TR-239

Anadic Tuples in Prolog

by

K. Mukai

March, 1987

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

Anadic Tuples in Prolog*

Kuniaki Mukai

Institute for New Generation Computer Technology

ABSTRACT

A record-like data structure is introduced into Prolog, which is fundamental to represent and analyse linguistic information. The domain of the new language is an extension of Herbrand universe with *partially tagged trees*. The usual first order unification is extended to a record-like structure, which is called a *partially specified term*. Both declarative and procedural semantics are given based on the framework of standard logic programming. It is proved that the language is sound and complete for the two semantics. However, this is done without using lifting lemma and unification lemma because of partiality of the semantics of the language. Some examples are given to demonstrate design motivations from recent linguistic theories, such as unification-based grammars and situation semantics.

1. Introduction

The main objective of this paper is to describe a model of a logic programming language over *partially tagged trees* (*PTT*s). A *PTT* is a model of record-like data structure. The syntactic form of record-like structure described in this paper is called a *partially specified term* (*PST*).

This research was motivated by the question of how to represent and to process linguistic information, including syntax, semantics, and pragmatics, more uniformly than existing Prolog. According to the unification-based grammar formalism, linguistic information is represented in complex features and phrase structure rules giving constraint in the features. In other words, it is a consensus among computational linguists that linguistic information should be encoded in a record-like data structure called a feature set, and that linguistic theories can be described as constraints in the feature sets.

The approach of this research was to combine unfication-based grammar formalism [Shieber et al 86] and situation semantics [Barwise and Perry 83] with logic programming, in particular, Prolog. Through the approach, it turned out that the extension of the first order term to a record-like structure gives a more natural and unified view for natural language processing, and that a record-like data structure is more suitable for representing semantical objects than the usual first order term.

It is well known that Prolog has Definite Clause Grammar (DCG) [Pereira and Warren 80] formalism embedded and that DCG is an instance of unification-based grammar. In DCG, a feature system is represented in

* This is an extended version of what was presented by the author at Workshop on Foundations of Deductive Databases and Logic Programming, August 1986, Washington, DC., organized by J. Minker.

usual terms. Syntactic features (the agreement feature, for instance) are represented implicitly in the physical order of argument places in the term. However, in nature, there is no intrinsic order in the features. First order term in Prolog needs to be extended to a record-like data structure to represent linguistic information more explicitly.

Thus, the feature set in *PST* notation, proposed here,

```
{plural/+, person/ 2nd}
```

is identical to

```
{person/ 2nd, plural/ +}.
```

The example below shows how *PST*s are used to describe linguistic and semantic information in a natural and flexible way. Clause (1) below says that a situation X is a discourse_situation if X has five named parameters, **sit** (situation), **sp** (speaker), **hr** (hearer), dl (discourse location) , and **exp** (expression), such that sit parameter S has three state of affairs (soa) as specified in the body of the clause. Clauses (2) and (3) are the usual ones for membership. Clause (4) defines the pronoun "you," as a function which extracts the hearer from the given discourse situation.

```
(1) discourse_situation({ sit/S, sp/I, hr/You, dl/Here, exp/Exp} ):-
        member(soa(speaking, { ag/I, loc/Here} ), S),
        member(soa(addressing, { recip/You, loc/Here} ), S),
        member(soa(utter, { obj/Exp, loc/Here} ), S).

(2) member(X, [X|Y]).
(3) member(X, [Y|Z]):-member(X, Z).

(4) noun(you, { ip/X, ds/{ sp/X} } ).
```

Introducing the record-like structure into Prolog might seem trivial and only a matter of notational variance. Actually, it is not trivial because it turned out that there were subtle points between the usual first order term and thet record-like structure. At first, the logic programming scheme proposed in Jaffar, Lassez and Maher[83], which is based on the equality theory, was applied to try to give the semantics of the extended Prolog. Soon, it was found to be difficult because of the partiality of the record-like structure *PST*, proposed here. Intuitively, *PST*s can be merged so as to form an indefinitely *wide* structure. In this aspect, the *PST* can be seen as a representation of *anadic relations* in Pollard [85]. Thus, *PST* has no notion of ***arity***. Indeed, in giving semantics to *PST*s, there can can be no counterpart of the lifting lemma and unification lemma, which play basic roles in giving soundness and completeness results in Lloyd[84], for instance. This paper shows that there is a restricted notion of ***solution to an equation*** such that the natural unification process proposed here *preserves* the set of solutions. This is the one of the main technical motivations in this study.

Although it is not the purpose of this paper to introduce situation semantics [Barwise and Perry 83] or situation theory [Barwise 85], a few words pertinent here. Situation semantics was the choice for underlying semantics of natural language. One of the major reasons for that was that the theory views the three components of languages, i.e., syntax, sematics, and pragmatics in a uniform way, treating pragmatics and even syntax as semantics using situations and constraints. Situation theory is a meta-theory for situation semantics. The theory is understood as a kind of set theories for doing semantics, and is expected to play a similar role for natural language semantics to what the set theory has done for mathematics. Semantic objects should be described by situation theory just as mathematical objects should be described in some formal version of the Cantorian set theory, ZFC, for instance, in the end.

Now, the central objects of situation semantics are situations. A situation is represented by a set of state of affairs. A state of affairs is a triple $\langle R, a, p\rangle$ where R, a, p are *relation*, *assignment*, and *polarity*. Each relation, 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 $arg(R)$. Thus, an assignment is the new type of object to be represented in computer languages for semantic analysis of natural language. The main idea behind the research approach was that the PST serves as representation for both linguistic information and the assignment, which is basic in representing situations. Indeed, 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.

Another point from situation semantics used in the programming language proposed here is *complex indeterminates* (CI)[Barwise and Perry 83]. A CI is a kind of description and is represented by an ordered pair of an *indeterminate* and a *condition*. A condition may have more than two *parameters*. CIs play such a basic role in situation semantics that earlier research attempted to introduce CIs into Prolog by extending standard Prolog unification to CIs as follows. Let `[x|c]` and `[y|d]` be two CIs, where `x` and `y` are indeterminates and `c` and `d` are conditions. The unification for these is reduced to unification `x` with `y`, and to solving conditions `c` and `d`.

```
unify([x|c], [y|d]) ← unify(x,y), solve(c), solve(d).
```

However, there was a problem in identifying parameters of the two conditions given. The solution to this problem was very simple [Mukai 85, Mukai and Yasukawa 85]. The user of the language must give identifiers to parameters so that the unifier can identify which pairs of parameters of the conditions should be unified.

From these observations on feature set, state-of-affair, and CI, it was concluded that a record-like structure is a necessary extension to standard Prolog for a unified representation of linguistic and semantic information. With these motivations in mind, a semantics of PSTs based on the domain of PTTs is given in this research.

A record-like structure alone is not sufficient to treat intensional aspects of description. Porto[86] for instance, is relevant to this problem. This is another problem in further research and is out of the scope of this paper.

As in Ait-Kaci[84], the first order term is extended to record-like structure. However, this research differs from Ait-Kaci in the semantic domains. The PTT domain is taken as a natural extension of Herbrand universe as will be shown in the final chapter. On the ohter hand, Ait-Kaci interprets the record-like structure as a type description with semi-lattice theoretic order. Furthermore, Ait-Kaci tries to extend the record-like structure to have disjunction and negation embedded in the structure. Also Kasper and Round [86] discusses these topics using the automata theory. This paper does not discuss such built-in representation for two reasons. One is because the research was intended to concentrate on *PTT* semantics as a natural extension to Herbrand universe. The other is that disjunctive information is represented using sets of Horn clauses and that lazy evaluation control should be able to treat such disjuntive information in a practical sense. Based on a practical intuition, the researchers take a view that negative information can be treated by lazy evaluation, and are interested in the idea of constraint logic programming by Jaffar and Lassez[86] to cope with this problem. However, this is out of the scope of the paper.

This paper shows that the proposed language has soundness and completeness properties with respect to the *PTT* interpretation just as the usual Horn clause logic does. However, the completeness part of Ait-Kaci[84] is still open. This is not surprising because Ait-Kaci works in general semi-lattice theoretic framework and this research, on the other hand, uses a fixed domain, which is a natural extension of Herbrand universe.

