CIS 500

Software Foundations

Fall 2005

19 October, 2005

Midterm Exam

Midterm Exam

- Exam solutions on web page.
- Look at your exam in Cheryl Hickey's office.
- Submit regrade request (in writing) before October 26.
- You can pick up your exam from Cheryl after October 26.

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
- Next week, we'll develop a simple type system for the lambda-calculus, following TAPL Ch.9.
- 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, if `t : T`, then evaluation of `t` will not get stuck

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Typing Rules

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

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

Typing Rules

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

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

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

Typing Rules

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

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

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

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

Typing Derivations

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

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

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

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.

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

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

Type Safety Theorem: If $\mathtt{t}:\mathtt{T}$ and $\mathtt{t} \longrightarrow^* \mathtt{t'}$ and $\mathtt{t'} \not\longrightarrow$ then $\mathtt{t'}$ is a value.

We usually prove type safety by showing the following two properties:

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

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

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

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

Inversion

Lemma:

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

Inversion

Lemma:

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

Proof: ...

Inversion

Lemma:

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

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

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 `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 T-True, T-False, and T-Zero cases are immediate, since `t` in these cases is a value.

Progress

Theorem: Suppose $\mathtt{t}$ is a well-typed term (that is, $\mathtt{t} : \mathtt{T}$ 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 $\mathtt{t} : \mathtt{T}$.

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

Case T-If: $\mathtt{t} = \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$

$\mathtt{t}_1 : \mathtt{Bool} \qquad \mathtt{t}_2 : \mathtt{T} \qquad \mathtt{t}_3 : \mathtt{T}$

Progress

Theorem: Suppose $\mathtt{t}$ is a well-typed term (that is, $\mathtt{t} : \mathtt{T}$ 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 $\mathtt{t} : \mathtt{T}$.

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

Case T-If: $\mathtt{t} = \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$

$\mathtt{t}_1 : \mathtt{Bool} \qquad \mathtt{t}_2 : \mathtt{T} \qquad \mathtt{t}_3 : \mathtt{T}$

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 $\mathtt{true}$ or $\mathtt{false}$, 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 $\mathtt{t} : \mathtt{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\mathtt{t}' : \mathtt{T}$.

Preservation

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

Proof: ...