Chapter 8: Typed Arithmetic Expressions

Types

The Typing Relation

Safety = Progress + Preservation

Reall: Syntax

t ::=	terms:
true	constant true
false	constant false
if t then t else t	conditional constant
0	zero
succ t	successor predecessor
pred t	zero test
iszero t	

Evaluation Results

- Values

v ::=		*values:*
	true	*true value*
	false	*false value*
	nv	*numeric value*
nv ::=		*numeric values:*
	0	*zero value*
	succ nv	*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 ::=		*types:*
	Bool	*type of booleans*

New typing rules $\boxed{t : T}$

$$\texttt{true} : \texttt{Bool} \quad \text{(T-TRUE)}$$

$$\texttt{false} : \texttt{Bool} \quad \text{(T-FALSE)}$$

$$\frac{t_1 : \texttt{Bool} \qquad t_2 : \boxed{T} \qquad t_3 : \boxed{T}}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T} \quad \text{(T-IF)}$$

Typing Rules for Numbers

New syntactic forms

T	::=	...	*types:*
		Nat	*type of natural numbers*

New typing rules $\boxed{t : T}$

$$0 : \text{Nat} \qquad \text{(T-Zero)}$$

$$\frac{t_1 : \text{Nat}}{\text{succ } t_1 : \text{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{t_1 : \text{Nat}}{\text{pred } t_1 : \text{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{t_1 : \text{Nat}}{\text{iszero } t_1 : \text{Bool}} \qquad \text{(T-IsZero)}$$

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 : `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 : `Nat`.
6. If `pred` t_1 : R, then R = `Nat` and t_1 : `Nat`.
7. If `iszero` t_1 : R, then R = `Bool` and t_1 : `Nat`.

Typing Derivation

$$\dfrac{\dfrac{\dfrac{}{0 : \mathtt{Nat}}\ \textsc{T-Zero}}{\mathtt{iszero}\ 0 : \mathtt{Bool}}\ \textsc{T-IsZero} \qquad \dfrac{}{0 : \mathtt{Nat}}\ \textsc{T-Zero} \qquad \dfrac{\dfrac{}{0 : \mathtt{Nat}}\ \textsc{T-Zero}}{\mathtt{pred}\ 0 : \mathtt{Nat}}\ \textsc{T-Pred}}{\mathtt{if\ iszero\ 0\ then\ 0\ else\ pred\ 0} : \mathtt{Nat}}\ \textsc{T-If}$$

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 Bool, then v is either true or false.
  - If v is a value of type Nat, then v is a numeric value according to the grammar for nv.

v ::=		*values:*
	`true`	*true value*
	`false`	*false value*
	nv	*numeric value*
nv ::=		*numeric values:*
	0	*zero value*
	succ nv	*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 → 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