A Study on "Path" function in LALR Parsing Method

Lee, Myung-Joon
Dept. of Computer Science
(Received september 27, 1985)

⟨**Abstract**⟩

The set-valued function "Path" is presented recently as a new tool for the analysis of LALR parsing method.

In this paper, a new algorithm using Digraph is introduced for efficient computation of Path. Several characteristics of path is also studied with a new relation and an associated graph.

LALR Parsing 방법에서의 "Path" function에 관한 연구

이　　명　　준
전 자 계 산 학 과
(1985. 9. 27 접수)

⟨요　　약⟩

"Path" function은 LALR formalism에 대한 새로운 해석을 위한 도구로서 정의된 함수이다. 본 논문에서는 이 함수의 계산을 위하여 Digraph algorithm을 이용한 효율적인 계산방법이 고안되었으며, 이 함수의 여러가지 특성이 새로운 relation과 관계된 graph를 통하여 연구되었다.

I. Introduction

The set-valued function "Path" is presented recently as a new tool for the analysis of LALR parsing method[1]. The main result of this new analysis can be stated by the following formula.

$$\begin{aligned}&\mathrm{LA}_k(P,\ [A\longrightarrow\alpha_1\cdot\alpha_2])\\&=_s\{x\mid x\in\mathrm{Path}_k(A',\ A)\oplus_k\mathrm{FIRST}_k(\beta_2)\\&\qquad\oplus_k\mathrm{LA}_k(q,\\&\qquad\quad[B\rightarrow\beta_1.\ A'\beta_2]),\\&\quad q\in\mathrm{PRED}(q,\alpha_1),\ A_1\ L^*A,\\&\quad[B\longrightarrow\beta_1.\ A'\beta_2]\in K_q\}.\end{aligned}$$

Hence an efficient computation of Path_1 is necessary to compute LALR(1) lookahead sets efficiently.

Moreover Path has its own interesting characteristics.

But these are not been fully studied. In this paper, Path is studied deeply. In addition, a new computation algorithm for Path_1 is introduced by some new relations and Digraph algorithm[4].

II. Basic terminology and definitions

The basic terminology and definitions in this paper are consistent with those of Aho and Ullman [2,3]. Notational conventions are also stated.

A context-free grammar(*CFG*) is a quadruple (N,T,P,S), where $N,T,P,$ and S stand, respectively, for a set of nonterminal symbos, a set of terminal symbols, a set of productions (each of which is of the form $A \rightarrow \alpha$), and a start symbol in N. Given a grammar G, V(the vocabulary) stands for $N \cup T$ and V^* for the reflexive-transitive closure of V; the transitive closure is indicated by the superscript$^+$.

Lower-case Greek letters such as α, β, γ, and ω denote strings in V^*, lowercase Roman letters toward the beginning of the alphabet(a, b, and c) are terminals, whereas those near the end (u, v, and w) are strings in T^*; upper-case letters in the beginning of the alphabet (A, B, and C) are nonterminals, whereas those near the end (X, Y, and Z) are symbols in V. An empty string is denoted by Λ. Given a relation R, R^* stands for the reflecive-transitive closure of R; and R^+ for the transitive closure of R and R^{-1} for the inverse relation of R; and following notations are used: $R(A) = \{B \mid ARB\}$ and $R^{-1}(A) = \{B \mid BRA\}$.

We take for granted concepts such as FIRST$_k$ and $\ominus_k$. And RHS (or LHS) stands for the right-hand (or left-hand) side of a production or a formula.

The following concepts are introduced by [1].

[**Definition 2.1**] We define a left dependency relation

$$L \subseteq N \times N : A \; L \; B \text{ iff } A \rightarrow B\gamma \in P. \quad (2.1)$$

We also call the directed graph associated with relation L the L-graph. It is constructed by representing each instance of (2.1) as a pair of vertices connected by a directed edge:

$$\begin{tikzcd} \textcircled{A} \arrow[r, "\gamma"'] & \textcircled{B} \end{tikzcd}$$

Note that the edge has been labeled with the remaining part of the production RHS that follows B. The following definition is also from [1].

