T}_\mathcal{C} \models c \rightarrow G$, then from Lemma 5.2, $P, \mathcal{T}_\mathcal{C} \models c \rightarrow G$. It follows from Theorem 4.12 that G has answers $c_1, \ldots, c_n$ such that

$$\mathcal{T}_\mathcal{C} \models c \rightarrow (c_1 \vee \ldots \vee c_n). \quad \Box$$

5.2. Soundness

In order to prove soundness of finite failure we need to develop a stronger relationship between a state and the states it can be reduced to. Our first result is a generalization of Theorem 4.1.

Lemma 5.3. Let P be a $CLP(\mathcal{C})$ program. If $\langle G \mid c\rangle$ is reducible, and using selected literal L may be reduced to any of the states $\langle G_1 \mid c_1\rangle, \ldots, \langle G_m \mid c_m\rangle$ then

$$P^\star, \mathcal{T}_\mathcal{C} \models (G \wedge c) \leftrightarrow \bigvee_{i=1}^{m} \bar{\exists}_{vars(G\wedge c)}(G_i \wedge c_i).$$

PROOF. Let G be of the form $L_1, \ldots, L_n$ where L_i is the selected literal. There are four cases to consider.

The first case is when L_i is a primitive constraint and $solv(c \wedge L_i) \neq false$. In this case, $\langle G \mid c\rangle$ is reducible to the single state $\langle G' \mid c'\rangle$ where G' is $L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$ and c' is $c \wedge L_i$. Thus $G' \wedge c'$ is $L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_n \wedge c \wedge L_i$ and so,

$$P^\star, \mathcal{T}_\mathcal{C} \models (G \wedge c) \leftrightarrow \bar{\exists}_{vars(G\wedge c)}(G' \wedge c').$$

The second case is when L_i is a primitive constraint and $solv(c \wedge L_i) = false$. In this case, $\langle G \mid c\rangle$ is reducible to the single state $\langle G' \mid c'\rangle$ where G' is $\Box$ and c' is $false$. As the solver is correct with respect to the theory, this means that $L_i \wedge c$ and hence $G \wedge c$ are unsatisfiable in any model of $\mathcal{T}_\mathcal{C}$. Thus,

$$P^\star, \mathcal{T}_\mathcal{C} \models (G \wedge c) \leftrightarrow \bar{\exists}_{vars(G\wedge c)}(G' \wedge c').$$

Otherwise L_i is an atom. Let L_i be of the form $p(\vec{s})$. The third case is when there are rules defining p in P. Let them be

$$\begin{array}{lll} p(\vec{t_1}) & \texttt{:-} & B_1 \\ & \vdots & \\ p(\vec{t_m}) & \texttt{:-} & B_m. \end{array}$$

Then $\langle G \mid c\rangle$ can be reduced to $\langle G_1 \mid c_1\rangle, \ldots, \langle G_m \mid c_m\rangle$ where c_i is c and G_i is

$$L_1, \ldots, L_{i-1}, \vec{s} = \rho_i(\vec{t_i}), \rho_i(B_i), L_{i+1}, \ldots, L_n$$

where ρ_i renames the i'th rule from the variables in the original state.

Choose $\vec{z}$ to be distinct new variables. Because $\mathcal{T}_\mathcal{C}$ treats equality as identity,

$$\mathcal{T}_\mathcal{C} \models p(\vec{s}) \leftrightarrow \exists \vec{z}\ \vec{s} = \vec{z} \wedge p(\vec{z}). \tag{5.1}$$

From the definition of $P^\star$, it contains the definition of p, which is the sentence

$$\forall \vec{x}\ p(\vec{x}) \leftrightarrow (\exists \vec{y_1}\ \vec{x} = \vec{t_1} \wedge B_1) \vee \cdots \vee (\exists \vec{y_m}\ \vec{x} = \vec{t_m} \wedge B_m).$$

Hence

From (5.1),

$$P^\star, \mathcal{T}_\mathcal{C} \models p(\vec{s}) \leftrightarrow \exists \vec{z}\ \vec{s} = \vec{z} \wedge (\exists \vec{y_1}\ \vec{z} = \vec{t_1} \wedge B_1) \vee \cdots \vee (\exists \vec{y_m}\ \vec{z} = \vec{t_m} \wedge B_m)).$$

Thus,

$$P^\star, \mathcal{T_C} \models p(\vec{s}) \leftrightarrow \bigvee_{i=1}^{m} (\exists \vec{z}\ \vec{s} = \vec{z} \wedge \exists \vec{y_i}\ \vec{z} = \vec{t_i} \wedge B_i). \tag{5.2}$$

and so

$$(\exists \vec{z}\ \vec{s} = \vec{z} \wedge \exists \vec{y_i}\ \vec{z} = \vec{t_i} \wedge B_i) \leftrightarrow (\exists \vec{z}\ \vec{s} = \vec{z} \wedge \exists \rho_i(\vec{y_i})\ \vec{z} = \rho_i(\vec{t_i}) \wedge \rho_i(B_i)).$$

As ρ_i renames away from $\vec{s}$,

$$(\exists \vec{z}\ \vec{s} = \vec{z} \wedge \exists \vec{y_i}\ \vec{z} = \vec{t_i} \wedge B_i) \leftrightarrow (\exists \rho_i(\vec{y_i}) \exists \vec{z}\ \vec{s} = \vec{z} \wedge \vec{z} = \rho_i(\vec{t_i}) \wedge \rho_i(B_i)).$$

