ICOT Technical Report: TR-454

TR-454

Partial Unification over Records

by

K. Mukai

March, 1989

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

Partial Unification over Records

Kuniaki Mukai

Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan

Abstract

This paper[1] introduces a domain of infinite trees for linguistic information. An infinite tree is an extension of a list of attribute-value pairs or a directed acyclic graph. A tree is supposed to be arity-free. The main results of this paper are that (1) a theory of partial unification is presented, (2) a solution of partial equality is defined over the domain, and (3) the equivalence between partial unifiability and solvability is established.

Based on these results, a class of Horn clause programs is introduced and is given both a maximal semantics and an operational SLD semantics over the domain. Soundness and completeness are obtained with respect to the two semantics. Furthermore, as just a contraposition of it, soundness and completeness of the negation as failure rule are obtained.

1 Introduction

Linguistic theories are theories about *linguistic information* and they are described in terms of constraints over linguistic information. Then, how is linguistic information represented? A non-controversial answer to this may be that linguistic information is represented in a record-like data structure.

In unification-based grammar formalism, for instance, a linguistic object is represented in a kind of *(finite) directed acyclic graph (DAG)*. DAGs are often called feature sets. The formalism studies constraints over linguistic objects, whose basic form is almost always given in phrase structure rules.

Now, what is the declarative semantics of these linguistic constraints over the DAG domain? Pereira and Shieber [84] gave a general picture for such semantics based on Scott's domain theory. However, they did not give enough details so that completeness and soundness results, for instance, would be obtained for the semantics.

The objective of this paper is to propose a domain of a record-like structure for linguistic information. As a record-like structure, we have in mind a list of attribute-value pairs. The domain is called the PTT domain, which can be seen as an extension of the (complete) Herbrand universe. We will give a simple constraint theory similar to the equality axioms for finite Herbrand terms. The

[1]This is a revised version of the original one by eliminating several errors and confusions. Also this version will appear under the title " A Constraint Logic Programming over Record Algebra" in the proceeding of the Japanese-Swedish-Italian joint workshop on Concurrent Logic Programming and Constraint Logic Programming at Pisa, June 26-27-28, 1989.

most important aspect of our theory is that we can construct a solution for any *consistent* set of our constraints. Based on this lemma, we will prove that our semantics of a grammar or a program is *sound* and *complete*. Our semantics, however, takes the largest model instead of the least one which is usual. Thus, we obtain a unified declarative semantics for PATR-II like grammar formalism and logic programming over the PTT domain.

Let us explain one of the non-trivial points of this research. Linguistic information has *partiality*, that is, there is a partial order relation between items of information. For instance, in our notation, introduced later, the information $\{m/a, n/b, l/c\}$ *is said to be larger* than $\{m/a, n/b\}$. Moreover, $\{m/a, n/b\}$ and $\{l/c, n/b\}$ can *be merged* into $\{m/a, n/b, l/c\}$. By contrast, in the standard logic programming over the Herbrand universe, two terms which have different numbers of arguments, say, $f(a,b)$ and $f(a,b,c)$, are quite different objects. Thus, this partiality in our domain causes a new difficulty in generalizing the satisfiability notion of equations in the Herbrand universe to our PTT domain. The technical heart of this paper is to define a new satisfiability relation so that satisfiability and unifiability are equivalent as in the standard case.

Now let us make some comments on related researches from our point of view. First of all, Definite Clause Grammar (DCG) [Pereira and Warren 80] is an instance of unification grammar formalism, which is embedded in Prolog. However, linguistic information is required to be represented in a usual first order term (FOT) in DCG. Because FOT is a *degenerated form*(Shieber) of record representation, our research may be said to give a more linguistically motivated semantics for DCG formalism.

The PTT domain is designed to give a computational model for situation semantics [Barwise and Perry 83] or situation theory [Barwise 85] in logic programming. Situations are central objects in situation semantics. A situation is represented by a set of *state of affairs*. A state of affairs is a triple of the form of $\langle\langle R, a, p\rangle\rangle$, where R, a, and p are *relation*, *assignment*, and *polarity*. Each relation, say, R, is associated with a set of argument places, $arg(R)$. The order of argument places is meaningless. An assignment is a partial function which assigns objects to argument places in $arg(R)$. Consequently, an assignment is the new type of object to be represented in computer languages for semantic analysis of natural language. Then, the PTT domain was designed to represent both linguistic information and the assignment, which is basic in representing situations.

The built-in unification between PSTs, described later in this paper, is the same as the standard merging operation not only between complex linguistic features but also between those assignments in Barwise [85].

We note that Pollard [85] proposed *anadic relations*, which do not use the notion of arity, according to the partial assignment proposed in [Barwise 85]. Our PTT domain can be used as a theoretical model for arguments of anadic relations.

Here is another general remark about relationship between the situation theory and our research. Barwise [87] showed a model of the situation theory in the universe of non-well-founded sets of Aczel [88]. It is known that the class of infinite trees is a model for non-well-founded sets. However, our notion of tree in this paper is restricted to that which is at most a limit of a countable sequence of trees of finite depth, while tagged graphs in the decoration axiom of ZFC/AFA of Aczel are arbitrary. From this restriction, however, we can obtain a basic proposition called a *solution lemma* within the standard ZFC set theory.

In his thesis, Aït-Kaci [84] extended the first order term to record-like structure. Aït-Kaci interprets the record-like structure as a type description with semi-lattice theoretic ordering. Aït-Kaci also proposed to extend the record-like structure to have disjunction and negation embedded in the structure. Kasper and Round [86] proposed Round-Kasper Logic over feature structure with disjunction using the automata theory. Recently, Smolka [88] gave a complete theory of feature logic coping with disjunction and negation.

