ICOT Technical Report: TR-113

TR-113

Unification over Complex Indeterminates in Prolog

by
Kuniaki Mukai

July, 1985

©1985, ICOT

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

UNIFICATION OVER COMPLEX INDETERMINATES IN PROLOG

Kuniaki Mukai

ICOT Research Center
Institute for New Generation Computer Technology
Tokyo, Japan

ABSTRACT

Complex indeterminates and their notation are introduced into Prolog. The syntax and semantics of the new language are described. The language builds in an extension to the usual unification. A set of equality axioms are given to make the semantics clear. Some examples are given to demonstrate ideas of discourse understanding processing based on situation semantics in our proposed language.

1 INTRODUCTION

We proposed a programming language named CIL in Mukai 1985a. Main objectives of CIL are to support ideas of situation semantics [Barwise & Perry 1983, Barwise 1984] in application to discourse understanding processing. As a first step, complex indeterminates (indeterminates for short) were introduced into Prolog, and the unification was extended to the indeterminates. Also the control primitive "freeze" in Prolog II [Colmerauer 1982] was introduced to make CIL a kind of constraint language. We expressed these ideas in the equation:

CIL = Horn clause logic
+ types and indeterminates
+ freeze

We propose a reconstruction of the basic part of CIL around the indeterminate. That is, the main points of this paper are:

(1) Introducing the notation for complex indeterminate.
Description of the syntax and semantics of CIL focusing on the indeterminate.

As the result of this reconstruction, the two predicates named "instance" and "del(declare)" in the previous version become unnecessary.

In Section 2, we illustrate the motivation for CIL by very simplified example in discourse understanding processing. It is a revised version of the one in the previous report [Mukai 1985a]. The new example shows that CIL is simpler but has more expressive power. For instance, the example in the previous report used the two primitives "instance" and "del." In Section 3, we give an outline of CIL. Some examples show the basic idea of our unification over indeterminates and the way in which they are used. Descriptions of built-in predicates including "freeze" in CIL are omitted except two important primitive predicates: "create_type" and "type_of", which are explained by a question-answering example. In Section 4, we describe a model theoretic semantics for CIL. It is necessary for the Herbrand universe to be extended to be the domain of CIL. Roughly speaking, the new domain is the set of all directed graphs with labels which has a loop possibly. The semantics of a CIL program is defined to be the minimal model satisfying the program. The semantics of CIL is described within the framework of the foundations of logic programming language developed in Jaffar & Lassez & Maher 1983, Emden & Lloyd 1984 and Lloyd 1984. In Section 5, we give computation rules for CIL. Then we give the conjecture that the computation rules are complete and sound. In Section 6, we give some concluding remarks.

We have omitted the description of the implementation of the extended unification because it is essentially the same as the one in the previous report [Mukai 1985a].

2 UTTERANCE AND DISCOURSE SITUATION

We describe an illustrative program named "discourse_constraint" in detail, showing some ideas of situation semantics programming in CIL. The predicate discourse_constraint is a fragment of the discourse analysis model. This example includes the following features in discourse models:

(1) definitions of lexical elements as "parameterized types (types for short),"
(2) a definition of a grammar, including syntax and semantics descriptions, as constraints over these types,
(3) the definition of discourse situation as a type,
(4) the definition of meaningful option as a type,
(5) the simple conversational constraint over discourse that the roles of hearer and speaker change from one to the other in turn, and
(6) the interpretation of a sentence depends on the discourse situation.

The last feature will be demonstrated by the sentence "i(=I) love you." The sentence will have two interpretations depending on the discourse situations involved. The definitions of discourse situation and meaningful option are taken from Barwise & Perry 1983. No connective situations are included in the example for simplicity.

Now, we review the definition of "type" and "indeterminate" in Barwise 1984 briefly. A complex indeterminate is an ordered pair of another indeterminate and a condition. It is written

x|c

where x and c are the indeterminate and the condition. The complex indeterminate determines the type of which it is an instance. The type is written

[x|c].

