Journal of Applied Mathematics and Computational Mechanics 2014, 13(3), 67-74

AN ANALYSIS OF THE CONCURRENT CALCULATION OF THE FIRST SETS

Piotr Jeruszka

Institute of Computer and Information Science, Czestochowa University of Technology
Częstochowa, Poland
piotr.jeruszka@icis.pcz.pl

Abstract. This paper is focused on the process of computing First Sets. The First Sets are used to build structures which control a syntax analyser (also known as parser). Three methods of creating First Sets were compared in terms of execution time. The first method is known sequential algorithm and the author's own methods are concurrent computing sets for each non-terminal symbol (called the CEN method) and concurrent computing sets for each production (called the CEP method). These methods have been tested on personal computer. Three programming languages (including the C language) were used in the research. The results and the analysis of calculations allow the author to hypothesise that the problem of computing First Sets is hard to concurrence.

Keywords: *First Sets, concurrent computing, construction of languages, parsing*

Introduction

The engineering simulations are made with technical software. Such software allows the scientists to describe the research using the programming language adapted to the specification of the technical issue. Packages with their own programming languages are, for instance, MatLab and Octave.

The syntax analyzer (parser) handles the analysis of the written program's syntax. Parser performs on the basis of the context-free grammar (CFG) [1]. There are tools which generate the syntax analyzers on the basis of the grammar. Such analysers check the syntax of the input data only.

This paper is focused on the process of calculation of the First Sets. Such calculation is required to generate the parser. The exemplary grammars, C language and the author's own simple language and unique dummy language (with several thousand productions and only one terminal symbol) were used in this research.

Computed First Sets can be used to construct a parsing table [2]. The table controls parsing process of the source code. However, the parsing is a different issue than creating the parser so the syntax analyse process is not considered in this paper.

Zhang in [3] presented the First Sets computing parallel method. The parallel method concentrates into data and the processors synchronization. The already presented concurrent methods are not based on hardware opportunities but they are based only on software management of data.

1. Problem of Computing First Sets

A context-free grammar [4] written with a tuple of four elements is the basis of the parser:

$$G = (\Sigma, N, P, S)$$

where Σ is a set of terminal symbols, N - a set of non-terminals, S - highlighted starting terminal symbol. P is a set of production:

$$A \rightarrow \alpha$$

where A is exactly one non-terminal symbol and α is a set of symbols. If set of α is empty, then the production (empty production) is written as:

$$A \rightarrow \varepsilon$$

The symbols used in the grammar are represented in the structure of the programming language. Terminal symbols can be compared to the individual language elements such as keywords, literal numbers or assignment operators. Non-terminal symbols represent more advanced syntactic structures such as loops, conditional statements or definitions of the data types.

Context-free grammar generates a context-free language. If the source code is considered as a set of terminal symbols then the parser is able to determine whether the code should be the language of the grammar.

Derivation is a sequence of grammar transformation that computes the start symbol (of CFG) to the sequence of terminal symbols which is equal to the source code. The term *leftmost* means that the leftmost non-terminal is always converted to sequence of symbols. The leftmost derivation is shown in the example below:

$$S \rightarrow \alpha AB\beta \rightarrow \alpha\gamma B\beta$$

The parser and the parser generator used in research are able to make only leftmost derivations which means the parser has to know all possible configurations of symbols in derived sequence. That information is collected on two kinds of sets: First and Follow. The set of terminals that are in the first position in all possible sequences derived from X is the First Set for non-terminal X. The set of terminals that are immediately on the right of X in any possible sequences derived from CFG is known as the Follow Set for non-terminal X.

The sequential algorithm which was presented in [2] is a recursive one.
For each non-terminal symbol X in grammar G the following steps are made:
(1) Create blank set: FIRST(X).
(2) For each production $X \rightarrow \alpha$ of non-terminal:

(2.1) If α is a single terminal symbol or ε then add α to FIRST(X). Go to the next production of X non-terminal (back to step 2).
(2.2) If α is a sequence of symbols like $Y_1 Y_2 \dots Y_n$ then for i = 1..n–1 do the following steps:
(2.2.1) If $\varepsilon \in$ FIRST(Y_i) then add FIRST(Y_i) / ε to FIRST(X).
(2.2.2) If not $\varepsilon \in$ FIRST(Y_i) then add FIRST(Y_i) to FIRST(X) and go to the next production of X non-terminal.
(2.3) Add FIRST(Y_n) to FIRST(X).

