BUP Optimization by Pre-evaluation and Term Generalization

Hideki HIRAKAWA, Hideki YASUKAWA, Ko SAKAI

ICOT Research Center
1-4-28 Minato-ku, Tokyo, 108, Japan

Abstract

This paper describes an optimization method of the BUP parsing system using pre-evaluation of a grammar for extracting an information and a term generalization method for transforming the obtained information.

1. Introduction

One purpose of pre-evaluating a logic program is to extract useful information for optimizing and verifying the program. For example, Sato and Tamaki described a pre-evaluation method and its application to optimization of a logic program where a nondeterministic program is transformed into a deterministic program using the variable binding information [1]. Pereira has proposed using the Earley deduction system to extract various types of information from a logic grammar [3].

It is important to establish methods for utilizing information obtained by pre-evaluation. For example, for program optimization, program transformation algorithms utilizing extracted information need to be established. In this case, not only must algorithms improve the efficiency of the program but also guarantee equivalence of the execution results with the original program [2].

Following F.Pereira's work, this paper describes the use of Earley deduction system for extracting grammar information for BUP [4] optimization. BUP, a bottom up parser in Prolog, adopts an optimization method based on the reachability check on the derivation path [6]. The reachability relation is called 'link' relation in BUP. This optimization is performed by embedding link check codes in a BUP program. In this paper, the link relation and its uses in BUP are extended so as to include more grammar information. This improves the parsing efficiency of BUP even more. The computation of extended link relations requires two kinds of pre-evaluations, pre-evaluation of link relation and a pre-evaluation of calling patterns each of which uses a modified Earley deduction system. The result of link pre-evaluation forms the base for the extended link relation. The calling pattern information is used to transform the link

relation in order to guarantee determinacy of the BUP execution. A term generalization method called 'least common covering' is a key technique in this transformation.

Section 2 describes the logic grammar and the BUP's link relation. Section 3 shows the extended link relation and its uses. Section 4 describes the link computation using Pereira's pre-evaluation method and discusses its correctness. Section 5 describes the extended link computation and the generalization method.

2. Logic Grammar and Link Relation

2.1 Logic Grammar

The grammar handled in this paper is a context free grammar (CFG) extended by argument attachment to grammar categories. We call it a logic grammar. A logic grammar is defined by grammar rules and a starting literal set. A grammar rule is written in the following format.

```
H => G1,G2,...,Gn.  (1 =< n)
     where H is a literal and Gi (1 =< i =< n) is a literal or
     a list of terminals.
```

Extra conditions and the 'cut' operator introduced in Definite Clause Grammar [5] are not allowed in logic grammar.

The starting literal set ST(G) is a set of literals corresponding to the starting symbol in CFG [6]. We write a starting literal set as follows:

```
ST(G) = {S1,...,Sn} where Si (1 =< i =< n) are literals.
```

2.2 Link Relation in BUP

To make the discussions of the latter sections clear, we briefly explain the BUP parsing algorithm and the link relation in BUP. BUP provides a left-corner, bottom-up parser with top-down prediction. Assume the following grammar rule:

```
s(s(Np,Vp)) => np(Np), vp(Vp).
```

BUP interprets this rule as "If a partial parsing for 'np' is obtained then try to parse 'vp'. If it succeeds the partial parsing 's' is obtained." A grammar rule is translated to a Prolog clause (BUP clause) according to the BUP interpretation. For example, the above grammar rule is translated to the following BUP clause.

BUP clause :

```
np(Goal,[Np],S0,S,Arg) :-
     link(s,Goal),
     goal(vp,[Vp],S0,S1),
     s(Goal,[s(Np,Vp)],S1,S,Arg).
```

For the details of a BUP clause, see [4]. Here we give only a brief explanation of this clause. This clause is executed when 'np' is called, which means partial parsing of 'np' has succeeded. 'goal' predicate performs top-down prediction and tries to parse 'vp'. If it succeeds, 's' is called. This means partial parsing of 's' has succeeded. 'link' is a predicate for optimization by testing reachability between 's' and 'Goal'. The variable 'Goal' is a current goal of a partial parse. it corresponds to a head of a partial parsing tree currently predicted. The variable 'Goal' is instantiated to the grammar category when the clause is called. 'link(cat1,cat2)' succeeds when c1 can be the leftmost descendant of c2 in a parse tree. Therefore, if 'link(s,Goal)' fails, subsequent parsing will eventually fail. The link relation is computed by considering all grammar rules.

3. Extension of Link

The extended link relation is defined in terms of the leftmost decendant relation which is defined over parsing trees corresponding to sentences generated by a grammar.

3.1 Parsing Tree and Leftmost Decendant Relation

A partial parsing tree reflects the derivation steps of a substring corresponding to a nonterminal literal of G. A partial parsing tree is a tree whose nodes are terms corresponding to nonterminal literals or terminals.

A tree is represented in the following syntax.

PtHead{Arg1,..,Argm} (0 < m)
     where 'PtHead' is a nonterminal literal and 'Arg's are trees or terminals.

The noterminal literal PtHead is called the head of the tree. The head of a tree t is denoted by Head(t).

Partial parsing trees of a grammar and dotted derived rules are defined recursively. In what follows @, @1, @2 represent substitutions. For example, @t1=@t2 means that @ is a unifier of t1 and t2.

