Target Code Selection by Tilling AST with the Use of Tree Pattern Pushdown Automaton

Jan Janoušek and Jaroslav Málek

Department of Theoretical Computer Science
Faculty of Information Technologies
Czech Technical University in Prague
Technická 9, 160 00 Prague 6, Czech Republic
Jan.Janousek@fit.cvut.cz

Abstract

A new and simple method for target code selection by tilling an abstract syntax tree is presented. As it is usual, tree patterns corresponding to target machine instructions are matched in the abstract syntax tree. Matching tree patterns is performed with the use of tree pattern pushdown automaton, which accepts all tree patterns matching the abstract syntax tree in the linear postfix bar notation and represents a full index of the abstract syntax tree for tree patterns. The use of the index allows to match patterns quickly, in time depending on the size of patterns and not depending on the size of the tree. The selection of a particular target instruction corresponds to a modification of the abstract syntax tree and also a corresponding incremental modification of the index is performed. A reference to a fully functional prototype is provided.

1998 ACM Subject Classification D.3.4 Processors: Software, Programming languages, Code Generation, Compilers

Keywords and phrases code generation, abstract syntax tree, indexing, tree pattern matching, pushdown automata

Digital Object Identifier 10.4230/OASIcs.SLATE.2014.159

1 Introduction

A compiler backend transforms an intermediate code representation (IR), which is produced by a compiler frontend, to a target code [1]. One of the most used type of the IR is an abstract syntax tree (AST). The task of the target code selection from the AST can be performed by tilling the AST by tree patterns that correspond to target machine code instructions. This task is usually ambiguous and therefore often the best possible such tilling is to be found. For this reason a cost function of the particular machine code instructions is used so that the tilling with a minimal overall cost would be computed.

During the tilling of the AST tree pattern matching methods are used. For many linear notations of trees it holds that the linear notation of a subtree is a substring of the linear notation of the tree [12]. A tree pattern is a tree whose leaves can be labelled by a special symbol S, which serves as a placeholder for any subtree. A tree pattern in a linear notation corresponds to a substring of the linear notation of the tree, where the symbols S are replaced with the linear notations of subtrees. Therefore, tree pattern matching is analogous to the problem of matching patterns with specific gaps. See [2, 3, 8, 9, 12] for the basic tree pattern matching methods.

Since the problem of tilling the AST is generally ambiguous and NP-hard, many heuristics and methods are used for a "good" selection of particular instructions. Some of the methods perform one pass of the AST, some others perform more passes of the AST. From another

3rd Symposium on Languages, Applications and Technologies (SLATE'14).
Editors: Maria João Varanda Pereira, José Paulo Leal, and Alberto Simões; pp. 159–165

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

point of view, various models of computation are used for the description of target code selection methods. A code selection method based on deterministic finite tree automata can be found in [4], where the cost function is computed by an additional semantic evaluation. On the other hand, [7, 10, 13] describe the code selection methods based on deterministic pushdown automata performing the tree pattern matching, where the tree patterns are represented by rules of a context-free grammar, and in this way generally ambiguous and non-LR(0) context-free grammars are created. Consequently, the LR(0) parsers for those grammars contain conflicts. In [7] these conflicts are resolved by some heuristics; in [10, 13] a special construction of a deterministic parser is used, which corresponds to a determinization of the above–mentioned LR(0) parser with the conflicts. We mention also a family of tools BURG, IBURG, etc. (see [5, 6] for example), which use another model of computation, so-called tree rewriting systems, for the tree pattern matching in the code selection problem.

In this paper a new and simple method for target code selection by tilling an abstract syntax tree is presented. As it is usual, tree patterns corresponding to target machine instructions are matched in the AST. Matching tree patterns is performed with the use of tree pattern pushdown automaton, which accepts all tree patterns matching the abstract syntax tree in the linear bar postfix notation and represents a full index of the abstract syntax tree for tree patterns. Tree pattern pushdown automaton is described in details in [12]. The use of the index allows to match patterns quickly, in time depending on the size of patterns and not depending on the size of the tree. The selection of a particular target instruction corresponds to a modification of the AST and also a corresponding incremental modification of the tree pattern pushdown automaton is performed. Given an AST of size n, the number of distinct tree patterns which match the AST is $\mathcal{O}(2^n)$. For the sake of a better space complexity we use the nondeterministic version of the tree pattern pushdown automata. This is feasible and reasonable because tree patterns that correspond to target instructions are not typically large, and the space complexity of the nondeterministic tree pattern pushdown automaton is $\mathcal{O}(n)$. Experimental results of a fully functional prototype can be found in [11].

