编程语言的设计原理

Design Principles of Programming Languages

Zhenjiang Hu, Haiyan Zhao,

胡振江　赵海燕

Peking University, Spring, 2022

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

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

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

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

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

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

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

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

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
  – 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": 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 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.

$$\dfrac{\dfrac{\dfrac{}{\texttt{0 : Nat}}\ \text{T-ZERO}}{\texttt{iszero 0 : Bool}}\ \text{T-ISZERO} \qquad \dfrac{}{\texttt{0 : Nat}}\ \text{T-ZERO} \qquad \dfrac{\dfrac{}{\texttt{0 : Nat}}\ \text{T-ZERO}}{\texttt{pred 0 : Nat}}\ \text{T-PRED}}{\texttt{if iszero 0 then 0 else pred 0 : Nat}}\ \text{T-IF}$$

- 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.

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

Using this rule, we cannot assign a type to

$$\mathtt{if\ true\ then\ 0\ else\ false}$$

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 $\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \mathsf{R}$, then $\texttt{t}_1 : \mathsf{Bool}$, $\texttt{t}_2 : \mathsf{R}$, and $\texttt{t}_3 : \mathsf{R}$.
4. If $0 : \mathsf{R}$, then $\mathsf{R} = \mathsf{Nat}$.
5. If $\texttt{succ } \texttt{t}_1 : \mathsf{R}$, then $\mathsf{R} = \mathsf{Nat}$ and $\texttt{t}_1 : \mathsf{Nat}$.
6. If $\texttt{pred } \texttt{t}_1 : \mathsf{R}$, then $\mathsf{R} = \mathsf{Nat}$ and $\texttt{t}_1 : \mathsf{Nat}$.
7. If $\texttt{iszero } \texttt{t}_1 : \mathsf{R}$, then $\mathsf{R} = \mathsf{Bool}$ and $\texttt{t}_1 : \mathsf{Nat}$.

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.

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 ⟶ t′.

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

- *case* T-True: true : Bool OK?
- *case* T-If: : $t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3$

$$\frac{t_1 : \text{Bool},\quad t_2 : T,\quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T \ldots}$$

OK?

Preservation

- **Theorem** [Preservation]:

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

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

  – *case* T-True: t = true T = Bool true : Bool   OK?

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

  t1 : Bool, t2 : T, t3 : T

  ----------------------------   OK?

  if t1 then t2 else t3 : 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 8.3.7