PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT.

cu-Prolog for Constraint-Based Grammar

Hiroshi TSUDA
Institute for New Generation Computer Technology (ICOT)
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
E-mail : tsuda@icot.or.jp

Abstract

cu-Prolog is a constraint logic programming (CLP) language appropriate for natural language processing such as a Japanese parser based on JPSG. Compared to other CLP languages, cu-Prolog has several unique features. Most CLP languages take algebraic equations or inequations as constraints. cu-Prolog, on the other hand, takes the Prolog atomic formulas of user-defined predicates. cu-Prolog, thus, can describe symbolic and combinatorial constraints that are required for constraint-based natural language grammar description. As a constraint solver, cu-Prolog uses unfold/fold transformation dynamically with some heuristics.

JPSG (Japanese Phrase Structure Grammar) is a constraint-based and unification-based Japanese grammar formalism beging developed by the PSG-working group at ICOT. Like HPSG (Head-driven Phrase Structure Grammar), JPSG is a phrase structure whose nodes are feature structures. Its grammar description is mainly formalized by local constraints in phrase structures.

This paper outlines cu-Prolog and its application to the disjunctive feature structure and JPSG parser.

1 Introduction

Two aspects are considered to classify contemporary natural language grammatical theories[Carpenter *et al.* 91]. Firstly, They must be classified according to whether they have transformation operations among different structure levels.

One current version of *transformational grammar* is GB (Government and Binding) theory[Chomsky 81]. So called *unification-based grammars*[Shieber 86], such as GPSG (Generalized Phrase Structure Grammar), LFG (Lexical Functional Grammar), HPSG (Head-driven Phrase Structure Grammar)[Pollard and Sag 87], and JPSG (Japanese Phrase Structure Grammar)[Gunji 86] are categorized as *non-transformational grammars*. Unification-based grammar is a phrase structure grammar whose nodes are feature structures. It uses unification as its basic operation. In this respect, it is congenial to logic programming.

Secondly, classification must be made as to whether a language's grammar description is *rule-based* or *constraint-based*[1]. GPSG and LFG fall into the former category. The latter includes GB theory, HPSG, and JPSG. From the viewpoint of procedural computation, rule-based approaches are better. However, by constraint-based approaches, more general and richer grammar formalisms are possible because morphology, syntax, semantics, and pragmatics are all uniformly treated as constraints. Also, the most important feature of constraints, the declarative grammar description, allows various information flows during processing.

Consider the programming languages used to implement these grammatical theories. For rule-based grammars, many approaches have been attempted, such as FUG[Kay 85] and PATR-II[Shieber 86]. As yet, however, no leading work has been done on constraint-based grammars.

Our constraint logic programming language *cu-Prolog* [Tsuda *et al.* 89b, Tsuda *et al.* 89a] aims to provide an implementation framework for constraint-based grammars. Unlike most CLP languages, cu-Prolog takes the Prolog atomic formulas of user-defined predicates as constraints.

cu-Prolog originated from the technique of *constrained unification* (or *conditioned unification* [Hasida and Sirai 86]) – a unification between two constrained Prolog patterns. The basic component of cu-Prolog is a *Constrained Horn Clause (CHC)* that adds constraints in terms of user-defined Prolog predicates to Horn clauses. Their domain is suitable for symbolic and combinatorial linguistic constraints. The constraint solver of cu-Prolog uses the unfold/fold [Tamaki and Sato 83] transformation dynamically with certain heuristics.

This paper illustrates

- the outline of cu-Prolog,
- treatment of disjunctive feature structures with PST(Partially Specified Term)[Mukai 88] in cu-Prolog, and

[1] Constraint-based approaches are also called *information-based* or *principle-based* approaches.

- the JPSG parser as its most successful application.

2 Linguistic Constructions

As an introduction, this section explains the various types of linguistic constraints in constraint-based grammar formalisms.

2.1 Disjunctive Feature Structure

Unification-based grammars utilize *feature structures* as basic information structures. A feature structure consists of a set of pairs of labels and their values. In (1), *pos* and *sc* are called *features* and their values are n and a singleton set $< [pos = p] >$.

$$\left[\begin{array}{l} pos = n \\ sc = \left\langle \left[\ pos = p\ \right] \right\rangle \end{array}\right] \tag{1}$$

Morphological, syntactic, semantic, and pragmatic information are all uniformly stored in a feature structure.

Moreover, natural language descriptions essentially require some framework to handle ambiguities such as polysemic words, homonyms, and so on. *Disjunctive feature structures* are widely used to handle disjunctions in feature structures[Kay 85]. Disjunctive feature structures consist of the following two types.

Value disjunction A value disjunction specifies the alternative values of a single feature. The following example states that the value of the *pos* feature is n or v, and the value of the *sc* feature is <> (empty set) or $< [pos = p] >$.

$$\left[\begin{array}{l} pos = \{n, v\} \\ sc = \left\{ \begin{array}{l} \langle\rangle \\ \left\langle \left[\ pos = p\ \right] \right\rangle \end{array} \right\} \end{array}\right] \tag{2}$$

General disjunction A general disjunction specifies alternative groups of multiple features. In the following structure, $sem = love(X, Y)$ is common, and the rest is ambiguous.

