A new approach to denotational semantics by categories

William Steingartner

william.steingartner@tuke.sk

Technical University of Košice
Faculty of Electrical Engineering and Informatics
Department of Computers and Informatics

Formal semantics

- provides unambiguous meaning of programs written in programming language,
- helps designers to prepare good and useful programming languages,
- serves for designers to write correct compilers,
- encourages users/programmers how to use language constructions properly.

Semantic methods

- denotational semantics,
- operational semantics,
- natural semantics,
- axiomatic semantics,
- action semantics,
- game semantics,
- ...

Categories

- mathematical structures consisting of objects and morphisms between them,
- objects can be various mathematical structures, data structures, types,
- categories have become useful for modeling computations, processes, programs, program systems,
- are basic structures for coalgebraic behavioral models.

Categories in teaching

- quite simple mathematical structures,
- graphical representations useful for illustration of examples,
- understandable for our students.

Basic Concepts: Category theory

Category

- $Ob(\mathscr{C})$, objects of category $\mathscr{C}$, e.g. $A, B, \ldots$,
- $Morph(\mathscr{C})$, morphisms of category $\mathscr{C}$, e.g. $f : A \to B$,
- identity morphism for each object of $\mathscr{C}$, $\mathrm{id}_A : A \to A$,
- composition of morphisms: for $f : A \to B$ and $g : B \to C$ is $g \circ f : A \to C$.

Functor

- is a morphism between categories, $F : \mathscr{C} \to \mathscr{D}$,
- maps objects of $\mathscr{C}$ to objects of $\mathscr{D}$,
- maps morphism $C_1 \to C_2$ in $\mathscr{C}$ to morphism $FC_1 \to FC_2$ in $\mathscr{D}$,
  1. $F(f : A \to B) = F(f) : F(A) \to F(B)$,
  2. $F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$,
  3. $F(g \circ f) = F(g) \circ F(f)$.

Categorical semantics

- denotational semantics uses category of types where objects are types and morphisms are functions,
- algebraic semantics uses institutions as complex structures based on categories of signatures,
- game semantics uses category of arenas.

Why categorical operational semantics

- provides illustrative view of dynamics of states,
- provides simply understandable mathematical model of programs,
- appropriate for designers writing compilers,
- serves for creating skills to work with formal methods.

Basic ideas of our approach

Construction of category of states

- we consider simple imperative language,
- our language has only two implicit types,
- we do not consider exception, jumps and recursion,
- we construct category of states for every program as categorical model (representation),
- so simplified model is understandable without losing exactness.

Language *Jane*

- consists of traditional syntactic constructions of imperative languages,
- 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,
$D \in \mathbf{Decl}$	— for sequences of variable declarations.

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

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

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

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

The variables used in programs have to be declared. We consider $D \in \mathbf{Decl}$ as a sequence of declarations:

$$D ::= \texttt{var}\ x; D \mid \varepsilon.$$

As the statements $S \in \mathbf{Statm}$ we consider five Dijkstra's statements together with block statement and input statement:

$$S ::= x := e | \texttt{skip} | S; S | \texttt{if}\ b\ \texttt{then}\ S\ \texttt{else}\ S | \texttt{while}\ b\ \texttt{do}\ S | \texttt{begin}\ D; S\ \texttt{end} | \texttt{input}\ x.$$

Specification of states

State

- can be considered as some abstraction of computer memory,
- change of state means change of value in memory,
- because of block structure of *Jane*, we have to consider also a level of block nesting,
- every variable occurring in a program has to be allocated.

The signature Σ_{State} for states

$$\Sigma_{State} = \begin{array}{ll} \underline{types:} & State, Var, Value \\ \underline{opns:} & init :\rightarrow State \\ & alloc : Var, State \rightarrow State \\ & get : Var, State \rightarrow Value \\ & del : State \rightarrow State \end{array}$$

Denotational semantics

Categorical model

- we construct operational model of *Jane* as the category $\mathscr{C}_{State}$ of states,
- we assign to states their representation,
- because of block structure of *Jane*, we have to consider also a level of block nesting ($l \in \mathbf{Level}, \mathbf{Level} \subseteq \mathbf{N}$),
- representation of type $State$ has to express variable, its value with respect to the actual nesting level,

