Denotational Semantics of PL/0

Abstract Syntax:

Syntax Domains:	**Abstract Syntax Rules:**
P: Program	**P ::= B .**
B: Block	**B ::= CD VD PD S**
CD: Const Decl	**CD::= CD Id = N** | ε
VD: Var Decl	**VD::= VD Id** | ε
PD: Proc Decl	**PD::= PD procedure Id ; B ;** | ε
S: Statement	**S ::= S_1 ; S_2** | **Id := E** | **call Id** | **if C then S** | **while C do S** | ε
C: Condition	**C ::= odd E** | **E_1 R E_2**
E: Expression	**E ::= +E** | **–E** | **E_1 + E_2** | **E_1 – E_2** | **E_1 * E_2** | **E_1 / E_2** | **Id** | **N**
R: Relation	**R ::= =** | **<>** | **<** | **>** | **<=** | **>=**
N: Integer	
Id: Identifier	

Semantic Domains:

I. Boolean Values

Domain b ∈ Boolean = { *true, false* }°

II. Integers

Domain n ∈ N = {. . . , –2, –1, 0, 1, 2, . . .}°

Operations:

$AO[\![+]\!]\ n_1\ n_2 = (\ n_1\ +\ n_2\ )$ — *AO* returns an integer value

$AO[\![-]\!]\ n_1\ n_2 = (\ n_1\ -\ n_2\ )$

$AO[\![*]\!]\ n_1\ n_2 = (\ n_1\ \times\ n_2\ )$

$AO[\![/]\!]\ n_1\ n_2 =$ *if* $(n_2 = 0)$ *then* $\top$ *else* $(n_1\ /\ n_2\ )$

$RO[\![=]\!]\ n_1\ n_2 = (\ n_1\ =\ n_2\ )$ — *RO* returns a Boolean value

$RO[\![<>]\!]\ n_1\ n_2 = (\ n_1\ \neq\ n_2\ )$

$RO[\![<]\!]\ n_1\ n_2 = (\ n_1\ <\ n_2\ )$

$RO[\![<=]\!]\ n_1\ n_2 = (\ n_1\ \leq\ n_2\ )$

$RO[\![>]\!]\ n_1\ n_2 = (\ n_1\ >\ n_2\ )$

$RO[\![>=]\!]\ n_1\ n_2 = (\ n_1\ \geq\ n_2\ )$

odd : N → Boolean

III. Identifiers

Domain id ∈ Identifier

IV. Categories

Domain c ∈ Category = { *constant, variable, procedure* }

V. Values

Domain v ∈ Value = N ∪ Location ∪ Procedure

VI. Denotable–values

Domain d ∈ Denotable–value = {(Category × Value)}°

Operations:

denotable–value : Category → Value → Denotable–value

denotable–value c v = (c, v) "**constructs** denotable value tuple"

category : Denotable–value → Category

category d = (d ↓ 1) "**selects** the category component"

value : Denotable–value → Value

value d = (d ↓ 2) "**selects** the value component"

VII. Procedures

Domain p ∈ Procedure = Store → Store

Operations:

call : Procedure → Store → Store

call p store = ...

VIII. Environment

Domain env ∈ Environment = Location × (Id → Denotable–value)

Operations:

init–env : Id → Denotable–value

init–env = (0, λ Id . ⊥)

next–location : Environment → Location

next–location env = (env ↓ 1)

map : Environment → (Id → Denotable–value)

map env = (env ↓ 2)

access–env : Id → Environment → Denotable–value

access–env Id env = (*map* env) ⟦ Id ⟧

update–env : Id → Denotable–value → Environment → Environment

update–env Id d env = (*map* env) [d / Id]

IX. Locations

Domain loc ∈ Location = N

X. Store

Domain store $\in$ Store = Location $\rightarrow$ N

Operations:

init–store : Location $\rightarrow$ N

init–env = λ loc . $\perp$

access–store : Location $\rightarrow$ N

access–store loc store = store $[\![$ Id $]\!]$

update–store : Location $\rightarrow$ N $\rightarrow$ Store $\rightarrow$ Store

update–store loc n store =store [n / loc]

