

Languages-beta: SL-2-Expressions *

The PPlanCompS Project

SL-2-Expressions.cbs | PLAIN | PRETTY

Language "SL"

2 Expressions

Syntax $Expr : expr ::=$

- int
- $string$
- $'true'$
- $'false'$
- $expr \ '+' \ expr$
- $expr \ '/' \ expr$
- $expr \ '*' \ expr$
- $expr \ '-' \ expr$
- $expr \ '==' \ expr$
- $expr \ '<=' \ expr$
- $expr \ '<' \ expr$
- $expr \ '!=' \ expr$
- $expr \ '>=' \ expr$
- $expr \ '>' \ expr$
- $expr \ ' \&\&' \ expr$
- $expr \ ' \|\| ' \ expr$
- $'!' \ expr$
- $id \ '(' \ expr\text{-}list? \ ')'$
- id
- $id \ '=' \ expr$
- $expr \ '.' \ id$
- $expr \ '.' \ id \ '=' \ expr$
- $expr \ '.' \ id \ '(' \ expr\text{-}list? \ ')'$
- $'(' \ expr \ ')'$

Rule $\llbracket '(' \ Expr \ ')' \rrbracket : expr = \llbracket Expr \rrbracket$

Type $sl\text{-}values \rightsquigarrow$ $booleans \mid integers \mid strings \mid objects \mid null\text{-}type$

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

Semantics $\text{eval} \llbracket \text{Expr} : \text{expr} \rrbracket : \Rightarrow \text{sl-values}$

Rule $\text{eval} \llbracket \text{Int} \rrbracket = \text{int-val} \llbracket \text{Int} \rrbracket$

Rule $\text{eval} \llbracket \text{String} \rrbracket = \text{string-val} \llbracket \text{String} \rrbracket$

Rule $\text{eval} \llbracket \text{'true'} \rrbracket = \text{true}$

Rule $\text{eval} \llbracket \text{'false'} \rrbracket = \text{false}$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'+' Expr}_2 \rrbracket =$
 $\text{integer-add-else-string-append}(\text{eval} \llbracket \text{Expr}_1 \rrbracket, \text{eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'/' Expr}_2 \rrbracket =$
 $\text{checked integer-divide}(\text{int eval} \llbracket \text{Expr}_1 \rrbracket, \text{int eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'*'} \text{Expr}_2 \rrbracket =$
 $\text{integer-multiply}(\text{int eval} \llbracket \text{Expr}_1 \rrbracket, \text{int eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'-'} \text{Expr}_2 \rrbracket =$
 $\text{integer-subtract}(\text{int eval} \llbracket \text{Expr}_1 \rrbracket, \text{int eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'==' Expr}_2 \rrbracket =$
 $\text{is-equal}(\text{eval} \llbracket \text{Expr}_1 \rrbracket, \text{eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'<=' Expr}_2 \rrbracket =$
 $\text{is-less-or-equal}(\text{int eval} \llbracket \text{Expr}_1 \rrbracket, \text{int eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'<'} \text{Expr}_2 \rrbracket =$
 $\text{is-less}(\text{int eval} \llbracket \text{Expr}_1 \rrbracket, \text{int eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'!=' Expr}_2 \rrbracket =$
 $\text{not is-equal}(\text{eval} \llbracket \text{Expr}_1 \rrbracket, \text{eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'>=' Expr}_2 \rrbracket =$
 $\text{is-greater-or-equal}(\text{int eval} \llbracket \text{Expr}_1 \rrbracket, \text{int eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'>'} \text{Expr}_2 \rrbracket =$
 $\text{is-greater}(\text{int eval} \llbracket \text{Expr}_1 \rrbracket, \text{int eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'\&\&'} \text{Expr}_2 \rrbracket =$
 $\text{if-true-else}(\text{bool eval} \llbracket \text{Expr}_1 \rrbracket, \text{bool eval} \llbracket \text{Expr}_2 \rrbracket, \text{false})$

Rule $\text{eval} \llbracket \text{Expr}_1 \text{'||'} \text{Expr}_2 \rrbracket =$
 $\text{if-true-else}(\text{bool eval} \llbracket \text{Expr}_1 \rrbracket, \text{true}, \text{bool eval} \llbracket \text{Expr}_2 \rrbracket)$

Rule $\text{eval} \llbracket \text{'!' Expr} \rrbracket = \text{not}(\text{bool eval} \llbracket \text{Expr} \rrbracket)$

Rule $\text{eval} \llbracket \text{'new' ' (' ')} \rrbracket =$
 $\text{object}(\text{fresh-atom}, \text{"Object"}, \text{initialise-local-variables})$

Rule $\text{eval} \llbracket \text{'println' ' (' Expr ')} \rrbracket = \text{print-line sl-to-string eval} \llbracket \text{Expr} \rrbracket$

Rule $\text{eval} \llbracket \text{'readln' ' (' ')} \rrbracket = [\text{read-line}]$

Rule $\text{eval} \llbracket \text{'defineFunction' ' (' Expr ')} \rrbracket = \text{fail}$

Rule $\text{eval} \llbracket \text{'nanotime' ' (' ')} \rrbracket = \text{fail}$

Rule $\text{eval} \llbracket \text{'stacktrace' ' (' ')} \rrbracket = \text{fail}$

Otherwise $\text{eval} \llbracket \text{Id ' (' ExprList? ')} \rrbracket =$
 $\text{apply}(\text{fun global-bound eval} \llbracket \text{Id} \rrbracket, \text{eval-list} \llbracket \text{ExprList?} \rrbracket)$

Rule $\text{eval} \llbracket \text{Id} \rrbracket =$
 $\text{else}(\text{assigned local-variable id} \llbracket \text{Id} \rrbracket, \text{str id} \llbracket \text{Id} \rrbracket)$

Rule $\text{eval} \llbracket \text{Id '=' Expr} \rrbracket =$
 $\text{give}(\text{eval} \llbracket \text{Expr} \rrbracket,$
 $\text{sequential}(\text{local-variable-assign}(\text{id} \llbracket \text{Id} \rrbracket, \text{given}),$
 $\text{given}))$

Rule $\text{eval} \llbracket \text{Expr '.' Id} \rrbracket =$
 $\text{scope-closed}(\text{object-feature-map obj eval} \llbracket \text{Expr} \rrbracket,$
 $\text{else}(\text{assigned local-variable}_2 \text{id} \llbracket \text{Id} \rrbracket, \text{null-value}))$

Rule $\text{eval} \llbracket \text{Expr}_1 '.' \text{Id '=' Expr}_2 \rrbracket =$
 $\text{give}(\text{eval} \llbracket \text{Expr}_1 \rrbracket,$

Syntax $ExprList : \text{expr-list} ::= \text{expr } (', ' \text{expr-list})?$

Semantics $\text{eval-list}[_ : \text{expr-list?}] : \Rightarrow \text{lists}(\text{sl-values})$

Rule $\text{eval-list}[] = \text{nil}$

Rule $\text{eval-list}[Expr] = \text{cons}(\text{eval}[Expr], \text{nil})$

Rule $\text{eval-list}[Expr \text{ ' , ' } ExprList] =$
 $\text{cons}(\text{eval}[Expr], \text{eval-list}[ExprList])$