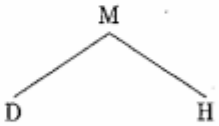
$$\left[\begin{array}{l} \left\{ \left[\begin{array}{l} pos = n \\ pos = v \\ vform = vs \\ sc = \left\langle \left[\ pos = p\ \right] \right\rangle \end{array}\right] \right\} \\ sem = love(X, Y) \end{array}\right] \tag{3}$$

One serious problem in treating disjunctive feature structures is the computational complexity of their unification problem because it is essentially NP-complete[Kasper and Rounds 86]. Some practically efficient algorithms to deal with disjunctions have been studied by [Kasper 87] and [Eisele and Dörre 88].

2.2 Structural Principles

Unification-based grammars are phrase structures whose nodes are feature structures. Their grammar descriptions consist of both phrase structure rules and local constraints in a phrase structure. In current unification-based grammars, such as HPSG and JPSG, phrase structure rules become very general and grammars are mainly described with a set of local constraints called *structural principles*.

JPSG has only one phrase structure rule, as follows.



M, D and H are the *mother*, the *dependent* daughter, and the *head* daughter respectively. This phrase structure is applicable to both the *complementation structure* and *adjunction structure* of Japanese[2]. In complementation structures, D acts as a complement. In adjunction structures, D works as a modifier.

Structural principles are relations between the features of three nodes (M, D and H) in a local tree. In the following, we explain some features and their constraints.

mod: The *mod* feature specifies the function of D in a phrase structure. When the value is +, D works as a modifier, and when −, it works as a complement.

head features: Features such as *pos*, *gr*, *case*, and *infl* are called *head features*. These conform to the following *head feature principle*.

> The value of a head feature of M unifies with that of H.

subcat features: Features *subcat* and *adjacent* are called *subcat features*. They take a set of feature structures that specify adjacent categories such as complements, and nouns. The *subcat feature principle* is

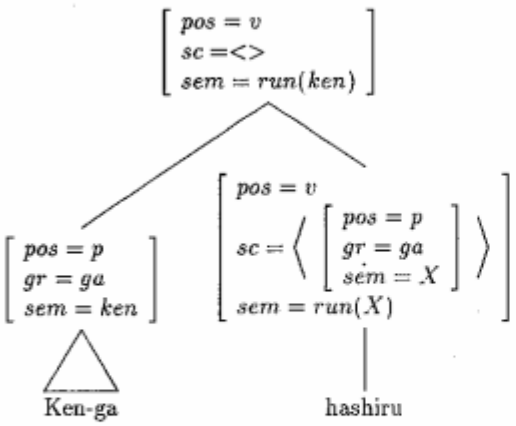
> In the complementation structure, the value of a subcat feature of M unifies with that of H minus D. In the adjunction structure, the value of a subcat feature of M unifies with that of H.

sem: The *sem* feature specifies semantic information.

> In the complementation structure, the *sem* value of M unifies with that of H. In the adjunction structure, the *sem* value of M unifies with that of D.

[2]For example, "Ken-ga aisuru (Ken loves)" is the complementation structure, and "ooki-na yama (big mountain)" is the adjunction structure.

Below is the analysis for "Ken-ga hashiru (Ken runs)."



3 cu-Prolog

3.1 Conventional Approaches

Prolog is often used as an implementation language for unification-based grammars. However, its execution strategy is fixed and procedural, i.e., always from left to right for AND processes, and from top to bottom for OR processes. Prolog programmers have to align goals such that they are solved efficiently. Prolog, therefore, is not well-suited for constraint-based grammars because it is impossible to stipulate in advance which type of linguistic constraints are to be processed in what order.

Some Prolog-like systems such as PrologII and CIL[Mukai 88] have bind-hook mechanisms that can delay some goals (constraints) until certain variables bind. As the mechanism, however, can only check constraints by executing them, it is not always efficient.

Most CLP languages, such as CLP(R)[Jaffar and Lassez 87], PrologIII, and CAL, take the constraints of algebraic domain with equations or inequations. Their constraint solvers are based on algebraic algorithms such as Gröbner bases, and solving equations. However, for AI applications and especially natural language processing systems, symbolic and combinatorial constraints are far more desirable than algebraic ones. cu-Prolog, on the other hand, can use symbolic and combinatorial constraints because its constraint domain is the Herbrand universe.

3.2 Constrained Horn Clause (CHC)

The basic component of cu-Prolog is the *Constrained Horn Clause (CHC)*[3].

[Def] 1 (CHC) *The* Constrained Horn Clause (CHC) *is*

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

$HEAD$, called **head**, *is an atomic formula, and $B_1, \ldots, B_n$, called* **body**, *is a sequence of atomic formulas. $C_1, \ldots, C_m$, called* **constraint**, *is a sequence of atomic formulas or equal constraints of the form: $Variable = Term$. Body or constraint can be empty.* □

From the viewpoint of declarative semantics, the above clause is equivalent to the following Horn Clause.

