A Visualizing Tool for Graduate Course: Semantics of Programming Languages

Steingartner, William; Perháč, Ján and Biliňski, Alexander

Abstract: ***The aim of this paper is to describe a simple software tool that is used for translation of a code written in model language Jane into an inner code in JSON format. This inner code is a base for visual simulation of the semantics of a program given as input. Our software tool represents a simulation of semantics method based on category theory – categorical denotational semantics. The purpose of this software tool is to help students to understand an abstraction of categorical representation of denotational semantics of imperative programming languages.***

Index Terms: ***categorical semantics, compiler, Java programming language, learning tool, visualizing tool***

1. Introduction

More and more, computer science makes use of formal models to aid the understanding of complex software systems and to reason about their behavior. For purposes of teaching, there are many useful software tools that help to understand the formal background of software development. All of these tools and techniques are grounded in formal models of system execution which are themselves rooted in the formal semantics of the underlying programming languages.

We bring in this work a description of a software tool needed for teaching the course Semantics of programming languages. Teaching young software experts in the field of formal methods is quite a big challenge now. A lot of formal methods require deep knowledge of mathematical principles and interconnection or association to other parts of theoretical computer science. Because semantics plays a crucial *rôle* in the field of formal methods, it is usually part of a graduate or undergraduate curriculum. Students attending the course on Semantics learn the principles of reasoning about programs, and they will be able to apply semantic methods to find critical places in programs or to find a result of a program before real running it on a machine.

Semantic methods provide a meaning of particular program constructs even of the whole program. Using semantic methods students can find out whether the program has correct behavior or whether it provides the final states with expected results. On the other hand, semantic methods can identify the situations when the program cannot terminate correctly (when it simply aborts or fails, roughly speaking when it crashes) or they can help with solving the semantic errors.

Our software tool is able to analyze input source code in a model language *Jane* presented in [16] or in [18]. By language Jane we mean standard model (non-real) imperative language which consists of all standard (universal) imperative constructs: assignment, empty step, sequencing of statements, two-ways conditional statement and loop statement (prefix logical cy-

Manuscript received Mar, 2019.

This work was supported ▪ by the Slovak Research and Development Agency under the contract No. SK-AT-2017-0012: Semantics technologies for computer science education; ▪ by the project KEGA 002 TUKE - 4/2017 Innovative didactic methods in education at university and their importance in increasing education mastership of teachers and development of students competences); and ▪ by the funding of Faculty of Electrical Engineering and Informatics, Technical University of Košice under the contract No. FEI-2018-59: Semantic Machine of Source-Oriented Transparent Intensional Logic.

The authors are with the Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia (e-mail: william.steingartner@tuke.sk, jan.perhac@tuke.sk, alexander.bilinski.93@gmail.com).

cle). In some publications, this language is referred to as language While [10]. Such kind of model language is easy to understand and simply transformable to any real imperative programming language.

After reading a source, program realizes standard lexical, syntax and semantic analysis. During an analysis pass, the input code is analyzed and all variables are found. The program provides results in visual form – a graph which represents a path in the category of states.

Categorical representation of denotational semantics, or simply categorical denotational semantics was presented in [15, 17, 18]. This method is very well-acceptable for students of a graduate course on semantics and our tool is ready to help students to figure out and to understand how a program is modeled in the mentioned semantic method.

Furthermore, the program provides also a state table – a tabular representation of memory states during the program execution. By state, we mean a mathematical abstraction of memory, a snapshot of actual variables used by the program. For the definition of state representation is reader referred e.g. to [10].

Our program can analyze the source, simulate the semantics of the program and draw the correct output graph of an input program. Moreover, it can provide an export of a drawn graph into a file in some standard graphical formats, an export of a state table in text format, and modifying and saving source code.

Language Jane can be extended with new syntactic constructs, then also its semantics must be extended and newly defined for its extensions. Our program is ready to be extended for the new constructs in the grammar of Jane and their semantic properties.

We present in Section 2 some basic aspects, notions and introduction to the semantics of programming languages and to language Jane. In Section 3, which is the main part of this paper, we present an implementation of our solution with some example. In Section 4, we present a simple example of using our program. The last section – Conclusion (Section 5) brings the summary and some new plans in our future work.