State representation

Sequence

$$s : Var \times Level \rightarrow Value$$

Each state s can be expressed as a sequence of ordered pairs $((x, l), v)$:

$$s = \langle ((x, 1), v_1), \ldots, ((z, l), v_n) \rangle$$

Table

variable	level	value
x	1	v_1
$\vdots$		
z	l	v_n

Representation of operations

The operation $[\![init]\!]$

$$[\![init]\!] = s_0 = \langle((\bot, 1), \bot)\rangle$$

creates the initial state of a program with no declared variable.

variable	level	value
$\bot$	1	$\bot$

The operation $[\![alloc]\!]$

$$[\![alloc]\!](x, s) = s \diamond ((x, l), \bot),$$

sets actual nesting level to declared variable. Because of undefined value of declared variable, the operation $[\![alloc]\!]$ does not change the state.

variable	level	value
$\vdots$	$\vdots$	$\vdots$
x	l	$\bot$

Representation of operations

The operation $[\![get]\!]$ returns a value of a variable declared on the highest nesting level,

$$[\![get]\!](x, \langle \ldots, ((x, l_i), v_i), \ldots, ((x, l_k), v_k), \ldots \rangle) = v_k,$$

where $l_i < l_k$, i<k for all i, from the definition of state.

The operation $[\![del]\!]$ deallocates (forgets) all variables declared on the highest nesting level l_j:

$$[\![del]\!](s \diamond \langle ((x_i, l_j), v_k), \ldots, ((x_n, l_j), v_m) \rangle) = s.$$

variable	level	value
$\vdots$	$\vdots$	$\vdots$
x	l_i	v
x_i	l_j	v_k
$\vdots$	$\vdots$	$\vdots$
x_n	l_j	v_m

Declarations

Declarations

A declaration

$$\texttt{var}\ x$$

is represented as an endomorphism:

$$[\![\texttt{var}\ x]\!]_D : s \to s$$

for a given state s and defined by

$$[\![\texttt{var}\ x]\!]s = [\![alloc]\!](x, s).$$

A sequence of declarations

$$[\![\texttt{var}\ x; D]\!]s = [\![D]\!] \circ [\![alloc(x, s)]\!].$$

A declaration creates a new entry for declared variable with the actual level of nesting and an undefined value

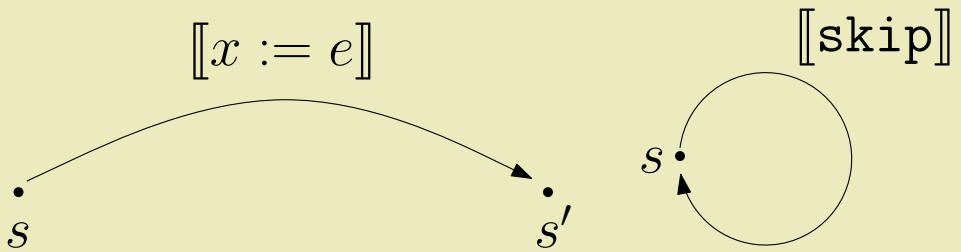
$$((x, l), \bot).$$

Statements

$$[\![S]\!] : s \to s$$

$$[\![x := e]\!]s = \begin{cases} s\left[\left((x,l),v\right) \mapsto \left((x,l),[\![e]\!]s\right)\right], & \text{for } \left((x,l),v\right) \in s, \\ s_\perp, & \text{otherwise.} \end{cases}$$

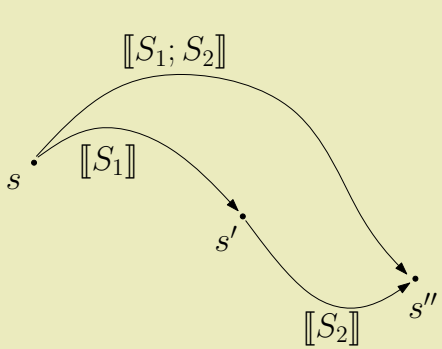
$$[\![\texttt{skip}]\!] = \mathsf{id}_s, \qquad [\![\texttt{skip}]\!]s = s$$