[**Definition 2.2**] For any nonterminals A and B,

$$\begin{aligned} \text{Path}_k\ (A, B) = \cup \{\text{FIRST}_k\ (\beta_n \cdots \beta_2 \beta_1) \mid\ & B_0 = A, \\ & B_n = B,\ n \geqslant 0,\ B_0 \rightarrow B_1 \beta_1 \in P, \\ & B_1 \rightarrow B_2 \beta_2 \in P, \cdots, \\ & B_{n-1} \rightarrow B_n \beta_n \in P\}, \qquad (2.2) \end{aligned}$$

where the sequence $\beta_1, \cdots, \beta_n$ describes a path from A to B in the L-graph of the form

$$\begin{tikzcd} \textcircled{A} \arrow[r, "\beta_1"'] \arrow[d, phantom, "\shortparallel"] & \bigcirc \arrow[r, "\beta_2"'] \arrow[d, phantom, "\shortparallel"] & \cdots\cdots \arrow[r, "\beta_n"'] & \textcircled{B} \arrow[d, phantom, "\shortparallel"] \\ B_0 & B_1 & & B_n \end{tikzcd}$$

In this formula, the $\cup$ stands for union overall such paths from A to B.

Now we redefine this concept for clarifying further discussion.

[**Definition 2.3**]

For any nonterminals A and B,

$$\text{Path}_k^1\ (A, B) = \{\text{FIRST}_k(u) \mid A \rightarrow Bu \in P\} \qquad (2.3)$$

$$\text{Path}_k^n(A, B) = \cup(\text{Path}_k^1\ (C_i, B) \oplus_k \text{Path}_k^{n-1}\ (A, C_i)) C_i \in \{C \mid A\, L^{n-1}\, C,\ CLB\} \qquad (2.4)$$

for $n \geqslant 2$.

$$\text{Path}_k\ (A, B) = \bigcup_{i=1}^{\infty} \text{Path}_k^i\ (A, B) \qquad (2.5)$$

Observe that a simple consideration give us the fact that formula (2.2) and formula (2.5) are equivalent. The formula (2.2) can be restated by

$$\begin{aligned} \text{Path}_k\ (A, B) = \cup \{\beta_n \ominus_k \beta_{n-1} \ominus_k \cdots \oplus_k \beta_2 \oplus_k \beta_1 \mid\ & B_0 = A,\ B_n = B,\ n \geqslant 1, \\ & \beta_i \in \text{Path}_k^1(B_{i-1},\ B_i) \text{ for } \\ & 1 \leqslant i \leqslant n\} \qquad (2.6) \end{aligned}$$

with the same descriptions in (2.2).

Now we define a new useful relation η.

[**Definition 2.4**]

$$A \; \eta \; B \text{ iff } \Lambda \in \text{Path}_k^1\ (A, B) \text{ for } k \geqslant 1. \quad (2.7)$$

In addition, notice that the followings are obviously true.

$$A \; L \; B \text{ iff } \text{Path}_k^1\ (A, B) \neq \phi \qquad (2.8)$$

$$A \; L \; B \text{ if } A \eta B. \qquad (2.9)$$

$$A \; \eta^+ \; B \text{ iff } \Lambda \in \text{Path}_k\ (A, B). \qquad (2.10)$$

$$A \; L^+ \; B \text{ iff } \text{Path}_k\ (A, B) \neq \phi. \qquad (2.11)$$

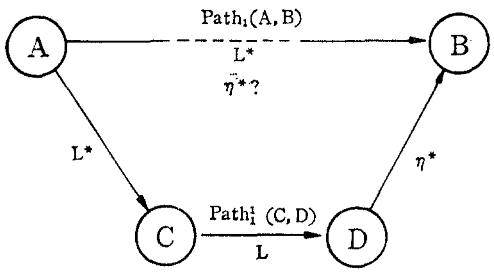
Ⅲ. Computing Path$_1$ by a direct method.

In this section, *a* useful theorem for computing Path directly is introduced, and also derived algorithm is stated.

[Theorem 3.1]

$$\mathrm{Path}_1(A,B) = \bigcup_{\substack{C\in L^*(A)\\ D\in\eta^{-1*}(B)\\ C\,L\,D}} (\mathrm{Path}'_1(C,D)-\{\Lambda\}) \cup \{\Lambda \mid A\eta^+B\} \quad (3.1)$$

Proof. First, we will prove LHS(3.1)$\subseteq$RHS (3.1). Assume $\Lambda\in\mathrm{Path}_1(A,B)$. Then $A\eta^+B$ by formula (2.10). Hence $\Lambda\in$RHS(3.1). Assume $a\in\mathrm{Path}_1(A,B)$. Then by formula (2.6), a must be some β_i, i.e., $a\in\mathrm{Path}'_1(B_{i-1}, B_i)$. Then the fact that $A\ L^*\ B_{i-1}$ and $B_{i-1}\ L\ B_i$ is simply true by the concept of L^* and(2.8). Further, since $\Lambda\in\mathrm{Path}_1\ (B_i,B)$ or $B_i=B$ by the property of $\oplus_1$ operator, $B_i\ \eta^*\ B$ is true by (2.10). Hence $a\in$RHS(3.1)



⟨**Fig. 3.1**⟩

Next let us prove LHS(3.1) $\supseteq$ RHS(3.1). If $\Lambda\in$RHS(3.1), then $A\ \eta^+\ B$.

