

Languages-beta: SIMPLE-2-Expressions *

The PPlanCompS Project

SIMPLE-2-Expressions.cbs | PLAIN | PRETTY

Language "SIMPLE"

2 Expressions

Syntax $Exp : exp ::=$

- | 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 $\llbracket '(' Exp ')' \rrbracket : exp = \llbracket Exp \rrbracket$

*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 $\text{rval}[\text{'sizeOf' ' (' Exp ') '}] = \text{length}(\text{vector-elements}(\text{rval}[Exp]))$

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

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

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

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

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

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

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

Syntax $LExp : \text{lexp} ::= \text{id} \mid \text{lexp } '[' \text{exps } ']'$

Rule $[[LExp \text{ '[' } Exp \text{ ', ' } Exps \text{ '}]] : \text{lexp} =$
 $[[LExp \text{ '[' } Exp \text{ '}]] \text{ '[' } Exps \text{ '}]]$

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

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

Rule $\text{lval}[LExp \text{ '[' } Exp \text{ '}] =$
 $\text{checked index}(\text{integer-add}(1, \text{rval}[Exp]), \text{vector-elements}(\text{rval}[LExp]))$