Definition: Partial parsing tree and dotted derived rule

a. If H => G1,...,Gn is a rule in G, then H => *,G1,...,Gn is

a dotted derived rule.

b. If H => ...*,Gi... is a dotted derived rule such that Gi is a terminal, then H => ...Gi,*... is a dotted derived rule.

c. If H => ...*,Gi... is a dotted derived rule and T is a partial parsing tree such that @Head(T)=@Gi, then @(H => ...T,*...) is a dotted derived rule.

d. If H => T1,...,Tn,* is a dotted derived rule then H{T1,...,Tn} is a partial parsing tree.

For a dotted derived rule H => T1,...,Tm,*,G1,...,Gn (0 =< m,n), then this rule is called m-dotted derived rule, G1 the dotted literal, and H the head of the rule. Fig 1 shows the tree representing a m-dotted derived rule.

In Fig 1, H is the head of the rule and G1 is the dotted literal.

The set of all the dotted derived rule for a grammar G is denoted by DR(G). The set of all the partial parsing tree of a grammar G is denoted by PPT(G).

A parsing tree is a partial parsing tree whose head is unifiable with a start literal of a grammar G. The set of parsing trees is denoted by PT(G).

If there exists a partial parsing tree M{D1,D2,...,Dm} in PPT(G), the head of D1 is called a leftmost daughter of M.

The leftmost decendant relation is defined as the reflexive and transitive closure of the leftmost daughter relation.

Definition: Leftmost descendant set of a grammar G

Given a grammar G, the leftmost decendant set of G LD(G) is defined as the set of terms representing the leftmost descendant relation.

LD(G) = {link(t1,t2) | t1 is a leftmost descendant of t2}

3.2 Link Set Definition

In the original BUP, link relations are defined only in terms of grammatical category. In the extension described above, they are defined over terms corresponding to nonterminal literals taking into account of their argument, and can be used to extract more information from a parse. However, since the leftmost descendant set is not necessarily finite, we must consider a link set in actual application.

A 'cover' relation between two terms is defined as follows.

Definition: cover of a term

A term g covers a term t iff there exists a substitution @ such that @(g) = t.

A set of terms G covers a set of terms T iff for any term t in T there exists at least one term g in G such that g covers t.

A link set is defined as follows.

Definition: Link set of a grammar G

a finite cover of the leftmost decendant set LD(G) and denoted by L(G).

3.3 Indirect use of link set

With the definition of a leftmost descendant set LD(G) described above, a typical appilication of a link set is checking the reachability as in the original BUP.

The typical use of a link set in BUP is shown below. The link terms in a link set is used as the unit clauses called link clause in BUP.

```
s(s(A,B)) => a(A),b(B).          (Grammar rule)

a(G,[A],S0,S,Arg) :-             (BUP clause)
     link_check(s(s(A,B)),G),
     goal(b(B),[B],S0,S1),
     s(G,[s(A,B)],S1,S,Arg).

link(s(X),s(X)).                 (A link clauses)
link(a(X),a(X)).
link(b(X),b(X)).
link(a(A),s(s(A,B))).

link_check(X,Y) :- not_link(X,Y),!,fail.
link_check(_,_).

not_link(X,Y) :- link(X,Y),!,fail.
not_link(_,_).
```

The BUP clause above is invoked when a partial parsing tree for the nonterminal literal a(A) is obtained. Suppose the current goal is s(_). That is, the current parsing process aims at obtaining a partial parsing tree coresponding to the nonterminal literal s(S). If the head of the partial parsing tree s(s(A,B)) to be obtained by the invokation of the BUP clause does not stand in link relation with the curent goal, the invocation of the BUP

clause falls into fail eventually and should be prohibited. In the above case, however, the predicate link_check succeeds and the BUP clause is invoked to obtain a partial parsing tree for s(s(A,B)). The point of the predicate link_check is that no substitution occurs by its execution and there is no alternatives for it, in other words, only the existence of the unifiable link clause is checked . We call this 'indirect use' of a link set. The indirect use of a link set is complete and sound in the sense that all and only the legal substrings for nonterminals in G are accepted.

3.4 Direct use of link set

In contrast to the indirect use of link relations, it is possible to use a link set directly by embedding a call for a link clause in a BUP clause. In this case, a link clause call has the function of instantiating the arguments of a nonterminal literal to be analyzed in a succesive parsing process. We call this 'direct use' of a link set.

An example of direct use is shown below.

```
s(s(A,B)) => a(A),b(B).          (Grammar rule)

a(G,[A],S0,S,Arg) :-             (BUP clause)
     link(s(s(A,B)),G),
     goal(b(B),[B],S0,S1),
     s(G,[s(A,B)],S1,S,Arg).
```

In original BUP and the indirect use, it is impossible to pass the arguments in a top-down manner. The direct use provides the opportunity to pass some arguments in a top-down manner. For example, consider the following grammar.

```
s => np(Num),vp(Num).
vp(Num) => verb1(Num),np(Num1).
verb1(Num) => tv(Num).
```

Consider the following BUP clauses and the link set for this grammar.

```
link(s,s).
link(np(X),np(X)).
link(vp(X),vp(X)).
link(verb1(X),verb1(X)).
link(tv(X),tv(X)).
link(np(N),s).
link(verb1(N),vp(N)).
link(tv(N),verb1(N)).
link(tv(N),vp(N)).

np(G,[Num],S0,S,Arg) :-
      link(s,G),
```

```
        goal(vp(Num),[Num],S0,S1),s(G,[],S1,S,Arg).
verb1(G,[Num],S0,S,Arg) :-
        link(vp(Num),G),
        goal(np(Num1),[Num1],S0,S1),vp(G,[Num],S1,S,Arg).
tv(G,[Num],S0,S,Arg) :-
        link(verb1(Num),G),verb1(G,[Num],S0,S,Arg).
```