This notation should not be confused with a term for a list in Prolog. The condition c is typically the one on situations. A situation is a set of elements called (abstract) facts. A fact is an abstract located fact or an abstract unlocated fact. The example uses the former only. The form

$$(l, (r, a_1, ..., a_n), p)$$

is the notation for the abstract located fact that the objects $a_1, ..., a_n$ stand ($p = 1$) or do not stand ($p = 0$) in the relation r at the location l.

We introduce the complex indeterminate into Prolog with a slight modification for computational efficiency as seen below. Let s, i, you, exp, and here be indeterminates. The following type of discourse situation

discourse_situation = [x|c],

where

c = in(s, (here, (speaking, i), 1))&
in(s, (here, (addressing, you),1))&
in(s, (here, (utter, exp),1)),

is translated directly into a Horn clause of CIL as follows:

```
discourse_situation(
        (S with speaker:= I,
                hearer:= You,
        discourse_location:= Here,
                expression:= Exp
           where
        has(S,(Here, (speaking,I), 1))&
        has(S,(Here, (addressing, You),1))&
        has(S,(Here, (utter, EXP), 1))
           )<—true.
```

This Horn clause is a declaration of the following:

(1) Any situation S is an instance of the type "discourse_situation" if S has three circumstances of the forms
(Here, (speaking,I), 1),
(Here, (addressing, You),1), and
(Here, (utter, Exp),1).

(2) The symbols "speaker," "hearer," "discourse_location," and "expression" are names of the roles of the objects anchored by the indeterminates I, You, Here, and Exp, respectively.

(3) For any discourse situation D in this sense, the values of the roles speaker, hearer, discourse_situation, and expression can be expressed by "speaker!D," "hearer!D," "discourse_location!D," and "expression!D" respectively.

Thus, a term of the form (X with Y where Z) above are the notation for complex indeterminates. The atomic formula "has(S, X)" is a constraint (condition) which is defined in CIL with the semantics that the set S includes X as a member.

The following Horn clause is a description of the type "meaningful_option" of a sentence.

```
meaningful_option(
   (MO with discourse_situation:= DS
        where
    sentence(MO, expression!DS, [ ])
      })<—true.
```

This clause states the fact that the meaningful option of the given utterence depends on the discourse situation. The dependency constraint is described by the relation "sentence".

As explained above, the term "expression!DS" refers to the value of the "expression slot" in the complex indeterminate DS.

The syntax categories in the example language are "sentence," "noun," and "verb." The lexical elements are "I," "you," and "love." Each category is defined by the predicate with the same name. Each predicate has three arguments. The first argument is an indeterminate including all or part of the following roles depending on the predicate.

(1) discourse_situation : the discourse situation,

(2) agent : the agent case,

(3) object : the object case,

(4) situation : the situation to which the category makes some contribution,

(5) polarity : the polarity of the sentence.

The last two arguments taken together as a difference list represent the given sequence of surface words. Let p and x be a syntax category symbol and a set of features respectively, and let h and t be two sequences of words. The atomic formula of the form p(x,h,t) means that the sequence of the words given as the difference between h and t is in the phrase category p and that x is the features for the sequence. This is the idea of the definite clause grammar embedded in Prolog [Pereira & Warren 1980].

The sentential form and its semantics allowed in the example language are described by the predicate "sentence." Any sentence must be a sequence of noun, verb, and another noun in order. The sentence is assumed to represent a meaningful option (described situation). The discourse situation is an environment for computing the option. The Horn clause of the form H<—B is read : B implies H. The symbol "&" is logical conjunction.

```
sentence({MO with
      discourse_situation:= DS),A,B)<—
  noun({Agent with
      discourse_situation:= DS), A,A1) &
  verb({MO with
      discourse_situation:= DS,
                        agent:= Agent,
                       object:= Object,
                     polarity:= 1),
           A1,A2) &
  noun({Object with
      discourse_situation:= DS), A2,B).
```

The notation (X with Y) above is a shorthand for the complex indeterminate (X with Y where true). This rule means that the agent and object cases of the verb are the first and second nouns respectively and that the meaningful option of the sentence contains this circumstance within it.