J. Goguen suggested to the researchers 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 algebera as associative, commutative, idempotent with unit. The proof is expected to appear later.

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 patiality is a new feature in the semantics of the language.

This paper starts with the definition of partially tagged trees (*PTT*). All *PTT*s form the semantic domain. A *PTT* is a tree which is allowed to have tags only at the tip nodes. However, as will be show at the final chatpter, it is easy to extend the tree so that it can have tags at intermediate nodes. A tree here may be infinite. Also, a tree can have an infinite number of branches at a node in the tree. Then, the the record-like data structure with variables will be introduced, which is called *partially specified term (PST)*, as a syntactic device to denote incompletely a tree in the domain. A *PST* is always finite. There are no predicate symbols, even equality symbol =, in the language, and no function symbols. This restriction is not theoretical but rather expository. There is no problem, as mentioned just before, in introducing function symbols and predicate symbols by allowing the function symbols at intermediate nodes in *PTT*s. Indeed, such an extension will be defined in the final chapter. A *PST* in a program clause behaves like a term some of whose argument places are missing because they have nothing to do with what the local clause wants to do. More precisely, supposing that only l, m and n are labels in the language, any *PST* can be translated into a first order term with a fixed functor f of arity 3. Let l,m, and n correspond to 1st, 2nd, 3rd argument places. Then a *PST* $\{l/x, m/y\}$ is translated into the first order term $f(x, y, z)$ for some new variable z. However, the researchers do not contend with this interpretation, because, as argued above, linguistic information has nothing to do with the notion arity and has no predefined order, and because linguistic information is partial. Rather, the logic programing should give semantics of partially specified terms not through a first order term but in a more intrinsic way.

The technical heart of the paper defines the *unification* over *PST*s and *solution* to an equation. They will be defined so that the unification *preserves* the set of solutions to the equation. The unification algorithm is proved to have 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 solution to an equation between two *PST*s. One might think that it will be 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 demonstrates that this simple definition does not work well. This is the reason for using a special kind of assignments called a *matching assignment* for a given *PST* and *PTT*. The *solution to an equation* is defined based on the matching assignment. It is proved 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.

As a computational device, *environments* will be introduced to be sets of *PST*s. Environments are basic in this formalism. They are an extension of the ordinary formalism, which is a system of normal form equations. This formalism shares some ideas with CLP [Jaffar and Lassez 86] in that an environment is a constraint on solutions over the domain of *PTT*s, which is an ordered set induced by the subset relation between trees. As a basic property of environments, the lemma will be proved that every congruent and conflict-free environment has a solution. This is a counterpart of the existence proof of a solution to a normal system of equations over infinite trees [Colmerauer 82].

A program in the language is a set of Horn clauses. That is, a Horn clause is defined as a pair (p, Q) of a *PST* p and a set Q of *PST*s. A model of a given program is defined as a set of *PTT*s which is closed under implication. Replacing ground substitution for assignment, the proof is similar to the standard proof. Also, a *correct answer environment* is a translation of a *correct answer substitution* of Lloyd[84], which works to define the declarative semantics of Horn clause logic programming. *A configuration* of SLD derivation is represented by a pair of goal and an environment. A computation will be defined based on the transition relation between configurations. The *soundness* of this derivation rule is proved 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 *maps* the goal into the model of the program. As a 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 the form of the *completeness* result of the language.

The final chapter of this research will extend the Herbrand universe with the *PTT*s, so that even nested structure of first order terms and *PST*s can be used. The paper concludes by demonstrating unification based-grammar formalism in the extended language.

PST and its *PTT* semantics has already implemented in several versions, named CIL, on top of Edinburgh Prolog and ESP [Chikayama 84], and has been used for natural language processing, among ohter applications.

2. Partially Tagged Trees

We fix a set *LABEL* throughout the paper. An element in *LABEL* is called a *label*. $\langle a_1, ..., a_n \rangle$ stands for the string of labels $a_1, ..., a_n$. In particular, $\langle\rangle$ stands for the empty string. The length of the empty string is zero.

Definition. We define *concatenation*, $*$, between strings by the following equations:

$$\langle\rangle * x = x,$$

$$x * \langle\rangle = x,$$

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

Definition. A *tree* is a non-empty set T of finite strings which is closed under prefix. That is, if $x * y \in T$ then $x \in T$. Each element in T is called a *node* of T. Note that every tree has the empty string $\langle\rangle$.

Definition. Let T and x be a tree and a string of labels. We define $T//x$ to be the set of strings y of labels such that $x * y$ is in T.

Clearly, if $T//x$ is not empty then also it is a tree. A node in a tree T may have infinite branches. That is, there may be infinite labels a such that $T//\langle a\rangle$ is defined, i.e., non empty.

Definition. A node x in a tree T is a *leaf node* iff x is not a prefix of any other node in T. Namely, for any y if $x * y$ is in T then $y = \langle\rangle$. We define $leaf(T)$ to be the set of leaf nodes of T.

Definition. Let T_1 and T_2 be trees. T_1 is a *subtree* of T_2 iff $T_1 \subset T_2$.

Proposition 2.1. *Let T_1 and T_2 be trees. The union $T_1 \cup T_2$ is a tree. The intersection $T_1 \cap T_2$ is a tree.*

Proof. Obvious. ∎

Definition. For a string x and a set T of strings, we define a tree $x * T$ by the following equation:

$$x * T = \{z | \exists u(\exists y \in T \ z * u = x * y)\}.$$

It is obvious that $x * T$ is a tree. For a label a, we will use the convention $a * T = \langle a\rangle * T$.

Example.

$$\langle a\rangle * \{\langle\rangle, \langle b\rangle, \langle c\rangle, \langle c, d\rangle\} = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle, \langle a, c\rangle, \langle a, c, d\rangle\}.$$

We call the tree, $\{\langle\rangle\}$, the trivial tree.

Proposition 2.2. *For any non trivial tree T, there are possibly infinite number of labels $a_1, ..., a_n, ...$ and trees $T_1, ..., T_n, ...$ such that*

$$T = a_1 * T_1 \cup \cdots \cup a_n * T_n \cup \cdots.$$

The labels and trees are determined uniquely by T.

Proof. Let $L = \{a | \exists x \ \langle a\rangle * x \in T\}$. It is clear that

$$T = \bigcup\{a * (T//\langle a\rangle) | a \in L\}$$

and

$$a * (T//\langle a\rangle) \cap b * (T//\langle b\rangle) = \{\langle\rangle\} \quad (a \neq b).$$

Also it is clear that for any label x and tree A such that $x * A \subset T$ there is some label $a \in L$ such that $x = a$ and $A \subset T//\langle a\rangle$. By this property, uniqueness is easy. ∎

Definition. Let f and g be functions. f and g are compatible iff the (set theoretical) union of f and g is a function. The union is called the *merge of f and g*.

Example. Assuming $f = \{(a,1),(b,2)\}$ and $g = \{(b,2),(c,3)\}$, the function $\{(a,1),(b,2),(c,3)\}$ is the merge of f and g. On the other hand, the two functions $\{(a,1),(b,2)\}$ and $\{(b,4),(c,3)\}$ can not have their merges.

Definition. An ordered pair (T,f) of a tree T and a function f is a *partially tagged tree* (PTT) iff $dom(f) \subset leaf(T)$. The domain $dom(f)$ may be the empty set ϕ . The PTT $(\{\langle\rangle\},\phi)$ is called *trivial*.

If there is no confusion, we write simply a for the PTT $(\{\langle\rangle\},\{(\langle\rangle,a)\})$. Supposing that f is a function defined on strings of labels, and that x is a string we define $f//x$ to be the function such that $(f//x)(y) = z$ iff $f(x * y) = z$. For a PTT t and a string x, we define $t//x = (T//x, f//x)$, where $t = (T,f)$.

Definition. Let $t_1 = (T_1, f_1)$ and $t_2 = (T_2, f_2)$ be PTTs. 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. The merge of t_1 and t_2 is written $t_1 + t_2$. We define $t_1 \leq t_2$ if $T_1 \subset T_2$ and $f_1 \subset f_2$.

It is easy to check that the set of PTTs is a commutative, associative and idempotent semigroup with the trivial PTT as the identity with respect to the merge operation.

