Languages-beta: SIMPLE-2-Expressions

The PLanCompS Project

`Languages-beta/SIMPLE/SIMPLE-2-Expressions/SIMPLE-2-Expressions.cbs`[*]

Language "SIMPLE"

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

2 Expressions

```
Syntax Exp : exp  ::=  ( exp )
                     | value
                     | lexp
                     | lexp = exp
                     | ++ lexp
                     | - exp
                     | exp ( exps? )
                     | sizeOf ( exp )
                     | read ( )
                     | exp + exp
                     | exp - exp
                     | exp * exp
                     | exp / exp
                     | exp % exp
                     | exp < exp
                     | exp <= exp
                     | exp > exp
                     | exp >= exp
                     | exp == exp
                     | exp != exp
                     | ! exp
                     | exp && exp
                     | exp || exp
```

```
Rule ⟦ ( Exp ) ⟧ : exp =
    ⟦ Exp ⟧
```

```
Semantics rval⟦ _ : exp ⟧ : ⇒ values
    Rule rval⟦ V ⟧ =
            val⟦ V ⟧
    Rule rval⟦ LExp ⟧ =
            assigned(lval⟦ LExp ⟧)
    Rule rval⟦ LExp = Exp ⟧ =
            give(rval⟦ Exp ⟧,
              sequential(assign(lval⟦ LExp ⟧,
                    given),
                 given))
    Rule rval⟦ ++ LExp ⟧ =
            give(lval⟦ LExp ⟧,
              sequential(assign(given,
                    integer-add(assigned(given),
                      1)),
                 assigned(given)))
    Rule rval⟦ - Exp ⟧ =
            integer-negate(rval⟦ Exp ⟧)
    Rule rval⟦ Exp ( Exps? ) ⟧ =
            apply(rval⟦ Exp ⟧,
              tuple(rvals⟦ Exps? ⟧))
    Rule rval⟦ sizeOf ( Exp ) ⟧ =
            length(vector-elements(rval⟦ Exp ⟧))
    Rule rval⟦ read ( ) ⟧ =
            read
    Rule rval⟦ Exp1 + Exp2 ⟧ =
            integer-add(rval⟦ Exp1 ⟧,
              rval⟦ Exp2 ⟧)
    Rule rval⟦ Exp1 - Exp2 ⟧ =
            integer-subtract(rval⟦ Exp1 ⟧,
              rval⟦ Exp2 ⟧)
    Rule rval⟦ Exp1 * Exp2 ⟧ =
            integer-multiply(rval⟦ Exp1 ⟧,
              rval⟦ Exp2 ⟧)
    Rule rval⟦ Exp1 / Exp2 ⟧ =
            checked  integer-divide(rval⟦ Exp1 ⟧,
                rval⟦ Exp2 ⟧)
    Rule rval⟦ Exp1 % Exp2 ⟧ =
            checked  integer-modulo(rval⟦ Exp1 ⟧,
                rval⟦ Exp2 ⟧)
    Rule rval⟦ Exp1 < Exp2 ⟧ =
            is-less(rval⟦ Exp1 ⟧,
              rval⟦ Exp2 ⟧)
    Rule rval⟦ Exp1 <= Exp2 ⟧ =
            is-less-or-equal(rval⟦ Exp1 ⟧,
Syntax Exps : exps  ::=  exp ( , exps)?

Semantics rvals⟦ _ : exps? ⟧ : (⇒ values)*
    Rule rvals⟦  ⟧ =
            ( )
    Rule rvals⟦ Exp ⟧ =
            rval⟦ Exp ⟧
    Rule rvals⟦ Exp , Exps ⟧ =
            rval⟦ Exp ⟧,
            rvals⟦ Exps ⟧

Syntax LExp : lexp  ::=  id
                      |  lexp [ exps ]

Rule ⟦ LExp [ Exp , Exps ] ⟧ : lexp =
    ⟦ LExp [ Exp ] [ Exps ] ⟧

Semantics lval⟦ _ : lexp ⟧ : ⇒ variables
    Rule lval⟦ Id ⟧ =
            bound(id⟦ Id ⟧)
    Rule lval⟦ LExp [ Exp ] ⟧ =
            checked  index(integer-add(1,
                  rval⟦ Exp ⟧),
                vector-elements(rval⟦ LExp ⟧))
```