ELSEVIER Science of Computer Programming 25 (1995) 117-147

Science of Computer Programming

Foundational issues in implementing constraint logic programming systems

James H. Andrews*

Department of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6

Abstract

Implementations of Constraint Logic Programming (CLP) systems are often incomplete with respect to the theories they are intended to implement. This paper studies two issues that arise in dealing with these incomplete implementations. First, the notion of "satisfiability function" (the analogue of unification) is formally defined, and the question of which such functions are reasonable is studied. Second, techniques are given, based on the notion of satisfiability function, for formally (proof-theoretically) specifying an intended CLP theory or characterizing an existing CLP system. Such proof-theoretic characterizations have applications in proving soundness and completeness results, and proving properties of programs. Notions from substructural logic and the notion of Henkinness of the theory are shown to be important here.

1. Introduction

The semantics of Constraint Logic Programming (CLP) languages is now well understood. Implementations of CLP languages, however, are often not complete with respect to their intended theories, that is, there are goals G such that G is satisfiable in the intended theory, but the CLP system cannot discover G to be satisfiable. In this paper, I study the foundations of such incomplete implementations, by giving a recursion-theoretic, rather than model-theoretic, basis for CLP operational semantics. Based on this work, I then study proof-theoretic techniques for specifying the *intended* theory of a CLP language, or giving characterizations of the *actual* theory implemented by a CLP language.

A simple example shows that some CLP implementations necessarily incomplete. Consider a first-order language, structure and theory in which the terms encode untyped lambda-expressions, and in which disequality $\neq$ holds between two terms iff they are not extensionally equivalent. The theory and structure can be made to meet

* E-mail: jamie@cs.sfu.ca.

0167-6423/95/$09.50 © 1995—Elsevier Science B.V. All rights reserved
SSDI 0167-6423(95)00011-9

Jaffar and Lassez' original conditions [12] for a reasonable CLP language (that is, the so-called "satisfaction-completeness" and "solution-compactness" conditions). The CLP scheme [12] defines a theoretical operational semantics for this theory such that a ground query $\mathbf{s} \neq \mathbf{t}$ succeeds iff $\mathbf{s}$ and $\mathbf{t}$ are not extensionally equivalent. However, in practice we have no terminating algorithm for testing whether two lambda-terms are extensionally equivalent. Thus, given any particular implementation which is sound with respect to this theory, there is some satisfiable query which cannot be discovered to be satisfiable, not even by using the standard breadth-first technique of complete Herbrand-domain logic programming interpreters.

Furthermore, many implementations of CLP languages are incomplete for efficiency reasons. For instance, CLP(R) [13] implements an efficient but incomplete linear equation solver rather than Tarski's complex algorithm for deciding real arithmetic; and many constraint languages operating over finite domains, such as CHIP [24], implement incomplete algorithms for finite domain constraint satisfaction to avoid exponential time complexity.

To deal with these considerations, we need to develop a theoretical framework, incorporating notions of computability, in which we can study issues of how complete a CLP implementation is. This framework can then be used as a basis for comparing a CLP implementation with its intended theory, or specifying the smaller theory that an incomplete implementation actually implements. This paper is intended to make steps in the direction of such a framework.

In the remainder of this introduction, I present some basic definitions. In Section 2, I define a class of recursive functions called "satisfiability functions", which formalize the basic step in CLP languages (the analogue of unification). I show how operational semantics can be defined based on satisfiability functions, define what it means for a structure to "realize" such a function, and give a necessary and sufficient, non-logical condition for a stisfiability function to be realizable. In Section 3, I use satisfiability functions as a basis for studying various techniques for characterizing CLP theories with proof systems, and show how the proof systems could be used to prove the soundness and completeness of implementations. In Section 4, I present some conclusions and discuss related work. An appendix contains proofs of the characterization theorems given in Section 3.

1.1. Definitions and notation

Definition 1.1. A *first-order language* $\mathscr{L}$ is a tuple $\langle F, P, V \rangle$, where F is a recursive set of function symbols, each with an associated arity, P is a recursive set of predicate names, each with an associated arity, and V is a recursive set of variable names. We define the terms and formulae of $\mathscr{L}$ in the standard way.

Let $\mathscr{C}$ be a set of predicate names of $\mathscr{L}$, $\mathscr{C} \subseteq P(\mathscr{L})$. Constrs$(\mathscr{L}, \mathscr{C})$ is the set of all atomic formulae from $\mathscr{L}$ formed using a predicate name from $\mathscr{C}$. We also call these atomic formulae "constraints".

We define notions of structure, valuation, satisfiability w.r.t. a structure, and model in the standard Tarskian way.

Definition 1.2. A *theory* is a set of closed (ground) formulae.

Notation: We will use A, B, C, G to stand for formulae, c to stand for a constraint, s, t to stand for terms, Γ and Π to stand for multisets of formulae, and S, T to stand for sets (usually sets of constraints). These are fairly standard in the literature on constraint logic programming. We will also use the following notation:

$$\exists[B] = \exists x_1 \cdots \exists x_n(B)$$

where $x_1, \ldots, x_n$ are all the free variables of B. Where $S = \{B_1, \ldots, B_n\}$, we will define

$$\exists[S] = \exists[B_1 \& \cdots \& B_n]$$

We will define $\forall[B]$, $\forall[S]$ similarly.

2. Satisfiability functions

The basic step in constraint logic programming interpreters (even incomplete ones) is the step which decides whether a new constraint is consistent with the previously processed constraints. Every CLP interpreter has an algorithm for doing this for its intended theory; if the algorithm returns "true", the interpreter goes further down the same branch in the search true, and if it returns "false", the interpreter backtracks.[1]

This section studies the theory of such "satisfiability functions", which will be defined as *partial* recursive functions from finite sets of constraints to results *including* "true" and "false". The motivation for doing so is to build a homogeneous theory of CLP systems, whether they use partial or total satisfiability functions, and whether those functions map to "true", "false" or some other outcome.

In the first subsection, I define the notion of satisfiability function, and show how an operational semantics can be built on the basis of that notion (rather than the notion of constraint theory). In the second subsection, I point out that not all satisfiability functions correspond to actual constraint theories which "realize" them, and that these functions' operational semantics thus do not define sensible logic programming systems. I give, however, a condition on satisfiability functions which is necessary and sufficient for realizability. In the final subsection, I point out that for a given constraint theory, there are either 0 or 1 maximal satisfiability functions which are realized by it.

[1] This does not actually describe completely the operation of all languages referred to as constraint logic programming languages. (For instance, it does not take into account disjunctive constraints.) However, it describes one common framework for CLP operational semantics, and is also the one considered in much of the standard literature on CLP, e.g. [16].

2.1. *Satisfiability functions and operational semantics*

Definition 2.1. A *satisfiability function* (in a language $\mathscr{L}$ with constraint predicates $\mathscr{C}$) is a partial recursive function whose domain is the set of finite sets of constraints in Constrs($\mathscr{L}, \mathscr{C}$), and whose range is a countable set $\mathscr{T}$ of truth values containing at least *true* and *false*.

In the sequel, we will not mention the language and set of constraint predicates if they are implicit from the context.

This definition of satisfiability function is general enough to capture the behaviour of a wide variety of complete and incomplete implementations. We always interpret a result of *false* as "unsatisfiable" and *true* as "satisfiable"; but other results, or no result, are also possible. To capture the very well-behaved satisfiability functions, we make the following definition:

Definition 2.2. A *strict* satisfiability function is one which is never undefined and always returns *true* or *false*.

The satisfiability function associated with basic Herbrand-domain Prolog, for instance, is strict.[2] Here is a non-strict example.

Example. The basic behaviour of the CLP(R) system [13] on real-number constraints can be characterized with the following satisfiability function *sat*, whose codomain of truth values is $\{true, false, unsure\}$.

- If S contains a subset T consisting of unsatisfiable, linear constraints, $sat(S)$ returns *false*.
- Otherwise, if S contains no non-linear constraints, $sat(S)$ returns *true*.
- Otherwise (i.e. S contains non-linear constraints but its linear constraints are satisfiable), $sat(S)$ returns *unsure*.

The operational semantics defined from a satisfiability function is a function from states to states.

Definition 2.3. A *state* (of a CLP operational semantics) on $(\mathscr{L}, \mathscr{C}, \mathscr{T})$ is either a truth value in $\mathscr{T}$, or a pair $\langle G, S\rangle$, where G is a multiset of non-constraint atoms of $\mathscr{L}$ and S is a set of constraints of $\mathscr{L}, \mathscr{C}$.

In basing an operational semantics on a satisfiability function *sat*, we may want to give a definition that takes into account the various truth values which can act as

[2] Correct Prolog using unification with occurs check, that is. The widely used unification algorithms without occurs check are incorrect with respect to the Herbrand domain, but correspond to satisfiability functions nonetheless. The satisfiability functions they correspond to are non-strict, however, because they can get into infinite loops.

results of *sat*. We must make the following minimum requirements, following Maher [16].

Definition 2.4. Given a satisfiability function *sat* whose range is the truth values in $\mathcal{T}$, a binary relation $\rightarrow$ between states is an *operational transition relation* for *sat* with program P if:

1. The relation includes the transition

$$\langle G \cup H\theta, S \rangle \rightarrow \langle G \cup B\theta, S \cup S'\theta \rangle$$

if the program P contains (some renaming of) the clause $(H \leftarrow S', B)$, and $sat(S \cup S'\theta) \simeq true$.

2. The relation includes the transition

$$\langle G \cup H, S \rangle \rightarrow false$$

if for every (renamed) clause in P of the form $(H' \leftarrow S', B)$ and substitution θ such that $H'\theta$ is H, we have that $sat(S \cup S'\theta) \simeq false$.

We say that a goal G *succeeds* if $\langle G, \emptyset \rangle \rightarrow^* \langle \emptyset, S \rangle$ and $sat(S) \simeq true$; we define *fair* derivations in the usual way and say that G *fails* if every fair derivation ends in the state *false*. Note that if *sat* is strict, then the least such relation is an operational semantics similar to that defined by Maher [16].

Example. Based on the definition of *sat* for CLP(R) above, we can characterize an operational semantics for CLP(R) as follows.[3] The transition relation is the least transition relation for (sat, P) having the additional property that:

- We have the transition

$$\langle G \cup H\theta, C \rangle \rightarrow \langle G \cup B\theta, C \cup C'\theta \rangle$$

  if the program P contains (some renaming of) the clause $H \leftarrow C', B$, and $sat(C \cup C'\theta) \simeq unsure$.

In this operational semantics, even if we are unsure of the satisfiability of the resulting set of constraints (if the system of equations is not linear), we go on as if it were satisfiable. We may wish to say that a goal G is *indeterminate* if it does not fail, but every fair derivation ends in either *false* or *unsure*.

2.2. Realizable satisfiability functions

We would like our operational semantics to define sensible logic programming systems. To achieve that goal, we have to put a condition on the satisfiability functions

[3] Actually, CLP(R) is more powerful in some ways and less powerful in other ways than what is described here. CLP(R) uses unification over regular first-order terms, negation as failure, etc.; but it also uses the usual incomplete depth-first search algorithm of Prolog, etc. The example ignores these features, which are irrelevant to the present discussion.

we use. The condition is an analogue of Kleene's notion of "realizability", and is best defined model-theoretically.