The noun category represents an object standing in some relation which is the interpretation of the verb phrase. The noun category is assumed to have a context playing "discourse_situation" role. The object is in this context.

The function of the verb category is to add the circumstance to the given situation. The constituents of the circumstance are integrated into the verb category. The verb category itself represents the situation. In situation semantics terminology, the verb category is the indexed type of situation parameterized by the roles "discourse_situation," "object," and "agent." The parameter "discourse_situation" provides the location of the circumstances.

The following are the definitions of the lexical elements. The first clause defines the pronoun "i" as the speaker of the given discourse situation. The others are defined in similar ways.

```
noun({speaker!DS with
     discourse_situation:= DS),
     [i|A],A)<—true.

noun({hearer!DS with
     discourse_situation:= DS),
     [you|A],A)<—true.

verb({S with
       discourse_situation:= DS,
                     agent:= Agent,
                    object:= Object,
                  polarity:= Pol
         where
       has(S, {discourse_location!DS,
                 {love, Agent, Object},
                  Pol})),
       [love|Z],Z)<—true.
```

The predicate "discourse_constraint" is a constraint over the conversational discourse between two persons. Let D1, ..., Dn be a sequence of discourse situations and let M1, ..., Mn be a sequence of situations. The condition

```
discourse_constraint([D1, ..., Dn],
                     [M1, ..., Mn])
```

holds if for each 1=< i =< n, Mi is a meaningful option of Di and for each 1< j=< n, Dj is the next discourse situation to Dj-1. The constraint concerning the conversational roles is described by the predicate "change_role" below.

```
discourse_constraint([ ],[ ])<—true.
discourse_constraint([X],[M])<—
     meaningful_option((M with
          discourse_situation:=X)).
discourse_constraint([X,Y|Z],
                     [Mx,My|R])<—
     change_role(X, Y)&
     meaningful_option(
          (Mx with discourse_situation:=X))&
     discourse_constraint([Y|Z],
                          [My|R]).
```

The following is the conversational "maxim" describing the fact that the roles of the speaker and the hearer must alternate in turn every time the speaker has uttered a sentence. The special form is introduced to represent successive discourse locations for convenience. It is a difficult problem to handle location in detail and is outside of the scope of the paper.

```
change_role(CDS,
   (NDS:discourse_situation
        where
          speaker:= hearer!CDS,
           hearer:= speaker!CDS,
        discourse_location
                :=
          next(discourse_location!CDS))
                  )<—true.
```

The notation of the form x:y above is a shorthand for (x where y(x)). Execution looks like this:

```
?- discourse_constraint(
    [(@discourse_situation with
           expression:= [i,love,you],
              speaker:= jack,
               hearer:= betty,
     discourse_location:= loc01),
      (@discourse_situation with
           expression:= [i,love,you])],
    MOs)&
  print(MOs).

OUTPUT:
   [ (loc01, (love, jack, betty), 1),
     (next(loc01), (love, betty, jack), 1) ]
```

The notation of the form @p above is a shorthand for (x where p(x)), where x is an anonymous variable. The result shows that the same sentence has different interpretation depending on the discourse situations.

3 EXTENDED UNIFICATION

Here, we give an outline of CIL, focusing on extended unification.

3.1 Outline of CIL

CIL is a Horn clause based programming language like DEC-10 Prolog [Bowen 1982]. A program consists of Horn clauses. CIL has some primitives to freeze goals and to access the parameters of complex indeterminates. Conditional statements are used like that in Lisp instead of "cut" statements. Complex indeterminates are central objects of the language and can be written anywhere freely in the program. The freeze statement is a control primitive to support delayed conditions specified in complex indeterminates.