Example. Assume $t_1 = (\{\langle\rangle,\langle a\rangle\},\{(\langle a\rangle,1)\})$ and $t_2 = (\{\langle\rangle,\langle b\rangle\},\{\langle b\rangle,2\})$. Then

$$t_1 + t_2 = (\{\langle\rangle,\langle a\rangle,\langle b\rangle\},\{(\langle a\rangle,1),(\langle b\rangle,2)\}).$$

Also

$$(\{\langle\rangle,\langle a\rangle\},\phi) + (\{\langle\rangle,\langle a\rangle,\langle a,b\rangle\},\{\langle a,b\rangle,1\}) = (\{\langle\rangle,\langle a\rangle,\langle a,b\rangle\},\{(\langle a,b\rangle,1)\}).$$

For any PTT t,

$$t + (\{\langle\rangle\},\phi) = t.$$

Note that both $t_1 + (\{\langle\rangle,\langle a\rangle\},\{(\langle a\rangle,3)\})$ and $(\{\langle\rangle\},\{(\langle\rangle,4)\}) + (\{\langle\rangle,\langle a\rangle\},\phi)$ are undefined.

A set of PTTs is called *consistent* iff any pair of t and t' in the set has the merge $t + t'$.

Proposition 2.3. *A consistent set of PTTs has the least upper bound with respect to the order $\leq$.*

Proof. Obvious. ▮

3. Partially Specified Term

Let $VARIABLE$ and $ATOM$ be disjoint sets. An element in $VARIABLE$ and $ATOM$ is called *a variable* and *a constant*. The set $LABEL$ was introduced in the previous section. We assume that $LABEL$ and $VARIABLE$ have no member in common.

The following auxiliary symbols are used:

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

It should be noted that there are no function symbols and predicate symbols in our language. Indeed, we have not even the equality symbol =, until the final chapter.

Now we introduce syntactical representation to denote *PTT*s.

Definition. We call an element of Q a *partially specified term (PST)*, where Q is the least set satisfying:

(a) Q has all constants and variables,

(b) for any *finite* $n \geq 0$, and distinct labels $a_1, ..., a_n$, if all elements $p_1, ..., p_n$ are in Q, then the set $\{(a_1/p_1), ..., (a_n/p_n)\}$ is in Q.

A *PST* is called *proper* if it satisfies condition (b). In particular, the empty PST ϕ is proper.

Here, we gather utility functions concerning *PST*s for later use.

Definition. Let p and q be *PST*s. Let a be a label.

$keys(p)$ is the set of labels a such that (a/q) is in p for some *PST* q.

$assoc(a, p) = q$ iff $(a/q) \in p$.

$pairs(p, q) = \{assoc(a, p) \doteq assoc(a, q) | a \in keys(p) \cap keys(q)\}$,

$vars(x) =$ the set of variables which occur in x.

Example.

$keys(\{a/x, b/y\}) = \{a, b\}$.

$assoc(b, \{a/x, b/y\}) = y$.

$pairs(\{a/x\}, \{a/1, b/y\}) = \{x \doteq 1\}$.

$pairs(\{a/1\}, \{a/2, b/y\}) = \{1 \doteq 2\}$.

Definition. Let p be a *PST*. For a string $\alpha = \langle a_1, ..., a_n\rangle$, we define $p//\alpha$ inductively:

$$p//\langle\rangle = p,$$

and

$$p//\langle a_1, a_2, ..., a_n\rangle = assoc(a_1, p)//\langle a_2, ..., a_n\rangle.$$

Example.

$$\{a/\{b/\{c/1\}\}\}//\langle a, b, c\rangle = 1.$$

Definition.

$frontier(p, p) = \phi$: if p is a constant,

$frontier(p, q) = \{p \doteq q\}$: if either p or q is a constant,

$frontier(p, q) = \{p \doteq q\}$: if either p or q is a variable,

$frontier(p, q) = \bigcup\{frontier(u, v) | u \doteq v \in pairs(p, q)\}$: otherwise.

Example. $frontier(\{a/2, b/\{c/x\}\}, \{a/3, b/y\}) = \{2 \doteq 3, \{c/x\} \doteq y\}$.

4. Equality and Environment

Definition. An *assignment* is a partial function defined on variables which assigns *PTT*s to them. Let ρ be an assignment. We can extend ρ to the *PST*s naturally by induction as below. Let k be a constant, and let a_i be a label.

(a) $\rho(k) = (\{\langle\rangle\}, \{(\langle\rangle, k)\})$,

(b) $\rho(\phi) = (\{\langle\rangle\}, \phi)$,

(c) $\rho(\{a_1/p_1, ..., a_n/p_n\}) = a_1 * \rho(p_1) + ... + a_n * \rho(p_n)$.

Example. For any assignment ρ , we have,

$$\rho(\{a/1, b/\{c/2\}\}) = (\{\langle\rangle, \langle a\rangle, \langle b\rangle, \langle b, c\rangle\}, \{(\langle a\rangle, 1), (\langle b, c\rangle, 2)\}).$$

If $\rho(x) = (\{\langle\rangle\}, \phi)$ and $\rho(y) = (\{\langle\rangle, \langle c\rangle\}, \{(\langle c\rangle, 3)\})$ then

$$\rho(\{a/x, b/y\}) = (\{\langle\rangle, \langle a\rangle, \langle b\rangle, \langle b, c\rangle\}, \{(\langle b, c\rangle, 3)\}).$$

The notion of solution to an equation is basic in the semantics of *PST*. First we define the matching assignment.

Definition. Let p and t be a *PST* and a *PTT*. An assignment θ is called a *matching assignment from p to t* if θ satisfies the following conditions:

(1) If p is a variable then $\theta(p) = t$,

(2) if p is a constant then $\theta(p) = t$, and

(3) if $p = \{a_1/p_1, ..., a_n/p_n\}$ then there are a finite number of *PTT*s $t_1, ..., t_n$, and s such that $t = a_1 * t_1 + ... + a_n * t_n + s$, and that for each $i (1 \le i \le n)$, θ is a matching assignment from p_i to t_i, where $s//\langle a_i\rangle$ is undefined for each a_i.

We write $match(p, t)$ for the set of matching assigments from p to t.

Example. Let x be a variable. Suppose that assignments θ, ρ , a *PST* p, and a *PTT* t satisfy the following:

$$\begin{aligned} \theta(x) &= c * 1 + d * 2, \\ \rho(x) &= c * 1, \\ p &= \{a/x\}, \\ t &= a * (c * 1 + d * 2) + b * (e * 3 + f * 4). \end{aligned}$$

Then, θ is a matching assignment from p to t, but ρ is *not*.

Definition. An assignment ρ is defined inductively to be a *solution to the equation* $p \doteq q$ by the following conditions:

(a) $\rho \in match(q, \rho(p))$ if p is a variable,

(b) $\rho \in match(p, \rho(q))$ if q is a variable,

(c) $\rho(p) = \rho(q)$ if either p or q is a constant, and

(d) ρ is a solution to all equations in $pairs(p, q)$ if both p and q are proper PSTs.

An assignment ρ is called *a solution to a system of equations* iff ρ is a solution to all equations in the system. We define $sol(P)$ to be the set of solutions to P.

Definition. The *height* of a constant or variable is 1. The *height* of a PST of the form $\{a_1/p_1, ..., a_n/p_n\}$ is $1+$ the maximum of the heights of $p_1, ..., p_n (n \geq 0)$. The *height* of the empty PST is 1.

Example. The height of $\{a/b, c/\{d/e\}\}$ is 3.

Lemma 4.1. *Let t and θ be a PTT and an assignment. Let p and q be PSTs. If θ is a matching assignment from both p and q to t , then θ is a solution of the equation $p \doteq q$. That is,*

$$match(p, t) \cap match(q, t) \subset sol(\{p \doteq q\}).$$

Proof. We prove by induction on the minimum h of the heights of p and q.

1) Let $h = 1$ (Base). Let $\theta \in match(p, t)$ and $\theta \in match(q, t)$. First suppose p is a variable. By the definition of *match*, we have $\theta(p) = t$. Then we have $\theta \in match(q, t) = match(q, \theta(p))$. Therefore θ is a solution to $p \doteq q$. It is similar to the case that q is a variable. Suppose both p and q are constants. By the definition of *match*, we have $\theta(p) = t$ and $\theta(q) = t$. Then we have $\theta \in sol(p \doteq q)$. Since $h = 1$, it is impossible for p and q to be proper PSTs. It remains only the case that one of p and q is a constant and the other is a proper PST. Without loss of generality, we can supposing p is a constant and q is a PST. From the definition of *match*, it is easy to show that $match(p, t) \cap match(q, t)$ is empty. This concludes the base step.