From the fact that $\mathcal{T_C}$ treats equality as identity,

$$\mathcal{T_C} \models (\exists \vec{z}\ \vec{s} = \vec{x} \wedge \exists \vec{y_i}\ \vec{z} = \vec{t_i} \wedge B_i) \leftrightarrow (\exists \rho_i(\vec{y_i})\ \vec{s} = \rho_i(\vec{t_i}) \wedge \rho_i(B_i)).$$

Thus from (5.2),

$$P^\star, \mathcal{T_C} \models p(\vec{s}) \leftrightarrow \bigvee_{i=1}^{m} (\exists \rho_i(\vec{y_i})\ \vec{s} = \rho_i(\vec{t_i}) \wedge \rho_i(B_i)).$$

Clearly, since ρ_i renames the variables $\vec{y_i}$ away from the variables in the original goal,

$$P^\star, \mathcal{T_C} \models G \wedge c \leftrightarrow \bigvee_{i=1}^{m} \exists \rho_i(\vec{y_i})\ (L_1 \wedge \ldots L_{i-1} \wedge \vec{s} = \rho_i(\vec{t_i}) \wedge \rho_i(B_i) \wedge L_{i+1} \wedge \ldots \wedge L_m \wedge c).$$

and from the definition of each G_i and c_i

$$P^\star, \mathcal{T_C} \models G \wedge c \leftrightarrow \bigvee_{i=1}^{m} \exists \rho_i(\vec{y_i})\ G_i \wedge c_i.$$

Hence,

$$P^\star, \mathcal{T_C} \models (G \wedge c) \leftrightarrow \bigvee_{i=1}^{n} \bar{\exists}_{vars(G \wedge c)}\ (G_i \wedge c_i).$$

The fourth case is when there are no rules in P defining p. This means that L_i and hence $G \wedge c$ are unsatisfiable in any model of $P^\star$. In this case, $\langle G \mid c \rangle$ is reducible to the single state $\langle G' \mid c' \rangle$ where G' is $\Box$ and c' is $false$. Thus,

$$P^\star, \mathcal{T_C} \models (G \wedge c) \leftrightarrow \bar{\exists}_{vars(G \wedge c)}(G' \wedge c'). \quad \Box$$

Now we are in a position to relate the answers of finitely evaluable goals to the logical semantics. A goal is finitely evaluable if it has a finite derivation tree.

Theorem 5.5. Let $\mathcal{T_C}$ be a theory for constraint domain $\mathcal{C}$ and P be a $CLP(\mathcal{C})$ program. Let G be a goal which is finitely evaluable with answers $c_1, \ldots, c_n$. Then

$$P^\star, \mathcal{T_C} \models G \leftrightarrow (c_1 \vee \ldots \vee c_n).$$

PROOF. The proof is by induction on the partial derivation trees[1] $T_1, \ldots T_k$ constructed from G where T_k is the final derivation tree. The induction hypothesis

[1] Partial derivation trees are a generalization of derivation trees in which nodes that can reduce may have no children. A partial derivation tree represents an as yet incomplete search of a derivation tree.

is that at stage i, if the leaves of T_i are the states $\langle G_1 \mid c_1 \rangle$, ..., $\langle G_m \mid c_m \rangle$, then

$$P^\star, \mathcal{T_C} \models G \leftrightarrow \bigvee_{i=1}^{m} \bar{\exists}_{vars(G)}(G_i \wedge c_i).$$

The base case, when $i = 1$ is obvious as T_1 is just $\langle G \mid true \rangle$ and clearly

$$P^\star, \mathcal{T_C} \models G \leftrightarrow \bar{\exists}_{vars(G)}(G \wedge true).$$

We now prove the induction step. Assume that the induction hypothesis holds for T_i where $i < k$. We shall show that it holds for T_{i+1}. Let the leaves of T_i be $\langle G_1 \mid c_1 \rangle$, ..., $\langle G_m \mid c_m \rangle$. By induction hypothesis,

$$P^\star, \mathcal{T_C} \models G \leftrightarrow \bigvee_{i=1}^{m} \bar{\exists}_{vars(G)}(G_i \wedge c_i). \tag{5.3}$$

Now T_{i+1} is constructed from T_i by choosing a leaf state, say $\langle G_j \mid c_j \rangle$ and adding as children the states, $\langle G'_1 \mid c'_1 \rangle$, ..., $\langle G'_{m'} \mid c'_{m'} \rangle$ which $\langle G_j \mid c_j \rangle$ can be reduced to using the selected literal. By construction, therefore, the leaves of T_{i+1} are

$$\langle G_1 \mid c_1 \rangle, \cdots, \langle G_{j-1} \mid c_{j-1} \rangle \langle G'_1 \mid c'_1 \rangle, \cdots, \langle G'_{m'} \mid c'_{m'} \rangle \langle G_{j+1} \mid c_{j+1} \rangle, \cdots, \langle G_m \mid c_m \rangle.$$

From Lemma 5.3, we have that

$$P^\star, \mathcal{T_C} \models (G_j \wedge c_j) \leftrightarrow \bigvee_{i=1}^{m'} \bar{\exists}_{vars(G_j \wedge c_j)}(G'_i \wedge c'_i).$$

Thus from (5.3),

$$P^\star, \mathcal{T_C} \models G \leftrightarrow (\bigvee_{i=1, i \neq j}^{m} \bar{\exists}_{vars(G)}(G_i \wedge c_i)) \vee \bar{\exists}_{vars(G)}(\bigvee_{i=1}^{m'} \bar{\exists}_{vars(G_j \wedge c_j)}(G'_i \wedge c'_i)).$$

As the variables introduced in the reduction are disjoint from those in G,

$$P^\star, \mathcal{T_C} \models G \leftrightarrow (\bigvee_{i=1, i \neq j}^{m} \bar{\exists}_{vars(G)}(G_i \wedge c_i)) \vee (\bigvee_{i=1}^{m'} \bar{\exists}_{vars(G)}(G'_i \wedge c'_i)).$$

Thus the induction hypothesis holds for T_{i+1}.

