JAM CM Journal of Applied Mathematics and Computational Mechanics 2024, 23(2), 79-92
www.amcm.pcz.pl p-ISSN 2299-9965
DOI: 10.17512/jamcm.2024.2.07 e-ISSN 2353-0588

PROOF OF EQUIVALENCE OF SEMANTIC METHODS FOR A SELECTED DOMAIN-SPECIFIC LANGUAGE

William Steingartner[1], Valerie Novitzká[1], Wolfgang Schreiner[2]

[1] *Faculty of Electrical Engineering and Informatics, Technical University of Košice*
Košice, Slovakia
[2] *Research Institute for Symbolic Computation, Johannes Kepler University*
Linz, Austria
william.steingartner@tuke.sk, valerie.novitzka@tuke.sk, wolfgang.schreiner@risc.jku.at

Received: 6 December 2023; Accepted: 17 February 2024

Abstract. This paper focuses on the formal semantics of programming languages, with a specific focus on Domain-Specific Languages (DSLs). It introduces the *Robot* DSL, characterized by total semantic functions, an infinite network size, and an obstacle-free environment. The study explores denotational and natural semantics, aiming to define and prove their equivalence. This work contributes to the understanding of programming languages with unique features, laying the groundwork for future developments in language design and formal semantics.

MSC 2010: *68Q55, 00A71, 97P20, 97Q99*
Keywords: *denotational semantics, domain-specific language, language design, natural semantics, proof of equivalence, semantic function, structural induction*

1. Introduction

In the field of formal semantics, the study and analysis of programming languages have been crucial in advancing our understanding of computational processes. This paper provides an overview with a particular focus on domain-specific languages. With this article, we follow up on previous research to delineate the environment of domain-specific languages and their role in shaping different computing domains.

A domain specific language is a programming language with a higher level of abstraction optimized (tailored) for a specific class of problems [1]. A DSL uses the concepts and rules from the field or domain. A domain specific language is usually less complex than a general-purpose language, although the boundary is not clear as it could be [2]. DSLs offer significant value as they simplify programming compared to traditional libraries, enhancing overall programmer productivity – an invaluable asset. Additionally, well-designed DSLs can facilitate improved communication with domain experts, addressing a key challenge in software development [3]. To provide a structured framework and the best practices for designing and implementing these specialized languages, design guidelines for DSLs were formulated [4]. In this paper,

we delve into the specific attributes of an artificial DSL we call *Robot* (for interested reader, more examples and the state of the art in domain-specific languages in real robotics can be found in [5]). This language is characterized by the use of total semantic functions, an infinite size of network (for simplicity) and the absence of obstacles, which opens the way for interesting possibilities in programming paradigms, especially in future research, considering the possibilities of its extension and thus and approximation to a real language. The research includes an analysis of the denotational semantics of the *Robot* language, offering insight into its mathematical underpinnings. Furthermore, examining the inherent semantics of *Robot* provides a more intuitive understanding of a kind of operational semantics. The denotational approach to this language was defined in [6]; we extended the pool of semantic methods by defining the natural semantics in [7].

Our goal is to show how to define those two semantic methods for the selected domain-specific language. For the execution of n-step statements, we present three ways to define denotational and natural semantics. We verify the correctness of our approaches by proving the equivalence of these two semantic methods for *Robot*. We focus on the mentioned approach so that it is easy, especially for students. We plan to apply it in teaching such that students learn to use the proof of equivalence for semantic methods.

By establishing the equivalence between different forms of semantics, we aim to contribute to the formal underpinnings of programming languages and deepen our comprehension of the intricacies associated with characteristics such as an infinite network and a obstacle-free environment. This exploration seeks to provide a foundation for future advances in language design and formal semantics.

The paper is organized as follows. In Section 2, we present a basic definition of the language *Robot*. Section 3 contains the definition of semantic domains and the representation of the concept of *state*. In Section 4, the basics of the denotational semantics, and in Section 5, the basics of the natural semantics of our language are introduced. Section 6 is the core of the paper, and it contains the proof of equivalence of both methods. Section 7 concludes our paper.

2. The definition of the language

In this section, we introduce the formal syntax of the language *Robot*, serving as a concrete example for illustrating the process of defining the formal semantics. This language is a simple domain-specific language that enables the robot to move across a two-dimensional orthogonal grid. The simplicity and clarity of the *Robot* language make it an ideal illustration for teaching the formal definition of DSLs, particularly within our mandatory course, *Semantics of Programming Languages*.

