ICOT Technical Report: TR-128

TR-128

The Correctness of Two Traslation Methods from Definit Clause Grammers into Prolog Programs

by
K. Sakai, H. Hirakawa, Y. Tanaka
and H. Yasukawa

July, 1985

ICOT

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

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

The Correctness of Two Translation Methods from Definite Clause Grammars into Prolog Programs

K. Sakai, H. Hirakawa, Y. Tanaka, H. Yasukawa

ICOT Research Center
Tokyo, Japan

ABSTRACT

A least fixed point semantics is given for definite clause grammar similar to that for the logic programming language Prolog. Two translation methods, namely, top-down and bottom-up translation from grammars into Prolog are described and their correctness and completeness are shown with respect to least fixed point semantics. Finally, the termination condition of the bottom-up parser is discussed.

Keywords: definite clause grammar, Prolog, least fixed point semantics, parser, correctness, completeness, termination.

1. Introduction

Grammar description and parsing are very important problems for natural and/or artificial language processing. Prolog is a promising programming language for implementing language processing systems [2, 4] since a grammar description language can be naturally introduced as an extention of Prolog itself and parsing for a given grammar can be treated as an extention of program execution.

Definite clause grammar (DCG) [4] is a typical Prolog-based grammar description and parsing system. DCG is a natural extention of context free grammar, which is now very often used to describe grammars for various languages. Besides, since nonterminal symbols in DCG are allowed to have parameters, we can express all kinds of control. For example, adjustment of gender, tense, number, and even the number of occurences of some terminal symbols. Therefore, it provides us with a familiar and powerful way to describe grammars. Moreover, since parsing is a form of extended program execution, the basic mechanism of Prolog execution is fully utilized for efficient parsing.

As stated above, what is refered to as DCG actually contains two ideas; grammar description and parsing. In this paper, however, we will discuss them separately. In what follows, DCG will be used to mean the grammar description only. The parsing technique is refered to as the top-down parsing of DCG.

Since an ordinary Prolog system is based on depth-first search, top-down parsing, which simply uses the Prolog execution mechanism, often falls into infinite loops if there is a left recursive rule. In writing a grammar, however, left recursive rules are inevitable both for natural languages and formal languages.

To relieve DCG from this shortcoming Matsumoto and others proposed another translation method for DCG [3]. Making use of the parsers generated by this method, called the bottom-up parser (BUP), we can avoid infinite loops caused by left recursive rules.

Many experiments have confirmed the correctness, efficiency, and termination of BUPs. The correctness of this translation method, however, is not very clear from the theoretical point of view and there has not been much study of termination of BUPs. In this paper, we will show the correctness of a bottom-up translation method for DCG, which is essentially equivalent to Matsumoto's, as well as the correctness of the top-down parsing. Next, we will prove that the translation method generates a terminating parser for a grammar under a certain condition. The condition is very natural and introduces few constraints on actually writing a grammar.

2. Definite Clause Grammar

Definite clause grammar (DCG) was suggested together with a translation method to Prolog programs [4] from the resemblance between sentence generation by context free grammars and the execution of programs written in Prolog. The framework of DCG is described in this section.

Assume that the following disjoint sets of symbols are given.

(1) An enumerable set of *variables* (denoted by strings beginning with an uppercase letter)

(2) A finite set of *function symbols* (denoted by strings beginning with a lowercase letter)

(3) A finite set of *predicate symbols* (denoted by strings beginning with a lowercase letter)

(4) A finite set of *terminal symbols* (denoted by boldface letters)

As usual, we define *terms*, *atomic formulas (atoms)*, and *clauses*. Terminal symbols and atoms are called *syntactic elements* and denoted by greek letters (with or without suffix). A *DCG* is a pair (R, S) such that

(1) R is a set of *syntactic rules* of the form

$$\alpha \rightarrow \alpha_1 \ldots \alpha_n \qquad (n >= 0)$$

where α is an atom and each α_i is a syntactic element.

