ICOT Technical Memorandom: TM-0984

TM-0984

Parsing as Constraint Transformation -an extension of cu-Prolog

by
H. Tsuda & K. Hasida

December, 1990

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Parsing as Constraint Transformation — an extension of cu-Prolog

TSUDA, Hiroshi HASIDA, Kôiti
Institute for New Generation Computer Technology (ICOT)
Mita Kokusai Bldg. 21F, 1-4-28 Mita, Minato-ku, Tokyo 108, JAPAN
Tel: +81-3-456-3069
E-mail: tsuda@icot.or.jp, hasida@icot.or.jp

Abstract

Natural language processing involves a very complex flow of information which cannot be stipulated in terms of procedural programs. This necessitates some sort of constraint programming.

In order to embody this picture, we have developed a constraint logic programming (CLP) language **cu-Prolog**. Unlike most CLP systems, cu-Prolog allows user defined and dynamically defined predicates to represent symbolic/combinatorial constraints. The constraint satisfaction in cu-Prolog is regarded as a sort of unfold/fold transformation of logic programming. This paper extends cu-Prolog and presents a constraint-based approach to sentence comprehension.

First, local constraints on the feature structure of categories of the unification-based grammar such as HPSG and JPSG are treated as constraints of cu-Prolog. In this respect, mainly feature constraint is processed not as procedures but as constraints.

Second, the constraint transformation method is extended here, by incorporating global variables and a tabulation technique, so that parsing of a context-free language is subsumed in constraint transformation. This process naturally corresponds to chart parsing.

Finally, these two aspects naturally fit each other in our constraint-based approach, unlike most NLP systems, which have distinct procedure modules to deal with the above two aspects of parsing and make them interact. Some examples are provided which demonstrate homogeneous treatment of feature structure and phrase structure.

1 Introduction

One serious difficulty in artificial intelligence in general and natural language processing(NLP) in particular is that it is practically impossible to stipulate which type of information (such as syntactic, semantic, and pragmatic) to process in what order, given the innumerable, unpredictable pieces of information coming into play. It does not appear at all useful to tailor a procedure to deal with only some types of information. Thus, a parser to handle just syntactic information does not contribute very much to the total design of NLP systems.

A promising approach to NLP, we believe, is not to have several procedures each dealing with one particular type of information, but to have just one procedure to deal with every type of information.[1] This means that we should divide the entire task of NLP into modules of constraints rather than modules of procedures as has been done traditionally. We consider that what the common procedure does is to transform constraints, so that sentence parsing, generation, semantic and pragmatic inferences, and so on should all be implemented in terms of constraint transformation.

To make this a reality, we have developed a constraint logic programming (CLP) language **cu-Prolog**. Unlike most CLP systems which deal with numerical or finite constraints on the basis of a fixed set of reserved predicates, cu-Prolog allows user-defined and dynamically defined predicates to represent symbolic/combinatorial constraints. The constraint satisfaction in cu-Prolog is regarded as a sort of unfold/fold transformation of logic programming. A program clause of cu-Prolog is called CAHC (Constraint Added Horn Clause), which is a normal Horn clause followed by constraints.

Section 2 outlines cu-Prolog and treats constraint in terms of feature structure of the unification-based grammar. Section 3 applies constraint transformation technique to simple CFG parsing and shows the chart parsing algorithm is naturally derived. Section 4 applies constraint transformation to CFG with feature structure.

2 cu-Prolog

In [14], we introduced a symbolic constraint logic programming (CLP) language **cu-Prolog** [2] and indicated a JPSG(Japanese Phrase Structure Grammar[2]) parser as its application. By treat-

[1] Thus we consider that GPS was basically on the right track. Its alleged failure was due to the immaturity of programming technologies.

[2] cu is an abbreviation of *constraint unification*[3]

ing grammatical principles or ambiguity of polysemic words or homonyms as constraints, syntactic and semantic ambiguity is integrated in the constraint transformation process. Therefore a special technique for storing ambiguity is not necessary.