Our research, however, seems to differ from other research in that we formalize our theory by introducing a category of *record algebras* and show that the PTT domain is *canonical* in the category based on the intuition that the PTT domain is a natural extension of the Herbrand universe. This paper focuses on description of a declarative semantics of simplified grammars based on logic programming. It shows that the proposed language has soundness and completeness properties with respect to the *PTT* interpretation just as the usual Horn clause logic does. By contrast, the completeness part of Aït-Kaci [84] is still open. But, this is not surprising because Aït-Kaci [84] works on the general semi-lattice theoretic framework, while our research uses the category of what we call record algebras. However, we do not discuss disjunctive or negative information structure in our representation. This is an interesting topic, but outside the scope of this paper.

J. Goguen suggested that technical results in this paper may be achieved by the theory of Order Sorted Algebra [Goguen and Meseguer 85] within the equality theory of algebra as associative, commutative, and idempotent with unit. However, it should be noted that our semantics of the language takes the largest model, not the least model or initial model as in the standard semantics of logic programming. That is, our domain includes infinite trees. A. Colmerauer[82] introduced infinite trees into Prolog. However he did not give enough logical foundation. B. Courcelle[83] gives a fundamental theory of infinite trees, and M. Maher[87] gives complete axiomatization of infinite trees. However, these work studies trees of with fixed arities.

The organization of this technical discourse follows Lloyd [84]. The method in this paper differs from the standard theory in that the *lifting lemma* and *unification lemma*[Lloyd 84] for the *PTT* semantics cannot be used, because partiality is a new feature in the semantics of the language. Our theory starts with the definition of partially tagged trees (*PTT*s). A *PTT* is a tree which is allowed to have tags only at the tip nodes. However, as will be shown in the final section, it is easy to extend the notion so that the tree can have tags at intermediate nodes. A tree here not only may be infinite but may also have an infinite number of branches at a node in the tree. Then, a record-like data structure with variables will be introduced, which is called a *partially specified term (PST)*, as a syntactical objects to denote a tree in the domain. A *PST* is designed to be always finite for computational reasons while *PTT* is possibly infinite.

The technical heart of this paper defines the *unification* over *PST*s and *solution* to an equation so that the unification *preserves* the set of solutions to the equations. The unification algorithm is proven to have the termination property. Assignment of variables to *PTT*s and notion of interpretation will be defined as usual. Since a partial data structure is used, it is not easy to define the notion of a solution to an equation between two *PST*s. One might think that it is enough to define that an assignment is a solution if and only if there is a compatible interpretation of both sides of the equations in the sense of natural order of *PTT*s. Unfortunately, there is a simple counter example which shows that this simple definition does not work well. This is the reason for using a special kind of satisfaction based on which a *solution to an equation* is defined based on. It is proven that the unification preserves the set of solutions. This will be one of the key lemmas to establish the soundness and completeness of the language.

We will present two systems of equality axioms. One is for defining the class of record algebras including the PTT domain as the canonical model of the theory. The other is for the constraint of partial equality. A system of partial equalities which is saturated by the inferences of partial equality is fundamentally important in this formalism. In particular, we will prove a powerful lemma called the "solution lemma" which guarantees the existence of a solution to any of these systems. The solution lemma is the core of this work.

A program in the language is a set of Horn clauses. That is, a Horn clause is defined as a pair (p, B) of a PST, p, and a set, B, of PSTs. A model of a given program is defined as the *maximum set* of PTTs which is closed under program clauses roughly read that if there is an instance of the head of the clause then there are instances of each element of the body.

A *configuration* of SLD derivation is represented by a pair of a goal and an environment, which is the system of partial equations touched on above. A computation will be defined based on the transition relation between configurations. The *soundness* of this derivation rule is proven in a similar way to the standard one. This soundness will be that if there is a refutation then each solution to the final environment makes an instance in the model of the program. As the final result, it will be proven that a *correct answer environment* will always be *displayed* as a final environment of some refutation for the goal. This is a strong form of the *completeness* result of the language. The correct answer environment is a counter part of the correct answer substitution in Lloyd[84].

The final section of this paper will extend the Herbrand universe with the PTTs, so that even nested structures of first order terms and PSTs can be used. The paper concludes by demonstrating unification based-grammar formalism in the extended language.

PST and its PTT semantics has already been implemented in several versions, named CIL, on Edinburgh Prolog and ESP [Chikayama 84], and has been used for natural language processing [Mukai 88].

2 Partially Tagged Trees

We fix the set, $\mathcal{L}$, of *labels*. A notation $\langle a_1, ..., a_n\rangle$ stands for the *path* consisting of labels $a_1, ..., a_n$, whose *length* is n, where n is a non-negative integer. In particular, when $n = 0$, $\langle\rangle$ stands for the *empty* path. The length of the empty path is zero.

For two paths, α and β, $\alpha * \beta$ denotes the *concatenation* of α and β. The operator "$*$" may be omitted as usual. The concatenation is defined by the following equations:

$$\langle\rangle\alpha = \alpha,$$

$$\alpha\langle\rangle = \alpha,$$

$$\langle a_1, ..., a_n\rangle\langle b_1, ..., b_m\rangle = \langle a_1, ..., a_n, b_1, ..., b_m\rangle.$$

Also the set of paths can be defined to be the *free semi-group generated over* $\mathcal{L}$. We will often identify a label a with the path $\langle a\rangle$ of length 1 when the context is clear.

A *tree* is a non-empty set, T, of paths which is closed under prefix. That is, if $\alpha\beta \in T$ then $\alpha \in T$. By definition, every tree has the empty path $\langle\rangle$. The special singleton set of the empty path is a *unit tree*. For two trees, T_1 and T_2, the union

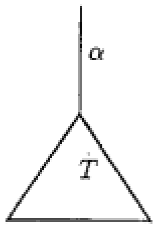

Figure 1: αT, *where α is a path T is a tree.*

$T_1 \cup T_2$ is a tree and the intersection $T_1 \cap T_2$ is also a tree. Moreover both the intersection and the union of any family of trees are also a tree.

