Semantics (meaning) of Arithmetic Expressions

Denotational Semantics

Every expression (term) denotes either a natural number or a truth value (True or False)

For example the expression

succ (succ (succ zero))

denotes the number 3

We use "semantic brackets" $[\![-]\!]$ for the interpretation:

$$[\![\text{succ}\,(\text{succ}\,(\text{succ zero}))]\!] = 3$$

The interpretation is defined by structural recursion on the syntax:

$$[\![\text{zero}]\!] = 0$$

$$[\![\text{false}]\!] = \text{False}$$

$$[\![\text{true}]\!] = \text{True}$$

$$[\![\text{succ } e]\!] = [\![e]\!] + 1$$

if this is a number

$$[\![\text{pred } e]\!] = \begin{cases} 0 & \text{if } [\![e]\!] = 0 \\ n-1 & \text{if } [\![e]\!] = n \end{cases}$$

number

$$[\![\text{iszero } e]\!] = \begin{cases} \text{True} & \text{if } [\![e]\!] = 0 \\ \text{False} & \text{otherwise} \end{cases}$$

($[\![e]\!]$ must be a number)

$$[\![\text{if } e_1 \text{ then } e_2 \text{ else } e_3]\!] =$$

$$\begin{cases} [\![e_2]\!] & \text{if } [\![e_1]\!] = \text{True} \\ [\![e_3]\!] & \text{if } [\![e_1]\!] = \text{False} \end{cases}$$

($[\![e_2]\!]$ and $[\![e_3]\!]$ must be of the same type: either both numbers or both truth values)

Some valid expressions don't have a denotation

if (pred zero) then true
else (succ false)

This seems a meaningless term

We can use the symbol $\bot$ (read "bottom") to denote meaningless terms

$[\![\text{succ false}]\!] = \bot$

$[\![\text{if (pred zero)} \cdots]\!] = \bot$

(Alternative approach: give meaningless terms an arbitrary denotation)

Operational Semantics

The meaning of terms is their computation

We specify how terms/expressions are computed

$\rightsquigarrow$ one-step reduction relation
it tells us how a term can be simplified

$\rightsquigarrow^*$ many-steps reduction

Example: if (iszero (succ zero))
then (succ (succ zero))
else (pred (succ zero))
$\rightsquigarrow^*$ zero

Reduction Rules

(how we do the reduction steps)

if true then e_2 else $e_3 \rightsquigarrow e_2$

if false then e_2 else $e_3 \rightsquigarrow e_3$

pred (succ e) $\rightsquigarrow$ e

pred zero $\rightsquigarrow$ zero

iszero zero $\rightsquigarrow$ true

iszero (succ e) $\rightsquigarrow$ false

Structural rules:
reductions can be performed in context:
we can reduce a subterm of a larger term:

$$\frac{e \rightsquigarrow e'}{\text{succ } e \rightsquigarrow \text{succ } e'}$$

$$\frac{e_1 \rightsquigarrow e_1'}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \rightsquigarrow \text{if } e_1' \text{ then } e_2 \text{ else } e_3}$$

and similar rules operating on e_2 and e_3

... other structural rules

This language is not really useful

To make it more interesting we must add variables and recursion

Summary of Module Content

You will learn to

- Define the syntax of your own programming language
- Give mathematical meaning to programs (denotational semantics)
- Specify how programs run (operational semantics)
- Use formal logic to reason about programs