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

3.3 Derivation Rule

cu-Prolog expands the derivation rule of Prolog by adding a constraint transformation operation.

$$\frac{\overbrace{A.\,\mathbf{K}:\mathbf{C}.}^{goal} \qquad \overbrace{A' :- \mathbf{L}:\mathbf{D}.}^{program} \qquad \overbrace{\theta = mgu(A, A')}^{substitution} \qquad \overbrace{\mathbf{C}' = mf(\mathbf{C}\theta + \mathbf{D}\theta)}^{constraint\ transformation}}{\underbrace{\mathbf{L}\theta.\,\mathbf{K}\theta:\mathbf{C}'.}_{new\ goal}}$$

A and A' are heads. $\mathbf{K}$ and $\mathbf{L}$ are bodies. $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{C}'$ are constraints. $mgu(A, A')$ is the most general unifier between A and A'. $mf(Cstr)$ is a canonical form of a constraint that is equivalent to $Cstr$.

As a computational rule, when the transformation of $\mathbf{C}\theta + \mathbf{D}\theta$ fails, the above derivation rule is not applied.

3.4 PST

cu-Prolog adopts PST (Partially Specified Term) [Mukai 88] as a data structure that corresponds to the feature structure in unification-based grammars.

[Def] 2 (Partially Specified Term (PST)) PST *is a term of the following form :*

$$\{l_1/t_1, l_2/t_2, \ldots, l_n/t_n\}.$$

l_i, called **label**, *is an atom and $l_i \neq l_j (i \neq j)$. t_i, called* **value**, *is a term.* □

A recursive PST structure is not allowed.

[Def] 3 (constrained PST) *In cu-Prolog, PST is stored as an equal constraint with other relevant constraints :*

$$\mathtt{X} = \mathtt{PST}.\, \mathtt{c}_1(\mathtt{X}), \mathtt{c}_2(\mathtt{X}), \ldots, \mathtt{c}_n(\mathtt{X})$$

We call the above type of constraints constrained PST. *X=PST corresponds to [Kasper 87]'s unconditional conjunct, and $\mathtt{c}_1(\mathtt{X}), \mathtt{c}_2(\mathtt{X}), \ldots, \mathtt{c}_n(\mathtt{X})$ corresponds to the conditional conjunct.* □

In the next subsection, we give its canonical form *modular*. The constrained PST can naturally describe disjunctive feature structures of unification based grammars.

[3] Or *Constraint Added Horn Clause (CAHC).*

3.5 Canonical form of a constraint

The canonical form of a constraint in CHC is called *modular*. First, we give an intuitive definition of modular without PST.

[Def] 4 (modular (without PST)) *A sequence of atomic formulas $C_1, \ldots, C_m (m > 1)$ is* **modular** *when all its arguments are different variables.*

For example,

`member(X,Y),member(U,V)` is modular,
`member(X,Y),member(Y,Z)` is not modular, and
`append(X,Y,[a,b,c,d])` is not modular.

We expand the definition of *modular* for constrained PST.

[Def] 5 (component) *The* **component** *of an argument of a predicate is a set of labels to which the argument may bind. Here, an atom or a complex term is regarded as a PST of the label* `[]`. □

`Cmp(p,n)` stands for the component of the nth argument of a predicate `p`. `Cmp(T)` represents a set of labels of a PST `T`. In a constraint of the form `X=t`, variable `X` is regarded as taking `Cmp(t)`.

Components can be computed by static analysis of the program [Tsuda 91]. *Vacuous argument places*[Tsuda and Hasida 90] are arguments whose components are ϕ.

Consider the following example.

```
c0({f/b},X,Y):-c1(Y,X).
c0(X,b,_):-X={g/c},c2(X).
c1(X,X).
c1(X,[X|_]).
c2({h/a}).
c2({f/c}).
```

The components are computed as follows.

```
Cmp(c0,1)={f,g,h}
Cmp(c0,2)=Cmp(c1,2)={[]}
Cmp(c0,3)=Cmp(c1,1)={}
Cmp(c2,1)={f,h}
```

[Def] 6 (dependency) *A constraint has* **dependency** *when*

1. *a variable occurs in two distinct places where their components have common labels.*
2. *a variable occurs in two distinct places where one component is* `{[]}` *and another component does not contain* `[]`, *or*
3. *the binding of an argument whose component is not* ϕ. □

[Def] 7 (modular (with PST)) *A constraint is* **modular** *when it contains no dependency. A Horn clause is* **modular** *when its body has no dependency.* □

User-defined predicates in a constraint must be defined with modular Horn clauses [4].

3.6 Constraint Transformation

The constraint solver ($mf(Cstr)$) transforms non-modular constraints into modular ones by deriving new predicates. In the following, we refer to this solver as the *constraint transformer*. The constraint transformer uses the *unfold/fold transformation* dynamically. [Tamaki and Sato 83]

3.6.1 Unfold/fold transformation

Let $\mathcal{T}$ be a set of program Horn clauses, Σ be initial constraints $\{C_1, \ldots, C_n\}$ that contain variables $x_1, \ldots, x_m$, and p be a new m-ary predicate.

