CIS 500

Software Foundations

Fall 2004

18 October

Announcements

- Homework 5 will be posted today–due in one week.
- Normal office hours/recitation schedule.
- No good date for midterm. Looks like 12/20 1:30-3:30 is the best.

Midterm 1 results

- Did you lose a watch?
- Pick up exams from Cheryl Hickey.
- Statistics
  - Max score: 78
  - Min score: 24
  - Average: 57
  - Median: 60
  - Std dev: 12
- Regrade policy: Must submit request to me in writing within 2 weeks (by Nov 1).

Extra Credit

Course grades can be improved after the semester ends in two ways:

1. A 1/3 letter grade improvement can be obtained by doing a substantial extra credit project ($\sim$30 hours work) during the Spring semester.
2. Larger grade improvements can (only) be obtained by sitting in on the course next year and turning in all homeworks and exams.

Plans for today

- Simply-typed lambda-calculus
- Extensions of those simple types
- Connection to untyped calculus
- Connection to logic

The Simply Typed Lambda-Calculus

Lambda-calculus with booleans

```
t ::=                          terms
        x                        variable
        λx:T.t                   abstraction
        t t                      application
        true                     constant true
        false                    constant false
        if t then t else t       conditional
v ::=                          values
        λx.t                     abstraction value
        true                     true value
        false                    false value
T ::=                          types
        Bool                     type of booleans
        T→T                      types of functions
```

Operational Semantics

$$(\lambda \mathtt{x}:\mathtt{T}.\mathtt{t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \qquad \text{(E-AppAbs)}$$

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

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

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1\ t_2} \longrightarrow \mathtt{t_1'\ t_2}} \qquad \text{(E-App1)}$$

$$\frac{\mathtt{t_2} \longrightarrow \mathtt{t_2'}}{\mathtt{v_1\ t_2} \longrightarrow \mathtt{v_1\ t_2'}} \qquad \text{(E-App2)}$$

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

Typing rules

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

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

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

$$\frac{\texttt{x}:\texttt{T} \in \Gamma}{\Gamma \vdash \texttt{x} : \texttt{T}} \qquad \text{(T-Var)}$$

$$\frac{\Gamma, \texttt{x}:\texttt{T}_1 \vdash \texttt{t}_2 : \texttt{T}_2}{\Gamma \vdash \lambda \texttt{x}:\texttt{T}_1 . \texttt{t}_2 : \texttt{T}_1 \to \texttt{T}_2} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_{11} \to \texttt{T}_{12} \qquad \Gamma \vdash \texttt{t}_2 : \texttt{T}_{11}}{\Gamma \vdash \texttt{t}_1 \ \texttt{t}_2 : \texttt{T}_{12}} \qquad \text{(T-App)}$$

Properties of $\lambda_{\rightarrow}$

As before, the fundamental property of the type system we have just defined is soundness with respect to the operational semantics.

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

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

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

   If $\Gamma \vdash \texttt{t} : \texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\Gamma \vdash \texttt{t}' : \texttt{T}$.

Proving Preservation

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

Proof: By induction

Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]

Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]

Case T-App: Given $t = t_1\ t_2$

$\Gamma \vdash t_1 : T_{11} \rightarrow T_{12}$

$\Gamma \vdash t_2 : T_{11}$

$T = T_{12}$

Show $\Gamma \vdash t' : T_{12}$

Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]

Case T-APP: Given $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$

$\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12}$

$\Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$

$\mathtt{T} = \mathtt{T}_{12}$

Show $\Gamma \vdash \mathtt{t}' : \mathtt{T}_{12}$

By the inversion lemma for evaluation, there are three subcases...

Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]

Case T-App: Given $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$

$\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12}$

$\Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$

$\mathtt{T} = \mathtt{T}_{12}$

Show $\Gamma \vdash \mathtt{t}' : \mathtt{T}_{12}$

By the inversion lemma for evaluation, there are three subcases...

Subcase: $\mathtt{t}_1 = \lambda \mathtt{x}{:}\mathtt{T}_{11}.\ \mathtt{t}_{12}$

$\mathtt{t}_2$ a value $\mathtt{v}_2$

$\mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$

Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]

Case T-App: Given $t = t_1\ t_2$

$\Gamma \vdash t_1 : T_{11} \to T_{12}$

$\Gamma \vdash t_2 : T_{11}$

$T = T_{12}$

Show $\Gamma \vdash t' : T_{12}$

By the inversion lemma for evaluation, there are three subcases...

Subcase: $t_1 = \lambda x{:}T_{11}.\ t_{12}$

t_2 a value v_2

$t' = [x \mapsto v_2]t_{12}$

Uh oh.

The “Substitution Lemma”

Lemma: Types are preserved under substitition.