2.1 CAHC

cu-Prolog handles a Constraint Added Horn Clause (CAHC), which is a normal Horn Clause followed by a semicolon and constraints (Body or Constraint can be eliminated):

$$\overbrace{H}^{Head} :- \overbrace{B_1, B_2, \ldots, B_n}^{Body} ; \overbrace{C_1, C_2, \ldots, C_m}^{Constraint} .$$

Note that above CAHC is equivalent to the following normal Horn clause in terms of declarative semantics.

$$H :- B_1, B_2, \ldots, B_n, C_1, C_2, \ldots, C_m.$$

The Prolog part (before the semicolon) of CAHC is processed procedurally like in Prolog, and the constraint part is dynamically transformed with the unfold/fold transformation [10] during the execution of the former part. The following is the inference rule of cu-Prolog

$$\frac{A, \mathbf{K}; \mathbf{C}. \ , \ A' : -\mathbf{L}; \mathbf{D}. \qquad \theta = mgu(A, A'), \mathbf{C}' = mf(\mathbf{C}\theta, \mathbf{D}\theta)}{\mathbf{L}\theta, \mathbf{K}\theta; \mathbf{C}'}$$

A and A' are atomic formulas. $\mathbf{K}$, $\mathbf{L}$, $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{C}'$ are sequences of atomic formulas. $mgu(A, A')$ is the most general unifier between A and A'.

$mf(\mathbf{C})$ is a modular constraint that is equivalent to $\mathbf{C}$. If $\mathbf{C}$ is inconsistent, the application of the above inference rule fails because $mf(\mathbf{C})$ does not exist. When $\mathbf{C}$ is divided into equivalent classes in terms of the variables as

$$\mathbf{C} = \mathbf{C}_1 + \cdots + \mathbf{C}_n$$

then,

$$mf(\mathbf{C}) = mf(\mathbf{C}_1), \ldots, mf(\mathbf{C}_n).$$

For example,

mf(**memb(X, [a, b, c]), memb(X, [b, c, d]), app(U, V)**)

returns a new constraint `c0(X),app(U,V)`, where the definition of `c0` is

```
c0(b).      c0(c).
```

and (Let **memb** be **member** predicate.)

mf(**memb(X, [a, b, c]), memb(X, [k, l, m])**)

is not defined.

2.2 JPSG parser in cu-Prolog

In cu-Prolog, the unification-based grammar such as HPSG(Head-driven Phrase Structure Grammar[8]) or JPSG is naturally implemented by adding constraints to the program clauses representing lexical entries or phrase structure grammar. Here, we show two examples of JPSG representation in CAHC.

The first is an example of the lexicon of a Japanese polysemic noun "hasi" that means bridge, chopsticks, or edge. However, such ambiguity is unified by attaching constraints to one lexical entry and the ambiguity is reduced during the parsing process.

```
lexicon(hasi,[...sem(SIT,SEM)]);hasi_sem(SIT,SEM).
```

and predicate `hasi_sem` is defined as follows.

```
hasi_sem(structure,bridge).
hasi_sem(tool,chopsticks).
hasi_sem(place,edge).
```

In normal Prolog, such lexicon is divided into separate program clauses or the constraint `hasi_sem(SIT,SEM)` is processed as a procedure. In any cases, the process may be ineffective.

In the second example, various feature principles of unification-based grammar are embedded in a phrase structure rule as constraints. The following clause shows the foot feature principle of JPSG: the foot feature value of the mother unifies with the union of those of her daughters.

```
psr([foot(MS)],[foot(LDS)],[foot(RDS)]);union(LDS,RDS,MS).
```

In Prolog, the processing order of literals is inevitably fixed in advance and hence the process may be ineffective.

Figure 1 shows an example of the JPSG parser processing an ambiguous sentence. [3]

3 CFG parsing as constraint transformation

The JPSG parser noted in the previous section 2, however cannot handle ambiguity on syntactic parse trees [4] because the parsing algorithm is written procedurally in the Prolog part of CAHC. This section introduces the extension of the constraint transformation of cu-Prolog, called Dependency Propagation (DP)[5] which regards constraint transformation as computation. As an example of DP, a simple CFG is parsed only with constraint transformation.