Definition 2.5. An $\mathscr{L}$-structure $\mathfrak{R}$ *realizes* a satisfiability function *sat* if:
1. whenever $sat(S) \simeq true$, S is $\mathfrak{R}$-satisfiable; and
2. whenever $sat(S) \simeq false$, S is not $\mathfrak{R}$-satisfiable.

If $\mathfrak{R}$ realizes *sat*, we also say that *sat implements* $\mathfrak{R}$.

Not every satisfiability function is realizable, not even the strict ones. For instance, if *sat* maps $\{p(x)\}$ onto *false* but $\{p(x), q(x)\}$ onto *true*, then it will not have any realizing structure, because any valuation satisfying $\{p(x), q(x)\}$ will surely satisfy $\{p(x)\}$. An operational semantics based on such a satisfiability function would give unexpected results.

The realizability of *sat* can be given an equivalent characterization in terms of the theory associated with *sat*.

Definition 2.6. Θ_{sat}, the *theory associated with sat*, is defined as

$$\{\exists[S] \mid sat(S) \simeq true\} \cup \{\neg\exists[S] \mid sat(S) \simeq false\}$$

where $\exists[S]$ is the existential closure of the conjunction of the constraints in S.

We have the following property:

Theorem 2.7. *sat is realizable iff Θ_{sat} is consistent.*

Proof. ($\rightarrow$) If some $\mathfrak{R}$ realizes *sat*, then by the definitions of realization of *sat* and satisfaction of a formula, every formula in Θ_{sat} is true in $\mathfrak{R}$. Θ_{sat} therefore has $\mathfrak{R}$ as a model, so it must be consistent.

($\leftarrow$) Let $\mathfrak{R}$ be a model of Θ_{sat}. By the definition of satisfaction of a formula, every S such that $\exists[S] \in \Theta_{sat}$ is $\mathfrak{R}$-satisfiable, so every S such that $sat(S) \simeq true$ is $\mathfrak{R}$-satisfiable; similarly, every S such that $sat(S) \simeq false$ is not $\mathfrak{R}$-satisfiable. But then $\mathfrak{R}$ realizes *sat*. $\square$

2.3. A condition equivalent to realizability

The consistency of Θ_{sat} is a necessary and sufficient condition for the realizability of *sat*. However, if we have a precise description or an algorithm for a given *sat* in hand, it would be useful to have a more direct method of testing whether it is realizable (more direct than translating it into a theory and testing the theory's consistency). The "reliability" condition allows us to do this. One interesting aspect of reliability is that it is a somewhat weaker condition than we might expect.

But first, some technical definitions.

Definition 2.8. Let S be a set of constraints and let $c \in S$. The *variable sharing class* $S|_c$ is the smallest subset of S such that:

- $c \in S|_c$;
- if $b \in S|_c$, and $a \in S$ shares a free variable with b, then $a \in S|_c$.

Definition 2.9. Let *sat* be a satisfiability function. A set of constraints S is *sat-covered* if for all $c \in S$, there is a set $T \supseteq S|_c$ such that $sat(T) \simeq true$. A set of constraints S is *sat-consistent* if there is some substitution θ such that $S\theta$ is sat-covered.

Basically, the variable sharing classes of S form a partition of S such that no variable is referred to by constraints in any two distinct elements of the partition. A set S is *sat*-consistent if, for some θ, every element of this partition of $S\theta$ is satisfiable in any structure realizing *sat*.

Definition 2.10. A satisfiability function *sat* is *reliable* if whenever $sat(S) \simeq false$, S is not *sat*-consistent.

For non-reliable satisfiability functions, some sets S are considered unsatisfiable despite the fact that some instance of S can be partitioned into sets which are generalizations of sets considered satisfiable. This is a situation which does not meet with our intuitions, and indeed the next theorem proves that reliability is a necessary condition for realizability.

Theorem 2.11. *If a satisfiability function sat is realizable, then it is reliable.*

Proof. Assume (toward a contradiction) that *sat* is realized by some $\mathfrak{R}$ but not reliable. By non-reliability, there must be an S and θ such that $sat(S) \simeq false$ but $S\theta$ is *sat*-covered. Let $T_1, T_2, \ldots, T_n$ be all the (disjoint) variable sharing classes in $S\theta$. By the definition of *sat*-covering, each T_i is a subset of a set T_i' such that $sat(T_i') = true$; so since *sat* is realized by $\mathfrak{R}$, there must be some valuation v_i satisfying T_i in $\mathfrak{R}$. But the union v of these valuations must be a valuation satisfying $S\theta$ in $\mathfrak{R}$; therefore the valuation $v \circ \theta$ must satisfy S in $\mathfrak{R}$. But we had assumed that $sat(S) \simeq false$, so the assumption that $\mathfrak{R}$ realizes *sat* is contradicted. □

Reliability is also sufficient for realizability, as we will see next.

Theorem 2.12. *If a satisfiability function sat is reliable, then it is realizable.*

Proof. By Theorem 2.7, it is sufficient to prove that if *sat* is reliable, Θ_{sat} is consistent. Assume that *sat* is reliable. By compactness, it is sufficient to prove that every finite subset of Θ_{sat} has a model.

Let S be a finite subset of Θ_{sat}, where

$$S = \{\exists[S_1], \ldots, \exists[S_m]\} \cup \{\neg\exists[T_1], \ldots, \neg\exists[T_n]\}$$

Let $\mathfrak{R}$ be the minimal structure which contains a unique element $e_{i,k}$ for each free variable in each S_i, and in which each S_i is satisfiable by a valuation mapping variables to these elements. (It is left to the reader to construct the structure $\mathfrak{R}$ with these properties.) Clearly this structure is a model of the positive formulae in S; we have only to prove that it is a model of the negated formulae too.

Assume, toward a contradiction, that some T_j is satisfied by some valuation v in $\mathfrak{R}$. There is some θ and v' such that $v = \theta v'$, v' maps free variables directly to elements $e_{i,k}$ of the domain of $\mathfrak{R}$, and $T_j\theta$ is satisfied by v'. But by the construction and minimality of $\mathfrak{R}$, this means that each variable sharing class of $T_j\theta$ is a free-variable variant of a subset of some S_i; thus there is some θ' such that each variable sharing class of $T_j\theta'$ is a subset of some S_i. T_j is therefore *sat*-consistent. But by the construction of Θ_{sat}, we know that $sat(T_j) \simeq false$, thus contradicting our initial assumption that *sat* was reliable.

Therefore no T_j can be satisfied by any valuation in $\mathfrak{R}$; $\mathfrak{R}$ is therefore a model of S, and therefore a model of Θ_{sat}; thus Θ_{sat} is consistent. □

Reliability is a necessary and sufficient condition for satisfiability, but various conditions which appear at first glance to be equivalent are in fact not. Consider the following two conditions:

C1. If $sat(S) \simeq false$, then for every superset T of S, $sat(T) \simeq false$.

C2. If $sat(S) \simeq false$, then for every substitution θ, $sat(T\theta) \simeq false$.

C1–C2 together seem to be a strong condition, but in fact are incomparable to reliability. An example of a satisfiability function which meets C1–C2 but is not reliable is the minimal satisfiability function which maps $S = \{p(x), q(y)\}$ to *false*, every superset of S to *false*, every set $S\theta$ to *false*, but maps $\{p(3)\}$ and $\{q(4)\}$ to *true*. A satisfiability function which is reliable but does not meet C1–C2 is any one that maps one set with a free variable to *false* and all other sets to *unsure*. Now let us add a third condition:

C3. If $sat(S) \simeq false$, then for every variable-sharing class T of S, $sat(T) \simeq false$.

C1–C3 together imply reliability now, and in fact are properties we might reasonably expect of a satisfiability function in an implementation of a CLP language. But they are stronger than necessary, as shown by the second example function in the last paragraph.

2.4. Maximality results

Finally, some words about maximality. Given a particular constraint theory, what is the "biggest" satisfiability function which implements it? It turns out that either a theory has a unique, maximal, strict satisfiability function, or else there is no maximal satisfiability function which implements it.

Definition 2.13. For two satisfiability functions sat_1 and sat_2, we say that $sat_1 \sqsubseteq sat_2$ if:

1. whenever $sat_1(S) \simeq true$ we have that $sat_2(S) \simeq true$; and
2. whenever $sat_1(S) \simeq false$ we have that $sat_2(S) \simeq false$.

A satisfiability function *sat* is a *maximal implementation* of $\mathfrak{R}$ if it implements $\mathfrak{R}$, and there is no *sat'* which implements $\mathfrak{R}$ such that $sat' \neq sat$ and $sat \sqsubseteq sat'$.

Theorem 2.14. *A structure* $\mathfrak{R}$ *has either* 0 *or* 1 *maximal implementations.*

Proof. Consider the set of all finite, $\mathfrak{R}$-satisfiable sets of constraints. If this set is recursive, then clearly there is a unique, strict, maximal implementation of $\mathfrak{R}$. Otherwise, for any *sat* which implements $\mathfrak{R}$, there is either a finite, $\mathfrak{R}$-satisfiable set S of constraints such that $sat(S)\uparrow$ or $sat(S) \notin \{true, false\}$, or a finite, non-$\mathfrak{R}$-satisfiable set S of constraints such that $sat(S)\uparrow$ or $sat(S) \notin \{true, false\}$. In the first case, let *sat'* be the satisfiability function such that $sat'(S) \simeq true$ and $sat'(T) \simeq sat(T)$ for all T not identical to S such that $sat(T)\downarrow$. Then $sat \sqsubseteq sat'$. The second case is analogous. Thus from any *sat* which implements $\mathfrak{R}$, we can build a "bigger" *sat'* which also implements $\mathfrak{R}$; so there can be no maximal implementation of $\mathfrak{R}$. $\square$

Thus for the example structure given in the Introduction (extensional non-equivalence of lambda-expressions), there is no maximal implementation. Whenever we have a satisfiability function (and thus a constraint logic programming system) which implements this structure, we can always do better.

3. Specification and characterization with proof systems

We have seen that incomplete implementations of CLP languages are sometimes desirable or even necessary. We have also seen that such incomplete implementations can be given a coherent theoretical basis. But in order to make practical use of incomplete implementations, we need to be able to compare them directly with descriptions of theories.

To do so, we really need *formal* (syntactic) descriptions of the theories to be compared to; the kinds of informal descriptions found in the literature are sometimes too imprecise to be used in formal proofs, and this imprecision is multiplied when we have several groups of interacting constraints (Herbrand, rational tree, integer, real, etc.). There are at least two other practical reasons for developing formal descriptions of CLP languages:

- With increasing prominence of constraint systems, it will become necessary to develop some standard formalism for describing constraint theories, much as BNF was developed to describe programming language syntax.
- Syntactic, or more specifically logical, characterizations of constraint theories will be absolutely necessary to any program-logic system which intends to prove properties of constraint logic programs.

