Simulation and Visualization of Parsing

Petr Šaloun

Department of Computer Science, FEI VŠB–Technical University, 17. listopadu 15,
708 33 Ostrava, Czech Republic, *email:* `Petr.Saloun@vsb.cz`

Abstract

Parsing of context–free languages is based on the push–down automaton. Parsing is directed by the parsing table. The set of developed classes is applicable for both kinds of the context–free parsing, for the bottom–up parsing and the top–down parsing. The article gives an overview of the hierarchy of classes. The simulator allows to visualize the process of parsing step–by–step. The simulator is used for science as well as for education. The simulator is implemented in object oriented programming system Smalltalk. The Model–View–Controller paradigm allows to use the parser alone.

1 Introduction and Motivation

Topic of my PhD. thesis is *Parallel Parsing*. The side effect of my theoretical considerations is effort to simulate them for better understanding of their mechanisms. It seems be useful for me as well as for MSc. students of mine.

Article [Sal96] describes complex relations of parsing in simulated parallel environment. This simulation gives both results – *partial* results, and *final* results. The results are presented at the end of the simulation.

This article gives much less sophisticated result. The step–by–step *simulation* and *visualization* of the sequential parsing.

2 Object Model of CFG

The object model of the context free grammar (*CFG*) is based on the properties of CFG given by *Noam Chomsky*. The standard book on the theory and the practice of compiling is [ASU87].

The model is described in [Sal94] and implemented in Smalltalk – *E*nvironmentally based *P*ure *O*bject oriented *L*anguage (*EPOL*), see [Dig92] for more details.

The `GrammarBase class` contains a quadruple of items as well, as the Chomsky's paradigm. They are sets of `Terminals` and `Nonterminals`, dictionary of `Rules` and the `Start` symbol. One rule consist of a (nonterminal) symbol on the left side, and `GRSide` — an array of symbols (terminals, nonterminals, the empty string) on the right side of the rule.

The `GrammIn class` creates the internal form of a grammar from the textual form on the input no matter on the class of the grammar. The GramIn's lifetime is very short.

The `GrammarBase class` is a root of the class hierarchy of inheritance. The classes `GrammarFF` and `GrammarLx` extend the properties of GrammarBase by functions such as `first` and `follow`.

3 Parser and it's Extensions

The early approach of mine covers only the *top–down* parsing, see [Sal94]. The object model of CFG allows to expand the early approach by the *bottom–up* parsing. The expansion is transparent from the messages point of view. The meaning of messages such as `makeReady`, `doParseTable`, and `parseIt`, depends on the class of the receiver.

The `GrammarLx class` is the root node of two subtrees of inheritance — `GrammarLL` and `GrammarLRx`, because of two kinds of parsing.

`GrammarLL` defines methods such as `parseIt`, `doParseTable`, `expand`, `compare`, `accept`, and `checkLL1`. The methods are tightly coupled with the *top–down* parsing.

`GrammarLRx` is the base class for the *bottom–up* parsing. It defines methods such as `EFF`, `before`, `doExtendedGrammar`, and `checkExtendedGrammar`. Subclasses of the `GrammarLRx class` define common methods for the *bottom–up* parsing, such as `goto`, `getLRItems`, `shift`, `reduce`, as well as methods joined with some kind of LR grammar, such as `doParseTable`, and `parseIt`.

The *visual* part of the parser is a *MDI window*. The visual part is linked with the model via messages and *dependencies*. At the end of an action is the content of the visual part changed. The result is displayed. The user's interface, grammar's definition, input string, and the output are displayed in Fig. 1. The grammar is a LALR(1) grammar. The input string is `bgd`. The output `TextPane` contains all states of the parser from the *initial state* to the *final state*.

4 Step–by–Step Simulator

The parser described above has a disadvantage. It displays *partial results* and the *final result* at the same time. It is much more better for understanding to watch the states of the parser step–by–step from the beginning of the parsing to its end. For such a view see Fig. 2.

On Fig. 2 is displayed a state of the pushdown automaton, called parser. A parser is controlled by the `parsing table`. The parser reads symbols from the `input tape`, and writes the *left parse* or *right parse* to the `output tape`. The parser uses the `stack` (pushdown store). `Next action` of the parser is displayed too. The window contains two buttons: `do it` – which does the displayed action, and `exit` – which terminates the step–by–step parsing.

The object implementation of the parser allows to extend it to the step–by–step simulator. The extension preserves the model of CFG, adds few methods to the parser model and uses new visual part for displaying of the parser's state.

The Fig 3 shows the ClassBrowser of the VisualWorks 1.0. The upper left pane contains two Grammar *categories*. The `Grammar Interface` category contains `GRWatchDialog class` which is responsible for step–by-step visualization and control of the parser model. The bottom pane of the window contains the `step` method. The method sends the step message to the grammar and then it displays current state of the parser, until the end state or exit action is reached.

The polymorphism, the base object paradigm, allows to send all the time only the simple unary message `printNextAction`, no matter on the base class of a grammar. This message always displays the proper next action as well, as the proper parse table is displayed.