2. Basic Aspects about Semantics of Programming Languages

The semantics of programming languages is taught as a graduate or undergraduate course. During this course, students learn how selected semantic methods provide a result of a given program and its behavior. Each semantic method uses precise semantic rules for providing the meaning of particular program constructs [6, 7]. Although a lot of semantic methods are known, mostly the following ones are taught: operational semantics [11], denotational semantics [13], algebraic [4] or axiomatic semantics [5] (introduced also in [10]), and kind of new experimental method – action semantics [9]. One of the new methods is categorical denotational semantics – a method presented in [18] which provides the meaning of program constructs based on mathematical structures defined in category theory.

2.1 Formal Semantics

In programming languages theory, semantics is a field concerned with the rigorous mathematical study of the meaning of programming languages. Formal semantics is focused only on providing the results of correct programs without considering the implementation details or the technical (hardware) details of a concrete machine on which the program is being executed. All those details are ignored. Formal semantics focuses on the relationship between an input program and its meaning as the output, correct program's termination or on the reason why the program cannot terminate correctly.

2.2 Categorical Denotational Semantics

Category theory [1, 2] is a theoretical framework providing very useful properties for defining the semantics. A category is a mathematical structure consisting of a class of objects and class of morphisms, together with some other conditions (identity morphism for each object, composition of morphisms, associativity of composition).

There are known some approaches, like categories of arenas for game semantics or category of types for denotational semantics de-

fined by J.-Y. Girard [3]. Our approach of defining the denotational semantics in categorical structures models behavior of running program in such way that particular memory states are separate objects of the category of states, and mappings from state to state are category morphisms. As an impact of this method, we emphasize the graphical (visual) representation where the executed program is modelled as a path in category consisting of (possibly) more states. Because of some definitions of a category in [1], we can consider a visual model of a program as a graph where nodes (vertices) are states and mappings are edges. Such visual model is easy to understand for reasoning about program behavior. Similar approach on visualization has been used in [14].

2.3 Jane Programming Language

For teaching the basic principles of programming languages, their syntax and semantics, none of the concrete languages is suitable. Each language is based on its own philosophy, with some details that sometimes cannot be applied in other languages. From this point of view, a need for some "universal" language is required. There are some non-real languages presented in the literature which are suitable for teaching, explaining and observing the standard properties and behavior. One of those languages is well-known the so-called *While* example language, presented in [10]. We adopted this language and we have been inspired by its syntax. From the pedagogical reasons, we refer to this language as Jane. This language contains a standard assignment of an expression into a (possibly declared) variable, empty step, sequencing of statements, two-ways conditional statement and prefix loop statement. The basic syntax of the Jane language is as follows:

$$\begin{aligned} S ::= \quad & x := e \mid \texttt{skip} \mid S; S \\ & \mid \texttt{if } b \texttt{ then } S \texttt{ else } S \\ & \mid \texttt{while } b \texttt{ do } S \end{aligned}$$

where S stands for a meta-variable, an element of syntactic domain for well-structured statements, e and b represent well-structured arithmetic or Boolean expression, resp.:

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

$$\begin{aligned} b ::= \; & \texttt{true} \mid \texttt{false} \mid \neg b \mid b \wedge b \mid e = e \\ & \mid e \leq e \mid (b) \mid \ldots \end{aligned}$$

Jane language is for simplicity implicitly typed: arithmetic expressions are of integer type and they can be assigned into variables. Boolean values are of Boolean two-value type (not a subtype of integer) and they can be used only as transient expressions in conditional or loop statement – they cannot be assigned to variables nor stored in memory. For a deeper description of Jane, the reader can access resources, e.g. [16] or [18].

3. Implementation of our Solution

In this section, we describe our software solution: how our program is developed; and its main functions.

3.1 An Input Code

Our software tool accepts an input code prepared in Jane language. Input code can be inserted manually or loaded from a file. Then, the user can assign initial values to the variables manually. When starting the process of program analysis, standard lexical, syntax and semantic analysis are executed. If during the analysis some lexical, syntax or semantic errors are found, the program displays an error message with the description of the error. If the program does not contain any error, then it is ready for visualization. All program steps are evaluated and prepared for final visual simulation as particular parts of final program graph (path in category). In this section, we describe our software solution: how our program is developed; and its main functions.

