2011 International Conference on Computer Science and Information Technology (ICCSIT 2011)
IPCSIT vol. 51 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V51.93

A New Computing Method of First and Follow Sets

Jing Zhang[a,*]

[a]School of Computer Science & Software Engineering,Tianjin Polytechnic University,No 399, Binshui West street, Xiqing District,Tianjin 300387, P.R China

Abstract. First and Follow functions associated with a grammar G is important when conducting LL and LR (SLR, LR (1), LALR) parser, because the setting up of parsing table is aided by them. For the larger scale grammar, computing First and Follow sets by hand according to the definition brings not only huge intolerable workload but also mistakes, even if adopting automated tools it will cost more time. In order to solve this problem, based on the parallel algorithm already existing, a new method of computing First and Follow sets was brought forward in this paper. This method can not only improve computing efficiency but also avoid errors, and it has both the theoretical and realistic significance for parallel compiler processing.

Keywords: First and Follow Sets, parallel processing, effeicency, avoid errors

1. Introduction

There are three general types of parsers for grammars. Universal parsing methods such as the Coche-Younger-Kasami algorithm and Earley's algorithm can parse any grammar. These methods, however, are too inefficient to use in production compilers. The methods commonly used in compilers are classified as being either top-down or bottom-up. As indicated by their names, top-down parsers build parse trees from the top (root) to the bottom (leaves), while bottom-up parsers start from the leaves and work up to the root. The most efficient top-down and bottom-up methods work only on subclasses of grammars, but several of these subclasses, such as the LL and LR grammars are expressive enough to describe most syntactic constructs in programming language, parsers implemented by hand often work with LL grammars, parsers for the larger class of LR grammars are usually constructed by automated tools[1].

Both LL and LR (SLR, LR (1), LALR) parser is aided by two functions, they are First and Follow. If α is any string of grammar symbols, let First(α) be the set of terminals that begin the strings derived from α, if α*=>ε, then ε is also in First(α). Define Follow(A), for non-terminal A, to be the set of terminals a that can appear immediately to the right of A in some sentential form, that is, the set of terminals a such that there exists a derivation of the form S*=>αAaβ for some α and β, note that there may, at some time during the derivation, have been symbols between A and a, but if so, they derived ε and disappeared. This paper is mainly about how to compute First and Follow sets based on the algorithm of Yuqiang Sun [2].

2. Computing First and Follow

To compute First(X) for all grammar symbols, apply the Following rules until no more terminals or ε can be added to any First set [3].

1. If X is terminal, then First(X) is X.
2. If X→ ε is a production, then add ε to First(X).
3. If X is non-terminal and $X \rightarrow Y_1 Y_2 \cdots Y_K$ is a production, then place a in First(X) if for some i, a is in First (Y_i), and ε is in all of First (Y_1)…First (Y_{i-1}); that is, $Y_1 \cdots Y_{i-1}$*=>ε. If ε is in First (Y_j) for all j=1, 2, ···, k, then add ε to First(X)[4].

* Corresponding author.
E-mail address: China_ximeng@sohu.com.

Now define First (α) for any string $\alpha=X_1X_2\ldots X_n$ as follows. First(α) contains First(X_1)-{ε}.For each i=2,…,n, if First(X_k) contains ε for all k=1,…,i-1, then First(α) contains First(X_i)-{ε}.Finally, if for all i=1,…,n, First(X_i) contains ε, then First (α) contains ε.

Given a non-terminal A, the set Follow (A), consisting of terminals, and possibly $, is defined as Follows.

1. If A is the start symbol, then $ is in Follow (A).

2. If there is a production B→ αAγ, then First (γ)-{ε} is in Follow (A).

3. If there is a production B→ αAγ such that ε is in First (γ), then Follow (A) contains Follow (B).

For example, in the grammar E→ E+T | T; T→ T*F | F; F→ (E) | i , we can get:

First (E) = { (, i }

First (T) = { (, i }
First (F) = { (, i }
Follow (E) = { +,), $ }
Follow (T) = { +, *,), $ }
Follow (F) = { +, *,), $ }

The computing method of First and Follow sets can adopt the algorithm of sets and matrix, in order to improve the efficiency of computing for larger scale grammar, we can conduct parallel processing.