Let $\mathcal{P}_i$ and $\mathcal{D}_i$ be sequences of sets of clauses that are initially defined as follows.

$$\begin{aligned} \mathcal{D}_0 &= \{p(x_1, \ldots, x_m) :- C_1, \ldots, C_n.\} \\ \mathcal{P}_0 &= \mathcal{T} \cup \mathcal{D}_0 \end{aligned}$$

$mf(\Sigma)$ returns $p(x_1, \ldots, x_m)$, if and only if there exists a sequence of program Horn clauses

$$\mathcal{P}_0, \ldots, \mathcal{P}_l$$

and every clause in $\mathcal{P}_l$ is modular.

$\mathcal{P}_{i+1}$ and $\mathcal{D}_{i+1}$ are derived from $\mathcal{P}_i$ and $\mathcal{D}_i$ by one of the following three types of transformations ($0 \le i < l$).

1. unfolding

$$\frac{\begin{array}{c}\mathcal{P}_i = \{H :- A, \mathbf{R}\} \cup \mathcal{P}_i' \\ A_j :- \mathbf{B}_j \in \mathcal{P}_i, \quad A_j\theta_j = A\theta_j \quad (1 \le j \le m)\end{array}}{\begin{array}{c}\mathcal{P}_{i+1} = \bigcup_{j=1}^{m} H\theta_j :- \mathbf{B}_j\theta_j, \mathbf{R}\theta_j \cup \mathcal{P}_i' \\ \mathcal{D}_{i+1} = \mathcal{D}_i\end{array}}$$

Here, A, A_j are atomic formulas and $\mathbf{R}, \mathbf{B}_j$ are sequences of atomic formulas ($1 \le j \le m$).

2. folding

$$\frac{\begin{array}{c}\mathcal{P}_i = \{H :- \mathbf{C}, \mathbf{R}\} \cup \mathcal{P}_i' \\ A :- \mathbf{B} \in \mathcal{D}_i, \quad \mathbf{B}\theta = \mathbf{C}\end{array}}{\begin{array}{c}\mathcal{P}_{i+1} = H :- A\theta, \mathbf{R} \cup \mathcal{P}_i' \\ \mathcal{D}_{i+1} = \mathcal{D}_i\end{array}}$$

Here, $\mathbf{C}$ and $\mathbf{R}$ have no common variables.

[4] For example, `member/2`, `append/3`, and finite predicates are defined with modular Horn clauses.

3. definition
Let $\mathbf{B}$ be a sequence of atomic formulas, $x_1, \ldots, x_n$ be variables in $\mathbf{B}$, and p be a new predicate.

$$\mathcal{D}_{i+1} = \mathcal{D}_i \cup \{p(x_1, \ldots, x_n) : -\mathbf{B}.\}$$
$$\mathcal{P}_{i+1} = \mathcal{P}_i$$

3.6.2 Example of Constraint Transformation

The following example shows a transformation of `member(A,Z),append(X,Y,Z)`.

$\mathcal{T}$ = { `T1,T2,T3,T4` }, where

```
T1 = member(X,[X|Y]).
T2 = member(X,[Y|Z]):-member(X,Z).
T3 = append([],X,X).
T4 = append([A|X],Y,[A|Z]):-append(X,Y,Z).
```

and

$$\Sigma = \{\texttt{member(A,Z)}, \texttt{append(X,Y,Z)}\}$$

With a new predicate `p1/4` derived as `D1`,

```
D1 = p1(A,X,Y,Z):-member(A,Z),append(X,Y,Z).
```

we get

$$\mathcal{D}_0 = \{\texttt{D1}\}\ \mathcal{P}_0 = \mathcal{T} \cup \{\texttt{D1}\}$$

Step 1: By unfolding of the first formula of `D1`'s body (`member(A,Z)`), we get

```
T5 = p1(A,X,Y,[A|Z]):-append(X,Y,[A|Z]).
T6 = p1(A,X,Y,[B|Z]):-member(A,Z),append(X,Y,[B|Z]).
```

$$\mathcal{P}_1 = \mathcal{T} \cup \{\texttt{T5}, \texttt{T6}\}$$

Step 2: By defining new predicates `p2/4` and `p3/5` as `D2,D3`,

```
T5' = p1(A,X,Y,[A|Z]):-p2(X,Y,A,Z).
T6' = p1(A,X,Y,[B|Z]):-p3(A,Z,X,Y,B).
D2  = p2(X,Y,A,Z):-append(X,Y,[A|Z]).
D3  = p3(A,Z,X,Y,B):-member(A,Z),append(X,Y,[B|Z]).
```

we get

$$\mathcal{D}_2 = \{\texttt{D1},\texttt{D2},\texttt{D3}\}\ \mathcal{P}_2 = \mathcal{T} \cup \{\texttt{T5}', \texttt{T6}', \texttt{D2}, \texttt{D3}\}$$

Step 3: By unfolding `D2`,

```
T7 = p2([],[A|Z],A,Z).
T8 = p2([B|X],Y,A,Z):-append(X,Y,Z).
```

$$\mathcal{P}_3 = \mathcal{T} \cup \{\texttt{T5}', \texttt{T6}', \texttt{T7}, \texttt{T8}, \texttt{D3}\}$$

