ICOT Technical Report: TR-562

TR-562

# Coinductive Semantics of Horn Clauses with Compact Constraint

by
K. Mukai

May, 1990



ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191~5
Telex ICOT J32964

**Institute for New Generation Computer Technology**

# Coinductive Semantics of Horn Clauses with Compact Constraint*

Kuniaki Mukai

Institute for New Generation Computer Technology
Mita Kokusai-Build. 21F
4-18, Mita 1-Chome, Minato-ku, Tokyo 108 Japan

**Abstract**

A constraint language $L$ with an interpretation in the domain $V_{A,\kappa}$ of non well-founded sets over a finite set $A$ of atoms which is hereditary finite with a cardinality no more than $\kappa$ is proposed. $L$ is a quantifier free first order sublanguage with equality, subsumption, disjunction, and negation. It is proved that the domain $V_{A,\kappa}$ is solution compact and $L$ is satisfaction complete in the sense of constraint logic programming (CLP) schema. According to the schema we have now CLP(AFA) just as CLP(R), where $R$ is the domain of real numbers. By general theory of the CLP scheme completeness and soundness theorem are obtained for the class of canonical programs. A characterization of the canonical programs is given in terms of bisimulation relation. Operationally this is equivalent to that variables in negative constraints must be grounded in a finite number of steps on the computation.

These results are shown directly based on the AFA set theory. A declarative semantics and an operational semantics of a given Horn clause program over the constraint language $L$ are defined coinductively in the domain $V_{A,\kappa}$. Soundness and completeness are proved by showing a *simulation* relation between two semantic domains. Basic computational concepts of CLP schema are well understood in this method of ZFC$^-$/AFA set theory.

A large part of existing constraint logic programming and unification grammar formalisms are reconstructed in the domain $V_{A,\kappa}$.

## 1 Introduction

We assume that the reader is familiar with basic notions of non-well founded sets theory[1], constraint logic programming[7], and unification grammar formalisms[8].

*ICOT-TR562, 1990. This short note is a revised version of the extended abstract which was presented for explaining some basic ideas of 'AFA Prolog' at an informal joint workshop of NLU and PSG working group at ICOT, 19th-20th March 1990. This paper was accepted to be read at the second conference on Situation Theory and its Application at Scotland, September 1990. Also this paper will appear in the proceedings of the annual conference of logic programming Japan, Tokyo, 1990.

Aczel[1] proposed the universe of non-well-founded sets for modelling circularly structured objects. He provides very powerful coinductive method like the inductive one in the standard well-founded sets universe. Barwise and Etchemendy[4] and Barwise[3] applied Aczel's theory to circular situations and unification of feature set respectively. The present work was inspired by Barwise[3] and extends it so that negation and disjunction are included into AFA-based unification grammar formalism. In logic programming, Colmerauer[5] used infinite trees domain for modelling such kind of circularity. Now I am trying in the present paper to replace the infinite trees with non well-founded sets to see how neatly constraint logic programming schema and unification grammar formalisms can be reconstructed.

According to the CLP schema, the key point of the problem is to find a language $L$ and subclass $W$ of non-well-founded sets of ZFC$^-$/AFA[1] theory such that $W$ is solution compact and $L$ is satisfaction complete.

Our language $L$ is a constraint language consisting of set equations, subsumption (hereditary subset relation), conjunction (of course), disjunction, and negation. A term of $L$ is a parametric hereditary finite *well-founded* set which has at most $k$ elements, where $k$ is a certain fixed finite integer.

On the other hand, our domain $V_{A,\kappa}$ for $W$ is the class of hereditary finite subsets which has a bounded size $\kappa$. It is proved that $V_{A,\kappa}$ and $L$ are what we want. I believe that this discovery will give a new integrated view to both logic programming and unification grammar formalism.

We use Aczel's theory in two ways. One is for defining semantics of the program coinductively and the other is as a domain of logic programming as mentioned above. In the former two classes are defined coinductively. One is a class of non-well-founded triples for the declarative semantics of the program, the other is the class of non-well-founded pairs for SLD-like fair computation trees for queries. Soundness and completeness results are proved by showing a simulation relation between them. Indeed the simulation relates solutions with computations in a hereditary way.

The subdomain of records in CLP(AFA) has an efficient implementation for a theory of $=$ and $\neq$. As a fact, these constraint can be combined straightforwardly with well known Boolean constraint solving methods and UNION-FIND technique[2] into implementing practical computation.

The present paper is a report of a recent drastic progress of a logic programming system CIL[10], which has been based on records and lazy control on practical basis.

# 2 Preliminary

We use a Aczel's non well founded set theory ZFC$^-$/AFA[1] as the metatheory of the present paper throughout. The following concepts are from Aczel[1]: coinductive definition, solution lemma, bisimulation relation[11].

Let $X$ be a set of variables and $(b_x)$ $\{x \in X\}$ be a family of *sets*. The set of equations $x = b_x$ is called a *system of equations*. The following theorem is a special case of Aczel[1], but is enough for our present purpose.

**Theorem 1 (Solution Lemma(Aczel[1]))** *Every system of equations has a unique solution.*

[1]ZFC minus the axiom foundation plus Aczel's anti foundation axiom.

A *system* is a class of ordered pairs such that for any node $a$ in the system the all the successors of $a$ form a *set*, which is written $a_M$. Let $M$ be a system. A binary relation $R$ on the system $M$ is a *bisimulation* on $M$ if $R \subset R^+$, where for $a, b \in M$

$$aR^+b \Leftrightarrow \forall x \in a_M \exists y \in b_M xRy \& \forall y \in b_M \exists x \in a_M xRy.$$

A bisimulation relation $R$ is represented as the set $c_R$ of equations $a = b$ such that $aRb$. A set $c$ of equations is called a *bisimulation constraint* if $c = c_R$ for some bisimulation relation $R$.

The main technical result of the present work owes to Barwise[3], which gives a characterization of feature structure unification through existence of simulation pair which includes the given unification data. The definition of solution compactness and satisfaction complete are from Stuckey[9].

# 3 Constraint Language

We borrow from Smolka[12] the following definition. A *constraint language* is a quadruple $(VAR, CON, \mathcal{V}, INT)$, where

- $VAR$ is a set of *variables*,
- $CON$ is a set of *constraints*,
- $\mathcal{V}$ is a function which assigns to each constraint, say $\psi$, a set $\mathcal{V}\psi$ of variables,
- $INT$ is a set of *interpretations*. An *interpretation* is given as a pair $(D, S)$ of a *domain* $D$ and a function $S$ such that for any constraint $\psi \in CON$, $S$ assigns a set $S\psi$ of partial functions from $VAR$ into $D$. An element of $S\psi$ is called a *solution* of $\psi$. If the restriction $f$ to $\mathcal{V}\psi$ is the same as that of some solution in $S\psi$, $f$ must be in $S\psi$.

A constraint language $L$ is *compact* if any set $c$ of constraints in $L$ is satisfiable whenever every finite subset of $c$ is satisfiable.

## 3.1 Constraint Language $L$

In this section we introduce a constraint language $L$. Roughly speaking, a constraint is a quantifier–free first order formula over the following items:

- *well–founded* hereditary finite parametric sets
- $=, \sqsubset$
- $\wedge, \vee, \neg$

Let us define the language $L = (VAR, CON, \mathcal{V}, INT)$ more formally. We assume in the rest of the paper that $VAR$ is a countably infinite set of variables. $\phi$ is a *reserved constant* symbol, which denotes the *empty* set. $\phi = \{\}$. We assume that our language has only *finite* number of constant symbols. This assumption will be used later in the proof of compactness of the domain $V_{A,\kappa}$. A *term* is defined inductively as follows.

- A constant is a term.
- A variable is a term.
- A set $\{a_1, \cdots, a_n\}$ is a term, where $a_1, \cdots, a_n$ is a term.

For the sake of simplicity we use the following conventions in the formal part of the paper.

- If a term is a set then each element of the term is a constant or a variable.
- A constant denotes itself.

Note that by this convention $\phi$ is the only constant which denotes a set; i.e. other constants behaves as an *urelement*. An *atomic constraint* is an equation $a = b$ or a subsumption $a \sqsubset b$ where $a$ and $b$ are terms. A *literal* is an atomic constraint or negation $\neg a$ where $a$ is an atomic constraint. A *constraint* is a literal, $\neg a$, $a \vee b$, or $a \wedge b$ where $a$ and $b$ are constraints. We write $a \neq b$ and $a \not\sqsubset b$ for $\neg(a = b)$ and $\neg(a \sqsubset b)$ respectively.

We assume that every term in the language $L$ has a bounded size in a uniform way. That is, there is an integer $\kappa$ such that the cardinality of every set in the language is no more than $\kappa$.

Note that our formalism includes that of the 'record as function' as introduced later as a special case.