That sequence of steps repeats until no more symbols can be added to any First Set.

2. Concurrent solutions

The context-free grammar may contain some terminal symbols (X_i) which cost calculations of First Set (X_i) is t_i. This paper assumes that the only factor considered is the cost of computation time.

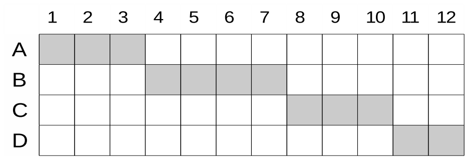

Fig. 1. Gantt diagram for sequence algorithm of finding First Set for 4 examples of non-terminals with following computing time: A - 3, B - 4, C - 3, D - 2

Figure 1 shows the Gantt chart representing the execution of a single iteration of all non-terminal symbols of the grammar G. The grammar has four non-terminal symbols (N= {A, B, C, D}) with the computation time of 3, 4, 3 and 2. The total time required for a single iteration of the non-terminal is at least 12. As a result, the hypothesis of minimal time needed to calculate the First Sets for grammar having n non-terminal symbols can be concluded.

$$T_{seq} = \sum_{i=1}^{n} t_i, \ n = |N| \tag{1}$$

Concurrent calculations occur when two or more processes of the task run on a single physical processor. For this work an assignment of tasks to the processors is determined by the Java Virtual Machine.

The usage of concurrency, however, requires solving programming and hardware problems such as:

a) mutual exclusion (also known as *dead lock*),
b) synchronization,
c) loss of performance due to the synchronization.

Issues of mutual exclusion and synchronization were solved with the software. The problem of performance degradation was not dealt with. It was considered that the Java Virtual Machine is properly optimized in threads managing.

Assuming that an even distribution of tasks, launching the calculations at the same time and the lack of relationship between the terminal symbols occurred, it could be acknowledged that the concurrent computing of the First Set will take:

$$T_{synch} = \max(t_1, t_2, \dots, t_n), \quad n = |N| \tag{2}$$

according to the Gantt chart shown in Figure 2.

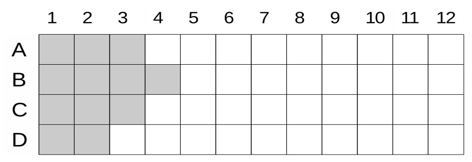

Fig. 2. Gantt diagram for concurrent algorithm of finding First Set for 4 examples of non-terminals with following computing time: A - 3, B - 4, C - 3, D - 2

The above equations will not be compared with results (there are no finding time of every calculation of the First Set), but total time of computing may be presented by the following model:

$$T_{synch} \leq T_{seq} \tag{3}$$

2.1. Finding the First Set for each non-terminal symbol

The construction of a computer program to calculate the First Sets with the author's own concurrent method of calculation for each non-terminal symbol (shortly called CEN) begins with the designation of the list of steps and the required data types. The basic data type is FirstThread that represents the thread (single task) computing the First Set for the non-terminal symbol.

Each FirstThread object has bit field named *computed*. This field (initially set on *False*) is useful in thread synchronization process.

The following list of steps present the CEN method:

For each non-terminal symbol X in grammar G do the following steps:

(1) Create and start new FirstThread(X) which will compute First Set for non-terminal X.

(2) For each production $X \rightarrow \alpha$ of non-terminal:

(2.1) If α is a single terminal symbol or ε then add α to FIRST(X).

(2.2) If α is a sequence of symbols like $Y_1 Y_2 \dots Y_n$ then check the type for each symbol on right side production.

(2.2.1) If $Y \in \Sigma$ then add Y to FIRST(X).

