Types

Outline

1. begin with a set of terms, a set of values, and an evaluation relation
2. define a set of *types* classifying values according to their "shapes"
3. define a *typing relation* `t : T` that classifies terms according to the shape of the values that result from evaluating them
4. check that the typing relation is *sound* in the sense that,
    4.1 if `t : T` and $\texttt{t} \longrightarrow^* \texttt{v}$, then `v : T`
    4.2 if `t : T`, then evaluation of `t` will not get stuck

Review: Arithmetic Expressions – Syntax

`t ::=`		*terms*
	`true`	*constant true*
	`false`	*constant false*
	`if t then t else t`	*conditional*
	`0`	*constant zero*
	`succ t`	*successor*
	`pred t`	*predecessor*
	`iszero t`	*zero test*
`v ::=`		*values*
	`true`	*true value*
	`false`	*false value*
	`nv`	*numeric value*
`nv ::=`		*numeric values*
	`0`	*zero value*
	`succ nv`	*successor value*

Evaluation Rules

$$\texttt{if true then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_2 \qquad \text{(E-IfTrue)}$$

$$\texttt{if false then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_3 \qquad \text{(E-IfFalse)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{if } \texttt{t}_1' \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3} \qquad \text{(E-If)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{succ t}_1 \longrightarrow \texttt{succ t}_1'} \qquad \text{(E-Succ)}$$

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

$$\texttt{pred (succ nv}_1\texttt{)} \longrightarrow \texttt{nv}_1 \qquad \text{(E-PredSucc)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{pred t}_1 \longrightarrow \texttt{pred t}_1'} \qquad \text{(E-Pred)}$$

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

$$\texttt{iszero (succ nv}_1\texttt{)} \longrightarrow \texttt{false} \qquad \text{(E-IszeroSucc)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{iszero t}_1 \longrightarrow \texttt{iszero t}_1'} \qquad \text{(E-IsZero)}$$

Types

In this language, values have two possible “shapes”: they are either booleans or numbers.

`T` ::=		*types*
	`Bool`	*type of booleans*
	`Nat`	*type of numbers*

Typing Rules

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

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

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

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

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{succ } \texttt{t}_1 : \texttt{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{pred } \texttt{t}_1 : \texttt{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{iszero } \texttt{t}_1 : \texttt{Bool}} \qquad \text{(T-IsZero)}$$

Typing Derivations

Every pair $(\mathtt{t}, \mathtt{T})$ in the typing relation can be justified by a *derivation tree* built from instances of the inference rules.

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

Proofs of properties about the typing relation often proceed by induction on typing derivations.

Imprecision of Typing

Like other static program analyses, type systems are generally *imprecise*: they do not predict exactly what kind of value will be returned by every program, but just a conservative (safe) approximation.

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

Using this rule, we cannot assign a type to

```
if true then 0 else false
```

even though this term will certainly evaluate to a number.