

PhD Qualifying exam on Programming Languages

Adriana Compagnoni

September 2003

This is an open-books and open-notes exam.

The language EXP is defined by the following grammar:

$$\begin{aligned}
 E ::= & \text{ true} | \text{ false} | N \\
 & | E + E | E - E | E \wedge E | E \vee E | \text{ not } E \\
 & | \text{ if } E \text{ then } E \text{ else } E | (E)
 \end{aligned}$$

where N is a countable set of numeric constants, and we will say $n \in N$ to express the fact that n is an element of the set N .

The typed language T-EXP consists of adding types Bool and Num to EXP expressions with the following axioms and rules:

$$\begin{array}{ll}
 \text{true} : \text{Bool} & (\langle \text{T-BOOL} \rangle) \\
 \text{false} : \text{Bool} & (\langle \text{F-BOOL} \rangle) \\
 \frac{n \in N}{n : \text{Num}} & (\langle \text{N-NUM} \rangle) \\
 \frac{e : \text{Num} \quad f : \text{Num}}{e + f : \text{Num}} & (\langle +\text{-NUM} \rangle) \\
 \frac{e : \text{Num} \quad f : \text{Num}}{e - f : \text{Num}} & (\langle -\text{-NUM} \rangle) \\
 \frac{e : \text{Bool} \quad f : \text{Bool}}{e \wedge f : \text{Bool}} & (\langle \wedge\text{-BOOL} \rangle) \\
 \frac{e : \text{Bool} \quad f : \text{Bool}}{e \vee f : \text{Bool}} & (\langle \vee\text{-BOOL} \rangle) \\
 \frac{e : \text{Bool}}{\text{not } e : \text{Bool}} & (\langle \text{NOT-BOOL} \rangle) \\
 \frac{e : \text{Bool} \quad f : T \quad g : T}{\text{if } e \text{ then } f \text{ else } g : T} & (\langle \text{IF-T} \rangle) \\
 \frac{e : T}{(e) : T} & (\langle ()\text{-T} \rangle)
 \end{array}$$

Observation: How to read axioms and rules

Consider the following examples.

The axiom $\langle T\text{-BOOL} \rangle$ means that the constant *true* has type *Bool*.

The rule $\langle +\text{-NUM} \rangle$ means that if expression *e* has type *Num* and expression *f* has type *Num*, then expression $e + f$ has type *Num* as well.

This notation allows us to justify a statement of the form

$$e \text{ has type } T$$

constructing the derivation tree of a statement of the form

$$e : T$$

in the language T-EXP.

For example, to show that

$$(3 + 4) - 1 \text{ has type } Num$$

we construct a derivation tree ending with conclusion:

$$(3 + 4) - 1 : Num$$

as follows:

$$\begin{array}{c}
 \begin{array}{cc}
 3 \text{ in } N & 4 \text{ in } N \\
 \langle n\text{-Num} \rangle \text{ -----} & \langle n\text{-Num} \rangle \text{ -----} \\
 3 : N & 4 : N \\
 \langle +\text{-Num} \rangle \text{ -----} & \\
 3+4 : Num & \\
 \langle ()\text{-T} \rangle \text{ -----} & \langle n\text{-Num} \rangle \text{ -----} \\
 (3+4) : Num & 1 \text{ in } N \\
 & 1 : N \\
 \langle -\text{-Num} \rangle \text{ -----} & \\
 & (3+4)-1 : Num
 \end{array}
 \end{array}$$

On the other hand,

$$6 \wedge true : Num$$

is not derivable in T-EXP, because the rule $\langle \wedge\text{-BOOL} \rangle$, the only rule to derive a statement of the form $e \wedge f : T$, requires *T* to be *Bool*.

Exercises

1. Show that the expression

if 3 then 4 else true

is in the language EXP. In other words, write a derivation of:

$E \rightarrow^* \textit{if 3 then 4 else true}$

2. Show that

if 3 then 4 else true

cannot be derived using the given typing rules of T-EXP. In other words, show that

if 3 then 4 else true

is an expression that cannot be given a type in the language T-EXP.

3. Write a derivation in T-EXP of

if (not true) then (3 + 4) else 0 : Num

4. Implement a typechecker for T-EXP. You may use ML or SCHEME.

Good luck!