8. Typed Arithmetic Expressions (Types and Programming Languages)

Kwanghoon Choi

Software Languages and Systems Laboratory
Chonnam National University

Week 4

A Plan

Now we augment the arithmetic PL with static types. The type system itself is nearly trivial, but it provides concepts that will recur.

8.1 Remind: terms, values, and “stuck”

Recall the syntax for arithmetic expressions:

$$\begin{array}{rcl} \mathtt{t} & ::= & \mathtt{true} \mid \mathtt{false} \mid \mathtt{if\ t\ then\ t\ else}\ t \\ & \mid & \mathtt{0} \mid \mathtt{succ\ t} \mid \mathtt{pred\ t} \mid \mathtt{iszero\ t} \end{array}$$

Evaluating a term can either result in a value

$$\begin{array}{rcl} \mathtt{v} & ::= & \mathtt{true} \mid \mathtt{false} \mid \mathtt{nv} \\ \mathtt{nv} & ::= & \mathtt{0} \mid \mathtt{succ\ nv} \end{array}$$

or else get *stuck*, for example, by reaching a term like `pred false`.

Stuck terms correspond to erroneous programs.

Types

We would like to be able to tell, without actually evaluating a term, that its evaluation will definitely not get stuck.

To do this, we need to be able to distinguish between numeric terms (whose result will be a numeric value) and boolean terms (whose result will be a boolean value).

We introduce two types for classifying terms in this way.

$$\textit{Types} \quad \mathtt{T} \quad ::= \quad \mathtt{Nat} \mid \mathtt{Bool}$$

A term `t` has type `T` :

- `true : Bool`, `0 : Nat`.
- `if true then false else true` has type `Bool`.
- `pred (succ (pred (succ 0)))` has type `Nat`.

8.2 The Typing Relation: typing rules or type inference rules

The typing relation is written as "`t : T`" (a term `t` has type `T`).

The typing rules

$$\texttt{true : Bool} \qquad \text{(T-True)}$$

$$\texttt{false : Bool} \qquad \text{(T-False)}$$

$$\frac{\texttt{t1 : Bool} \quad \texttt{t2 : T} \quad \texttt{t3 : T}}{\texttt{if t1 then t2 else t3 : T}} \qquad \text{(T-IF)}$$

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

$$\frac{\texttt{t1 : Nat}}{\texttt{succ t1 : Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\texttt{t1 : Nat}}{\texttt{pred t1 : Nat}} \qquad \text{(T-Pred)}$$

$$\frac{\texttt{t1 : Nat}}{\texttt{iszero t1 : Bool}} \qquad \text{(T-IsZero)}$$

The Typing Relation: How to read typing rules

(1) Axiom `true : Bool`

- The term `true` has type `Bool` *unconditionally*.

(2) Inference rule

$$\frac{\texttt{t1 : Nat}}{\texttt{iszero t1 : Bool}}$$

- The term `iszero t1` has type `Bool` if `t1` has type `Nat`.
- "`t1 : Nat`" above the line is called a condition or a premise, and
- "`iszero t1 : Bool`" below the line is called a conclusion.

The Typing Relation: Well-typed terms

A term `t` is typeable (or well typed)
if there is some `T` such that `t: T`.

Q. `if true then false else true : ???`

Q. `pred (succ (pred (succ 0))) : ???`

Q. `if iszero 0 then 0 else pred 0 : ???`

Q. `pred false : ???`

cf. "A term is typeable" means that a term can have some type.

The Typing Relation: typing derivation

A typing derivation is a tree of instances of the typing rules.

For example,

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

The Typing Relation: Some properties

Lemma 8.2.2 [Inversion of the typing relation]

- If `true : T` then `T = Bool`.
- If `false : T` then `T = Bool`.
- If `if t1 then t2 else t3 : T`
  then `t1 : Bool`, `t2 : T`, and `t3 : T`.
- If `0 : T` then ???
- If `succ t1 : T` then ???
- If `pred t1 : T` then ???
- If `iszero t1 : T` then ???

Q. Complete the lemma by filling in the holes ???.

Note that the inversion properityis is not always true for arbitrary type systems but it is true for the one in the arithmetic PL.

The Typing Relation: Some properties

Lemma 8.2.4 [Uniquness of Types] Each term `t` has at most one type. That is, if `t` is typable, then its type is unique.

Moreover, there is just one derivation of this typing built from the inference rules for the arithmetic expressions.