We provide examples to help explain some basic elements of CIL. The symbols '>' and '>>' are system and user input prompt respectively.

3.2 Complex Indeterminate

As seen in the previous section, we have extended the notion of complex indeterminate given in Barwise & Perry 1983 and Barwise 1984 so that the parameters can be accessed from outside it. The complex indeterminate looks like a frame. However unification is the only way to sign a value to a slot. The current slot value is always instance of the previous one.

Complex indeterminates will be useful not only for semantics and pragmatics but also for syntax. As pointed out in Mukai 1985b, the complex indeterminate provides a unified view over the features of GPSG [Gazdar & Klein & Pullum & Sag 1985] and the functional structure of LFG [Bresnan 1983]. A feature is a complex indeterminate whose condition part is trivial, i.e., "true". A functional structure is a complex indeterminate whose condition is composed of only equality, conjunction, disjunction, and negation. On the other hand, the condition of a complex indeterminate in CIL can have any relations defined by the Horn clauses as its constituents.

3.3 Unification

The central idea of our extended unification is shown by the following clause:

(1) unify((x with r:=a, s:=b where p), (y with s:=c, t:=d where q)) if unify(x, y) & unify(b,c) & solve(p) & solve(q).

(2) unify((x with g where p), y) if unify(x, y) & solve(p), provided y is not a complex indeterminate.

3.4 Using Indeterminates and Roles

The following example shows how to use roles in complex indeterminates. Notice that the indeterminates Boy and Child in the example are unknown except that they are identical to each other. It is not so straightforward to represent the equivalent inference in the usual Horn clause logic.

```
> jack=father!Boy & Boy=Child &
> Who=father!Child & print(Who).
jack
success
```

It is possible to leave the role name unknown:

```
> jack= What ! B & Y= father! Z &
  B=betty & B=Z & What=father &
  print((Y,What,Z)).
(jack,father,betty)
success
```

We will abuse the equality symbol in the following sense. Let x and b be a complex indeterminate and a constant. The effect of the goal x=b is that the value of x becomes the constant b. In other words, x=b means that x is anchored to the value b. For any complex indeterminates x and y, the effect of the goal x=y is that x and y are unified with each other including roles. That is, x and y become identical to each other.

Of course, it is possible that two distinct indeterminates have the same value, as in the following:

```
> a!X=1 & a!Y=2 & X= Y.
fail
> a!X=1 & a!Y=2 & X=3 & Y=3.
success
```

3.5 Equality Theory

The following are the axioms for the unification in CIL:

(1) x=x.

(2) x=y => y=x.

(3) x=y & y=z => x=z.

(4) x1=y1 & ... & xn=yn => f(x1, ..., xn)=f(y1, ..., yn), for any functor symbol f.

(5) x=y => r!x=r!y for any role symbol r.

(6) z= (x with ..., a:=p, ... where c) & c => a!z=p.

(7) c => a ! (x with ..., a:=p, ... where c) = p.

(8) x=y & p1=q1 & ... & pn=qn & c & d => (x with u where c) = (y with v where d), provided { p1=q1, ..., pn=qn } is the set of equations p=q such that (a:=p) is in u and (a:=q) is in v for some role symbol a.

(9) x=y & c => (x with g where c)=y, provided y is not a complex indeterminate.

These equalities make the extended unification clear. However, there is a problem. The equality theory above does not fall into the framework of the foundation of logic programming with equality developed in Jaffar & Lassez & Maher 1983 and Emden &Lloyd 1984 because the complex indeterminate has a condition as a part. The Hilbert epsilon notation in Hilbert & Bernays 1939 [Maehara 1976] seems to give the right interpretation of the complex indeterminate. Further study is needed on this topic.

3.6 Type and Instance

The goal "create_type(x, g, c, t)" creates a type t such that the complex indeterminate "(x with g where c)" is an instance of the type t. The goal "type_of(x, t)" succeeds if x is an instance of type t.

