Chapter 8: Typed Arithmetic Expressions

Types

The Typing Relation

Safety = Progress + Preservation
Reall: Syntax and Semantics

\[ t ::= \begin{align*} & \text{true} \\
& \text{false} \\
& \text{if } t \text{ then } t \text{ else } t \\
& 0 \\
& \text{succ } t \\
& \text{pred } t \\
& \text{iszero } t \end{align*} \]

Evaluation:
\[ \frac{t \rightarrow t'}{\text{if } t \text{ then } t_2 \text{ else } t_3 \rightarrow t_2} \quad \text{(E-IFTRUE)} \]
\[ \frac{t_1 \rightarrow t'_1}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{if } t'_1 \text{ then } t_2 \text{ else } t_3} \quad \text{(E-IF)} \]
\[ \frac{t_1 \rightarrow t'_1}{\text{pred } t_1 \rightarrow \text{pred } t'_1} \quad \text{(E-PRED)} \]
\[ \frac{\text{pred } 0 \rightarrow 0}{\text{(E-PREDZERO)}} \]
\[ \frac{\text{pred } (\text{succ } n v_1) \rightarrow n v_1}{\text{(E-PREDSUCC)}} \]
\[ \frac{t_1 \rightarrow t'_1}{\text{iszero } t_1 \rightarrow \text{iszero } t'_1} \quad \text{(E-ISZERO)} \]
\[ \frac{\text{iszero } 0 \rightarrow \text{true}}{\text{(E-ISZEROZERO)}} \]
\[ \frac{\text{iszero } (\text{succ } n v_1) \rightarrow \text{false}}{\text{(E-ISZEROSUCC)}} \]
Evaluation Results

- Values

\[
\begin{align*}
v & ::= \\
& \quad \text{true} \\
& \quad \text{false} \\
& \quad \text{nv}
\end{align*}
\]

\[
\begin{align*}
nv & ::= \\
& \quad 0 \\
& \quad \text{succ nv}
\end{align*}
\]

- 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

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

message

New typing rules

\[ \text{true: Bool} \]

(T-TRUE)

\[ \text{false: Bool} \]

(T-FALSE)

\[ t_1: \text{Bool} \quad t_2: T \quad t_3: T \]

if \( t_1 \) then \( t_2 \) else \( t_3 \): T
Typing Rules for Numbers

New syntactic forms

\[ T ::= \cdots \]

- Nat

New typing rules

- 0 : Nat

<table>
<thead>
<tr>
<th>types:</th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| succ \( t_1 \) : Nat | \( t_1 \) : Nat | (T-SUCCE)
| pred \( t_1 \) : Nat | \( t_1 \) : Nat | (T-PRED)
| iszero \( t_1 \) : Bool | \( t_1 \) : Nat | (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 \( t_1 \) then \( t_2 \) else \( t_3 \) : R, then \( t_1 : \text{Bool} \), \( t_2 : R \), and \( t_3 : R \).
4. If 0 : R, then R = Nat.
5. If succ \( t_1 \) : R, then R = Nat and \( t_1 : \text{Nat} \).
6. If pred \( t_1 \) : R, then R = Nat and \( t_1 : \text{Nat} \).
7. If iszero \( t_1 \) : R, then R = Bool and \( t_1 : \text{Nat} \).
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.
Canonical Form

• Lemma [Canonical Forms]:
  – If \( v \) is a value of type \( \text{Bool} \), then \( v \) is either true or false.
  – If \( v \) is a value of type \( \text{Nat} \), then \( v \) is a numeric value according to the grammar for \( \text{nv} \).

\[
\begin{align*}
v & ::= \\
& \quad \text{true} \\
& \quad \text{false} \\
& \quad \text{nv} \\
\end{align*}
\]

\[
\begin{align*}
nv & ::= \\
& \quad 0 \\
& \quad \text{succ } nv \\
\end{align*}
\]

values:
- true value
- false value
- numeric value

numeric values:
- zero value
- successor value
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:
  \[
  \begin{align*}
  t1 : \text{Bool}, & \ t2 : T, \ t3 : T \\
  \text{------------------------------------------} & \ \text{OK}?
  \end{align*}
  \]
  \[
  \begin{align*}
  \text{if } t1 \text{ then } t2 \text{ else } t3 : T
  \end{align*}
  \]
- ...
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?