The actual position of the robot is determined by two coordinates x and y. The robot can move in the directions left, right, up, and down. For specifying the syntax, we need the following syntactic domains:

$n \in$ **Num** — strings of digits (numerals),
$d \in$ **Dir** — directions of movement,
$S \in$ **Statm** — statements.

The elements of the syntactic domain **Num** are strings of digits (numerals). This is from the semantics's point of view, a trivial syntactic domain without inner structure.

The structure of directions is defined by the production rule

$$d ::= \textbf{left} \mid \textbf{right} \mid \textbf{up} \mid \textbf{down}$$

and we define the structure of statements as follows:

$$S ::= d \mid d\ n \mid \textbf{reset} \mid \textbf{skip} \mid S;S.$$

Here

- d represents the movement of the robot by one step in the given direction;
- $d\ n$ represents the movement of the robot by n steps in the given direction;
- **reset** defines the movement to the starting position;
- **skip** is the empty statement;
- $S;S$ is a sequence of statements.

3. Semantic domains and semantic functions

Semantic domains are the sets containing meanings of program phrases. First, we define the semantic domains and then the semantic functions.

The basic semantic domain is the set $\mathbf{Z}$ of integers. The set **Point** expresses the position of the robot by its coordinates:

$$\textbf{Point} = \mathbf{Z} \times \mathbf{Z} \tag{1}$$

where any element $p \in$ **Point** has the form of an ordered pair $p = [x,y]$ with the first coordinate x and the second coordinate y. We consider a net of infinite size and we do not assume any obstacle in it yet.

To express a move in a particular direction, we can use the projections

$$\pi_i : \mathbf{Z} \times \mathbf{Z} \to \mathbf{Z}$$

for $i = 1,2$. Let $p = [x,y]$. Then $\pi_1(p) = x$ and $\pi_2(p) = y$.

The set **Point** can be considered as the state space of the robot. We denote the starting position of the robot by

$$p^* = [x_0, y_0]$$

that is determined by the user. The meaning of the language constructions is defined by the semantic functions. We define the semantic function $[\![\ ]\!]$ for strings of numerals

$$[\![\ ]\!] : \mathbf{Num} \to \mathbf{Z} \tag{2}$$

from strings of digits to integers by $[\![n]\!] = \mathbf{n}$, where $\mathbf{n} \in \mathbf{Z}$ denotes the integer represented by numeral n in the usual way (as for instance in [8]).

4. Denotational semantics of *Robot*

Denotational semantics [9] describes a program's meaning by semantic domains and functions between them. Depending on the starting position of the robot, denotational semantics provides its target position after the execution of a program.

The denotation of statements we define by the semantic function [6]

$$[\![\ ]\!]_{ds} : \mathbf{Statm} \to (\mathbf{Point} \to \mathbf{Point}),$$

which defines how a position of the robot is being changed. The subscript ds identifies the denotational semantic function. We assume in this present version that $[\![\ ]\!]_{ds}$ is totally defined; in later extensions of the language *Robot*, it could become partial.

Let $p = [x,y]$ be a position before executing a statement. We define the denotational semantics of the one-step statements, the sequence of statements and the special statements as follows:

$$\begin{aligned}
&[\![d]\!]_{ds}[x,y] = \begin{cases} [x \ominus \mathbf{1}, y], & \text{if } d = \mathbf{left}, \\ [x \oplus \mathbf{1}, y], & \text{if } d = \mathbf{right}, \\ [x, y \ominus \mathbf{1}], & \text{if } d = \mathbf{down}, \\ [x, y \oplus \mathbf{1}], & \text{if } d = \mathbf{up}, \end{cases} \\
&[\![\mathbf{reset}]\!]_{ds}[x,y] = [x_0, y_0], \\
&[\![\mathbf{skip}]\!]_{ds}[x,y] = [x,y], \\
&[\![S_1; S_2]\!]_{ds} = [\![S_2]\!]_{ds} \circ [\![S_1]\!]_{ds}.
\end{aligned} \tag{3}$$