[3]cu-Prolog is implemented in the C language on UNIX 4.2/3BSD[15]. This example is on SYMMETRY.

[4]For example, the ambiguity in "I saw a man with a telescope".

```
_:-p([ken,ga,ai,suru]).

v[Form_675, AJN{Adj_677}, SC{SubCat_679}]:SEM_681---[suff_p]
 |
 |--v[vs2, SC{p[wo]}]:[love,ken,Obj0_415]---[subcat_p]
 |    |
 |    |--p[ga]:ken---[adjacent_p]
 |    |    |
 |    |    |--n[n]:ken---[ken]
 |    |    |
 |    |    |__p[ga, AJA{n[n]}]:ken---[ga]
 |    |
 |    |__v[vs2, SC{p[ga], p[wo]}]:[love,ken,Obj0_415]---[ai]
 |
 |__v[Form_675, AJA{v[vs2,SC{p[wo]}]}, AJN{Adj_677}, SC{SubCat_679}]:SEM_681---[suru]

cat      cat(v, Form_675, [], Adj_677, SubCat_679, SEM_681)
cond      c7(Form_675, SubCat_679, Obj0_415, Adj_677, SEM_681)
True.
CPU time = 0.050 sec

_:-c7(F,SC,_,A,SEM).
  F = syusi  SC = [cat(p, wo, [], [], [], Obj00_30)]  A = []  SEM = [love,ken,Obj00_30];
  F = rentai  SC = []  A = [cat(n, n, [], [], [], inst(Obj00_38, Type3_36))]
  SEM = inst(Obj00_38, [and,Type3_36,[love,ken,Obj00_38]])
no.
CPU time = 0.017 sec
```

The first line is a user's input. "Ken ga ai suru" has two meanings: "Ken loves (someone)" or "(someone) whom Ken loves".
The parser draws a parse tree and returns the category and constraint of the top node. In this example, the ambiguity of the sentence is shown in the two solutions of the constraint `c7(F,SC,_,A,SEM)`.

Figure 1: An example of an ambiguous sentence

For the sake of expository simplification, here we restrict ourselves to Horn clauses, although DP is not actually so limited.

3.1 Definitions

As a trigger of constraint transformation, DP considers *dependency* among literals. A variable occurring in more than two distinct non-vacuous places in a clause has *dependency*. When an argument place of a predicate is a variable in all of its definition clauses, the argument place is called a *vacuous argument place*. For example, the first argument place of **member** defined below is vacuous.

(1) a. `member(E,[E|_]).`
b. `member(E,[_|S]) :- member(E,S).`

We put # after the vacuous variable that has dependency with other literals as follows:

(2) `member(X#,Y),c0(Y,Z)`

In the above, though variable Y occurs in two places, there is no dependency because the first Y is vacuous.

DP also introduces a *transclausal variable* that corresponds to a global variable of Pascal or C Programming Language and is treated as if it were a constant in some context. We put * in front of a transclausal variable as follows.

(3)
`:-vp(*V0,B),*V0=[see|*V1],*v1=[a,man|*V2].`

Constraint transformation is executed so as to eliminate dependency of goal clauses or a body of program clauses, therefore is more general than Earley deduction [7] which executes the body of each clause in the fixed left-to-right order.

3.2 Penetration

To process vacuous variables and transclausal variables, we introduce *penetration* operation in addition to the unfold/fold transformation.