(2.2.2) If $Y \in N$ then check if FirstThread(Y) is marked as $True$. If not then wait for computation until the type changes. After that:

(2.2.2a) If $\varepsilon \in$ FIRST(Y) then add FIRST(Y)/ε to FIRST(X).

(2.2.2b) If not $\varepsilon \in$ FIRST(Y) then add FIRST(Y) to FIRST(X) and go back to step 2 (take next production of non-terminal X).

2.2. Finding the First Set for each type of production

This method (computing First Set for each production, shortly CEP) is an extended version of CEN method. Each thread dealing with the calculation of the First Set for the terminal creates new threads. These new threads (type Production-Thread objects) calculate the First Set on the basis of the only one production.

The following list of steps present the CEP method:
For each non-terminal symbol X in grammar G do the following steps:

(1) Create and start new FirstThread(X) which will compute the First Set for non-terminal X.

(2) For each production $X \rightarrow \alpha$ of non-terminal:

(2.1) Create and start new ProductionThread($X \rightarrow \alpha$) which will compute the First Set from production $X \rightarrow \alpha$.

(2.2) In each ProductionThread calculating First for α is respectively as in way for each non-terminal (from step 2.1 in concurrent computing First Set for each non-terminal method).

3. Results

The calculations were made with the *LLVisualizer* tool which makes it possible to analyze the LL(1) [5] class of grammar. The threads managing module could be transferred to Java Virtual Machine as a result of this tool being written in Java. *LLVisualizer* was running on 4 threads platform (2 physics cores with 2 thread execution in the same time). Three programming languages were investigated.
The first, the author's own language, called *Simple programming language*, is described by following CFG with sets:

$$\Sigma = \{\, sem, label, asign, to, ident, int, +, -, do, goto\}$$
$$N = \{\, E, ASIGN\_E, ARI\_E, R\_ARI\_E, IF\_E, R\_IF\_E, RR\_IF\_E\}$$
$$P = \{E \rightarrow ASIGNE\ sem\ E; E \rightarrow IFEXPR\ sem\ EXPR; E \rightarrow label\ sem\ EXPR \ldots\}$$

This construction allows the parser to check syntax with a kind of expression used in simulations such as: assign expression, arithmetic expression, conditional statement and loop statement.

The second examined language is known the grammar of C programming language [6]. This grammar describes syntax for C and it has 63 non-terminal symbols and 198 productions. This language had to be transformed due to the fact that the parser needs to have a grammar without recursion.

The third programming language used in the research is called *Dummy programming language*. Its construction has the following features: many of non-terminals with only one production in each and there is only one terminal. Both the non-terminal set and the production set have 5001 elements:

$$\Sigma = \{ int\}$$
$$N = \{ X_1, X_2, X_3, \dots, X_{5001}\}$$
$$P = \{ X_1 \rightarrow X_2, X_2 \rightarrow X_3, \dots, X_{5000} \rightarrow X_{5001}, X_{5001} \rightarrow int\}$$

All of the above methods compute the valid First Sets. The results were checked with ANTLR - syntax analyser generator [7]. Only the time of calculating will be compared in the section below. There were 100 completed calculatings (computing all First Sets for each non-terminal symbol on CFG) for each method. An average calculating time is presented in Figures 3 and 4. 50 calculations were completed in dummy programming language (results are presented in Figure 5).

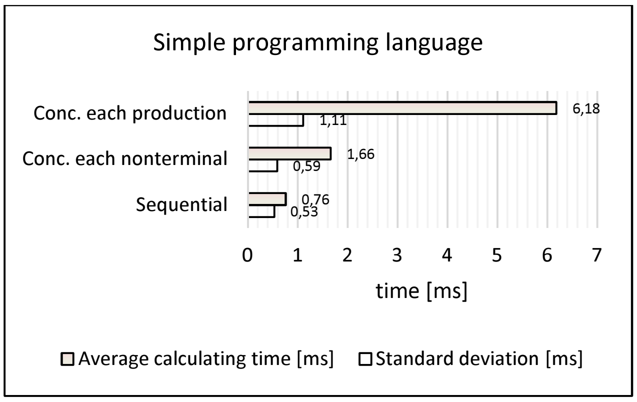

Fig. 3. Chart of average calculating time and standard deviation of First Sets computing in Simple programming language