The following are the list of rules of the constraint language $L$.

- $x = x$.
- If $x = y$ then $y = x$.
- If $x = y$ and $y = z$ then $x = z$.
- If $\{x_1, \cdots, x_n\} = \{y_1, \cdots, y_m\}$ then $x_1 = y_{j(1)}, \cdots, x_n = y_{j(n)}$, and $x_{k(1)} = y_1$, $\cdots, x_{k(m)} = y_m$, for some function

  $$j : \{1, \cdots, n\} \to \{1, \cdots, m\}$$

  and

  $$k : \{1, \cdots, m\} \to \{1, \cdots, n\}.$$
- If $\{x_1, \cdots, x_n\} \neq \{y_1, \cdots, y_m\}$ then either $x_j \neq y_1, \cdots$, and $x_j \neq y_m$ for some $1 \leq j \leq n$ or $x_1 \neq y_k, \cdots$, and $x_n \neq y_k$ for some $1 \leq k \leq m$.
- $x \sqsubset x$.
- If $x \sqsubset y$ and $y \sqsubset z$ then $x \sqsubset z$.
- If $x = y$ and $x \sqsubset z$ then $y \sqsubset z$.
- If $x = y$ and $z \sqsubset x$ then $z \sqsubset y$.
- If $x \sqsubset y$ and either $x$ or $y$ is a constant but $\phi$ then $x = y$.
- If $\{x_1, \cdots, x_n\} \sqsubset \{y_1, \cdots, y_m\}$ then $x_1 \sqsubset y_{j(1)}, \cdots$, and $x_n \sqsubset y_{j(n)}$ for some function

  $$j : \{1, \cdots, n\} \to \{1, \cdots, m\}.$$
- If $\{x_1, \cdots, x_n\} \not\sqsubset \{y_1, \cdots, y_m\}$ then $x_j \not\sqsubset y_1, \cdots$, and $x_j \not\sqsubset y_m$ for some $1 \leq j \leq n$.

These rules are Horn–like in the sense that positive and negative information are derived only from positive and negative one respectively. The rules are combinatorial in the sense that every piece of information on the right–hand side of the rule appears in that on the left–hand side. These two properties of the rules will be used in the rest of the paper implicitly.

These rules are not intended to be efficient for computation but only for a theoretical exposition. Record structure should be used for practical computation. The record structure shall be discussed later part in the paper for implementation purpose.

We can restrict the two transitive laws above in such a way that it is applied only in the case that the intermediate term is a variable. By this restriction, the constraint $x = \phi \wedge y = \phi$ can not yield the information $x = y$. Also the rules which treats negative information is redundant for the purpose of the present paper.

**Remark** The $\sqsubset$ relation is not assumed to be a partial order. As a fact the following are true, which show a failure of the anti symmetric law:

$$\{\phi, \{\phi\}\} \sqsubset \{\{\phi\}\}$$

$$\{\{\phi\}\} \sqsubset \{\phi, \{\phi\}\}.$$

## 3.2 Support

In the rest of the paper, by a set of literals we mean the conjunction of possibly infinitely many literals in the set if the context is clear.

A *support* is a set of literals which has no complementary pair of literals and is closed under the constraint rules of the language $L$. For example, a bisimulation constraint is a support. A support is a constraint. The support is used to characterize the satisfiability of a given constraint.

A constraint $c'$ *is a support* of a constraint $c$ if $c'$ is a support and one of the following holds.

1. $c$ is a literal and $c \in c'$.
2. $c$ is $a \wedge b$, and both $a$ and $b$ have the support $c'$.
3. $c$ is $a \vee b$ and either $a$ or $b$ has the support $c'$.

Note that as a special case $c'$ is a support of $c$ if $c'$ is a support and $c \subset c'$. Also we say that a constraint $c$ has a support $c'$ if $c'$ is a support of $c$.

A variable $x$ *is bound* in a set $c$ of literals if $c$ has an equation $x = b$ for some *set* $b$.

A *normal constraint* $c$ is a set of literals such that each variable $x$ appearing in $c$ is bound in $c$.

A *canonical constraint* $c$ is a set of literals such that for each negative literal $q$ in $c$ there is a finite subset $c'$ of $c$ such that every variable accessible from some variable appearing in $q$ is bound in $c'$. Note that a normal constraint is a canonical constraint but a canonical constraint is not always a normal constraint.

## 3.3 Bounded Hereditary Finite Set

Let $A$ and $\kappa$ be a finite set of *atoms* and a positive integer respectively. Let $V_{A,\kappa}$ be the largest subset $D$ of the universe of non–well–founded sets generated over $A$ such that if $x \in D$ then there are some $x_1, \cdots, x_r$ in $D$ such that $x = \{x_1, \cdots, x_r\}$, where $0 \leq r \leq \kappa$. An element of $V_{A,\kappa}$ is a *Bounded Hereditary Finite Set with* $\kappa$. It will be proved later that a set $c$ of atomic constraints of $L$ is satisfiable in $V_{A,\kappa}$ if and only if every finite subset of $c$ is satisfiable in $V_{A,\kappa}$. This theorem is referred as Compactness Theorem.

Here are a couple of remarks about the compactness. Firstly, Herbrand universe $H$ is not compact. That is, we can not replace $V_{A,\kappa}$ in the theorem with $H$. To see this, take a set $c$ of atomic constraints

$$x_1 = f(x_2), x_2 = f(x_3), \cdots, x_n = f(x_{n+1}), \cdots$$

where $f$ is a unary function symbol. Every finite subset of $c$ is satisfiable, but $c$ itself is not satisfiable in $H$. Secondly, the compactness theorem above holds only for the positive constraint. For the constraint with negation, even $V_{A,\kappa}$ is not compact. To see this, take a set of constraints

$$x_1 \neq x_2, x_1 = \{x_2\}, x_2 = \{x_3\}, \cdots, x_n = \{x_{n+1}\}, \cdots. \tag{1}$$

Every finite subset of these constraints is satisfiable. However by the solution lemma the global solution of the equations is uniquely determined as

$$x_1 = x_2 = x_3 = \cdots = \{\{\{\cdots\}\}\},$$

which does not satisfies the unequation $x_1 \neq x_2$. Note that the constraint (1) is normal support but not canonical.

# 4 Unification over Non–Well–Founded Sets

We relate ZFC$^-$/AFA to the constraint logic programming scheme[7]. We show that the constraint language $L$ introduced above satisfies the criteria of the schema.

Here let us recall the definition of 'solution compact' and 'satisfaction complete'. A many sorted structure $\mathcal{R}$ is called *solution compact* by Jaffar and Lassez[7, 9] if the following hold.

(1) every element in $\mathcal{R}$ is the unique solution of a finite or infinite set of constraints.

(2) for every finite constraint $c$ and choice of $n$ there exists a finite or infinite family of constraints $c_i$ containing $n$ variables $x_1, \cdots, x_n$ such that:

$$\mathcal{R}^n - \{(f(x_1), \cdots, f(x_n)) | f \text{ is } \mathcal{R}\text{- solution of } c\}$$

$$= \bigcup_i \{(g(x_1), \cdots, g(x_n)) | g \text{ is } \mathcal{R}\text{- solution of } c_i\}.$$

We shall treat only the case of $n = 1$, since the general case in which variables $x_1, \cdots, x_n$ in the given constraint $c$ are chosen is reduced to this simple case by adding the equation $x = (x_1, \cdots, x_n)$ to $c$ in which $x$ is the only variable chosen, where $x$ is a new variable.

A theory $\mathcal{T}$ is *satisfaction complete* if

$$\mathcal{T} \models \forall y_1 \cdots \forall y_m \neg c \qquad \text{whenever not } \mathcal{T} \models \exists y_1 \cdots \exists y_m\ c$$

where $c$ is a finite constraint and $y_1, \cdots, y_m$ are all the variables appearing in $c$. In other words, a theory $\mathcal{T}$ is *satisfaction complete* if there is a model $M$ of $\mathcal{T}$ in which $c$ is unsolvable then $c$ is indeed unsolvable in every model of $\mathcal{T}$.

$TC(c)$ is the set of sets, parameters, or constants appearing in the constraint $c$. For example

$$TC(\{x = a, y \neq \{b, c\}\}) = \{x, a, y, \{b, c\}, b, c\}$$

where $x$,$y$, $a$, $b$, and $c$ are a constant or atom. It is easy to see that $TC(c)$ is finite if $c$ is finite. A support $c'$ of $c$ is *small* if the field of $c'$ is a subset of $TC(c)$.

**Lemma 1** *For a finite set $c$ of atomic constraints, the following are equivalent.*

*(1) $c$ has a finite support.*

*(2) $c$ has a support.*

**Proof** Condition (2) follows obviously from Condition (1). For the proof of the converse, suppose that $c'$ is a support of $c$. The restriction of $c'$ to $TC(c)$ is a support of $c$ since $c'$ is closed under the constraint rules. □

The following lemma is a corollary of 'simulation pair theorem' of Barwise[3].

