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
- succ t
- pred t
- iszero t

Evaluation $\boxed{t \longrightarrow t'}$

$$\text{if true then } t_2 \text{ else } t_3 \longrightarrow t_2 \quad \text{(E-IfTrue)}$$

$$\text{if false then } t_2 \text{ else } t_3 \longrightarrow t_3 \quad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \longrightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3} \quad \text{(E-If)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{succ } t_1 \longrightarrow \text{succ } t_1'} \quad \text{(E-Succ)}$$

$$\text{pred } 0 \longrightarrow 0 \quad \text{(E-PredZero)}$$

$$\text{pred (succ } nv_1) \longrightarrow nv_1 \quad \text{(E-PredSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{pred } t_1 \longrightarrow \text{pred } t_1'} \quad \text{(E-Pred)}$$

$$\text{iszero } 0 \longrightarrow \text{true} \quad \text{(E-IszeroZero)}$$

$$\text{iszero (succ } nv_1) \longrightarrow \text{false} \quad \text{(E-IszeroSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{iszero } t_1 \longrightarrow \text{iszero } t_1'} \quad \text{(E-IsZero)}$$

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

$\mathtt{T} ::=$ *types:*

$\quad \mathtt{Bool}$ *type of booleans*

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

$$\mathtt{true} : \mathtt{Bool} \qquad \text{(T-TRUE)}$$

$$\mathtt{false} : \mathtt{Bool} \qquad \text{(T-FALSE)}$$

$$\frac{\mathtt{t}_1 : \mathtt{Bool} \qquad \mathtt{t}_2 : \boxed{\mathtt{T}} \qquad \mathtt{t}_3 : \boxed{\mathtt{T}}}{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}} \qquad \text{(T-IF)}$$

Typing Rules for Numbers

New syntactic forms

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

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

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

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

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

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

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

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.

- case T-True: true : Bool OK?
- case T-If:

$$\frac{t1 : Bool,\ t2 : T,\ t3 : T}{\text{if } t1 \text{ then } t2 \text{ else } t3 : T} \quad \text{OK?}$$

- ...

Preservation

- **Theorem** [Preservation]:

If t : T and t → t′, then t′ : T.

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

- case T-True: true : Bool   OK?

- case T-If:

t1 : 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? □