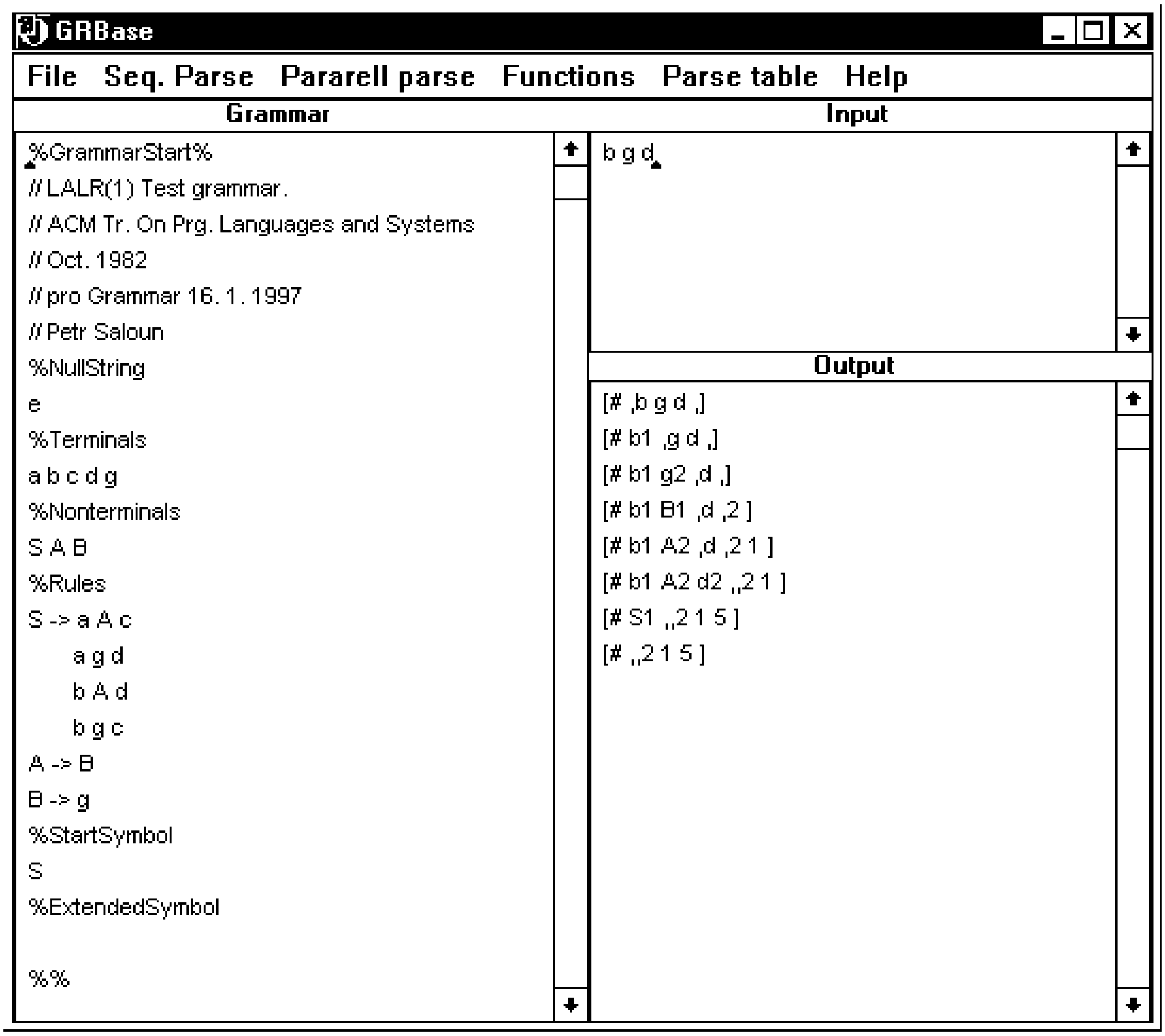

Figure 1: LALR1 Grammar and it's Right Parse.