Hence $\Lambda\in\mathrm{Path}_1(A,B)$. Assume $a\in$RHS(3.1). Then a must be contained some direct path, say, $\mathrm{Path}'_1\ (B_{i-1}, B_i)$ with nonterminals B_{i-1} and B_i satisfying appropriate conditions. Then figure (3.1) simply shows that $\mathrm{Path}'_1(B_{i-1},B_i) \subseteq\mathrm{Path}_1(A,B)$ by the property of η^* and formula (2.6).

Hence $a\in\mathrm{Path}_1(A,B)$. This completes the proof. The following formula is equivalent to formula (3.1) and more useful for computing Path.

$$\mathrm{Path}\ (A,B) = \bigcup_{\substack{C\in L^*(A)\\ D\in\eta^{-*}(B)\\ CLD}} \mathrm{Path}'_1(C,D)\cup\{\Lambda\mid A\eta^+B\} - \{\Lambda\mid \mathrm{not}(A\eta^+B\} \quad (3.2)$$

This formula yields the following algorithm for computing Path.

⟨**Algorithm 3.1**⟩

```
Compute $L^+$
Compute $\eta^+$
/* initially path$_1$ $(A,B)::=\phi$ */
for $A\in N$ do
for $B\in N$ do
if $A\ L^+\ B$ then
for $C\in L^*(A)$ do
for $D\in\eta^{-1*}(B)$ do
if $C\ L\ D$ then
Path$_1(A,B)$ : =Path$_1(A,B)\cup$Path$'_1$ $(C,D)$
endif
endfor
endfor
if $(A\ \eta^+\ B)$ then Path$_1(A,B)$
  : =Path$_1(A,B)\cup\{\Lambda\}$
else Path$_1(A,B)$: =Path$_1(A,B)-\{\Lambda\}$
  endif
endif
endif
endfor
endfor
```

Ⅳ. Computing Path by Traversing a Digraph

Algorithm 3.1 correctly computes Path, but the same thing is computed more than once in finding all Path. This inefficiency can be eliminated by the following formalisms and the derived computing algorithm.

[Lemma 4.1]

If $A\ L^+B,\ A\ L\ C,$ and $C\ L^+B,$

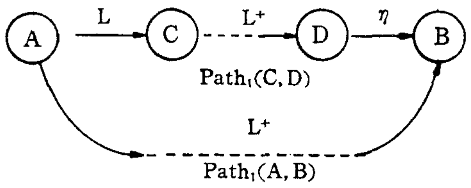
Then $\mathrm{Path}_1(A,B)\supseteq\mathrm{Path}_1(C,B)-\{\Lambda\}$

This Lemma is trivial from following figure and formula(2.6).

But it should be noted that when Path_1 (C,B) contains Λ, $\mathrm{Path}_1(A,B)$ include Path_1 (C,B) except Λ. $\mathrm{Path}_1(A,B)$ can contain Λ if and only if $A\ \eta^+\ B$.

[**Lemma 4.2**] If $A\ L^+\ B$, ALC, $C\ L^+\ D$ and $D\eta B$, Then $\mathrm{Path}_1(A,B) \supseteq \mathrm{Path}_1\ (C,D)-\{A\}$

This Lemma is also trivial from following figure and formula (2.6).



Notice that $\mathrm{Path}_1(A,B)$ includes $\mathrm{Path}_1(C,D)$ except A by the property of $\ominus_1$ operator and $\mathrm{Path}_1(D,B)$ contains A.

The above two inclusions can be captared by the following relation I on $(N\times N)\times(N\times N)$.

[**Definition 4.1**] For any nonterminals A,B, C and D, $(A,B)\ I\ (C,D)$ iff $A\ L^+B$, $A\ L\ C$, $C\ L^+\ D$ and $(D\ \eta\ B$ or $D=B)$.

Thus $\mathrm{Path}_1(A,B) \supseteq \mathrm{Path}_1(C,D)-\{A\}$ if (A,B) $I\ (C,D)$.

Now the results from above consideration can be stated in the following theorem.