3. Parallel processing

Parallel processing is the simultaneous processing of the same task on two or more microprocessors in order to obtain faster results: 1) To be run using multiple CPUs; 2) A problem is broken into discrete parts that can be solved concurrently 3) Each part is further broken down to a series of instructions, Instructions from each part execute simultaneously on different CPUs [5].

Before using parallel processing, parallelization processing was needed for the program, that is to say, it should allot each part of work to different processor. The processors can deal with different aspects of the same program at the same time.

Parallel computing have many advantages, such as: 1) Save time and/or money: In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings. 2) Solve larger problems: Many problems are so large and/or complex that it is impractical or impossible to solve them on a single computer, especially given limited computer memory.3) Provide concurrency: A single compute resource can only do one thing at a time. Multiple computing resources can do many things simultaneously.

For parallel processing's advantages, Yuqiang Sun has provided a parallel computing method of First and Follow sets [2]. However, this method can only compute First sets accurately, when it comes to Follow sets, it can only compute accurately for partial grammars. That is, it is easy to omit something and make errors when it is used to compute Follow sets for some grammar. For example, in the grammar E→ E+T | T; T→ T*F | F; F→ (E) | i, if we use his algorithm, we can get:

First (E) = { (, i }
First (T) = { (, i }
First (F) = { (, i }
Follow (E) = { +,), $ }
Follow (T) = { *, $ }
Follow (F) = Φ

From the result, we can see that it can only compute First sets and Follow (E) correctly, but when it comes to Follow (T) and Follow (F), it can not give the correct answer.

Based on his theory, the author give a complementary to his algorithm in this paper, it perfects his algorithm. The main theories and algorithms are provided in the following two parts.

4. Theoretical foundation

In a given grammar G, S is the starting symbol, *m* is the number of elements in set V_N and *n* is the number of elements in set V_T, that is, m=|V_N|, n=|V_T|. Alphabet set V= $V_N \cup V_T$, $V_N \cap V_T = \Phi$. Let $S_k \in V_N$, and S_i , $S_j \in V$, here k=1,2,···,m; i, j=1,2,···, m + n[2].

Definition 1 If there is a production $S_k \rightarrow S_i \cdots$, it is called: S_i is the start of S_k, and it is written as S_k Start S_i; If there is a production $S_k \rightarrow \cdots S_i\ S_j \cdots$, it is called: S_j is the right of S_i, and it is written as S_i Right S_j.

Definition 2 If there is a derivation S_k*=> $S_i \cdots$, it is written as S_k First S_i; If there is a derivation S*=>··· S_i $S_j \cdots$ it is written as S_i Follow S_j.

Specifications：

(1) If $S_k \rightarrow \varepsilon$, then S_k First ε

(2) S is the starting symbol, then S Follow $.

The Following theorems can be derived from Definition 1 and Definition 2.

Theorem 1: Let $S_i \in V_N$, $S_j \in V_T$ (i=1, 2, ··· ,m; j=1,2,···, n)

(1) If S_i Start S_j, then S_i First S_j

(2) If S_i Right S_j, then S_i Follow S_j

Theorem 2: Let S_i, $S_k \in V_N$, $S_j \in V_T$ (k, i, =1, 2,···,m ; j=1,2,···, n)

(1) If S_k Start S_i, S_i Start S_j , then S_k First S_j

(2) If S_k Right S_i, S_i Start S_j, then S_k Follow S_j

Theorem 3**:** Let S_k, $S_i \in V_N$, $S_j \in V_T$ (k, i,=1,2,···, m; j=1,2,···, n)

(1) In the same production, if there are relations of S_k Start S_i and S_i Right S_j, and in the grammar G, S_i First ε , then S_k First S_j

(2) In the same production, if there are relations of S_k Right S_i, S_i Right S_j, and in the grammar G, S_i First ε , then S_k Follow S_j

According to the definitions of First and Follow sets:

For A$\in V_N$: First (A) = {a | A*=>a···, a $\in V_T$}

Follow (A) = {a | S*=> ···Aa···, a $\in V_T$}

According to the given theorems, we can get some conclusions:

(1)The case of A First a, that is, a ∈First (A), consists of some possible cases as Follows:

(1.1) A Start a

(1.2) A Start S_1, S_1 Start S_2, ···, S_i Start a (i =1, 2, ···, m)

(1.3) In the same production, if there are relations of A Start S_i and S_i Right a, and in the grammar G, S_i First ε.