One possible framework for such formal descriptions is proof theory, which has been used to good advantage in the past to describe standard Herbrand-domain logic programming [8,9,17]. The use of proof theory as a general framework for characterizing CLP, which has not to my knowledge been studied before, is the topic of this section. I will present proof-theoretic characterizations here in the form of sequent calculi, both because they have a clear and natural logical interpretation, and because one of the characterizations involves modal relevance logic, which can be described most easily in a sequent-calculus form. The notion of satisfiability function will be used to build very general schemata of characterizing proof systems.

There are two ways in which proof-theoretic characterizations could be used:

(a) to *specify* an intended theory; and
(b) to *characterize* an existing implementation.

Examples of (a) would include giving an axiomatization of Horn clauses with Presburger arithmetic, to which we could then compare individual CLP implementations. Examples of (b) would include giving a proof-theoretic characterization of the CLP(R) implementation [13], to see how it looks compared to an axiomatization of real arithmetic. The examples I will discuss here will concern mostly characterizing existing systems, in order to introduce the subject with familiar material.

In this section, I will first present an enriched syntax of goals and clauses, and an operational semantics based on that syntax. Then I will discuss a simple proof system schema for characterizing CLP systems with strict, Henkin satisfiability functions. A theory is "Henkin" if it has a closed "witness" term for every existential truth, and the analogous property of satisfiability functions will be shown to be an important issue. I will then discuss a somewhat more complex proof system schema, still characterizing CLP systems with strict satisfiability functions, but with the Henkin restriction lifted. Finally, I will discuss a characterization of one important system with a non-strict satisfiability function: the simplified version of CLP(R) referred to earlier. Notions from substructural logic will be shown to be important here.

3.1. Goals, clauses, and operational semantics

We will find it convenient at this point to slightly expand Maher's framework [16] by enriching the syntax of goals and modifying the operational semantics to have more and simpler rules. Here I will present the enriched syntax, and the modified operational semantics.

This enriched syntax is based on Miller et al.'s syntax of goals and definitions for uniform-proof-based logic programming systems [17]. Goals:

$$G ::= c \mid A \mid G\&G \mid \exists x G$$

where c is a constraint and A is any predicate application formula (atom).

Clauses (definitions):

$$D ::= p(x_1, \ldots, x_n) \leftarrow G \mid \forall x D$$

We say that a *program* is a set of ground clauses. We assume that there is at least one satisfiable contraint T; what would be an atom in a Prolog program would be a clause with body T. (This is a reasonable assumption for all practical purposes.)

The expanded syntax of goals and definitions allows us to modify Maher's operational semantics, to describe the actions associated with different forms of goals. This will have the effect of simplifying the individual rules and facilitating the proofs of characterization.

Definition 3.1. The operational semantics $\mathrm{OS}_{P,sat}$ for a given realizable satisfiability function *sat* and program P is given by the $\Rightarrow_{P,sat}$ relation between states, defined as follows. (Γ is a multiset of goals, and S is a set of constraints. (G, Γ) stands for the multiset union of $\{G\}$ and Γ.)

- Con (constraint): $\langle (c, \Gamma), S\rangle \Rightarrow_{P,sat} \langle \Gamma, S \cup \{c\}\rangle$ where $sat(S)\downarrow$ and $sat(S \cup \{c\}) \not\simeq false$
- Fail: $\langle (c, \Gamma), S\rangle \Rightarrow_{P,sat} false$ where $sat(S \cup \{c\}) \simeq false$
- & (conjunction): $\langle (G_1 \& G_2, \Gamma), S\rangle \Rightarrow_{P,sat} \langle (G_1, G_2, \Gamma), S\rangle$
- $\exists$ (existential): $\langle (\exists x G_1, \Gamma), S\rangle \Rightarrow_{P,sat} \langle (G_1[x := y], \Gamma), S\rangle$ where y is a variable not appearing on the left
- Def (defined predicate): $\langle (A\theta, \Gamma), S\rangle \Rightarrow_{P,sat} \langle (G\theta, \Gamma), S\rangle$ where P contains (the universal closure of) the clause $A \leftarrow G$

We again say that a state S succeeds if $\mathrm{S} \Rightarrow^*_{P,sat} \langle \varepsilon, S\rangle$, where ε is the empty multiset of goals, that S fails if all fair computations from S lead to *false*, and that a given goal G succeeds (fails) if $\langle G, \emptyset\rangle$ succeeds (fails).

3.2. *Systems with strict, Henkin satisfiability functions*

Strict satisfiability have pleasant properties which we can exploit to provide simple, general proof-theoretic characterizations of the systems they are based on. Satisfiability functions that also have the "Henkin" property, such as those of Herbrand-domain Prolog and Presburger arithmetic, have even simpler characterizations. In this section, I present (with examples and proofs) a proof system schema which characterizes systems with strict, Henkin satisfiability functions.

A *Henkin* theory [22] is one in which, for every sentence $\exists xA$ of the language, there is a constant e in the language such that $T \models (\exists xA) \supset (A[x := e])$. In analogy with this notion, I give a definition of a Henkin satisfiability function.

Definition 3.2. A satisfiability function *sat* is *Henkin* if, whenever $sat(S) \simeq true$, there is a substitution θ of ground terms for variables such that $S\theta$ is ground and $sat(S\theta) \simeq true$.

Many useful satisfiability functions are Henkin; others, such as those for rational trees and real arithmetic, are not. For those satisfiability functions which are strict and Henkin, there is a particularly simple characterizing proof stystem.

Characterizing proof system

Definition 3.3. SH_{sat} is defined by taking the rules B given in Fig. 1 and adding a rule:

$$\text{Con/r: } \overline{\Gamma \vdash c}$$

for any closed constraint c such that $sat(\{c\}) = true$.

The result we will be concerned with about SH_{sat} is that it characterizes $\mathrm{OS}_{P,sat}$ in the following sense:

For any closed goal formula G, program P and strict, realizable, Henkin satisfiability function sat, we have that $\langle G, \emptyset \rangle$ succeeds with respect to $\mathrm{OS}_{P,sat}$ iff the sequent $P \vdash G$ is derivable in SH_{sat}.

The reason we are interested in this result is, of course, that SH_{sat} is a simple, clear and logical description of the meanings of goals and programs. The rules B in Fig. 1 are almost exactly those of Gentzen's original sequent calculus LJ [6], and the additional rules $\Gamma \vdash c$ capture the notion of truth according to the satisfiability function sat.

I will prove this characterization result in an appendix; here I will just give some examples.

Example (*Herbrand-domain logic programming*). In the language of this system, the single constraint predicate is $=$, equality. The function and predicate symbols can be anything, as long as there is at least one nullary function symbol. (Otherwise sat is not Henkin; for instance, $\{x = x\}$ is true but there is no closed s such that $\{s = s\}$ is true.) The satisfiability function sat is defined as the one which is true of a set S iff there is some substitution θ such that $s\theta$ is identical to $t\theta$ for all formulae $s = t \in S$; that is, sat performs standard unification.

$$\&/\text{r: } \frac{\Gamma \vdash G_1 \quad \Gamma \vdash G_2}{\Gamma \vdash G_1 \& G_2} \qquad \exists/\text{r: } \frac{\Gamma \vdash G[x := t]}{\Gamma \vdash \exists x(G)}$$

$$\text{Dup/l: } \frac{D, D, \Gamma \vdash G}{D, \Gamma \vdash G} \qquad \forall/\text{l: } \frac{D[x := t], \Gamma \vdash G}{\forall x D, \Gamma \vdash G}$$

$$\leftarrow/\text{l: } \frac{\Gamma \vdash G}{(p(t_1, \ldots, t_n) \leftarrow G), \Gamma \vdash p(t_1, \ldots, t_n)}$$

Fig. 1. Basic rules B for characterizing a constraint logic programming language. Γ is a multiset of formulae.

The characterizing proof system can be given by adding to B the single rule

$$\frac{}{\Gamma \vdash s = s}$$

for any closed s, since these constitute the closed true constraint formulae of *sat*.

The resulting proof system is more or less that given by Andrews [2], adapted to Miller et al.'s sequent calculus style [17]. Examples of its use can be found in [2].

Example (*Presburger arithmetic logic programming*). The single constraint predicate is $=$, equality. The function symbols consist of the nullary 0, the unary s, and the binary $+$. *sat* is the function which can be derived from Presburger's algorithm for satisfaction of arithmetic formulae over s and $+$ [18].

The characterizing proof system can be given by adding to B the rule

$$\frac{}{\Gamma \vdash c}$$

where c is a closed constraint such that $sat(\{c\}) \simeq true$. However, we can give an equivalent proof system which does not depend on a reference to *sat* by adding to B the rules in Fig. 2. These rules are derived from Kleene's Hilbert-style system for Presburger arithmetic [14], and the proof that the systems are equivalent can be derived from Kleene's proof of completeness. Note that this proof system (and all the proof systems I discuss here) is quite fragile: the addition of any more rules may require a different satisfiability function, or may prevent us from having a strict satisfiability function altogether.

An example of a derivation is SH_{sat} for this satisfiability function is given in Fig. 3. Readers are invited to do the corresponding computation in OS_{sat}.

3.3. *Systems with general, strict satisfiability functions*

Not all useful satisfiability functions (or constraint theories) are Henkin, not even the strict ones. A simple example is the theory of rational trees: we have that $\exists x(x = f(x))$ is true, but there is no closed t such that $t = f(t)$. Another important example is the theory of real arithmetic, in which we have no closed term t such that $t \times t = 2$. For these theories, we have to use other methods of characterization.

Characterizing proof system: The characterizing proof system schema for strict satisfiability functions in general allows for the introduction of elements from the theory associated with *sat* on the left-hand side of the sequent, and allows us to reason about them.

$$\frac{\Gamma \vdash a = b}{\Gamma \vdash b = a} \qquad \frac{\Gamma \vdash a = b \qquad \Gamma \vdash b = c}{\Gamma \vdash a = c} \qquad \frac{\Gamma \vdash a = b}{\Gamma \vdash s(a) = s(b)}$$

$$\frac{}{\Gamma \vdash a + 0 = a} \qquad \frac{}{\Gamma \vdash a + s(b) = s(a + b)}$$

Fig. 2. Additional rules for a proof-theoretic characterization of a Presburger arithmetic constraint logic programming language.

Definition 3.4. Proof system S_{sat} is formed by taking the rules B from Fig. 1 and adding the rules in Fig. 4.

This new proof system schema is more complex than SH_{sat}, but does not restrict us to looking at only Henkin satisfiability functions. We can prove the same kind of thing about S as we did about SH:

For any closed goal formula G, program P and strict, realizable satisfiability function *sat*, we have that $\langle G, \emptyset \rangle$ succeeds with respect to $\Rightarrow_{P,sat}$ iff $P \vdash G$ is derivable in S_{sat}.

Again, the rules added are almost exactly those of Gentzen's LJ [6] except for the Con/l rules; the Con/l rules capture the notion of satisfiable set of constraints given by *sat*.

