Languages-beta: SIMPLE-4-Declarations

The PLanCompS Project

`Languages-beta/SIMPLE/SIMPLE-4-Declarations/SIMPLE-4-Declarations.cbs`*

Language "`SIMPLE`"

4 Declarations

Syntax *Decl* : decl ::= vars-decl
| func-decl

Semantics declare⟦ _ : decl ⟧ : ⇒ environments

4.1 Variable Declarations

Syntax *VarsDecl* : vars-decl ::= `var` declarators `;`
Declarators : declarators ::= declarator (`,` declarators)?

Rule ⟦ `var` *Declarator* `,` *Declarators* `;` *Stmts*? ⟧ : stmts =
⟦ `var` *Declarator* `;` `var` *Declarators* `;` *Stmts*? ⟧
Rule ⟦ `var` *Declarator* `,` *Declarators* `;` *Decls*? ⟧ : decls =
⟦ `var` *Declarator* `;` `var` *Declarators* `;` *Decls*? ⟧
Rule declare⟦ `var` *Declarator* `;` ⟧ =
var-declare⟦ *Declarator* ⟧

Syntax *Declarator* : declarator ::= id
| id = exp
| id ranks

*Suggestions for improvement: `plancomps@gmail.com`.
Issues: `https://github.com/plancomps/CBS-beta/issues`.

```
Semantics var-declare⟦ _ : declarator ⟧ : ⇒ environments
    Rule var-declare⟦ Id ⟧ =
            bind(id⟦ Id ⟧,
              allocate-variable(values))
    Rule var-declare⟦ Id = Exp ⟧ =
            bind(id⟦ Id ⟧,
              allocate-initialised-variable(values,
                rval⟦ Exp ⟧))
    Rule var-declare⟦ Id Ranks ⟧ =
            bind(id⟦ Id ⟧,
              allocate-nested-vectors(ranks⟦ Ranks ⟧))
```

4.2 Arrays

```
Syntax Ranks : ranks ::= [ exps ] ranks?

Rule ⟦ [ Exp , Exps ] Ranks? ⟧ : ranks =
    ⟦ [ Exp ] [ Exps ] Ranks? ⟧
```

Compare this with p28 of the K version.

```
Semantics ranks⟦ _ : ranks ⟧ : (⇒ nats)+
    Rule ranks⟦ [ Exp ] ⟧ =
            rval⟦ Exp ⟧
    Rule ranks⟦ [ Exp ] Ranks ⟧ =
            rval⟦ Exp ⟧,
            ranks⟦ Ranks ⟧

Funcon allocate-nested-vectors(_ : nats +) : ⇒ variables
  Rule allocate-nested-vectors(N : nats) ⇝ allocate-initialised-variable(vectors(variables), vector(left-to-right
  Rule allocate-nested-vectors(N : nats, N+ : nats +) ⇝ allocate-initialised-variable(vectors(variables), vector
```

4.3 Function Declarations

```
Syntax FuncDecl : func-decl ::= function id ( ids? ) block
```

```
Rule declare⟦ function Id ( Ids? ) Block ⟧ =
        bind(id⟦ Id ⟧,
          allocate-variable(functions(tuples(values*),
              values)))

Semantics initialise⟦ _ : decl ⟧ : ⇒ null-type
    Rule initialise⟦ var Declarators ; ⟧ =
            null
    Rule initialise⟦ function Id ( Ids? ) Block ⟧ =
            assign(bound(id⟦ Id ⟧),
              function  closure(scope(match(given,
                          tuple(patts⟦ Ids? ⟧)),
                      handle-return(exec⟦ Block ⟧)))))

Syntax Ids : ids  ::=  id (, ids)?

Semantics patts⟦ _ : ids? ⟧ : patterns*
    Rule patts⟦  ⟧ =
            ( )
    Rule patts⟦ Id ⟧ =
            pattern  closure(bind(id⟦ Id ⟧,
                    allocate-initialised-variable(values,
                      given)))
    Rule patts⟦ Id , Ids ⟧ =
            patts⟦ Id ⟧,
            patts⟦ Ids ⟧
```