Unstable-Languages-beta: SIMPLE-THR-2-Expressions *

The PLanCompS Project

`SIMPLE-THR-2-Expressions.cbs` | PLAIN | PRETTY

Language "SIMPLE-THR"

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
| 'spawn' block

Rule ⟦ '(' *Exp* ')' ⟧ : exp = ⟦ *Exp* ⟧

---

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

```
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 ⟧, rval⟦ Exp2 ⟧)
    Rule   rval⟦ Exp1 '>' Exp2 ⟧ = is-greater(rval⟦ Exp1 ⟧, rval⟦ Exp2 ⟧)
    Rule   rval⟦ Exp1 '>=' Exp2 ⟧ = is-greater-or-equal(rval⟦ Exp1 ⟧, rval⟦ Exp2 ⟧)
    Rule   rval⟦ Exp1 '==' Exp2 ⟧ = is-equal(rval⟦ Exp1 ⟧, rval⟦ Exp2 ⟧)
    Rule   rval⟦ Exp1 '!=' Exp2 ⟧ = not(is-equal(rval⟦ Exp1 ⟧, rval⟦ Exp2 ⟧))
    Rule   rval⟦ '!' Exp ⟧ = not(rval⟦ Exp ⟧)
    Rule   rval⟦ Exp1 '&&' Exp2 ⟧ = if-else(rval⟦ Exp1 ⟧, rval⟦ Exp2 ⟧, false)
    Rule   rval⟦ Exp1 '||' Exp2 ⟧ = if-else(rval⟦ Exp1 ⟧, true, rval⟦ Exp2 ⟧)
```

SIMPLE uses natural numbers to identify threads; the use of allocate-index(_) below associates a natural number with the thread-id given by thread-activate. The use of postpone-after-effect(_) supports automatic release of locks when threads terminate.

```
Rule   rval⟦ 'spawn' Block ⟧ =
         allocate-index
           thread-activate thread-joinable
             thunk closure
               postpone-after-effect
                 exec⟦ Block ⟧
```

```
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* ⟧))