Plum

v0.0.1

https://github.com/SillyFreak/typst-plum

Clemens Koza

Abstract

Plum lets you create UML class diagrams in Typst; inspired by but *not* compatible with PlantUML.

Contents

I Introduction

This is a template for typst packages. It provides the `parse()` and `eval()` functions:

lib.typ

```typ
#let parse(expr) = {
  // this is the "interesting" part: calling into the Rust parser
  cbor.decode(_p.parse(cbor.encode(expr)))
}
/// -> int
#let eval(expr, ..vars) = {
  assert(vars.pos().len() == 0)
  let vars = vars.named()

  let inner-eval(expr) = {
    if expr.type == "number" { expr.value }
    else if expr.type == "variable" { vars.at(expr.name) }
    else if expr.type == "binary" {
      let (operator, left, right) = expr
      (left, right) = (inner-eval(left), inner-eval(right))
      if operator == "add" { left + right }
      else if operator == "sub" { left - right }
      else if operator == "mul" { left * right }
      else if operator == "div" { left / right }
      else { panic("unexpected binary operator: " + operator) }
    }
    else { panic("unexpected expression type: " + expr.type) }
  }

  inner-eval(parse(expr))
}
```

Here they are in action:

```typ
$2 * (2 + x) arrow.double.long$ #plum.parse("2 * (2 + x)")
```

$2 * (2 + x) \Longrightarrow$ (
  type: "binary",
  operator: "mul",
  left: (type: "number", value: 2),
  right: (
    type: "binary",
    operator: "add",
    left: (type: "number", value: 2),
    right: (type: "variable", name: "x"),
  ),
)

```typ
$2 * (2 + x) arrow.double.long^(x=3)$ #plum.eval("2 * (2 + x)", x: 3)
```

$2 * (2 + x) \overset{x=3}{\Longrightarrow} 10$

II Module reference

II.a `plum`

`parse(expr: str) -> dict`

Parses an expression via a WASM plugin.

```typ
#plum.parse("1") \
#plum.parse("1 + 1") \
#plum.parse("foo") \
#plum.parse("foo + 1")
```

```
(type: "number", value: 1)
(
  type: "binary",
  operator: "add",
  left: (type: "number", value: 1),
  right: (type: "number", value: 1),
)
(type: "variable", name: "foo")
(
  type: "binary",
  operator: "add",
  left: (type: "variable", name: "foo"),
  right: (type: "number", value: 1),
)
```

Parameters:

`expr` (`str`) – the expression to parse

`eval(expr: str, ..vars: arguments) -> int`

Evaluates an expression in Typst, by traversing the abstract syntax tree (AST) created in Rust.

```typ
#plum.eval("1") \
#plum.eval("1 + 1") \
#plum.eval("foo", foo: 1) \
#plum.eval("foo + 1", foo: 1)
```

```
1
2
1
2
```

Parameters:

`expr` (`str`) – the expression to evaluate

`..vars` (`arguments`) – the variable assignments in the expression