After calling np, the goal vp(Num) is resolved. Suppose its argument has been already instantiated by the parse of np and a terminal of a category tv is found on the input string. Then, tv is called with the current goal vp(Num). The link relation between verb1(Num') and the current goal vp(Num) is checked where Num' is given by the terminal of a category tv . The link clause corresponding to this link check is link(verb1(N),vp(N)) and Num and Num' is unified. If they are unifiable, then the parse is continued, otherwise fails. Thus, the goal vp(Num) carries the information given by the analysis of the np in a top-down mannar. The top-down information passing is impossible in original BUP, but it is useful in blocking useless parsing early.

As each of the two arguments of a link relation call matches a head of a parsing tree, the direct use of link carries the information between parsing trees by unification.

4. Pre-evaluation of Link Relations

In order to compute a link set for a given grammar, the grammar is pre-evaluated by the Earley deduction system with some modification. First, the short introduction of the Earley deduction system developped by F. Pereira is given.

4.1 Earley deduction system

The Earley deduction system developped by F. Pereira [7] is a general proof procedure for definite clauses with a practical strategy for unit resolution.

The Earley deduction system is given a set of definite clauses and the goal to be solved. The syntax of a definite clause is as follows:

```
<clause> ::== <term> <- <goals>
<goals>  ::== <goal>
<goals>  ::== <goals> <goals>
<goal>   ::== true
<goal>   ::== <term>
<goal>   ::== { <prolog goals> }
```

New derived clauses are derived by resolution from the definite clauses (input clauses). For example, given a set of definite clauses

```
a(C) <- b(B),c(B,C).          ... (1)
b(b) <- .                     ... (2)
c(b,a) <- .                   ... (3)
```

and a goal a(X), the clause (1) becomes the input clause for refuting the goal and the new derived clause.

```
a(Y) <= b(X),c(X,Y)           ... (4)
```

As in (4), derived clauses are represented by '<=' while definite clauses are represented by '<-'. This inference step is called instantiation. The leftmost goal of the derived clause (4), that is b(X), is selected as a candidate subgoal. Then the unit clause (2) becomes the input clause and new derived clause.

```
b(b) <= .                     ... (5)
```

At this point, the subgoal b(B) is refuted. The derived unit clause corresponds to the proved instance of a goal. Then the derived unit clause (5) is used to derive new derived clauses, that is, the derived clause (4) is reduced by the derived unit clause (5) and the new derived clause (6) is generated. If there exists other derived clauses whose leftmost goals are unifiable with (5), they are also reduced by (5) and the new derived clauses are generated. This step of inference is called reduction.

```
a(Y) <= c(b,Y).               ... (6)
```

In the next step the goal of (6), that is c(b,Y), is selected as a candidate clause. The unit clause (3) becomes the input clause for the subgoal and is instantiated.

```
c(b,c) <= .                   ... (7)
```

Then the derived clause (6) is reduced by (7).

```
a(c) <= .                     ... (8)
```

No more deduction is possible at this point and the deduction of the goal a(X) is terminated.

The derived clauses are stored during the deduction process and used for subsumption check at instantiation steps, that is, derivation of a derived clause from a definite clause is blocked if it is subsumed by the derived clause already derived. The notion subsume is equivalent to the notion cover defined in the previous section.

There will be a set of derived unit clauses that subsume all the provable instances of the goal among the derived clauses.

The instantiation step corresponds to top-down expectation and the reduction step corresponds to bottom-up resolution. The strategy of selecting a subgoal at an instantiation step is to select the leftmost goal of the body of the clause, while the order in which derived clauses are considered is breadth-first.

4.2 Link computation program

The basic algorithm for computing a descendant set of a grammar is represented by the following Earley program [3].

```
(1) link(A,A) <- category(A).
(2) link(A,B) <- starts(A,B).
(3) link(A,B) <- link(A,X),link(X,B).

(4) category(A) <- (A=>_).
(5) starts(A,B) <- (B=>A1),first(A,A1),nonterminal(A).
```

Clause (1) represents the reflexivity of leftmost descendant relations. That is, a term corresponding to a head of a grammar rule stands in link relation with itself. As each node of a partial parsing tree is an instantiation of a head of a grammar rule, this clause generate a reflexive leftmost descendant relation.

Clause (2) represents the leftmost daughter relation between the head of a partial parsing tree and its leftmost daughter. The predicate 'starts' extracts pairs consisting of a head of a grammar rule and its leftmost daughter literal.

Clause (3) describes the transitivity of leftmost descendant relation between the head of a partial parsing tree and its leftmost descendants.

