

Compiladores II

Fabio Mascarenhas - 2014.2

<http://www.dcc.ufrj.br/~fabiom/comp2>

SmallLua - sintaxe

```
bloco <- stat* (ret / '')
stat  <- "while" exp "do" bloco "end" / "local" "id" "=" exp /
      "id" "=" exp / "function" "id" "(" (ids / '') ")" bloco "end" /
      "if" exp "then" bloco ("else" bloco / '') "end" /
      pexp
ret   <- "return" exp
ids   <- "id" ("," "id")*
exps  <- exp ("," exp)*
exp   <- lexp ("or" lexp)*
lexp  <- rexp ("and" rexp)*
rexp  <- cexp (rop cexp)*
cexp  <- aexp ".." cexp / aexp
aexp  <- mexp (aop mexp)*
mexp  <- sexp (mop sexp)*
sexp  <- "-" sexp / "not" sexp / "false" / "true" / "number" /
      string "string" / lmb / pexp
lmb   <- "function" "(" (ids / '') ")" bloco "end"
pexp  <- "(" (" exp ")" / "id") "(" (" (exps / '') ")" )"*
rop   <- "<" / "==" / "~="
aop   <- "+" / "-"
mop   <- "*" / "/"
```

boolean boolean number expressions

Dojo

- Implementem o restante do verificador de tipos de SmallLua, como uma função definida por casos
- O verificador deve produzir um tipo e o ambiente com as declarações do programa, ou lançar um erro com o primeiro erro de tipo encontrado

$\Gamma \vdash n : \text{number}$

$\Gamma \vdash \text{true} : \text{boolean}$

$\Gamma \vdash x : T[x]$

$\Gamma \vdash s : \text{string}$

$\Gamma \vdash \text{false} : \text{boolean}$

$\Gamma \vdash e : t \quad t = T[x]$
 $\Gamma \vdash x = e : ()_1 T$

$\Gamma \vdash f : (t_1, \dots, t_n) \rightarrow t_n \quad \Gamma \vdash e_k : t_k$

$\Gamma \vdash f(e_1, \dots, e_n) : t_n, \textcircled{T}$

Sequências

- Para o sistema de tipos de SmallLua ficar interessante, vamos acrescentar um tipo estruturado: *sequências*
- Um tipo $\{ t \}$ é uma sequência de itens de tipo t
- Construimos sequências com os operadores $\text{seq}(\dots)$, que recebe os elementos da sequência como argumentos, e \dots , que concatena duas sequências
- Decompomos sequências com os operadores $\text{byte}(s, i)$, que retorna o i -ésimo elemento da sequência, e $\text{sub}(s, i, j)$, que retorna a subsequência entre os elementos i e j (inclusive)

Regras de dedução - sequências

$$\frac{T \vdash e_k : t}{T \vdash \text{seq}(e_1, \dots, e_n) : \{t\}}$$

$$\frac{T \vdash e_1 : \{t\} \quad T \vdash e_2 : \{t\}}{T \vdash e_1 \cdot e_2 : \{t\}}$$

$$\frac{T \vdash e_1 : \{t\} \quad T \vdash e_2 : \text{min} \quad T \vdash e_3 : \text{max}}{T \vdash \text{sub}(e_1, e_2, e_3) : \{t\}}$$

$$\frac{T \vdash e_1 : \{t\} \quad T \vdash e_2 : \text{number}}{T \vdash \text{byte}(e_1, e_2) : t}$$

Operadores vs funções

- Classificamos `seq`, `byte` e `sub` como operadores, ao invés de funções, por quê?
- Qual seria o tipo de `seq()` (uma sequência vazia)?
- Nosso sistema de tipos é *monomórfico*: cada termo do sistema pode ter apenas um tipo, e isso vale para as funções
- Logo, se `byte` e `sub` fossem funções só poderiam receber sequências de um tipo pré-determinado, e `seq()` sempre retornaria uma sequência do mesmo tipo
- Podemos resolver esses problemas com um sistema *polimórfico*