The notation $[\![d]\!]_{ds}[x,y]$ indicates a one-step movement in a particular direction. For the statements with n steps, we introduce three possible approaches for how to define denotational semantics for $d\ n$. In these definitions, we assume $n \geq 0$, where the case $n = 0$ serves only for defining semantics. The following equalities hold in all definitions:

$$[\![d\ 0]\!]_{ds}[x,y] = [\![\mathbf{skip}]\!]_{ds}[x,y], \tag{4}$$

$$[\![d\ 1]\!]_{ds}[x,y] = [\![d]\!]_{ds}[x,y]. \tag{5}$$

I_{ds} The first approach is direct; it uses only the semantics of n:

$$\begin{array}{ll} [\![\textbf{left } n]\!]_{ds}[x,y] = & [x \ominus [\![n]\!], y], \\ [\![\textbf{right } n]\!]_{ds}[x,y] = & [x \oplus [\![n]\!], y], \\ [\![\textbf{up } n]\!]_{ds}[x,y] = & [x, y \oplus [\![n]\!]], \\ [\![\textbf{down } n]\!]_{ds}[x,y] = & [x, y \ominus [\![n]\!]]. \end{array} \tag{6}$$

II_{ds} The second approach defines the denotational semantics using rewriting:

$$[\![d\ n]\!]_{ds}p = [\![d\ (n-1)]\!]_{ds}([\![d]\!]_{ds}p) \tag{7}$$

for $[\![n]\!] > [\![0]\!]$ (where $[\![d\ 0]\!]_{ds}$ is defined by equation (4)). The first step $[\![d]\!]_{ds}p$ of a move is performed, and the following $n-1$ steps are repeatedly executed.

III_{ds} The third approach is inductive definition of the semantics. It uses the following equivalence:

$$[\![d\ n]\!]_{ds} \equiv [\![d]\!]_{ds}^{[\![n]\!]}.$$

Then we can define the semantics of n-steps movements as follows:

$$\begin{array}{l} [\![d]\!]_{ds}^{\mathbf{0}}p = p, \\ [\![d]\!]_{ds}^{\mathbf{n+1}}p = [\![d]\!]_{ds}([\![d]\!]_{ds}^{\mathbf{n}}p). \end{array} \tag{8}$$

5. Natural semantics of *Robot*

Natural semantics is a kind of operational semantics, sometimes called the semantics of *big steps*. This method was defined by Kahn in [10]. Its aim is to describe how states are changed after the execution of the statements, and it is mostly used for imperative languages. However, it could be applied also in the area of domain-specific languages. The meaning of the statements is specified by a transition system, and particular steps are expressed as transition relations.

Consider the syntax of *Robot* introduced in Section 2. The semantics of numerals is defined as in (2). The state space is the semantic domain **Point** of robot's positions defined in (1).

The basic notion of natural semantics for the language *Robot* is a configuration (an ordered tuple) $\langle S, p\rangle$ expressing that a statement S will be executed in a state $p = [x,y]$, i.e. in a position $[x,y] \in \textbf{Point}$. The execution of a statement S is described by a transition relation $\langle S, p\rangle \to p'$ where p' is a position of robot after execution of a statement S in a state p.

For natural semantics, we introduce the semantic function $[\![\ ]\!]_{ns}$ as:

$$[\![\ ]\!]_{ns} : \textbf{Statm} \to (\textbf{Point} \to \textbf{Point})$$

defined by

$$[\![S]\!]_{ns}\ p = \begin{cases} p', & \text{if } \langle S,p\rangle \to p', \\ \bot, & \text{otherwise.} \end{cases}$$

The index *ns* indicates the natural semantic function.

Function for natural semantics for the statements is defined by the derivation rules of the following form [8]:

$$\frac{\langle S_1,p_1\rangle \to p_2,\ldots,\langle S_n,p_n\rangle \to p'}{\langle S,p_1\rangle \to p'} \text{ (ns-name)}$$

where $S_1,\ldots,S_n$ are substatements of S, the statement S starts its execution in a state p_1 and ends in a state p'. The states $p_2,\ldots,p_n$ are the intermediate states during the execution of S. If a rule has no assumption, it is an axiom.