Example (*Rational tree logic programming*). In this CLP system, the single constraint predicate is $=$, equality. There are no restrictions on the function symbols. *sat* is the function which returns *true* iff its set of equations can be reduced to a set of equations in Colmerauer's "solvable form" [3]. van Emden and Lloyd [23] and Maher [15] have given Hilbert-style logical descriptions of the structure which realizes this *sat*; the fact that this *sat* is strict follows from Colmerauer's proof that his reduction algorithm terminates [4].

$$\begin{array}{cl}
\overline{P \vdash 2 = 2} & \text{(Con/r)}\\
\overline{(pred(2,1) \leftarrow 2 = 2), P \vdash pred(2,1)} & (\leftarrow/\mathrm{l})\\
\overline{\forall y(pred(2,y) \leftarrow 2 = s(y)), P \vdash pred(2,1)} & (\forall/\mathrm{l})\\
\overline{\forall x \forall y(pred(x,y) \leftarrow x = s(y)), P \vdash pred(2,1)} & (\forall/\mathrm{l})\\
\overline{P \vdash pred(2,1)} \qquad \overline{P \vdash 2+1=3} & \text{(Dup/l, Con/r)}\\
\overline{P \vdash (pred(2,1) \& 2+1=3} & (\&/\mathrm{r})\\
\overline{P \vdash \exists y(pred(2,y) \& 2+y=3)} & (\exists/\mathrm{r})\\
\overline{P \vdash \exists x \exists y(pred(x,y) \& x+y=3)} & (\exists/\mathrm{r})
\end{array}$$

Fig. 3. Example derivation in SH_{sat} for the Presburger arithmetic satisfiability function. P is the program consisting of the single clause $\forall x \forall y(pred(x,y) \leftarrow x = s(y))$. To save space, I write 1, 2, 3 instead of $s(0)$, $s(s(0))$, $s(s(s(0)))$.

$$\text{Con/l:}\ \frac{\exists[S], \Gamma \vdash G}{\Gamma \vdash G} \qquad \exists/\text{l:}\ \frac{B[x := y], \Gamma \vdash G}{\exists x B, \Gamma \vdash G}$$

$$\&/\text{l:}\ \frac{B, C, \Gamma \vdash G}{B \& C, \Gamma \vdash G} \qquad \text{Axiom:}\ \frac{}{c, \Gamma \vdash c}$$

Fig. 4. Additional rules for S_{sat}, a proof-theoretic characterization of a CLP language with a strict satisfiability function. S is any set of constraints such that $sat(S) \simeq true$, and y is a new variable.

$$\begin{array}{cl}
\dfrac{\overline{z'=f(g(z'))\vdash z'=f(g(z'))} \qquad \overline{z'=f(g(z'))\vdash g(z')=g(z')}}{z'=f(g(z'))\vdash z'=f(g(z'))\&g(z')=g(z')} & \begin{array}{l}\text{(Axioms)}\\ (\&/\mathrm{r})\end{array} \\
\dfrac{}{z'=f(g(z'))\vdash \exists y(z'=f(y)\&y=g(z'))} & (\exists/\mathrm{r}) \\
\dfrac{}{z'=f(g(z'))\vdash \exists x\exists y(x=f(y)\&y=g(x))} & (\exists/\mathrm{r}) \\
\dfrac{}{\exists z(z=f(g(z)))\vdash \exists x\exists y(x=f(y)\&y=g(x))} & (\exists/\mathrm{l}) \\
\dfrac{}{\vdash \exists x\exists y(x=f(y)\&y=g(x))} & (\mathrm{Con}/\mathrm{l})
\end{array}$$

Fig. 5. Example derivation in the augmented S_{sat} for rational trees. P is in this case the empty program.

As in the case for Presburger arithmetic, the unmodified proof system S_{sat} may be somewhat clumsy to work with. We can simplify the proof system by adding rules such as

$$\overline{\Gamma\vdash t=t}$$

where t is any term, and

$$\frac{\Gamma\vdash s_1=t_1 \quad \cdots \quad \Gamma\vdash s_n=t_n}{\Gamma\vdash f(s_1,\ldots,s_n)=f(t_1,\ldots,t_n)}$$

where f is any function symbol. It is easy to prove that these are admissible rules of the original system. Fig. 5 gives an example derivation of a simple goal in this augmented S_{sat}.

Note that we can in fact restrict the proof system S_{sat} somewhat and still obtain the same soundness and comleteness results. If a set of constraints S such that $sat(S)\simeq true$ contains disjoint variable-sharing classes $S|_x$ and $S|_y$, then $sat(S|_x)\simeq sat(S|_y)\simeq true$, and we can assert $\exists[S|_x]$ and $\exists[S|_y]$ as separate assumptions. Also, if for a set S such that $sat(S)\simeq true$ we have a substitution θ mapping variables to closed terms such that $sat(S\theta)\simeq true$, then clearly any derivation that asserts $\exists[S]$ could instead assert $\exists[S\theta]$. Thus we can restrict the side-condition on the assumption introduction rule to

> where S is a set of constraints such that $sat(S)\simeq true$, for all variables x free in S we have $S|_x=S$, and there is no variable x and closed term t such that $sat(S[x:=t])\simeq true$.

3.4. Non-strict satisfiability functions: CLP(R)

We were able to characterize CLP systems with strict satisfiability functions with the very general proof system schemata SH_{sat} and S_{sat} via the least operational transition relation for *sat*. However, CLP systems with non-strict satisfiability functions can have many different operational semantics; therefore there can be no one proof system schema which characterizes them. In this subsection, therefore, I will discuss a characterization of only one example system, CLP(R), and suggest how the knowledge of this characterization could be applied to other systems.

The characterizing proof system for CLP(R) is best described as a modal relevance logic, in that it uses a standard modal operator $\square$ ("necessitation") and standard relevance logic methods for requiring that assumptions are relevant to conclusions. As such, it is closely related to Girard's linear logic [7], although the system which it is closest to is the "calculus of relevance and necessity" $R^{\square}$ described by Read [19]. In this section, I will present the proof system and its use, give a mechanical and a philosophical justification for the use of the modal relevance logic framework, and discuss how this result may generalize to other non-strict CLP systems.

Characterizing proof system: The characterizing proof system for CLP(R) is presented in Fig. 6. It is called MR_{clpr}, where *clpr* is the satisfiability function for CLP(R) described earlier in this paper. It is very similar to the S_{sat} schema, since it also uses the idea of introducing the existential closure of a satisfiable set on the left, and drawing inferences from it. However, there are a few subtle differences.

1. In the &/r rule, we are allowed to "split" the left-hand side of the conclusion between the left and right premiss.

2. In the $\exists$/r rule, we are required to use a *variable*, not just any term, as a witness to the existentially quantified variable. This variable does not have to be new, however.

3. In the Axiom, the formulae on the left-hand side, other than the constraint which matches the right-hand side, must all be formulae preceded by the necessitation operator $\square$.

4. There is a new rule, Nec/l, which states that if G can be derived from D and Γ, then it can be derived using the stronger assumption that D is necessarily true.

The combined effect of these rules is that any formula appearing on the left *without* a $\square$ preceding it must be used, somewhere in the derivation, in an Axiom; that is, all the assumptions must be relevant to the conclusion.

The characterization theorem is also subtly different from that of S; note the presence of the $\square$ operator.

For any closed goal formula G and program P, we have that $\langle G, \emptyset \rangle$ succeeds with respect to $\Rightarrow_{P,clpr}$ iff $\square P \vdash G$ is derivable in MR_{clpr}.

$$\&/\mathrm{r}: \ \frac{\Gamma_1 \vdash G \qquad \Gamma_2 \vdash H}{\Gamma_1, \Gamma_2 \vdash G \& H} \qquad \exists/\mathrm{r}: \ \frac{\Gamma \vdash G[x := z]}{\Gamma \vdash \exists x(G)} \qquad \text{Axiom}: \ \frac{}{c, \square\Gamma \vdash c}$$

$$\text{Dup/l}: \ \frac{D, D, \Gamma \vdash G}{D, \Gamma \vdash G} \qquad \text{Nec/l}: \ \frac{D, \Gamma \vdash G}{\square D, \Gamma \vdash G}$$

$$\forall/\mathrm{l}: \ \frac{D[x := t], \Gamma \vdash G}{\forall x D, \Gamma \vdash G} \qquad \leftarrow/\mathrm{l}: \ \frac{\Gamma \vdash G}{(p(t_1, \ldots, t_n) \leftarrow G), \Gamma \vdash p(t_1, \ldots, t_n)}$$

$$\text{Con/l}: \ \frac{\exists[c_1 \& \cdots \& c_k], \Gamma \vdash G}{\Gamma \vdash G} \qquad \exists/\mathrm{l}: \ \frac{B[x := y], \Gamma \vdash G}{\exists x B, \Gamma \vdash G} \qquad \&/\mathrm{l}: \ \frac{B, C, \Gamma \vdash G}{B \& C, \Gamma \vdash G}$$

Fig. 6. The proof system schema MR_{clpr} characterizing the behaviour of CLP(R). Γ is any sequence of formulae, $\square\Gamma$ is a sequence of formulae preceded by $\square$, y is a new variable, z is any variable, and $c_1, \ldots, c_k$ are constraints such that $clpr(\{c_1, \ldots, c_k\}) \simeq true$.

In the theorem, we use the following notation:

Notation 3.5. $\Box P$, where P is a set of formulae $\{B_1, \dots, B_n\}$, denotes the set of formulae $\{\Box B_1, \dots, \Box B_n\}$.

Thus when we use the proof system, we will always place the clauses of the program on the left-hand side, preceded by the operator which states each clause is necessarily true. The clauses can then be duplicated, the $\Box$ removed via the Nec/l rule, and then instantiated as before. However, when we strip off the $\Box$, we must eventually *use* the assumption. Furthermore, all assumptions arising from the use of the Con/l rule must be used as well, since the assumption introduced does not come with a preceding $\Box$.

Example. An example derivation in MR_{clpr} is given in Fig. 7. P, in this example, is the program consisting of the single clause

$$\forall z(p(z) \leftarrow z = 2)$$

$\Box P$ is therefore

$$\Box\forall z(p(z) \leftarrow z = 2)$$

In the derivation, the first steps (reading up from the bottom) introduce on the left-hand side the satisfiable set of constraints that we will need to prove the result, and break it down via the $\exists$/l and &/l rules. The next steps instantiate the existential variables on the right to refer to the variables in the satisfiable set. There is then an &/r step, which splits the list of assumptions so that the precise set of assumptions needed is transferred to each side. On the right-hand branch, there is then a standard sequence of proving a predicate call goal by duplicating and instantiating a clause of the program.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x'\cdot y' = 4 \vdash x'\cdot y' = 4} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y' = 2, \Box P \vdash y' = 2}\ (\text{Axiom})}{y' = 2, \Box P, (p(y') \leftarrow y' = 2) \vdash p(y')}\ (\leftarrow/\text{l})}{y' = 2, \Box P, \forall z(p(z) \leftarrow z = 2) \vdash p(y')}\ (\forall/\text{l})}{y' = 2, \Box P, \Box\forall z(p(z) \leftarrow z = 2) \vdash p(y')}\ (\text{Nec/l})}{y' = 2, \Box P \vdash p(y')}\ (\text{Axiom, Dup/l})}{x'\cdot y' = 4, y' = 2, \Box P \vdash x'\cdot y' = 4 \& p(y')}\ (\&/\text{r})}{x'\cdot y' = 4, y' = 2, \Box P \vdash \exists y(x'\cdot y = 4 \& p(y))}\ (\exists/\text{r})}{x'\cdot y' = 4, y' = 2, \Box P \vdash \exists x \exists y(x\cdot y = 4 \& p(y))}\ (\exists/\text{r})}{(x'\cdot y' = 4 \& y' = 2), \Box P \vdash \exists x \exists y(x\cdot y = 4 \& p(y))}\ (\&/\text{l})}{\exists y'(x'\cdot y' = 4 \& y' = 2), \Box P \vdash \exists x \exists y(x\cdot y = 4 \& p(y))}\ (\exists/\text{l})}{\exists x' \exists y'(x'\cdot y' = 4 \& y' = 2), \Box P \vdash \exists x \exists y(x\cdot y = 4 \& p(y))}\ (\exists/\text{l})}{\Box P \vdash \exists x \exists y(x\cdot y = 4 \& p(y))}\ (\text{Con/l})
$$

Fig. 7. An example derivation in MR_{clpr}, the characterizing proof system for CLP(R).