2) Suppose that $h > 1$ and that this lemma holds for any $h' < h$. By the definition of height, we can assume that both p and q are proper PSTs. Suppose that $(a/u) \in p$, $(a/v) \in q$, $t//\langle a \rangle = t'$ for some label a. By the definition of *match*, we have $\theta \in match(u, t') \cap match(v, t')$. Applying the induction hypothesis, θ is a solution to $u \doteq v$. Since $u \doteq v$ is an arbitrary element in $pairs(p, q)$, we conclude that $\theta \in sol(pairs(p, q)) = sol(\{p \doteq q\})$. ∎

Lemma 4.2. *If ρ is a solution to $p \doteq q$, then $\rho \in match(p, \rho(p) + \rho(q))$.*

Proof. We prove by induction on the minimum height h of p and q.

(1) Suppose $h = 1$ (Base). From the definition of height, either p or q must be either a constant, variable, or the empoty PST. From the definition of solution, since ρ is a solution to $p \doteq q$, we have $\rho(p) = \rho(q)$. So, we have $\rho(p) + \rho(q) = \rho(p) = \rho(q)$. It follows directly from the definition of *match* that $\rho \in match(p, \rho(p))$. Since $match(p, \rho(p)) = match(p, \rho(p) + \rho(q))$, we have the conclusion.

(2) Suppose $h > 1$. We assume that the lemma holds for $h' < h$. Both p and q must be proper PSTs. Since ρ is a solution to the equation $p \doteq q$, ρ is also a solution to $pairs(p, q)$ of equations. Let $p' \doteq q'$ be an equation in $pairs(p, q)$. Then there is some label a such that $(a/p') \in p$, $(a/q') \in q$, and $\rho \in sol(\{p' \doteq q'\})$. Clearly,

the minimum height of p' and q' is less than h by definiton. Applying the induction hypothesis, we have $\rho \in match(p', \rho(p') + \rho(q'))$. So, from the definition of *match*, we have $\rho \in match(\{a/p'\}, a * (\rho(p') + \rho(q')))$. Recalling that the label a is arbitrary, from the definition of the solution, we have $\rho \in match(p, \rho(p) + \rho(q))$.

∎

Definition. A set E of sets of PSTs is an *environment* if for any variable x, there is at most one element w in E such that $x \in w$. The unique element w is denoted by $class(x, E)$.

Example. Assume $E = \{\{x\}, \{y, z\}, \{u, \{b/1, c/y, d/x\}\}\}$ is an environment. Then $class(y, E) = \{y, z\}$.

Definition. An environment E is *congruent* iff for any $w \in E$, $p, q \in w$, and $u \doteq v \in frontier(p, q)$ there is some $w' \in E$ such that $u \in w'$ and $v \in w'$.

Example. $E = \{\{x, \{a/y\}, y, \{a/x\}\}\}$ is a congruent environment.

Definition. Let ρ and E be an assignment and an environment. ρ is an *solution to* E iff for every w in E and every p and q in w, ρ is a solution to $p \doteq q$. The set of solutions to E is denoted by $sol(E)$.

Definition. An assignment θ is *compatible with an environment* E iff $\theta(p)$ and $\theta(q)$ can be merged for any $w \in E$, $p, q \in w$.

Definition. Let $\theta_0, ..., \theta_n, ...$ be a monotone sequence of variable assignments, i.e., $\theta_n(x) \leq \theta_{n+1}(x)$ for any variable x. We define the *limit assignment* θ by the pointwise limit of θ_i: $\theta(x) = sup\{\theta_n(x) | n \geq 0\}$. We write $\theta = sup\{\theta_n | n \geq 0\}$.

Proposition 4.3. *For any monotone sequence of assignments $\theta_0, ..., \theta_n, ...$ and a PTT p, the following equation holds:*

$$sup\{\theta_n(p) | n \geq 0\} = (sup\{\theta_n | n \geq 0\})(p).$$

Proof. Let $\theta = sup\{\theta_n | n \geq 0\}$. We prove by structural induction on p.

(1) If p is a constant then the proposition is true since both sides of the equation are p.

(2) If p is a variable then we have $(sup\{\theta_n | n \geq 0\})(p) = sup\{\theta_n(p) | n \geq 0\}$ by the definition of $sup\{\theta_n | n \geq 0\}$.

(3) If $p = \{a_1/p_1, ..., a_r/p_r\}$ then

$$\begin{aligned} sup\{\theta_n(p) | n \geq 0\} &= sup\{\theta_n(\{a_1/p_1, ..., a_r/p_r\}) | n \geq 0\} \\ &= sup\{a_1 * \theta_n(p_1) + ... + a_r * \theta_n(p_r) | n \geq 0\} \\ &= a_1 * sup\{\theta_n(p_1) | n \geq 0\} + ... + a_r * sup\{\theta_n(p_r) | n \geq 0\} \\ &= a_1 * \theta(p_1) + ... + a_r * \theta(p_n) \quad \text{(by the structural induction)} \\ &= \theta(p). \end{aligned}$$

∎

Definition. An equation $p \doteq q$ is a *conflict* iff one of the following three holds:

(1) p and q are distinct constants;

(2) p is a constant and q is a proper PST;

(3) p is a proper PST and q is a constant.

We say that a set P of PSTs is *consistent* when for every p and q in P, there is no conflict in $frontier(p, q)$. Also, we say an environment E is *consistent* when all classes in E are consistent.

Example. The set of PSTs $\{\{a/1, b/1\}, \{a/1, b/2\}\}$ is *not* consistent.

We say an assignment ψ is *normal* for E iff $\psi(x) \geq \psi(p)$ for any variable x and a PST p with $p \in class(x, E)$.

Proposition 4.4. *Every congruent and consistent environment has a solution.*

Proof. Let E be and congruent and consistent environment. Let ψ be an assignment compatible with E and normal for E. We have the assignment defined by

$$\Phi(\psi)(x) = sup\{\psi(y) | y \in class(x, E)\}.$$

Clearly, $\Phi(\psi)$ is normal for E. We prove $\Phi(\psi)$ is compatible with E. Let ψ' an ψ'' be any assignments such that for any variable x there are some p and q in $class(x, E)$ such that

$$\psi'(x) = \psi(p) \leq \psi(x)$$

and

$$\psi''(x) = \psi(q) \leq \psi(x).$$

We show that $\psi'(r)$ and $\psi''(s)$ can be merged for any r and s in the same class in E. Let $g \doteq h \in frontier(r, s)$. Since E is congruent and consistent, g and h are in the same class and either g or h is a variable. Suppose g is a variable. Then $\psi'(g) = \psi(g')$ for some g' in $class(g, E)$ by the assumption. Since g' and h are in the same class, $\psi(g')$ and $\psi(h)$ can be merged. Also from the assumption, $\psi(h) \geq \psi''(h)$. Then $\psi'(g)$ and $\psi''(h)$ can be merged. It is similar to the case in which h is a variable. Namely $\psi'(r)$ and $\psi''(s)$ can be merged. It is easy to see that $\Phi(\psi)(r)$ is the merge of all forms $\psi'(r)$ above. That is,

$$\Phi(\psi)(r) = \psi'(r) + \psi''(r) + \psi'''(r) + \cdots$$

and

$$\Phi(\psi)(s) = \psi'(s) + \psi''(s) + \psi'''(s) + \cdots.$$

Since any $\psi^{(i)}(r)$ and $\psi^{(j)}(s)$ in the right hand sides above can be merged, the left hand sides can be mergerd. This means that $\Phi(\psi)$ is compatible with E.

Now, we define the sequence of assignment $\psi, \theta_1, ..., \theta_n, ...$ inductively: if a constant k is in $class(x, E)$ then we define $\psi(x) = k$. Otherwise, we define $\psi(x) =$ the trivial tree, namely, $(\{()\}, \phi)$. We define $\theta_1 = \Phi(\psi)$, and $\theta_{n+1} = \Phi(\theta_n)$ for $n > 0$. Clealy θ_1 is compatible with E and normal for E. By what proved above, every θ_n is compatible with E and normal for E. It is easy to see that $\theta_n(x) \leq \theta_{n+1}(x)$ for any variable x. So, we obtain the assingment θ defined by $\theta(x) = sup\{\theta_n(x) | n \geq 1\}$. It is routine to see that θ is normal for E and compatible with E.