For a path α and a tree T, αT denotes the minimum tree which includes the path $\alpha\beta$ for any path β in T:

$$\alpha T = \{\beta | \exists\beta'(\exists\alpha' \in T\ \beta\beta' = \alpha\alpha')\}.$$

For a set T of paths and a path α, the expression $T/\!\!/\alpha$ denotes the largest set T' of paths such that $\alpha T'$ is a subset of T. By definition of tree, the set $T/\!\!/\alpha$ is a tree if T is a tree and α is a path in T. In our definition, a path α in a tree T may have an *infinite number of branches*. That is, there may be infinitely many labels a such that $(T/\!\!/\alpha)/\!\!/a$ is defined, i.e., non-empty. A path α in a tree T is a *leaf* iff α is not a prefix of any other path in T. Namely, if $\alpha\beta$ is in T then $\beta = \langle\rangle$. We define $leaf(T)$ to be the set of leaves of T.

Example 1 *If $T = \{\langle\rangle, \langle b\rangle, \langle c\rangle, \langle c, d\rangle\}$, then $aT = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle, \langle a, c\rangle, \langle a, c, d\rangle\}$.*

It is easy to show that for any non-unit tree, T, there is a possibly infinite number of distinct labels $a_1, ..., a_n, ...$ and trees $T_1, ..., T_n, ...$ such that

$$T = a_1T_1 \cup \cdots \cup a_nT_n \cup \cdots.$$

The labels and trees are determined uniquely by T.

Let $\mathcal{A}$ be a set of *tags*. Also a tag is called a *constant* or *a value* depending on the context. A *tagging of a tree T* is a partial function f from $leave(T)$ into $\mathcal{A}$. The domain $dom(f)$ is possibly empty even when $leave(T)$ is not so.

Definition 1 *A partially tagged tree (PTT) is an ordered pair (T, f) of a tree T and a tagging f of T.*

The PTT whose first component is the unit tree and whose second component is the empty fuction is called the *unit PTT*. For a constant c, we write $\bar{c}$ for the PTT (e, f) where e is the unit tree and e is tagged with c, that is, $ran(f) = \{c\}$. We often identify c with $\bar{c}$ if the context is clear.

For a *PTT* $t = (T, f)$ and a path α, we define $t/\!\!/\alpha = (T/\!\!/\alpha, g)$, where g is a tagging defined by the equation $g(\beta) = f(\alpha\beta)$. Let $t_1 = (T_1, f_1)$ and $t_2 = (T_2, f_2)$ be two *PTT*s. The pair $t = (T_1 \cup T_2,\ f_1 \cup f_2)$ is called the *merge of t_1 and t_2* if t is a *PTT* and is written $t_1 + t_2$. It is easy to check that the set of *PTT*s forms a *commutative*, *associative*, and *idempotent (partial) semi-group* with the unit *PTT* as the unit element with respect to the merge operation. Moreover, a tree is operated by paths from both sides as written αt and $t/\!\!/\alpha$ respectively.

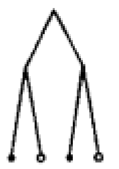

Figure 2: A PTT: *A partially tagged tree: a white leaf means that the node is given a tagg and a black leaf means that the node is given no tagg.*

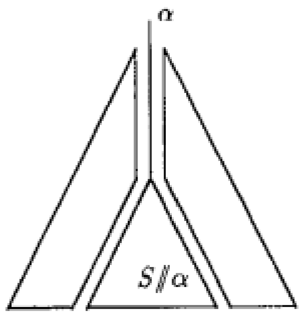

Figure 3: *$S/\!\!/\alpha$ denotes a subtree of S which is accessed through the path α.*

Here is a list of properties of PTT:

$\epsilon + t = t.$	(UNIT)
$t + \epsilon = t.$	(UNIT)
$(t_1 + t_2) + t_3 = t_1 + (t_2 + t_3)$	(ASSOCIATIVE)
$t_1 + t_2 = t_2 + t_1$	(COMMUTATIVE)
$t + t = t$	(IDEMPOTENT)
$()t = t.$	(UNIT)
$(\alpha\beta)t = \alpha(\beta t)$	(ASSOCIATIVE)
$\alpha(t_1 + t_2) = \alpha t_1 + \alpha t_2$	(DISTRIBUTIVE)
$\alpha(t_1 + \cdots + t_\lambda + \cdots = \alpha t_1 + \cdots + \alpha t_\lambda + \cdots$	(DISTRIBUTIVE)
$t/\!\!/ <>= t$	(SUBTREE)
$(t/\!\!/\alpha)/\!\!/\beta = t/\!\!/(\alpha\beta)$	(SUBTREE)
$(t_1 + t_2)/\!\!/\alpha = t_1/\!\!/\alpha + t_2/\!\!/\alpha$	(DISTRIBUTIVE)
$(t_1 + \cdots + t_\lambda + \cdots)/\!\!/\alpha = t_1/\!\!/\alpha + \cdots + t_\lambda/\!\!/\alpha + \cdots$	(DISTRIBUTIVE)
$(\alpha t)/\!\!/\alpha = t$	(CANCELLATION)

Two paths are said to *be incomparable* if each of them is not prefix of the other. It is obvious that if α and β are incomparable then $\alpha u + \beta v$ is a PTT for any PTTs u and v.

A *record algebra* is a partial algebra $(R, G, +, \cdot, /\!\!/, \epsilon)$ which satisfies these properties where R is a set of *records*, G is a semi-group which operates on records from both left and right sides, ϵ is an element of R, and the three fuctions $+ : R \times R \to R$, $\cdot : G \times R \to R$, $/\!\!/ : R \times G \to R$ are partial. A homo-morphism between record algebras is defined as usual. The PTT algebra is characterized as the initial algebra of the category of record algebras with the homo-morphisms[Mukai 89].