(2) S is a finite set of atoms called the *starting atoms.*

An atom plays the role of a nonterminal symbol in ordinary context free grammar. Let $r = \alpha \rightarrow \alpha_1 \ldots \alpha_n$ be a rule and $s = \beta_1 \ldots \beta_{i-1}\beta_i\beta_{i+1} \ldots \beta_m$ be a string of syntactic elements. If α and β_i have a common instance, i.e. there exist substitutions θ_1 and θ_2 such that $\theta_1\alpha = \theta_2\beta_i$, we will denote it by

$$s \Rightarrow_{r,\theta_1,\theta_2} s'$$

where $s' = \theta_2\beta_1 \ldots \theta_2\beta_{i-1}\theta_1\alpha_1 \ldots \theta_1\alpha_n\theta_2\beta_{i+1} \ldots \theta_2\beta_n$. By appropriately renaming variables in r, we can select the same substitution θ for both θ_1 and θ_2. Then the above can be written as

$$s \Rightarrow_{r,\theta} \theta\beta_1 \ldots \theta\beta_{i-1}\theta\alpha_1 \ldots \theta\alpha_n\theta\beta_{i+1} \ldots \theta\beta_n$$

In what follows, renaming variables in a rule is often implicit and notation like the above is used directly.

For a given set R of rules, $s \Rightarrow s'$ means there exist a rule $r \in R$ and a substitution θ such that $s \Rightarrow_{r,\theta} s'$, and the symbol $\overset{*}{\Rightarrow}$ represents the reflexive transitive closure of $\Rightarrow$. If $s \overset{*}{\Rightarrow} s'$, s' is said to be *derived* from s by R.

The language $L(G)$ generated by a grammar $G = (R, S)$ is defined as follows:

$$L(G) = \{w \mid w \text{ is a string of terminal symbols only and } \gamma \overset{*}{\Rightarrow} w \text{ for some } \gamma \in S\}$$

Example 2.1

Let G consist of the starting atom s and the rules

$$\begin{aligned} s &\rightarrow a(X)b(X)c(X), \\ a(1) &\rightarrow \mathbf{a}, \\ a(s(X)) &\rightarrow \mathbf{a}a(X), \\ b(1) &\rightarrow \mathbf{b}, \\ b(s(X)) &\rightarrow \mathbf{b}b(X), \\ c(1) &\rightarrow \mathbf{c}, \\ c(s(X)) &\rightarrow \mathbf{c}c(X). \end{aligned}$$

Then,

$$L(G) = \{\mathbf{a}^i\mathbf{b}^i\mathbf{c}^i \mid i > 0\} \quad \blacksquare$$

Remark. If all the atoms in a grammar G are ground, i.e. have no variables, then G is equivalent to a context free grammar and, therefore, $L(G)$ is a context free language.

Remark. If all the rules in R have no terminal symbols, then R is a Prolog program and $\{\gamma \mid \gamma \overset{*}{\Rightarrow} \epsilon$ (the empty string) $\}$ is the set of all successful queries.

3. Herbrand Universe and Herbrand Base

Van Emden and Kowalski proposed the least fixed point semantics for Prolog programs and showed that it agree with the operational semantics by SL-resolution [5]. In this section, the least fixed point semantics for DCG is defined. The reader can easily see that it includes the least fixed point semantics for Prolog programs as a special case.

The Herbrand universe HU is the set of all ground terms. For simplicity, we assume that HU is not empty, i.e., there exists at least one 0-ary function symbol (constant symbol). The Herbrand base HB is the set of all ground atoms. We will define the extended Herbrand base HB' for DCG as follows:

$$HB' = \{\alpha \mapsto w \mid \alpha \text{ is a ground atom and } w \text{ is a string of terminal symbols}\}$$

For a given set D of definite clauses, the set function T_D on the power set $\mathcal{P}(HB)$ of HB is defined as follows:

$$T_D(H) = \{\theta\alpha \in HB \mid \text{there exist a definite clause } \alpha \to \alpha_1 \ldots \alpha_n \text{ and a substitution } \theta \text{ such that each } \theta\alpha_i \text{ is in } H\}$$