By induction we therefore have that for the leaves, $\langle G_1 \mid c_1 \rangle$, ..., $\langle G_p \mid c_p \rangle$ of T_k,

$$P^\star, \mathcal{T_C} \models G \leftrightarrow \bigvee_{i=1}^{p} \bar{\exists}_{vars(G)}(G_i \wedge c_i).$$

As T_k is the final derivation tree, each G_i is the empty goal. Thus,

$$P^\star, \mathcal{T_C} \models G \leftrightarrow \bigvee_{i=1}^{p} \bar{\exists}_{vars(G)} c_i.$$

Now the answers to G are exactly those constraints $\bar{\exists}_{vars(G)} c_i$ which are not $false$. Thus the result follows. □

An immediate corollary to this is logical soundness of finite failure, as this is the special case when there are no answers and $\bigvee \emptyset$ is just $false$.

Corollary 5.2. (Logical Soundness of Finite Failure)

Let $\mathcal{T_C}$ be a theory for constraint domain $\mathcal{C}$ and let P be a CLP($\mathcal{C}$) program. If *goal G finitely fails then* $P^\star, \mathcal{T}_\mathcal{C} \models \neg \tilde{\exists} G$. $\square$

Soundness of finite failure for the algebraic semantics is an immediate consequence of the soundness of finite failure for the logical semantics, as any intended interpretation of the constraint domain is a model of the constraint theory.

Theorem 5.6. (Algebraic Soundness of Finite Failure)
Let P *be a* $CLP(\mathcal{C})$ *program. If goal* G *finitely fails then:*

- $P^\star, \mathcal{D}_\mathcal{C} \models \neg \tilde{\exists} G$, *and*
- $gm(P^\star, \mathcal{C}) \models \neg \tilde{\exists} G$. $\square$

5.3. Logical completeness

Proving completeness of finite failure is more problematic. We begin by investigating completeness with respect to the logical semantics. The first reason is that the solver can be incomplete, and so not detect that a derivation is failed with respect to the theory. For example, a solver which delays non-linears will not determine that the goal $sqr(X, -1)$ with the $CLP(Real)$ program

$$sqr(X, X * X).$$

should fail. For this reason we require the solver to be theory-complete

The second restriction concerns fairness of the literal selection rule—as we have seen selection rules which are not fair may turn failed derivations into infinite derivations.

Example 5.3. Consider the program

$$q \;\; \texttt{:-} \;\; p, 1 = 2.$$
$$p \;\; \texttt{:-} \;\; p.$$

Clearly $gm(P^\star, \mathcal{C}) \models \neg \tilde{\exists} q$, but the goal q will not finitely fail with a left-to-right selection rule.

The example above shows that for completeness we require a scheduling strategy which is fair.

As long as the solver is theory complete and the literal selection strategy is fair, completeness of finite failure holds .

Theorem 5.7. (Logical Completeness of Finite Failure)
Let $\mathcal{T}_\mathcal{C}$ *be a theory for constraint domain* $\mathcal{C}$*, let* P *be a* $CLP(\mathcal{C})$ *program, and let* G *be a goal. If*

$$P^\star, \mathcal{T}_\mathcal{C} \models \neg \tilde{\exists} G$$

then G *finitely fails for any fair selection rule, provided the solver used is theory complete.*

Proof. The proof is rather complex. We prove the contrapositive: if G does not finitely fail for a fair selection rule then the goal is satisfiable in some model of $\mathcal{T}_\mathcal{C}$

and $P^\star$. Clearly this is true if G has a successful derivation. The case of interest is when G has an infinite fair derivation

$$\langle G_0 \mathbin{\rule{1pt}{6pt}} c_0\rangle \Rightarrow \langle G_1 \mathbin{\rule{1pt}{6pt}} c_1\rangle \Rightarrow \langle G_2 \mathbin{\rule{1pt}{6pt}} c_2\rangle \Rightarrow \cdots$$

The key idea is to build a non-standard model of $\mathcal{T_C}$ and $P^\star$ which makes each state in the derivation true. This provides a model of $P^\star, \mathcal{T_C}$ in which G is satisfiable.

First consider the sequence $c_0,\ c_1,\ \ldots$ of constraints. Let c be $\bigwedge_{i=0}^{\infty} c_i$. As the solver is theory complete we know that for each c_i, $\mathcal{T_C} \not\models \neg\, \tilde{\exists} c_i$. From the Compactness Theorem, therefore, $\mathcal{T_C} \not\models \neg\, \tilde{\exists} c$. Thus there is a model I of $\mathcal{T_C}$ and a valuation σ such that

$$I \models_\sigma c.$$

The next step is to build an I-model of $P^\star$. Let M' be the I-interpretation,

$$\{\sigma(A) |\ \text{atom } A \text{ is in goal } G_i \text{ for some } i\}$$

where σ is arbitrarily extended to all variables in the derivation.