Note that the derivation would not have gone through if the query had been simply $\exists x, y(x \cdot y = 4)$, because we would have had the extra formula $y' = 2$ to contend with. Conversely, if we had allowed axioms to be simply of the form $(c, \Gamma \vdash c)$, we would have been able to prove $\exists x, y(x \cdot y = 4)$, even though that is not a linear equation and thus not solvable in the operational semantics.

Mechanical justification of MR_{clpr}: One way of seeing why a modal relevance logic is required for this system is to look at the mechanics of proving formulae.

- The restricted Axiom forces each constraint assumption to be used at least once, as mentioned above.
- The ability to split the axiom list in the &/r rule is then a necessary adjunct, because if &/r were more like the rule in S_{sat}, we would essentially be required to duplicate every assumption every time we used it; each of the duplicate constraint assumptions would then have to be used.
- We sometimes do want the ability to duplicate constraint assumptions, however. For instance, if the query is $\exists x(x = 2 \& x = 2)$, we will need to prove it by introducing $\exists x(x = 2)$ on the right:

$$\begin{array}{cl}
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{x' = 2 \vdash x' = 2} \quad \overline{x' = 2 \vdash x' = 2}}{x' = 2, x' = 2 \vdash x' = 2 \& x' = 2}}{x' = 2, x' = 2 \vdash \exists x(x = 2 \& x = 2)}}{x' = 2 \vdash \exists x(x = 2 \& x = 2)}}{\exists x(x = 2) \vdash \exists x(x = 2 \& x = 2)}}{\vdash \exists x(x = 2 \& x = 2)} & \begin{array}{l}(\text{Axiom})\\(\&/\text{r})\\(\exists/\text{r})\\(\text{Dup/l})\\(\exists/\text{l})\\(\text{Con/l})\end{array}
\end{array}$$

  It is because of this need to duplicate constraint assumptions that *relevance* logic rather than *linear* logic is necessary. In linear logic, assumptions without ! (the analogue of $\Box$) not only must be used at least once, they cannot be duplicated (i.e. they must be used exactly once). Here, we want to use non-$\Box$ assumptions one *or more* times, so the Dup/l rule is not restricted as it would be in linear logic.
- Finally, although we want constraint assumptions to be used at least once, we want to put no such restriction on the program clauses. In a given query derivation, any given program clause may be used once, more than once, or not at all; we therefore adopt the solution (used by the Lolli system, among others [10]) of "protecting" the program clauses with $\Box$. The program clauses can then be used 0 or more times, and can end up in Axioms without having been used.

I should also mention the mechanical reason for the restriction in the $\exists$/r rule to only variables as witness terms. Without this restriction, we would be able to substitute any term for the variable in the $\exists$/r rule; so for instance, $\exists x \exists y(x \cdot y = 4)$ would be easy to prove from the satisfiable constraint set $2 \cdot 2 = 4$. Because the constraints arising from the expansion of the existential formula must eventually be matched with formulae on the right-hand side, the restriction effectively means that the witness term must be a variable already appearing on the left-hand side. This forces the Axioms to always have the same form as they would in the computation, correctly forbidding us from substituting satisfying terms that would not be found by CLP(R).

Philosophical justification: There is also a more philosophical exegesis of the rules of MR_{clpr} which does not depend merely on showing mechanically how a derivation can or cannot be built. It hinges on the logico-philosophical notions of necessary and contingent fact, relevant implication, and universe of discourse, and explains more fully why MR_{clpr} has a modal relevance logic form.

A *necessary* fact is one which we take as logically unavoidable, and true in all situations we want to consider. By placing program clauses on the left-hand side of the sequent with the necessitation operator $\Box$ in front of them, we are stating that those clauses are necessarily true, regardless of the particular goal formula we want to prove.

A *contingent* fact is one which happens to be the case in a particular circumstance, but is not always true. By allowing the introduction of satisfiable constraint sets on the left *without* the necessitation operator, we are stating that at any particular time, we are free to assume these sets as contingent facts, but that they are not the case in all circumstances.

By *relevant implication* from a set of necessary and contingent facts to a conclusion, we mean (following the standard references [1,19]) a deduction of the conclusion from the facts in which all aspects of all the contingent facts are in some way *used* to prove the conclusion. The study of relevance logic was initially undertaken in order to disallow dubious implications such as "if the moon is made of green cheese, then $2 + 2 = 4$" (which are true with the standard "material" implication of classical and intuitionistic logic); the proof systems that have arisen to formalize the notion of relevant implication are all essentially sequent-calculus systems with forms very similar to that of MR_{clpr}.

Finally, when we restrict the $\exists$/r rule to only variables, we are essentially saying that our universe of discourse consists not only of the real numbers, but of a number of other elements which may stand in the equality relation to real numbers or which may be of unknown value. The existential quantifier is taken to range over this set of additional elements, not the real numbers themselves; the constraint sets S such that $clpr(S) \simeq true$ are taken to state the relationship between the additional elements and the real numbers.

Application to other systems: Are the techniques used to characterize CLP(R) applicable to other systems? Only some properties of the CLP(R) satisfiability function, which I have been calling *clpr*, are needed for the proofs of soundness and completeness, and any system having a satisfiability function *sat* with these properties will suffice. As the name MR_{clpr} implies, we could have MR_{sat} for any such *sat*.

The properties that *clpr* has that allow the proofs of soundness and completeness to go through are as follows:

1. *clpr* always terminates and always returns one of *true*, *false* or *unsure*.
2. $clpr(\emptyset) \simeq true$.

3. If $clpr(S) \simeq clpr(T) \simeq true$, then if θ is a substitution which renames variables of T apart from those appearing in S, then $clpr(S \cup T\theta) \simeq true$. This is because the requirement that a set of formulae be linear and solvable is maintained under union of sets with disjoint sets of variables.

4. If $clpr(S\theta) \simeq true$, where θ is a substitution mapping variables to other variables, then $clpr(S) \simeq true$. (For instance, $clpr(\{x \cdot x = 4\})$ must return *unsure* because $clpr(\{x \cdot y = 4\})$ does.) This is the case again because the requirements that a set of formulae be linear and solvable are not affected by a substitution of one variable by another.

While these properties seem reasonable, it is not clear how many useful systems have them. In particular, the fourth property seems problematic; one can easily imagine a satisfiability function in which some S is of *unsure* satisfiability, but $S[x := y]$ is considered definitely satisfiable.

However, it may be sufficient to assume that a function which has properties 1–3 is used to determine satisfiability during the normal course of computation, but a function with property 4 is used at the end of the computation to determine the satisfiability of the final set of constraints. I have not explored this idea in any detail, since it seems to complicate the theory unnecessarily. The characterization of CLP(R), one of the most important constraint logic programming languages, seems to sufficiently motivate the study of MR_{clpr}.

4. Conclusions, related and future work

I have shown that the class of satisfiability functions adequately characterizes the behaviour of a wide variety of implementations of CLP languages, and that there is a simple, non-model-theoretic condition ("reliability") for testing whether a satisfiability function is reasonable.

I have also discussed techniques for specifying and characterizing CLP systems with sequent calculi. I have pointed out that the question of whether the theory is Henkin is important, and that the notation and proof theory of substructural logics can help in these characterizations.

The definition of a reliable satisfiability function is closely related to Scott's definition of an *information system* [21]. Saraswat [20] uses information systems to describe the domains of constraint systems. However, neither the space of satisfiability functions, nor the space of information systems (under a reasonable mapping from one notion to the other), are proper subsets of the other.

Höhfeld and Smolka [11] and Frühwirth [5] have both explored the idea of formally describing constraint theories. Höhfeld and Smolka describe an alternative framework to Jaffar and Lassez's for constraint systems; like Jaffar and Lassez, they do not consider explicitly any computability restrictions on constraint satisfaction algorithms. Frühwirth gives a Horn-clause-based language for defining constraint simplification rules, or SiRs, for any given domain. While SiRs have a logical form, they do not necessarily take the form of a simple and intuitive axiomatization or proof system.

There are several directions for future work in this area:

- Case studies. I would very much like to see these ideas applied for the purpose of fully and precisely characterizing existing, practical systems.
- Negation. I have avoided talking about negation in this paper because it poses general problems for logic programming theory which have not been adequately answered yet. A framework which characterizes the failure of constraint queries as well as their success would be desirable.
- "Ask" constraints. I have considered here only constraints that would be described by Saraswat [20] as "tell" constraints. I would also like to characterize "ask" constraints; this may involve allowing some description of implication in characterizing proof systems. (Thanks to Andreas Podelski for interesting discussions on this point.)
- Moving toward a standard description language. It would be premature at this point to propose some standard for describing constraint systems, but this would bring many benefits if done, much as BNF brought a standard manner of describing programming language syntax.

Acknowledgements

I appreciate the helpful comments and suggestions I have received from Veronica Dahl, Alistair Lachlan, Sanjeev Mahajan, Fred Popowich, Stephan Wehner (all of SFU), Thom Frühwirth, Nevin Heintze, Tim Hickey and Gert Smolka, as well as Torkel Franzen and the anonymous referees. This research has been supported by the Natural Sciences and Engineering Research Council of Canada, the SFU Centre for Systems Science, and the SFU President's Research Grant Committee, via Infrastructure Grants NSERC 06-4231, CSS 02-7960 and PRG 02-4028, Equipment Grants NSERC 06-4232, CSS 02-7961 and PRG 02-4029, Operating Grants OGP0002436 (Dahl) and OGP0041910 (Popowich), and the author's NSERC Postdoctoral Fellowship.

Appendix A. Proofs of characterization theorems

A.1. SH characterizes strict, Henkin systems

We will prove the SH characterization result by proving two very general theorems, one about the soundness of the operational semantics OS_{sat} with respect to the SH_{sat} proof system, and the other about its completeness.

Theorem A.1 (Soundness of OS with respect to SH). *Let sat be a realizable, strict, Henkin satisfiability function. Let $G_1, \ldots, G_n$ be goals, and $S = \{c_1, \ldots, c_m\}$ be a set of constraints. If $\langle (G_1, \ldots, G_n), S\rangle \Rightarrow^*_{P,sat} \langle \varepsilon, T\rangle$ and $sat(T) \simeq true$, then there is some substitution θ such that $P \vdash G_i\theta$ and $P \vdash c_j\theta$ in SH_{sat}, for all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$.*

Proof. By induction on the number of steps in the computation.