Downward penetration is to replace a literal that contains transclausal variables with a new literal that contains no transclausal variable defining a new predicate with unfold/fold transformation. For example,

(4) a. `:-p(*V0,B),*V0=[a|*V1].`
b. `p([a|X],X).`
c. `p(X,Z):-p(X,Y),p(Y,Z).`

is transformed to (5). p0 is a new predicate and p0(B) is equivalent to p(*V0,B). The dependency concerning *V0 in the first goal of (4) is dissolved.

```
(5)   a. :-p0(B),*V0=[a|*V1].
      b. p0(*V1).
      c. p0(Z):-p(*V1,Y),p(Y,Z).
```

Upward penetration is to reduce a unit clause containing transclausal variables so as to change some argument places to begin vacuous. For example, let (6) be all the clauses that contain p0. The argument place of p0 is not vacuous because the transclausal variable *V1 in the first clause is considered as a constant.

```
(6)   a. p0(*V1).
      b. p0(Z):-p0(Y),p(Y,Z).
```

By replacing p0(*V1) with a new predicate p1 and (6) is transformed to (7).

```
(7)   a. p1.
      b. p0(Z):-p1,p(*V1,Z).
      c. p0(Z):-p0(Y#),p(Y,Z).
```

Then the argument place of p0 becomes vacuous.

3.3 Parsing an ambiguous CFG

Let us consider the following simple ambiguous context-free grammar. This corresponds to the syntactic ambiguity of "I see a man with a telescope."

(8) $VP \rightarrow V, NP$
$VP \rightarrow VP, PP$
$NP \rightarrow NP, PP$
$V \rightarrow$ *see*
$NP \rightarrow$ *a man*
$PP \rightarrow$ *with a telescope*

Parsing program in terms of this grammar can be formulated as follows.

```
(C0) *V0=[see|*V1],
     *V1=[a,man|*V2],
     *V2=[with,a,telescope|*V3],
     *V3=NIL
(C1) :-vp(*V0,B).
(C2) v([see|W],W).
(C3) np([a,man|W],W).
(C4) pp([with,a,telescope|W],W).
(C5) vp(X,Z):-v(X,Y#),np(Y,Z).
(C6) vp(X,Z):-vp(X,Y#),pp(Y,Z).
(C7) np(X,Z):-np(X,Y#),pp(Y,Z).
```

There is only one dependency to be eliminated: *V0 in (C1). Here, we introduce a new predicate vp0 as vp0(V) = vp(*V0,V). By downward penetration, (C1) is replaced with the following clauses.

```
(C1') :-vp0(B).
(C8) vp0(V):-vp(*V0,Y),pp(Y,V).
(C9) vp0(V):-v(*V0,Y),np(Y,V).
```

By folding the first literal in the body of (C8), we have

```
(C8') vp0(V):-vp0(Y),pp(Y,V).
```

Next, we process the dependency concerning *V0 in the body of (C9). Let v0(V)=v(*V0,Y) and by downward penetration, we get

```
(C9') vp0(V):-v0(Y),np(Y,V).
(C10) v0(*V1).
```

As v0 has only one definition clause (C10), then it is reduced.

```
(C9') vp0(V):-np(*V1,V).
```

Let np1(V)=np(*V1,V) and by downward penetration,

```
(C9") vp0(V):-np1(V).
(C11) np1(*V2).
(C12) np1(V):-np(*V1,Y),pp(Y,V).
```

Fold the first literal of the body of (C12), then

```
(C12') np1(V):-np1(Y),pp(Y,V).
```

The argument place of np1 is not vacuous, then we apply downward penetration to np1. Here, np12=np1(*V2).

```
(C11') np12.
(C13) np1(V):-np12,pp(*V2,V).
(C12') np1(V):-np1(Y),pp(Y,V).
```

We have to consider the dependency of the second literal of the body of (C13). Here, let pp2(V)=pp(*V2,V).

```
(C13') np1(V):-np12,pp2(V).
(C14) pp2(*V3).
```

pp2 has only one definition, then is reduced.

```
(C13") np1(*V3).
```

The remaining definition of np0 is (C11') and (C13"). Then (C9") is reduced.

```
(C9-1) vp0(*V2).
(C9-2) vp0(*V3).
```

Apply upward penetration to (C8'), (C9-1), and (C9-2) introducing a new predicate as vp02=vp0(*V2) and vp03=vp0(*V3), then the definition of vp0 becomes as follows:

```
vp02.
vp03.
vp0(V):-vp02,pp(*V2,V).
vp0(V):-vp03,pp(*V3,V).
vp0(V):-vp0(Y),pp(Y,V).
```

Finally, it is transformed to

```
vp02.
vp03.
vp03.
vp0(V):-vp0(Y),pp(Y,V).
```

The two occurrences of **vp03** correspond to the two meanings of "I see a man with a telescope".

