CIS 500

Software Foundations

Fall 2004

4 October, 2004

Types

Type Systems

- currently, active and successful topic in PL research
- “light-weight” formal methods
- “enabling technology” for all sorts of other things, e.g. language-based security
- the “skeleton” around which modern programming languages are often designed

Approaches to Typing

- A strongly typed language prevents programs from accessing private data, corrupting memory, crashing the machine, etc.
- A weakly typed language does not.
- A statically typed language performs type-consistency checks at when programs are first entered.
- A dynamically typed language delays these checks until programs are executed.

	Weak	Strong
Dynamic		Lisp, Scheme, Perl, Python, Smalltalk
Static	C, C++	ML, ADA, Java*

*Strictly speaking, Java should be called "mostly static"

Plan

- For today, we'll go back to the simple language of arithmetic and boolean expressions and show how to give it a (very simple) type system
- On Wednesday, we'll develop a simple type system for the lambda-calculus, following TAPL Ch.9. This lecture will not be covered on the first midterm.
- We'll spend a good part of the rest of the semester adding features to this type system

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,
   (a) if `t : T` and $t \longrightarrow^* v$, then `v : T`
   (b) if `t : T`, then evaluation of `t` will not get stuck

(N.b.: we actually state #4a in a slightly more general way...)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Typing Rules

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

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

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

Typing Rules

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

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

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

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

Typing Rules

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

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

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

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

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

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

Typing Rules

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

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

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

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

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

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

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

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

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

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

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

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

Type Safety

The safety (or soundness) of this type system can be expressed by two properties:

1. Progress: A well-typed term is not stuck

   If `t : T`, then either `t` is a value or else `t` $\longrightarrow$ `t'` for some `t'`.

2. Preservation: Types are preserved by one-step evaluation

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

Typing Derivations

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

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

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

Inversion

Lemma:

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

Inversion

Lemma:

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

Proof: ...

Inversion

Lemma:

1. If `true : R`, then `R = Bool`.
2. If `false : R`, then `R = Bool`.
3. If `if t₁ then t₂ else t₃ : R`, then `t₁ : Bool`, `t₂ : R`, and `t₃ : R`.
4. If `0 : R`, then `R = Nat`.
5. If `succ t₁ : R`, then `R = Nat` and `t₁ : Nat`.
6. If `pred t₁ : R`, then `R = Nat` and `t₁ : Nat`.
7. If `iszero t₁ : R`, then `R = Bool` and `t₁ : Nat`.

Proof: ...

This leads directly to a recursive algorithm for calculating the type of a term...

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

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: ...

Progress

Theorem: 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 $t \longrightarrow t'$.

Progress

Theorem: 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:

Progress

Theorem: 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`.

Progress

Theorem: Suppose $\mathtt{t}$ is a well-typed term (that is, $\mathtt{t : T}$ for some $\mathtt{T}$). Then either $\mathtt{t}$ is a value or else there is some $\mathtt{t}'$ with $\mathtt{t} \longrightarrow \mathtt{t}'$.

Proof: By induction on a derivation of $\mathtt{t : T}$.

The T-True, T-False, and T-Zero cases are immediate, since $\mathtt{t}$ in these cases is a value.

Progress

Theorem: Suppose $\mathtt{t}$ is a well-typed term (that is, $\mathtt{t : T}$ for some $\mathtt{T}$). Then either $\mathtt{t}$ is a value or else there is some $\mathtt{t}'$ with $\mathtt{t} \longrightarrow \mathtt{t}'$.

Proof: By induction on a derivation of $\mathtt{t : T}$.

The T-True, T-False, and T-Zero cases are immediate, since $\mathtt{t}$ in these cases is a value.

Case T-If: $\mathtt{t} = \mathtt{if}\ \mathtt{t_1}\ \mathtt{then}\ \mathtt{t_2}\ \mathtt{else}\ \mathtt{t_3}$

$\mathtt{t_1 : Bool} \qquad \mathtt{t_2 : T} \qquad \mathtt{t_3 : T}$

Progress

Theorem: Suppose $\mathtt{t}$ is a well-typed term (that is, $\mathtt{t : T}$ for some $\mathtt{T}$). Then either $\mathtt{t}$ is a value or else there is some $\mathtt{t'}$ with $\mathtt{t} \longrightarrow \mathtt{t'}$.

Proof: By induction on a derivation of $\mathtt{t : T}$.

The T-True, T-False, and T-Zero cases are immediate, since $\mathtt{t}$ in these cases is a value.

Case T-If: $\mathtt{t} = \mathtt{if}\ \mathtt{t_1}\ \mathtt{then}\ \mathtt{t_2}\ \mathtt{else}\ \mathtt{t_3}$

$\mathtt{t_1 : Bool} \qquad \mathtt{t_2 : T} \qquad \mathtt{t_3 : T}$

By the induction hypothesis, either $\mathtt{t_1}$ is a value or else there is some $\mathtt{t_1'}$ such that $\mathtt{t_1} \longrightarrow \mathtt{t_1'}$. If $\mathtt{t_1}$ 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 $\mathtt{t}$. On the other hand, if $\mathtt{t_1} \longrightarrow \mathtt{t_1'}$, then, by E-If, $\mathtt{t} \longrightarrow \mathtt{if}\ \mathtt{t_1'}\ \mathtt{then}\ \mathtt{t_2}\ \mathtt{else}\ \mathtt{t_3}$.

Preservation

Theorem: If $\mathtt{t : T}$ and $\mathtt{t} \longrightarrow \mathtt{t'}$, then $\mathtt{t' : T}$.

Preservation

Theorem: If $\mathtt{t : T}$ and $\mathtt{t} \longrightarrow \mathtt{t'}$, then $\mathtt{t' : T}$.

Proof: ...