Languages-beta: SL-2-Expressions

The PLanCompS Project

`Languages-beta/SL/SL-2-Expressions/SL-2-Expressions.cbs`*

Language "SL"

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

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-list? `)`
| id
| id `=` expr
| expr `.` id
| expr `.` id `=` expr
| expr `.` id `(` expr-list? `)`
| `(` expr `)`

Rule ⟦ `(` *Expr* `)` ⟧ : expr =
⟦ *Expr* ⟧

Type sl-values ⇝ booleans | integers | strings | objects | null-type

```
Semantics eval⟦ Expr : expr ⟧ : ⇒ sl-values
    Rule eval⟦ Int ⟧ =
        int-val⟦ Int ⟧
    Rule eval⟦ String ⟧ =
        string-val⟦ String ⟧
    Rule eval⟦ true ⟧ =
        true
    Rule eval⟦ false ⟧ =
        false
    Rule eval⟦ Expr1 + Expr2 ⟧ =
        integer-add-else-string-append(eval⟦ Expr1 ⟧,
          eval⟦ Expr2 ⟧)
    Rule eval⟦ Expr1 / Expr2 ⟧ =
        checked integer-divide(int eval⟦ Expr1 ⟧,
            int eval⟦ Expr2 ⟧)
    Rule eval⟦ Expr1 * Expr2 ⟧ =
        integer-multiply(int eval⟦ Expr1 ⟧,
          int eval⟦ Expr2 ⟧)
    Rule eval⟦ Expr1 - Expr2 ⟧ =
        integer-subtract(int eval⟦ Expr1 ⟧,
          int eval⟦ Expr2 ⟧)
    Rule eval⟦ Expr1 == Expr2 ⟧ =
        is-equal(eval⟦ Expr1 ⟧,
          eval⟦ Expr2 ⟧)
    Rule eval⟦ Expr1 <= Expr2 ⟧ =
        is-less-or-equal(int eval⟦ Expr1 ⟧,
          int eval⟦ Expr2 ⟧)
    Rule eval⟦ Expr1 < Expr2 ⟧ =
        is-less(int eval⟦ Expr1 ⟧,
          int eval⟦ Expr2 ⟧)
    Rule eval⟦ Expr1 != Expr2 ⟧ =
        not is-equal(eval⟦ Expr1 ⟧,
            eval⟦ Expr2 ⟧)
    Rule eval⟦ Expr1 >= Expr2 ⟧ =
        is-greater-or-equal(int eval⟦ Expr1 ⟧,
          int eval⟦ Expr2 ⟧)
    Rule eval⟦ Expr1 > Expr2 ⟧ =
        is-greater(int eval⟦ Expr1 ⟧,
          int eval⟦ Expr2 ⟧) 3
    Rule eval⟦ Expr1 && Expr2 ⟧ =
        if-true-else(bool eval⟦ Expr1 ⟧,
          bool eval⟦ Expr2 ⟧,
          false)
    Rule eval⟦ Expr1 || Expr2 ⟧ =
        if-true-else(bool eval⟦ Expr1 ⟧,
```

Syntax *ExprList* : expr-list ::= expr (, expr-list)?

Semantics eval-list⟦ _ : expr-list? ⟧ : ⇒ lists(sl-values)

Rule eval-list⟦ ⟧ =

nil

Rule eval-list⟦ *Expr* ⟧ =

cons(eval⟦ *Expr* ⟧,

nil)

Rule eval-list⟦ *Expr* , *ExprList* ⟧ =

cons(eval⟦ *Expr* ⟧,

eval-list⟦ *ExprList* ⟧)