We give a question-answering processing example to show the following three ideas from situation semantics:

(1) Discourse structure could be represented by a set of situations.

(2) Question sentence is a type with a context parameter.

(3) The answer is an instance of the type.

Example. Given the following sentences:

"The aircraft is flying in the sky." and
"The pacific ocean spreads below widely.".

Imagine the following "question-answering."

Question: Where is the aircraft flying ?
Answer: Over the pacific ocean.

The total process of this "story understanding" is described using "has," "create_type," and "type_of" predicates as follows:

```
has(S0, (pacific_ocean_loc,
        (spread_widely, pacific), 1)) &
has(S0, (the_sky_loc,
        (flying, air_craft), 1)) &
has(S0, (lu, (under,
              pacific_ocean_loc,
              the_sky), 1)) &
create_type(M,sit:=S,
      in(S,(L,(flying, _), 1))&
      in(S,(_,(under, M, L), 1)),
         T) &
type_of((X with sit:=S0), T).
```

CIL gets the "pacific_ocean_loc" for X as the answer by solving these goals.

4 MODEL THEORETIC SEMANTICS

We give a model theoretic semantics for a subset of the language. The subset is the pure Prolog plus complex indeterminate in our sense. Let us fix a set of the following symbols:

(1) atoms,

(2) functors,

(3) predicates,

(4) variables,

(5) roles,

constructors,

(7) delimiters: (,), ,(comma) , [,], with, where, :=, |, etc.

We suppose each symbol has only one of these types. "|," "&," and "<—" are constructor symbols.

A syntactical object constructed from these symbol is called a form. A form is called a ground term if and only if it has no symbols other than functor symbols or atomic symbols. The Herbrand universe is the set of all first order ground terms.

Definition. A form x is called a generalized term (term for short) if x is

(1) a variable,

(2) a complex indeterminate,

(3) a role indeterminate,

(4) an atomic symbol, or

(5) written h(x1, x2, ..., xn) for some functor h of arity n and for some generalized terms, x1, ..., xn.

Definition. A form is called a role indeterminate if it is written

(1) (r ! z) for some role symbol r and variable z, or

(2) (s ! u) for some role symbol s and role indeterminate u.

Definition. A complex indeterminate is written (x with g where c), where x, g, and c are a generalized term, a role connection, and a condition respectively.

Definition. A form x is called an indeterminate if x is a variable, role indeterminate or complex indeterminate.

Definition. A list of pairs, which is possibly empty, is called a role connection if it is written:

(1) (a1:=v1, ..., an:=vn) for some positive integer n, some list of distinct role symbols a1, ..., an, and some list of generalized terms v1, ..., vn, or

(2) ().

A finite set of atomic formulas is called a condition. We write a1& ...& an for the condition {a1, ..., an}. The empty condition is written {} or "true".

Definition. An ordered pair of an atomic formula and a condition is called a Horn clause. We write
a<—b1& ...&bn (n>0), or
c<—true

for the conditions (a, {b1, ..., bn}) and (c, {}), respectively.

Definition. A program is a finite set of Horn clauses.

Definition. A question is a set of atomic formulas.

Definition. An atomic formula is called ground if it has no indeterminate. Herbrand base is the set of all ground atomic formulas.

Given two sets X and Y, let [X->Y] denote the set of all partial finite functions from X into Y, excluding the empty function.

For two sets X and Y, XxY is the Cartesian product { <x, y> : x is in X and y is in Y }.

Let Fi be the set of functor symbols of arity i>=1. Let A be the set of atomic symbols. Let Pj be the set of j-ary predicates.

Definition. Three sets D, K, and P defined by the following domain equations are called the description universe, the extended Herbrand universe and the extended Herbrand base respectively:

$D = K + Kx[R->D]$. (disjoint sum)

$K = A + F1xD + F2xDxD + \ldots$. (disjoint sum)

$P = P0 + P1xD + P2xDxD + \ldots$. (disjoint sum)

Proposition. An element of the extended Herbrand universe can be written h(a1, ..., an) uniquely for some functor h of arity n>=0 and for some a1, ..., an in D.