By repeating SUBTREE axioms, $t = u$ follows from $\alpha t = \alpha u$. A set of *PTTs* is *compatible* iff any pair of t and t' in the set has the merge $t + t'$. By $t_1 \leq t_2$, we mean $T_1 \subset T_2$ and $f_1 \subset f_2$. It is quite easy to see that the set of PTTs forms a complete partial ordered set with respect to the relation $\leq$.

Proposition 1 *A compatible set of PTTs has the least upper bound with respect to the order $\leq$.*

It follows from this proposition that the PTT domain is *chain complete* with respect to $\leq$.

2.1 Partially Specified Terms

Let the symbols $\mathcal{L}$ and $\mathcal{A}$ denote the set of labels and the set of constants as above respectively. We introduced a set $\mathcal{V}$ of *indeterminates* or *variables*. We assume that these three sets are disjoint to each other for simplicity. The following auxiliary symbols are used:

$$\{ \} / , ( ) .$$

Definition 2 *A partially specified term (PST) is defined inductively as follows:*

(1) A constant is a PST;

(2) A variable is a PST;

(3) The set $\{(a_1/p_1), ..., (a_n/p_n)\}$ is a PST for any finite *$n \geq 0$ and labels $a_1, ..., a_n$, provided that all elements $p_1, ..., p_n$ are PSTs.*

Note that we do not assume that labels i are distinct to each others. In general the expression $\alpha c + \alpha c'$ is a PST even if c and c' are distinct two constants. We use a notation $a_1p_1 + \cdots + a_np_n$ for the PST $\{(a_1/p_1), ..., (a_n/p_n)\}$. A PST is be regarded as a *parametric finite* PTT which may have variables as tags [2]. Therefore, αp and $p /\!\!/ \alpha$ are defined just as in the case of a PTT. Thus, a PST is represented as a finite set of the form αx where α is a path, and x is a variable or *constant*. A PST p has a unique *normal form*, *i.e.*, $p = \Sigma \alpha u$, where α is a path and u is either a variable or constant. Two PTTs, u and v, are a *conflicting pair* if

(1) u and v are distinct constants, or

(2) One of u and v is a constant and the other is neither a constant nor a variable.

A PST p *has a conflict* if it is written as $p = \alpha u + \alpha v + q$ where α is a path and u and v are a conflicting pair. A PST is *conflict-free* if it has no conflict.

3 Theory of Partial Equations

3.1 Axioms of Partial Equations $\mathcal{E}$:

In what follows, x, y, and z are variables, α is a path, u, v and w are arbitrary PSTs. An *atomic formula* is of the form $u \bowtie v$.

(1) $x \bowtie x$.
$x \bowtie y \Longrightarrow y \bowtie x$.
$x \bowtie y,\ y \bowtie z \Longrightarrow x \bowtie z$.

[2]The notion of parametricity is rigorously defined in Mukai[89]

(2) $u \bowtie u$.
$u \bowtie v \Longrightarrow v \bowtie u$.
(*There is no general transitive law.*)

(3) $u + v \bowtie w \Longrightarrow u \bowtie w$

(4) $\alpha u \bowtie \alpha v \Longrightarrow u \bowtie v$.

(5) $\alpha u + \alpha v \bowtie w \Longrightarrow u \bowtie v$.

(6) $x \bowtie u,\ x \bowtie v \Longrightarrow u \bowtie v$. (*Restricted transitive law.*)

Note that there is no transitive law as indicated above. The three axioms of (1) state that the restriction of the binary relation $\bowtie$ to the variables is an equivalence relation between them.

Let S be a set of atomic formulas. The *closure* of S is the set of atoms which is derivable from S by these axioms. In other words, the closure is the minimum set of atomic formulas which satisfies these rules. From the form of each rules, it is easy to see that the closure of S exists. It is a simple consequence of the theory that if $\alpha x \bowtie \alpha\beta y$, then $x \bowtie \beta y$.

Definition 3 (PET) *A set E of partial equations is called a PET if E has no conflict and is closed under all inference rules of the PET theory.*

3.2 Solving Partial Equations

Unification over the PTT domain is formalized simply as the *closure* operation defined above. The input is a set of atomic formulas (i.e., of the form $p \bowtie q$). The output is the closure of the input, or $FAILURE$ if the closure contains a conflicting pair.

By observing the equality axioms, every partial equation in the closure of a given set E has at both sides sub-expressions appearing in E. Also the closure is a monotonically increasing operation with regard to set inclusion. Since the cardinality of the set of sub-expressions of those in the input is finite, there is a unification algorithm which terminates. As a fact it is easy to see that there is a straightforward UNION-FIND-like algorithm of unification. Although details are omitted here, in our unification algorithm, a set of equations is represented by using only those equations one side of which is a variable for space efficiency. The following example shows a simplified version of our method.

Example 2 *Let X and Y be variables. Then,*

$$
\begin{aligned}
&(\{aX + bX \bowtie bY + a1\}, \{\{X\}, \{Y\}\}) \\
&\Rightarrow (\{X \bowtie 1, X \bowtie Y\}, \{\{X\}, \{Y\}\}) \\
&\Rightarrow (\{X \bowtie Y\}, \{\{X, 1\}, \{Y\}\}) \\
&\Rightarrow (\{X \bowtie Y, 1 \bowtie Y\}, \{\{X, 1, Y\}\}) \\
&\Rightarrow (\{X \bowtie Y\}, \{\{X, 1, Y\}\}) \\
&\Rightarrow (\phi, \{\{X, Y, 1\}\}).
\end{aligned}
$$