(2)The case of A Follow a, that is, a ∈Follow (A), consists of some possible cases as Follows:

(2.1) A Right a

(2.2) A Right S_1, S_1 Start S_2, ···, S_i Start a (i =1,2,···,m)

(2.3) In the same production, if there are relations of A Right S_i and S_i Right a, and in the grammar G, S_i First ε.

5. The parallel computing method of First and Follow sets

Let M_S, M_R, M_f, M_F be the relation matrix of Start, Right, First and Follow respectively. From all the productions in grammar G, we can get relation matrix Ms and M_R. The elements in relation matrix Ms are defined as Follows:

a_{ij} = 1, if S_i Start S_j, else a_{ij}= 0

The elements in relation matrix M_R are defined as Follows:

b_{ij} =1, if S_i Right S_j, else b_{ij}=0

Here i=1, 2, ···, m; j=1,2, ···, m + n

Algorithm 1: The parallel algorithm for computing relation matrix M_f [2]

(1) Construct relation matrix M_S from all the productions in grammar G;

(2) If ε is the right side of a production in grammar G, according to theorem 3(1), the relation matrix M_S should be revised, or else, go to (3);

(3) Each row of relation matrix M_S is assigned to a processor, the number of processors is m, and row i is assigned to P_i (i=1, 2, ···, m).

(4) i = m

(4.1) for all a_{ik}（$1 \le k \le i$）in P_i, par-do:

(4.1.1) if a_{ik} =1, then the elements in the processor P_k are transferred to processor P_i, and logical additions are performed with the original elements in P_i on the principle of left-justify, and the results are stored in P_i , replacing the originals in P_i , and marking a_{ik} as "used"

(4.1.2) take the element a_{ik} in P_i continually, until every element in P_i with the value of 1 has been marked;

(4.2) i=i-1;

(4.3) if i≥1 then go to (4.1) else go to (5);

(5) End.

Finally, the elements from m+1 to m + n in P_i of the finished matrix show the relation matrix M_f, and from M_f we can get the elements of First sets.

Algorithm 2: The parallel algorithm for computing relation matrix M_F

(1)Construct relation matrix M_R from all the productions in grammar G;

(2) If ε is in the right side of a production in grammar G, according to theorem 3(2), relation matrix M_R should be revised, else go to (3);

(3) Each row of relation matrix M_R is assigned to a processor, the number of processors is m, row i is assigned to Q_i (i=1, 2, ···, m). All non-terminal can be named respectively as S_i (i=1, 2, ···, m).

(4) i =m

(4.1) for each b_{ik}（$1 \le k \le i$）in Q_i, par-do:

(4.1.1) if b_{ik} =1, First, the elements expressing the relation of S_i First ε are removed from the processor P_k, then the other elements are transferred to Q_i , and logical additions are performed with the original elements in Q_i on the principle of left-justify, and the results are stored in the processor Q_i, replacing the originals in Q_i , and marking b_{ik} as "used";

(4.1.2) continue to take the element b_{ik} in Q_i , until every element in Q_i with the value of 1 has been marked;

(4.2) i=i-1;

(4.3) if i≥1 then go to (4.1) else go to (5);

(5) For i= 1 to m, par-do:

If there is a production $S_i \rightarrow S_j$ or $S_i \rightarrow \cdots S_j$ or $S_i \rightarrow S_j\beta$ (S_i, $S_j \in V_N$) and $\beta^* \Rightarrow \varepsilon$ then the elements of Q_i are transferred to Q_j, and logical additions are performed with the original elements in Q_j on the principle of left-justify, and the results are stored in the processor Q_j, replacing the originals in Q_j.

(6) End.