3.4 Complexity

This subsection reviews the complexity of parsing on constraint transformation. (9) is a simple CFG example mentioned in [5].

(9) $P \rightarrow a$
$P \rightarrow PP$

Parsing the string "aa ...a"(length is n) under this grammar may be formulated in terms of a set of constraints (10).

(10) $\mathtt{:-\ p(A^0,B),\ A^0=[a|A^1],\ \cdots,\ A^{n-1}=[a|A^n].}$
$\mathtt{p([a|X],X).}$
$\mathtt{p(X,Z)\ :-\ p(X,Y),\ p(Y,Z).}$

After some transformation steps, (11) is finally obtained.

(11) $\mathtt{:-\ q,\ A^0=[a|A^1],\ \cdots,\ A^{n-1}=[a|A^n].}$
$\mathtt{q\ :-\ p_0(B^0).}$
$\mathtt{q\ :-\ p_{0,i},\ B^0=A^i.}\ (0 < i \leq n)$
$\mathtt{p_i(Z)\ :-\ p_{i,j},\ p_j(Z).}\ (0 \leq i < j < n)$
$\mathtt{p_i(Z)\ :-\ p_i(Y),\ p(Y,Z).}\ (0 \leq i < n)$
$\mathtt{p_{i,i+1}.}\ (0 \leq i < n)$
$\mathtt{p_{i,k}\ :-\ p_{i,j},\ p_{j,k}.}\ (0 \leq i < j < k < n)$

Part of (11) amounts to a well-formed substring table, as in CYK algorithm, Earley's algorithm [1], chart parser, tabulation technique [11], and so on. For instance, the existence of clause $\mathtt{p_{i,k}:-p_{i,j},\ p_{j,k}}.$ means that the part of the given string from position i to position k has been parsed as having category P and is subdivided at position j into two parts, each having category P. Note that the computational complexity of the above process is $O(n^3)$ in terms of both space and time.

Moreover, the space complexity is reduced to $O(n^2)$ if we delete the literals irrelevant to instantiation of variables, which preserves the semantics of the constraints in the case of Horn programs. That is, the resulting structure would be:

(12) $\mathtt{:-\ q,\ A^0=[a|A^1],\ \cdots,\ A^{n-1}=[a|A^n].}$
$\mathtt{q\ :-\ p_0(B^0).}$
$\mathtt{q\ :-\ B^0=A^i.}\ (0 < i \leq n)$
$\mathtt{p_i(Z)\ :-\ p_j(Z).}\ (0 \leq i < j < n)$
$\mathtt{p_i(Z)\ :-\ p_i(Y),\ p(Y,Z).}\ (0 \leq i < n)$
$\mathtt{p_{i,j}.}\ (0 \leq i < j < n)$

The process illustrated above corresponds best to Earley's algorithm. Our procedure may be generalized to employ bottom-up control, so that the resulting process should be regarded as chart parsing, left-corner parsing, and so on.

4 Parsing CFG with feature structure as constraint transformation

This section tries to handle various types of constraints such as the constraints on feature structure or on phrase structures mentioned in the previous two sections. We have to investigate some heuristics to determine which constraint is processed earlier than the others.

4.1 Heuristics

In the following discussion, we consider only Horn clause constraint, and two types of linguistic constraint: constraint on feature structure and on phrase structure.

Following is a heuristic used in this paper. This heuristic guarantees that the computation takes place in such a way that it may be looked upon as phrase-structure computation annotated with constraints on feature structures as in the approach of Section 2, just as people would like to regard parsing processes to be.

- A variable occurring in both types of constraint does not have dependency.
- Dependencies concerning feature structures should be eliminated earlier than those concerning phrase structures.
- Literals concerning phrase structures should be unfolded first when you attempt to eliminate dependency between literals about phrase structures and literals about feature structures.