Example 3 *Let X and Y be variables. Then,*

$$
\begin{aligned}
&(\{X \bowtie aY + bY, Y \bowtie aX + bX, X \bowtie Y\}, \{\{X\}, \{Y\}\}) \\
&\Rightarrow (\{X \bowtie aY + bY, Y \bowtie aX + bX, X \bowtie Y\}, \{\{X, aY + bY\}, \{Y\}\}) \\
&\Rightarrow (\{Y \bowtie aX + bX, X \bowtie Y\}, \{\{X, aX + bY\}, \{Y\}\})
\end{aligned}
$$

$$\begin{aligned}
&\Rightarrow (\{Y \bowtie aX + bX, X \bowtie Y\}, \{\{X, aY + bY\}, \{Y, aX + bX\}\}) \\
&\Rightarrow (\{X \bowtie Y\}, \{\{X, aY + bY\}, \{Y, aX + bX\}\}) \\
&\Rightarrow (\{X \bowtie Y, Y \bowtie X\}, \{\{X, Y, aY + bY, aX + bX\}\}) \\
&\Rightarrow (\{Y \bowtie X\}, \{\{X, Y, aY + bY, aX + bX\}\}) \\
&\Rightarrow (\phi, \{\{X, Y, aY + bY, aX + bX\}\})
\end{aligned}$$

The result means a singleton graph with two self loops with labels a and b.

3.3 Satisfiability

An *assignment* is a partial function whose domain is a set of variables. The definition of assignment can be extended to PSTs as usual. We define the *satisfiability relation*, $\models$, between record algebras, assignments, and formulas in the language. In the following, an assignment f assigns an element in the given record algebra to each variable in the relevant expressions.

Let R be a record algebra. An assignment f *satisfies* $\alpha x \bowtie \alpha\beta y$ and, symmetrically, $\alpha\beta y \bowtie \alpha x$ if $f(x)/\!\!/\beta = f(y)$. We write $R, f \models \alpha x \bowtie \alpha\beta y$ or $R, f \models \alpha\beta y \bowtie \alpha x$. It is easy to see that the relation $\models$ is well-defined and exists.

Example 4 $ad1 + bd2 \bowtie aX + bX$. *This constraint is not satisfiable, that is, for any assignment f in R it is not the case that $R, f \models ad1 + bd2 \bowtie aX + bX$.*

Proof. Assume that an assignment f satisfies the given constraint. By the definition of satisfiability, $f(X)/\!\!/d = 1$ and $f(X)/\!\!/d = 2$ must hold. However, since the constants 1 and 2 make a conflicting pair, this is contradiction. **QED**

We add the following rule to the rules for PET:

$$x \bowtie \alpha y, y \bowtie \beta z \Longrightarrow x \bowtie \alpha\beta z$$

We call this the *unfolding rule*. For a given set C of basic formulas, the notion of the closure C is defined in the same way as the original one. We call it the extended closure of C. Note the extended closure is infinite in general. Also note that the extended closure of C has the same set of solutions as the closure of C.

Let T_x denote the set of paths, α, such that $\alpha p \bowtie x$ is in a PET for some PST p, where x is a variable. By definition of tree, T_x is a tree. A tagging, φ_x, over the tree T_x is defined $\varphi_x(\alpha) = c$ if $x \bowtie \alpha c$ for some constant c. Since PET is conflict-free, φ_x is well defined. We call the assignment f defined $f(x) = (T_x, \varphi_x)$ a *formal solution* to the PET.

Lemma 1 *If $\alpha x \bowtie \alpha\beta y$ is in the PET, then $f(y) = f(x)/\!\!/\beta$. That is, $Q, f \models \alpha x \bowtie \alpha\beta y$, where Q is the PTT algebra.*

Proof. First of all, we prove that the set of paths in $f(y)$ is equal to the set of paths in $f(x)/\!\!/\beta$. Let γ be any path in the PTT $f(y)$. By definition of f, we obtain $y \bowtie \gamma z$. By the unfolding rule above, we obtain $x \bowtie \beta\gamma z$. By definition of f, $\beta\gamma$ is in the $f(x)$. From this, it follows that γ is in $f(x)/\!\!/\beta$. For the reverse direction, let γ be any path in $f(x)/\!\!/\beta$. By construction of the formal solution, $x \bowtie \beta\gamma z$ is in the PET for some variable or constant z. Applying the axiom to this and the hypothesis $x \bowtie \beta y$, we obtain $\gamma z \bowtie y$. By definition of f, path γ is in the $f(y)$.

As the second step, we prove that the two taggings are equal. Suppose that $y \bowtie \gamma c$ is in the PET for a path γ and a constant c. By definition of the formal solution we obtain $\varphi_y(\gamma) = c$. From this and the unfolding rule we have $x \bowtie \beta\gamma c$ in the PET. By definition of formal solution again, $\varphi_x(\beta\gamma) = c$.

Let γ be a leaf path of $f(x)/\!\!/\beta$ such that $x \bowtie \beta\gamma c$ is in the PET. Since $x \bowtie \beta y$ is in the PET, we have $y \bowtie \gamma c$. This means that the taggings have the same value c at leaf γ in $f(x)/\!\!/\beta$. **QED**

Proposition 2 *Any PET Eis satisfiable in the PTT algebra Q.*

Proof. E is a set of constraints of the form $\alpha x \bowtie \beta y$. Let f be a formal solution of the E. By the previous lemma, $Q, f \models \alpha x \bowtie \beta y$. This means that $Q, f \models PET$. **QED**

Theorem 3 (Compact) *For any constraint $C = \{p_1 \bowtie q_1, ..., p_n \bowtie q_n, ...\}$, if every finite subset of C is satisfiable then C is satisfiable.*

Proof. Let $F = \bigcup_{d \in pow'(C)} E_d$, where $pow'(X)$ is the set of *finite* subsets of X and E_d is the closure of d. Since d is satisfiable, E_d is a PET. It is easy to see that F is a PET. Therefore by proposition 2, F is satisfiable. **QED**

Theorem 4 *The following conditions (1) and (2) are equivalent:*

(1) p and q are unifiable.