Q. Prove the lemma by structural induction on `t`.

8.3 Safety = Progress + Preservation

The most basic property of this type system or any other is *safety* (also called *soundness*):

- Well-typed terms do not "go wrong."

Terms that went wrong are what evaluate to a term who got stuck.

What we want to know is that well-typed terms do not get stuck.

We show the type safety by proving two steps, commonly known as the *progress* and *preservation* theorems.

- 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.

Safety: Progress

Progress: A well-typed term is not stuck (either it is a value or it can take a step according to the evaluation rules).

Theorem[Progress] If `t : T` then

- either `t` is a value
- or there is some `t'` with `t` $\rightarrow$ `t'`.

Q. Prove this theorem by induction on a derivation of `t : T`.

Safety: Preservation

Preservation: If a well-typed term takes a step of evaluation, then the resulting term is also well-typed.

Theorem[Preservation] If `t : T` and `t` $\rightarrow$ `t'` then `t' : T`.

Q. Prove this theorem by induction on a derivation of `t : T`.

Summary: The typed arithmetic programming language

The syntax of terms and types

Terms	`t`	::=	`v` | `if t then t else t`
		|	`0` | `succ t` | `pred t` | `iszero t`
Values	`v`	::=	`true` | `false` | `nv`
Num. values	`nv`	::=	`0` | `succ nv`

Types	`T`	::=	`Nat` | `Bool`

Summary: The typed arithmetic PL (cont.)

The (dynamic) semantics

$$\texttt{if true then t2 else t3} \rightarrow \texttt{t2} \qquad \text{(E-IFTrue)}$$

$$\texttt{if false then t2 else t3} \rightarrow \texttt{t3} \qquad \text{(E-IFFalse)}$$

$$\frac{\texttt{t1} \rightarrow \texttt{t1'}}{\texttt{if t1 then t2 else t3} \rightarrow \texttt{if t1' then t2 else t3}} \qquad \text{(E-IF)}$$

$$\frac{\texttt{t1} \rightarrow \texttt{t1'}}{\texttt{succ t1} \rightarrow \texttt{succ t1'}} \qquad \text{(E-Succ)}$$

$$\texttt{pred 0} \rightarrow \texttt{0} \qquad \text{(E-PredZero)}$$

$$\texttt{pred (succ nv)} \rightarrow \texttt{nv} \qquad \text{(E-PredSucc)}$$

$$\frac{\texttt{t1} \rightarrow \texttt{t1'}}{\texttt{pred t1} \rightarrow \texttt{pred t1'}} \qquad \text{(E-Pred)}$$

$$\texttt{iszero 0} \rightarrow \texttt{false} \qquad \text{(E-IsZeroZero)}$$

$$\texttt{iszero (succ nv)} \rightarrow \texttt{true} \qquad \text{(E-IsZeroSucc)}$$

$$\frac{\texttt{t1} \rightarrow \texttt{t1'}}{\texttt{iszero t1} \rightarrow \texttt{iszero t1'}} \qquad \text{(E-IsZero)}$$

Summary: The typed arithmetic PL (cont.)

The type system

$$\texttt{true : Bool} \qquad \text{(T-True)}$$

$$\texttt{false : Bool} \qquad \text{(T-False)}$$

$$\frac{\texttt{t1 : Bool} \quad \texttt{t2 : T} \quad \texttt{t3 : T}}{\texttt{if t1 then t2 else t3 : T}} \qquad \text{(T-IF)}$$

$$\frac{\texttt{t1 : Nat}}{\texttt{succ t1 : Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\texttt{t1 : Nat}}{\texttt{pred t1 : Nat}} \qquad \text{(T-Pred)}$$

$$\frac{\texttt{t1 : Nat}}{\texttt{iszero t1 : Bool}} \qquad \text{(T-IsZero)}$$

Summary: The arithmetic PL - Typing derivations

A typing derivation tree for

- `if iszero 0 then 0 else pred 0 : Nat`

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

Summary: The arithmetic PL - Type safety

Type safety

- If `t : T` then there is no term `t'` such that $t \to^* t'$ and `t'` is stuck.

A technique to show the type safety is to prove the progress and the preservation properties.

- Progress: If `t : T` then either `t` is a value or there exists a term `t'` such that $t \to t'$.
- Preservation: If `t : T` and $t \to t'$ then `t' : T`.