Step 4: Unfolding the second formula of `D3`'s body (`append(X,Y,[B|Z])`) gives

```
T9  = p3(A,Z,[],[B|Z],B):-member(A,Z).
T10 = p3(A,Z,[B|X],Y,B):-member(A,Z),append(X,Y,Z).
```

$$\mathcal{P}_4 = \mathcal{T} \cup \{\texttt{T5}', \texttt{T6}', \texttt{T7}, \texttt{T8}, \texttt{T9}, \texttt{T10}\}.$$

Step 5: Folding `T10` by `D1` generates

```
T10' = p3(A,Z,[B|X],Y,B):-p1(A,X,Y,Z).
```

Accordingly,

$$\mathcal{P}_5 = \mathcal{T} \cup \{\texttt{T5}', \texttt{T6}', \texttt{T7}, \texttt{T8}, \texttt{T9}, \texttt{T10}'\}.$$

Every clause in $\mathcal{P}_5$ is modular. As a result, `member(A,Z),append(X,Y,Z)` has been transformed into `p1(A,X,Y,Z)`. preserving equivalence. and new predicates `p1/4`.`p2/4`. and `p3/5` have been defined with `T5'`.`T6'`.`T7`.`T8`.`T9`. and `T10'`.

3.6.3 Example of constrained PST unification

Unification between constrained PSTs is done with PST unification followed by the transformation of relevant constraints.

The following example from [Eisele and Dörre 88] shows unification between two disjunctive feature structures:

$$\left[\texttt{a} = \left\{ \left[\begin{array}{l}\texttt{b} = + \\ \texttt{c} = -\end{array}\right], \left[\begin{array}{l}\texttt{b} = - \\ \texttt{c} = +\end{array}\right] \right\}\right] \text{ and } \left[\begin{array}{l}\texttt{a} = \left[\texttt{b} = \texttt{V}\right] \\ \texttt{d} = \texttt{V}\end{array}\right]$$

These disjunctive feature structures are encoded in the two constrained PSTs. `X={a/U},s(U)` and `Y={a/{b/V},d/V}`. where

```
s({b/+,c/-}).    % definition of s/1
s({b/-,c/+}).
```

PST unification between `X` and `Y` gives

```
X=Y={a/U,d/V},U={b/V},s(U).
```

There is a dependency in terms of a label `b` because `Cmp(s,1)={b,c}`.

By defining a new predicate `c1/2`. `U={b/V},s(U)` becomes equivalent to `U={b/V},c1(U,V)`. Here. `c1/2` is defined as follows. [5]

```
c1({c/-},+).
c1({c/+},-).
```

Note that `X=Y={a/U,d/V},U={b/V},c1(U,V)` does not have any dependency because `Cmp(c1,1)={c}`.

[5] Precisely, *abstraction* operation in [Tsuda 91] is used in this transformation. In *abstraction*. PST unifications are made in terms of relevant labels alone for efficiency.

4 JPSG parser in cu-Prolog

JPSG (Japanese Phrase Structure Grammar)[Gunji 86] is a constraint-based and unification-based grammar designed specifically for Japanese. It is being developed by the PSG working group at ICOT.

To implement unification-based grammars, we have to consider how to describe and process feature structures for the first time. In cu-Prolog, PST enables the natural implementation of non-disjunctive feature structures. The labels of PST correspond to the features of a feature structure. As mentioned earlier, disjunctive feature structures correspond to constained PSTs.

In cu-Prolog, both disjunctive feature structures and structural principles are treated as constarints in CHC.

4.1 Encoding Lexical Ambiguity

As an example of the disjunctive feature structures, this subsection explains lexical ambiguities in this subsection. Consider the lexicons of homonyms or polysemic words. If the lexicon of an ambiguous word is divided into plural entries in terms of its ambiguity, the parsing process may be inefficient in that it sometimes backtracks to consult the lexicon. In constraint-based natural language processing, such ambiguity is packed as a constraint in a lexicon.

Below is a sample lexicon of Japanese auxiliary verb "reru." "reru" follows a verb whose inflection type is `vs` or `vs1`. If the adjacent verb is transitive, "reru" indicates plain passive. If the verb is intransitive, "reru" indicates affective passive [6]. These combinations are represented by adding constraints of `reru_form/1` and `reru_sem/4` in one lexical entry.

```
%% lexical entry of ''reru''
lex(reru,{sc/SC, sem/Sem,
       adjacent/{pos/v,infl/I,sc/VSC,sem/Sem}});
   reru_form(I),     % inflection (constraint)
   reru_sem(VSC,Vsem,SC,Sem).
     % combination of subcat and sem (constraint)

%%%%%% definition of constraints %%%%%%
reru_form(vs). % inflection type of the adjacent verb
refu_form(vs1).

% constraint for intransitive (affective) passive
reru_sem([{form/ga,sem/Sbj}],Sem,
       [{form/ga,sem/A},{form/ni,sem/Sbj}],
       affected(A,Sem)).
% constraint for transitive (normal) passive
reru_sem([{form/ga,sem/Sbj},{form/wo,sem/Obj}],
       Sem,
       [{form/ga,sem/Obj},{form/ni,sem/Sbj}],
       Sem).
```

[6] For example, "Ken ga ame ni fu-ra-reru" (Ken is affected by the rain.)

