编程语言的设计原理

Design Principles of Programming Languages

Haiyan Zhao，Di Wang

赵海燕，王迪

Peking University, Spring Term 2023

Chapter 8:
Typed Arithmetic Expressions

Types

The Typing Relation

Safety = Progress + Preservation

Review: Arithmetic Expression – 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*

Review: Arithmetic Expression – Evaluation Rules

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

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

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

Review: Arithmetic Expression – Evaluation Rules

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

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

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

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

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

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

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

Evaluation Results

- Either values

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

- Or stuckness
  - e.g, $pred\ false$

Types of Terms

- Can we tell, **without actually evaluating a term**, that the term evaluation will **not get stuck**?

- If we can 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

Types

- Values have two possible "shapes": 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 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*.

Typing Derivation

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

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

- Proofs of properties about the typing relation often proceed by induction on typing derivations.
- **Statements** are formal assertions about the typing of programs.
- **Typing rules** are implications between **statements.**
- **Derivations** are deductions based on **typing rules.**

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.

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

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

Properties of
The Typing Relation

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.

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

Typechecking Algorithm

```
typeof(t) = if t = true then Bool
            else if t = false then Bool
            else if t = if t1 then t2 else t3 then
              let T1 = typeof(t1) in
              let T2 = typeof(t2) in
              let T3 = typeof(t3) in
              if T1 = Bool and T2=T3 then T2
              else "not typable"
            else if t = 0 then Nat
            else if t = succ t1 then
              let T1 = typeof(t1) in
              if T1 = Nat then Nat else "not typable"
            else if t = pred t1 then
              let T1 = typeof(t1) in
              if T1 = Nat then Nat else "not typable"
            else if t = iszero t1 then
              let T1 = typeof(t1) in
              if T1 = Nat then Bool else "not typable"
```

Canonical Forms

- Lemma:

  1. If v is a value of type Bool, then v is either true or false.
  2. If v is a value of type Nat, then v is a numeric value.

- *Proof* :

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

For part 1, if v is true or false, the result is immediate. But v cannot be 0 or succ nv, since the inversion lemma tells us that v would then have type Nat, not Bool.

Part 2 is similar.

Uniqueness of Types

- **Theorem** [Uniqueness of Types]:

  Each term *t* has at most one type. i.e.,

  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***.

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 $t \longrightarrow t'$.

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

- *case* T-True: true : Bool OK?
- *case* T-False, T-Zero are immediate, since **t** in these cases is a value.
- *case* T-If: $t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3$

  $t_1 : \text{Bool}, \ t_2 : T, \ t_3 : T$

By the induction hypothesis, either t_1 is a value or there is some t_1' such that $t_1 \longrightarrow t_1'$.

If t_1 is a value, then the canonical forms lemma tells us that it must be either true or false, in which case either E-IFTrue or E-IFFalse applies to t.

On the other hand, if $t_1 \longrightarrow t_1'$, then, by E-IF, $t_1 \longrightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3$.

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 $t \longrightarrow t'$.

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

– *The cases for rules T-Zero, T-Succ, T-Pred, and T-IsZero are similar.*

Preservation

Theorem [Preservation]:

If $t : T$ and $t \longrightarrow t'$, then $t' : T$.

Proof: By induction on **a derivation** of t : T. Each step of the induction assumes that the desired property holds for all sub-derivations and proceed by case analysis on the final rule in the derivation.

– *case* T-IF: $t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3$ $\quad t_1 : Bool, \; t_2 : T, \; t_3 : T$

There are three evaluation rules by which and $t \longrightarrow t'$ can be derived:

E-IFTrue, E-IFFalse, and E-IF. Consider each case separately.

− *Subcase* E-IFTrue: $t_1 = true \quad t' = t_2$

Immediate, by the assumption $t_2 : T$.

E-IFFalse subcase: similar.

Preservation

Theorem [Preservation]:

If $t : T$ and $t \longrightarrow t'$, then $t' : T$.

Proof: By induction on **a derivation** of t : T. Each step of the induction assumes that the desired property holds for all sub-derivations and proceed by case analysis on the final rule in the derivation.

– *case* T-IF: $t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3$ $t_1 : Bool,\ t_2 : T,\ t_3 : T$

There are three evaluation rules by which and $t \longrightarrow t'$ can be derived: E-IFTrue, E-IFFalse, and E-IF. Consider each case separately.

– ***Subcase*** E-IF : $t_1 \longrightarrow t_1'$, $t' = \text{if } t_1' \text{ then } t_2 \text{ else } t_3$

Applying the IH to the subderivation of $t_1 : Bool$ yields $t_1' : Bool$. ombining this with the assumptions that , $t_2 : T$, and $t_3 : T$, we can apply rule T-IF to conclude that if $\text{if } t_1' \text{ then } t_2 \text{ else } t_3 : T$, that is, $t' : T$

Preservation

Theorem [Preservation]:

If $t : T$ and $t \longrightarrow t'$, then $t' : T$.

The preservation theorem is often *called subject reduction property* (or *subject evaluation property*)

Recap: Type Systems

- Very successful example of a *lightweight formal method*
- big topic in PL research
- enabling technology for all sorts of other things, e.g., language-based security
- the skeleton around which modern programming languages are designed

Homework

- Read Chapter 8.
- Do Exercises 8.3.6