**Theorem 2 (Simulation Pair Theorem )** *Given a set $c$ of atomic constraints the following are equivalent.*

- *$c$ has a normal support.*
- *$c$ has a solution in $V_{A,\kappa}$.*

Remember that every set term in $L$ has only at most $\kappa$ elements.

**Lemma 2** *Given a constraint $c$, the following are equivalent.*

*1. $c$ has a support.*

*2. either $c \cup \{x = a\}$ or $c \cup \{x = \{y_1, \cdots, y_\kappa\}\}$ has a support*

*where $x \in \mathcal{V}(c)$, $a$ is a constant and $y_1$, $\cdots$, $y_\kappa$ are new variables.*

**Proof** The proof from (2) to (1) is obvious. Suppose (1). Let $c'$ be a support of $c$. Firstly suppose that $a$ is a constant and that $x = a \in c'$. Then $c'$ is also a support of $c \cup \{x = a\}$. We are done. Secondly suppose that for any constant $a$, $x = a$ is not in $c'$. Let $X = \bigcup\{b |\ b$ is a set, $x = b \in c'\}$. By the assumption about $L$, if $X$ is non empty then every element of $X$ is a constant or variable. From the constraint rules and the limit size $\kappa$ of sets, there are at most $\kappa$ equivalence classes, say $e_1, \cdots, e_r$, in $X$ induced by the bisimulation relation contained in $c'$, where $r \leq \kappa$. Now we can divide the set $\{y_1, \cdots, y_\kappa\}$ into $r$ non empty groups, say $g_1, \cdots, g_r$. (Let $r = \kappa$ if $X$ is empty.) Let $d$ be the set of literals $l\theta$ where $l$ is a literal in $c'$ and $\theta$ is a substitution such that if $\theta(u) = v$ then $u \in e_i$ and $v \in g_i$ for some $0 \leq i \leq r$. It is easily seen that $d$ is a support of $c \cup \{x = \{y_1, \cdots, y_\kappa\}\}$. □

The following is a key lemma of the present paper.

**Lemma 3 (Normal Support Lemma)** *Given a constraint $c$, the following are equivalent.*

*(1) Every finite subset of $c$ has a support.*

*(2) $c$ has a normal support.*

**Proof** Condition (1) follows (2) obviously. Suppose Condition (1). Without loss of generality we assume that $c = \{q_n | 0 \leq n\}$, where each $q_i$ is a literal. Let us make an inductive definition $d_0 = \phi$ and $d_n = d_{n-1} \cup \{q_{n-1}\} \cup V_n$ in such a nondeterministic way that

- $V_n = \{x = e_x | x \in \mathcal{V}(d_{n-1} \cup \{q_{n-1}\}) \backslash \mathcal{V}(d_{n-1})\}$
- $e_x$ is a constant $a$ or $\{x_1, \cdots, x_\kappa\}$ where $x_1, \cdots, x_\kappa$ are new variables and $\kappa$ is the upper bound size of sets.

Furthermore, by the previous lemma, we can choose the sequence $d_n$ $(n \geq 0)$ so that the constraint $\bigcup\{d_j | j \in J\}$ has a support for each finite set $J$ of integers. Let $SP_n$ be the set of small supports of $d_n$. It follows from the assumption that $SP_n$ is non empty finite set and that for any $Q \in SP_n$ $(n \geq 1)$ there is a support $Q' \in SP_{n-1}$ such that $Q' \subset Q$. Let $T$ be the set of all finite or countable sequences $Q_j$ $(0 \leq j \leq \nu)$ of supports such that $Q_j \in SP_j$ and $Q_{j-1} \subset Q_j$ $(j \neq 0)$. Applying the standard argument of König's lemma to this situation, there is an infinite sequence $Q_j$ $(0 \leq j)$ in $T$ such that $Q_j \in SP_j$ and $Q_{j-1} \subset Q_j$ $(j \neq 0)$. Let $Q$ be the union of the sequence: $Q = \bigcup\{Q_j | 0 \leq j\}$. It is easily checked that $Q$ is a normal support of $c$. □

**Proposition 3 (Normal Unification Lemma)** *For any normal support $c$ the following are equivalent.*

*(1) $c$ has a solution.*

*(2) For each negative literal $l \in c$, $p \cup \{l\}$ has a solution, where $p$ is the set of positive literals in $c$.*

**Proof** (1) ⇒ (2): Obvious. For the converse suppose (2). Let $l$ be a negative literal and $f_l$ be a solution $p \cup \{l\}$. Then $f_l$ is a solution of $p$. As $p$ is normal, by the solution lemma $f_l$ is independent of $l$. Thus $f_l$ is also a solution of every negative literal in $c$. Hence $f_l$ is a solution of $c$. □

**Lemma 4** *For atomic constraint $d$ and any normal support $p$ consisting of positive literals, the following are equivalent.*

*(1) $p \cup \{\neg d\}$ has a solution.*

*(2) $p \cup \{d\}$ has no solution.*

**Proof** (2) ⇒ (1): Obvious.

(1) ⇒ (2): For the converse suppose that $p \cup \{\neg d\}$ has a solution $f$. Suppose that also $p \cup \{d\}$ has a solution $g$. As $p$ has at most one solution, it must be the case $f = g$. Hence $f$ is a solution of $p \cup \{d\}$. It follows from this $f$ is not a solution of $\neg d$, which is a contradiction. Thus $p \cup \{d\}$ has no solution. □

The above theorem and lemma are a basis of a naive unification algorithm. Let $p$ and $d$ be a finite normal support and an atomic constraint such that $p \cup \{d\}$ is normal. Applying the saturation method of our constraint rules to $p \cup \{d\}$ it is decidable whether there is a normal support $p'$ such that $p \cup \{d\} \subset p'$. So it is decidable whether $p \cup \{d\}$ is solvable or not. This remark will be used in a later section for application to the usual term structure and record structure.

We use Barwise's theorem also to show the compactness of the domain $V_{A,\kappa}$ with respect to the constraint language $L$. It is important that $L$ is assumed to have a fixed integer $\kappa$ such that any parametric set in the language has at most $\kappa$ elements.

**Lemma 5 (Compactness Lemma)** *For a positive constraint $c = \{p_n | 0 \leq n\}$ the following are equivalent.*

*(1) Each finite subset of $c$ has a solution in $V_{A,\kappa}$.*

*(2) $c$ has a solution in $V_{A,\kappa}$*

**Proof** Suppose (1). Then every finite subset of $c$ has a support. Hence by the previous lemma, $c$ has a normal support $c'$. Then by the Barwise's theorem, there is a solution $f$ to $c'$ in $V_{A,\kappa}$. The second half of the proof is obvious because a solution of a constraint $d$ is a solution of any subset of $d$. □

**Remark** We can not dispense with having the upper bound $\kappa$ of the size of sets. To see this, let $c = \{x_n = a_n | n \geq 0\} \cup \{x_n \sqsubset y | n \geq 0\}$, where $a_0 = \phi$, $a_n = a_{n-1} \cup \{a_{n-1}\}$. Note that $a_n$ has $n$ elements. Then $y$ must be not in $V_{A,\kappa}$, for any $\kappa$, though $y$ can be a hereditary finite set as a solution for each finite subset of $c$.

The next two lemmas are just corollaries of Compactness Lemma.

**Lemma 6** *For a set $c$ of atomic constraints in $L$, the following are equivalent.*

1. *There is no support of $c$.*
2. *There is some finite set $c'$ of $c$ such that there is no support of $c'$.*

**Lemma 7** *Given a positive constraint $c$ in $L$, the following are equivalent.*

1. *$c$ has a support.*
2. *$c$ has a solution in $V_{A,\kappa}$.*

Now we prove a more general case for compactness in which *negation* is involved. Note that a canonical constraint has not always a solution. For example, the constraint consisting of the following literals

$$x \neq y, x = \{x\}, y = \{y\}$$

is a canonical support but has no solution.

**Theorem 4 (Canonical Compactness)** *If $c$ is a canonical constraint in $L$ then the following are equivalent.*

1. *$c$ has a solution in $V_{A,\kappa}$.*
2. *Every finite subset of $c$ has a solution in $V_{A,\kappa}$.*

**Proof** Let $c = P \cup N$, where $P$ and $N$ are the set of positive and negative literals of $c$ respectively.

(2) $\Rightarrow$ (1): Suppose (2), i.e. every subset of $c$ has a solution. Since every subset of $P$ has a solution, it follows from the compactness of positive supports that there is a solution $f$ of $P$. We show that $f$ is also a solution of $N$.