The new contributions of this paper are:

- The incremental modification of the tree pattern pushdown automaton for a specific modification of the AST.
- A simple use of the tree pattern pushdown automaton for the selection of target code, using modification mentioned in the previous item.

The rest of the paper is organised as follows. The tree pattern pushdown automaton, which is constructed for the AST, and its incremental modification are described in the second section. The third section describes the selection of a target instruction by tilling the AST. Experimental results of a fully functional prototype are presented in the fourth section. The last section is the conclusion.

2 Tree Pattern Pushdown Automaton and its Incremental Modification

Tree pattern pushdown automaton and its modification are demonstrated on a running example. Fig. 1 shows a tree t_1. Its linear postfix bar notation [12], for which the automaton is constructed, is $bar(t_1) = [[a[[b[aab$. The nondeterministic tree pattern pushdown automaton is illustrated in Fig. 2. This automaton accepts all tree patterns which match the tree in the postfix bar notation. In all figures of pushdown automata in this paper, the edges, which

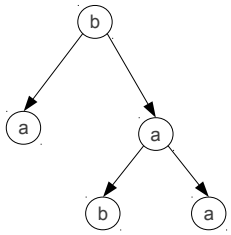

Figure 1 Tree t_1.

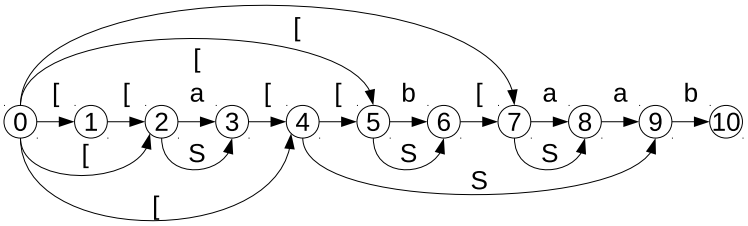

Figure 2 Tree pattern pushdown automaton for tree t_1.

represent transitions, are labelled only by an input symbol that is read by the transition and the pushdown operations are not illustrated. The pushdown operations ensure that a correct (sub)tree is processed in its linear notation. The automaton is input-driven, which means that each pushdown operation is unambiguously given by the input symbol. In our case, reading bar symbol [pushes one symbol onto the pushdown store, whereas reading other symbols pops one symbol from the pushdown store [12].

An example of tilling the AST by tree pattern [[b[aa in the postfix bar notation with a modification to tree pattern [[ed in the postfix bar notation is demonstrated in Figs. 3, 4 and 5. The formal algorithm of this modification can be found in [11].

3 Target Code Selection by Tilling AST

As it is mentioned in the Introduction, the tilling AST by target instructions is an ambiguous problem in general. The tree pattern pushdown automaton is used for computing a set of possible instructions that can be selected. In [11] the instructions that are selected are simply fixed from the set of all possibilities according to a cost function. We note that also other heuristics for the selection from the set of possibilities can be considered.

Fig. 6 shows the AST for statement x[3] = temp*(temp+10).

The AST in Fig. 7 is after the selection the instruction `store`. The tree pattern pushdown automaton is modified accordingly.

The AST in Fig. 8 is after the selection the instruction `add`. The tree pattern pushdown automaton is modified accordingly.

The next instruction to be selected is `load`, which occurs twice in the AST. Both occurrences are selected and the result is shown in Fig. 9.