We show that θ is a fixpoint of Φ . Let x be a variable. Then

$$\begin{aligned}
\theta(x) &= sup\{\theta_n | n \geq 1\}(x) \\
&= sup\{\theta_n(x) | n \geq 1\} \\
&= sup\{\Phi(\theta_{n-1})(x) | n \geq 1\} \\
&= sup\{sup\{\theta_{n-1}(y) | y \in class(x, E)\} | n \geq 1\} \\
&= sup\{sup\{\theta_{n-1}(y) | n \geq 1\} | y \in class(x, E)\} \\
&= sup\{\theta(y) | y \in class(x, E)\} \\
&= \Phi(\theta)(x).
\end{aligned}$$

This means that $\theta = \Phi(\theta)$. It is essential in these derivations that the merge function $t_1 + t_2$ is *continuous* with respect to t_1 and t_2. Finally, we prove that θ is a solution to E. Let $w \in E$, and let x, y be variables. Let p and α be a PST and a string of labels such that $x \in w$, $p \in w$, and $p//\alpha = y$. It suffices to show that $\theta(x)//\alpha = \theta(y)$. Since $\theta(p) \leq \theta(x)$ by the construction of θ , we obtain $\theta(p//\alpha) \leq \theta(x)//\alpha$. Now we prove $\theta(x)//\alpha \leq \theta(y)$ to conclude the proof.

$$\begin{aligned}
\theta(x)//\alpha &= sup\{sup\{\theta_n(q//\alpha) | n \geq 1\} | q \in w\} \\
&\leq sup\{sup\{\theta_n(q//\alpha) | q \in w\} | n \geq 1\} \\
&\leq sup\{sup\{\theta_n(r) | r \in class(y, E)\} | n \geq 0\} \\
&= sup\{\theta_n(y) | n \geq 1\} \\
&= \theta(y).
\end{aligned}$$

∎

5. Unification

Definition. A *unification configuration* is an ordered pair (P, E) of a system P of equations and an environment E. We define $sol(P, E) = sol(P) \cap sol(E)$. An element in the set is called a *solution to the configuration* (P, E).

We define a unfication algorithm over PST's in the following three steps. First, we define a function *next*:

$$\langle F, A \rangle = next(p, q, E),$$

where E is an environment and both p and q are PSTs. Second, we define a binary relation $\Rightarrow$ between configurations, simply by $(P, E) \Rightarrow ((P - \{p \doteq q\}) \cup A, F)$, where $\langle F, A\rangle = next(p, q, E)$. Finally, we describe our unification algorithm to be a procedure to compute sequences of unification configurations

$$(P_1, E_1) \Rightarrow ... \Rightarrow (P_n, E_n).$$

We use the notation $merge(X, Y, E) = (E - \{X, Y\}) \cup \{X \cup Y\}$ in the next definition.

Definition. For PSTs p, q and an environment E, we define $u = next(p, q, E)$ by the following conditions:

(a) $u = \langle E, \phi\rangle$: if p is a variable and $q \in class(p, E)$,

(b) $u = \langle E, \phi\rangle$: if q is a variable and $p \in class(q, E)$,

(c) $u = \langle merge(class(p, E), class(q, E), E),\ \{u \doteq v | u \in class(p, E), v \in class(q, E)\}\rangle$: if both p and q are variables and $class(p, E) \neq class(q, E)$,

(d) $u = \langle E, pairs(p, q)\rangle$: if both p and q are proper PSTs,

(e) $u = \langle (E - class(p, E)) \cup \{class(p, E) \cup \{q\}\}, \{s \doteq q | s \in class(p, E)\}\rangle$: if only p is a variable,

(f) $u = next(q, p, E)$: if only q is a variable,

(g) $u = \langle E, \phi\rangle$: if p and q are the same constant, and

(h) $u = \langle\langle undefined\rangle\rangle$: otherwise.

Example. Let x and y be variables. Let E be an environment. Then,

$next(1, 1, E) = (E, \phi)$.

$next(1, 2, E) = \langle\langle undefined\rangle\rangle$.

$next(x, y, \{\{x\}, \{y\}\}) = (\{\{x, y\}\}, \{x \doteq y\})$.

$next(x, y, \{\{x, y\}\}) = (\{\{x, y\}\}, \phi)$.

$next(\{a/1, b/x\}, \{b/y, c/2\}, E) = (E, \{x \doteq y\})$.

Definition. We define a binary relation $\Rightarrow$ between unification configurations by the following:

$(P, E) \Rightarrow ((P - \{p \doteq q\}) \cup A, F)$ if $next(p, q, E) = \langle F, A\rangle$ for some $p \doteq q \in P$.

Unification Algorithm:

Input: A system of equations.

Output: Either an enviornment or $FAILURE$.

Method: Let P_0 be the given system of equations P_0. Let E_0 be the environment $E_0 = \{\{x_1\}, ..., \{x_n\}\}$ where $x_1, ..., x_n$ are the variables occuring in P_0. If there is a finite sequence $(P_0, E_0) \Rightarrow ... \Rightarrow (P_n, E_n)$ with $P_n = \phi$, then return E_n as the output, otherwise return $FAILURE$.

Example. Let X and Y be variables. Then,

$$(\{\{a/X, b/X\} \doteq \{b/Y, a/1\}\}, \{\{X\}, \{Y\}\})$$
$$\Rightarrow (\{\{X \doteq 1\}, \{X \doteq Y\}\}, \{\{X\}, \{Y\}\})$$
$$\Rightarrow (\{X \doteq Y\}, \{\{X, 1\}, \{Y\}\})$$
$$\Rightarrow (\{X \doteq Y, 1 \doteq Y\}, \{\{X, 1, Y\}\})$$
$$\Rightarrow (\{X \doteq Y\}, \{\{X, 1, Y\}\})$$
$$\Rightarrow (\phi, \{\{X, Y, 1\}\}).$$

Example. Let X and Y be variables. Then,

$$(\{X \doteq \{a/Y, b/Y\}, Y \doteq \{a/X, b/X\}, X \doteq Y\}, \{\{X\}, \{Y\}\})$$
$$\Rightarrow (\{X \doteq \{a/Y, b/Y\}, Y \doteq \{a/X, b/X\}, X \doteq Y\}, \{\{X, \{a/Y, b/Y\}\}, \{Y\}\})$$
$$\Rightarrow (\{Y \doteq \{a/X, b/X\}, X \doteq Y\}, \{\{X, \{a/Y, b/Y\}\}, \{Y\}\})$$
$$\Rightarrow (\{Y \doteq \{a/X, b/X\}, X \doteq Y\}, \{\{X, \{a/Y, b/Y\}\}, \{Y, \{a/X, b/X\}\}\})$$
$$\Rightarrow (\{X \doteq Y\}, \{\{X, \{a/Y, b/Y\}\}, \{Y, \{a/X, b/X\}\}\})$$
$$\Rightarrow (\{X \doteq Y, Y \doteq X\}, \{X, Y, \{a/Y, b/Y\}, \{a/X, b/X\}\})$$
$$\Rightarrow (\{Y \doteq X\}, \{X, Y, \{a/Y, b/Y\}, \{a/X, b/X\}\})$$
$$\Rightarrow (\phi, \{X, Y, \{a/Y, b/Y\}, \{a/X, b/X\}\})$$

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

Proposition 5.1. *The unification algorithm terminates.*

Proof. From simple combinatorics, there are only finite environments appearing on the sequence of configurations of a unification process. The environments on a given unification process are monotone in the sense that if $w \in E$ and $(P, E) = (P', E')$, $w \subset w'$ for some $w' \in E'$.

Assume that the algorithm does not terminate. Then there must be some integer k such that all the environments after initial k steps are saturated. So, the sequence can be written

$$... \Rightarrow (P_k, E_k) \Rightarrow (P_{k+1}, E_k) \Rightarrow (P_{k+2}, E_k) \Rightarrow ...$$

where each $P_j \neq \phi$ with $j \geq k$.