4.2 Example

The program below is another formulation of the simple CFG (8). We consider only one feature called **pos**

[5] that takes a part of speech such as np,vp,pp,and so on. pos feature follows the constraint that corresponds to the first three rules of (8):

> The combination of the value of **pos** feature of mother, left daughter, and right daughter category is (vp,n,np),(np,np,pp), or (vp,vp,pp).

In the following, constraints concerning phrase structure (predicate cst) and those concerning feature structure (predicate p) are separated by '|'.

```
(P0)  *V0=[see|*V1],
      *V1=[a,man|*V2],
      *V2=[with,a,telescope|*V3],
      *V3=NIL
(P1)  :-p(*V0,B,C).
(P2)  p(X,Z,Cat):-p(X,Y#,LC),p(Y,Z,RC) |
                   cst(LC,RC,C).
(P3)  p([see|W],W,v).
(P4)  p([a,man|W],W,np).
(P5)  p([with,a,telescope|W],W,pp).
(P6)  cst(v,np,vp).
(P7)  cst(np,pp,np).
(P8)  cst(vp,pp,vp).
```

The dependency to be processed is in terms of *V0 in (P1) because LC and RC in (P2) do not have dependencies. Process downward penetration. Here, p0(B,C)=p(*V0,B,C).

```
(P1') :-p0(B,C).
(P9)  p0(*V1,v).
(P10) p0(B,C):-p(*V0,Y#,LC),p(Y,B,RC) |
                cst(LC,RC,C).
```

Fold the first literal of (P10).

```
(P10') p0(B,Cat):-p0(Y,LC),p(Y,B,RC) |
                      cst(LC,RC,Cat).
```

Upward penetration. Let p01=p0(*V1,v).

```
(P9) p01.
(P11) p0(B,Cat):-p01,p(*V1,B,RC) |
                     cst(v,RC,Cat).
(P10') p0(B,Cat):-p0(Y#,LC),p(Y,B,RC) |
                     cst(LC,RC,Cat).
```

Unfold the feature constraint of (P11).

```
(P11') p0(B,vp):-p01,p(*V1,B,np).
```

Downward penetration. Here, p1(B)=p(*V1,B,np).

```
(P11") p0(B,vp):-p01,p1(B).
(P12) p1(*V2).
(P13) p1(Z):-p(*V1,Y,np),p(Y,Z,RC) |
               cst(np,RC,np).
```

Unfold the feature constraint of (P13) and fold p1.

```
(P13') p1(Z):-p1(Y),p(Y,Z,pp).
```

[5]It is different from pos feature of JPSG.

Upward penetration. Let p12=p1(*V2).

```
(P12') p12.
(P14) p1(Z):-p12,p(*V2,Z,pp).
(P13') p1(Z):-p1(Y#),p(Y,Z,pp).
(P15) p0(*V2,vp):-p01,p12.
```

Downward penetration p2(B)=p(*V2,Z,pp).

```
(P14') p1(Z):-p12,p2(Z).
(P16) p2(*V3).
(P17) p2(Z):-p(*V2,Y,LC),p(Y,Z,RC) |
               cst(LC,RC,pp).
```

Unfold the feature constraint of (P17), however it fails because there is no clause matching cst(LC,RC,pp). p2 is reduced.

```
(P14") p1(*V3).
```

By upward penetration introducing p13=p1(*V3), (P11') becomes

```
(P11-1) p0(*V3,vp).
```

and is corresponds to "see (a man with a telescope)." From (P15) let p02=p0(*V2,vp) and apply upward penetration.

```
(P10") p0(B,Cat):-p(*V2,B,RC) | cst(vp,RC,Cat).
```

Unfold the feature constraint of (P10").

```
(P10-3) p0(B,vp):-p(*V2,B,pp).
```

It is finally becomes

```
(P10-4) p0(*V3,vp).
```

and corresponds to "(saw a man) with a telescope."

