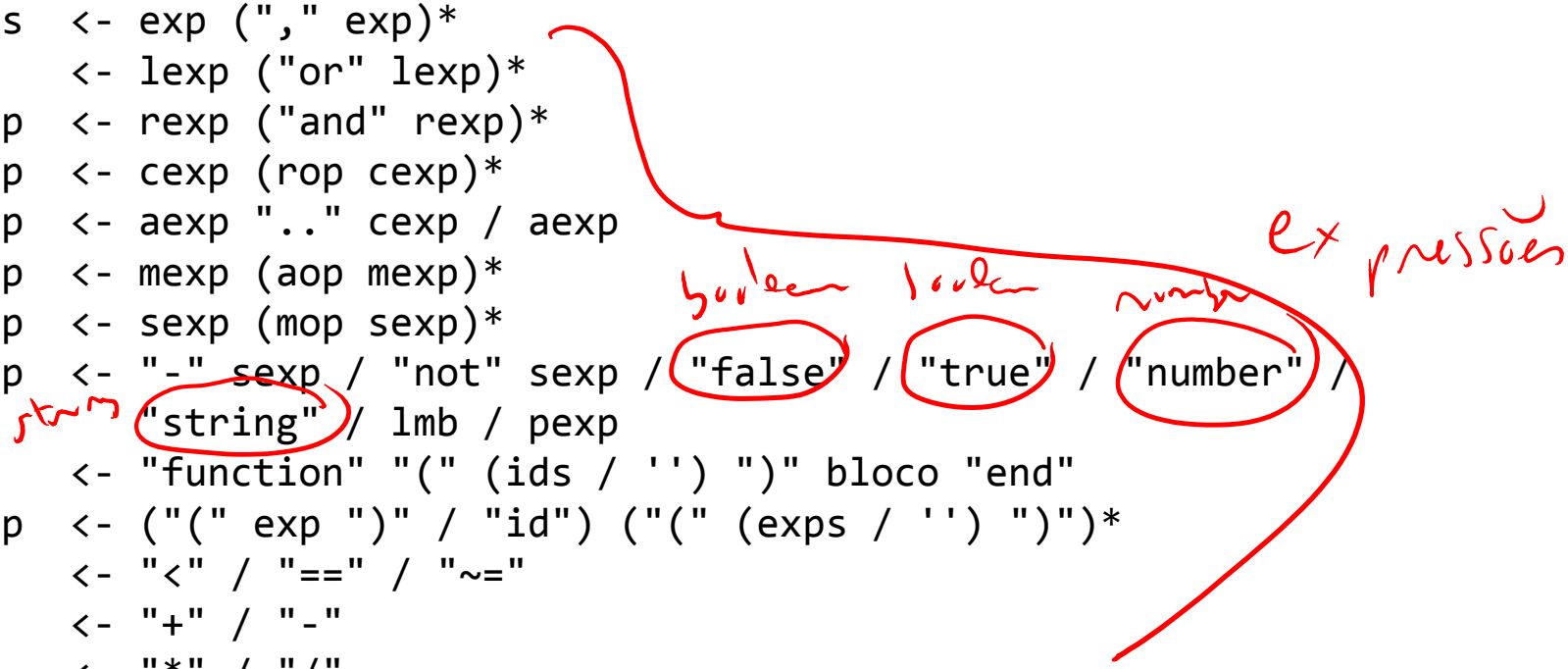
Compiladores II

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" / lmb / pexp
lmb   <- "function" "(" (ids / '') ")" bloco "end"
pexp  <- ("(" exp ")" / "id") ("(" (exps / '') ")")*
rop   <- "<" / "==" / "~="
aop   <- "+" / "-"
mop   <- "*" / "/"
```



Inferência de tipos - especialização

- A especialização de tipos agora pode deixar variáveis de tipo em aberto, que vão seguir no processo de inferência, permitindo inferir tipos como o da função abaixo:

```
function primeiro(s)
    return byte(s, 1)
end
```

- Também podemos inferir tipos mais específicos:

```
function foo(s)
    return byte(s, 1) + 0
end
```

Sobrecarga

- A operação de concatenação em SmallLua é *sobrecarregada*: ela funciona tanto para strings quanto para sequências

- Podemos fazer a verificação de tipos da concatenação tentando primeiro unificar seus argumentos com strings, para depois unificar com tipos sequência

- Mas isso nos impede de escrever funções genéricas usando concatenação:

```
function concat(s)
    if len(s) == 1 then
        return byte(s, 1)
    else
        return byte(s, 1) ..
               concat(sub(s, 2, len(s)))
    end
end
```

Tags

- A ideia para permitir funções genéricas sobrecarregadas é ter *tags* associadas a variáveis de tipo em aberto e a parâmetros de tipo

- Essas tags limitam a variável ou parâmetro a tipos que suportam dado conjunto de operações sobrecarregadas

- Em SmallLua vamos usar duas tags: `eq` para tipos que podem ser comparados com `==` e `concat` para tipos que podem ser concatenados

- Quando unificamos duas variáveis em aberto unimos seus conjuntos de tags, e quando unificamos uma variável em aberto com outro tipo qualquer verificamos se esse tipo aceita as tags da variável