It is obvious that the terms 'link' computed by the above program is a leftmost decendant set of a given grammar. But, in general, the leftmost descendant set is not a link set for a grammar, because it eventually be a infinite set of terms. The problem arises when cyclic application of a certain kind of grammar rule generates an infinite number of terms. For example, the appilication of clause (3) for the grammar rule like

```
np(np(Np,Rel)) => np(Np),rel(Rel)
```

falls into infinite derivation of link terms like the following :

```
link(np(Np),np(np(Np,Rel)))                (by the clause (2))
link(np(Np),np(np(np(Np,Rel1),Rel)))       (by the clause (3))
             .
             .    (by invoking clause (3) repeatedly)
             .
link(np(Np),np(np(..np(np(Np,Relk),Relk-1),..,Rel1),Rel))
```

In order to block infinite derivation of link terms, some kind of term abstraction method must be applied to the derivation of link terms and the derivation of link terms by invoking clause (3) must be subsumed by it. Several term abstraction methods can be used for this purpose, for example, k-level abstraction [1], etc. (See Appendix A for details). For the purpose of generating link terms for this grammar rule, the more general link term

```
link(np(Np),np(np(X,Y)))
```

would be sufficient for prediction in BUP. That is, an np can starts with an np whose argument is np(X,Y). Thus, the abstraction method sufficient for this purpose abstracts a nested term like np(np(..np(Np,Rel),...) with a free variable at some level of a resulting link relation. This abstraction method is developped by F.C.N. Pereira and called the weakening method. Weakening means that each occurence of a subterm within a term is replaced by a free variable when both of them have the same principal functor. For example, the result of weakening the term np(np(np(X),and,np(Y))) is np(np(V1,and,V2)). Originally Pereira used weakning embedding in the link computation program as prolog goal. In contrast, we use weakening as abstraction method for derived clauses and embed it into the Earley deduction system. The Earley deduction system with term abstraction can be used to pre-evaluate the program and extract the information of it. For the purpose of computing link, weakening is a good term abstraction method and the Earley deduction system with weakening is a pre-evaluater.

In a Earley deduction system with term abstraction, each term in a derived clause is abstracted. For an arbitrary term T and its abstracted term T', T' covers T. Since, the number of the given definite clauses and the symbols of the domain of a program is finite, it is obvious that the number of the derived clauses generated during Earley deduction with the weakening is finite. The infinite derivation of link terms as in the above example is blocked by subsumption check. Thus, Earley deduction with term abstraction always terminates and generates finite number of answer.

For the above example grammar rule, the term

```
link(np(Np),np(np(X,Y)))
```

derived by the Earley deduction system with the weakening covers the infinite set of terms representing leftmost descendant derived by the original Earley deduction system.

4.3 Correctness of the Computed Link Set

In this section, the proof of the correctness that the set of link terms L' given by the Earley deduction system with term abstraction is presented. The weakening is one possible term abstraction method. Correctness means that L' satisfies the definition of a link set described in Section 3.2.

The set L of link terms given by the definite clauses (1)-(5) by the original Earley deduction covers the leftmost descendant set of a grammar. Consequently, it is sufficient to show that L' covers L.

Let the link computation program be P. Suppose a grammar G and the goal link(X,Y) are given. Let the set of all derived clauses generated during the execution of P with G by the original Earley deduction system E be D and that generated by the Earley deduction system with term abstraction E' be D'. Notice that D' is a finite set while D is possibly a infinete set.

At first, the following lemma will be proved.

Lemma:

D' covers D

Proof:

The proof of the above lemma is based on induction on the number of the steps from the goal to some point in the deduction. Di and Di' represent the sets of all derived clauses at the i-th step for of E and E' respectively.

The base case (the first step) is obvious.

Consider the induction step and assume that Dk' covers Dk.

There are two cases of the inference step, at the (k+1)-th step, to be considered in the Earley deduction system, the instantiation step and the reduction step.

case 1 (reduction step)

Suppose the (k+1)-th step is the reduction step both for E and E'.

There exists two derived clauses (1) and (2) in Dk used for the reduction.

```
H <= G,Goals.          (1)
F <= .                 (2)
```

such that there exist a most general unifier u of G and F.

And there exist two derived clauses (3) and (4) in Dk' each of which covers the derived clauses (1) and (2). Renaming of the variables is implicit. Thus, (3) and (4) have no common variables.

H' <= G',Goals'. (3)
F' <= . (4)
such that there exists a most general unifier u' of G' the F'.

There exists

(H <= G,Goal) = g(H' <= G',Goal')

(F <=) = g(F' <=)

At the (k+1)-th step, the following two derived clauses are generated by the reduction :

uH <= uGoals (5) (in the case of E)

u'H' <= u'Goals' (6) (in the case of E')

It is proved that the derived clause (6) covers the derived clause (5).

We have

u(gG') = uG = uF = u(gF').

u' is the most general unifier of G' and F', hence ug = g'u' for some substitution g'. Thus,

g2(u'(H' <= Goals')) = u(g(H' <= Goals')) = u(H <= Goals).

Though Dk+1 is the union of Dk and (5) and D'k+1 is the union of D'k and the abstraction of (6) where / means set union, the abstraction of (6) also covers (5). Thus D'k+1 covers Dk+1.

case 2 (instantiation step)

Suppose the (k+1)-th step is the instantiation step both for E and E'.
There exists the derived clause (7) in Dk which is the candidate clause for the instantiation.

H <= G,Goals (7).

And there exist the derived clause (8) in Dk' which covers the derived clause (7) and is the candidate clause of the instantiation (renaming of the variables is implicit).

H' <= G',Goals' (8).

Thus, for some substitution g, (H <= G,Goals) = g(H' <= G',Goals').

Suppose there exists the set of clauses Cls and Cls' such that

Cls = {Cl| Cl is a definite clause in P whose head is unifiable with G}
Cls' = {Cl'| Cl' is a definite clause in P whose head is unifiable with G'}.

They are the input clauses at the (k+1)-th step for instantiating G and G' and their derived clauses are generated. Let DC be the set of derived clauses for Cls, and DC' be the set of derived clauses for Cls'. Similarly as in the case 1, it is proved that DC' covers DC.

