Compiler Module of Abstract Machine Code for Formal Semantics Course

William Steingartner

Technical University of Košice, Slovakia

SAMI 2021

IEEE 19th World Symposium on Applied Machine Intelligence and Informatics

January 21-23, 2021, Herľany, Slovakia

Introduction

Motivation

- Now the online (distant) teaching in the time of the world pandemic is very actual and possibly the one method for providing the full lectures.
- Educators face a great challenge, as in teaching online, without contact with students, to clearly explain various topics.
- In teaching the course Semantics of Programming Languages, we faced the problem of how to clearly explain the formal foundations of semantic methods.
- One of the suitable forms seemed to be the visualization and animation of methods using educational software.
- At present, when the contact of educators and students is limited mostly to the online events, it is the interactive teaching software tools that play an irreplaceable *rôle* in the teaching process.

Aim of our work

- The aim of this work is to describe a software tool for generating code for the abstract implementation of the language from input code in a language *Jane*.
- This application can be used within the course Semantics of Programming Languages.
- It provides exact translation from a text form written in the language *Jane* into the source code of assembler of an abstract machine for the structural operational semantics.
- One of the suitable forms seemed to be the visualization and animation of methods using educational software.

Language *Jane*

Language *Jane*

- a simple abstract language for defining the semantic methods and proving of their properties and equivalences is used;
- it is a non-real programming language grounded in imperative paradigm, epitomizing a tiny core fragment of conventional mainstream languages: standard imperative constructs as sequences of statements, selection (conditional), repetition (loops) and handling the values in memory (variables assignment);
- for defining formal syntax of *Jane* the following syntactic domains are introduced:

$n \in \mathbf{Num}$ – for digit strings;
$x \in \mathbf{Var}$ – for variable names;
$e \in \mathbf{Expr}$ – for arithmetic expressions;
$b \in \mathbf{Bexpr}$ – for Boolean expressions;
$S \in \mathbf{Statm}$ – for statements.

Language *Jane* – Syntax

The elements $n \in \mathbf{Num}$ and $x \in \mathbf{Var}$ have no internal structure from semantic point of view.

The syntactic domain $\mathbf{Expr}$ consists of all well-formed arithmetic expressions created by the following production rule

$$e ::= n \mid x \mid e + e \mid e - e \mid e * e \mid (e).$$

Boolean expression from $\mathbf{Bexpr}$ can be of the following structure:

$$b ::= \texttt{false} \mid \texttt{true} \mid e = e \mid e \leq e \mid \neg b \mid b \wedge b \mid (b).$$

As the statements $S \in \mathbf{Statm}$ we consider five elementary Dijkstra's statements:

$$S ::= x := e \mid \texttt{skip} \mid S; S \mid \texttt{if}\ b\ \texttt{then}\ S\ \texttt{else}\ S \mid \texttt{while}\ b\ \texttt{do}\ S.$$

Provably correct implementation

- A formal specification of the semantics of a programming language is useful when implementing it.
- This is usually realized as translation of the higher-level language into a structured form of assembler code for an abstract machine.
- An abstract machine is an intermediate language with a small-step operational semantics, it provides an intermediate language stage for compilation.
- First, the meaning of the abstract machine instructions are defined by an operational semantics.
- Then translation functions that will map expressions and statements in the higher-level language into sequences of such instruction are defined.
- The correctness result then states that if a program is translated into code and the code is executed on abstract machine then the same result must be provided as by semantic functions for natural or structural operational semantics.

Specification of AM

The description of particular computational steps of abstract machine is usually given by configurations of the form

$$\langle c, \sigma, s \rangle ,$$

where

- c stands for a code – the sequence of instructions to be executed,
- σ is the evaluation stack, and
- s is a storage.

Semantic domain for stacks:

$$\mathbf{Stack} = (\mathbf{Z} \cup \mathbf{B})^* .$$

The language of abstract machine is a structured assembler, which consists of instructions:

$$\begin{array}{lll} instr & ::= & \mathtt{PUSH}-n \mid \mathtt{ADD} \mid \mathtt{SUB} \mid \mathtt{MULT} \mid \\ & & \mathtt{TRUE} \mid \mathtt{FALSE} \mid \mathtt{EQ} \mid \mathtt{LE} \mid \mathtt{AND} \mid \mathtt{NEG} \mid \\ & & \mathtt{FETCH}-x \mid \mathtt{STORE}-x \mid \\ & & \mathtt{EMPTYOP} \mid \mathtt{BRANCH}(c, c) \mid \mathtt{LOOP}(c, c) \\ & & \\ c & ::= & \varepsilon \mid instr : c \end{array}$$

Generating code for the abstract machine

A code for the abstract machine is generated by the translating functions:

$$\mathscr{T}\mathscr{E} : \textbf{Expr} \rightarrow \textbf{Code}$$

$$\mathscr{T}\mathscr{B} : \textbf{Bexpr} \rightarrow \textbf{Code}$$

$$\mathscr{T}\mathscr{S} : \textbf{Statm} \rightarrow \textbf{Code}$$

Example of translation:

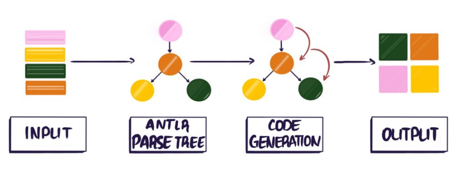
$$\begin{aligned}
\mathscr{T}\mathscr{E}[\![n]\!] &= \quad \texttt{PUSH}-n \\
\mathscr{T}\mathscr{E}[\![x]\!] &= \quad \texttt{FETCH}-x \\
\mathscr{T}\mathscr{E}[\![e_1 + e_2]\!] &= \quad \mathscr{T}\mathscr{E}[\![e_2]\!] : \mathscr{T}\mathscr{E}[\![e_1]\!] : \texttt{ADD} \\
\mathscr{T}\mathscr{E}[\![e_1 - e_2]\!] &= \quad \mathscr{T}\mathscr{E}[\![e_2]\!] : \mathscr{T}\mathscr{E}[\![e_1]\!] : \texttt{SUB} \\
\mathscr{T}\mathscr{E}[\![e_1 * e_2]\!] &= \quad \mathscr{T}\mathscr{E}[\![e_2]\!] : \mathscr{T}\mathscr{E}[\![e_1]\!] : \texttt{MULT}
\end{aligned}$$

Program specification

- the input code is written in *Jane* language;
- after reading an input source code (input program), the task of the application is to find out whether the given input is syntactically correct according to the rules of the *Jane* language;
- in case of an incorrectly given input program, the application provides an error message referring to the problem in a source;
- the user can choose from two types of listings of the resulting program:
  - the first type is a direct result – this output form contains only the generated instructions of the abstract machine;
  - the second type is an extended form with particular steps of generating the output, it contains the entire sequence of instruction generation;

Program specification

- Program was developed as a part of educational project KEGA 011TUKE-4/2020: „A development of the new semantic technologies in educating of young IT experts";
- the main task of this application is the translation of an input code written in *Jane* language into the instructions of an abstract machine for the structural operational semantics;
- program reads an input, provides lexical and syntax analysis and produces the output code: input code is converted via an ANTLR parse tree and code generation to output code;
- since the input code is a source in one language and output code is again a source in different language, such kind of compiler is known also as a source-to-source compiler:



Implementation of a parser

- for the implementation of a grammar for *Jane* programming language, we decided to use *ANother Tool for Language Recognition* – ANTLR;
- the grammar contains rules for parser and lexer;
- for the language Jane, a full grammar contains rules to which we assign a transcription to abstract machine code;
- the grammar is written in the form of extended Backus-Naur form.

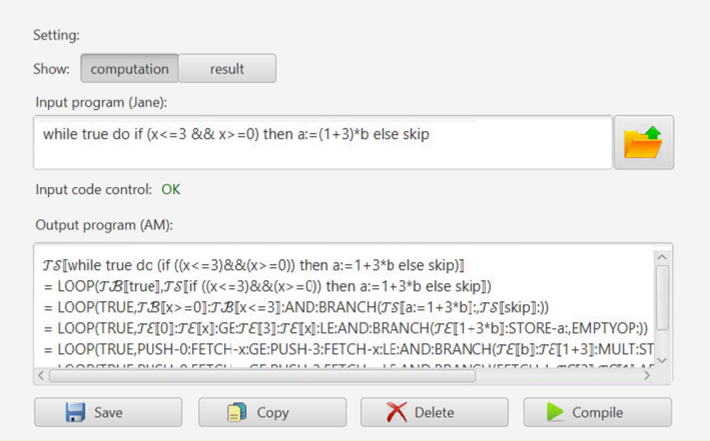
Part of grammar:

```
grammar Grammatik;

sequence: command (';' command)* (';')?;
command: assignment | if_condition
      | while_loop | comparison
      | operation | SKIP_RULE;
assignment: NAME ':=' (operation);
operation: '(' operation ')'
 | <assoc=right> operation MULT operation
 | <assoc=right> operation SYMBOL operation
 | value;
value: NAME | NUMBER;
```

Teaching tool

Graphic User Interface



Conclusion

- We presented a software tool which serves as source-to-source compiler from an abstract language *Jane* to the instructions of abstract machine for structural operational semantics.
- Our application is ready for use in the teaching process in classes and during the online teaching for both teachers and students.
- Because the distant teaching plays a crucial *rôle*, we consider as the main contribution of our work an interactive method of providing information during the educational process using the visualizing software tools.
- Our aim is to prepare complex learning environment for an attractive teaching the formal methods that are grounded in semantics.
- We are convinced that visualization tools will help to significantly understand and make the teaching of formal methods more attractive for future IT specialists.

Thank you for your attention