This lexicon is a representation of the following disjunctive feature structure.

$$\left[\left\{\left[\begin{array}{l} adjc = \left[\begin{array}{l} sc = \left\langle \left[\begin{array}{l} pos = ga \\ sem = S1 \end{array}\right] \right\rangle \\ sem = Sem1 \end{array}\right] \\ sc = \left\langle \left[\begin{array}{l} form = ga \\ sem = A \end{array}\right], \left[\begin{array}{l} form = ni \\ sem = S1 \end{array}\right] \right\rangle \\ sem = affected(A, Sem1) \end{array}\right], \left[\begin{array}{l} adjc = \left[\begin{array}{l} sc = \left\langle \left[\begin{array}{l} pos = ga \\ sem = S2 \end{array}\right], \left[\begin{array}{l} pos = wo \\ sem = O2 \end{array}\right] \right\rangle \\ sem = Sem2 \end{array}\right] \\ sc = \left\langle \left[\begin{array}{l} pos = ga \\ sem = O2 \end{array}\right], \left[\begin{array}{l} pos = ni \\ sem = S2 \end{array}\right] \right\rangle \\ sem = Sem2 \end{array}\right]\right\}, \left[\begin{array}{l} adjc = \left[\begin{array}{l} pos = v \\ infl = \{vs1, vs2\} \end{array}\right] \end{array}\right]\right]$$

Although the lexicon is ambiguous, however, many kinds of constraints are automatically accumulated for solving during parsing. The disambiguation process in parsing is naturally realized by the constraint transformation of cu-Prolog. It has no need to write any special procedure for disambiguation.

4.2 Encoding Structural Principle

As mentioned in Section 2, the structural principles of JPSG are relations among features of three categories in a local tree. Intuitively, structure principles are encoded as constraints to a phrase structure rule:

$$psr(M, D, H); sp_1(M, D, H), \ldots, sp_n(M, D, H).$$

Here, $psr/3$ is a phrase structure rule and each $sp_i/3$ is a structure principle.

In cu-Prolog, these structural principles are evaluated flexibly with heuristics. In Prolog, however, above phrase structure rule is represented as

$$psr(M, D, H) :- sp_1(M, D, H), \ldots, sp_n(M, D, H).$$

Each principle is always evaluated sequentially. Prolog, therefore, is not well-suited for constraint based grammars because it is impossible to stipulate in advance which kind of linguistic constraints must be processed in what order.

As the first example, the principle of the *sem* feature in Section 2 is encoded as a constraint `sfp(M,D,H)`, where

```
sfp({sem/HS},{mod/+},{sem/HS}).
sfp({sem/HS},{mod/+},{sem/HS}).
```

As the second example, the *Foot Feature Principle* is defined as follows[Gunji 86].

> The value of FOOT feature of the mother unifies with the union of those of her daughters.

It is represented as constraint `ffp(M,D,H)`, where

```
ffp({foot/MF},{foot/DF},{foot/HF}) :-
  union(DF,HF,MF).
```

5 Implementation

cu-Prolog has been implemented in the C language of UNIX4.2/3BSD and the Apple Macintosh[Sirai 91]. cu-PrologIII [Tsuda *et al.* 92] is the latest implementation.

This section presents some implementation issues that relate particularly to the constraint transformer.

5.1 Constraint Transformer

5.1.1 Constraint Transformation Strategy

cu-Prolog uses the following three clause pools during constraint transformation.

`DEFINITION`: derivation clauses of new predicates

`NON-MODULAR`: non-modular clauses

`MODULAR`: modular clauses

The following is the transformation procedure of cu-Prolog.

1. If `DEFINITION` is not empty, remove one clause from `DEFINITION` and unfold it.
2. If `DEFINITION` is empty but `NON-MODULAR` is not empty, remove one clause N from `NON-MODULAR`. If N's head is modular, unfold N. If not, fold N or derive new predicates to N's body.
3. Repeat the above operations until `DEFINITION` and `NON-MODULAR` are both empty.

5.1.2 Heuristics

One of the outstanding features of cu-Prolog is the heuristics used in the constraint transformation.

The following three choices are available.

- selection of a clause from `DEFINITION`
- selection of a clause from `NON-MODULAR`
- selection of a formula to unfold

`DEFINITION` and `NON-MODULAR` are implemented by stacks, that is, the constraint transformer selects the latest. In unfolding, the *activation value* of each atomic formula is computed from the following formulas and the atomic formula of the highest value is unfolded.

$Arity$ = arity of the formula
$Const$ = number of arguments that bind to constants
$Vnum$ = total number of occurrence of variables in the formula
$Funct$ = number of arguments that bind to complex terms
Rec = If the predicate is recursively defined then 1, otherwise 0
$Defs$ = number of definition clauses of the predicate
$Units$ = number of unit clauses in the predicate definition
$Facts$ = If $Defs = Units$ then 1, otherwise 0

The activation value A of an atomic formula is computed using the following formula.

$$A = 3 * Const + 2 * Funct + Vnum - Defs + Units - 2 * Rec + 3 * Facts$$

We define each factor of the activation value as including some empirical heuristics of [Tsuda *et al.* 89a]. There may, however, be more effective heuristics with more factors or with a non-linear formula[Hasida 91].