[**Theorem 4.1**] For any nonterminals A,B,C and D,

$$\mathrm{Path}_1(A,B) = \overline{\mathrm{Path}_1'}(A,B) \cup \ \cup\{\overline{\mathrm{Path}_1}(C,D) \mid (A,B)\ I\ (C,D)\}$$

with the notations: $\overline{\mathrm{Path}_1}(A,B) = \mathrm{Path}_1(A,B) \cup \{A\}$

and $\overline{\mathrm{Path}_1'}(A,B) = \mathrm{Path}_1'(A,B) \cup \{A\}$.

This theorem is straightforward from above Lemmas and formula(2.6).

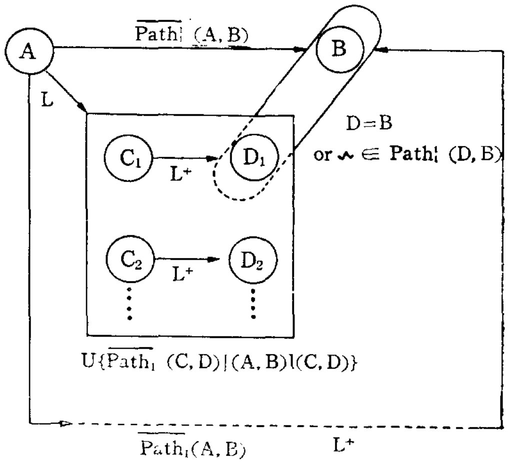
The following figure essentially captures this fact.

Now, we introduce an useful theorem in computing Path from [4].

[**Theorem 4.2**] Let R be a relation on a set X. Let F be a set-valued function such that for all $x\in X$,

$$F(X) = F'(x) \cup \ \cup\{F(y) \mid xRy\}$$

where $F'(x)$ is given for all $x\in X$. Let $G=(X,R)$ be the digraph induced by R, that is, G has vertex set X and (x,y) is an edge iff



xRy. Then the algorithm Digraph correctly computes F in linear order of G. [4]

```
Algorithm Digraph:
input $R$, a relation on $X$, and $F'$, a function
        form $X$ to sets.
output $F$, a function from $X$ to sets such
        that $F\ x$ satisfies(4.1).
let $S$ be an intially empty stack of elements
        of $X$
let $N$ be an array of zeros indexed by elem-
        ents of $X$
for $x\in X$ such that $N\ x=p$ do call Traverse
        $x$ od
where recursive Traverse $x=$
call Push $x$ on $S$
con $d$: Depth of $S$
assign $Nx\leftarrow d$ ;$Fx\leftarrow F'x$
for $y\in X$ such that $xRy$
do if $Ny=0$ then call Traverse $y$ fi
assign $Nx\leftarrow \mathrm{Min}(Nx,\ Ny)$; $Fx\leftarrow Fx\cup Fy$
od
if        $Nx=d$
then repeat assign $N(\text{Top of } S)\leftarrow$Infinity ;
        $F(\text{Top of } S)\leftarrow Fx$
        until (Top of $S)=x$
fi
end Traverse
```

end Digraph

In virtue of [Theorem 4.1] and [Theorem 4.2], we can compute $\overline{\mathrm{Path}_1}$ in a effcient way.

Now, we can get the final result by the following formula.

$\mathrm{Path}_1(A, B) = \overline{\mathrm{Path}_1}(A, B) - \{A \mid \text{not } (A\ \eta^+\ B)\}$.

Ⅴ. Conclusion

Several characteristics of Path are studied, and formalisms for computing Path are developed.

In virture of these formalisms, the meaning of Path is clarified and an efficient computation method is derived. This method uses Digraph algorithm associated with relation *"I"* which nicely captures inclusions in Path sets. The application of this method to LALR(1) parser generating system can reduce the computation time for LALR(1) lookahead sets.

References

1. Park, Joseph C.H., Choe, K.M., and Chang, C.H. "A New Analysis of LALR Formalisms", ACM Transactions on Programming Languages and Systems, Vol. 7, No. 1, Jan 1985.
2. Aho, A.V., and Ullman, J.D., "The Theory of Parsing, Translation, and Compiling" Vol. 1 : Parsing, Prentice-Hall, Inc. 1972.
3. Aho, A.V., and Ullman, J.D., "The Theory of Parsing Translation, and Compiling" Vol. 2 : Compiling, Prentice-Hall, Inc. 1972.
4. DeRemer, Frank, and Pennello, Theomas "Efficient Computation of LALR(1) Look-Ahead Sets", ACM Transactions on Programming Languages and Systems, Vol. 4, No. 4, October 1982.