Let dc be a derived clause in DC and dc' be its cover in DC'.

Considering the covering (subsumption) and the abstraction on DC' / D'k, D'k+1 covers Dk+1. □

As a consequence of the above lemma, the following theorem can be given.

Theorem:

The set of link terms L' computed by Earley deduction with term abstraction is a link set for a grammar G.

Proof:

The set of derived clause D contains the subset L of all provable instance of the goal link(X,Y). The set of derived clause D' contains the finite subset L' which covers L. Thus, L' satisfies the definition of a link set and L' is a link set for G. □

5. Generation of Strong Link Set

This section discusses properties of strong link set and a method for generating a strong link set from a given link set and grammar.

5.1 Strong Link Set Properties

The link computation method described in the previous section guarantees that it produces a link set for any grammar. For example, consider a grammar containing the following rules.

```
starting literals: {s(_)}
rules:
 s(s(Np,Vp)) => np(Np),vp(Vp).
 np(np1(Noun))=> noun(Noun).
 np(np2(Np1,Np2)) => np(Np1), [and], np(Np2).
                :
                :
```

Consider the link clauses for this grammar obtained by pre-evaluation.

```
(l1) link(np(np2(A,B)),s(s(np2(A,B),C))).
(l2) link(np(np2(A,B)),s(s(np2(C,D),E))).
(l3) link(np(np1(A)),s(s(np1(A),B))).
(l4) link(np(np1(A)),s(s(np2(B,C),D))).
```

(l1) and (l2) are link relations between 'np' with two arguments and 's', while (l3) and (l4) are those between 'np' with one argument and 's'. (l2) and (l4) are obtained by term abstraction during the pre-evaluation.

Consider the following BUP clause with direct link use obtained from the third grammar rule.

```
np(Goal,[Np1],S0,S,Arg) :-
     link(np(np2(Np1,Np2)),Goal),
     S0 = [and|S1],
     goal(np(Np2),[Np2],S1,S2)
     np(Goal,[Np1,Np2],S2,S,Arg).
```

When this clause is called, if the variable 'Goal' and 'Np1' is instantiated to 's(X)' and 'john' respectively, the 'calling pattern' of the link clause is 'link(np(np2(john,Np2)),s(X))'. Since both alternatives (l1) and (l2) successes, there can be equivalent two parses caused by these alternatives. A strong link set must not cause this type of redundancies in a parse. We call this property a 'uniqueness' property of a link set.

Consider the following BUP clause with direct link use obtained from the second grammar rule.

```
noun(Goal,[Noun],S0,S1,Arg) :-
     link(np(np1(Noun)),Goal),
     np(Goal,[np1(Noun)],S0,S1,Arg).
```

When this clause is called, if the variable 'Goal' and 'Noun' is instantiated to 's(X)' and 'john' respectively, the calling pattern is 'link(np(np(john)),s(X))'. Since this is unifiable with both (l1) and (l2), there are two disjoint alternatives. At least one of these alternatives will eventually succeeds. Therefore (l3) and (l4) preserve the uniqueness. However, a backtrack caused by these link clauses causes an efficiency problem. If there are no alternatives in link execution, this problem is avoided. We call this property a 'determinacy' property of link set.

The check of determinacy property is requires an analysis of calling patterns. For example, if the variable 'Goal' in the above explanation is always instantiated to either 's(np(X),Y)' or 's(np(X,Y),Z)', then (l1) and (l2) do not violate determinacy.

From the discussion above, a strong link set should have the following properties.

(1) Completeness and soundness
(2) Uniqueness
(3) Determinacy

Obviously the determinacy condition entails the uniqueness condition.

5.2 Link Calling Pattern in BUP

To define a strong link set formally, we introduce the set of link calling pattern of a grammar.

Definition: link calling pattern set of a grammar G

Let G be a grammar and I be a non-negative integer, link calling pattern set LCP(G,I) is defined as follows:

LCP(G,I) = {link(t1,t2) | t1 is the head of a I-dotted derived rule and t2 is a element of starting literal set or the dotted literal of a dotted derived rule.}

Considering the link call position in a BUP clause, LCP(G,1) corresponds to a link calling pattern of BUP (Fig 2). we call LCP(G,1) BLCP(G).

The following is the definition of strong link set:

Definition: Strong Link Set

Let G be a grammar. A link set L is a strong link set of G, iff for every link clause l in BLCP(G) there exists at most one link clause l1 such that @l = @l1.

Link calling pattern set is usually a infinite set. We refer to a finite set which covers link calling pattern set BLCP(G) as CBLCP(G) (Cover of BLCP(G)).

5.3 Least Common Cover

Before explaining the generation method, we define a least common cover (lcc) for two arbitrary terms.

Definition : Least Common Cover of two terms

The term c is a least common cover of the terms t1 and t2 iff c covers t1 and t2, and for any cover c' of t1 and t2, there exists a substitution @ such that @c' = c.

The following algorithm generates the least common cover for two terms. In the algorithm we introduce a new variable X(t1,t2) for each pair of terms t1 and t2.

Algorithm 1:
LCC(t1,t2) (the lcc of term t1 and t2) is recursively defined as follows:

1. If either t1 or t2 is a variable then LCC(t1,t2) = X(t1,t2).
2. If t1 and t2 differ either in their principal functors or in their arities, then LCC(t1,t2) = X(t1,t2).
3. If t1 = f(t11,..,t1n) and t2 = f(t21,...,t2n) then LCC(t1,t2) = f(LCC(t11,t21),...,LCC(t1n,t2n)).