Since $c$ is canonical, then for each negative literal $q$ in $N$ there is a finite subset $Q$ of $c$ which is a normal constraint containing $q$. By Condition 2, $Q$ has a solution, say, $h$. By the uniqueness of the solution, $f$ and $h$ must coincide on each variable of $q$. Thus $f$ satisfies every negative constraints in $N$. Therefore $f$ is a solution of $c$.

(1) $\Rightarrow$ (2): Obvious. □

**Theorem 5 (Solution Compactness)** *$V_{A,\kappa}$ is solution compact with respect to $L$.*

**Proof** We prove the case in which the given constraint $c = p \cup q$ is a (finite) constraint such that $p$ and $q$ are the positive and negative part of $c$ respectively. We assume that $q = \{\neg d_1, \cdots, \neg d_n\}$ for some integer $n \geq 0$. By transforming the given constraint into the disjunctive normal form, the general case is straightforwardly reduced to the simple case.

Let $x$ be a distinguished variable in $c$. Let $t \in V_{A,\kappa}$. Let $E$ be a system of equations

$$x_0 = b_0, x_1 = b_1, \cdots, x_n = b_n, \cdots$$

which defines $t$ as the unique solution to $x_0$. Assume that the constraint $c$ is not satisfiable with $x = t$. We divide it into two cases. Firstly, suppose that the constraint $E \cup p$ is not satisfiable. Then there is some finite subset $E'$ of $E$ such that

$$\{x = x_0\} \cup E' \cup p$$

is not satisfiable, for otherwise we can build a support for

$$\{x = x_0\} \cup E \cup p.$$

But this means that $x = t$ is a solution of $p$, which contradicts to the assumption. Now we have obtained a *neighborhood* $\{x = x_0\} \cup E'$ of $t$ which separates $t$ off the set of solutions to $x$ of $c$.

Secondly suppose that the constraint $E \cup p$ is satisfiable. Suppose that $E \cup p \cup \{d_i\}$ has a support for some $1 \leq i \leq n$ then $p \cup \{d_i\}$ is a neighborhood of $t$ which separates $t$ off $c$.

Finally, suppose that for any $1 \leq i \leq n$, $E \cup p \cup \{d_i\}$ has no support. Then no solution of $\{x = t\} \cup p$ can be extended to that of $\{d_i\}$. Then any solution of $\{x = t\} \cup p$ satisfies $\neg d_i$. Hence any solution of $\{x = t\} \cup p$ satisfies $\{\neg d_1, \cdots, \neg d_n\}$. Hence $x = t$ can be a solution of $p \cup q$. This is a contradiction. Therefore the last case is impossible. This concludes the proof. □

Our language $L$ has no functor symbol. Each constant (atom) $a$ of $L$ is interpreted as the atom $a$ itself. Also recall that $V_{A,\kappa}$ is defined as the largest class $D$ of $V_A$ such that the following axiom holds:

$$\forall x \in D(x = \phi \vee x \in A \ \vee \exists x_1 \in D \cdots \exists x_\kappa \in D \ x = \{x_1, \cdots, x_\kappa\}.$$

In ZFC$^-$/AFA, the structure $(V_{A,\kappa}, \{=, \sqsubset\})$ is characterized as a unique model of the constraint language $L = L_{A,\kappa}[X]$ Thus the satisfaction completeness of $L$ is trivially true because there is only one model, i.e., $V_{A,\kappa}$.

**Theorem 6 (Satisfaction Completeness)** *$L$ is satisfaction complete.*

We have proved the necessary properties required by the constraint logic programming schema to have CLP($V_{A,\kappa}$). Also note that solvability of constraints in $L$ is fully characterized using a syntactical notion of the support. I think that it is the following kind of theorem that CLP schema requires as for our case of CLP($V_{A,\kappa}$).

**Theorem 7** *Let $c = p \cup q$ be a set of literals in $L$ with a negative part $q$ and a positive part $q$. Then the following hold.*

- *$c$ is solvable.*
- *$p$ has a normal support $p'$ such that for each $(\neg d) \in c$ there is no support of $\{d\} \cup p'$.*

**Proof** Use lemma 4. □

## 4.1 Solving Equations with Unequations (=, ≠)

We describe here a result on constraints which is similar to Colmerauer's fundamental result[5]. Let $V = V_{A,\mathcal{K}}$. In this subsection, basic constraints are of the form $x = y$ or $x \neq y$.

A variable $x$ is *free in* $c$ if there is no equation $e$ in $c$ such that one side of $e$ is $x$ and the other side of $e$ is not $x$. A variable $x$ which is not free in $c$ is *redundant in* $c$ if for any equation $e$ in $c$ such that the one side of $e$ is $x$ the other side of $e$ is a variable. A set $c$ of equations is *trivial* if $c$ has a small support $c'$ in which each variable is free. An *identity* is an equation which has the same expression on both sides. A constraint $c$ is *valid* if $c$ is true for every assignment for $c$.

**Lemma 8** *Let $c$ be a set of equations. Then the following are equivalent.*

*(1) $c$ is valid.*

*(2) $c$ is trivial.*

*(3) $c$ has identities only.*

**Proof** (2)⇒ (3): Suppose (2). Let $c'$ be a small support of $c$ in which every variable is free. Let $u = v$ be any equation in $c$. As every variable is free in c', if either $u$ or $v$ is a variable then $u$ and $v$ must be the same variable. Also if either $u$ or $v$ is a set then both of them must be a set and moreover according to the definition of support the two sets must the same because c is trivial. Thus every equations in $c$ is identities.

(3) -> (1): As every identity is valid, every conjunction of identities is valid.

(1) ⇒ (2): Let $c$ be an equation in $c$. As $e$ is valid, it must be an identity. Hence $c$ has the support $\{u = u \mid u \in TC'(c)\}$, which is obviously trivial. □

**Lemma 9** *Let $c = p_1 \vee \cdots \vee p_n$ where each $p_i$ is a conjunction of equations. If no identity appears in $c$ then $c$ is not valid.*

**Proof** We prove by induction on the number $n$ of variables appearing in $c$. Assume that $n = 0$; i.e. there is no variable in $c$. The lemma is obvious in this case.

Assume that $n > 0$ and that the lemma is true for $n - 1$. Suppose that there is no identity in $c$. Let $x$ be any variable of $c$. Let $b$ be a set without any variable such that there is no occurrence of $b$ in $c$. Substitute $b$ for each occurrence of $x$ in $c$ to obtain $c'$. From the construction of $b$, $c'$ has no identity. The number of variables in $c'$ is $n - 1$. Then by the induction hypothesis $c'$ is not valid. Hence $c$ is not valid. □

**Lemma 10** *If $c = p_1 \vee \cdots \vee p_n$ is a disjunction such that each $p_i$ is a conjunction of equations. Then the following are equivalent.*

*(1) $c$ is valid.*

*(2) Some disjunct of $c$ is valid.*

**Proof** (2) ⇒ (1): Obvious.

(1) ⇒ (2): Suppose (1). Suppose that (2) is not the case. Then every disjunct $p_i$ of $c$ has at most one equation which is not an identity. Let $q_i$ be the set of equations $e$ in $p_i$ such that $e$ is not an identity. As $p_1 \vee \cdots \vee p_n$ is valid, by a simple logical operation, it follows that $q_1 \vee \cdots \vee q_n$ is valid. On the other hand, by the previous lemma, $q_1 \vee \cdots \vee q_n$ can not be valid since it has no identity. Here we have contradiction. Hence (2) must be the case. □

This lemma can be extended into the following more general one.

**Lemma 11** *Let $p$ be a normal support. If $c = p_1 \vee \cdots \vee p_n$ is a disjunction such that each $p_i$ is a conjunction of equations. Then the following are equivalent.*

*(1) $c$ is true whenever $p$ is true.*

*(2) There is some disjunct $p_i$ of $c$ such that $p_i$ is true whenever $p$ is true.*

*(3) There is a support $q$ of $p \cup \{p_i\}$ for some disjunct $p_i$ of $c$ such that the set of free variables in $q$ is the same as that of $p \cup \{p_i\}$. still free in $q$.*

**Proof** (2)⇒ (1): Obvious.

(3)⇒ (2): Obvious.