3.2 Processing the Input Strings

After the analysis, usually, the phase of generating the code follows. Here, our program provides also some kind of generating the output form: input program is transformed to inner representation understandable for visualizing module. Inner form of the input program

is represented in JSON format. We also used an XML format in some previous approaches [8]. But JSON format seems to be more effective and shorter. Philosophy is, that any object of the program (a statement, expression etc.) has some pattern which represents a given object. As an example, we take the assignment statement. A key that represents an assignment statement together with an expression (R-value) to be assigned are stored in JSON format as follows:

$$\{\text{"assign"} : \text{"x} - \text{y"}\}.$$

Another example is when a statement to be executed is located in some block, e.g. in a conditional statement, in any of its branches. In this case, statements are stored in an array. JSON format allows putting more elements of one key (more statements here) into one object, e.g.

$$\{\text{"thenBlock"} : [\{\text{"assign"} : \text{"x} - \text{y"}\}, \{\text{"assign"} : \text{"x} + \text{y"}\}]\}.$$

Using this approach, program builds up the whole program as a JSON object. When an input program is translated, an inner form can be used for any other purpose – for example for simulation in another semantic method. If the analysis of an input program finishes without errors, then it is ready for simulation.

3.3 Simulation of Program Execution – the Semantics of Program

When an input program is translated into JSON format without errors, it can be visualized in particular steps. In this phase, our software simulator works only with the inner form of input converted to JSON format.

A simulation of program steps works as follows:

- the whole JSON object for the program is read;
- all fields that represent assignments are evaluated as changes of states (actualizations) – for every variable in expression the last value from a state stored under the appropriate key is used for calculation of a new value which is then stored into a new actualized state;
- similarly, when the Boolean expression in conditional or loop statement is found in the JSON object, it is evaluated and its Boolean value is stored under the appropriate key;
- based on values of conditions located in conditional or loop statements, reading of JSON object is tailored to those constructs.

The main result of the program simulation is then an object of states and variables located inside the states together with their values. This object is then used when simulator handles the table of variables and states and for the visualization.

3.4 Drawing the Output

The visual output of the software tool is a graph of nodes that represent the states and edges as mappings among them. This graphical representation is considered also as a part of the category of states. All parts of the graph are drawn based on simulation steps and table of variables.

The process of drawing the graph is active during the simulation of an input program. During the simulation, the behavior of the program is followed. For example, when the condition is found, based on its value can simulator continue by choosing the correct branch in a conditional statement or to unfold the next iteration step in loop statement.

At the beginning of a simulation, a JSON object with the program is posted to the simulation module. The JSON object is read in particular steps and each step is after its evaluation drawn into output canvas. Particular steps of a program stored in JSON object are evaluated according to the semantic rules for categorical denotational semantics (see [18]).

As it was mentioned, programs in Jane contain mostly assignment statements which change the memory states and in visual output, they are represented by simple oriented edge (an arrow) between two nodes – states. The other possible constructs are conditional

and loop statements. Both these constructs are defined as functions that read their inner blocks of instructions (steps) based on Boolean expressions and they draw their particular steps out. An empty statement can also be included in the program, and it is drawn as an (endo-) arrow over the one node.

A conditional statement is evaluated as follows: first, the Boolean expression is evaluated. Then the correct sequence of statements is chosen according to the value of the condition.

Loop statement is evaluated as follows: its inner block is repeated while the Boolean condition is true and every iteration is drawn as a separate step. When the condition is false, an evaluation of a loop is finished and one more arrow connecting both outermost states of the loop statements (the first and the last one) is drawn to emphasize the semantics of the loop statement. This new arrow represents the composition of all arrows inside the visualization of a loop statement.

4. Example of Using the Software Tool

As an example, we show how our software tool is being used. Let's take the following code for computing the integer quotient and remainder after division:

```
begin
  z := 0;
  while (y ≤ x) do (
    z := z + 1;
    x := x − y; )
end
```

Let the initial values in an initial state s_0 be: $s_0x = \mathbf{17}$, $s_0y = \mathbf{5}$, or simply $s_0 = [x \mapsto \mathbf{17}, y \mapsto \mathbf{5}]$ (see Figure 1).

The categorical denotational semantics of the given program (denoted as P) is expressed as follows:

$$[\![P]\!] =$$

$$[\![\texttt{while } (y \leq x) \texttt{ do } (z := z + 1; x := x - y)]\!] \circ$$

$$\circ\, [\![z := 0]\!]$$

After assigning the initial state s_0 into this semantic function, the program after an analysis of an input and building up the JSON object draws the visual output which represents the following categorical expression of the semantic function given as a composition of particular morphisms:

$$[\![P]\!] =$$

$$([\![x := x - y]\!] \circ [\![z := z + 1]\!])^3 \circ [\![z := 0]\!]$$

where the upper index represents (for simplicity) the number of iterations.