An element of the extended Herbrand base can be written h(a1, ..., an) uniquely for some n-ary predicate symbol h and for some a1, ..., an in D. For any x in D, if x is not in K then x is written <k, g> uniquely for some x in K and g in [R->D].

Suppose that an element d in D is written <k, g> for some k in K and g in [R->D]. It is possible that d is the value of the function application g(r) for some role symbol r. That is, the domain D can represent labeled directed graphs which are possibly cyclic. However labels are restricted to finite ground terms. This condition is expressed as the existence of some projections from the extended Herbrand universe and the extended Herbrand base into the Herbrand universe and the Herbrand base respectively.

Definition. The function proj from the description universe D into the Herbrand universe is defined to be the solution of the following defining equations.

(1) proj(x) = x for any atom x.

(2) proj(h(a1, ..., an)) = h(proj(a1), ..., proj(an)) for any functor h of arity n and ai in D (1=<i=<n).

(3) proj(<k,g>) = proj(k) for any k in K and any g in [R->D].

Definition. The function proj is extended to the function from the extended Herbrand base to the Herbrand base naturally as follows.

(1) proj(p) = p for any predicate p of arity 0,

(2) proj(q(a1, ..., an))=q(proj(a1), ..., proj(an)) for any predicate symbol q of arity n and ai in the description universe (1=<i=<n).

Definition. A function f from a set of indeterminates to the Herbrand universe is called an anchor. A function f from a set of indeterminates into the extended Herbrand universe is called an extended anchor. An extended anchor is said to be total for a given form if the anchor is defined at any indeterminate appearing in the form.

We put an restriction on any anchor f.

(1) if f is defined at the indeterminate (x with g where c) then it is also defined at x.

(2) whenever f((x with g where c))= <k, s> and f(x) = <l, t> for some extended Herbrand terms k and l and some s and t in [R->D], then

2.1) proj(k)=proj(l),

2.2) the set {(a1,f(v1)), ...,(an, f(vn))} forms a function u, where the role connection g is of the form (a1:=v1, ..., an:=vn), and,

2.3) s is exactly the union of the functions t and u.

Definition. Given any form q and extended anchor f, the result of application of f to q, denoted by f(q), is defined as follows:

(1) f(q) : if q is in the domain of f;

(2) h(f(a1), f(a2), ..., f(an)) : if q = h(a1, a2, ..., an) for some functor or predicate h and generalized terms a1, a2, ..., an;

(3) undefined otherwise.

Definition. Given a subset M of the extended Herbrand base, an extended anchor f is said to satisfy M if 1) and 2) hold:

(1) For any complex indeterminate of the form (x with g where c) in the domain of f, f(c) is a subset of M, where f(c) = { f(p) | p is an atomic formula in the condition c}.

(2) For any role indeterminate of the form r!s in the domain of f, f(r!s)= g(r) provided f(s)=<k, g> for some k in the extended Herbrand universe and g in [R->D] with r in the domain of g.

Definition. A subset M of the extended Herbrand base is called stable with regard to the given Horn clause, h<—c, if 1) below implies 2) for any extended anchor f which satisfies M and total for the clause.

(1) f(c) is a subset of M.

(2) f(h) is in M.

Definition. A subset M of the extended Herbrand base is called an extended model of the given program P if M is stable with regard to any clause in P.

Proposition. The intersection of two given extented models is also an extended model.

Definition. A subset N of the Herbrand base is called a model of the given program P if for some extended model M of the program P, N is the image of M under the function proj, i.e., N = proj(M).

Definition. The semantics of a program is the minimum model of the program.

Definition. A condition c is said to be satisfiable if there is a total anchor f of c such that f satisfies the semantics M of the given program and f(c) is the subset of M.