Similarly, for a given rule set R, the set function T_R on the power set $\mathcal{P}(HB')$ of HB' is defined as follows:

$$T_R(H) = \{\theta\alpha \mapsto w_1 \ldots w_n \in HB' \mid \text{there exist a rule } \alpha \to \alpha_1 \ldots \alpha_n \text{ and a substitution } \theta \text{ such that, for each } i, \alpha_i \text{ is a terminal symbol and } w_i = \alpha_i, \quad \text{or} \quad \alpha_i \text{ is an atom and } \theta\alpha_i \mapsto w_i \in H\}$$

The function T_D is clearly monotone, i.e. for all H and H' if $H \subseteq H'$ then $T_D(H) \subseteq T_D(H')$, and continuous, i.e.

$$\bigcup_{i=1}^{\infty} T_D(H_i) = T_D(\bigcup_{i=1}^{\infty} H_i) \qquad \text{for any sequence } H_1 \subseteq H_2 \subseteq \cdots,$$

and so is T_R.

Therefore, T_D has the least fixed point M_D, i.e.

(1) $M_D = T_D(M_D)$ and

(2) $M_D \subseteq H$ for all sets H satisfying $H = T_D(H)$

and T_R also has the least fixed point M_R. Moreover, $M_D[M_R]$ is characterized also as the least set H such that $T_D(H)[T_R(H)] \subseteq H$ and is computed by

$$M_D = \bigcup_{i=1}^{\infty} {T_D}^i(\emptyset) \qquad [\quad M_R = \bigcup_{i=1}^{\infty} {T_R}^i(\emptyset) \quad].$$

The least fixed point semantic M_D of a Prolog program agrees with its operational semantics [5]. In the terms defined above, $\gamma \overset{*}{\Rightarrow} \epsilon$ if and only if $\theta\gamma \in M_D$ for some substitution θ. We have a similar theorem for DCG.

Theorem 3.1

For any atom γ and any string w of terminal symbols, $\gamma \overset{*}{\Rightarrow} w$ if and only if $\theta\gamma \mapsto w \in M_R$ for some substitution θ. ▮

This theorem can be easily shown in a similar way to [5] and the following corollary connects between the least fixed point semantics M_R and the operational semantics $L(G)$.

Corollary 3.2

For any grammar $G = (R, S)$,

$$L(G) = \{ w \mid \text{ there exists an atom } \gamma \in S \text{ such that } \theta\gamma \mapsto w \in M_R \text{ for some substitution } \theta \} \quad ▮$$

4. Top-down DCG translation

As stated before, the rule set of a DCG can be considered as an extension of a Prolog program and the starting atoms as queries. However, the extended features can be embedded into ordinary Prolog. In this section we discuss an embedding method, now a standard part of this kind of approach [1,4]. A DCG rule set is translated to a Prolog program, used as both a parser and a generator. For this translation we need two new predicates $dcg(\alpha, w)$ and $eq(w, y)$. The arguement α is an atom and w and y are strings consisting of terminal symbols. Therefore, strictly speaking, a typed theory with two types is necessary in the following discussion. We will avoid the construction of such a theory, because there is little fear of confusion.

The translation *Td*, called the top-down translation in contrast to the bottom-up translation discussed in the next section, from rules to definite clauses is defined as follows:

For any rule $r = \alpha \rightarrow \alpha_1 \ldots \alpha_n$, let $X_1, \ldots, X_n$ be new variables not contained in r. Then $Td(r) = \alpha' \rightarrow \alpha'_1 \ldots \alpha'_n$, where

$$\alpha' = dcg(\alpha, X_1 \ldots X_n)$$

$$\alpha'_i = \begin{cases} dcg(\alpha_i, X_i), & \text{if } \alpha_i \text{ is an atom;} \\ eq(\alpha_i, X_i), & \text{if } \alpha_i \text{ is a terminal symbol.} \end{cases}$$

For a rule set R, let

$$Td(R) = \{Td(r) \mid r \in R\} \bigcup \{eq(X, X) \rightarrow \epsilon\}.$$

Theorem 4.1

Let R be a rule set and $D = Td(R)$. Then, for any ground atom γ and any string w of terminal symbols, $dcg(\gamma, w) \in M_D$ if and only if $\gamma \mapsto w \in M_R$.