The proof of the correctness of this algorithm is given in Appendix C. Some examples of lcc's are shown below.

t1	t2	LCC(t1,t2)
a	b	X
f(a)	f(X)	f(V)
f(g(X),g(X))	f(a,a)	f(V,V)
f(g(X),g(X))	f(g(L),g(M))	f(g(V1),g(V2))

5.4 Algorithm for Generating a Strong Link Set

As shown in 5.1, generation of a strong link set requires calling pattern analysis. In this section we present a method for obtaining a strong link set of a grammar G from an L(G) and a CBLCP(G). The following is the definition of the algorithm.

Algorithm 2:
Let G be a grammar. Given link set L=L(G) and a cover of the calling pattern CBLCP=CBLCP(G), repeatedly perform the following transformation. If no transformation applies, stop.

Select a link clause l from CBLCP and if there exist two link clauses l1, l2 in L such that @1l1 = @1l and @2l2 = @2l then replace l1 and l2 with LCC(l1,l2).

Theorem:

Let the original link set L has been reduced to L' when the algorithm stops. The link set L' is a strong link set of G.

Proof:

Since the replace operation in the algorithm obviously preserve the link set property, L' is a link set of G just as L is (Link set).

For any link clause l in BLCP(G), there exists lc in CBLCP such that lc covers l. If there exist link clauses l1 and l2 in L' such that @1l=@1l1 and @2l=@2l2, then @3lc=@3l1 and @4lc=@4l2. In this case, l1 and l2 are reduced to LCC(l1,l2) by the definition of the algorithm. Therefore there exists at most one l' in L' such that @l=@l'. (Determinacy) □

This algorithm produces a strong link set of a grammar if a link set and a cover of the BUP link calling pattern set of the grammar are available. Since a link set of a given grammar can be computed in terms of the method in 4, we can get a strong link set by establishing the method for computing a CBLCP(G) of a grammar. It is also possible to apply pre-evaluation to get a CBLCP(G) of a grammar G. The computation of CLCP(G,I) is described in Appendix B.

6. Concluding Remarks

This paper presented an extension of link relations in a logic grammar which are predictions used in a bottom-up parser BUP. The basic idea of the extension is to incorporate the arguments of a nonterminal with link relations. A link set is obtained by the pre-evaluation of a grammar by using a modified Earley deduction system. There are two usages of the extended link relations, the indirect use and the direct use. In particular, the direct use of a link set has the advantage of passing the parts of the argument in a top-down mannar. In order to guarantee the uniqueness and determinacy of the direct use, runtime analysis of the parser, that is extracting the calling pattern of the nonterminals, and term abstraction method are required. In general, the combination of the pre-evaluation of a grammar results in an efficient parsing algorithm. The correctness of both usages of a link set was also demonstrated.

A example of a link set for a grammar is shown in Appendix D.

In this paper, the treatment of gap was not discussed. But it is easy to modify the link computation program to take into account gaps. In fact, it is used to generate the link set shown in Appendix D.

This paper described the first step in the research on extracting grammatical information from logic grammar by using pre-evaluation system based on a general proof procedure for definite clauses, that is modified Earley deduction system. There remains several possibilities along this direction. More specific runtime analysis would result in a more efficient parsing algorithm. For example, a BUP interpreter based on the modified Earley deduction system can be used to get more specific runtime information to optimize the BUP parsing algorithm. Another possibility is to incorporate lookahead strategy into BUP

and extract the grammatical information with it.

Acknowledgement

This paper originated from the cooperative work with Dr. F.Pereira during his stay at ICOT. He proposed many of the basic ideas described in above, i.e., the extraction of grammar information by Earley deduction and its application to the BUP system. We would also like to thank Dr. Kanamori for his helpful comments, and Dr. Fuchi (the director of ICOT) and Dr. Yokoi (manager of ICOT's Third laboratory) for giving us the oppotunity to work with Dr. F.Pereira.

References

[1] Sato,T. and Tamaki,H., Enumeration of Success Patterns in Logic Programs, Theoretical Computer Science 34, 1984.

[2] Tamaki,H. and Sato,T., A Transformation System for Logic Programs which Preserves Equivalency, ICOT Technical Report TR-018, 1983.

[3] Pereira,F.C.N., Using Earley deduction to Compute Properties of Logic Grammars, private communication

[4] Matsumoto,Y. et. al., BUP: A Bottom-Up Parser Embedded in Prolog, New Generation Computing, vol.2, OHM-Springer, 1983.

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

[6] Aho,A.V. and Ullman,J.D., The Theory of Parsing, Translating, and Compiling, Vols. I and II, Englewood Cliffs, N.J.: Prentice Hall, 1972 and 1973.

[7] Pereira,F.C.N, Parsing as Deduction, Technical Note 295, Artificial Intelligence Center, SRI International, Menro Park, California, 1983.

Appendices

Appendix A. Abstraction Methods

As stated in 4.2, several methods can be used to abstract the derived clauses in Earley deduction. Here one possible solution, k-level abstraction is described in addition to the weakening.

The k-level abstraction of a term was proposed by Sato and Tamaki for the purpose of enumerating the success patterns of logic programs [1]. The k-level abstraction of a term is defined in terms of the level of the term.

Definition: level of a term

Let t ge an arbitrary term.