Proposition. The semantics of the program exists for any program. Proof. Easy.

5 COMPUTATION MODEL

We formalize computation in CIL. Soundness and completeness of the computation is not obvious. They have not been proved yet.

Let T be the union of the set of all generalized terms and the set of all atomic formulas in the language. An element in T is called a form. T is called the form universe.

Definition. An equivalence relation E over the form universe is called congruent if the following hold:

(1) E(h(a1, ..., an), g(b1, ..., bm)) implies h=g and n=m and E(a1, b1) and ... and E(an, bn), provided h and g are functors or predicates,

(2) E((x with g where c), u) implies E(x, u),

(3) E((x with g where c), u) implies E(w, r!u), provided u is a variable or role indeterminate and the role connection g includes (r:=w), and

(4) E(r! (x with ..., r:=w, ... where c), w).

(5) E(s, t) implies E(r!s, r!t), where s and t are variables or role indeterminates and r is a role symbol.

Proposition. Given a set A of forms and a congruent relation E over the form universe, there is a congruence relation E' such that E' is coarser than E and A is a subset of one of the equivalence classes of E. Let S be the set of all these relations E'. Then S has the finest relation.

Definition. The finest congruence relation is called the finest extension of E by A and is written J(A, E).

Definition. For a congruent relation E and a generalized term t, let access(t, E) be the minimum set of all sets S satisfying the following conditions:

(1) S is a set of generalized terms.

(2) S includes t.

(3) If S includes x and E(x, y) then S includes y.

(4) If S includes (x with g where c) then S includes x.

(5) If S includes (x with r1:=v1, ..., rj:=vj, ..., rm:=vm where c) then S includes vj for any j (1=<j=<m).

Definition. Let E and t be as above. Let K(t,E) be the ,unction of all conditions c such that c is the condition at the condition part of some complex indeterminate in access(t, E).

Definition. An ordered triple (C, E, D) of a condition C and a congruence relation E and another condition is called a computation state (state for short).

A binary relation "->" between two states is defined as a computation rule as follows.

(C1, E1, D1) -> (C2, E2, D2) if and only if at least one of the following two conditions holds:

(1) For some equality x=y in C1,
E2 = J({x, y}, E1),
D2 = D1 & K(x, E) & K(y, E)), and
C2 = (C1 - {x=y}) & (D2 - D1).

(2) E2 = E1,
D2 = D1, and
for some atomic formula p in C1 and for some new instance, h<—b, of a Horn clause in the program:
C2 = (C1-{p}) & p=h & b.

Definition. A computation is a sequence of states s0, s1, ..., possibly infinite, satisfying the transition relation: s0->s1, s1->s2, s2->s3, ...

Definition. A state s is called a failure state if there is no state s' such that s -> s'.

Definition. A computation is fair if any atomic formula in the computation is "resolved upon" in finite steps. Notice that the atomic formula can appear in the condition part of some complex indeterminate.

Definition. A computation succeeds if it ends with a state of the form ({},...).

Now, we conjecture the following theorems by analogy with Jaffar & Lassez & Lloyd 1983:

Conjecture. (Soundness and completeness)

The following two conditions are equivalent:

(1) Some fair computation succeeds.

(2) The goal condition is satisfiable.

Conjecture. (Completeness of negation as failure rules)

The following two conditions about the ground goal are equivalent:

(1) All fair computations fail.

(2) The goal condition is not satisfiable.

6 CONCLUDING REMARKS

Complex indeterminates give the logic programming language a great power for syntax, semantics, and pragmatics processing of natural language. We think CIL gives not only a grammar formalism like PATR-II [Pereira & Shieber 1984, Shieber 1984], for instance, but also a general power to describe discourse models [Brady & Berwick 1982].