(2) For any record algebra R, there is some assignment f such that $R, f \models p \bowtie q$.

Proof. Let Q be the PTT algebra and R be any record algebra. Since unifiability, the closure is a PET. By theorem 2, a PET has a solution f in Q. Since there is a mapping from Q into R by sending element αc into αc in the sense of R, we can translate f into an assigment f' in R. Since R is a record algebra, it is easy to check that f' satisfies $p \bowtie q$.

To show (2)⇒(1), we note that for a record algebra R and an assignment f, if $R, f \models C$ and C' is directly inferred from C in the PET theory, then $R, f \models C'$. Hence it follows from the definition of satisfaction that the closure of $p \bowtie q$ is a PET. **QED**

Proposition 5 *If f satisfies $p \bowtie x$ and $q \bowtie x$, then f satisfies $p \bowtie q$, where x is a variable and p and q are PSTs.*

Proof. Assume that f satisfies $p \bowtie x$ and $q \bowtie x$. Let α and $\alpha\beta$ be leaf nodes of p and q respectively such that $p = \alpha u + p'$ and $q = \alpha\beta v + q'$. By definition of satisfaction, $f(u) = f(x)/\!\!/\alpha$ and $f(v) = f(x)/\!\!/\alpha\beta$. Applying the properties of record algebra, we obtain $f(v) = f(u)/\!\!/\beta$. Hence we obtain $f \models u \bowtie \beta v$. Therefore, $f \models p \bowtie q$. **QED**

Lemma 2 *If S' is the closure of S, they have the same set of solutions.*

Proof. It suffices to check that restricted transitive rule preserves the set of solutions. As a fact Proposition 5 guarantees this. **QED**

The following theorem is called the *satisfaction completeness theorem* by Jaffar and Lassez [84]. The theorem states that for any constraint C, if C is not satisfiable in some record algebra, then C is not satisfiable in any other record algebra.

Theorem 6 (Satisfaction Complete) *For any basic constraint $p \bowtie q$, if there is a record algebra in which $p \bowtie q$ cannot be satisfied, then $p \bowtie q$ cannot be satisfied in any other record algebra.*

Proof. Suppose that R is as above. Suppose that the closure C of $p \bowtie q$ is a PET, that is, conflict-free. Since every record algebra in our class of interpretation satisfies any PET, R satisfies $p \bowtie q$, which contradicts the assumption about R. Thereby, C must have a conflict. Since no record algebra model can satisfy a constraint which has a conflict, this concludes the theorem. **QED**

4 Program over Record Algebra

A *program clause* is a pair (p, B) of a *PST* p and a finite set B of *PST*s. A *goal* is a finite set of PSTs. A *program* is a finite set of program clauses. A program clause of the form $(p, \{p_1, ..., p_n\})$ $(n \neq 0)$ is written

$$p \leftarrow p_1, ..., p_n.$$

while a program clause of the form (p, ϕ) is called a *unit clause* and written

$$p.$$

We fix a program. A *configuration* is an ordered pair (G, E) of a goal G and a PET E.

Definition 4 *A resolution step is a pair of configurations written $(G, E) \rightarrow (G', E')$, satisfying the following conditions (1) and (2), where G and G' are goals, E and E' are PETs, and $G = \{p_1, ..., p_n\}$ for some PST's $p_1, ..., p_n$.*

(1) $G' = \{q_1^1, ..., q_{k_1}^1, ..., q_1^n, ..., q_{k_n}^n\}$ for n copies $q_i \leftarrow q_1^i, ..., q_{k_i}^i$ of some program clauses $1 \leq i \leq n$.

(2) E' is the closure of $\{p_1 \bowtie q_1, ..., p_n \bowtie q_n\} \cup E$.

The constraint $\{p_1 \bowtie q_1, ..., p_n \bowtie q_n\}$ here is called the *system of partial equations of the resolution step*. The *initial PET* of a goal is the finest PET over the variables appearing in the goal, that is, $\{x \bowtie x \mid x \in V\}$, where V is the set of variables in the goal. *A computation* is the maximal denumerable sequence of configurations such that for any configuration x which has the successor y in the sequence, $x \rightarrow y$ is a resolution step. A *success computation* is either an infinite computation or a finite computation whose last configuration has the empty goal. A *failure computation* is a computation which is not a success computation. Failure computation must be finite by definition.

Definition 5 *Let p, f, and α be a PST, an assignment, and a path. A PTT t is of type p with respect to f if $f(x) = t /\!\!/ \alpha$ for any variable or constant x and α such that $p = \alpha x + u$ for some PST u.*

By $t, f : p$, we mean that t is of type p with respect to f.

Definition 6 *A set, M, of PTTs is a model of a given program if, for any t in M, there is a program clause $p \leftarrow q_1, \cdots, q_n$ and an assignment f which satisfy the following conditions:*

1. $t, f : p$, that is, t is an instance of p w.r.t. f.

2. *There are PTTs t_i in M such that $t_i, f : p_i$ for any $1 \leq i \leq n$.*

If both M and M' are models, then $M \cup M'$ is also a model. That is, the class of models is closed under *union*. We take the largest one as the semantics of the program. A goal G *can be achieved by an assignment f in the given program model M* if for any p in G there is some t in M which is of type p w.r.t f.

Lemma 3 *If G is achieved by a solution f to a PET E, then there is a resolution step $(G, E) \to (G', E')$ for some G' and E' such that G' is achieved by some extension of f which satisfies E'.*

Proof. Let p be any PST in G. Since p has an instance in the program model by f, by definition of the program model, there is a copy, π, of a program clause whose body is achieved by some extension of f. Let C be the set of all partial equality $p \bowtie q$ such that q is the head of π above. Take G' as the union of all the bodies of such copies π. Take E' as the closure of $C \cup E$. By proposition 4, if two PSTs have a common instance then they are unifiable. Hence, E' is a PET. Thus, the resolution step $(G, E) \to (G', E')$ satisfies the lemma. **QED**

By the *PET of a computation*, we mean the union of all the PET components of configurations in the computation. Note that the union of a monotonically increasing sequence of PETs is a PET. In particular, the PET of a finite computation is the PET of only the last configuration in the computation.