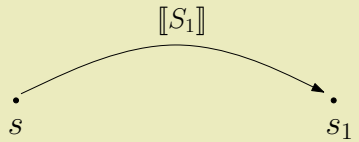


Statements

$$[\![S_1, S_2]\!] = [\![S_2]\!] \circ [\![S_1]\!], \qquad [\![S_1, S_2]\!]s = [\![S_2]\!]\,([\![S_1]\!])\,s$$

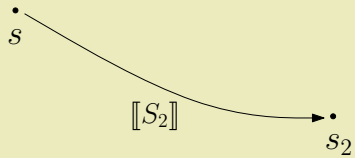
$$[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!]s = \begin{cases} [\![S_1]\!]s, & \text{if} [\![b]\!]s = \mathbf{true}, \\ [\![S_2]\!]s, & \text{otherwise.} \end{cases}$$





$[\![b]\!]s = \mathbf{true}$

(a)



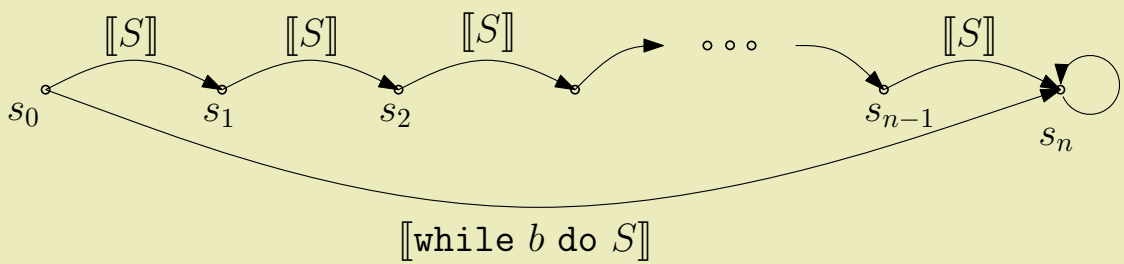
$[\![b]\!]s = \mathbf{false}$

(b)

Statements

$$[\![\texttt{while}\ b\ \texttt{do}\ S]\!]s = [\![\texttt{if}\ b\ \texttt{then}\ (S; \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else}\ \texttt{skip}]\!]$$

$$[\![\texttt{input}\ x]\!]s = \begin{cases} s' = s\,[v/x]\,, & \text{for } ((x, max\ l)\,, v') \in s, \\ \bot, & \text{otherwise.} \end{cases}$$



Block statement

$$\texttt{begin}\ D, S\ \texttt{end}$$

The following is a summary of the four steps used to execute of unnamed blocks:

- nesting level l is incremented. We represent this step by fictive entry in state table

  $$((\texttt{begin}, l+1), \perp)$$

  i.e. endomorphism $\mathbf{State} \to \mathbf{State}$,
- local declarations are elaborated on nesting level $l+1$,
- the body S of block is executed,
- locally declared variables are forgotten at the end of block. We model this situation using operation $[\![del]\!]$.

The semantics:

$$[\![\texttt{begin}\ D, S\ \texttt{end}]\!]s = [\![del]\!] \circ [\![S]\!] \circ [\![D]\!](s \diamond \langle((\texttt{begin}, l+1), \perp)\rangle)$$

Constructing the category

Now we can define the category $\mathscr{C}_{State}$ of states as follows:

- category objects are states as sequences of tuples for variables together with special state $s_\perp$,
- category morphisms are functions $[\![S]\!] : s \to s'$.

The category $\mathscr{C}_{State}$ has the following properties:

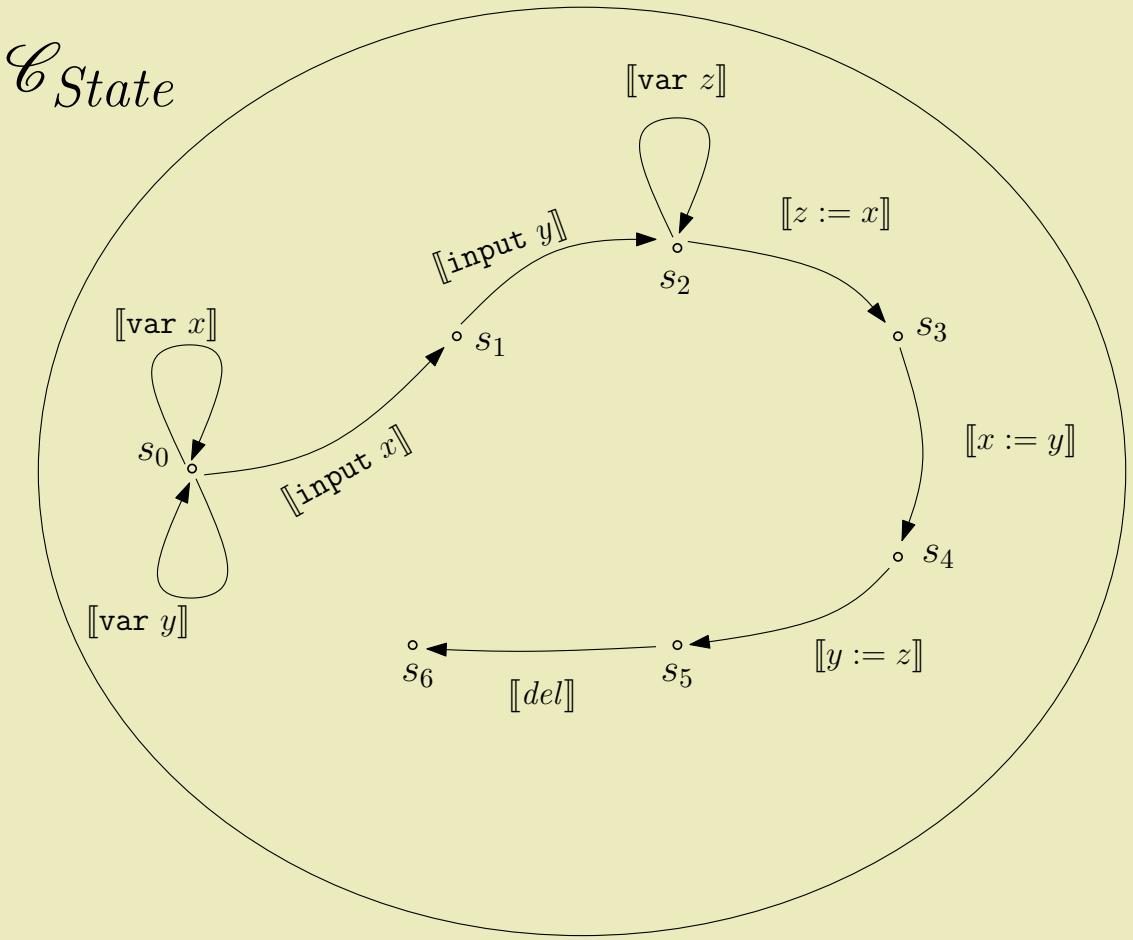
- the special object $s_\perp = \langle((\perp, \perp), \perp)\rangle$, undefined state, is a terminal object of our category, from any object there is a unique morphism to this state,
- the initial state $s_0 = \langle((\perp, 1), \perp)\rangle$ is the initial object of our category,
- the category $\mathscr{C}_{State}$ has no products, because a program written in $Jane$ cannot be simultaneously in more than one state.

We can state that $\mathscr{C}_{State}$ is a category without products and with initial and terminal objects.

Example

```
var x; var y;
input x;
input y;
if x <= y then
   begin
     z := x;
     x := y;
     y := z;
   end
 else
   skip;
```

Categorical representation of program



States during program execution

s_0		
x	1	$\bot$
y	1	$\bot$

s_1		
x	1	**3**
y	1	$\bot$

s_2		
x	1	**3**
y	1	**5**
z	2	$\bot$

s_3		
x	1	**3**
y	1	**5**
z	2	**3**

s_4		
x	1	**5**
y	1	**5**
z	2	**3**

s_5		
x	1	**5**
y	1	**3**
z	2	**3**

s_6		
x	1	**5**
y	1	**3**
z	~~2~~	$\bot$

Conclusion

- we presented a new approach to operational semantics by categories,
- we constructed category of states $\mathscr{C}_{State}$ where states of memory are objects and state changes (computations) are morphisms,
- the semantics of program is defined as composition of morphisms from initial state into final state and is represented in category as a path of all morphisms that represent each program step,

Future

- In our future research we want to extend our approach by types possibly extending states by new columns and we want to define the semantics of recursive procedures by appropriate endofunctors that ensure saving of particular states within the expansion of recursive calls.