Suzuki 1984 suggested a relation between types and frames. Our extended unification realizes the relation. The following features of frames are realized in our system of complex indeterminates: "property inheritance," "if-filled-demon" and "if-needed-demon." "If-removed-demon," however, is an impossible feature in our language.

The target machine on which we plan to implement CIL efficiently is the PSI machine [Uchida & Yokoi 1984], which is now available to us. The PSI has machine primitives for bind hook control. Merging two role connections is one of the key points of the efficient implementation of CIL. When all of role names are known to the compiler, the following are possible:

(1) Each role connection is represented by a vector with the fixed size n where n is the number of all the roles.

(2) Each role is assigned unique index in the vector.

This reduces the extended unification to usual vector unification and also makes roles to be accessible directly. For example, the unification:

unify((x with a:=y, b:=z), (u with b:=v, a:= w))

is equivalent to

unify_vector(<x, y, z>, <u, w, v>).

ACKNOWLEDGEMENTS

I would like to thank Mr. T. Kanamori of Mitsubishi Corporation for his comments to Section 4 and 5 on the foundation of CIL. Mr. H. Yasukawa of ICOT greatly has extended the example program in Section 2. Also, I would like to thank T. Yokoi and K. Furukawa, the Heads of the Second and First Research Laboratories at ICOT, for their useful comments and encouragement.

REFERENCES

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

[Barwise 1975] J. Barwise: Admissible Sets and Structures, Springer Verlag, 1975.

[Barwise 1984] J. Barwise: Lectures on Situation Semantics, Winter Quarter at CSLI, 1984.

[Bowen 1982] D. Bowen (ed.), L. M. Byrd, F.C.N. Pereira, L.M. Pereira & D.H.D. Warren : DECsystem-10 Prolog User's Manual, Department of Artificial Intelligence, University of Edinburgh, 1982.

[Brady & Berwick 1982] M. Brady & R. Berwick (eds.): Computational Models of Discourse, MIT Press, 1982.

[Bresnan 1983] J. Bresnan(eds.): Mental Representation of Grammatical Relations, The MIT Press, 1983.

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

[Emden & LLoyd 1984] M. H. van Emden & J. W. Lloyd : A Logical Reconstruction of Prolog II, the Journal of Logic Programming, Vol. 1, No. 2, Aug. 1984.

[Furukawa & Yokoi 1984] K. Furukawa & T. Yokoi: Basic Software System, in Proceedings of the International Conference on Fifth Generation Computer Systems 1984, edited by ICOT, pp.47-48, 1984.

[Gazdar & Klein & Pullum & Sag 1985] G. Gazdar, E. Klein, G.K. Pullum & I.A. Sag: Generalized Phrase Structure Grammar, (to be published), 1985.

[Hilbert & Bernays] D. Hilbert & P. Bernays: Grundlagen der Mathematik II, Springer, 1939.

[Jaffar & Lassez & Lloyd 1983] J. Jaffar, J-L. Lassez, and J. W. Lloyd: Completeness of the Negation-as-Failure Rule, Proceeding 8th IJCAI, Karlsruhe, Aug 1983, pp. 500-506.

[Jaffar Lassez & Maher 1983] 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.

[Lloyd 1984] J. W. Lloyd: Foundations of Logic Programming, Technical Report 82/7 (revised March 1984), Department of Computer Science, University of Melbourne.

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

[Mukai 1985b] K. Mukai: Feature System and Unification, in "Some Aspects of Generalized Phrase Structure Grammar" by ICOT Working Group 3.4, ICOT Technical Memorandum No. TM-103, March, 1985.

[Maehara 1976] S. Maehara: Suri-Ronri-Gaku (Mathematical Logic), Baifu-kan, 1976, (in Japanese).

[Pereira & Shieber 1984] F. C. N. Pereira & 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 1980] F. C. N. Pereira & 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.

[Shieber 1984] S. M. Shieber: The Design of a Computer Language for Linguistic Information, in the Proceedings of the Tenth International Conference on Computational Linguistics, Stanford University, California, 1984.