Now we are in a position to prove the following four goal theorems.

Theorem 7 (Soundess) *For any solution f to the PET of any success computation from a given goal and for any p in the goal, there is a PTT in the program model which is of type p with respect to f.*

Proof. The idea is that since the computation is successful, the PET of the computation is satisfiable. We prove that this solution gives an element in the model as an instance of the element in the goal.

Let G be the goal. Consider *any* success computation. Let E be the PET of the computation. By the compact theorem, E has a solution. Now let f be such a solution to E. Assume that p is in the given goal and the constraint $p \bowtie q$ is in the system of partial equations at the first resolution step in the computation. By definition of the model and the system of partial equations at a resolution step, $f(q)$ is in the model. Since p and q are unifiable, $f(p) + f(q)$ is also in the model. This means $f(p) + f(q), f : p$, which concludes the theorem. **QED**

Theorem 8 (Completeness) *If a goal G is achieved by an assignment f in the program model, then there is a computation from G whose PET has a solution which is an extension of f.*

Proof. The theorem is proven by repeating Lemma 3. **QED**

We write $sol(E)$ for the set of solutions to a PET E.

Lemma 4 *Let $(G, E) \to (G', E')$ be a resolution step. Let F be a PET such that $sol(E) \subset sol(F)$. Then we can construct a resolution step $(G, F) \to (G', F')$ by using the same copies of the program clauses in the same way as in step $(G, E) \to (G', E')$; that is, the two systems of partial equations of the two resolution steps are equal. Furthermore, we can do so that $sol(E') \subset sol(F')$ holds.*

Proof. Let C be the system of partial equations of the resolution step $(G, E) \to (G', E')$. From the hypothesis, it follows that $C \cup E$ has the closure PET. Let F' be the closure of $C \cup F$. By Theorem 4, F' is a PET. Since $sol(E) \subset sol(F)$, it follows from the construction of the closure that $sol(F') \supset sol(F)$. **QED**

Let Γ be a success computation starting from a configuration, say, (G, E). Also, let Γ' be a success computation starting from the initial PET for G which is constructed by repeating Lemma 4. Let both (H, D) and (H, D') be the n-th configurations in Γ and Γ' respectively. Then, by induction on the order of the position in the two computations, it is proven that D is the closure of $E \cup D'$, where n is any integer with $1 \leq n \leq$ the length of $\Gamma(\Gamma')$. Γ' is called the *computation associated with* Γ. Also, each configuration in Γ' is called the *configuration associated with* the corresponding one in Γ.

A set V of variables is *free in a PET E*, if for any $x \in V$ and y for which $x \bowtie y \in E$, y is a variable not in V.

Theorem 9 *Let (G, E) be a configuration such that any variable in E but not G is free in E. Suppose that for any solution f to E, every element in G has an instance in the program model w.r.t. f. Then there is a computation, Γ, from the initial configuration for G such that for any solution f of E can be extended to a solution of the PET of* Γ.

Proof. Let V be the set of variables appearing in E but not in G. We can assume that for each variable in V there is a new constant. Then, substitute the new constant for each variable in E, obtaining a PET E'. By repeating lemma 3, there is a computation, Γ', from (G, E'). Hence, by simulating this computation, we can construct a computation, Γ'', from (G, E) such that V is free w.r.t. the PET of Γ''. Let H be the PET of Γ''. Finally we get the Γ of the theorem as the associated computation to Γ''. Let K be the PET of Γ. By the remark above, H is the closure of $E \cup K$. Hence, since V is free in H, V is free also in K. That is, the set of solutions of E can be extended to a solution of K.

QED

Theorem 10 (Soundness of NAF) *If the computation fails then there is no PTT in the model which is of the type of the goal.*

Proof. This is the contraposition of theorem 7. **QED**

Theorem 11 (Completeness of NAF) *If there is not a PTT in the model which is of the type of the given goal then any computation fails.*

Proof. This theorem is the contraposition of theorem 8. **QED**

Thus, the soundness and completeness of the negation as failure rule are trivially proven in our domain and computation. The secret is that an infinite computation has meaning in our case, while it has no meaning in the Herbrand semantics. In other words, our semantics takes the maximum model while the standard semantics takes the minimum one. Maximum computation may be fitted to lazy evaluation computation and used for constraint processing.

5 Extended Definite Clause Grammar

As an application of the theory of *PST* and *PTT*, we give an extension to the DCG (Definite Clause Grammar) formalism. First of all, we give an extension of the Herbrand universe with *PTT*s. We extend the first order terms with *PST*s. In other words, we will use both first order terms and PSTs freely in the extended DCG. Thus, we have more natural and flexible representation for linguistic information, which is comparable with unification-based grammar formalisms, for instance, PATR-II.

Let $\mathcal{F}$ be a set of functors. We assume $\mathcal{F} \supset \mathcal{A}$.

5.1 Extended PTTs

We define an extended PTT as an ordered pair (T, f), such that $dom(f)$ is a subset of T and such that if $f(x) = h$ then $\{1, ..., n\} = \{a | xa \in T\}$, where $n \geq 0$ is the arity of h, where h is any functor. We can replace PST and PTT by EPST(Extended PST) and the extended PTT, respectively, in the proposition obtained so far in this paper.