Proof:

Only-if part: Let

$$EQ = \{eq(w, w) \mid w \text{ is a string of terminal symbols }\}$$

Since $dcg(\gamma, w)$ cannot be in EQ, it is sufficient to prove that

$$M_D \subseteq H = \{dcg(Q, w) \in HB \mid \gamma \mapsto w \in M_R\} \bigcup EQ.$$

From the remark about the least fixed point before Theorem 3.1, it is sufficient to prove that $T_D(H) \subseteq H$. Assume that $\gamma \in T_D(H)$. Then, there exists a substitution θ such that $\gamma = \theta eq(X, X)$ or there exist a rule $\alpha \rightarrow \alpha_1 \ldots \alpha_n$ and a substitution θ such that $\gamma = \theta\alpha'$ and each $\theta\alpha'_i$ is in H. In the former case, obviously $\gamma \in EQ$. In the latter case, for each i, α_i is a terminal symbol and $\theta X_i = \alpha_i$ or α_i is an atom and there exists w_i such that $\theta\alpha_i \mapsto \theta X_i$. Therefore $\theta\alpha \mapsto \theta X_1 \ldots \theta X_n \in T_R(M_R) = M_R$. Hence $\gamma = dcg(\theta\alpha, \theta X_1 \ldots \theta X_n) \in H$.

If part: Let

$$H = \{\gamma \mapsto w \in HB' \mid dcg(\gamma, w) \in M_D\}.$$

It is sufficient to prove that $T_R(H) \subseteq H$. Assume that $\gamma \mapsto w \in T_R(H)$. Then, there exist a rule $\alpha \rightarrow \alpha_1 \ldots \alpha_n$, a substitution θ, and strings $w_1, \ldots, w_n$ such that $\gamma = \theta\alpha$,

$w = w_1 \ldots w_n$, and for each i, α_i is a terminal symbol and $w_i = \alpha_i$, or α_i is an atom and $\theta\alpha_i \mapsto w_i \in H$. In either case, $\theta\alpha'_i \in M_D$ by setting $\theta X_i = w_i$. Therefore

$$\theta\alpha' = dcg(\theta\alpha, \theta X_1 \ldots \theta X_n) = dcg(\gamma, w) \in T_D(M_D) = M_D.$$

Hence $\gamma \mapsto w \in H$. ∎

Let $G = (R, S)$ be a grammar and γ be a starting atom. Then the above theorem guarantees that the query $dcg(\gamma, w)$ to the derived program $Td(R)$ judges whether w is a sentence in the grammatical category γ and the query $dcg(\gamma, X)$ generates sentences in γ.

Remark. Actually implemented Prolog systems do not provide the string type and, therefore, strings are usually expressed by a list in Prolog. In this case, concatenating a number of lists like $X_1 \ldots X_n$ is not efficient. α' and α'_i below with difference lists instead of ordinary lists are usually used in an actual implementation of the top-down translation.

$$\alpha' = dcg(\alpha, X_0, X_n)$$

$$\alpha'_i = \begin{cases} eq(X_{i-1}, [\alpha_i \mid X_i]), & \text{if } \alpha_i \text{ is a terminal symbol;} \\ dcg(\alpha_i, X_{i-1}, X_i), & \text{if } \alpha_i \text{ is an atom.} \end{cases}$$

5. Bottom-up DCG translation

In actually implemented Prolog systems, the resolution is not necessarily complete, because the depth-first search strategy for input clauses is usually employed. In fact, derived Prolog programs may fall into infinite loops if there is a left recursive rule such as $vp \rightarrow vp, adv$. In writing a grammar, however, left recursive rules are inevitable both for natural languages and formal languages.

In this section, we will discuss another translation method, which is essentially equivalent to the BUP method suggested by Matsumoto and others [3] in order to generate parsers free from such infinite loops. Making use of parsers generated by this method, we can avoid infinite loops caused by left recursive rules. We call the method bottom-up translation due to the similarity to the BUP. In compensation, we lose empty rules, i.e. rules whose right hand sides are empty, because they may lead parsing into an infinite loop. In spite of this disadvantage, translation methods like BUP are promising, because when writing a grammar empty rules can be dispensed with far more easily than left recursive rules.

For bottom-up translation, besides our old friend $eq(w, y)$ in the previous section, we will employ two new predicates $t(\alpha, w)$ and $nt(\alpha, \beta, w)$, where the type of the arguments α and β is syntactic element and that of w is string of terminal symbols.

