Chapter 8: Typed Arithmetic Expressions

Types
The Typing Relation
Safety = Progress + Preservation
Reall: Syntax and Semantics

\[ t ::= \]

true
false
if \( t \) then \( t \) else \( t \)
0
\text{succ} \( t \)
\text{pred} \( t \)
iszero \( t \)

**Evaluation**

\[
\frac{t \rightarrow t'}{\text{if } t \text{ then } t_2 \text{ else } t_3 \rightarrow t_2} \quad \text{(E-IFTRUE)}
\]

\[
\frac{t \rightarrow t'}{\text{if } t \text{ then } t_2 \text{ else } t_3 \rightarrow t_3} \quad \text{(E-IFFALSE)}
\]

\[
\frac{t_1 \rightarrow t'_1}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{if } t'_1 \text{ then } t_2 \text{ else } t_3} \quad \text{(E-IF)}
\]

\[
\frac{t_1 \rightarrow t'_1}{\text{pred } t_1 \rightarrow \text{pred } t'_1} \quad \text{(E-PRED)}
\]

\[
\frac{t_1 \rightarrow t'_1}{\text{pred } (\text{succ } n v_1) \rightarrow n v_1} \quad \text{(E-PREDSUCC)}
\]

\[
\frac{t_1 \rightarrow t'_1}{\text{pred } 0 \rightarrow 0} \quad \text{(E-PREDZERO)}
\]

\[
\frac{t \rightarrow t'}{\text{iszero } 0 \rightarrow \text{true}} \quad \text{(E-ISZEROZERO)}
\]

\[
\frac{t \rightarrow t'}{\text{iszero } (\text{succ } n v_1) \rightarrow \text{false}} \quad \text{(E-ISZEROSUCC)}
\]

\[
\frac{t_1 \rightarrow t'_1}{\text{iszero } t_1 \rightarrow \text{iszero } t'_1} \quad \text{(E-ISZERO)}
\]
Evaluation Results

• Values

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

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

• 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

\[ T ::= \]

\[ \text{Bool} \]

\[ \text{type of booleans} \]

New typing rules

\[ \text{true: Bool} \]

\[ \text{false: Bool} \]

\[ \frac{t_1: \text{Bool} \quad t_2: T \quad t_3: T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3: T} \]
Typing Rules for Numbers

New syntactic forms

\[ T ::= \ldots \quad \text{types:} \]

- Nat

New typing rules

\[
\begin{align*}
0 & : \text{Nat} \\
\end{align*}
\]

(t-ZERO)

\[
\begin{align*}
t_1 & : \text{Nat} \\
\text{succ } t_1 & : \text{Nat} \\
\text{pred } t_1 & : \text{Nat} \\
\text{iszero } t_1 & : \text{Bool}
\end{align*}
\]

(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 $\text{true} : R$, then $R = \text{Bool}$.
2. If $\text{false} : R$, then $R = \text{Bool}$.
3. If $\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$, then $t_1 : \text{Bool}$, $t_2 : R$, and $t_3 : R$.
4. If $0 : R$, then $R = \text{Nat}$.
5. If $\text{succ } t_1 : R$, then $R = \text{Nat}$ and $t_1 : \text{Nat}$.
6. If $\text{pred } t_1 : R$, then $R = \text{Nat}$ and $t_1 : \text{Nat}$.
7. If $\text{iszero } t_1 : R$, then $R = \text{Bool}$ and $t_1 : \text{Nat}$.
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 \( \text{nv} \).

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

values:
\[
\begin{align*}
  \text{true value} \\
  \text{false value} \\
  \text{numeric value}
\end{align*}
\]

numeric values:
\[
\begin{align*}
  \text{zero value} \\
  \text{successor value}
\end{align*}
\]
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: \texttt{true} : \texttt{Bool} \quad \text{OK?}
- case T-If:
  
  \[ t1 : \texttt{Bool}, t2 : T, t3 : T \]
  
  \[ \text{-----------------------------} \quad \text{OK?} \]
  
  \[ \text{if } t1 \text{ then } t2 \text{ else } t3 : T \]
  
  - ...
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: \texttt{true} : \texttt{Bool} \ OK?
- case T-If:
  \( t1 : \texttt{Bool}, t2 : T, t3 : T \)
  
  -------------------------- \ OK?

  \( \text{if } t1 \text{ then } t2 \text{ else } t3 : T \)

- ...

Note: The preservation theorem is often called \textit{subject reduction property} (or \textit{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?