First, we define the natural semantics for the one-step moves, the sequence of statements and the special statements:

Let $p=[x,y]$. Then

$$\begin{array}{llll} \langle \textbf{left},[x,y]\rangle & \to [x\ominus [\![1]\!],y], & \langle \textbf{down},[x,y]\rangle & \to [x,y\ominus [\![1]\!]], \\ \langle \textbf{right},[x,y]\rangle & \to [x\oplus [\![1]\!],y], & \langle \textbf{up},[x,y]\rangle & \to [x,y\oplus [\![1]\!]. \end{array} \tag{9}$$

For the special statements:

$$\begin{array}{ll} \langle \textbf{skip},[x,y]\rangle & \to [x,y], \\ \langle \textbf{reset},[x,y]\rangle & \to [x_0,y_0]. \end{array} \tag{10}$$

For a sequence of statements $S=S_1;S_2$:

$$\frac{\langle S_1,p\rangle \to p',\quad \langle S_2,p'\rangle \to p''}{\langle S_1;S_2,p\rangle \to p''} \tag{11}$$

Now we define the semantics of movements n steps in three ways introduced for denotational semantics.

I_{ns} Move of n steps directly is defined by the following axioms:

$$\begin{array}{ll} \langle \textbf{left}\ n,[x,y]\rangle & \to [x\ominus [\![n]\!],y], \\ \langle \textbf{right}\ n,[x,y]\rangle & \to [x\oplus [\![n]\!],y], \\ \langle \textbf{down}\ n,[x,y]\rangle & \to [x,y\ominus [\![n]\!]], \\ \langle \textbf{up}\ n,[x,y]\rangle & \to [x,y\oplus [\![n]\!]]. \end{array} \tag{12}$$

II_{ns} Using rewriting (for $[\![n]\!]\geq [\![0]\!]$):

$$\frac{\langle d\ (n-1),p\rangle \quad \to p' \quad \langle d,p'\rangle \to p''}{\langle d\ n,p\rangle \to p''}$$

$$\langle d\ 0,p\rangle \to p \tag{13}$$

III$_{ns}$ An inductive definition for the move of n step needs the following transcript of the form of the transitions. It enables us to define natural semantics of n-steps moves inductively. Therefore, we introduce a new auxiliary rule:

$$\frac{\langle d,n,p\rangle \to p'}{\langle d\ n,p\rangle \to p'}$$

for transcription of the configurations. Then we can define the semantics inductively as follows:

$$\langle d,0,p\rangle \to p,$$

$$\frac{\langle d,n,p\rangle \to p' \quad \langle d,1,p'\rangle \to p''}{\langle d,n+1,p\rangle \to p''} \tag{14}$$

6. Equivalence between denotational and natural semantics of *Robot*

The semantics of a program must be unambiguous, independent of the semantic method used. Therefore the aim of this paper is to prove the equivalence of denotational and natural semantics of the language *Robot* defined in Sections 4 and 5. We formulate the following theorem:

Theorem: Let $[\![\ ]\!]_{ds}$ be the denotational semantic function for *Robot* and let $[\![\ ]\!]_{ns}$ be the natural semantic function for this language. Then, for every statement S of *Robot*, we have

$$[\![S]\!]_{ds} = [\![S]\!]_{ns}.$$

Proof. The functions $[\![\ ]\!]_{ds}$ and $[\![\ ]\!]_{ns}$ are the elements of the ordered set

$$(\mathbf{Point} \to \mathbf{Point}, \sqsubseteq),$$

where $\sqsubseteq$ is the ordering relation (see e.g. [8]). To show that two elements of this ordered set are equal, we need to show:

$$[\![\ ]\!]_{ds} \sqsubseteq [\![\ ]\!]_{ns}, \text{ and } [\![\ ]\!]_{ns} \sqsubseteq [\![\ ]\!]_{ds}.$$

That means, for each statement S, we need to prove the next two implications:

1. if $[\![S]\!]_{ds}p = p'$, then it exists a transition $\langle S,p\rangle \to p'$ in natural semantics;
2. if $\langle S,p\rangle \to p'$ is valid in natural semantics, then $[\![S]\!]_{ds}p = p'$.

We will prove both implications for each statement of the language *Robot*. For n-step statements ($d\ n$), the proof will follow according to the scheme in Figure 1. When we prove the equivalence of *II* and *I* and *II* and *III*, we do not need to prove the equivalence of *I* and *III*, because the properties of equivalence imply that it is transitive.