Since there are no empty rules, a rule can have the form

$$\alpha \to \beta\alpha_1 \ldots \alpha_n \qquad (n >= 0).$$

This rule is translated into a clause

$$\beta' \to \alpha'_1 \ldots \alpha'_n \alpha'$$

where

$$\beta' = \begin{cases} t(G, \beta X_1 \ldots X_n Y), & \text{if } \beta \text{ is a terminal symbol} \\ nt(\beta, G, X_1 \ldots X_n Y), & \text{if } \beta \text{ is an atom} \end{cases}$$

$$\alpha'_i = \begin{cases} eq(\alpha_i, X_i), & \text{if } \alpha_i \text{ is a terminal symbol} \\ t(\alpha_i, X_i), & \text{if } \alpha_i \text{ is an atom} \end{cases}$$

$$\alpha' = nt(\alpha, G, Y),$$

for some new variables $X_1, \ldots, X_n, Y$.

We will denote this translation by Bu. For a rule set R without empty rules, let

$$Bu(R) = \{Bu(r) \mid r \in R\} \bigcup \{eq(X, X) \to \epsilon, nt(A, A, \epsilon) \to \epsilon\}.$$

Theorem 5.1

Let R be a rule set without empty rules and $D = Bu(R)$. Then, for any ground atom γ and any string w of terminal symbols, $t(\gamma, w) \in M_D$ if and only if $\gamma \mapsto w \in M_R$.

Proof:

Only-if part: Let

$$Ht = \{t(\gamma, w) \in HB \mid \gamma \mapsto w \in M_R\}$$

$$Hnt = \{nt(\alpha, \gamma, w) \in HB \mid \text{ for any string of terminal symbols } v \text{ if } \alpha \mapsto v \in M_R, \text{ then } \gamma \mapsto vw \in M_R\}$$

Since $t(\gamma, w)$ cannot be in EQ or Hnt, it is sufficient to show that $M_D \subseteq H = Ht \bigcup Hnt \bigcup EQ$, i.e. $T_D(H) \subseteq H$. Let $\gamma \in T_D(H)$. There are four cases.

Case 1. There exists a substitution θ such that $\theta eq(X, X) = \gamma$. In this case $\gamma \in EQ$ clearly.

Case 2. There exists a substitution θ such that $\theta nt(A, A, \epsilon) = \gamma$. In this case $\gamma \in Hnt$ clearly.

Case 3. There exist a rule $\alpha \to \beta\alpha_1 \ldots \alpha_n$ and a substitution θ such that β is a terminal symbol, $\gamma = t(\theta G, \beta\theta X_1 \ldots \theta X_n \theta Y)$, $\theta\alpha'_i$s and $\theta\alpha'$ are in H. For each i, α_i is a terminal symbol and $\alpha_i = \theta X_i$ or α_i is an atom and $\theta\alpha_i \mapsto \theta X_i \in M_R$ and, therefore, $\theta\alpha \mapsto \beta\theta X_1 \ldots \theta X_n \in M_R$. Since $\theta\alpha' \in Hnt$, $\theta G \mapsto \beta\theta X_1 \ldots \theta X_n \theta Y \in M_R$. Thus $\gamma \in Hn$.

Case 4. There exist a rule $\alpha \to \beta\alpha_1 \ldots \alpha_n$ and a substitution θ such that β is an atom, $\gamma = nt(\theta\beta, \theta G, \theta X_1 \ldots \theta X_n \theta Y)$, $\theta\alpha_i'$s and $\theta\alpha'$ are in H. Assume that $\theta\beta \mapsto v$. Then a discussion similar to the above shows that $\theta G \mapsto v\theta X_1 \ldots \theta X_n \theta Y \in M_R$. Thus $\gamma \in Hnt$.