Now M' is a post-fixpoint of T_P^I. This is because, for each $\sigma(A) \in M'$, as the derivation is fair, and so A must have been selected, there is an instance of a rule in P of form

$$\sigma(A)\ \texttt{:-}\ \sigma(L_1), \ldots, \sigma(L_n)$$

such that each $\sigma(L_j)$ appears in the derivation. If L_j is an atom, then by definition of M', $\sigma(L_j) \in M'$. If L_j is a primitive constraint, then as the derivation is fair, the renaming of the constraint in the derivation corresponding to L_j will have been selected and placed in the constraint c_k for some k. Thus, $I \models_\sigma L_j$. Hence, $M' \subseteq T_P^I(M')$.

By a standard construction, it follows that there is a fixpoint M of T_P^I such that $M' \subseteq M$. From Lemma 5.1, M is a model of $P^\star$. By construction, for each G_i, $M \models_\sigma G_i$ and so $M \models \tilde{\exists} G$. $\square$

5.4. Algebraic completeness

Algebraic completeness of finite failure is the most difficult result to achieve. Clearly we require the solver to agree with the the domain of computation, on the satisfiability of constraints, that is it must be complete. Note that completeness of the solver implies that the constraint theory is strong enough to determine if every constraint is satisfiable or not, as the solver must agree with the theory. Hence the constraint theory must also be satisfaction complete.

We might expect that for completeness to hold for the algebraic semantics all we need is a complete solver and a fair computation rule. This not true, we require more.

Example 5.4. Consider the $CLP(Term)$ program P

$$\begin{array}{lll} q(a) & \texttt{:-} & p(X) \\ p(f(X)) & \texttt{:-} & p(X) \end{array}$$

$P^\star$ is

$$\forall Y\ (p(Y) \leftrightarrow \exists X\ (Y = f(X) \wedge p(X))) \wedge \forall Y\ (q(Y) \leftrightarrow \exists X\ (Y = a \wedge p(X))).$$

Now the only $Term$-model of $P^\star$ is $\emptyset$ but the atom $q(a)$ does not finitely fail with a complete solver for any selection rule.

Intuitively, the reason for the problem is that the atoms in $T_P^{\mathcal{C}} \downarrow \omega \setminus gfp(T_P^{\mathcal{C}})$ are true in some model, but not true in a $\mathcal{C}$-model.

Example 5.5. Consider the $CLP(Term)$ program P defined above. We can define a pre-interpretation I as follows, that is a model of $\mathcal{T}_{Term}$. Let the domain of I be the Herbrand terms $a, f(a), f(f(a)), \ldots$ as well as the integers. Interpret the functions a and f as follows: $a_I = a$, $f_I(t) = f(t)$ when t is Herbrand, and $f_I(t) = t+1$ when t is an integer. Now $I \models \mathcal{T}_{Term}$ and $\{q(a)\} \cup \{p(z) \mid z \in \mathcal{Z}\}$ is an I-model of $P^\star$ in which $q(a)$ holds.

The problem is that the greatest model of the completion may not be $T_P^{\mathcal{C}} \downarrow \omega$. We can only hope for equality in the case that the greatest model is $T_P^{\mathcal{C}} \downarrow \omega$.

Definition 5.4. A $CLP(\mathcal{C})$ program P is *canonical* if $T_P^{\mathcal{C}} \downarrow \omega = gfp(T_P^{\mathcal{C}})$.

Fortunately, for a large class of constraint domains, including all those of practical interest, every program has an equivalent canonical program (where by equivalent we mean a program with the same success and finite failure behavior as the original, on queries with predicates only from the original program). See [14, 23] for constructions of equivalent canonical programs for the constraint domain $Term$.

Before we show that this condition is sufficient to achieve completeness for the algebraic semantics, we require a number of technical lemmas to relate the ordinal powers of $T_P^{\mathcal{C}}$ and breadth-first derivations.

Definition 5.5. A breadth-first derivation D from state s is *compatible* with a valuation σ if for each state $\langle G \mid c\rangle$ in D, $\mathcal{D}_{\mathcal{C}} \models_\sigma \bar{\exists}_{vars(s)} c$.

Note that a failed BF-derivation is not compatible with any valuation.

The following lemma corresponds to the Lifting Lemma [17] but we are only interested in the case of BF-derivations.

Lemma 5.4. If goal G has a successful or infinite breadth-first derivation compatible with valuation σ and σ is a solution of constraint c, then $\langle G \mid c\rangle$ has a successful or infinite breadth-first derivation compatible with σ.

PROOF. Let G have the breadth-first derivation D,

$$\langle G \mid true\rangle \Rightarrow_{BF} \langle G_1 \mid c_1\rangle \Rightarrow_{BF} \cdots \Rightarrow_{BF} \langle G_i \mid c_i\rangle \Rightarrow_{BF} \cdots,$$

which is compatible with σ. We can assume that the variables introduced in the derivation are disjoint from the variables in c. Now consider the sequence of states, D',

$$\langle G \mid c\rangle \Rightarrow_{BF} \langle G_1 \mid c \wedge c_1\rangle \Rightarrow_{BF} \cdots \Rightarrow_{BF} \langle G_i \mid c \wedge c_i\rangle \Rightarrow_{BF} \cdots$$

We claim that this is a breadth-first derivation from $\langle G \mid c\rangle$. The only reason that it may not be a valid derivation is that for some state in the derivation, $\langle G_i \mid c \wedge c_i\rangle$ we have that c_i is unsatisfiable in the constraint theory. Now, as D is compatible with σ and σ is a solution of c, we have

$$\mathcal{D}_{\mathcal{C}} \models_\sigma c \wedge \bar{\exists}_{vars(G)} c_i.$$

As the introduced variables in D are distinct from c, $vars(G) \subseteq vars(c) \cap vars(c_i)$, and so

$$\mathcal{D}_{\mathcal{C}} \models_\sigma \bar{\exists}_{vars(G) \cup vars(c)} (c \wedge c_i).$$

Hence $c \wedge c_i$ is satisfiable in the constraint theory. It also follows that D' is compatible with σ. $\square$

The following two lemmas relate the breadth-first derivations of goals to the breadth-first derivations from their component literals. These lemmas are one of the chief reasons why we introduce breadth-first derivations, as the lemmas do not hold for ordinary derivations.