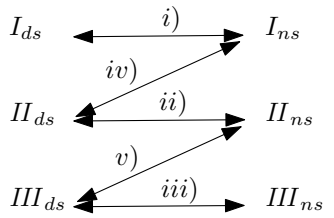

Fig. 1. A proof scheme for n-step statements

1. We prove the first implication. Let S be a statement executed in a state $p = [x, y]$. To prove this implication, we need to verify that for the denotational semantics of a statement S executed in a source state p there exists a transaction in natural semantics for S executed in p, providing the same final state. We prove it for each statements and the combinations I_{ds}–III_{ds} of semantic definitions according to Figure 1.

a) The proof for special statements is trivial:

$$\begin{array}{ll} \text{if } [\![\textbf{reset}]\!]_{ds}p = p^* & \text{then there is a transition } \langle \textbf{reset}, p\rangle \to p^*, \\ \text{if } [\![\textbf{skip}]\!]_{ds}p = p & \text{then there is a transition } \langle \textbf{skip}, p\rangle \to p. \end{array}$$

b) One-step moves. From (5) and (9), the following implication is valid:

$$\text{if } [\![d]\!]_{ds}p = p', \text{ then there is a transition } \langle d, p\rangle \to p',$$

where p' is the same result state depending on the actual direction.

c) For a sequence of statements, we use structural induction. From (3)

$$[\![S_1; S_2]\!]p = ([\![S_2]\!] \circ [\![S_1]\!])p = p''.$$

Both S_1 and S_2 are substatements (components) of the sequence $S_1; S_2$, therefore, we form the induction assumptions for them:

$$\begin{array}{ll} \text{if } [\![S_1]\!]p = p', & \text{then there is a transition } \langle S_1, p\rangle \to p', \\ \text{if } [\![S_2]\!]p' = p'', & \text{then there is a transition } \langle S_2, p'\rangle \to p''. \end{array}$$

The induction assumptions also imply the assumptions of the rule (11), therefore the conclusion

$$\langle S_1; S_2, p\rangle \to p''$$

is also valid.

d) For the n-steps moves, we prove the implication for our three semantic definitions and their combinations:

i) $I_{ds} \Rightarrow I_{ns}$ (direct approach):

If $[\![d\ n]\!]_{ds}p = p'$, then from (12) there exists a transition $\langle d\ n, p\rangle \to p'$ for each direction d.

ii) $II_{ds} \Rightarrow II_{ns}$ (rewriting approach):
The denotational semantics is defined from (7) for $[\![n]\!] \geq \mathbf{0}$ as

$$[\![d\ n]\!]_{ds} = [\![d\ (n-1)]\!]_{ds}([\![d]\!]_{ds}p).$$

We prove the implication by structural induction:
The statements $d\ (n-1)$ and d are the substatements of the statement $d\ n$, so we can formulate the following induction assumptions for them:

$$\begin{array}{ll} \text{if } [\![d]\!]_{ds}p = p', & \text{then there exists a transition } \langle d, p\rangle \to p', \text{ and} \\ \text{if } [\![d\ (n-1)]\!]_{ds}p' = p'', & \text{then there exists } \langle d\ (n-1), p'\rangle \to p''. \end{array}$$

The induction assumptions imply the assumptions of the natural semantics rule (13), which implies the validity of the conclusion:

$$\langle d\ n, p\rangle \to p'',$$

for each corresponding direction d.
If $[\![n]\!] = [\![0]\!]$, then from (10) there exists a transition $\langle d\ 0, p\rangle \to p$.

iii) $III_{ds} \Rightarrow III_{ns}$ (inductive approach):
The denotational semantics of $d\ (n+1)$ is defined by (8) as

$$[\![d]\!]_{ds}^{[\![0]\!]}p = p,$$

$$[\![d]\!]_{ds}^{[\![n+1]\!]} = [\![d]\!]_{ds}([\![d]\!]_{ds}^{[\![n]\!]}p).$$

We prove the implication by mathematical induction. Let $[\![n]\!] = [\![0]\!]$.
From (14), there exists the transition

$$\langle d, 0, p\rangle \to p.$$

Now let $[\![k]\!] < [\![n]\!]$ be an arbitrary natural number. We formulate the induction assumption for $[\![k]\!]$:

$$\text{if } [\![d]\!]_{ds}^{[\![k]\!]}p = [\![d]\!]_{ds}([\![d]\!]_{ds}^{[\![k-1]\!]}p), \text{ then there exists a transition } \langle d, k, p\rangle \to p'.$$