If part: We will prove a stronger claim. If $\gamma \mapsto w \in M_R$ and $nt(\gamma, \gamma', y) \in M_D$, then $t(\gamma', wy) \in M_D$ for any γ, γ', w, and y. Since it is clear that $nt(\gamma, \gamma, \epsilon) \in M_D$ for any ground term γ. This claim guarantees that $t(\gamma, w) \in M_D$ if $\gamma \mapsto w \in M_R$. Let

$$H = \{\gamma \mapsto w \in HB' \mid \text{ if } nt(\gamma, \gamma', y) \in M_D, \text{ then } t(\gamma', wy) \in M_D \text{ for any } \gamma' \text{ and } y\}.$$

It is sufficient to prove that $T_R(H) \subseteq H$. Assume that $\gamma \mapsto w \in T_R(H)$. Then, there exist a rule $\alpha \to \beta(= \alpha_0)\alpha_1 \ldots \alpha_n$, a substitution θ, and strings $w_0, w_1, \ldots, w_n$ such that $\gamma = \theta\alpha$, $w = w_0 w_1 \ldots w_n$, and for each i, α_i is a terminal symbol and $w_i = \alpha_i$, or α_i is an atom and $\theta\alpha_i \mapsto w_i \in H$. In either case, $\theta\alpha_i' \in M_D$ by setting $\theta X_i = w_i$. Assume that $nt(\gamma, \gamma', y) \in M_D$. Since $\beta' \to \alpha_1' \ldots \alpha_n' \alpha' \in D$, if we let $\theta G = \gamma'$ and $\theta Y = y$, then $\theta\beta' \in M_D$. If β is a terminal symbol, $\theta\beta' = t(\gamma', w_0 w_1 \ldots w_n y) = t(\gamma', wy)$. If β is an atom, $\theta\beta' = nt(\theta\beta, \gamma', w_1 \ldots w_n y)$ and, since $\theta\beta \mapsto w_0 \in H$, $t(\gamma', w_0 w_1 \ldots w_n y) = t(\gamma', wy) \in M_D$. Thus, in either case, $\gamma \mapsto w$ is shown to be in H. ∎

Remark. The following β' and α' and α_i's, with difference lists instead of ordinary lists, should be used in an actual bottom-up translation into Prolog:

$$\beta' = \begin{cases} t(G, [\beta \mid X_0], Z), & \text{if } \beta \text{ is a terminal symbol} \\ nt(\beta, G, X_0, Z), & \text{if } \beta \text{ is an atom} \end{cases}$$

$$\alpha_i' = \begin{cases} eq(X_{i-1}, [\alpha_i \mid X_i]), & \text{if } \alpha_i \text{ is a terminal symbol} \\ t(\alpha_i, X_{i-1}, X_i), & \text{if } \alpha_i \text{ is an atom} \end{cases}$$

$$\alpha' = nt(\alpha, G, X_n, Z),$$

and

$$Bu(R) = \{Bu(r) \mid r \in R\} \bigcup \{eq(X, X) \to \epsilon, nt(A, A, X, X) \to \epsilon\}$$

Even with the bottom-up translated program (called BUP), parsing of a sentence does not necessarily terminate, since there may be a recursive rule such as $np \to np$. Now let us turn to the termination of BUPs.

Let D be a Prolog program and s be a query, i.e., a string of terms. A *D-execution sequence* for s is a finite sequence $s_0(= s) \Rightarrow s_1 \Rightarrow \cdots \Rightarrow s_n$ such that, for each $i (0 \le i < n)$, there exists a definite clause $\alpha \rightarrow \alpha_1 \ldots \alpha_m$ such that

$$s_i = \beta\beta_1 \ldots \beta_k \text{ and } s_{i+1} = \theta(\alpha_1 \ldots \alpha_m \beta_1 \ldots \beta_k)$$

where θ is the most general unifier of α and β.

A D-execution sequence of length n is successful if s_n is the empty string ϵ, and fails if it is not successful but has no extensions, i.e. $s_n = \beta\beta_1 \ldots \beta_k$ and there is no definite clause whose head is unifiable with β.

A Prolog program D is said to *terminate* with respect to a query s, if the set of all D-executions for s is finite. This property guarantees that, even by depth-first search, the program never falls into infinite loops, i.e., it finds a finite number of solutions or fails in a finite number of steps.

We will show the relative termination of the actual versions of BUP with difference lists. A *unit rule* is a rule of the form $\alpha \rightarrow \beta$ where β is an atom.