The (traditional) denotational semantics of an input program is then:

$$[\![P]\!]s_0 = s_7,$$

and in categorical denotational semantics expressed as

$$[\![P]\!] : s_0 \to s_7.$$

The particular steps during the program execution are in a state table depicted in Figure 1.

States	Variables	Values
State 0	x	17
State 0	y	5
State 0	z	0
State 1	z	0
State 2	z	1
State 3	x	12
State 4	z	2
State 5	x	7
State 6	z	3
State 7	x	2

Figure 1: States during the program execution (simulation)

The complete composition of morphisms which represent a path in a category of states is depicted in Figure 2.

The screen capture of the program window with an input code, complete simulation and state table is depicted in Figure 3.

After finishing the simulation, we obtain correct results: quotient is $\mathbf{3}$ and modulus $\mathbf{2}$, and the values are stored in the state s_7.

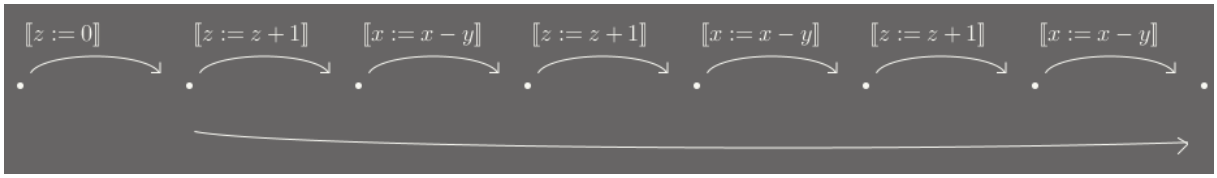

Figure 2: Composition of morphisms representing the path in category

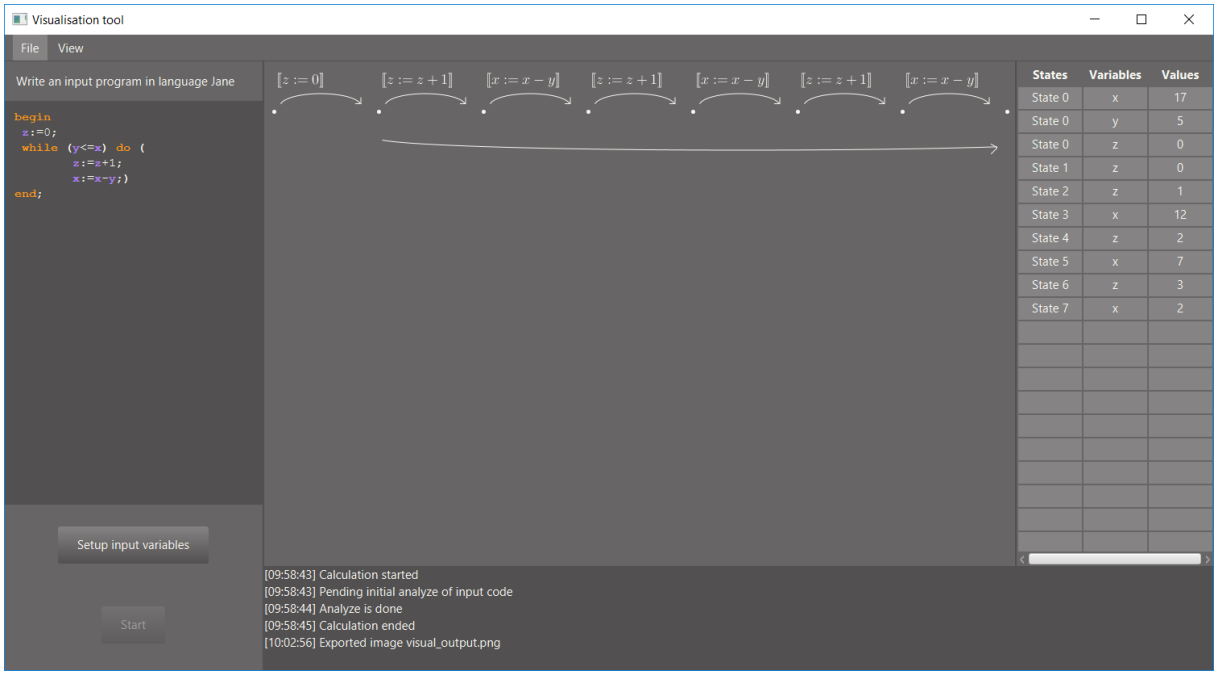

Figure 3: Software simulation tool

5. Conclusion

We presented in this work a software tool developed as teamwork mainly for the course on Semantics of programming languages that is taught at our university as a graduate course. The course on Semantics is considered as one of the main courses oriented to formal methods taught for young engineers and future IT-experts. This course covers more well-known semantics methods, as natural, operational or denotational semantics. On the other hand, there exist also some methods that are considered as new or experimental, e.g. action semantics. In our research, we developed a method which is based on mathematical structures grounded in category theory. We could say that this approach is another representation of well-known denotational semantics. Furthermore, it provides a very easy representation of the semantics of programs written with standard imperative constructs. For this reason, we use an example (toy) language Jane. Our application is developed as a standard compiler transforming input source code to inner code which serves for generating the output. In future, the application can be extended when a new element is included in syntax and its semantics is defined. We plan to use this program in the teaching process as a software tool for demonstrating the method and to integrate it into a complex software package that is being developed for the course on Semantics. The complex software package will contain tools for the main semantics methods and it could be also used for teaching using an OpenLab project [12], the open laboratory project hosted by Department of Computers and Informatics at the Technical University of Košice.

References

[1] Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[2] Martin Brandenburg. *Einführung in die Kategorientheorie*. Springer Spektrum, Berlin, Heidelberg, 2017. (in German).

[3] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, New York, NY, USA, 1989.

[4] Joseph A. Goguen and Grant Malcolm. *Algebraic Semantics of Imperative Programs*. MIT Press, Cambridge, MA, USA, 1996.

[5] Charles A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, Vol. 12(10):pp. 576—-580, 1969.

[6] Harley D. Eades III. *The semantic analysis of advanced programming languages*. PhD thesis, University of Iowa, 2014.