(1) ⇒ (3): We prove by induction on the number of free variables in $p \cup c$. Firstly suppose that there is no free variable in $p \cup c$. suppose that $p$ is true. Then there must be some disjunct $p_i$ such that $p \cup \{p_i\}$ is solvable. Hence $p \cup \{p_i\}$ has a small support. Secondly suppose that (1) ⇒ (3) is true for the number $m - 1$ of free variables in $p \cup c$, where $m \geq 1$. Let $x$ be a free variable in $p \cup c$. Let $a$ be a new atom. Replace $x$ in $p$ and $p_i$ by $a$ to obtain $p'$ and $p'_i$ respectively, where $1 \leq i \leq n$. Let $c' = \{p'_1, \cdots, p'_n\}$. By induction step there is a disjunct $p'_i$ of $c'$ such that of $p'_i$ is true whenever $p'$ is true. It follows directly from this that $p_i$ is true whenever $p$ is true. □

**Proposition 8** *Let $p$ be a solvable support consisting of equations with no redundant variable. Let $q$ be a set of unequations. Then the following are equivalent.*

*(1) For each unequation $u \neq v$ in $q$ there is no support $p'$ of $p \cup \{u = v\}$ such that the set of free variables of $p'$ is the same as that of $p$.*

*(2) $p \cup q$ is solvable.*

**Proof** (1)⇒ (2): Suppose (1). Suppose that the negation of (2) is the case. Equivalently suppose that for any solution $f$ of $p$ there is a unequation $u \neq v$ in $q$ such that $f$ is also a solution of $u = v$. Then by lemma 11 there is some $u \neq v$ in $q$ such that there exists a support $p'$ of $p \cup \{u = v\}$ such that the set of free variables of $p'$ is the same as that of $p$. This contradicts with (1). Thus (2) must be the case.

(2)⇒ (1): Suppose (2). Suppose that the negation of (1) is the case. Equivalently let $p'$ be a support of $p \cup \{u = v\}$ for some $u \neq v \in q$ such that the set of free variables of $p'$ is the same as that of $p$. By lemma 11, it follows that $p \supset u = v$ is always true. This is contradiction to (2). Hence (1) is concluded. □

**Lemma 12** *Let $p$ be a solvable support consisting of equations with no redundant variable. Then following are equivalent.*

*(1) $p \cup \{u \neq v\}$ is solvable.*

*(2) There is no support $p'$ of $p \cup \{u = v\}$ such that the set of free variables of $p'$ is the same as that of $p$.*

**Proof** (1)⇒ (2): Suppose (1) and let $p'$ be a support of $p \cup \{u = v\}$ such that the set of free variables of $p'$ is the same as that of $p$. By lemma 11, it follows that $p \supset u = v$ is always true. This is contradiction to (1). Hence (2) is concluded.

(2)⇒ (1): Suppose (2). Suppose that the negation of (1) is the case. Equivalently suppose that for any solution $f$ of $p$ such that $f$ is also a solution of $u = v$. Then by lemma 11 again there is some $u \neq v$ in $q$ such that there exists a support $p'$ of $p \cup \{u = v\}$ such that the set of free variables of $p'$ is the same as that of $p$. This contradicts with (2). Thus (1) must be the case. □

The following theorem is a easy combination of the previous two lemmas.

**Theorem 9 (Independence Theorem)** *Let $p$ be a solvable support consisting of equations with no redundant variable. Let $q$ be a set of unequations. Then the following are equivalent.*

*(1) $p \cup q$ is solvable.*

*(2) $p \cup \{l\}$ is solvable for each $l \in q$.*

**Remark** An analogy from a simple set theory might be helpful for understanding the independence theorem: Given sets $p, q_1, \cdots, q_n$ the following are equivalent, where $\overline{x}$ means the complement of $x$.

- $p \cap \overline{p_1} \cap \cdots \cap \overline{p_n} = \phi$
- $p \subset p_1 \cup \cdots \cup p_n$.

In addition to this general property, the independence theorem rests on a special property that the 'total space' $X$ can not be covered by any finite family of 'subspaces' with properly lower 'dimensions' than that of $X$. In general there is some constraint language in which the independence theorem does not hold. For example, take the set of literals,

$$x = x, x \neq a, x \neq b$$

in the domain $D = \{a, b\}$. Both of $x = x \wedge x \neq a$ and $x = x \wedge x \neq b$ are satisfiable, but the total constraint is unsolvable.

**Remark** It is interesting to investigate to extend this proposition for the subsumption relation. Normal Unification Lemma was for the case in which the constraint is *normal*, i.e. there is no free variable in the constraint.

# 5 Coinductive Semantics of Horn Clauses

## 5.1 Horn Clause with Constraint

Let $\Pi$, $\Phi$, and $\Gamma$ be sets of *predicate symbols*, *function symbols*, and *constant symbols*, respectively. If $p \in \Pi$ and $x_1, \cdots, x_n$ are variables then $p(x_1, \ldots, x_n)$ is an *atomic goal*. A *goal* is a finite set of atomic goals. A *constraint Horn clause* is a triple $(h, c, g)$ such that $h$ is an atomic goal $p(x_1, \ldots, x_n)$, $c$ is a constraint, and $g$ is a set of atomic goals, where $x_i \neq x_j$ $(i \neq j)$. We write for $(h, c, \{b_1, \cdots, b_n\})$

$$h : -c|b_1, \cdots, b_n.$$

A constraint Horn clause is also called *a program clause*.

## 5.2 Computation Tree

A *computation state* (state for short) is a triple $(c, g, V)$ of a goal $g$, a constraint $c$ which has a support, and a set $V$ of variables such that $\mathcal{V}(g) \cup \mathcal{V}(c) \subset V$.

For the given goal $g$, the *initial state* is the state $(true, g, \mathcal{V}(g))$. The atomic constraint *true* is the constraint which is always true. A function which assigns a program clause to each atomic goal in $g$ is called a *choice for* $g$. Note that the

set $S_g$ of choices for $g$ is finite since $g$ and the program are finite sets. The $V$-component of states are often omitted when the context is clear. Let $s = (c, g, V)$ and $s' = (c', g', V')$ be two states. Let $\gamma$ be a choice for $g$. A triple $(s, \gamma, s')$ is a *transition* written

$$s \rightarrow_\gamma s'$$

if there are 'renamed versions' $C_a$:

$$h_a : -c_a | g_a$$

of clauses $\gamma(a)$ for $a \in g$ such that the following hold.

- $\mathcal{V}(C_a) \cap \mathcal{V}(C_{a'}) = \phi$. $(a \neq a')$
- $V \cap \mathcal{V}(C_a) = \phi$.
- $g' = \bigcup\{g_a | a \subset g\}$.
- $c'$ is an equivalent constraint to the following.

$$c \wedge \bigwedge_{a \in g} \{a = h_a\} \wedge c_a.$$

- $V' = V \cup \bigcup\{\mathcal{V}(C_a) | a \in dom(\gamma)\}$.

Let $\mathcal{P}$ be the largest set $M$ such that if $x \in M$ then $x = (s, b)$ for some $s$ and $b$ such that the following hold.

- $s$ is a state.
- $b$ is a partial function defined on the set of choices for the goal component of $s$.
- $ran(b) \subset M$.
- If $\gamma \in dom(b)$ then $s \rightarrow_\gamma s'$, where $b(\gamma) = (s', b')$ for some $b'$.

A binary order $\prec$ is the maximum relation in $\mathcal{P}$ such that if $x \prec y$ then the following conjunction holds for some $s, b, b'$.

(1) $x = (s, b)$ and $y = (s', b')$.

(2) $dom(b) \subset dom(b')$.

(3) If $\gamma \in dom(b)$ then $b(\gamma) \prec b'(\gamma)$.

Note that we identify states up to the equivalent constraint component of states. A maximal element of $\mathcal{P}$ is called a *computation tree*.

## 5.3 Solution Tree

Let $D = V_{A,\kappa}$ be the domain of the constraint language $L$. An *interpretation* of the program is a function $\mathcal{I}$ which assigns a subset of $D^n$ to predicate symbols $p$, where $n$ is the arity of $p$. An interpretation $\mathcal{I}$ is a *model* of the program $P$ if $(a_1, \cdots, a_n) \in \mathcal{I}(p)$ then there are a program clause

$$p(x_1, \ldots, x_n) : -c \mid g$$

and assignment $f$ such that the following hold.

- $f(x_i) = a_i$ for $1 \leq i \leq n$.
- $V_{A,\kappa}, f \models c$.

- if $q(z_1,\ldots,z_m) \in g$ then $(f(z_1),\ldots,f(z_m)) \in \mathcal{I}(q)$.

where $m$ is the arity of the predicate symbol $q$. By general theory of Aczel[1], since the Horn clause program $P$ can be seen a monotone operator on interpretations, there exists the largest model $M_P$ of the program.

Let $c$ and $g$ be a constraint and a goal, respectively. Let $f$ be an assignment such that $\mathcal{V}(c) \cup \mathcal{V}(g) \subset dom(f)$. $f$ is a *solution* of $(c,g)$ if $V_{A,\kappa}, f \models c$ and $M_P, f \models a$ for each atomic subgoal $a$ of $g$.

Let $\mathcal{A}$ be the set of triples $(f,c,g)$ which satisfies the following.

- $\mathcal{V}(c) \cup \mathcal{V}(g) \subset dom(f)$.
- $f$ is a solution of $(c,g)$.
- $c$ has a support.