5.2 Extended Term

Assuming each functor is assigned an arity, a term is defined inductively as follows:

(1) A PST is a EPST;

(2) The set $\{a_1/p_1, \cdots, a_n/p_n\}$ is a EPST; where each a_i is a label and x_i is a EPST ;

(3) The form $h(p_1, \cdots, p_n)$ is a EPST where p_i is a EPST and h is a functor of arity $n > 0$.

For convenience, we use freely standard infix notations used in Prolog.

We add the following rule to the system of axioms for partial unification:

If $h(u_1, ..., u_n) \bowtie h(v_1, ..., v_n)$, then $u_1 \bowtie v_1$ and ... and $u_n \bowtie v_n$.

5.3 Extended DCG

We define the extended DCG by using the following examples. Rule (1) below is a grammar rule, which is a pair of a *PST* and a list of PSTs and equality constraints. Unit clauses (2) and (3) are for lexical items. The program semantics of the extended DCG is an extension of the standard Herbrand model. $=$ is interpreted as the partial equality, $\bowtie$.

```
(1) {cat/s, head/H}:-
        {cat/np, head/H1},
        {cat/vp, head/H},
        H={subject/H1}.

(2) lex(jack, {cat/np, head/jack}).
(3) lex(runs, {cat/vp, head/{subject/X,
                                  pred/run(X)}}).
```

Clauses (4) to (8) form a definition of the interpreter for the extended grammar. Unit clause (4) is to interpret the equality constraints in the grammar rules.

```
(4) X=X.
(5) parse(X-X, A=B):- A=B.
(6) parse([X|Y]-Y, F):- lex(X, F).
(7) parse(X-Y, (A, B)):-
        parse(X-Z, A),
        parse(Z-Y, B).
(8) parse(X-Y, F):-
        (F:-B),
        parse(X-Y, B).
```

Execution of the grammar looks like this:

```
?-parse([jack, runs]-[], F).

F={cat/s,{head/{subject/jack, pred/run(jack)}}}.
```

Acknowledgements

This research has been influenced by suggestions and comments from many people since publication of earlier version of this paper. Among them, in particular, I would like to thank J.A. Goguen, J-L. Lassez, and K. Furukawa.

References

[Aczel 88] P. Aczel: Non-well-founded sets, CSLI lecture note series, 1988.

[Aït-Kaci 84] H. Aït-Kaci: *A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures,* a dissertation in computer and information science, University of Pennsylvania, 1984.

[Barwise and Perry 83] J. Barwise and J. Perry : *Situations and Attitudes,* MIT Press, 1983.

[Barwise 85] J. Barwise: *The Situation in Logic-III: Situations, Sets and the Axiom of Foundation,* Center for the Study of Language and and Information, CSLI-85-26, 1985.

[Chikayama 84] T. Chikayama: *ESP Reference Manual,* ICOT technical report TR-044, 1984.

[Colmerauer 82] A. Colmerauer: *Prolog II: Reference Manual and Theoretical Model,* Internal Report, Groupe Intelligence Artificielle, Universite d'Aix-Marseille II, 1982.

[Goguen and Meseguer 85] J.A. Goguen and J. Meseguer: *Order-Sorted Algebra I: Partial and Overloaded Operators, Errors and Inheritance,* SRI International and CSLI, 1985.

[Courcelle 83] B. Courcelle: *Fundamental Properties of Infinitetrees,* Theoretical Computer Science 25(1983)95-169, 1983.

[Jaffar and Lassez 84] J. Jaffar and J-L. Lassez: *Constraint Logic Programming,* IBM Thomas J. Watson Research Center, 1986.

[Jaffar, Lassez, and Maher 83] J. Jaffar, J-L. Lassez, and J. Maher: *A Theory of Complete Logic Programs with Equality,* Journal of Logic Programming, Vol. 1, No.3, 1984.

[**Kasper and Rounds 86**] R.T. Kasper and W.C. Rounds: *A Logical Semantics for Features,* Association of Computational Linguistics 1986.

[**Lloyd 84**] J.W. Lloyd: *Foundations of Logic Programming,* Springer- Verlag, 1984.

[**Maher 88**] M. Maher: Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees, draft, 1988.

[**Mukai 85**] K. Mukai: *Horn Clause Logic with Parameterized Types for Situation Semantics Programming*, ICOT technical report TR-101, 1985.

[**Mukai 87**] K. Mukai: Anadic Tuples in Prolog, ICOT technical report TR-239, 1987.

[**Mukai 88**] K. Mukai: Partially Specified Term in Logic Programming for Linguistic Analysis, FGCS'88, Tokyo, 1988.

[**Mukai 89**] K. Mukai: *Merge Structure with Semi-Group Operation and Its Unification Theory* in Japanese, (English version will appear soon), ICOT-TR480, 1989.

[**Mukai and Yasukawa 85**] K. Mukai, and H. Yasukawa: *Complex Indeterminates in Prolog and Its Application to Discourse Models,* New Generation Computing, 3(1985) Ohmusha, Ltd. and Springer-Verlag, v1985.

[**Pereira and Shieber 84**] F.C.N. Pereira and S.M. Shieber: *The Semantics of Grammar Formalisms Seen as Computer Languages,* in the Proceedings of the Tenth International Conference on Computational Linguistics, Stanford University, California, 1984.

[**Pereira and Warren 80**] F.C.N. Pereira and D.H.D. Warren: *Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks,* Artificial Intelligence 13:231-278, 1980.

[**Pollard 85**] Carl J. Pollard: *Toward Anadic Situation Semantics* , (manuscript), Stanford University 1985.

[**Shieber et al 86**] S.M. Shieber, F.C.N. Pereira, L. Karttunen, and M. Kay: *A Compilation of Papers on Unification-based Grammar Formalisms Parts I and II,* Report No. CSLI-86-48, April, 1986.

[**Smolka 88**] G. Smolka: A Feature Logic with Subsorts, LILOG–Report 33, IBM Deutschland GmbH, May 1988.