Finally, the elements from m+1 to m + n in Q_i of the finished matrix show the relation matrix M_F, from which we can get the elements in Follow sets of non-terminals S_i.

6. Application

6.1. Considering a non-operator grammar

Given a grammar G_1:

E → E+T | T
T → T*F | F
F → (E) | i

There are three non-terminals in this grammar, so the elements in relation matrix M_S of E, T and F are stored in three processors P_1, P_2 and P_3 respectively, it is shown in fig. 1(a). According to algorithm 1(4), we can get the result of M_S after parallel processing, then we can get relation matrix M_f from it. Fig. 1(b) and fig. 1(c) demonstrate it.

		E	T	F	+	*	(	)	i
P_1	E	1	1	0	0	0	0	0	0
P_2	T	0	1	1	0	0	0	0	0
P_3	F	0	0	0	0	0	1	0	1

(a) Relation matrix M_S

		E	T	F	+	*	(	)	i
P_1	E	1	1	1	0	0	1	0	1
P_2	T	0	1	1	0	0	1	0	1
P_3	F	0	0	0	0	0	1	0	1

(b) Parallel process of M_S

		+	*	(	)	i
P_1	E	0	0	1	0	1
P_2	T	0	0	1	0	1
P_3	F	0	0	1	0	1

(c) Relation matrix M_f

Fig. 1. Process for computing relation matrix M_f

From each row of M_f, we can get:

First (E) = {(, i}
First (T) = {(, i}
First (F) = {(, i}

The elements in relation matrix M_R of non-terminals E, T and F are stored in three processors Q_1, Q_2 and Q_3 respectively (shown in Fig. 2(a)). According to algorithm 2(4), there is no change, then, according to algorithm 2(5), we can get relation matrix M_R after parallel processing. Correspondingly, we can get relation matrix M_F from it. Fig. 2(b) and fig.2(c) can demonstrate the process.

		E	T	F	+	*	(	)	i	$
Q_1	E	0	0	0	1	0	0	1	0	1
Q_2	T	0	0	0	0	1	0	0	0	0
Q_3	F	0	0	0	0	0	0	0	0	0

(a) Relation matrix M_R

		E	T	F	+	*	(	)	i	$
Q_1	E	0	0	0	1	0	0	1	0	1
Q_2	T	0	0	0	1	1	0	1	0	1
Q_3	F	0	0	0	1	1	0	1	0	1

(b) Parallel process of M_R

		+	*	(	)	i	$
Q_1	E	1	0	0	1	0	1
Q_2	T	1	1	0	1	0	1
Q_3	F	1	1	0	1	0	1

(c) Relation matrix M_F

Fig. 2. Process for computing relation matrix M_F

From each row of M_F, we can get

Follow (E) = {+,), $}
Follow (T) = {+, *,), $}
Follow (F) = {+, *,), $}

6.2. Considering a grammar with ε in productions

Given a grammar G_2:

S →AB | DC
A →aA | ε
B →bBc | ε
C →cC | ε
D →aDb | ε

There are five non-terminals in this grammar, so the elements in relation matrix M_S of S, A, B, C and D are stored in five processors P_1, P_2, P_3, P_4 and P_5 respectively. It is shown in fig. 3(a). We can revise relation matrix M_S, and according to algorithm 1(4), we can get the result of M_S after parallel processing, then we can get relation matrix M_f from it. Figs. 3(b) through (d) show the process.

		S	A	B	C	D	a	b	c	ε
P_1	S	0	1	0	0	1	0	0	0	0
P_2	A	0	0	0	0	0	1	0	0	1
P_3	B	0	0	0	0	0	0	1	0	1
P_4	C	0	0	0	0	0	0	0	1	1
P_5	D	0	0	0	0	0	1	0	0	1

(a) Relation matrix M_S

		S	A	B	C	D	a	b	c	ε
P_1	S	0	1	1	1	1	0	0	0	0
P_2	A	0	0	0	0	0	1	0	0	1
P_3	B	0	0	0	0	0	0	1	0	1
P_4	C	0	0	0	0	0	0	0	1	1
P_5	D	0	0	0	0	0	1	0	0	1