By the definition of $\Rightarrow$, we have either the following (1) or (2) for any $j \geq k$.

(1) $P_{j+1} \subset P_j$, $P_{j+1} \neq P_j$.

(2) $p \doteq q \in P_j$ and $P_{j+1} = (P_j - \{p \doteq q\}) \cup pairs(p, q)$ for some proper PSTs p and q. By double induction on the cardinality of P_k and the maximum height of the equations appearing in P_k, it is proved that there

can not be an inifinite sequence of finite sets which satisfies the above conditions. This is a contradiction. This concludes that unification terminates. ▮

Proposition 5.2. *The unification preserves the set of solutions. That is, if* $(P, E) \Rightarrow (P', E')$ *then* $sol(P, E) = sol(P', E')$.

Proof. By the definition of $(P, E) \Rightarrow (P', E')$, there is an equation $p \doteq q \in P$ such that $P' = (P - \{p \doteq q\}) \cup D$, and $next(p, q, E) = (E', D)$. According to the definition of $\Rightarrow$, we prove the theorem dividing into the following cases.

(a) Suppose p is a variable with $q \in class(p, E)$. Namely, we suppose $E' = E$ and $P' = P - \{p \doteq q\}$. It is clear that $sol(P, E) \subset sol(P', E)$. If $\theta \in sol(P', E)$ then θ is a solution of $p \doteq q$ since p and q are members of the same class in E. So θ is a solution to $p \doteq q$. Therefore θ is a solution to P. Then $\theta \in sol(P, E)$.

(b) Suppose q is a variable with $p \in class(q, E)$. It is similar to the case (a).

(c) Suppose both p and q are variables with $class(p, E) \neq class(q, E)$. That is, we suppose that

$$P' = (P - \{p \doteq q\}) \cup \{u \doteq v | u \in class(p, E), v \in class(q, E)\}$$

and

$$E' = merge(class(p, E), class(q, E), E).$$

If $\theta \in sol(P, E)$ then we have $\theta(p) = \theta(q)$ from $p \doteq q \in P$. Let $u, v \in class(p, E) \cup class(q, E)$. By the lemma 4.1 θ is a solution of $u \doteq v$ since $\theta \in match(u, \theta(p)) \cap match(v, \theta(p))$. From this, θ is a solution to both P' and E'. So we have $\theta \in Sol(P', E')$. Therefore, we have $sol(P, E) \subset sol(P', E')$.

We prove the reverse direction. Suppose $\theta \in sol(P', E')$. θ is a solution to $p \doteq q$ since $class(p, E) \cup class(q, E) \in E'$. So we have $\theta \in sol(P)$, that is, $sol(P', E') \subset sol(P)$. It is clear that $sol(E') \subset sol(E)$. Therefore we have $sol(P', E') \subset sol(P, E)$.

(d) Suppose both p and q are proper PSTs. That is, we suppose $E' = E$ and $P' = (P - \{p \doteq q\}) \cup pairs(p, q)$. The solutions to $p \doteq q$ are the solutions of $pairs(p, q)$ by definition. Then we have $sol(P, E) = sol(P', E')$.

(e) Suppose only p is a variable. That is, we suppose $E' = (E - \{class(p, E)\}) \cup \{class(p, E) \cup \{q\}\}$ and $P' = (P - \{p \doteq q\}) \cup \{u \doteq q | u \in class(p, E)\}$.

Suppose $\theta \in sol(P, E)$. With $\theta \in match(q, \theta(p))$ in mind, we have $\theta \in match(u, \theta(p)) \cap match(v, \theta(p))$ for any $u, v \in class(p, E) \cup \{q\}$. Applying lemma 4.1, θ is a solution of $u \doteq v$. So θ is a solution to E'. Suppose $u \in class(p, E)$ and $u \doteq q \in P'$. Applying lemma 4.1, from $\theta \in match(u, \theta(p))$ and $\theta \in match(q, \theta(p))$, it follows that θ is a solution to $u \doteq q$. So θ is a solution to P'. Therefore, we have $\theta \in sol(P', E')$. This concludes $sol(P, E) \subset sol(P', E')$.

We prove the reverse direction. Suppose $\theta \in sol(P', E')$. From $sol(E') \subset sol(E)$, we have $\theta \in sol(E)$. From $class(p, E) \cup \{q\} \in E'$, it follows that θ is a solution to $p \doteq q$. So θ is solution to P. Then, we have $\theta \in sol(P, E)$, namely, $sol(P', E') \subset sol(P, E)$.

(f) Suppose only q is a variable. It is similar to the case (e).

(g) Suppose both of p and q are an identical constant. That is, we suppose $P' = P - \{p \doteq q\}$ and $E' = E$. Clearly we have $sol(P, E) = sol(P', E')$.

(h) Suppose $p \doteq q$ is a conflict. There are no P' and E' in this case. So we need not consider this case. ∎

Remark. Let p and q be PSTs, and let ρ and σ be assignments. p and q may not be unifiable even if $\rho(p)$ and $\rho(q)$ can be merged. Also p and q may not unifiable even if $\sigma(q) \geq \sigma(p)$.

Example. Let $p = \{a/X, b/X\}$, $q = \{a/\{c/1\}, b/\{c/2\}\}$. Cleary, p and q are not unifiable. PTTs $\rho(p)$ and $\rho(q)$, however, can be merged for ρ with $\rho(X) = \{d/3\}$. Also, we have $\sigma(q) \geq \sigma(p)$ for σ with $\sigma(X) = \phi$.

Proposition 5.3. *Let P be a system of equations. Then conditions (1) and (2) are equivalent:*

(1) *P has a solution.*

(2) *P is unifiable.*

Proof. (1) $\Rightarrow$ (2). Let θ be a solution to P. We can soppose $P = \{s \doteq s'\}$ without loss of generality. Let $P_0 = P$ and $E_0 = \{\{x\} | x$ is a variable occuring in s or $s'\}$. Consider a maximal sequence $(P_0, E_0) \Rightarrow ... \Rightarrow (P_n, E_n)$ with length n. From the termination property of the algorithm, n is finite. Applying proposition 5.2, we have $sol(P_{i-1}, E_{i-1}) = sol(P_i, E_i)$. From $\theta \in sol(P_0, E_0)$, we have $sol(P_i, E_i) \neq \phi$ for any i with $0 \leq i \leq n$. From $sol(P_i, E_i) \neq \phi$, P_i can have no conflict. So, we have $P_n = \phi$ by the maximality of the sequence. This means just that s and s' are unifiable.

(2) $\Rightarrow$ (1). There is a sequence $(P_0, E_0) \Rightarrow ... \Rightarrow (\phi, E_n)$ with P_0, E_0 and E_n above. It is easy to show that E_n is congruent and has no conflict. So E_n has a solution by proposition 4.4. So P_0 has a solution since $sol(P_0) = sol(E_n)$ by proposition 5.2. ∎

Corollary 5.4. *If $\rho(p) = \rho(q)$ then p and q are unifiable.*

Proof. The equation $p \doteq q$ has a solution ρ . So, u and v are unifiable from proposition 5.3. ∎

6. Program and SLD Derivation

Definition. A *program clause* is a pair (p, B) consisting of a PST p and a finite set B of PSTs. A *goal* is a finite set of PSTs. is A *program* is a finite set of program clauses.

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

$$p.$$

A program clause of the form $(p, \{p_1, ..., p_n\})$ $(n \neq 0)$ is written

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

A derivation configuration is an ordered pair (G, E) of a goal G and an environment E.

Definition. We write $(G, E) \rightarrow (G', E')$ iff the condition (1) and (2) below are satisfied, where G and G' are goals, E and E' are environments and $G = \{p_1, ..., p_n\}$ for some *PTT*s $p_1, ..., p_n$.

(1) There are n copies $q_i \leftarrow q^i_1, ..., q^i_{k_i}$ of some program clauses such that $G' = \{q^1_1, ..., q^1_{k_1}, ..., q^n_1, ..., q^n_{k_n}\}$.

(2) All pairs of heads and bodies are unifiable. Namely, $(\{p_1 \doteq q_1, ..., p_n \doteq q_n\}, E) \Rightarrow ... \Rightarrow (\phi, E')$.

