

## Comp 411 Notes - 2 Feb 2005

### A SIMPLE TYPE SYSTEM

Reproduced by Mathias Ricken  
(mgricken@cs.rice.edu)

## 1 Important Points from Previous Lecture

- Good Morning, COMP 411!

## 2 A Simple Type System

Consider our language from a few lectures ago:

$$e \in \mathbb{E} ::= T \mid F \mid \text{if } e \text{ then } e \text{ else } e \\ 0 \mid e + \mid e - \mid \text{isZero } e$$

How do we begin defining a type system for this language? We can provide a syntax for the types:

$$t \in \mathbb{T} ::= \text{bool} \mid \text{nat}$$

We define the syntax because we need the types to be formal objects. We want to reason about them and provide a function  $\mathbb{E} \rightarrow \mathbb{T}$ , and for this,  $\mathbb{E}$  and  $\mathbb{T}$  need to be formally defined.

## 3 Well-Formed Types

It is sensible to first define what a sensible type is:

$$\overline{\vdash_T \text{bool}} \quad \overline{\vdash_T \text{nat}}$$

In these cases, *bool* and *nat* are always well-formed, they don't have any antecedents. Sometimes we do need to talk about the validity of a type, e.g. when we combine types such as  $a \times b$ ,  $a, b \in \mathbb{T}$ .

## 4 Types for Expressions

We now define a set of rules to determine the type of expressions:

$$\begin{array}{c}
\frac{}{\vdash_E T : \mathit{bool}} \quad \frac{\vdash_E e_1 : \mathit{bool} \quad \vdash_E e_2 : t \quad \vdash_E e_3 : t}{\vdash_E \mathbf{if} \ e_1 \ \mathbf{then} \ e_2 \ \mathbf{else} \ e_3 : t} \quad \frac{}{\vdash_E F : \mathit{bool}} \\
\frac{}{\vdash_E 0 : \mathit{nat}} \quad \frac{\vdash_E e : \mathit{nat}}{\vdash_E e+ : \mathit{nat}} \quad \frac{\vdash_E e : \mathit{nat}}{\vdash_E e- : \mathit{nat}} \\
\frac{\vdash_E e : \mathit{nat}}{\vdash_E \mathbf{iszero} \ e : \mathit{bool}}
\end{array}$$

### Rule Interpretation

Generally, a rule such as

$$\frac{\vdash_E e_1 : \mathit{bool} \quad \vdash_E e_2 : t \quad \vdash_E e_3 : t}{\vdash_E \mathbf{if} \ e_1 \ \mathbf{then} \ e_2 \ \mathbf{else} \ e_3 : t}$$

has the form

$$\frac{P(\bar{X}, \bar{Y})}{Q(\bar{X})} = \forall \bar{X}. (\exists \bar{Y}. P(\bar{X}, \bar{Y})) \Rightarrow Q(\bar{X}),$$

where  $\bar{X}$  and  $\bar{Y}$  are a vectors of variables  $(x_1, x_2, \dots, x_n)$  and  $(y_1, y_2, \dots, y_m)$ .

This can be read as “There are two predicates, P and Q, and for all  $\bar{X}$  if you can find a  $\bar{Y}$  such that  $P(\bar{X}, \bar{Y})$  is true for a particular  $\bar{X}$ , then  $Q(\bar{X})$  is true for that particular  $\bar{X}$ ”.

$\bar{X}$  is a vector of variables that appear both in the antecedent and the consequent, and  $\bar{Y}$  is a vector of variables that are used additionally in the antecedent.

In the rule for the **if** expression above, for example,  $\bar{X} = (e_1, e_2, e_3, t)$ ,  $\bar{Y} = ()$ ,  $P(\bar{X}, \bar{Y}) = [\vdash_E x_1 : \mathit{bool}] \wedge [\vdash_E x_2 : x_4] \wedge [\vdash_E x_3 : x_4]$ , and  $Q(\bar{X}) = [\vdash_E \mathbf{if} \ x_1 \ \mathbf{then} \ x_2 \ \mathbf{else} \ x_3 : x_4]$ .

### Type of 0–

Right now it is typed as *nat*. Is this bad? What is the type of 13/0 in other languages such as C or Java? It is usually *int* (or *nat* here). For now, we type 0– as *nat*.

## 5 Type System Uses

We can use this type system to compare and contrast different  $e \in \mathbb{E}$  that either type or do not type.

## 5.1 Type Safety Preservation

### A first attempt at a theorem

$$\forall e \in \mathbb{E}. \forall t \in \mathbb{T}. \vdash_E e : t \Rightarrow (e \mapsto^* v) \wedge (v : t)$$

This is **incorrect**.  $e$  might have a type, but the reduction might not terminate:  $e \not\mapsto^*$ .

Example:  $(0-) : nat, (0-) \notin v, (0-) \not\mapsto$  (stuck).

### Corrected theorem

$$\forall e \in \mathbb{E}. \forall t \in \mathbb{T}. \forall v \in \mathbb{V}. (\vdash_E e : t) \wedge (e \mapsto^* v) \Rightarrow v : t$$

This theorem correctly expresses what happens: “For all  $e, t, v$  such that  $e$  has type  $t$  and  $e$  reduces in a number of steps to  $v$ , that implies that  $v$  has the same type  $t$  as  $e$ .”

However, this is not type safety, which is why the “Safety” in the subheading above has been struck out. The property the theorem expresses can more adequately be described as “type consistency” or “type preservation” (in big-step semantics) and “subject reduction” (in small-step semantics.)

#### 5.1.1 Property: Results Must Have Same Type as $e$

The theorem does not imply that  $e$  must evaluate to a unique value  $v$ ; it might evaluate to a number of values  $v_1, v_2, \dots, v_n$ , but all of these must have the same type  $t$  as the  $e$ .

Note that our definition of big-step and small-step semantics prove that the  $v$  is unique.

#### 5.1.2 Property: Types of $e \subseteq$ Types of $v$

The theorem states that we must be able to assign  $v$  the all the types that we can assign to  $e$ . It does not state that both sets of types need to be the same. The set of types assignable to  $v$  might contain types we cannot assign to  $e$ ; thus, the set of types of  $e$  must be a subset of the set of types of  $v$ .

## 5.2 Progress

In small-step semantics, we had many cases where we did not know what to do since no rule applied. For this discussion, let us include the rule  $R_{0-} \frac{}{0- \mapsto 0}$ .

We want all cases without rules to be type errors.

**Theorem**

$$\forall e \in \mathbb{E}. \forall t \in \mathbb{T}. (e \notin \mathbb{V}) \wedge (\vdash_E e : t) \Rightarrow \exists e'. [(e \mapsto e') \wedge (\vdash_E e' : t)]$$

This can be interpreted as “One small step will not get us stuck”.

Type safety usually deals with multiple steps.