From (3):

$$\text{if } [\![d]\!]_{ds}p' = p'', \text{ then there exists a transition } \langle d, p'\rangle \to p''.$$

We prove the implication for $[\![k+1]\!]$. We have two assumptions of the rule (14) that imply validity of the conclusion:

$$\langle d, k+1, p\rangle \to p''.$$

iv) $II_{ds} \Rightarrow I_{ns}$. From (13), for $[\![d\ 0]\!]_{ds}p = [\![\mathbf{skip}]\!]_{ds}p = p$, there exists a transition

$$\langle d\ 0, p\rangle \to p.$$

We perform the proof by mathematical induction on $[\![n]\!]$.
Let $d = \mathbf{left}$ and $[\![k]\!] < [\![n]\!]$. We formulate the induction assumptions:

if $[\![\mathbf{left}]\!]_{ds}[x,y] = [x \ominus [\![1]\!], y]$,	then there exists a transition $\langle \mathbf{left}, [x,y]\rangle \to [x \ominus [\![1]\!], y]$,
if $[\![\mathbf{left}\ k]\!]_{ds}[x \ominus [\![k \ominus 1 + 1]\!], y] = [x \ominus [\![k]\!], y]$,	then there exists a transition $\langle \mathbf{left}\ k, [x,y]\rangle \to [x \ominus [\![k]\!], y]$.

For $[\![k+1]\!]$: if $[\![\mathbf{left}(k+1)]\!]_{ds}[x,y] = [\![\mathbf{left}\ k]\!]_{ds}([\![\mathbf{left}]\!]_{ds}[x,y])$, then there exists a transition $\langle \mathbf{left}\ ([\![k+1]\!]), [x,y]\rangle \to [x \ominus [\![k+1]\!], y]$.

v) $III_{ds} \Rightarrow II_{ns}$. The denotational semantics of the statement $d\ n$ is defined by

$$[\![d]\!]_{ds}^{[\![0]\!]}p = p,$$

$$[\![d]\!]_{ds}^{[\![n+1]\!]}p = [\![d]\!]_{ds}([\![d]\!]_{ds}^{[\![n]\!]}).$$

We use mathematical induction. For $[\![n]\!] = [\![0]\!]$:

$$\text{if } [\![d]\!]_{ds}^{[\![0]\!]}p = p, \quad \text{then there exists a transition } \langle d\ 0, p\rangle \to p.$$

Let $[\![k]\!] < [\![n]\!]$. We formulate the induction assumptions:

$$\text{if } [\![d]\!]_{ds}^{k}p = p', \quad \text{then there exists a transition } \langle d\ k, p\rangle \to p',$$

$$\text{if } [\![d]\!]_{ds}p' = p'', \quad \text{then there exists a transition } \langle d, p'\rangle \to p''.$$

Both induction assumptions are the assumption of the rule (13). Therefore the conclusion is valid for $[\![k+1]\!]$:

$$\langle d(k+1), p\rangle \to p''.$$

2. Now, we prove the opposite implication:

$$\text{if } \langle S, p\rangle \to p' \text{ then } [\![S]\!]_{ds}p = p'.$$

Let S be a statement executed in the state $p = [x,y]$.

a) The proof of special statements is simple:

if $\langle \mathbf{reset}, p\rangle \to p^*$,	then from (3)	$[\![\mathbf{reset}]\!]_{ds}p = p^*$,
if $\langle \mathbf{skip}, p\rangle \to p$,	then from (3)	$[\![\mathbf{skip}]\!]_{ds}p = p$.

b) Consider one-step moves. For each corresponding direction d it follows

$$\text{if } \langle d, p\rangle \to p', \quad \text{then from (3) } [\![d]\!]_{ds} p = p,$$

c) For sequence of statements $S_1; S_2$, we have the rule:

$$\frac{\langle S_1, p\rangle \to p' \quad \langle S_2, p'\rangle \to p''}{\langle S_1; S_2, p\rangle \to p''}$$

We use structural induction. The statements S_1 and S_2 are the substatements of the sequence, therefore we can formulate induction assumptions:

$$\begin{array}{ll} \text{if } \langle S_1, p\rangle \to p', & \text{then } [\![S_1]\!]_{ds} p = p', \quad \text{and} \\ \text{if } \langle S_2, p'\rangle \to p'', & \text{then } [\![S_2]\!]_{ds} p' = p''. \end{array}$$