Lemma 5.5. Let σ be a valuation on the common variables of literals L_1 and L_2. If there is a breadth-first derivation D_1 from literal L_1 and a breadth-first derivation D_2 from literal L_2 such that D_1 and D_2 are compatible with valuation σ, then there is a breadth-first derivation D_3 from goal L_1, L_2 such that:

1. *D_3 is compatible with σ.*
2. *If D_1 and D_2 are successful then so is D_3.*
3. *The length of D_3 is the maximum of the lengths of D_1 and D_2.*

Proof. Let D_1 be the derivation

$$\langle L_1 \,\|\, true \rangle \Rightarrow_{BF} \langle G_1 \,\|\, c_1 \rangle \Rightarrow_{BF} \cdots \Rightarrow_{BF} \langle G_i \,\|\, c_i \rangle \Rightarrow_{BF} \cdots$$

and D_2 be the derivation

$$\langle L_2 \,\|\, true \rangle \Rightarrow_{BF} \langle G'_1 \,\|\, c'_1 \rangle \Rightarrow_{BF} \cdots \Rightarrow_{BF} \langle G'_i \,\|\, c'_i \rangle \Rightarrow_{BF} \cdots$$

We can assume that the variables introduced in D_1 are disjoint from the variables in D_2 and vice versa. Let D_3 be

$$\langle L_1, L_2 \,\|\, true \rangle \Rightarrow_{BF} \langle G_1, G'_1 \,\|\, c_1 \wedge c'_1 \rangle \Rightarrow_{BF} \cdots \Rightarrow_{BF} \langle G_i, G'_i \,\|\, c_i \wedge c'_i \rangle \Rightarrow_{BF} \cdots$$

It is straightforward to verify that D_3 is a valid breadth-first derivation from L_1, L_2 which satisfies the conditions of the lemma. $\square$

Similarly we can prove that:

Lemma 5.6. If there is a breadth-first derivation D_3 from goal L_1, L_2, then there is a breadth-first derivation D_1 from literal L_1 and a breadth-first derivation D_2 from literal L_2 such that:

1. *If D_3 is compatible with valuation σ then so are D_1 and D_2.*
2. *If D_3 is successful then so are D_1 and D_2.*
3. *The length of D_3 is the maximum of the lengths of D_1 and D_2.* $\square$

Now we are able to relate the ordinal powers of T_P to breadth-first derivations. This result is the key for relating $T_P^{\mathcal{C}}$ to finitely failed derivations, and corresponds to Lloyd's Proposition 13.5 ([17]).

Lemma 5.7. $\sigma(A) \in T_P^{\mathcal{C}} \downarrow i$ *iff using a complete solver A has a breadth-first derivation which is compatible with σ and which is successful with length $< i$ or else has length i.*

PROOF. The proof is by induction on i. The base case is when $i = 0$. This holds because $\sigma(A) \in T_P^{\mathcal{C}} \downarrow 0$ for all A and σ, and every atom A has the breadth-first derivation of length 0 consisting of the initial state $\langle A \,\mathbf{|}\, true\rangle$ which is compatible with every valuation.

We now prove the inductive step. The induction hypothesis is that $\sigma(A) \in T_P^{\mathcal{C}} \downarrow i$ iff using a complete solver A has a breadth-first derivation which is compatible with σ and which is successful with length $< i$ or else has length i. We will prove that $\sigma(A) \in T_P^{\mathcal{C}} \downarrow i+1$ iff using a complete solver A has a breadth-first derivation which is compatible with σ and which is successful with length $\leq i$ or else has length $i+1$.

Consider $\sigma(A) \in T_P^{\mathcal{C}} \downarrow i+1$. Assume A is of form $p(\vec{s})$. From the definition of ordinal powers and the immediate consequence function, for some rule

$$p(\vec{t}) \;\texttt{:-}\; L_1, \ldots, L_n$$

in P and valuation σ' we have that $\sigma(A) = \sigma'(p(\vec{t}))$ and that

$$\mathcal{D}_{\mathcal{C}}, T_P^{\mathcal{C}} \downarrow i \models_{\sigma'} L_1 \wedge \cdots \wedge L_n.$$

We can assume that the variables in the rule are disjoint from the variables in A.

We first prove that each L_j has a breadth-first derivation compatible with σ'. If L_j is a primitive constraint, $\mathcal{D}_{\mathcal{C}} \models_{\sigma'} L_j$. Thus L_j has the successful breadth-first derivation

$$\langle L_j \,\mathbf{|}\, true\rangle \Rightarrow_{BF} \langle \Box \,\mathbf{|}\, L_j\rangle$$

which is compatible with σ' and of length 1. If L_j is an atom, then $\sigma'(L_j) \in T_P^{\mathcal{C}} \downarrow i$. From the induction hypothesis L_j has a breadth-first derivation which is compatible with σ' and which is successful with length $< i$ or else has length i. Thus from Lemma 5.5, the state $\langle L_1, \ldots, L_n \,\mathbf{|}\, true\rangle$ has a breadth-first derivation which is compatible with σ' and which is successful with length $< i$ or else has length i.