The tilled AST is illustrated in Fig. 10. The tilling generates the following sequence of instructions:

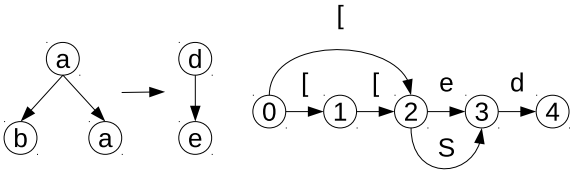

Figure 3 The selection of tree pattern with the modification of the AST and the corresponding part of the pushdown automaton.

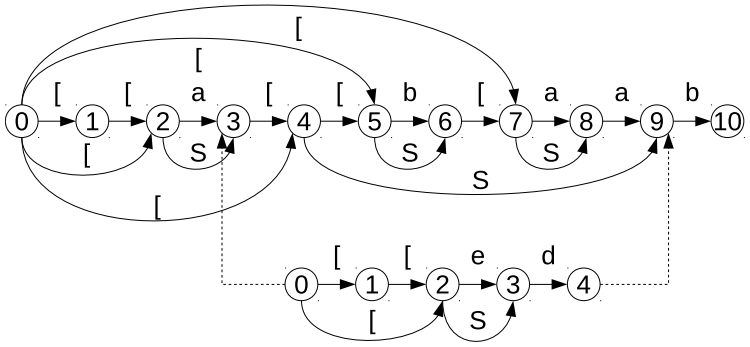

Figure 4 Part of the pushdown automaton to be modified.

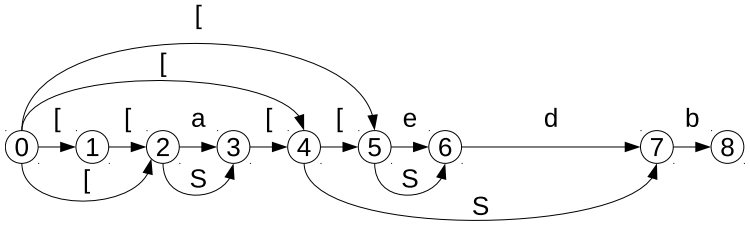

Figure 5 The resulting pushdown automaton after the modification.

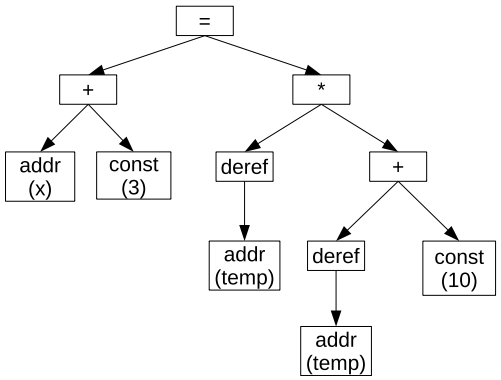

Figure 6 AST for statement x[3] = temp*(temp+10).

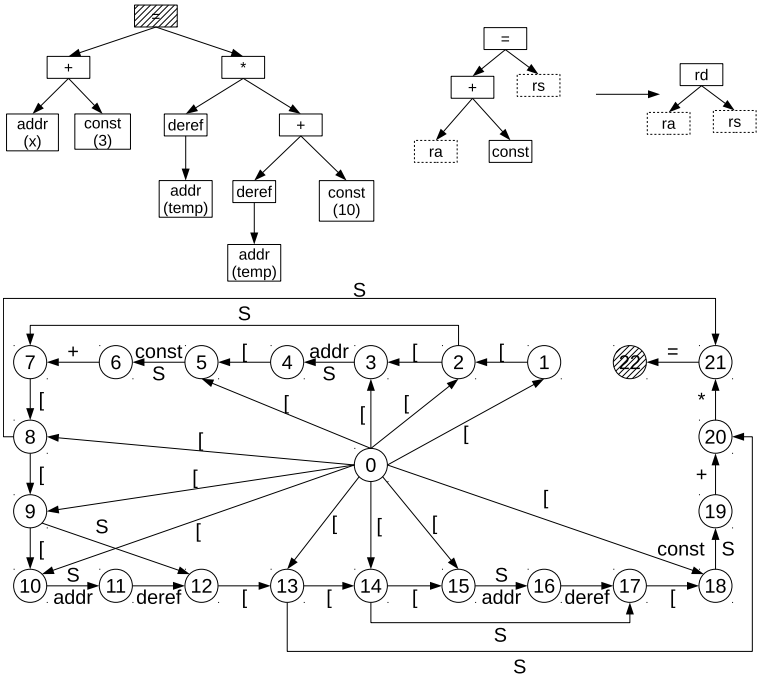

Figure 7 Tilling by tree pattern for instruction store.