- 0 steps: In this situation, $n = 0$ and $S = T$. By the Henkinness of *sat*, there must be some θ which closes all of the T formulae such that each $\{c_j\theta\}$ is considered satisfiable by *sat*. By the Con/r rule of SH, we must have that $P \vdash c_j\theta$ for each c_j.
- >0 steps: cases on the first rule used in the computation. Let Γ be the goals $G_2, \ldots, G_n$.
  - $\langle (c, \Gamma), S \rangle \Rightarrow_{P,sat} \langle \Gamma, S \cup \{c\} \rangle$: directly from the induction hypothesis.
  - $\langle (G\&H, \Gamma), S \rangle \Rightarrow_{P,sat} \langle (G, H, \Gamma), S \rangle$: From the induction hypothesis, we have that for some θ, $P \vdash G\theta$ and $P \vdash H\theta$; thus by using the & rule of SH, we have that $P \vdash (G\&H)\theta$. In this and all subsequent cases, the fact that the Γ and S formulae are derivable carries over directly from the induction hypothesis.
  - $\langle (\exists x G, \Gamma), S \rangle \Rightarrow_{P,sat} \langle (G[x := y], \Gamma), S \rangle$: From the induction hypothesis, we know that for some θ, $P \vdash G[x := y]\theta$. Choose the term t to be $y\theta$; then $P \vdash G[x := t]\theta$. By the $\exists$ rule of SH, we have that $P \vdash (\exists x G)\theta$.
  - $\langle (A\theta', \Gamma), S \rangle \Rightarrow_{P,sat} \langle (G\theta', \Gamma), S \rangle$: From the induction hypothesis, we have that for some θ, $P \vdash G\theta'\theta$. We therefore have the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P \vdash G\theta'\theta}{(A \leftarrow G)\theta'\theta, P \vdash A\theta'\theta}\ (\leftarrow/1)}{(\forall x_n(A \leftarrow G))\theta'\theta, P \vdash A\theta'\theta}\ (\forall/1)}{\vdots}}{\forall x_1 \cdots \forall x_n(A \leftarrow G), P \vdash A\theta'\theta}\ (\forall/1)}{P \vdash A\theta'\theta}\ (\text{dupl}/1)$$

G_1 is just $A\theta'$, so $P \vdash G_1\theta$. □

For the converse direction (completeness), it is useful to define the notion of meta-derivability. This will allow us to give an inductive proof of completeness.

Definition A.2 (*Derivative, meta-derivability*). A multiset of formulae Γ' is an *SH-derivative* of a multiset Γ if $\Gamma \vdash G$ can be derived from $\Gamma' \vdash G$ by a sequence of SH Dup/l and $\forall$/l rules:

$$\dfrac{\dfrac{\Gamma' \vdash G}{\vdots}}{\Gamma \vdash G}$$

A formula G is *meta-derivable* from a multiset Γ of formulae in SH if there is some SH-derivative Γ' of Γ such that $\Gamma' \vdash G$ is derivable in SH.

Of course, G is meta-derivable from P iff $P \vdash G$ is derivable. However, phrasing results in terms of meta-derivability rather than derivability will sometimes allow us to prove them using very simple reasoning about the Con/l and $\forall$/l cases. This is the case for the completeness theorem.

Theorem A.3 (Completeness of OS with respect to SH). *Let sat be a realizable, strict, Henkin satisfiability function. Let $G_1, \ldots, G_n$ be a sequence of goals, $S = \{c_1, \ldots, c_m\}$ be a set of constraints, and P be a program. If there is some substitution θ such that $G_1\theta \ldots G_n\theta$ and $c_1\theta \ldots c_n\theta$ are all meta-derivable from P in SH_{sat}, then there is a T such that $\langle (G_1, \ldots, G_n), S\rangle \Rightarrow^*_{P,sat} \langle \varepsilon, T\rangle$ and $sat(T) \simeq true$.*

(Intuition: if $G_1, \ldots, G_n$ are satisfiable according to the proof system, and so are $c_1, \ldots, c_m$, then the operational semantics can compute the fact that they are. We are really only interested in the case where $n = 1$ and $S = \emptyset$, but we need the more general statement to do the induction.)

Proof. If $n = 0$, then each $c_j\theta$ must be a closed constraint such that $sat(\{c_j\theta\}) \simeq true$ (there is no other way they could be meta-derivable). By the reliability and strictness of *sat*, we must have that $sat(S) \simeq true$; we simply choose T to be S.

Otherwise, each $G_i\theta$ is meta-derivable from P; that is, there is some derivative Π_i of P such that $\Pi_i \vdash G_i\theta$. We proceed by induction of the total size of all the derivations $\Pi_i \vdash G_i\theta$. Cases are on the last rule used in the derivation of $\Pi_1 \vdash G_1\theta$. Let $\Gamma = G_2, \ldots, G_n$.

- $G_1 \equiv (G\&H)$, and the last rule application is of the form

$$\frac{\Pi_1 \vdash G\theta \qquad \Pi_1 \vdash H\theta}{\Pi_1 \vdash (G\&H)\theta}$$

  From the & rule of the operational semantics, we know that $\langle ((G\&H), \Gamma), S\rangle \Rightarrow_{P,sat} \langle (G, H, \Gamma), S)\rangle$; but by the induction hypothesis, we know that for some T, $\langle (G, H, \Gamma), S\rangle \Rightarrow^*_{P,sat} \langle \varepsilon, T\rangle$.

- $G_1 \equiv \exists x G$, and the last rule application is of the form

$$\frac{\Pi_1 \vdash G[x := t]\theta}{\Pi_1 \vdash (\exists x G)\theta}$$

  We choose some y not free in any G_i or c_j. From the $\exists$ rule of the operational semantics, we know that $\langle (\exists x G, \Gamma), S\rangle \Rightarrow_{P,sat} \langle (G[x := y], \Gamma), S\rangle$. But then $G[x := y]$, the Γ formulae, and the S formulae are all meta-derivable from P under the substitution $[y := t]\theta$, so by the induction hypothesis, there is a T such that $\langle (G[x := y], \Gamma), S\rangle \Rightarrow^*_{P,sat} \langle \varepsilon, T\rangle$.

- G_1 is a constraint, and the last rule used is Con/r: $G_1\theta$ must be closed constraint on which *sat* returns *true*. Since $c_1\theta \ldots c_m\theta$ are also such constraints, and *sat* is realizable and strict, we know that $sat(\{c_1, \ldots, c_m, G_1\}) \simeq true$. By the Con rule of the operational semantics, we know that $\langle (G_1, \Gamma), S\rangle \Rightarrow_{P,sat} \langle \Gamma, S \cup \{G_1\}\rangle$; but then by the induction hypothesis, we know that there is a T such that $\langle \Gamma, S \cup \{G_1\}\rangle \Rightarrow^*_{P,sat} \langle \varepsilon, T\rangle$.

- The last rule used is either $\forall$/l or Dup/l: the left-hand side of the upper sequent is also a derivative of P, so the result follows directly from the induction hypothesis.

- G_1 is some atom $p(t_1,\dots,t_n)$, and the last rule is of the form

$$\frac{\Pi_1 \vdash G\theta}{(p(t_1,\dots,t_n)\theta \leftarrow G\theta), \Pi_1 \vdash p(t_1,\dots,t_n)\theta}$$

  By the Def rule of the operational semantics, we know that $\langle (p(t_1,\dots,t_n), \Gamma), S\rangle \Rightarrow_{P,sat} \langle (G, \Gamma), S\rangle$; but from the induction hypothesis, we know that there is a T such that $\langle (G,\Gamma), S\rangle \Rightarrow^*_{P,sat} \langle \varepsilon, T\rangle$. □

Thus summary of these generalized theorems is as follows.

Corollary A.4 (SH characterizes strict, Henkin systems). *For any closed goal formula G, program P and realizable, strict, Henkin satisfiability function sat, we have that $\langle G, \emptyset\rangle$ succeeds with respect to $\Rightarrow_{P,sat}$ iff $P \vdash G$ is derivable in* SH_{sat}.

Proof. ($\rightarrow$) An instance of Theorem A.1, with $S = \emptyset$ and $n = 1$. Since G is closed, if the substitution mentioned the theorem exists, then G succeeds by itself.

($\leftarrow$) An instance of Theorem A.3, with $n = 1$, $S = \emptyset$, and θ as the empty substitution. □

SH_{sat} can therefore be said to be a sound and complete proof-theoretic characterization of the operational semantics for the CLP system corresponding to *sat*.

A.2. S characterizes strict systems

The proof plan here will be the same: prove soundness of OS_{sat} with respect to S_{sat}, then completeness. The details, however, are slightly different.

Theorem A.5 (Soundness of OS with respect to S). *Let sat be a realizable, strict satisfiability function. Let $S = \{c_1,\dots,c_m\}$ be a set of constraints. If $\langle (G_1,\dots,G_n), S\rangle \Rightarrow^*_{P,sat} \langle \varepsilon, T\rangle$ and $sat(T) \simeq true$, then $T, P \vdash B$ in S_{sat} for all B in $G_1,\dots,G_n, c_1,\dots,c_m$.*

Proof. By induction on the number of steps in the computation.

- 0 steps: n must be 0, and $S = T$. Therefore all the elements of S must be derivable directly by using the Axiom rule.
- >0 steps: by cases on the first rule used in the computation. Let Γ be the formulae $G_2,\dots,G_n$.
  - $\langle (c,\Gamma), S\rangle \Rightarrow_{P,sat} \langle \Gamma, S \cup \{c\}\rangle$: directly from the induction hypothesis.
  - $\langle (G\&H, \Gamma), S\rangle \Rightarrow_{P,sat} \langle (G, H, \Gamma), S\rangle$: From the induction hypothesis, we have that $T, P \vdash G$ and $T, P \vdash H$; thus by using the & rule of B, we have that $T, P \vdash (G\&H)$. In this and all subsequent cases, the fact that the Γ and S formulae are derivable carries over directly from the induction hypothesis.

- $\langle(\exists xG, \Gamma), S\rangle \Rightarrow_{P,sat} \langle(G[x := y], \Gamma), S\rangle$: From the induction hypothesis, we know that T, $P \vdash G[x := y]$. By the $\exists$ rule of B, we have that $T, P \vdash (\exists xG)$.
- $\langle(A\theta', \Gamma), S\rangle \Rightarrow_{P,sat} \langle(G'\theta', \Gamma), S\rangle$: From the induction hypothesis, we have that $T, P \vdash G'\theta'$. We therefore have the following derivation:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{T, P \vdash G'\theta'}{(A \leftarrow G')\theta', T, P \vdash A\theta'}\ (\leftarrow/1)}{(\forall x_n(A \leftarrow G'))\theta', T, P \vdash A\theta'}\ (\forall/1)}{\vdots}}{\forall x_1 \cdots \forall x_n(A \leftarrow G'), T, P \vdash A\theta'}\ (\forall/1)}{T, P \vdash A\theta'}\ (\text{dupl}/1)$$

But $A\theta'$ is just G_1, so $T, P \vdash G_1$. $\square$

To prove the completeness of OS_{sat} with respect to S, we need to expand our definition of derivative and meta-derivability.