Let σ'' be the valuation defined by

$$\sigma''(x) = \begin{cases} \sigma(x) & \text{when } x \in vars(A) \\ \sigma'(x) & \text{otherwise.} \end{cases}$$

It follows that $\langle \vec{s} = \vec{t}, L_1, \ldots, L_n \,\mathbf{|}\, true\rangle$ has a breadth-first derivation which is compatible with σ'' and which is successful with length $< i$ or else has length i. Thus $\langle A \,\mathbf{|}\, true\rangle$ has a breadth-first derivation which is compatible with σ'' and which is successful with length $< i+1$ or else has length $i+1$. As σ'' and σ are identical over the variables in A, this derivation is also compatible with σ. Thus we have proved one direction of the required statement. The other direction is simple reversal of the above argument except that we use Lemma 5.6 instead of Lemma 5.5. $\Box$

Theorem 5.8. (Algebraic Completeness of Finite Failure)
Let P be a canonical $CLP(\mathcal{C})$ program, and let G be a ground goal. If

$$P^\star, \mathcal{D}_{\mathcal{C}} \models \neg\, \tilde{\exists} G$$

then G finitely fails for any fair selection rule, provided a complete solver is used.

PROOF. We prove the contrapositive. We first prove it for the case G is an atom. Assume that G does not finitely fail. Then G has a successful derivation or an infinite fair derivation. Then G has a successful breadth-first derivation or an infinite breadth-first derivation, D_{BF} say. As G is ground, D_{BF} is compatible with any valuation, say σ. From Lemma 5.7, it follows that for all i, $\sigma(G) \in T_P^{\mathcal{C}} \downarrow i$ and so $\sigma(G) \in T_P^{\mathcal{C}} \downarrow \omega$. As P is canonical, $\sigma(G) \in gfp(T_P^{\mathcal{C}})$, and so $\sigma(G) \in gm(P^\star, \mathcal{C})$. Thus,

$$P^\star, \mathcal{D}_{\mathcal{C}} \models \neg \tilde{\exists} G$$

does not hold. The case when G is a conjunction of literals follows a similar argument but uses Lemma 5.6. □

The restriction to canonical programs is not too severe, as almost all programs in practice are canonical. Notice that the completeness result provided by Theorem 5.7 was stronger in the sense that it did not require programs to be canonical or the goal to be ground.

Finally we consider the relationship of the logical and algebraic semantics to the "finite failure set" which is the analogue of the success set.

Definition 5.6. The *finite failure set* of a program P, FF_P, is the set of facts

$$\{A \texttt{ :- } c \mid \langle A \,|\, c\rangle \text{ finitely fails for } P \text{ via some selection rule}\}.$$

The relationship to the logical semantics is a straightforward corollary of Theorem 5.7.

Corollary 5.3. Let P be a $CLP(\mathcal{C})$ program, let A be an atom, and c a constraint. Then $A \texttt{ :- } c \in FF_P$ iff $P^\star, \mathcal{T}_{\mathcal{C}} \models \neg \tilde{\exists}(A \wedge c)$. □

We now examine the relationship of the finite failure set with the algebraic semantics.

Theorem 5.9. Let P be a $CLP(\mathcal{C})$ program. Then

$$[FF_P]_{\mathcal{C}} \subseteq \mathcal{C}\text{-}base_P \setminus T_P^{\mathcal{C}} \downarrow \omega.$$

Proof. The proof is by contradiction. Assume that $\sigma(A) \in T_P^{\mathcal{C}} \downarrow \omega$ and that $\sigma(A) \in [FF_P]_{\mathcal{C}}$. Now $\sigma(A) \in T_P^{\mathcal{C}} \downarrow \omega$, implies that for all i, $\sigma(A) \in T_P^{\mathcal{C}} \downarrow i$. From Lemma 5.7, either A has a successful breadth-first derivation which is compatible with σ or else A has breadth-first derivations of unbounded length which are compatible with σ. Thus, from Koenig's Lemma A either has a successful or an infinite breadth-first derivation which is compatible with σ. Now consider any c such that $\mathcal{C} \models_\sigma c$. Then from Lemma 5.4, $\langle A \,|\, c\rangle$ has a successful or an infinite breadth-first derivation which is compatible with σ. Thus $\langle A \,|\, c\rangle$ cannot finitely fail for any literal selection strategy. Thus $\sigma(A) \notin [FF_P]_{\mathcal{C}}$. □

Unfortunately the reverse inclusion does not hold in general. The most obvious reason is that the solver may not be complete, and so it will "incorrectly" not terminate a failing derivation. However, even if the solver is complete, there may still be an expressiveness problem. The problem is that the constraint domain may not allow the constraints in the fact to "cover" some of the elements.

Example 5.6. Let *Real** be the constraint domain with linear arithmetic equalities and the unary constraint $\neq \pi$ as the only primitive constraints and the usual functions and constants. Now consider the program

$$p(X) \quad \texttt{:-} \quad X \neq \pi.$$

Here $\mathcal{C}\text{-}base_P \setminus T_P^{\mathcal{C}} \downarrow \omega = \{p(\pi)\}$, but there is no constraint c and atom A with predicate symbol p such that the state $\langle A \mid c\rangle$ finitely fails for this program.

To overcome this problem we require a technical restriction on the constraint domain.