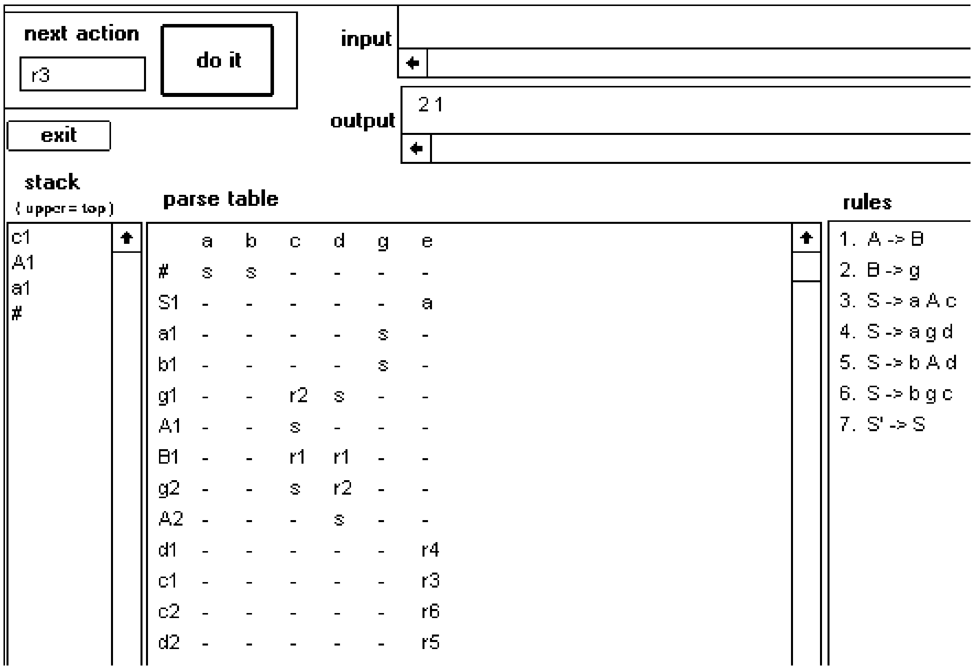

Figure 2: State of the Parser.

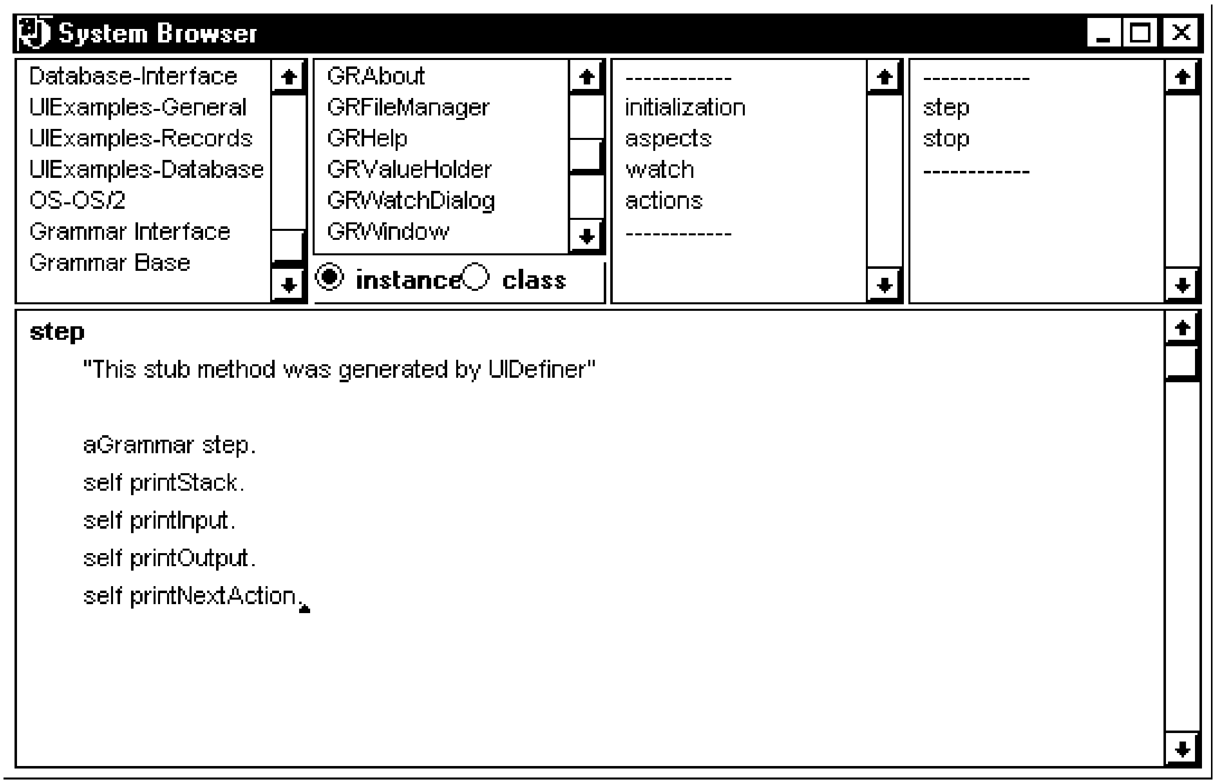

Figure 3: GRWatchDialog's method Step.

5 Conclusion

The paper describes the new *step–by–step* parser as the extension of the CFG model and parser model developed earlier. This extension is used by students in computer labs for self–studying of *Compilers* subject. Visualizer is available for Visual Works 1.0 and 2.5 respectively. Nowadays it is ported to the VisualAge for Smalltalk.

References

[ASU87] Aho, A.V. – Sethi, R. – Ullman, J.D.: *Compilers. Principles, Techniques, and Tools.* Addison-Wesley, 1987.

[Dig92] Digitalk: *Smalltalk/V: Tutorial and Programming Handbook.* Digitalk, 1992.

[Sal94] Šaloun, P.: *Syntax Analysis and Object Oriented Approach.* (in Czech). Proceedings of Modern Mathematical Methods in Engineering, JČMF, 1994, 110–114 pp, 1994.

[Sal96] Šaloun, P.: *Parallel Parsing Simulation.* Proc. of International Conference MOSIS'96, vol. 2. Krnov 1996, 148–153 pp.