When we need express explicitly the program clauses which are used in this transition, we write

$$(G, E) \rightarrow (G', E') \qquad (\Theta)$$

where $\Theta = \{(p_i, q_i \leftarrow q^i_1, ..., q^i_{k_i}))|1 \leq i \leq n\}$.

The *initial environment* of a goal is the set of singletons $\{x\}$ such that x is a variable occuring in the goal.

Definition. A finite sequence $(G, E_0) \rightarrow ... \rightarrow (\phi, E)$ is a (SLD) derivation of a goal G where E_0 is an *initial environment of G.*

7. Model

Definition. Let Π be a program. A non-empty set M of *PTT*s is a *model* of Π iff for any assignment θ, program clause $p \leftarrow p_1, ..., p_n$ in Π , a *PTT* t, and *PTT*s $t_1, ..., t_n \in M$; if $\theta \in match(p_i, t_i)$ and $\theta \in match(p, t)$, then $t \in M$.

Proposition 7.1. *The intersection of models of the program is a model of the program.*

Proof. Obvious. ∎

Theorem 7.2. *There exists the least model of the program.*

Proof. The interection of all models is the least model. ∎

Proposition 7.3. *The set of models of the program is chain complete. That is, if* $M_1 \subset ... \subset M_n \subset ...$ *is a chain of models then* $M = \bigcup\{M_i | i \geq 1\}$ *is a model.*

Proof. Suppose $t_1, ..., t_n \in M$, $p \leftarrow p_1, ..., p_n \in \Pi$ and $(\bigcap\{match(p_i, t_i)|1 \leq i \leq n\}) \cap match(p, t) \neq \phi$. There is an integer k such that $\{t_1, .., t_n\} \subset M_k$. Since M_k is a model, we have $t \in M_k$. Then we have $t \in M$. Therefore, M is a model. ∎

To characterize the least model, Let us define a function Ψ such that, for a set S of trees,

$$\Psi(S) = \{t | \text{there are some } \theta, \text{ a program clause } p \leftarrow p_1, ..., p_n \in \Pi, \text{ and } t_1, ..., t_n \in S$$
$$\text{such that } \theta \in \bigcap\{match(p_i, t_i)|1 \leq i \leq n\} \text{ and } \theta \in match(p, t)\}.$$

Proposition 7.4. *The transformation* Ψ *is monotone.*

Proof. Obvious. ∎

Proposition 7.5. *The condition* (1) *and* (2) *are equivalent.*

(1) $\Psi(S) \subset S$.

(2) *S is a model of the program.*

Proof. Obvious. ∎

Now, we have obtained the monotonic transformation over the chain complete lattice, which consists of all the models of the given program. By the lattice theory, the following theorem 7.6 and 7.7 are easily proved [Lloyd 84].

Theorem 7.6. *There exists the least fixpoint of the transformation Ψ.*

Theorem 7.7. $M_n \uparrow \omega =$ *the least model* $=$ *the least fixpoint of Ψ, where $M_0 = \phi$ and $M_{n+1} = \Psi(M_n)$ with* $n \geq 0$.

8. Soundness and Completeness

Let M be the least model of the program Π .

Proposition 8.1. (Soundness) *Let $G = \{p_1, ..., p_n\}$ be a non-empty goal, and let E be an initial environment of G. Let $(G, E) \rightarrow ... \rightarrow (\phi, E')$ be a derivation and assume that $\rho \in sol(E')$. Then for any PST in G and any solution ρ in $sol(E')$, there is some t in M such that ρ is a matching assignment from p to t:* $\rho \in match(p, t)$.

Proof. We prove by induction on the length l of the derivation. Suppose $G = \{p_1, ..., p_n\}$.

(1) Suppose $l = 1$. There are n copies $q_1, ..., q_n$ of some unit clauses of the program Π such that $(\{p_1 \doteq q_1, ..., p_n \doteq q_n\}, E) \Rightarrow ... \Rightarrow (\phi, E')$. Let $p = p_i$ be an element in the goal G. Applying proposition 5.2, ρ is a solution of each $p_i \doteq q_i$, since ρ is a solution to E'. Then, by lemma 4.2, we have $\rho \in match(q_i, \rho(p_i) + \rho(q_i))$. Then, since M is a model of Π , also we obtain that $t = \rho(p_i) + \rho(q_i)$ is in M. Applying lemma 4.2 again, we finally obtain $\rho \in match(p_i, t)$.

(2) Suppose that $l > 1$ and that this proposition holds for $l' < l$. Let E be the initial environment of G. The whole derivation can be written $(G, E) \rightarrow (G'', E'') \rightarrow ... \rightarrow (\phi, E')$, where, $G'' = \{q^1_1, ..., q^1_{k_1}, ..., q^n_1, ..., q^n_{k_n}\}$ for some copies $q_i \leftarrow q^i_1, ..., q^i_{k_i}$ of program caluses $(1 \leq i \leq n)$. From the definition of the derivation, we can suppose $(\{p_1 \doteq q_1, ..., p_n \doteq q_n\}, E) \Rightarrow ... \Rightarrow (\phi, E'')$. By the induction hypothesis, for any solution to E', there are $k_1 + ... + k_n$ $PTTs$, $t^1_1, ..., t^1_{k_1}, ..., t^n_1, ..., t^n_{k_n}$ in M such that $\rho \in match(q^i_j, t^i_j)$ for any i, j with $(1 \leq i \leq n, 1 \leq j \leq k_i)$. For each i, by lemma 4.2 with $t = \rho(q_i) + \rho(p_i)$, we obtain $\rho \in match(q_i, t)$, since ρ is a solution to $p_i \doteq q_i$. Because M is a model of Π , we have $t \in M$. Applying lemma 4.2 again, we finally obtain $\rho \in match(p_i, t)$. ∎

We prove the completeness of our SLD derivation. Let Π be a porogram and let $M_0 = \phi$ and $M_{n+1} = \Psi(M_n)$.

Suppose that we are given the following data:

m: a positive integer,

t_i: $t_i \in M_m (1 \leq i \leq n)$,

G: a goal $G = \{p_1, ..., p_n\}$,

E: an environment,

θ : $\theta \in sol(E)$ and $\theta \in match(p_i, t_i)$ for any $i(1 \leq i \leq n)$.

Lemma 8.2. *For the above data, there are the following:*

G': *a goal,*

E': *an environment,*

$q_i \leftarrow q^i_1, ..., q^i_{k_i}$: *copies of program clauses* $(k_i \geq 0, 1 \leq i \leq n)$,

$t^1_1, ..., t^1_{k_1}, ..., t^n_1, ..., t^n_{k_n}$: *elements in* M_{m-1}, *and*

ρ : *an extension of* θ , *satisfying the following conditions.*

(1) $(\{p_1, ..., p_n\}, E) \rightarrow (G', E')$.

(2) $G' = \{q^1_1, ..., q^1_{k_1}, ..., q^n_1, ..., q^n_{k_n}\}$,

(3) $\rho \in sol(E) \cap sol(E') \cap sol(\{p_1 \doteq q_1, ..., p_n \doteq q_n\})$,

$\rho \in match(p_i, t_i)$,

$\rho \in match(q_i, t_i)$,

$\rho \in match(q^i_j, t^i_j)$ $(1 \leq i \leq n, 1 \leq j \leq k_i)$.

(4) $(\{p_1 \doteq q_1, ..., p_n \doteq q_n\}, E) \Rightarrow ... \Rightarrow (\phi, E')$.

Proof. Since $t_i \in M_m$, from the definition of M_m, we obtain program clauses $q_i \leftarrow q^i_1, ..., q^i_{k_i} \in \Pi$, *PTT*s $t^i_1, ..., t^i_{k_i} \in M_{m-1}$, and assignments ρ_i such that $\rho_i \in match(q_i, t_i)$ and $\rho_i \in match(q^i_j, t^i_j)$ for each i, j $(1 \leq i \leq n, 1 \leq j \leq k_i)$. Since copies of two distict program clauses can be assumed to share no variable, we can choose them so that $dom(\rho_i) \cap dom(\rho_j) = \phi \quad (i \neq j)$. Also, we can suppose that for each variable x occuring in $q_i \leftarrow q^i_1, ..., q^i_{k_i}$ appears in only in the class $class(x, E) = \{x\}$ in E and that $\theta(x)$ is undefined, because x is a new variable. With this and the fact that p_i and q_i have no variable in common, i.e., $dom(\theta) \cap dom(\rho_i) = \phi$, we can define $\rho = \theta \cup \rho_1 \cup ... \cup \rho_n$.