5.2 Example of cu-PrologIII

Figure 1 is an example of disjunctive feature unification in [Kasper 87].

Figure 2 is an example of the JPSG parser in cu-PrologIII. For ambiguous sentences, the parser returns the corresponding feature structure with constraints.

6 Concluding Remarks

This paper outlined cu-Prolog, then covered the disjunctive feature structure and parsing JPSG.

We would like to stress that every feature mentioned in this paper was equally processed in the same framework as a constraint transformation. In comparison with many conventional approaches, our approaches, including Hasida's DP (Dependency Propagation) [Hasida 91], are far more general and flexible frameworks for constraint-based natural language processing.

Acknowledgment

The author thanks Hidetosi Sirai of Chukyo University for his cooperation in implementing cu-Prolog. Thanks are also due to Kazumasa Yokota, Hideki Yasukawa, and Kôiti Hasida and other members of ICOT for their comments.

References

[Carpenter *et al.* 91] Bob Carpenter, Carl Pollard, and Alex Franz. The Specification and Implementation of Constraint-Based Unification Grammar. In *Proc. of Second International Workshop on Parsing Technologies*, pages 143–153. Sigparse ACL, February 1991.

[Chomsky 81] Norm Chomsky. *Lectures on Government and Binding.* Foris. Dordrecht, 1981.

[Eisele and Dörre 88] Andreas Eisele and Jochen Dörre. Unification of Disjunctive Feature Descriptions. In *Proc. of 26th Annual Meeting of ACL*, pages 286–294, June 1988.

[Gunji 86] Takao Gunji. *Japanese Phrase Structure Grammar.* Reidel. Dordrecht, 1986.

[Hasida 91] Kôiti Hasida. Common Heuristics for Parsing, Generation, and Whatever. In *Workshop on Reversible Grammar in Natural Language Processing*, Berkeley, 1991.

[Hasida and Sirai 86] Kôiti Hasida and Hidetosi Sirai. Conditioned Unification. *Computer Software*, 3(4):28–38, 1986. (in Japanese).

[Jaffar and Lassez 87] Joxan Jaffar and Jean-Louis Lassez. Constraint Logic Programming. In *Proc. of 14th ACM POPL Conference*, pages 111–119. Munich, 1987.

[Kasper 87] Robert T. Kasper. A Unification Method for Disjunctive Feature Descriptions. In *Proc. of 25th Annual Meeting of ACL*, pages 235–242. July 1987.

[Kasper and Rounds 86] Robert T. Kasper and William C. Rounds. A Logical Semantics for Feature Structure. In *Proc. of 24th ACL Annual Meeting*, pages 257–266. 1986.

[Kay 85] Martin Kay. Parsing in Functional Unification Grammar. In David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, editors. *Natural Language Parsing*, chapter 7, pages 251–278. Cambridge University Press, 1985.

[Mukai 88] Kuniaki Mukai. Partially Specified Term in Logic Programming for Linguistic Analysis. In *Proc. of the International Conference of Fifth Generation Computer Systems*, pages 479–488. ICOT, OHMSHA and Springer-Verlag, 1988.

[Pollard and Sag 87] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics. Vol.1 Fundamentals.* CSLI Lecture Notes Series No.13. Stanford:CSLI. 1987.

[Shieber 86] Stuart M. Shieber. *An Introduction to Unification-Based Approach to Grammar.* CSLI Lecture Notes Series No.4. Stanford:CSLI. 1986.

[Sirai 91] Hidetosi Sirai. A Guide to MacCUP. unpublished. 1991. (available by anonyous ftp from csli.stanford.edu (pub/MacCup)).

[Tamaki and Sato 83] Hisao Tamaki and Taisuke Sato. Unfold/Fold Transformation of Logic Programs. In *Proc. of Second International Conference on Logic Programming*, pages 127–137. 1983.

[Tsuda 91] Hiroshi Tsuda. Disjunctive Feature Structure in cu-Prolog. In *8th Conf. Proc. of Japan Society of Software Science and Technology*, pages 505–508. 1991. (in Japanese)

[Tsuda and Hasida 90] Hiroshi Tsuda and Kôiti Hasida. Parsing as Constraint Transformation — an Extension of cu-Prolog. In *Proc. of the Seoul International Conference on Natural Language Processing*, pages 325–331. 1990.

[Tsuda *et al.* 89a] Hiroshi Tsuda, Kôiti Hasida, and Hidetosi Sirai. cu-Prolog and its Application to a JPSG Parser. In K.Furukawa, H.Tanaka, and T.Fujisaki, editors, *Logic Programming '89*, pages 134–143. Springer-Verlag LNAI-485, 1989.