A binary relation $\rightarrow$ is defined to be the largest relation on $\mathcal{A}$ such that if

$$(f,c,g) \rightarrow (f',c',g')$$

is defined then there is a function $\gamma$ from $g$ to the program $P$ such that there are 'renamed program clauses' $C_a$:

$$h_a : -c_a | g_a$$

of $\gamma(a)$ which satisfies the following.

- $f'$ is an extension of $f$.
- $g' = \bigcup\{g_a | a \in g\}$.
- $c'$ is the constraint equivalent to

$$c \wedge \bigwedge_{a \in g} \{a = h_a\} \wedge c_a.$$

- $\mathcal{V}(C_a) \cap \mathcal{V}(C_{a'}) = \phi$ $(a \neq a')$.
- $dom(f) \cap \mathcal{V}(C_a) = \phi$ for any $a \in g$.

We write $(f,c,g) \rightarrow_\gamma (f',c',g')$ indicating $\gamma$ explicitly.

Let $P$ be the given program. Let $\mathcal{W}$ be the largest set $M$ such that if $x \in M$ then $x = (s,b)$ for some $s = (f,c,g) \in \mathcal{A}$ and a function $b$ such that the following hold.

- $dom(b)$ consists of choices for $g$.
- $ran(b) \subset M$.
- $s \rightarrow_\gamma s'$ if $b(\gamma) = (s',b')$ for some $b'$.

Let $\prec$ be the largest binary relation on $\mathcal{W}$ such that if $x \prec y$ then the following conjunction holds.

- $x = (s,b)$ and $y = (s,b')$ for some $s, b, b'$.
- $dom(b) \subset dom(b')$.
- If $\gamma \in dom(b)$ then $b(\gamma) \prec b'(\gamma)$.

A *solution tree* is a maximal element of $(\mathcal{W}, \prec)$. A *path* is a minimal element of $\mathcal{W}$ with respect to $\prec$.

**Lemma 13** *If $x = (s,b)$ is a path such that $dom(b) \neq \phi$ then the following conjunction hold for some $\gamma$, $s'$, and $b'$.*

- $dom(b) = \{\gamma\}$
- $b(\gamma) = (s', b')$ *is a path.*
- $s \rightarrow_\gamma s'$.

**Proof** It is obvious by the definition of path. □

Let $\mathcal{Z}$ be either $\mathcal{W}$ or $\mathcal{P}$. Let $X$ and $Y$ be disjoint sets of new variables. Let $E$ be a system of equations. and $z$ is a distinguished variable in $X$. A quadruple $(E, X, Y, z)$ is is a *construction with the root* $z$ if the following hold.

- For each $x \in X$ there is an equation $x = (u, y) \in E$ such that $u \in \mathcal{Z}$ and $y \in Y$.
- For each $y \in Y$ there is an equation $y = b \in E$ such that $b$ is a function which assigns variables in $X$ to choices.
- If $x = (u, y) \in E$, $y = b \in E$, $b(\gamma) = x'$, and $x' = (u', y') \in E$ for some $y'$ then $u \rightarrow_\gamma u'$.
- every variable in $X$ can be accessed from $z$.

By *the value of construction* $(E, X, Y, z)$ we mean the value of the root variable $z$ of the solution of the construction with respect to $X \cup Y$.

**Lemma 14** *Given* $(a_1, \cdots, a_n) \in \mathcal{I}(p)$ *and a path* $p - ((f, true, \{p(x_1, \cdots, x_n)\}), q)$ *where* $f(x_i) = a_i$ $(1 < i \leq n)$, *there is a construction* $E$ *with* $z$ *such that the value of the construction is* $p$.

**Proof** $E$ is constructed in a standard way by induction on depth from the root variable $z$. □

**Lemma 15** *The following are equivalent.*

*(1)* $(a_1, \cdots, a_n) \in \mathcal{I}(p)$.

*(2) There is a path* $((f, true, \{p(x_1, \cdots, x_n)\}), b)$ *for some* $b$ *such that* $f(x_i) = a_i$ $(1 \leq i \leq n)$.

**Proof** Suppose (1). By the previous lemma, there exists a construction $E$ with the root $z$ which has an equation $z = ((f, true, \{p(x_1, \cdots, x_n)\}), z') \in E$, where $f(x_i) = a_i$ $(1 \leq i \leq n)$. By the solution lemma, there exists a unique solution of the system. The solution to the distinguished variable $z$ is clearly a solution tree which satisfies Condition (2).

For the converse, suppose (2). Then $f$ is a solution of $p(x_1, \ldots, x_n)$ by definition. Hence $(a_1, \cdots, a_n) = (f(x_1), \cdots, f(x_n)) \in \mathcal{I}(p)$. □

**Lemma 16** $(\mathcal{W}, \prec)$ *is chain complete.*

**Proof** Suppose that the following monotone sequence is given.

$$x_0 \prec x_1 \prec \cdots \prec x_n \prec \cdots.$$

Let us define a sequence $X_i$ $(i \geq 0)$ inductively.

- $X_0 = \{x_i | i \geq 0\}$.

- $X_{i+1} = \{b(x) | \exists s\ (s,b) \in X_i, x \in dom(b)\}$.

Let $X = \bigcup\{X_i | i \geq 0\}$ and $\Lambda = \bigcup\{dom(b) | \exists s\ (s,b) \in X\}$. Let $\Lambda^*$ be the set of finite sequences over $\Lambda$.

Let us define a partial operation of $\alpha \in \Lambda^*$ on $x = (s,b) \in X$ inductively.

- $x(\varepsilon) = x$.
- $x(\gamma\alpha) = b(\gamma)(\alpha)$ where $(\gamma \in \Lambda)$.

where $\varepsilon$ is the empty sequence.

Then consider the system of equations for $\alpha \in \Lambda^*$

$$y_\alpha = (a_\alpha, Y_\alpha)$$

such that

- there is some $x_i$ and $b$ such that $x_i(\alpha) = (a_\alpha, b)$
- $(\gamma, y_{\alpha\gamma}) \in Y_\alpha$ if and only if there is some $x_j$ such that $x_j(\alpha\gamma)$ is defined.

This system of equations are well defined, though some $\alpha \in \Lambda$ may fail to have the corresponding equation. Hence by the solution lemma there is a unique solution of the system. Let $y$ be the solution to $y_\varepsilon$. Now let us show that $y$ is the least upper bound of the family $x_i$ $(0 \leq i)$. Suppose that $z$ is an upper bound of the family, i.e. $x_i \prec z$ for all $0 \leq i$. By comparing the defining system of equations for $z$ with that for $y$, it follows that $y \prec z$. □

**Lemma 17 (Maximal Solution Tree Lemma)** *If $x$ is a solution tree in $\mathcal{W}$ there is a maximal one $x'$ in $\mathcal{W}$ such that $x \prec x'$.*

**Proof** Let $X = \{z \in \mathcal{W} | x \prec z\}$. By previous lemma, any monotone family of elements of $X$ has the limit with respect $\prec$. Hence, by Zorn's lemma, there exists a maximal element $x'$ in $X$. □

## 5.4 Soundness and Completeness

Our formulation of soundness and completeness is that there is a *simulation* between $\mathcal{W}$ and $\mathcal{P}$.

Let $\mathcal{T}(S,L)$ be the largest set $X$ of sets such that if $x \in X$ then $x = (a,b)$ where $a \in S$ and $b$ is a partial function from $L$ to $X$. An element of $X$ is called a *tree* over $S$ and $L$. Let $X_i = \mathcal{T}(S_i, L)$ for $i = 1, 2$ and $f$ is a function from $S_1$ to $S_2$ then there is a function $F$ from $X_1$ to $X_2$ such that

$$F((x,y)) = ((f(x), F``y))$$

where $F``y = \{F(z) | z \in dom(y)\}$. We write $f^*$ for $F$. Particularly we call $f^*$ a *hereditary projection* if $f$ is a projection.

A subset $R$ of $\mathcal{W} \times \mathcal{P}$ is a *simulation between $\mathcal{W}$ and $\mathcal{P}$* if the following hold whenever $xRy$.

- $x = ((f,c,g),b)$ and $y = ((c,g),b')$.
- If $dom(b) \neq \phi$ then $b(\gamma)Rb'(\gamma)$ for some $\gamma \in dom(b) \cap dom(b')$.

**Lemma 18** *There is the maximum simulation.*

**Proof** Take the union of all simulations between $\mathcal{W}$ and $\mathcal{P}$. □

**Lemma 19 (Simulation Lemma)** *For the maximum simulation $R$, the following are equivalent.*

*(1)* $xRy$.

*(2) There exist paths* $p \prec x$ *and* $q \prec y$, *such that* $pRq$.