Semantic Functions:

M : Program $\rightarrow$ Store

B : Block $\rightarrow$ Environment $\rightarrow$ Store $\rightarrow$ Store

CD : Const Decl $\rightarrow$ Environment $\rightarrow$ Environment

VD : Var Decl $\rightarrow$ Environment $\rightarrow$ Environment

PD : Proc Decl $\rightarrow$ Environment $\rightarrow$ Environment

S : Statement $\rightarrow$ Environment $\rightarrow$ Store $\rightarrow$ Store

E : Expression $\rightarrow$ Environment $\rightarrow$ Store $\rightarrow$ Expressible–value

Semantic Equations:

M $[\![$**B .**$]\!]$ = *B* $[\![$**B**$]\!]$ *init–env init–store*

B $[\![$**CD VD PD S**$]\!]$ env store = *S* $[\![$**S**$]\!]$ (*PD* $[\![$**PD**$]\!]$ (*VD* $[\![$**VD**$]\!]$ (*CD* $[\![$**CD**$]\!]$ env))) store

CD $[\![$**CD Id = N**$]\!]$ env = *update–env* $[\![$**Id**$]\!]$ (*denotable–value constant* (*N* $[\![$**N**$]\!]$)) (*CD* $[\![$**CD**$]\!]$ env)

CD $[\![]\!]$ env = env

VD $[\![$**VD Id**$]\!]$ env =

update–env $[\![$**Id**$]\!]$ (*denotable–value variable* (*next–location* env)) (*VD* $[\![$**VD**$]\!]$ env)

VD $[\![]\!]$ env = env

PD $[\![$**PD procedure Id; B;**$]\!]$ env =

letrec env_p = *update–env* $[\![$**Id**$]\!]$ (*denotable–value procedure* (*B* $[\![$**B**$]\!]$ env_p)) (*PD* $[\![$**PD**$]\!]$ env)

in env_p

end

PD $[\![]\!]$ env = env

S ⟦**S_1; S_2**⟧ env store = S ⟦**S_2**⟧ env (S ⟦**S_1**⟧ env store)

S ⟦**Id := E**⟧ env store =
    let d = (*access–env* ⟦**Id**⟧ env)
    in if ((*category* d) ≠ *variable*) then "error"
        else (*update–store* (*value* d) (E ⟦**E**⟧ env store) store)
    end

S ⟦**call Id**⟧ env store =
    let d = (*access–env* ⟦**Id**⟧ env)
    in if ((*category* d) ≠ *procedure*) then "error"
        else *call* (*value* d) store
    end

S ⟦**if C then S**⟧ env store = if (C ⟦**C**⟧ env store) then (S ⟦**S**⟧ env store) else store

S ⟦**while C do S**⟧ env store =
    if (C ⟦**C**⟧ env store) then (S ⟦**while C do S**⟧ env (S ⟦**S**⟧ env store)) else store

C ⟦**odd E**⟧ env store = *odd* (E ⟦**E**⟧ env store)

C ⟦**E_1 rop E_2**⟧ env store = RO ⟦**rop**⟧ (E ⟦**E_1**⟧ env store) (E ⟦**E_2**⟧ env store)
    where **rop** ∈ {== , <> , < , > , <= , >=}

E ⟦**+E**⟧ env store = E ⟦**E**⟧ env store

E ⟦**–E**⟧ env store = – (E ⟦**E**⟧ env store)

E ⟦**E_1 aop E_2**⟧ env store = AO ⟦**aop**⟧ (E ⟦**E_1**⟧ env store) (E ⟦**E_2**⟧ env store)
    where **aop** ∈ { + , – , * , / }

E ⟦**Id**⟧ env store = **Variable Access**

```
let d = (access–env ⟦Id⟧ env )
in    if (category d = constant) then
            (value d)
      else if (category d = variable) then
            let e = (access–store (value d) (store store))
            in if e = ⊥ then "error" else e
            end
      else "error"
end
```

E ⟦**N**⟧ env store = N ⟦**N**⟧