[Tsuda *et al.* 89b] Hiroshi Tsuda, Kôiti Hasida, and Hidetosi Sirai. JPSG Parser on Constraint Logic Programming. In *Proc. of 4th ACL European Chapter*, pages 95–102. 1989.

[Tsuda *et al.* 92] Hiroshi Tsuda, Kôiti Hasida, and Hidetosi Sirai. cu-PrologIII System. ICOT Technical Memorandum. ICOT TM-1160. 1992.

```
cc1({voice/passive,trans/trans,subj/X,goal/X}). % definition of the unconditional conjuncts
cc1({voice/active, subj/X,actor/X}).
cc2({trans/intrans, actor/{person/third}}).
cc2({trans/trans, goal/{person/third}}).
cc3({numb/sing, subj/{numb/sing}}).
cc3({numb/pl, subj/{numb/pl}}).

% disjunctive feature unification (user input)
@ U={rank/clause, subj/{case/nom}}, cc1(U),cc2(U),cc3(U), U={subj/{lex/you,person/second,numb/pl}}.

% answer: equivalent constraint
solution = c0(U_0, {subj/{case/nom}, rank/clause}, {subj/{person/second, numb/pl, lex/you}})

% definitions of a new predicate (c0)
c0(_p1, _p1, _p1) :- cc2(_p1), cc1(_p1);
  _p1={subj/{person/second, numb/pl, case/nom, lex/you}, numb/pl, rank/clause}.

CPU time = 0.150 sec (Constraints Handling = 0.000 sec)

>:-c0(X,_,_).       % solve the new constraint
success.            % X is the final answer of the unification.
 X = {voice/active, trans/trans, subj/{person/second, numb/pl, case/nom, lex/you},
 goal/{person/third}, actor/{person/second, numb/pl, case/nom, lex/you}, numb/pl, rank/clause};
```

Lines beginning with "@" are user inputs. To this input, cu-Prolog returns equivalent modular constraint and definition clauses of newly defined predicates.

Figure 1: Disjunctive feature unification

```
_:-p([ken,ga,ai,suru]).      % user input of ''Ken ga ai-suru.''

    %%% parse tree
{sem/[love,V7_2030,V6_2029], core/{form/Form_1381, pos/v}, sc/V1_2024,
 refl/[], slash/V3_2026, psl/[], ajn/[], ajc/[]}---[suff_p]
 |
 |--{sem/[love,V7_2030,V6_2029], core/{pos/v}, sc/V0_2023, refl/[],
    slash/V2_2025, psl/[], ajn/[], ajc/[]}---[subcat_p]
 |  |
 |  |--{sem/ken, core/{form/ga, pos/p}, sc/[], refl/[], slash/[],
      psl/[], ajn/[], ajc/[]}---[adjacent_p]
 |  |  |
 |  |  |--{sem/ken, core/{form/n, pos/n}, sc/[], refl/[], slash/[],
         psl/[], ajn/[], ajc/[]}---[ken]
 |  |  |
 |  |  |__{sem/ken, core/{form/ga, pos/p}, sc/[], refl/[], slash/[],psl/[], ajn/[],
         ajc/[{sem/ken, core/{pos/n}, sc/[], refl/ReflAC_70}]}---[ga]
 |  |
 |  |__{sem/[love,V7_2030,V6_2029], core/{form/vs2, pos/v}}---[ai]
 |
 |__{sem/[love,V7_2030,V6_2029], core/{form/Form_1381, pos/v}, sc/[], refl/[],
    slash/[], psl/[], ajn/[], ajc/[{sem/[love,V7_2030,V6_2029],
    core/{form/vs2, pos/v}, sc/[], refl/ReflAC_1493}]}---[suru]

category= {sem/[love,V7_2030,V6_2029], core/{form/Form_1381, pos/v},
    sc/V1_2024, refl/[], slash/V3_2026, psl/[], ajn/[], ajc/[]} %category

constraint= c40(V0_2023, V1_2024, V2_2025, V3_2026, V4_2027, V5_2028,
    {sem/ken, core/{form/ga, pos/p}, sc/[], refl/[], slash/[], psl/[],
    ajn/[], ajc/[]}, V6_2029, {sem/V6_2029, core/{form/wo, pos/p}}, V7_2030,
    {sem/V7_2030, core/{form/ga, pos/p}}),
  syu_ren(Form_1381) %constraint about the category
true.
CPU time = 2.217 sec (Constraints Handling = 1.950 sec)

_:-c40(V1, _, _, V3, _, _,_, V6,_ , V7,_).      %solve constraint
 V1 = []  V3 = [{sem/V0_4}]  V6 = V0_4  V7 = ken;     % solution 1
 V1 = [{sem/V0_4, core/{form/wo, pos/p}}]  V3 = []  V6 = V0_4  V7 = ken;  % solution 2
no.
CPU time = 0.017 sec (Constraints Handling = 0.000 sec)
```

The parsing of "Ken ga ai-suru" that has two meanings: "Ken loves (someone)" or "(someone) whom Ken loves." The parser draws a corresponding parse tree and returns the category of the top node with constraints. In this example, the ambiguity of the sentence is shown in the two solutions of the constraint c40.

Figure 2: JPSG parser: disambiguation