**Proof** Suppose (2). Then by definition of $p \prec x$, it follows that $x = (s, b)$, $p = (s, \{(\gamma, p')\})$, and $p' \prec b(\gamma)$ for some path $p'$. Since $pRq$ also it must be the case that $y = (u, b')$, $q = (u, \{(\gamma, q')\})$, $q' \prec b'(\gamma)$ and $p'Rq'$ for some $u$, $b'$, and $q'$. Hence from the last two formulae $b(\gamma)Rb'(\gamma)$*isobtained*. Since $R$ is maximal, $xRy$ is obtained.

For the converse, suppose (1). By definition of $xRy$ we can construct a sequence $x_i$, $y_i$ $(i \geq 0)$, where $x_0 = x$ and $y_0 = y$, such that the following hold.

- $x_i R y_i$.
- $x_i = ((f_i, c_i, g_i), b_i)$.
- $b_i(\gamma) = x_{i+1}$.
- $y_i = ((c_i, g_i), d_i)$.
- $d_i(\gamma) = y_{i+1}$.

From this we write a system of equations as follows.

$$u_i = ((f_i, c_i, g_i), \{(\gamma_i, u_{i+1})\})$$

$$v_i = ((c_i, g_i), \{(\gamma_i, v_{i+1})\})$$

By the solution lemma, let $p$ and $q$ be the solutions to $u_0$ and $v_0$ respectively. By definition of $\prec$ and $R$, it is easy to see that $p \prec x$, $q \prec y$, and $pRq$. □

Let $\pi$ be a projection such that $\pi((x, y, z)) = (y, z)$.

A path of a computation tree *is canonical derivation* if the conjunction of the constraint appearing in the path is canonical. A computation tree *is canonical* if each path of the tree is canonical.

**Theorem 10 (Soundness)** *For any canonical computation tree $y$ in $\mathcal{P}$ there is a solution tree $x$ in $\mathcal{W}$ such that $xRy$.*

**Proof** Let $q$ be any path of a computation tree $y$ in $\mathcal{P}$. We apply the canonical compactness theorem to select a global solution $f$ to the path $q$. Make a family of solutions by restricting $f$ to each step on the path of $y$. Write a system of equations

$$y_i = ((c_i, g_i), z_i)$$

$$z_i = \{(\gamma_i, y_{i+1})\}$$

for $i \geq 0$ such that $y_0 = q$, i.e. $q$ is the solution to $y_0$.

Apply the solution lemma to the following system of equations

$$x_i = ((f_i, c_i, g_i), w_i)$$

$$w_i = \{(\gamma_i, x_{i+1})\}$$

where $f_i$ is the above mentioned restriction of $f$ to $\mathcal{V}(c_i) \cup \mathcal{V}(g_i)$ for $i \geq 0$. Then let $p$ be the solution to $x_0$. It is clear that $\pi^*(p) = q$. By Maximal Solution Tree Lemma, there is a solution tree $x$ such that $p$ is a path of $x$. Hence by Simulation Lemma $xRy$. □

**Theorem 11 (Completeness)** *For any solution tree $x$ there is a computation tree $y$ in $\mathcal{P}$ such that $xRy$.*

**Proof** By definition of $\prec$, $xR\pi^*(x)$ is true. By Maximal Solution Lemma for $\mathcal{P}$, there exists a computation tree $y$ in $\mathcal{P}$ such that $\pi^*(x) \prec y$. Therefore $xRyisobtained.$ □

**Theorem 12 (Negation As Failure)** *Let $g$ be a goal with variables $x_1, \cdots, x_n$ and $q$ be the computation tree of $g$, i.e. $q = ((true, g), b)$ for some $b$. Assume that an assignment $f$ such that $f(x_i) = \xi_i$ where $\xi_i \in V_{A,\kappa}$ is not a solution of $g$. Then there is a finite constraint $c$ such that $f$ is a solution of $c$ and that for any $b'$ there is no computation tree of the form $((c, g), b')$.*

**Proof** For any path $p \prec q$ such that

$$q_0 = p = ((c_0, g_0), \{(\gamma_0, q_1)\}),$$

$$q_1 = ((c_1, g_1), \{(\gamma_1, q_2)\}),$$

$$q_2 = ((c_2, g_2), \{(\gamma_2, q_3)\}),$$

$$\vdots$$

where $c_0 = true$, $g_0 = g$, and $\gamma_0 \in dom(b)$, there is some $q_j$ in the path such that no extension of $f$ satisfies $c_j$, for otherwise $f$ can be extended to the solution of $q$. By applying the standard argument of König's lemma, the set $d$ of such constraints $c_j$ is finite:

$$d = \{c_1, \cdots, c_n\}$$

for some integer $n$. Then by Solution Compactness Theorem we can give a finite 'cover' constraint $c$ of $(\xi_1, \cdots, \xi_n)$ such that $c$ has no common solution with $c_j$ above. It follows from the construction of the constraint $c$ that for any $b'$ there is no computation tree of the form $((c, g), b')$ □

# 6 Towards Application to Term and Record

In this section we see that the following standard constraint languages are embedded into $L$.

- Unification theory over (infinite) trees.
- Unification theory over (infinite) records.

The domains of trees and records are two special subclasses of $V_{A,\kappa}$, respectively.

Let us recall some notations: $X$ is an infinite set of *parameters*. $A$ is a finite set of *atoms*. $\kappa$ is a positive integer. $V_{A,\kappa}[X]$ is the set of bounded hereditary finite sets with $\kappa$ over $A \cup X$.

## 6.1 Term

Let $\Sigma$ be a signature. Each symbol $\sigma \in \Sigma$ is assigned a non-negative integer arity. A function $b$ *is an assignment for* a symbol $\sigma \in \Sigma$ if $dom(b) = \{1, \cdots, n\}$ , where $n$ is the arity of $\sigma$.

Let $T_\Sigma[X]$ be the least set $M$ such that the following hold.

- $X \subset M$.
- If $\sigma \in \Sigma$ and $b$ is an assignment for $\sigma$ such that $ran(b) \subset M$ then $\sigma \in M$.

Let $T^*_\Sigma[X]$ be the largest set $M$ such that if $x \in M$ then one of the following hold.

- $x \in X$.
- $x = (\sigma, b)$ for some $\sigma \in \Sigma$ and and an assignment $b$ for $\sigma$ such that $ran(b) \subset M$. .

An element of $T^*_\Sigma[X]$ is called a *term*. An element of $T_\Sigma[X]$ is called a *(first order) finite term*. An element of $T^*_\Sigma[X] \backslash T_\Sigma[X]$ is called an *infinite term*. An element of $T^*_\Sigma[\phi]$ is called a *ground term*.

## 6.2 Record as Hereditary Function

Let $\Delta$, $\Gamma$ be two finite sets of *labels* and *constants*, respectively. Let $R^*_{\Delta,\Gamma}[X]$ be the largest set $M$ such that if $f \in M$ then $f$ is a function from a subset of $\Delta$ into $\Gamma \cup X \cup M$. Let $R_{\Delta,\Gamma}[X]$ be the smallest set $M$ of functions $f$ from a subset of $\Delta$ into $\Gamma \cup X \cup M$.

Elements of $R^*_{\Delta,\Gamma}[X]$ and $R_{\Delta,\Gamma}[X]$ are called *a record* and *finite record* respectively. A non-finite record is called an *infinite record*. The null function $\phi$, i.e. the empty set, is a record by definition. An elements of $R^*_{\Delta,\Gamma}[\phi]$ is a *pure* record.

A binary relation $\sqsubset$ is defined to be the largest relation between pure records such that if $x \sqsubset y$ then the following hold.

- If either $x \in \Gamma$ or $y \in \Gamma$ then $x = y$.
- If $x$ and $y$ are functions then $dom(x) \subset dom(y)$ and $x(a) \sqsubset y(a)$ for any $a \in dom(x)$.

It is clear that the set of pure records is a partial ordered by $\sqsubset$. We call this relation *hereditary subfunction order*. This fact is in contrast with that of the hereditary subset relation, which is not partial order relation on $V_{A,\kappa}$.

## 6.3 Unification over Terms and Records (=, $\sqsubset$)

The input of the unification algorithm is a finite set of equations over parametric terms and records. The output is either the set of solved forms of the input if it exists or undefined otherwise. The solved form is a system of equations.

The unification algorithm goes as follows: Given a set of equations, repeat the following applicable steps to the set as far as possible until there is no applicable one. When it terminates, check whether there is a *conflicting* equations or not. It is easy to see that any sequence of these steps will always terminate. This algorithm is an extension of the standard unification to the records.

(1) If $x = x$ is in the set then remove it.

(2) If $x = y$ is in the set then replace all the occurrences of $y$ with $x$.

(3) If $u = x$ is in the set for a non variable $u$ then replace it with $x = u$.

(4) If $x = u$ and $x = v$ are in the set for non–variable terms $u$ and $v$ then remove one of the equations whose size is not less than the other and add $u = v$.