Definition A.6. A multiset of formulae Γ' is an *S-derivative* of a multiset Γ if $\Gamma \vdash G$ can be derived from $\Gamma' \vdash G$ by a sequence of applications of the S rules Dup/l, $\forall$/l, Con/l, $\exists$/l and &/l:

$$\cfrac{\cfrac{\Gamma' \vdash G}{\vdots}}{\Gamma \vdash G}$$

A formula G is *meta-derivable* from a multiset Γ of formulae in S if there is some S-derivative Γ' of Γ such that $\Gamma' \vdash G$ is derivable in S.

Theorem A.7 (Completeness of OS with respect to S). *Let sat be a realizable, strict satisfiability function. Assume that $G_1, \ldots, G_n$ are a sequence of goals and $S = \{c_1, \ldots, c_m\}$ are a set of constraints. Assume further that there is substitution θ and set $T \supseteq S\theta$ of constraints such that $sat(T) \simeq true$, and $G_1\theta \ldots G_n\theta$ are all meta-derivable from (T, P) in S_{sat}. Then there is some set T' of constraints such that $sat(T') \simeq true$ and $\langle(G_1, \ldots, G_n), S\rangle \Rightarrow^*_{P,sat} \langle\varepsilon, T'\rangle$.*

Proof. Because $sat(T) \simeq true$, $S\theta \subseteq T$, and sat is strict, we know that $sat(S) \simeq true$. If $n = 0$, we choose T' to be S and we have the result (the computation takes 0 steps).

Otherwise, for each $G_i\theta$ there is a derivative Π_i of (T, P) such that $\Pi_i \vdash G_i\theta$ is derivable in S. We prove the result by induction on the total size of all these derivations. Cases are on the last rule used in the derivation of $\Pi_1 \vdash G_1\theta$. Let Γ be $G_2, \ldots, G_n$.

- $G_1 \equiv (G\&H)$, and the last step in the derivation of $G_1\theta$ is the form

$$\frac{\Pi_1 \vdash G\theta \qquad \Pi_1 \vdash H\theta}{\Pi_1 \vdash (G\&H)\theta}$$

Thus $G\theta$ and $H\theta$ are both meta-derivable from (T, P). By the & rule of the operational semantics, $\langle((G\&H), \Gamma), S\rangle \Rightarrow_{P,sat} \langle(G, H, \Gamma), S\rangle$; but by the induction hypothesis, there is a T' such that $\langle(G, H, \Gamma), S\rangle \Rightarrow^*_{P,sat} \langle\varepsilon, T'\rangle$.

- $G_1 \equiv \exists xG$, and the last step of the derivation of G_1 is of the form

$$\frac{\Pi_1 \vdash (G[x := t])\theta}{\Pi_1 \vdash (\exists xG)\theta}$$

Thus $G[x := t]\theta$ is meta-derivable from (T, P). We choose x' to be a new variable; then, by the $\exists$ rule of the operational semantics, $\langle(\exists xG, \Gamma), S\rangle \Rightarrow_{P,sat} \langle(G[x := x'], \Gamma), S\rangle$. But the substitution $\theta[x' := t]$ is such that $B\theta[x' := t]$ is meta-derivable from (T, P) in S_{sat} for all B in $G[x := x'], \Gamma$. We can therefore apply the induction hypothesis. We conclude that for some T', $\langle(G[x := x'], \Gamma), S\rangle \Rightarrow^*_{P,sat} \langle\varepsilon, T'\rangle$.

- The last rule applied is a Dup/l, $\forall$/l, Con/l, $\exists$/l, or &/l rule: The left-hand side of the premiss sequent is still a derivative of (T, P), so the result follows directly from the induction hypothesis.
- G_1 is a constraint, and the last rule is an Axiom rule:

$$\frac{}{\Pi_1 \vdash G_1\theta}$$

There are two situations in which this can happen. The first is when $G_1\theta \in T$; the other situation is when $G_1\theta$ is a closed constraint such that $sat(\{G_1\theta\}) \simeq true$, and Π_1 is obtained by introducing $G_1\theta$ on the left with a Con/l rule. In either situation, $sat(T \cup \{G_1\theta\}) \simeq true$, so by the realizability and strictness of *sat*, $sat(S \cup \{G_1\}) \simeq true$. By the Con rule of the operational semantics, $\langle(G_1, \Gamma), S\rangle \Rightarrow_{P,sat} \langle\Gamma, S \cup \{G_1\}\rangle$. But if $G_2 \ldots G_n$ and $c_1 \ldots c_m$ are meta-derivable from T, P, they must be meta-derivable from $T, G_1\theta, P$; so we can apply the induction hypothesis. There is therefore a T' such that $\langle\Gamma, S \cup \{G_1\}\rangle \Rightarrow^*_{P,sat} \langle\varepsilon, T'\rangle$.

- G_1 is a predicate call, and the last rule applied is of the form

$$\frac{\Pi_1 \vdash G\theta}{(p(t_1, \ldots, t_n) \leftarrow G)\theta, \Pi_1 \vdash p(t_1, \ldots, t_n)\theta}$$

By the Def rule of the operational semantics, $\langle(p(t_1, \ldots, t_n), \Gamma), S\rangle \Rightarrow^*_{P,sat} \langle(G, \Gamma), S\rangle$; but by the induction hypothesis, there is a T' such that $\langle(G, \Gamma), S\rangle \Rightarrow^*_{P,sat} \langle\varepsilon, T'\rangle$. □

We can again summarize these results in a "characterization" theorem.

Corollary A.8 (S characterizes strict satisfiability functions). *For any closed goal formula G, program P and strict, realizable satisfiability function sat, we have that* $\langle G, \varepsilon\rangle$ *succeeds with respect to* $\Rightarrow_{P,sat}$ *iff* $P \vdash G$ *in* S_{sat}.

Proof. ($\rightarrow$) By Theorem A.5, we know that for some T such that $sat(T) \simeq true$, we have that $T, P \vdash G$. But then we can construct a derivation leading from this sequent as follows:

$$\dfrac{\dfrac{\dfrac{T, P \vdash G}{\vdots}}{\exists[T], P \vdash G}\ \text{(some \&/l and $\exists$/l)}}{P \vdash G}\ \text{(Con/l)}$$

($\leftarrow$) An instance of Theorem A.7, with $S = \emptyset$, $n = 1$, θ as the empty substitution, and $T = \emptyset$. (We know that $sat(\emptyset) \simeq true$ by the strictness and realizability of sat). $\square$

A.3. MR$_{clpr}$ *characterizes CLP(R)*

As with the other proof systems, we will prove first soundness, then completeness.

Theorem A.9 (Soundness of OS with respect to MR). *If* $\langle (G_1, \ldots, G_n), S \rangle \Rightarrow^*_{P,clpr} \langle \varepsilon, T \rangle$ *and* $clpr(T) \simeq true$, *then there are sets* $T_1, \ldots, T_n$ *such that* $T_1 \cup \cdots \cup T_n \cup S = T$, *and* $(T_i, \Box P) \vdash G_i$ *for all* i, $1 \leqslant i \leqslant n$.

Proof. By induction on the number of steps in the computation.

- 0 steps: n must be 0, and $S = T$.
- >0 steps: by cases on the first rule used in the computation. Let Γ be the formulae $G_2, \ldots, G_n$.
  - $\langle (c, \Gamma), S \rangle \Rightarrow_{P,clpr} \langle \Gamma, S \cup \{c\} \rangle$: From the induction hypothesis, we know that there are $T_2, \ldots, T_n$ such that $T_2 \cup \cdots \cup T_n \cup (S \cup G_1) = T$, and $T_i, \Box P \vdash G_i$ for all i, $2 \leqslant i \leqslant n$. But then we can choose T_1 to be just G_1 itself, and derive T_1, $\Box P \vdash G_1$ by one Axiom.
  - $\langle (G \& H, \Gamma), S \rangle \Rightarrow_{P,clpr} \langle (G, H, \Gamma), S \rangle$: From the induction hypothesis, we have that there are T_1^1 and T_1^2 such that $T_1^1, P \vdash G$ and $T_1^2, P \vdash H$; thus by using the & rule of B, we have that $T_1^1, T_1^2, P \vdash (G \& H)$. We choose T_1 to be $T_1^1 \cup T_1^2$; this is not necessarily the same as the multiset union (T_1^1, T_1^2), but we can produce T_1 by a sequence of Dup/l rules:

    $$\dfrac{\dfrac{\dfrac{T_1^1, P \vdash G \qquad T_1^2, P \vdash H}{T_1^1, T_1^2, P \vdash G \& H}\ \text{(\&/r)}}{\vdots}}{T_1, P \vdash G \& H}\ \text{(some Dup/ls)}$$

    And in this and all subsequent cases, the fact that the Γ and S formulae are derivable carries over directly from the induction hypothesis.
  - $\langle (\exists x G, \Gamma), S \rangle \Rightarrow_{P,clpr} \langle (G[x := y], \Gamma), S \rangle$: From the induction hypothesis, we know that there is a T_1 such that $(T_1, P) \vdash G[x := y]$. By the $\exists$/r rule, we have that $(T_1, P) \vdash (\exists x G)$.

– $\langle A\theta', \Gamma), S\rangle \Rightarrow_{P,clpr} \langle G'\theta', \Gamma), S\rangle$: From the induction hypothesis, we know that for some $T_1, (T_1, P) \vdash G'\theta'$. We therefore have the following derivation:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{T_1, P \vdash G'\theta'}{(A \leftarrow G)\theta', T_1, P \vdash A\theta'}\ (\leftarrow/1)}{(\forall x_n(A \leftarrow G))\theta', T_1, P \vdash A\theta'}\ (\forall/1)}{\vdots}}{\forall x_1 \cdots \forall x_n(A \leftarrow G), T_1, P \vdash A\theta'}\ (\forall/1)}{\Box\forall x_1 \cdots \forall x_n(A \leftarrow G), T_1, P \vdash A\theta'}\ (\text{Nec}/1)}{T_1, P \vdash A\theta'}\ (\text{dupl}/1)$$

But $A\theta'$ is just G_1, so $(T_1, P) \vdash G_1$. □

To prove the completeness of OS_{clpr} with respect to MR_{clpr}, we again need to slightly adjust our definition of derivative and meta-derivability. We add the new rule Nec/l, and because of the structure of our completeness proof, remove the rule Con/l from the rules that can be used to produce a derivative.

Definition A.10. A multiset of formulae Γ' is an *MR-derivative* of a multiset Γ if $\Gamma \vdash G$ can be derived from $\Gamma' \vdash G$ by a sequence of MR Dup/l, $\forall$/l, $\exists$/l &/l and Nec/l rules:

$$\cfrac{\cfrac{\Gamma' \vdash G}{\vdots}}{\Gamma \vdash G}$$

A formula G is *meta-derivable* from a multiset Γ of formulae in MR if there is some MR-derivative Γ' of Γ such that $\Gamma' \vdash G$ is derivable in MR.