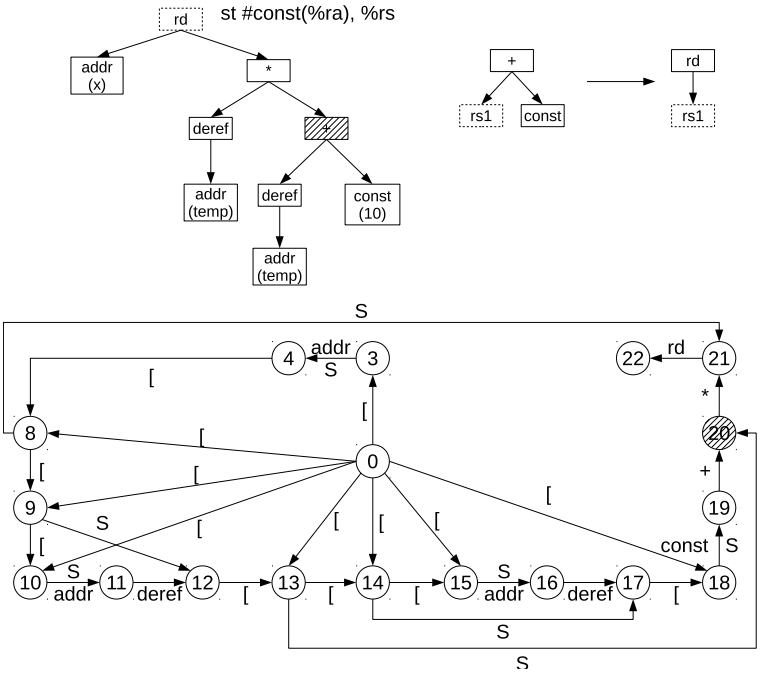

Figure 8 Tilling by tree pattern for instruction add.

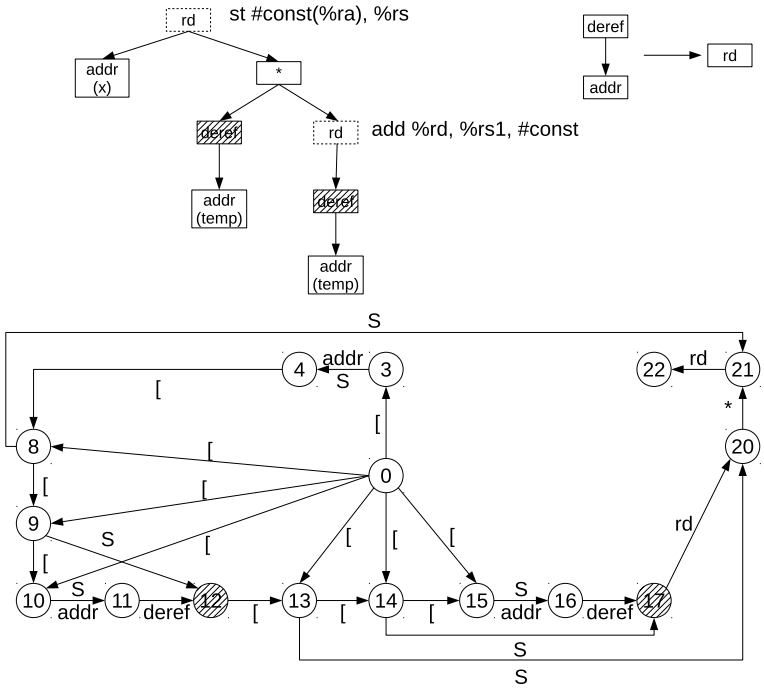

Figure 9 Tilling by tree pattern for instruction load.