Then

$$[\![S_1; S_2]\!]_{ds} p = [\![S_2]\!]_{ds}([\![S_1]\!]_{ds} p) = [\![S_2]\!]_{ds} p' = p''.$$

d) For n-steps moves, we prove the combinations of our three approaches on how to define the semantics.

i) $I_{ns} \Rightarrow I_{ds}$ (direct approach)

In direct approach of natural semantics for each direction and $[\![n]\!]$ natural number exists a transition $\langle d\ n, p\rangle \to p'$. Then, in denotational semantics for each corresponding direction from (6):

$$[\![d\ n]\!]_{ds} p = p'.$$

ii) $II_{ns} \Rightarrow II_{ds}$ (rewriting)

We assume in natural semantics

$$\frac{\langle d\ (n-1), p\rangle \quad \to p' \quad \langle d, p'\rangle \to p''}{\langle d\ n, p\rangle \to p''}$$

$$\langle d\ 0, p\rangle \to p.$$

For $[\![n]\!] = [\![0]\!]$:

$$\text{if } \langle d\ 0, p\rangle \to p, \quad \text{then } [\![d\ 0]\!]_{ds} p = [\![\textbf{skip}\ p]\!] = p.$$

Let $[\![k]\!] < [\![n]\!]$. We formulate induction assumptions:

$$\begin{array}{ll} \text{if } \langle d\ k, p\rangle \to p', & \text{then } [\![d\ k]\!]_{ds} p = p', \\ \text{if } \langle d, p'\rangle \to p'', & \text{then } [\![d]\!]_{ds} p' = p''. \end{array}$$

From induction assumptions, we get:

$$[\![d\ (k+1)]\!]_{ds}p = [\![d\ k; d]\!]_{ds}p = [\![d]\!]_{ds}([\![d\ k]\!]_{ds}p) = [\![d]\!]_{ds}p' = p''.$$

iii) $III_{ns} \Rightarrow III_{ds}$ (inductive).

From natural semantics:

$$\langle d, 0, p\rangle \to p,$$

$$\frac{\langle d, n, p\rangle \to p' \quad \langle d, 1, p'\rangle \to p''}{\langle d, n+1, p\rangle \to p''}$$

Let $[\![n]\!] = [\![0]\!]$. Then

$$\text{if } \langle d, 0, p\rangle \to p, \quad \text{then } [\![d]\!]_{ds}^{[\![0]\!]}p = p.$$

Let $[\![k]\!] < [\![n]\!]$. We formulate induction assumptions:

$$\text{if } \langle d, k, p\rangle \to p', \quad \text{then } [\![d]\!]_{ds}^{[\![k]\!]}p = p', \text{and}$$

$$\text{if } \langle d, 1, p'\rangle \to p'', \quad \text{then } [\![d]\!]_{ds}p' = p''.$$

From the induction assumptions:

$$[\![d]\!]_{ds}^{k+1}p = [\![d]\!]_{ds}([\![d]\!]_{ds}^{[\![k]\!]}p) = [\![d]\!]p' = p''.$$

iv) $I_{ns} \Rightarrow II_{ds}$. Natural semantics is defined by $\langle d\ n, p\rangle \to p''$.

Let $[\![n]\!] = [\![0]\!]$. If $\langle d\ 0, p\rangle \to p$ then $[\![d\ 0]\!]_{ds}p = p$.

Let $[\![k]\!] < [\![n]\!]$. We formulate the induction assumptions:

$$\begin{array}{ll} \text{if } \langle d, p\rangle \to p', & \text{then } [\![d1]\!]_{ds}p = p', \\ \text{if } \langle d\ k, p'\rangle \to p'', & \text{then } [\![d\ k]\!]_{ds}p' = p''. \end{array}$$

Then it follows:

$$[\![d\ (k+1)]\!]_{ds}p = [\![d\ k]\!]_{ds}([\![d]\!]_{ds}p) = [\![d\ k]\!]_{ds}p' = p''.$$

v) $II_{ns} \Rightarrow III_{ds}$. Natural semantics is defined by the following rules:

$$\frac{\langle d\ (n-1), p\rangle \quad \to p' \quad \langle d, p'\rangle \to p''}{\langle d\ n, p\rangle \to p''}$$

$$\langle d\ 0, p\rangle \to p.$$

For $[\![n]\!] = [\![0]\!]$ it holds trivially that $[\![d]\!]_{ds}^{0}p = p$.

