Chapter 8: Typed Arithmetic Expressions

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

t ::= 
    true
    false
    if t then t else t
    0
    succ t
    pred t
    iszero t

terms:
    constant true
    constant false
    conditional constant
    zero
    successor predecessor
    zero test
Evaluation Results

• Values

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

• 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

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

Types:

- type of booleans

New typing rules

\[
\begin{align*}
\text{true} : \text{Bool} & \\
\text{false} : \text{Bool} & \\
\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} & \quad (T-IF)
\end{align*}
\]
Typing Rules for Numbers

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

\text{type of natural numbers}

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

\text{T-ZERO}

Types:
- \text{t_1 : Nat}
- \frac{\text{succ }}{\text{t_1 : Nat}} \quad \text{T-SUCC}
- \frac{\text{t_1 : Nat}}{\text{pred t_1 : Nat}} \quad \text{T-PRED}
- \frac{\text{t_1 : Nat}}{\text{iszero t_1 : Bool}} \quad \text{T-ISZERO}
• **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.
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 \( \text{nv} \).

\[
\begin{align*}
v & ::= \quad \text{values:} \\
& \quad \text{true value} \\
& \quad \text{false value} \\
& \quad \text{numeric value} \\
\text{true} \quad & \text{numeric values:} \\
\text{false} \quad & \text{zero value} \\
nv & ::= \quad \text{successor value} \\
& 0 \\
& \text{succ } nv
\end{align*}
\]
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 \( t : T \).
Preservation

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

  **Proof:** By induction on a derivation of \( t : T \).

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

- Read Chapter 8.
- Do Exercises 8.3.7