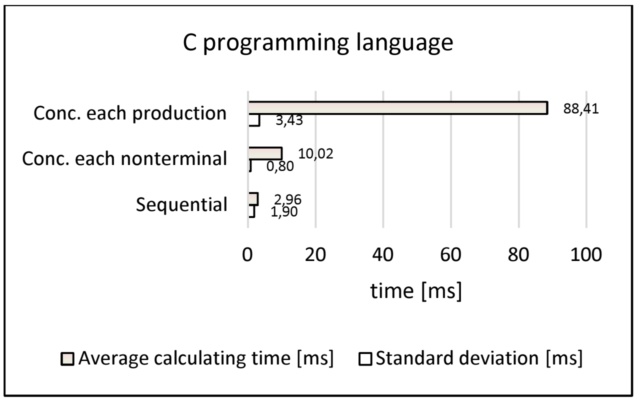

Fig. 4. Chart of average calculating time and standard deviation of First Sets computing in C programming language

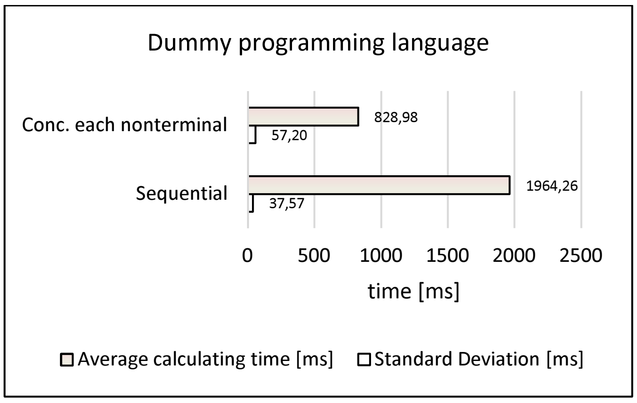

Fig. 5. Chart of average calculating time and standard deviation of First Sets computing in dummy programming language

The third method of concurrent calculating the First Sets (for each production) was being computed for over one hour (4295 seconds) and is not presented on the chart.

Conclusions and future work

Except for the dummy language, the research results do not achieve inequality (3) thus one of two options might be considered:

(A) the search for the First Sets is a difficult process to implement concurrent version;

(B) dependence (1) and/or (2) may be unfulfilled. It is necessary to focus on the accurate calculation of the values of the equations (1) and (2) in further studies.

Examination of the equations listed in paragraph (B) will allow the author to check the efficiency of the CEP method.

The CEN method proved to be faster in the examination of the third presented grammar. The classic algorithm ends its operation when no new element to any First Sets is added throughout all the productions. During the first iteration, 5001 calculations for each production are made but only the symbol int will be added to the First(X_{5001}). The same happens with the following iteration (5000, 4999 and so on). The CEN method does not perform these operations since each of the production is tested only once during the whole course.

The author will focus on investigating the presented methods of finding the First Sets in further research. Acceleration of development of the First Sets may increase the generation of the programming language compilers.

References

[1] Chomsky N., Three Models for the Description of Language, Massachusetts Institute of Technology, 1956.

[2] Aho A.V., Sethi R., Ullman J.D., Compilers: Principles, Techniques and Tools, Pearson Education, 1986.

[3] Zhang J., A New Computing Method of First and Follow Sets, ICCSIT 2011, 555-561.

[4] Huzar Z., Elementy logiki i teorii mnogości dla informatyków, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2007 (in Polish).

[5] Jeruszka P., Obiektowa implementacja parsera klasy LL(1), XV Studencka Konferencja Informatyczna WIMiI PCz, Częstochowa 2013 (in Polish).

[6] Lee J., ANSI C Yacc grammar, 1985 (at: http://www.lysator.liu.se/c/ANSI-C-grammar-y.html).

[7] Parr T., The Definitive ANTLR Reference: Building Domain-Specific Languages.