Definition 5.7. The constraint domain $\mathcal{C}$ is *solution compact* if for all constraints c, there is a possibly infinite set of constraints C such that

$$\mathcal{D}_{\mathcal{C}} \models \tilde{\forall}(\neg\, c \leftrightarrow \bigvee C).$$

All constraint domains occurring in practice are solution compact. Of course *Real** from Example 5.6 is not, but clearly that domain was a contrived and pathological case. The original definitions of solution compactness [7, 8] included a further condition that was later shown to be unnecessary [18].

Theorem 5.10. Let P be a $CLP(\mathcal{C})$ program. If $\mathcal{C}$ is solution compact and $solv_{\mathcal{C}}$ is a complete solver then

$$[FF_P]_{\mathcal{C}} = \mathcal{C}\text{-}base_P \setminus T_P^{\mathcal{C}} \downarrow \omega.$$

PROOF. From Theorem 5.9,

$$[FF_P]_{\mathcal{C}} \subseteq \mathcal{C}\text{-}base_P \setminus T_P^{\mathcal{C}} \downarrow \omega.$$

We now prove the reverse inclusion. Let $\sigma(A) \in \mathcal{C}\text{-}base_P \setminus T_P^{\mathcal{C}} \downarrow \omega$. Thus for some i, $\sigma(A) \notin T_P^{\mathcal{C}} \downarrow i$. Let $D_1, \ldots, D_n$ be the successful breadth-first derivations from A of length less than i and the breadth-first derivations from A of length i. From Lemma 5.7, no D_j will be compatible with σ. For each D_j, let c_j be the constraint in the last state. It follows that for each c_j, σ is not a solution of $\bar{\exists}_{vars(A)} c_j$. As the constraint domain is solution compact, there is a constraint c'_j such that $c'_j \wedge \bar{\exists}_{vars(A)} c_j$ is unsatisfiable but σ is a solution of c'_j. Let c be $\bigwedge_{j=1}^{n} c'_j$. By construction σ is a solution of c. It follows that $\langle A \mid c\rangle$ cannot have a successful breadth-first derivation or infinite breadth-first derivation, as otherwise from Lemma 5.4, A would have a successful breadth-first derivation or infinite breadth-first derivation compatible with σ. Thus $\langle A \mid c\rangle$ finitely fails for any fair literal selection rule and so $\sigma(A) \in [FF_P]_{\mathcal{C}}$. $\Box$

By combining the above theorem with the definition of canonical program and Theorem 5.1, we have the following result.

Theorem 5.11. Let $\mathcal{C}$ be a solution compact constraint domain and P be a canonical $CLP(\mathcal{C})$ program. If P is evaluated with a complete solver then $[FF_P]_{\mathcal{C}} = \mathcal{C}\text{-}base_P \setminus gm(P^\star, \mathcal{C})$. $\Box$

One should not read too much into Theorem 5.11. It does not guarantee that an atom (or goal) will finitely fail if the atom does not hold in any $\mathcal{C}$-model of the completion, even if the conditions of solution compactness, canonicity and solver completeness are met.

Example 5.7. Let P be the $CLP(Term)$ program

$$p(f(X)) \quad \texttt{:-} \quad p(X).$$

$P^\star$ is $\forall Y(p(Y) \leftrightarrow \exists X(Y = f(X) \wedge p(X)))$. The program is canonical with $T_P^{\mathcal{C}} \downarrow \omega = gfp(T_P^{\mathcal{C}}) = \emptyset$. Thus the program completion has the single $Term$-model $\emptyset$. Thus $\neg\, \exists X p(X)$ holds in all $Term$-models of $P^\star$. However, even with a complete solver the goal $p(X)$ will not finitely fail.

6. CONCLUSION

Constraint logic programs are a generalization of logic programs which are parameterized by the choice of the underling constraint domain. Constraints from the constraint domain can be understood in three complementary ways: operationally by means of a (possibly incomplete) constraint solver; logically by way of the constraint theory; and algebraically, by means of the domain of computation which is the constraint's intended interpretation. These three views are required to be coherent, that is, the domain of computation must model the constraint theory, while the constraint theory must agree with the constraint solver.

We have lifted these three semantics from the constraint domain to give operational, logical and algebraic semantics for constraint logic programs. As for the constraint domain, the semantics form a hierarchy: the operational semantics is the least strong, then the logical semantics, while the algebraic semantics is the strongest semantics. To prove correctness of the semantics we have employed breadth-first derivations and two fixpoint semantics so as to bridge the gap between the algebraic and the operational semantics.

In the case of a successful query each of the semantics agree on what is successful, although, if the solver is incomplete, the operational semantics may have successful derivations which are not satisfiable, producing pseudo-answers that do not correspond to a true success.

Accord between the three semantics for goals which finitely fail is somewhat more difficult to obtain and requires the constraint solver to be more powerful. For the operational semantics to agree with the logical semantics the solver must be theory-complete, and for the operational semantics to agree with the algebraic semantics we need the solver to be complete and a number of other technical conditions to be satisfied.

The diagram shown in Figure 6.1 summaries the relationships between the operational, algebraic and fixpoint semantics in the case. Each semantics is characterized by a subset of $\mathcal{C}$-$base_P$. The diagram shows the containment relationships between these sets and below the diagram gives conditions which imply where containment is actually equality.