(5) If $f(u_1, \cdots, u_n) = f(v_1, \cdots, v_n)$ is in the system then replace it with the $n$ equations $u_i = v_i$ for $1 \leq i \leq n$.

(6) If $\{(a_1, u_1), \cdots, (a_n, u_n)\} = \{(a_1, v_1), \cdots, (a_n, v_n)\}$ is in the system then replace it with the $n$ equations $u_i = v_i$ for $1 \leq i \leq n$.

A set $S$ of parametric terms and records is called a *conflict* if one of the following conditions hold.

- $S$ has both a record and a non variable term.
- $S$ has two terms with different prime functors to each other.

The equation $u = v$ is called a *conflict* if the set $\{u, v\}$ is a conflict. For example, equations, $1 = 2$, $f(x) = \{(a, y)\}$, $f(a) = f(a, b)$ are conflicts, respectively.

## 6.4 UNION–FIND Based Unification

There is an alternative algorithm based on UNION–FIND procedure. UNION–FIND algorithm can be applied to the records unification. As a fact UNION–FIND algorithm can be seen as computing minimum bisimulation relation generated by the given set of equations. The unification problem for a set of equations can be translated into a UNION FIND algorithm[2] in the following way, which is well known to have an almost linear complexity. First of all, we can assume without loss of generality that any argument of terms is either a constant or variable and that no equation has non–variable terms at the both sides. If there is an equation, say $x = \{(a, \{(b, c)\}), (d, e)\}$, which does not satisfy the assumption, we replace the subterm $\{(b, c)\}$ with a new variable $y$, and add the equation $y = \{(b, c)\}$ to the system. It is clear that by repeating this replacements we obtain in a finite number of steps a system of equations which satisfies the assumption. Also it is clear that the number of generated variables is proportional to the 'size' of the given problem.

Secondly, we transform the unification problem into the UNION–FIND problem by replacing each of the equations with UNION–FIND commands according to the following rules.

- $x = u \Rightarrow x = u$ where $u$ is a variable.
- $x = \{(a_1, u_1), \cdots, (a_n, u_n)\} \Rightarrow ARG^x_{a_1} = u_1, \cdots, ARG^x_{a_n} = u_n$.
- $x = f(u_1, \ldots, u_n) \Rightarrow FUN_x = f, ARG^x_1 = u_1, \cdots, ARG^x_n = u_n$.

where $x$ and $y$ are variables, $u$, $u_i$ are a variable or constant, $ARG^\beta_\alpha$ and $FUN^\beta_\alpha$ are new variables.

Let $\Omega$ be the set of variables and constants which appears in the final list of commands. It is easy to see that both the size of $\Omega$ and the size of the UNION–FIND commands are proportional to the size of the input problem. Since it is well known that UNION–FIND problem has an almost linear algorithm, it follows that our unification problem has an almost linear complexity algorithm. Note that this result does not depend on whether occur–check is performed or not.

## 6.5 Boolean Constraint

It is observed that constraint solving is often reduced to Boolean constraint, i.e., propositional calculus. The Boolean constraint may be useful in the case that the constraint language is given as a non Horn theory. In the non Horn case the naive closure method or saturation method will not work well, though these method does work well in the standard unification. The constraint may has negative or disjunctive constraints. It is often the case that there exists some translation method from the constraints into Boolean combination of atomic ones. We have shown above an important idea of the elimination of functor or set notation. The following example is to eliminate set expressions:

$$\{(a_1, u_1), \cdots, (a_n, u_n)\} \sqcap x \Rightarrow u_1 \sqsubset x/a_1 \wedge \cdots \wedge u_n \sqsubset x/a_n$$

where complex expressions like $x/a$ are treated a single variable. Fundamental operations are the closure operation of the congruence relation such as bisimulation. UNION FIND is one example of such kind of operation. Another important one is the transitive closure operation of partial order relation like subsumption on the domain of the records.

Many non-trivial class of constraint languages can be reduced to Boolean constraint language. For instance, Johnson's attribute-value logic[8] and Smolka's feature logic[13] are based on some translation from their logic into some sublanguage of quantifier free first order logic, i.e., roughly speaking, propositional calculus or Boolean constraint. So we review a Boolean constraint language and put here some necessary stuff in order from the point of constraint solving. By doing so we show that our constraint language over non-well-founded sets can be thought even as an implementable language of practical efficiency.

*A Boolean algebra* $B = (D, \wedge, \vee, \neg, 1, 0)$ and a *Boolean expression* are defined as usual. Also a Boolean ring $R = (E, \cdot, +, 1, 0)$ is defined as usual.

Given a Boolean algebra $B = (D, \wedge, \vee, \neg, 1, 0)$ we define the Boolean ring $R = (D, \cdot, +, 1, 0)$ by the following equations.

$$xy = x \cdot y = x \wedge y$$

$$x + y = (x \wedge \neg y) \vee ((\neg x) \wedge y).$$

It is easy to see that the following hold.

$$x \vee y = (x + y) + xy$$

$$\neg x = 1 + x$$

$$x \wedge y = xy.$$

These rule gives a translation $\tau$ of Boolean expressions $e$ into Boolean ring expressions $\tau(e)$. Of course $e$ is satisfiable in $B$ if and only if $\tau(e)$ is satisfiable $R$.

In Boolean ring, the followings hold:

- If $ax = b$ then $ab = b$.
- If $ax = b$ then $(1 + a)(x + b) + b = x$.
- If $x = (1 + a)y + b$ then $ax = b$.

As a corollary, the following conditions are equivalent.

- $ax = b$ is satisfiable.
- $ab = b$ is satisfiable.

- $x = (1 + a)y + b$ for some $y$.

Hence it is well understood that there is a well known satisfiability check algorithm of a Boolean ring expression based on variables elimination. See Dincbas et al[6], for instance.

# 7 Concluding Remark

By the use of non–well–founded sets and compact constraint, declarative and operational semantics becomes essentially the same one. In this point, our semantics is related to constructive type theory in the sense that elements of the semantics of the queries are proof trees which are decorated by satisfiable (normalizable) constraints at each node. A goal can be thought as a non-canonical constraint.

As an application of the present work, I would like to use it for semantics of meta predicates of logic programming languages, which is a related motivation of this work to STASS(Situation Theory and Situation Semantics). Meta predicates such as *var* or *cut* of Prolog is essentially defined operationally. I hope that the clear structure of our semantics in AFA sets will provide a good new setting for the semantics of these meta predicates. A guiding idea is that meaning of commands are as constraints of computation states or situations on the trees as hypersets. Also I feel that constraint logic programming can be safely renamed to be as *Infon Logic Programming* by seeing constraints as infon or soa in the sense of the STASS literature. For instance we would like to see that the infon $x = y$ is supported by a physical computation state $s$, i.e.

$$s \models \ll =, x, y \gg .$$

The situation or state $s$ have variable cells for $x$ and $y$ with pointers from $x$ to $y$. However details are outside of the present paper.

**Acknowledgments** The author would like to thank Hideki Yasukawa, Kaoru Yoshida, Makoto Imamura, Satoshi Tojo, Hideyuki Nakashima, Koiti Hasida for giving useful comments on this work.

# References

[1] P. Aczel. *Non-well founded set theoy.* CSLI lecture note series, 1988.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison–Wesley, 1974.

[3] J. Barwise. *The Situation in Logic.* CSLI Lecture Notes 17. CSLI Stanford, 1989.

[4] J. Barwise and J. Etchemendy. *The Liar: An Essay on Truth and Circular Propositions.* Oxford Univ. Press, 1987.

[5] A. Colmerauer. Equations and unequations on finite and infinite trees. In *Proceedings of the Second International Conference on Fifth Generation Computer Systems*, Tokyo, 1984.

[6] Mehmet Dincbas, Helmut Simonis, and Pascal van Hentenryck. Extending equation solving and constraint handling in logic programming. Technical Report IR-LP-2203, ECRC, 1987.

[7] J. Jaffar and J.-L. Lassez. Constraint logic programmig. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, 1987.

[8] M. Johnson. *Attribute-Value Logic and the Theory of Grammar*. CSLI Lecture Notes 16. Center for the Study of Language and Information, Stanford University, 1987.

[9] Peter J.Suckey. On the foundation of constraint logic programming. Technical report, Department of Computer Science Monash University, Victoria 3168, Australia, August 1987.

[10] K.Mukai. A system of logic programming for linguistic analysis. Technical Report TR-540, ICOT, 1990. To Appear also from SRI Tokyo series.

[11] D. Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI Conference*, number 104 in Springer Lecture Notes in Computer Science. Springer, 1981.

[12] G. Smolka. Feature constraint logics for unification grammars. Technical Report IWBS report 93, IBM Deutschland GmbH, 1989.

[13] G. Smolka. Feature logic with subsorts. Technical Report LILOG Report 33, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, W. Germany, May 1989.