5 Concluding Remarks

In this paper we have shown that various parsing techniques are subsumed in a general procedure of constraint transformation. Thus our conclusion is that no parser at all is needed in natural language processing. It is both desirable, as is discussed first in the paper, and possible, as we have so far demonstrated, for an NLP system to have no particular module for parsing sentences, just as a car has no particular part for driving towards the east or turning to the left.

For further research, an essential aspect of extended LR parsing methods such as the Tomita Parser [13] and the YAGLR method [6, 12] is automatically implemented as a corollary of our transformation procedure.

Constraint hierarchy mentioned in the last section is an important concept in representing and processing natural language grammar. There are many types of constraints to be evaluated (syntactic, semantic, pragmatic, and so on) and there seems to be a complex hierarchy between them. For example, semantic constraints sometimes work to reduce syntactic ambiguity, but sometimes overcome syntactic ambiguities in the processing of an ungrammatical sentence. We are now considering a hierarchical constraint logic programming language (HCLP) as an extended version of cu-Prolog to deal with such problems.

Equipped with an adequate control mechanism, our approach will capture sentence generation as well. In this connection, Shieber [9], among others, has also proposed a computational architecture by which to unify sentence parsing and generation, but his method is primarily specific to phrase-structure synthesis. A significant merit of our approach is that it is not in any way restricted to parsing or generation in context-free languages. Also, no additional mechanism is required to extend the underlying grammatical formalism so that grammatical categories may be complex feature bundles, as is the case with GPSG, LFG, HPSG, and so on, rather than monadic symbols. In such a more general case, the standard parsing algorithms are regarded as partially approximating DP in sentence comprehension.

References

[1] Earley, J. (1970) 'An Efficient Context-Free Parsing Algorithm', *Communications of ACM*, Vol. 13, pp. 94-102.

[2] Gunji. T. (1986) 'Japanese Phrase Structure Grammar', *Reidel,Dordrecht*,1986.

[3] Hasida, K. (1986) 'Conditioned Unification for Natural Language Processing', *Proceedings of the 11th COLING*.

[4] Hasida, K. and Ishizaki, S. (1987) 'Dependency Propagation: A Unified Theory of Sentence Comprehension and Generation', *Proceedings of the 10th IJCAI*, pp. 664-670.

[5] Hasida, K. (1990) 'Sentence Processing as Constraint Transformation', *Proceedings of ECAI'90*.

[6] Numazaki, H. and Tanaka, H. (1990) 'An Efficient Parallel Generalized LR Parsing based on Logic Programming', *Proceedings of the Logic Programming Conference'90*, pp. 191 198.

[7] Pereira, F.C.N. and Warren, D.H.D. (1983) 'Parsing as Deduction', *Proceedings of ACL'83*, pp. 137-144.

[8] Pollard, C. and Sag, I.A. (1987) *Information-Based Syntax and Semantics, Volume 1*, CSLI Lecture Notes No. 13.

[9] Shieber, S.M. (1988) 'A Uniform Architecture for Parsing and Generation', *Proceedings of the 12th COLING*, pp. 614-619.

[10] Tamaki, H. and Sato, T. (1983) 'Unfold/Fold Transformation of Logic Programs', *Proceedings of the Second International Conference on Logic Programming*, pp. 127-138.

[11] Tamaki, H. and Sato, T. (1984) 'OLD Resolution with Tabulation', *Proceedings of the Third International Conference on Logic Programming*, pp. 84-98.

[12] Tanaka, H. (1990) 'YAGLR method: Yet Another Generalized LR Parsing',*unpublished*.

[13] Tomita, M. (1987) 'An Efficient Augmented-Context-Free Parsing Algorithm', *Computational Linguistics*, Vol. 13, No. 1-2, pp. 31-46.

[14] Tsuda, H., Hasida, K., and Sirai, H. (1989) 'JPSG Parser on Constraint Logic Programming', *Proceedings of the European Chapter of ACL'89*, pp. 95-102.

[15] Tsuda, H., Hasida, K., Yasukawa, H. and Sirai, H. (1990) 'cu-Prolog V2 system', *ICOT TM-952*.