(1) For a given term t, t has level 0. t is called a level0 subterm of t.
(2) If the subterm f(t1,...,tn) of t has level k, then each ti (i = 1,...,n) has level k+1. Each ti is called a level k+1 term of t.

The k-level abstraction of a given term t and integer k is done by replacing every level k subterm of t by a free variable. The resulting term obviously covers t.

For example, 2-abstraction of the term f(g(X,a),Y,B) is f(g(V1,V2),Y,b).

The number of derived clauses abstracted by k-level abstraction is finite for a given integer k, because, the number of given definite clauses and the number of symbols in the domain of the definite clauses are finite.

The k-level abstraction method is used to compute the calling pattern of nonterminals in a given grammar G.

Appendix B. Computation of CLCP(G)

In general, computation of the cover of link calling patterns CLCP(G) of a grammar G amounts to computing all the possible derived rules. Currently, Earley deduction with k-level abstraction Ek is used to compute calling patterns of nonterminals.

The Earley deduction system with the k-level abstraction method differs from the original Earley deduction system in the following two points.

(1) each derived clause is in the form of a dotted derived rule, that is, a(Y) <= b(b),*,c(b,Y), while a(Y) <= c(b,Y) in the case of the original Earley deduction system.
(2) each literal in the derived clause is abstracted at the level k.

Thus, Ek can be used to get a cover of derived rule set of a grammar G, by giving it the definite clauses corresponding to G.

There is a tradeoff. The choice of the integer k significantly affects the amount of the computation and the information given by the computation. Currently, k is set to the deepest literal level in all the grammar rules in a given grammar.

By the definition of LCP(G) and the fact that Ek gives a cover of the derived rule set of G, CLCP(G) is directly computed.

Appendix C. The correctness of Algorithm 1

Lemma:
Let @1 = {Xi/ai} and @2 = {Xi/bi} be substitutions. If we let @ be {Xi/LCC(ai,bi)} then LCC(@1t,@2t) = @t for any term t.

Proof:
By induction on the structure of t.

1. If t is a variable then
   LCC(@1t,@2t) = t = @t when t is not in {Xi}
   LCC(@1t,@2t) = LCC(ai,bi) = @t when t = Xi
2. If t = f(t1,...,tn) then
   LCC(@1t,@2t) = LCC(f(@1t1,...,@1tn),f(@2t1,...,@2t1))
   = f(LCC(@1t1,@2t1),...,LCC(@1tn,@2tn))

By induction hypothesis LCC(@1ti,@2ti) = @ti.
Therefore, LCC(@t1,@t2) = f(@t1,...,@tn) = @t []

Theorem:
LCC(t1,t2) is the least common cover of t1 and t2.

Proof:
Let @1 = {X(a1,a2)/a1} and @2 = {X(a1,a2)/a2} be substitutions. Clearly @1LCC(t1,t2) = t1 and @2LCC(t1,t2) = t2. Therefore LCC(t1,t2) is a common cover of t1 and t2.

On the other hand, if g is a common cover of t1 and t2 then there exist @1 and @2 such that @1g = t1 and @2g = t2. Let us consider the substitution @ in the above lemma. Then @g = LCC(@1g,@2g) = LCC(t1,t2). Therefore, LCC(t1,t2) is a least common cover. []

Appendix D. Example of a link computation

D-1. Grammar

```
Starting terms : s(_)
Rules :

s(Result) => sentence(Result,nogap).
sentence(sentence(Np,Vp),nogap) => np(Np,Num,nogap), vp(Vp,Num).
```

```
sentence(sentence(Np,Vp),gap(Num)) => np(Np,Num,gap),vp(Vp,Num).
np(np(gap),_,gap) => [].
np(np(Noun),Num,nogap) => noun(Noun,Num).
np(np(Np,Srel),Num,nogap) => np(Np,Num,nogap), srel(Srel,Num).
srel(srel(Sent),Num) => sentence(Sent,gap(Num)).
vp(vp(Verb),Num) => verb1(Verb,Num).
verb1(Verb,Num) => verb(Verb,Num).
verb(walks,sin) => [walks].
verb(walk,_) => [walk].
noun(john,sin) => [john].
noun(they,plu) => [they].
```