(b) Revised M_S

		S	A	B	C	D	a	b	c	ε
P_1	S	0	1	1	1	1	1	1	1	1
P_2	A	0	0	0	0	0	1	0	0	1
P_3	B	0	0	0	0	0	0	1	0	1
P_4	C	0	0	0	0	0	0	0	1	1
P_5	D	0	0	0	0	0	1	0	0	1

(c) Parallel process of M_S

		a	b	c	ε
P_1	S	1	1	1	1
P_2	A	1	0	0	1
P_3	B	0	1	0	1
P_4	C	0	0	1	1
P_5	D	1	0	0	1

(d) Relation matrix M_f

Fig. 3. Process for computing relation matrix M_f

From each row, we can get:

First (S) = {a, b, c, ε}
First (A) = {a, ε}
First (B) = {b, ε}
First (C) = {c, ε}
First (D) = {a, ε}

The elements in relation matrix M_R of non-terminals S, A, B, C and D are stored in five processors Q_1, Q_2, Q_3, Q_4 and Q_5 respectively. After revising, the relation matrix M_R is shown in fig. 4(a). According to algorithm 2(4), we can get the relation matrix M_R after parallel processing. Correspondingly, we can get relation matrix M_F from it. Fig. 4(b) and fig. 4(c) can demonstrate the process. From each row, we can get:

Follow (S) = {$}
Follow (A) = {b, $}
Follow (B) ={c, $}
Follow (C) = {$}
Follow (D) = {b, c, $}

		S	A	B	C	D	a	b	c	$
Q_1	S	0	0	0	0	0	0	0	0	1
Q_2	A	0	0	1	0	0	0	0	0	0
Q_3	B	0	0	0	0	0	0	0	1	0
Q_4	C	0	0	0	0	0	0	0	0	0
Q_5	D	0	0	0	1	0	0	1	0	0

(a) Revised M_R

		S	A	B	C	D	a	b	c	$
Q_1	S	0	0	0	0	0	0	0	0	1
Q_2	A	0	0	1	0	0	0	1	0	1
Q_3	B	0	0	0	0	0	0	0	1	1
Q_4	C	0	0	0	0	0	0	0	0	1
Q_5	D	0	0	0	1	0	0	1	1	1

(b) Parallel process of M_R

		a	b	c	$
Q_1	S	0	0	0	1
Q_2	A	0	1	0	1
Q_3	B	0	0	1	1
Q_4	C	0	0	0	1
Q_5	D	0	1	1	1

(c) Relation matrix M_F

Fig. 4. Process for computing relation matrix M_F

7. Conclusions and discussions

Compared with the original algorithm according to the definition, this algorithm has both the same function and the same result, but it has a faster speed, which is greatly attributed to the advantages of parallel processing. However, it is not obvious because there are only three or five productions in the two examples, if there are tens or thousands of productions, it will show its advantages.

In addition, compared with the algorithm put forward by Yuqiang Sun, this one is a complement for it. There is only a little difference in the algorithm, which is presented in Algorithm 2, it is about the parallel algorithm for computing relation matrix M_F. The author just adds a processing step (step 5), and it is what really makes the difference. It makes the algorithm perfect, it can compute accurately and suit for more grammars, such as the two examples prevented above.

The research of parallel syntax analysis has an important signification for the development of computer discipline and the application of parallel computer and processor. There are still a lot we can research deeply.

8. References

[1] Alfred V.Aho, Monica S.Lam, Ravi Sethi, Jeffrey D.Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*, New Jersey: Addison Wesley, 2007.

[2] Yuqiang Sun. Design of Parallel Algorithm for FIRST and FOLLOW Sets. Computer Engineering,2004(11):pp.71-73.

[3] Kenneth C. Louden , *Compiler Construction Principles and Practice*, New Jersey :Addison Wesley, 2007.

[4] Arturo Trujillo, Computing First and Follow functions for feature-theoretic grammars, The 15th International Conference on Computational Linguistics,1994:pp.875-880.

[5] *Blaise Barney*. Introduction to Parallel Computing. https://computing.llnl.gov/tutorials/parallel_comp/