It is instructive to relate our results back to the semantic framework developed for logic programs. Pure logic programs can be viewed as an instance of the CLP Scheme based on the $Term$ constraint domain in which constraints are equations over terms. In the $Term$ constraint domain unification is the constraint solving

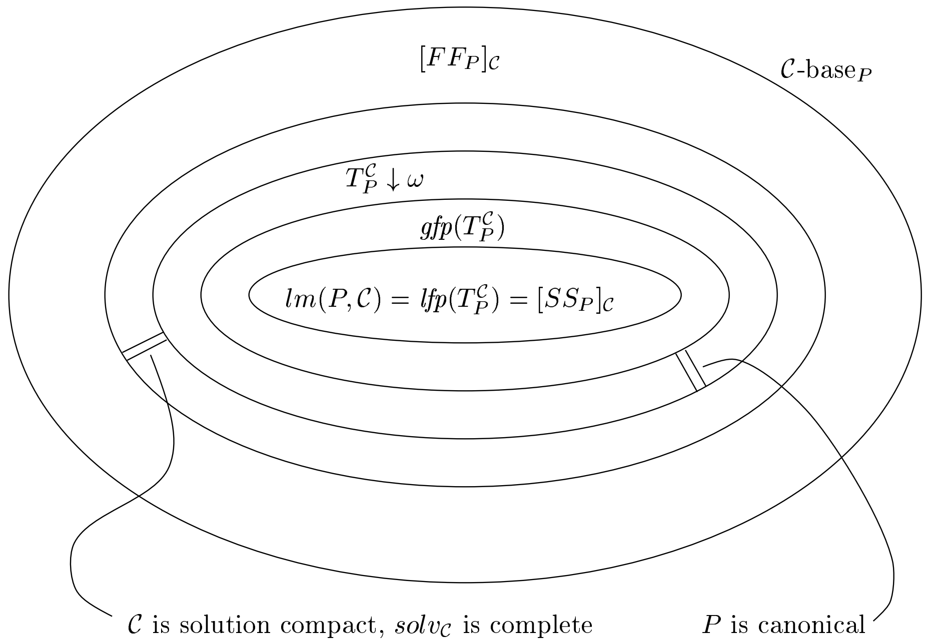

FIGURE 6.1. Relationship between subsets of $\mathcal{C}$-$base_P$

mechanism, the Herbrand universe is the computation domain and the axioms for free equality [2] form the constraint theory. Since the constraint solver is complete, the computation domain is solution compact and independence of constraints holds, we can use our generic results for CLP to immediately obtain the standard semantic theory of logic programs. Thus the semantic theory for CLP strictly generalizes that for logic programs, yet in many cases the statement of results is simpler and proofs are more direct than those standard for logic programming, largely because the vagaries of unification, substitutions and local variables can be factored out.

We thank Jean-Louis Lassez for his comments and discussions, over the years, on the topic of this paper This work was supported by Australian Research Council grants A49702580 and A49700519.

REFERENCES

1. K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, 1982.
2. K. L. Clark. Negation as failure. In *Logic and Databases* (H. Gallaire and J. Minker Eds.). Plenum Press, New York, 293–322, 1978.
3. A. Colmerauer. Prolog-II Manuel de Reference at Modele Theorique. Groupe Intelligence Artificelle, Universite d'Aix-Marseille II, 1982.
4. M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

5. M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative modeling of the operational behavior of logic languages. *Theoretical Computer Science* 69(3):289–318, 1989.
6. M. Gabbrielli and G. Levi, Modeling answer constraints in constraint logic programs, *Proc. 8th International Conference on Logic Programming,* 238–252, 1991.
7. J. Jaffar and J-L. Lassez, Constraint logic programming, Technical Report 86/73, Department of Computer Science, Monash University, 1986.
8. J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proc. Fourteenth Ann. ACM Symp. Principles of Programming Languages*, pages 111–119, 1987.
9. J. Jaffar, J.-L. Lassez and J.W. Lloyd. Completeness of the Negation as Failure Rule. *Proc. IJCAI-83*, 500–506, 1983.
10. J. Jaffar, J.-L. Lassez and M.J. Maher. A theory of complete logic programs with equality. *The Journal of Logic Programming* 3:211–223, 1984.
11. J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19&20, 503–581, 1994.
12. J. Jaffar, S. Michaylov, P. Stuckey & R. Yap, The $CLP(\mathcal{R})$ language and system, *ACM Transactions on Programming Languages*, 14(3), 339–395, 1992.
13. J. Jaffar and P. Stuckey, Semantics of infinite tree logic programming. *Theoretical Computer Science* 46:141–158, 1986.
14. J. Jaffar and P. Stuckey, Canonical logic programs, *Journal of Logic Programming* 3, 143–155, 1986.
15. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, **5**, 285–309, 1955.
16. J-L. Lassez and M.J. Maher, Closures and Fairness in the Semantics of Programming Logic, *Theoretical Computer Science 29* (1984) 167–184.
17. J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, Second Edition, 1987.
18. M. Maher. Logic semantics for a class of committed-choice programs. In *Proc. Fourth Int. Conf. on Logic Programming*, pages 858–876, The MIT Press, 1987.
19. A. Mal'cev, Axiomatizable Classes of Locally Free Algebras of Various Types, in: *The Metamathematics of Algebraic Systems: Collected Papers, 1936–1967*, Chapter 23, 262–281, 1971.
20. K. Marriott and P. Stuckey. *Programming with Constraints: An Introduction*, MIT Press, 1998.
21. E. Mendelson. *Introduction to Mathematical Logic*, Wadsworth and Brooks, Third Edition, 1987.
22. J.R. Shoenfield, *Mathematical Logic*, Addison-Wesley, 1967.
23. M. Wallace, A computable semantics for general logic programs, *Journal of Logic Programming* 6, 269–297, 1989.
24. D.A. Wolfram, M.J. Maher and J.-L. Lassez. A unified treatment of resolution strategies for logic programs. *Proc. Second Int. Conf. on Logic Programming*, Uppsala, 263–276, 1984.