D-2. Link Set obtained by the link computation program

```
link(noun(A,B,C),noun(A,B,C)).
link(noun(A,B,C),np(np(A),B,C,nogap)).
link(noun(A,B,C),np(np(D,E),B,C,nogap)).
link(noun(A,B,C),s(sentence(np(A),D))).
link(noun(A,B,C),s(sentence(np(D,E),F))).
link(noun(A,B,C),sentence(sentence(np(A),D),nogap)).
link(noun(A,B,C),sentence(sentence(np(D,E),F),nogap)).
link(np(A,B,C,D),np(A,B,C,D)).
link(np(np(A),B,C,nogap),np(np(D,E),B,C,nogap)).
link(np(np(A),B,C,nogap),np(np(np(A),D),B,C,nogap)).
link(np(np(A),B,C,nogap),s(sentence(np(A),D))).
link(np(np(A),B,C,nogap),s(sentence(np(D,E),F))).
link(np(np(A),B,C,nogap),sentence(sentence(np(A),D),nogap)).
link(np(np(A),B,C,nogap),sentence(sentence(np(D,E),F),nogap)).
link(np(np(A,B),C,D,nogap),np(np(E,F),C,D,nogap)).
link(np(np(A,B),C,D,nogap),s(sentence(np(E,F),G))).
link(np(np(A,B),C,D,nogap),
     sentence(sentence(np(A,B),E),nogap)).
link(np(np(A,B),C,D,nogap),
     sentence(sentence(np(E,F),G),nogap)).
link(np(np(gap),A,B,gap),
     sentence(sentence(np(gap),C),gap(A,B))).
link(np(np(gap),A,B,gap),
     srel(srel(sentence(np(gap),C)),gap(A,B))).
link(s(A),s(A)).
link(sentence(A,B),sentence(A,B)).
link(sentence(sentence(A,B),gap(C,D)),
     srel(srel(sentence(A,B)),gap(C,D))).
link(sentence(sentence(A,B),nogap),s(sentence(A,B))).
link(srel(A,B),srel(A,B)).
link(verb(A,B,C),sentence(sentence(np(gap),vp(A)),gap(B,C))).
link(verb(A,B,C),srel(srel(sentence(np(gap),vp(A))),gap(B,C))).
link(verb(A,B,C),verb(A,B,C)).
link(verb(A,B,C),verb1(A,B,C)).
link(verb(A,B,C),vp(vp(A),B,C)).
link(verb1(A,B,C),sentence(sentence(np(gap),vp(A)),gap(B,C))).
link(verb1(A,B,C),srel(srel(sentence(np(gap),vp(A))),gap(B,C))).
link(verb1(A,B,C),verb1(A,B,C)).
```

```
link(verb1(A,B,C),vp(vp(A),B,C)).
link(vp(A,B,C),vp(A,B,C)).
link(vp(vp(A),B,C),sentence(sentence(np(gap),vp(A)),gap(B,C))).
link(vp(vp(A),B,C),srel(srel(sentence(np(gap),vp(A))),gap(B,C))).
```

D-5. Strong Link Set

```
link(noun(A,B,C),noun(A,B,C)).
link(noun(A,B,C),np(D,B,C,nogap)).
link(noun(A,B,C),s(sentence(D,E))).
link(noun(A,B,C),sentence(sentence(D,E),nogap)).
link(np(A,B,C,D),np(E,B,C,D)).
link(np(np(A),B,C,nogap),s(sentence(D,E))).
link(np(np(A),B,C,nogap),sentence(sentence(D,E),nogap)).
link(np(np(A,B),C,D,nogap),s(sentence(np(E,F),G))).
link(np(np(A,B),C,D,nogap),sentence(sentence(np(E,F),G),nogap)).
link(np(np(gap),A,B,gap),sentence(sentence(np(gap),C),gap(A,B))).
link(np(np(gap),A,B,gap),
     srel(srel(sentence(np(gap),C)),gap(A,B))).
link(s(A),s(A)).
link(sentence(A,B),sentence(A,B)).
link(sentence(sentence(A,B),gap(C,D)),
     srel(srel(sentence(A,B)),gap(C,D))).
link(sentence(sentence(A,B),nogap),s(sentence(A,B))).
link(srel(A,B),srel(A,B)).
link(verb(A,B,C),sentence(sentence(np(gap),vp(A)),gap(B,C))).
link(verb(A,B,C),srel(srel(sentence(np(gap),vp(A))),gap(B,C))).
link(verb(A,B,C),verb(A,B,C)).
link(verb(A,B,C),verb1(A,B,C)).
link(verb(A,B,C),vp(vp(A),B,C)).
link(verb1(A,B,C),sentence(sentence(np(gap),vp(A)),gap(B,C))).
link(verb1(A,B,C),srel(srel(sentence(np(gap),vp(A))),gap(B,C))).
link(verb1(A,B,C),verb1(A,B,C)).
link(verb1(A,B,C),vp(vp(A),B,C)).
link(vp(A,B,C),vp(A,B,C)).
link(vp(vp(A),B,C),sentence(sentence(np(gap),vp(A)),gap(B,C))).
link(vp(vp(A),B,C),srel(srel(sentence(np(gap),vp(A))),gap(B,C))).
```

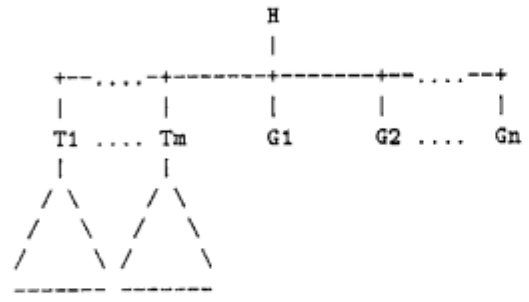

Fig 1 The tree representation of a m-dotted derived rule
H => T1,...,Tm,*,G1,...,Gn

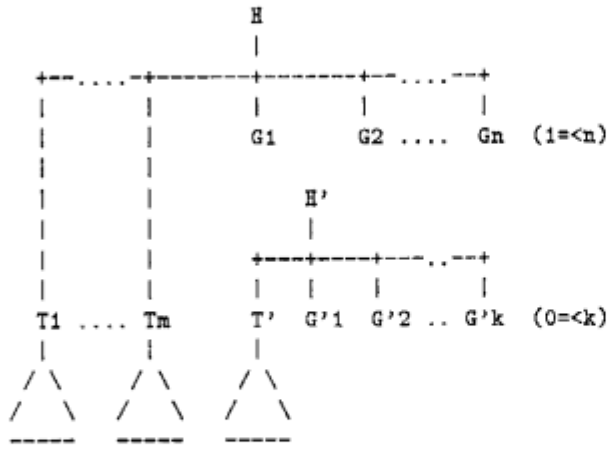

Fig 2 BUP link calling pattern link(H',G1)