Let $[\![k]\!] < [\![n]\!]$. Then we assume

$$\text{if } \langle d\ k, p\rangle \to p', \quad \text{then } [\![d]\!]_{ds}^{[\![k]\!]} p = p', \text{ and}$$

$$\text{if } \langle d, p'\rangle \to p'', \quad \text{then } [\![d]\!]_{ds}^{[\![1]\!]} p' = p''.$$

Then for $[\![k+1]\!]$ it follows

$$[\![d]\!]_{ds}^{[\![k+1]\!]} p = [\![d]\!]_{ds}^{\mathbf{1}}([\![d]\!]_{ds}^{[\![k]\!]})p = [\![d]\!]_{ds}^{1} p' = p''.$$

This completes the proof of the semantic equivalence.

7. Conclusions

In this paper, we have formulated a proof of the equivalence of the semantic methods for the simple language *Robot*. We have constructed a proof for the equivalence of operational and denotational semantics. This version of the language is based on simplifications, where we assume a (potentially) unlimited infinite network for the movement of the robot. In this network expressed in the coordinate system, there are currently no defined obstacles that the robot would have to overcome. Thanks to the mentioned simplifications, the semantic functions are thus defined as total. Moreover, in this model of the *Robot* language, we considered only integer coordinates for the robot, which allows (for pedagogical reasons) to work with the given abstraction more easily. In the future, it is also possible to consider the addition of the third dimension, which will create a three-dimensional coordinate space in which, for example, the movement of a drone can be modeled using the relevant DSL language. To bring our domain-specific *Robot* language closer to a real languages, in further research, we want to focus on extensions, such as incorporating obstacles (those can be solved by the conditional and loop statements), and expanding the net from two-dimensional to three- dimensional structures for more nuanced and sophisticated programming capabilities. These extensions not only diversify the linguistic landscape but also provide a foundation for addressing more complex computational challenges in future language design and implementation. Certainly, it is a challenge to respond to such an expanded language by expanding and verifying the relevant semantic methods.

Acknowledgment

This work was supported in the frame of the initiative project "Semantics-Based Rapid Prototyping of Domain-Specific Languages" under the bilateral program "Aktion Österreich-Slowakei, Wissenschafts- und Erziehungskooperation" granted by Slovak Academic Information Agency and by the project 030TUKE-4/2023 "Application of new principles in the education of

IT specialists in the field of formal languages and compilers", granted by Cultural and Education Grant Agency of the Slovak Ministry of Education.

References

[1] Mernik, M., Heering, J., & Sloane, A. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys*, *37*, 316-344. DOI: 10.1145/1118890.1118892.

[2] Voelter, M., et al. (2013). *DSL Engineering – Designing, Implementing and Using Domain-Specific Languages*. dslbook.org.

[3] Fowler, M. (2010). *Domain-Specific Languages*. Molecular Physics.

[4] Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., & Völkel, S. (2014). Design guidelines for domain specific languages [Available on https://arxiv.org/ftp/arxiv/papers/1409/1409.2378.pdf]. *CoRR*, *abs/1409.2378*. http://arxiv.org/abs/1409.2378.

[5] Nordmann, A, Hochgeschwender, N., & Wrede, S. (2014). A survey on domain-specific languages in robotics. In: D. Brugali, J.F. Broenink, T. Kroeger, & B.A. MacDonald (Eds.), *Simulation, Modeling, and Programming for Autonomous Robots*. Springer International Publishing, 195-206.

[6] Horpácsi, D., & Kőszegi, J. (2015). Formal Semantics [Accessed: 5.01.2022].

[7] Steingartner, W., & Novitzká, V. (2021). Natural semantics for domain-specific language. *New Trends in Database and Information Systems*, 181-192.

[8] Nielson, R.H., & Nielson, F. (2007). *Semantics with Applications: An Appetizer*. Springer Science & Business Media.

[9] Stoy, J.E. (1981). *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. The MIT Press. https://mitpressbookstore.com/denotational-semantics.

[10] Kahn, G. (1987). Natural semantics. *STACS 87, 4th Annual Symposium on Theoretical Aspects of Computer Science*. Passau, Germany, February 19-21, 1987, Proceedings, 22-39.