Chapter 8: Typed Arithmetic Expressions

Types

The Typing Relation

Safety = Progress + Preservation
t ::= 
  true 
  false 
  if t then t else t 
  0 
  succ t 
  pred t 
  iszero t

**Evaluation**

\[
\begin{align*}
\text{if true then } t_2 \text{ else } t_3 & \rightarrow t_2 \\
\text{if false then } t_2 \text{ else } t_3 & \rightarrow t_3 \\
\quad t_1 & \rightarrow t'_1 \\
\quad \text{if } t_1 \text{ then } t_2 \text{ else } t_3 & \rightarrow \text{if } t'_1 \text{ then } t_2 \text{ else } t_3
\end{align*}
\]
Evaluation Results

• Values

\[
v ::= \\
  \text{true} \\
  \text{false} \\
  \text{nv} \\
\]

\[
nv ::= \\
  0 \\
  \text{succ \ nv} \\
\]

values:
  true value
  false value
  numeric value

numeric values:
  zero value
  successor value

• Get stuck (i.e., pred false)
Types of Terms

• Can we tell, without actually evaluating a term, that the term evaluation will not get stuck?

• Distinguish two types of terms:
  – Nat: terms whose results will be a numeric value
  – Bool: terms whose results will be a Boolean value

• “a term t has type T” means that t “obviously” (statically) evaluates to a value of T
  – if true then false else true has type Bool
  – pred (succ (pred (succ 0))) has type Nat
The Typing Relation: $t : T$
Typing Rule for Booleans

New syntactic forms

\[
T ::= \text{Bool}
\]

types:

type of booleans

New typing rules

\[
\text{true : Bool} \\
\text{false : Bool}
\]

\[
\begin{array}{lll}
  t_1 : \text{Bool} & t_2 : T & t_3 : T \\
\hline
  \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T
\end{array}
\]
Typing Rules for Numbers

New syntactic forms
\[ T ::= \ldots \]
\[ \text{Nat} \]

New typing rules
\[ t : T \]
\[ 0 : \text{Nat} \]

Types:
\[ \frac{}{\text{succ } t_1 : \text{Nat}} \]
\[ \frac{}{\text{pred } t_1 : \text{Nat}} \]
\[ \frac{}{\text{iszero } t_1 : \text{Bool}} \]

(T-Succ)

(T-Pred)

(T-IsZero)
Typing Relation: Formal Definition

- **Definition**: the *typing relation* for arithmetic expressions is the *smallest binary relation* between terms and types satisfying all instances of the typing rules.

- A term $t$ is *typable (or well typed)* if there is some $T$ such that $t : T$. 


Inversion Lemma (Generation Lemma)

• Given a valid typing statement, it shows
  – how a proof of this statement could have been generated;
  – a recursive algorithm for calculating the types of terms.

**Lemma [Inversion of the Typing Relation]:**

1. If true : R, then R = Bool.
2. If false : R, then R = Bool.
3. If if t₁ then t₂ else t₃ : R, then t₁ : Bool, t₂ : R, and t₃ : R.
4. If 0 : R, then R = Nat.
5. If succ t₁ : R, then R = Nat and t₁ : Nat.
6. If pred t₁ : R, then R = Nat and t₁ : Nat.
7. If iszero t₁ : R, then R = Bool and t₁ : Nat.
Typing Derivation

Statements are formal assertions about the typing of programs.
Typing rules are implications between statements.
Derivations are deductions based on typing rules.
Uniqueness of Types

- **Theorem** [Uniqueness of Types]: Each term $t$ has at most one type. That is, if $t$ is typable, then its type is unique.

- Note: later on, we may have a type system where a term may have many types.
Safety = Progress + Preservation
Safety (Soundness)

• By safety, it means well-typed terms do not “go wrong”.

• By “go wrong”, it means reaching a “stuck state” that is not a final value but where the evaluation rules do not tell what to do next.
Safety = Progress + Preservation

Well-typed terms do not get stuck

- **Progress**: A well-typed term is not stuck (either it is a value or it can take a step according to the evaluation rules).

- **Preservation**: If a well-typed term takes a step of evaluation, then the resulting term is also well typed.
Progress

• **Theorem [Progress]:** Suppose $t$ is a well-typed term (that is, $t : T$ for some $T$). Then either $t$ is a value or else there is some $t'$ with $t \rightarrow t'$.

Proof: By induction on a derivation of $t : T$.

- case $T$-True: $\text{true} : \text{Bool}$ OK?
- case $T$-If:
  - $t1 : \text{Bool}$, $t2 : T$, $t3 : T$
  - ----------------------------- OK?
  - if $t1$ then $t2$ else $t3$ : $T$
- ...
Preservation

• **Theorem** [Preservation]:
  If $t : T$ and $t \rightarrow t'$, then $t' : T$.

Proof: By induction on a derivation of $t : T$.

- case T-True: $\text{true} : \text{Bool}$  OK?
- case T-If:
  $t1 : \text{Bool}, t2 : T, t3 : T$
  -----------------------------  OK?
  if $t1$ then $t2$ else $t3 : T$

- ...

Note: The preservation theorem is often called **subject reduction property** (or **subject evaluation property**)
Homework

• Read Chapter 8.
• Do Exercises 8.3.7

8.3.7 Exercise [RECOMMENDED, ★★]: Suppose our evaluation relation is defined in the big-step style, as in Exercise 3.5.17. How should the intuitive property of type safety be formalized?