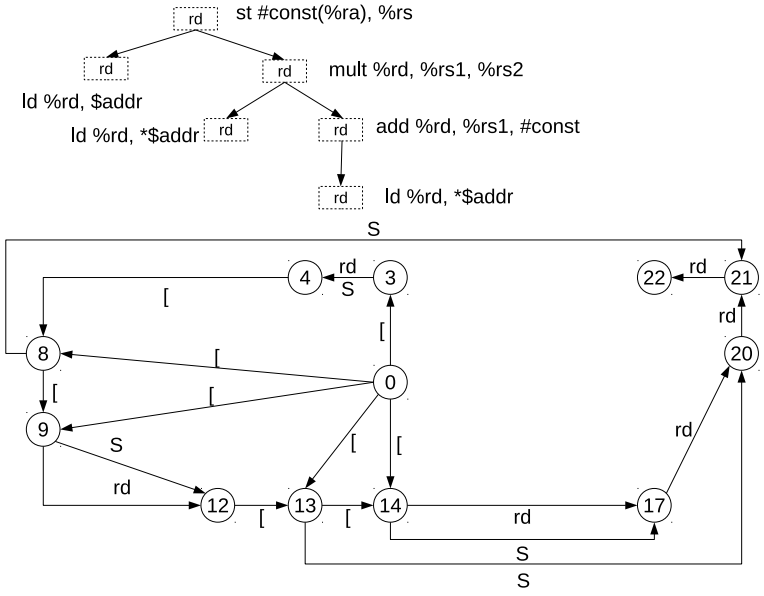

Figure 10 Tilled AST.

```
ld %rd, $addr
ld %rd, *$addr
ld %rd, *$addr
add %rd, %rs1, #const
mult %rd, %rs1, %rs2
st #const(%ra), %rs
```

4 Conclusion

A new and simple method of the code generation by tilling the AST with the use and incremental modification of the tree pattern pushdown automaton, which represents a full index of the AST for tree patterns, has been presented. More details, a prototype and its experimental results can be found in [11].

Acknowledgments. This work was partially supported by GAČR Grant No. GA13–03253S.

References

1 Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2Nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

2 P. Bille. *Pattern Matching in Trees and Strings*. PhD thesis, FIT University of Copenhagen, Copenhagen, 2008.

3 David R. Chase. An improvement to bottom-up tree pattern matching. In *ACM Symp. POPL*, pages 168–177, 1987.

4 Christian Ferdinand, Helmut Seidl, and Reinhard Wilhelm. Tree automata for code selection. *Acta Inf.*, 31(8):741–760, 1994.

5 Christopher W. Fraser, David R. Hanson, and Todd A. Proebsting. Engineering a simple, efficient code-generator generator. *LOPLAS*, 1(3):213–226, 1992.

6 Christopher W. Fraser, Robert R. Henry, and Todd A. Proebsting. Burg: fast optimal instruction selection and tree parsing. *SIGPLAN Notices*, 27(4):68–76, 1992.

7 R. Steven Glanville and Susan L. Graham. A new method for compiler code generation. In *POPL*, pages 231–240, 1978.

8 Christoph M. Hoffmann and Michael J. O'Donnell. Pattern matching in trees. *J. ACM*, 29(1):68–95, 1982.

9 Jan Lahoda and Jan Žďárek. Simple tree pattern matching for trees in the prefix bar notation. *Discrete Applied Mathematics*, 163, Part 3:343–351, January 2014.

10 Maya Madhavan, Priti Shankar, Siddhartha Rai, and U. Ramakrishna. Extending graham-glanville techniques for optimal code generation. *ACM Trans. Program. Lang. Syst.*, 22(6):973–1001, 2000.

11 J. Málek. Code generation with the use of an index of ast. *FIT, Czech Technical University in Prague, MSc thesis*, In Czech, 2014.

12 Bořivoj Melichar, Jan Janoušek, and Tomáš Flouri. Arbology: trees and pushdown automata. *Kybernetika*, 48, No.3:402–428, 2012.

13 Priti Shankar, Amitrajan Gantait, A. R. Yuvaraj, and Maya Madhavan. A new algorithm for linear regular tree pattern matching. *Theor. Comput. Sci.*, 242(1-2):125–142, 2000.