[7] Wolfgang Jeltsch. Categorical semantics for functional reactive programming with temporal recursion and corecursion. In Paul Levy and Neel Krishnaswami, editors, Proceedings 5th Workshop on *Mathematically Structured Functional Programming,* Grenoble, France, 12 April 2014, volume 153 of *Electronic Proceedings in Theoretical Computer Science*, pages pp. 127–142. Open Publishing Association, 2014.

[8] Žaneta Kochaníková, William Steingartner, and Mohamed Ali M. Eldojali. A code generator for an abstract implementation of imperative language. In *Electrical Engineering and Informatics VIII*, pages 342–347. Technical University of Košice, 2017.

[9] Peter D. Mosses. *Action Semantics*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

[10] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications: An Appetizer*. Undergraduate Topics in Computer Science. Springer, 2007.

[11] Gordon D. Plotkin. The origins of structural operational semantics. *The Journal of Logic and Algebraic Programming*, Vol. 60-61:pp. 3–15, 2004. Structural Operational Semantics.

[12] Jaroslav Porubän. Challenging the education in the open laboratory. In *2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages pp. 439–444, 2018.

[13] David A. Schmidt. *Denotational Semantics: A Methodology for Language Development*. William C. Brown Publishers, Dubuque, IA, USA, 1986.

[14] Wolfgang Schreiner and William Steingartner. Visualizing execution traces in RISCAL. Technical report, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, March 2018.

[15] William Steingartner and Valerie Novitzká. A new approach to semantics of procedures in categorical terms. In *2015 IEEE 13th International Scientific Conference on Informatics*, pages pp. 252–257, Nov 2015.

[16] William Steingartner and Valerie Novitzká. *Sémantika programovacích jazykov*. Technical University of Košice, 2015. (in Slovak).

[17] William Steingartner and Valerie Novitzká. Categorical model of structural operational semantics for imperative language. *Journal of Information and Organizational Sciences*, Vol. 40(No. 2):pp. 203–219, 2016.

[18] William Steingartner, Valerie Novitzká, Michaela Bačíková, and Štefan Korečko. New approach to categorical semantics for procedural languages. *Computing and Informatics*, Vol. 36(No. 6):pp. 1385–1414, 2017.

William Steingartner works as Assistant Professor of Informatics at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia. He defended his PhD thesis "The Rôle of Toposes in Informatics" in 2008. His main fields of research are the semantics of programming languages, category theory, compilers, data structures and recursion theory. He also works with software engineering.

Ján Perháč was born in Svidník, Slovakia. In 2015 he graduated (MSc.) at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University of Košice. Currently, he is a PhD student in the same department. He is concerned with GNU/Linux operating systems, network security, non-traditional logical systems, and category theory.

Alexander Biliňski graduated in 2018 at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University of Košice. The topic of his thesis was "Visualisation tool for course Semantics of programming languages". Currently, he works as Application Developer in T-Systems Slovakia s.r.o.