Since ρ is an extension of θ , we have $\rho \in match(p_i, t_i)$ and $\rho \in match(q_i, t_i)$, and ρ is a solution to E. That is, we have $\rho \in sol(\{p_1 \doteq q_1, ..., p_n \doteq q_n\}, E)$. So, by proposition 5.2, we have $(\{p_1 \doteq q_1, ..., p_n \doteq q_n\}, E) \Rightarrow ... \Rightarrow (\phi, E')$. Then $\rho \in sol(E')$, since our unification preserves the set of solutions. By the definition of SLD derivation, we obtain $(\{p_1, ..., p_n\}, E) \rightarrow (G', E')$, where $G' = \{q^1_1, ..., q^1_{k_1}, ..., q^n_1, ..., q^n_{k_n})\}$. Therefore, we obtain all from (1) to (4). ∎

Proposition 8.3. *Let m, G, E, and θ be a positive integer, the goal $\{p_1, ..., p_n\}$, an environment E and a solution to E. Assume that there are PTTs $t_i \in M_m$ and that $\theta \in match(p_i, t_i)$ for each $i (1 \leq i \leq n)$. Then there is a derivation of length m whose first configuration is (G, E).*

Proof. Prove this by repeated use of lemma 8.2 m times. ▮

Let x and t be a PST and a PTT. If t is in the least model of Π and $match(p, t) \neq \phi$. Then, from proposition 8.3, we have $(\{p\}, E) \rightarrow ... \rightarrow (\phi, E')$.

Lemma 8.4. *Let $G = \{p_1, ..., p_n\}$ and $\Theta = \{(p_1, q_1 \leftarrow \gamma_1), ..., (p_n, q_n \leftarrow \gamma_n)\}$. Let E and F be environments. Suppose*

(1) $(G, E) \rightarrow (G', E')$ (Θ), *and*

(2) $sol(E) \subset sol(F)$.

Then, we obtain

(3) $(G, F) \rightarrow (G', F')$ (Θ), *and*

(4) $sol(E') \subset sol(F')$ *for some environment* F'.

Proof. Let $P = \{p \doteq q | \exists \gamma (p, q \leftarrow \gamma) \in \Theta\}$. Then, from (1), we have $(P, E) \Rightarrow ... \Rightarrow (\phi, E')$. Then, $sol(P, E) \neq \phi$. From (2), $sol(P, F) \neq \phi$, since $sol(P, F) \supset sol(P, E)$. From proposition 5.2, $sol(F') = sol(P, F) \supset sol(P, E) = sol(E')$. So we obtain $sol(E') \subset sol(F')$. Applying the definition of the relation $\rightarrow$, we obtain,

$$(G, F) \rightarrow (G', F') \qquad (\Theta)$$

▮

Definition. An environment E is an *answer environment* of G iff the following conditions hold:

(1) $class(x, E)$ has exactly two elements if $x \in vars(G)$,

(2) $class(x, E) = \{x\}$ if x is not in $vars(G)$, and

(3) $vars(p) \cap vars(G) = \phi$ if $p \in class(x, E)$ and $p \neq x$.

Let M be the least model of the given program.

Definition. An answer environment E of a goal G is a *correct answer environment* iff for any solution $\rho \in sol(E)$ and $p \in G$, there is $t \in M$ such that $\rho \in match(p, t)$.

Definition. We define the set $accvars(G, E)$ of reachable variables for a derivation configuration (G, E) by the following recursive definitional equation:

$accvars(G, E) = vars(G) \cup (\bigcup\{accvars(p, E) | \exists x \in vars(G)\, p \in class(x, E)\})$.

Example. $accvars(\{\{a/x\}, \{b/y\}\}, \{\{x, r\}, \{y, s\}, \{u\}, \{v\}\}) = \{x, y, r, s\}$.

Proposition 8.5. (Completeness) *Let M,G,E, and E_0 be the least model of the program, a goal, a correct answer environment of G, and the initial environment of G. Then there is an environment E' satisfying the following:*

(a) *there is a derivation $(G, E_0) \rightarrow \dots \rightarrow (\phi, E')$, and*

(b) *the restriction of any solution $\theta \in sol(E)$ to $accvars(G, E)$ can be extended to some solution $\theta' \in sol(E')$.*

Proof. Let $Q = \{y | y$ is a variable, $\exists x \exists p \in class(x, E) p \neq x$ and $y \in vars(p)\}$. Let F be a renamed version of E by substituting a new constant for each occurrence of variable $x \in Q$ at every doubletons in E. Then also F is a correct answer environment.

So, applying lemma 8.3, there is a derivation

(1) $\quad (G, F) \rightarrow \dots \rightarrow (\phi, E'')$

for some environment E''. Let c be any constant which was introduced in the renaming above. Then, for any $w \in E''$ with $c \in w$, if $u \in w$ and $u \neq c$ then u is a variable. Also it is clear that if $y \in Q$ then $class(y, E'') = \{y\}$.

Applying lemma 8.4 repeatedly, the derivation (1) is *simulated* by (2) below, to obtain an environment E'''.

(2) $\quad (G, E) \rightarrow \dots \rightarrow (\phi, E''')$.

Since (2) is a simulation of (1), for each $y \in Q$, $class(y, E''')$ consists of only variables. Also applying lemma 8.4 to derivation (2) repeatedly, we obtain an environment E' as follows.

(3) $(G, E_0) \rightarrow \dots \rightarrow (\phi, E')$,

(4) $sol(E''') \subset sol(E')$.

Since $class(y, E''')$ consists of only variabls for each $y \in Q$, i.e., each y in $Q$is sl not instantiated, each restriction θ' of a solution of E to $accvars(G, E)$ can be extended to some element of E'''. With this and (4) together, θ' can be extended to some solution to E'. This concludes the proof. ▮

9. Extended Definite Clause Grammar

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

9.1. Extended Term

Let F be a set of functors. Assuming each functor is assigned an arity, we define the least set *TERM* satisfying:

(1) $TERM$ includes every variable;

(2) $TERM$ includes every constant;

(3) $TERM$ includes every form of $\{a_1/x_1, ..., a_n/x_n\}$, where each a_i is label and each x_i is in $TERM$; and

(4) $TERM$ includes every form of $z(x_1, ..., x_n)$, where each x_i is in $TERM$ and z is of arity $n > 0$.

Each element in $TERM$ is called a term.

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

```
    [{a/[X|Y], b/f(X,Y)}, X].
```

9.2. Extended PTT

We define an extended PTT to be an ordered pair (T, f), such that $dom(f)$ is a subset of T and such that if $f(x) = h$ then $\{1, ..., n\} = \{a|x * \langle a \rangle \in T\}$ where n is the arity of h. Replacing the $TERM$ and the extended PTT for PST and PTT in the proposition, it is easy to show that all propositions we have developed so far in the previous sections hold.

9.3 Extended DCG

We define the extended DCG by giving examples as follows. Rule (1) below is a grammar rule, which is a pair of a PST and a list of PST's and equality constraints. Unit clauses (2) and (3) are for lexical items.

```
    (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)}}).
```

The clauses from (4) to (8) below 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

I would like to thank Dr. C. Pollard and Dr.J. Goguen for their useful comments to this work. Also I would like to thank Dr. T. Yokoi and Dr. K. Furukawa for their encouragement.

REFERENCES

[Ait-Kaci 84] H. Ait-Kaci: *A Lattice Theoretic Approach to Compuation Based on a Calculus of Partially Ordered Type Structures,* a Dissertation in Computer and Information Science, University of Pennsylvania, 1984.

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

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

[Jaffar&Lassez& 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& Round 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.

[Mukai 85] K. Mukai: *Horn Clause Logic with Parameterized Types for Situation Semantics Programming,* ICOT Technical Report No. TR101, 1985.

[Mukai& 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, 1985.

[Pereira& 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& 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*, Aritificial Intelligence 13:231-278, 1980.

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

[Porto 86] A. Porto: *Semantic Unification for Knowledge Base Deduction,* in this Preprints, 1986.

[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.