

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

The PPlanCompS 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 $\llbracket '(Exp)' \rrbracket : exp = \llbracket Exp \rrbracket$

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

Semantics $\text{rval}[\![_ : \text{exp}]\!] : \Rightarrow \text{values}$

Rule $\text{rval}[\![V]\!] = \text{val}[\![V]\!]$

Rule $\text{rval}[\![LExp]\!] = \text{assigned}(\text{lval}[\![LExp]\!])$

Rule $\text{rval}[\![LExp \text{ '=' } Exp]\!] =$
 $\text{give}(\text{rval}[\![Exp]\!], \text{sequential}(\text{assign}(\text{lval}[\![LExp]\!], \text{given}), \text{given}))$

Rule $\text{rval}[\![\text{'++'} LExp]\!] =$
 $\text{give}(\text{lval}[\![LExp]\!], \text{sequential}(\text{assign}(\text{given}, \text{integer-add}(\text{assigned}(\text{given}), 1)), \text{assigned}(\text{given})))$

Rule $\text{rval}[\![\text{'-'} Exp]\!] = \text{integer-negate}(\text{rval}[\![Exp]\!])$

Rule $\text{rval}[\![Exp \text{ '(' } Exps? \text{ ')' }]\!] = \text{apply}(\text{rval}[\![Exp]\!], \text{tuple}(\text{rvals}[\![Exps?]\!]))$

Rule $\text{rval}[\![\text{'sizeof'} \text{ '(' } Exp \text{ ')' }]\!] = \text{length}(\text{vector-elements}(\text{rval}[\![Exp]\!]))$

Rule $\text{rval}[\![\text{'read'} \text{ '(' } \text{' ' } \text{' ') }]\!] = \text{read}$

Rule $\text{rval}[\![Exp_1 \text{ '+' } Exp_2]\!] = \text{integer-add}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '-' } Exp_2]\!] = \text{integer-subtract}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '*' } Exp_2]\!] = \text{integer-multiply}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '/' } Exp_2]\!] = \text{checked integer-divide}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '%' } Exp_2]\!] = \text{checked integer-modulo}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '<' } Exp_2]\!] = \text{is-less}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '<=' } Exp_2]\!] = \text{is-less-or-equal}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '>' } Exp_2]\!] = \text{is-greater}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '>=' } Exp_2]\!] = \text{is-greater-or-equal}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '==' } Exp_2]\!] = \text{is-equal}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '!=' } Exp_2]\!] = \text{not}(\text{is-equal}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!]))$

Rule $\text{rval}[\![\text{'!' } Exp]\!] = \text{not}(\text{rval}[\![Exp]\!])$

Rule $\text{rval}[\![Exp_1 \text{ '&\&' } Exp_2]\!] = \text{if-else}(\text{rval}[\![Exp_1]\!], \text{rval}[\![Exp_2]\!], \text{false})$

Rule $\text{rval}[\![Exp_1 \text{ '||' } Exp_2]\!] = \text{if-else}(\text{rval}[\![Exp_1]\!], \text{true}, \text{rval}[\![Exp_2]\!])$

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 $\text{rval}[\![\text{'spawn'} Block]\!] =$
 allocate-index
 $\text{thread-activate thread-joinable}$
 thunk closure
 $\text{postpone-after-effect}$
 $\text{exec}[\![Block]\!]$

Syntax $Exps : \text{exps} ::= \text{exp} \text{ '(' } \text{' ' } \text{exps} \text{ ')' } ?$

Semantics $\text{rvals}[_ : \text{expr}^?] : (\Rightarrow \text{values})^*$

Rule $\text{rvals}[\] = ()$

Rule $\text{rvals}[\text{Exp}] = \text{rval}[\text{Exp}]$

Rule $\text{rvals}[\text{Exp } ', ' \text{Exps}] = \text{rval}[\text{Exp}], \text{rvals}[\text{Exps}]$

Syntax $L\text{Exp} : \text{lexp} ::= \text{id} \mid \text{lexp } '[\text{exprs } ']'$

Rule $\llbracket L\text{Exp } '[\text{Exp } ', ' \text{Exps } ']' \rrbracket : \text{lexp} =$

$\llbracket L\text{Exp } '[\text{Exp } ']' '[\text{Exps } ']' \rrbracket$

Semantics $\text{lval}[_ : \text{lexp}] : \Rightarrow \text{variables}$

Rule $\text{lval}[\text{Id}] = \text{bound}(\text{id}[\text{Id}])$

Rule $\text{lval}[\llbracket L\text{Exp } '[\text{Exp } ']' \rrbracket] =$

$\text{checked_index}(\text{integer-add}(1, \text{rval}[\text{Exp}]), \text{vector-elements}(\text{rval}[\llbracket L\text{Exp } \rrbracket]))$