Theorem A.11 (Completeness of OS with respect to MR). *Let $G_1, \ldots, G_n$ be goal formulae, $T_1, \ldots, T_n, S$ be sets of constraints, and θ be a substitution which maps variables to variables. Then if $G_i\theta$ is meta-derivable from (T_i, P) in MR_{clpr}, for $1 \leqslant i \leqslant n$, and $clpr(T_1 \cup \cdots \cup T_n \cup S\theta) \simeq true$, then there is a set T of constraints such that $\langle (G_1, \ldots, G_n), S\rangle \Rightarrow^*_{P,clpr} \langle \varepsilon, T\rangle$ and $clpr(T) \simeq true$.*

Proof. If $n = 0$, then the union of $T_1, \ldots, T_n$ is empty, so $clpr(S\theta) \simeq true$; but then, because of property 4 of *clpr* (Section 3.4), $clpr(S) \simeq true$. We choose T to be S; the computation takes 0 steps.

Otherwise, for each $G_i\theta$ there is a derivative Π_i of T_i, P such that $\Pi_i \vdash G_i\theta$ is derivable in S. We proceed by induction on the total size of all these derivations. Cases are on the last rule used in the derivation of $\Pi_1 \vdash G_1\theta$. Let Γ be $G_2, \ldots, G_n$.

- G_1 is a constraint, and the last rule is an Axiom. The only situation in which this can occur is when Π_1 is $(G_1\theta, \Box P)$, which in turn can only happen if $T_1 = \{G_1\theta\}$. We therefore know that $clpr(T_2 \cup \cdots \cup T_n \cup (S \cup \{G_1\})\theta) \simeq clpr(T_2 \cup \cdots \cup T_n \cup S\theta \cup \{G_1\theta\}) \simeq clpr(T_1 \cup T_2 \cup \cdots \cup T_n \cup S\theta) \simeq true$. By the reliability of the satisfiability function *clpr*, we know that $clpr(S \cup \{G_1\}) \not\simeq false$ (although it could be that $clpr(S \cup \{G_1\}) \simeq unsure$). By the Con rule of the operational semantics, we therefore have that $\langle (G_1, \Gamma), S\rangle \Rightarrow_{P,clpr} \langle \Gamma, S \cup \{G_1\}\rangle$; and by the induction hypothesis, there is a T such that $\langle \Gamma, S \cup \{G_1\}\rangle \Rightarrow^*_{P,clpr} \langle \varepsilon, T\rangle$.

- $G_1 \equiv (G\&H)$, and the last step in the derivation of $G_1\theta$ is of the form

$$\frac{\Pi_1^1 \vdash G\theta \qquad \Pi_1^2 \vdash H\theta}{\Pi_1^1, \Pi_1^2 \vdash (G\&H)\theta}$$

(Π_1^1, Π_1^2) is a derivative of (T_1, P), so it must consist of T_1, P, their duplicates, and other formulae arising from applications of Nec/l and $\forall$/l. ($\exists$/l and &/l cannot be used because no existential or conjunctive formulae appear.) Therefore there are T_1^1 and T_1^2 such that $T_1^1 \cup T_1^2 = T_1$, Π_1^1 is a derivative of T_1^1, and Π_1^2 is a derivative of T_1^2. By the & rule of the operational semantics, $\langle ((G\&H), \Gamma), S\rangle \Rightarrow_{P,clpr} \langle (G, H, \Gamma), S\rangle$; but by the induction hypothesis, there is a T such that $\langle (G, H, \Gamma), S\rangle \Rightarrow^*_{P,clpr} \langle \varepsilon, T\rangle$.

- $G_1 \equiv \exists x G$, and the last step of the derivation of G_1 is of the form

$$\frac{\Pi_1 \vdash (G[x := z])\theta}{\Pi_1 \vdash (\exists x G)\theta}$$

Thus $G[x := z]\theta$ is meta-derivable from T_1, P. We choose x' to be a new variable; then, by the $\exists$ rule of the operational semantics, $\langle (\exists x G, \Gamma), S\rangle \Rightarrow_{P,clpr} \langle (G[x := x'], \Gamma), S\rangle$. But the substitution $\theta[x' := z]$ is such that $B\theta[x' := z]$ is meta-derivable from (T, P) in MR_{clpr} for all B in $G[x := x'], \Gamma, c_1, \ldots, c_m$. We can therefore apply the induction hypothesis. We conclude that for some T, $\langle (G[x := x'], \Gamma), S\rangle \Rightarrow^*_{P,clpr} \langle \varepsilon, T\rangle$.

- The last rule applied is a Dup/l, $\forall$/l, $\exists$/l, &/l, or Nec/l rule: The left-hand side of the premiss sequent is still a derivative of (T, P), so the result follows directly from the induction hypothesis.
- The last rule applied is a Con/l rule:

$$\frac{\exists[S'], \Pi_1 \vdash G_1\theta}{\Pi_1 \vdash G_1\theta}$$

If the upper sequent is derivable, then so is $S'\theta', \Pi_1 \vdash G_1\theta$, where θ' is a substitution which renames the free variables of S' apart from all those in Π_1 and $G_1\theta$. (In fact θ' could rename those variables apart from all those in all the Π_i's G_i's and S.) By property 3 of *clpr* (Section 3.4), we must therefore have that $clpr(S'\theta' \cup T_1 \cup \cdots \cup T_n \cup S\theta) \simeq true$. We can therefore apply the induction hypothesis, and conclude that for some T, regardless of what G_1 is, $\langle (G_1, \Gamma), S\rangle \Rightarrow^*_{P,clpr} \langle \varepsilon, T\rangle$.

- G_1 is a predicate call, and the last rule applied is of the form

$$\frac{\Pi_1 \vdash G\theta}{(p(t_1, \ldots, t_n) \leftarrow G)\theta, \Pi_1 \vdash p(t_1, \ldots, t_n)\theta}$$

By the Def rule of the operational semantics, $\langle (p(t_1, \ldots, t_n), \Gamma), S\rangle \Rightarrow^*_{P,clpr} \langle (G, \Gamma), S\rangle$; but by the induction hypothesis, there is a T such that $\langle (G, \Gamma), S\rangle \Rightarrow^*_{P,clpr} \langle \varepsilon, T\rangle$. $\square$

We can again summarize these results in a "characterization" theorem.

Corollary A.12 (MR$_{clpr}$ characterizes CLP(R)). *For any closed goal formula G and program P, we have that $\langle G, \emptyset\rangle$ succeeds with respect to $\Rightarrow_{P,clpr}$ iff $\square P \vdash G$ is derivable in* MR$_{clpr}$.

Proof. ($\rightarrow$) By an instance of Theorem A.9, with $n = 1$ and $S = \emptyset$, there is a set T_1 such that $clpr(T_1) \simeq true$, and $T_1, \square P \vdash G$. But if this is the case, then we can derive $\square P \vdash G$ by some uses of &/l and $\exists$/l and one use of Con/l:

$$\dfrac{\dfrac{\dfrac{\dfrac{T_1, \square P \vdash G}{\vdots}}{\&(T_1), \square P \vdash G}\ \text{(some \&/l)}}{\vdots}}{\dfrac{\exists[T_1], \square P \vdash G}{\square P \vdash G}\ \text{(Con/l)}}\ \text{(some }\exists\text{/l)}$$

($\leftarrow$) By property 2 of *clpr* (Section 3.4), we know that $clpr(\emptyset) \simeq true$. By an instance of A.11, with $n = 1$, $T_1 = \emptyset$, $S = \emptyset$, and θ as the empty substitution, we know that there is a T such that $\langle G, \emptyset\rangle \Rightarrow^*_{P,clpr} \langle \varepsilon, T\rangle$ and $clpr(T) \simeq true$; that is, that G succeeds. $\square$

References

[1] A.R. Anderson and N.D. Belnap, *Entailment: The Logic of Relevance and Necessity* (Princeton University Press, Princeton, NJ, 1975).

[2] J.H. Andrews, Proof-theoretic characterisations of logic programming, in: *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science Vol. 379, Porąbka-Kozubnik, Poland (Springer, Berlin, 1989) 145–154.

[3] A. Colmerauer, Prolog and infinite trees, in: K.L. Clark and S.-A. Tärnlund, eds., *Logic Programming* (Academic Press, New York, 1983) 231–251.

[4] A. Colmerauer, Equations and inequations on finite and infinite trees, in: *Proc. Internat. Conf. on Fifth Generation Computer Systems*, ICOT (1984) 85–99.

[5] T. Frühwirth, Constraint simplification rules, Technical Report 92-18, ECRC, Munich, Germany, July 1992.

[6] G. Gentzen, in: M.E. Szabo, ed., *The Collected Papers of Gerhard Gentzen* (North-Holland, Amsterdam, 1969).

[7] J.-Y. Girard, Linear logic, *Theoret. Comput. Sci.* **50** (1987) 1–102.

[8] M. Hagiya and T. Sakurai, Foundation of logic programming based on inductive definition, *New Generation Computing* **2** (1984) 59–77.

[9] L. Hallnäs and P. Schroeder-Heister, A proof-theoretic approach to logic programming I: clauses as rules, *J. Logic Comput.* **1** (1990).

[10] J.S. Hodas and D. Miller, Logic programming in a fragment of intuitionistic linear logic, *Inform. and Comput.* **110** (1994) 327.

[11] M. Höhfeld and G. Smolka, Definite relations over constraint languages, Technical Report 53, LILOG, IBM Deutschland, Stuttgart, Germany, October 1988; to appear in *J. Logic Programming.*
[12] J. Jaffar and J.-L. Lassez, Constraint logic programming, in: *Proc. Conf. on Principle of Programming Languages*, Munich, 1987, 111–119.
[13] J. Jaffar, S. Michaylov, P.J. Stuckey and R.H.C. Yap, The CLP($\mathscr{R}$) language and system, *ACM Trans. Programming Languages Systems* **14** (1992) 339–395.
[14] S.C. Kleene, *Introduction to Metamathematics*, Bibliotheca Mathematica, Vol. 1 (North-Holland, Amsterdam, 1952).
[15] M.J. Maher, Complete axiomatizations of the algebras of finite, rational and infinite trees, in: *Proc. 3rd Ann. Symp. on Logic in Computer Science*, Edinburgh (Computer Society Press, Washington, DC, 1988) 348–357.
[16] M.J. Maher, A logic programming view of CLP, in: *Proc. 10th Internat. Conf. on Logic Programming*, Budapest (MIT Press, Cambridge, MA, 1993) 737–753.
[17] D. Miller, G. Nadathur, F. Pfenning and A. Scedrov, Uniform proofs as a foundation for logic programming, *Ann. Pure Appl. Logic* **51** (1991) 125–157.
[18] J.D. Monk, *Mathematical Logic*, Graduate Texts in Mathematics, Vol. 37 (Springer, New York, 1976).
[19] S. Read, *Relevant Logic* (Blackwell, Oxford, 1988).
[20] V.A. Saraswat, Concurrent constraint programming languages, Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, January 1989.
[21] D. Scott, Domains for denotational semantics, in: *Internat. Colloq. on Automata, Languages, and Programming*, 1982.
[22] D. van Dalen, *Logic and Structure* (Springer, Berlin, 1980).
[23] M.H. van Emden and J.W. Lloyd, A logical reconstruction of Prolog II, *J. Logic Programming* **2** (1984) 143–149.
[24] P. Van Hentenryck, *Constraint Satisfaction in Logic Programming* (MIT Press, Cambridge, MA, 1989).