Theorem 5.2

Let R^* be the set of all the unit rules in R. If R^* (viewed as a Prolog program) terminates for any query, then $Bu(R)$ terminates for any query of the form $t(\gamma, L, [])$ where γ is a term, L is a list of terminal symbols, and $[]$ is the empty list.

Proof:

Let k be the maximum length of the R^*-execution sequences. By König's lemma, it is sufficient to show that the length of $Bu(R)$-execution sequences is bounded. Let $s_0 \Rightarrow s_1 \Rightarrow \cdots \Rightarrow s_n$ be any $Bu(R)$-execution sequence for $t(\gamma, L, [])$ which is not successful. Each s_i consists of three kinds of atoms, namely, atoms of the form

$$nt(\alpha, G, X, Y) \quad \text{or} \quad t(G, X, Y) \quad \text{or} \quad eq(X, [\alpha \mid Y]).$$

Let δ_i be the first atom of s_i and n_i be the number of atoms of the first form in s_i. Then it is easy to verify that, for each δ_i, X is a sublist L_i of L and, moreover,

(1) $L_{i+1} = L_i, n_{i+1} \leq n_i$ if δ_i is of the first form,

(2) $length(L_{i+1}) = length(L_i) - 1, n_{i+1} = n_i + 1$ if δ_i is of the second form,

(3) $length(L_{i+1}) = length(L_i) - 1, n_{i+1} = n_i$ if δ_i is of the third form.

Since in cases (2) and (3)

$$2length(L_{i+1}) + n_{i+1} < 2length(L_i) + n_i,$$

the number of δ_is of the second and third forms is not greater than $2length(L)$. Therefore, if the length of $Bu(R)$-execution sequences is not bounded and if we select a $Bu(R)$-execution sequence long enough, it contains a subsequence $s_j \Rightarrow s_{j+1} \Rightarrow \cdots \Rightarrow s_{j+k}$ such that $\delta_j, \delta_{j+1}, \ldots, \delta j + k$ are of the first form, $L_{j+i+1} = L_{j+i}$ and $n_{j+i+1} = n_{j+i}$ for all $i(0 \leq i < k)$. Since $n_{j+i+1} = n_{j+i}$ and δ_{j+i+1} is of the first form, s_{j+i+1} must have been obtained from s_{j+i} by a definite clause of the form

$$nt(\alpha_i, G, X_0, Z) \rightarrow nt(\beta_i, G, X_0, Z),$$

generated from the unit rule $\beta_i \rightarrow \alpha_i$. Then we can construct an R^*-execution sequence $\gamma_k \Rightarrow \gamma_{k-1} \Rightarrow \cdots \Rightarrow \gamma_1 \Rightarrow \gamma_0$ such that each γ_i is a common instance of α_i and β_{i-1} (we will omit the details of the construction). However, this contradicts the claim that k is the maximum length of the R^*-execution sequences. ∎

Since any context free language without the empty sentence has a grammar without empty rules or looped unit rules, we can generate a terminating parser for the language by bottom-up translation. In the general case, the termination condition of R^* does not seem to be very simple. In writing a practical DCG, however, we do not need unit rules with looped predicate symbols and the above theorem assures us that bottom-up translation generates a terminating parser for a DCG without such rules.

References

[1] Clocksin, W.F. and Mellish, C.S. (1981) Programming in Prolog, Springer-Verlag

[2] Colmerauer, A. (1978) "Metamorphosis Grammar," Natural language communication with computer (Bolc eds.), Lecture Notes in Computer Science, vol. 63, Springer-Verlag, 133-189

[3] Matsumoto, Y., Tanka, H., Hirakawa, H., Miyoshi, H., and Yasukawa, H. (1983) "BUP: A bottom-up parser embeded in Prolog," New generation Computing, vol. 2, 145-158

[4] Pereira, F.C.N. and Warren, D.H.D. (1980) "Definite clause grammar for language analysis – a survey of the formalism and a comparison with augmented transition networks," Artificial Intelligence 13, 231-278

[5] van Emden, M.H. and Kowalski, R.A. (1976) "The semantics of predicate logic as a programming language," Journal of the Association for Computing Machinery 23, No 4, 733-742