If Γ, $\mathtt{x}:\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.

The "Substitution Lemma"

Lemma: Types are preserved under substititition.

If $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.

Proof: ...

On to real programming languages...

The `Unit` type

`t`	::=	...	*terms*
		`unit`	*constant* `unit`
`v`	::=	...	*values*
		`unit`	*constant* `unit`
`T`	::=	...	*types*
		`Unit`	*unit type*

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\Gamma \vdash \texttt{unit} : \texttt{Unit} \qquad \text{(T-Unit)}$$

Sequencing

$$\begin{array}{lll} \texttt{t} & ::= & \ldots \qquad\qquad\qquad\qquad \textit{terms} \\ & & \texttt{t}_1\texttt{;t}_2 \end{array}$$

Sequencing

$$\mathtt{t} ::= \ldots \qquad \textit{terms}$$

$$\mathtt{t_1;t_2}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1;t_2} \longrightarrow \mathtt{t_1';t_2}} \qquad \text{(E-Seq)}$$

$$\mathtt{unit;t_2} \longrightarrow \mathtt{t_2} \qquad \text{(E-SeqNext)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{Unit} \qquad \Gamma \vdash \mathtt{t_2} : \mathtt{T_2}}{\Gamma \vdash \mathtt{t_1;t_2} : \mathtt{T_2}} \qquad \text{(T-Seq)}$$

Derived forms

- Syntatic sugar
- Internal language vs. external (surface) language

Sequencing as a derived form

$$\texttt{t}_1\texttt{;t}_2 \quad \stackrel{\textbf{def}}{=} \quad (\lambda\texttt{x:Unit.t}_2)\ \texttt{t}_1$$

$$\text{where } \texttt{x} \notin FV(\texttt{t}_2)$$

Equivalence of the two definitions

- $\mathtt{t} \longrightarrow_{\mathtt{E}} \mathtt{t}'$ iff $e(\mathtt{t}) \longrightarrow_{\mathtt{I}} e(\mathtt{t}')$
- $\Gamma \vdash^{\mathtt{E}} \mathtt{t} : \mathtt{T}$ iff $\Gamma \vdash^{\mathtt{I}} e(\mathtt{t}) : \mathtt{T}$

Ascription

New syntactic forms

$$\texttt{t} ::= \ldots \qquad \textit{terms}$$
$$\texttt{t as T} \qquad \textit{ascription}$$

New evaluation rules $\boxed{\texttt{t} \longrightarrow \texttt{t}'}$

$$\texttt{v}_1 \texttt{ as T} \longrightarrow \texttt{v}_1 \qquad \text{(E-Ascribe)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{t}_1 \texttt{ as T} \longrightarrow \texttt{t}_1' \texttt{ as T}} \qquad \text{(E-Ascribe1)}$$

New typing rules $\boxed{\Gamma \vdash \texttt{t} : \texttt{T}}$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}}{\Gamma \vdash \texttt{t}_1 \texttt{ as T} : \texttt{T}} \qquad \text{(T-Ascribe)}$$

Ascription as a derived form

$$\texttt{t as T} \stackrel{\textbf{def}}{=} (\lambda\texttt{x:T. x) t}$$

Let-bindings

New syntactic forms

$$\texttt{t} ::= \ldots \qquad \textit{terms}$$

$$\qquad \texttt{let x=t in t} \qquad \textit{let binding}$$

New evaluation rules $\boxed{\texttt{t} \longrightarrow \texttt{t}'}$

$$\texttt{let x=v}_1 \texttt{ in t}_2 \longrightarrow [\texttt{x} \mapsto \texttt{v}_1]\texttt{t}_2 \qquad \text{(E-LetV)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{let x=t}_1 \texttt{ in t}_2 \longrightarrow \texttt{let x=t}_1' \texttt{ in t}_2} \qquad \text{(E-Let)}$$

New typing rules $\boxed{\Gamma \vdash \texttt{t : T}}$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1 \qquad \Gamma, \texttt{x:T}_1 \vdash \texttt{t}_2 : \texttt{T}_2}{\Gamma \vdash \texttt{let x=t}_1 \texttt{ in t}_2 : \texttt{T}_2} \qquad \text{(T-Let)}$$

Pairs

t	::=	...	*terms*
		{t,t}	*pair*
		t.1	*first projection*
		t.2	*second projection*
v	::=	...	*values*
		{v,v}	*pair value*
T	::=	...	*types*
		$T_1 \times T_2$	*product type*

Evaluation rules for pairs

$$\{v_1, v_2\}.1 \longrightarrow v_1 \qquad \text{(E-PairBeta1)}$$

$$\{v_1, v_2\}.2 \longrightarrow v_2 \qquad \text{(E-PairBeta2)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.1 \longrightarrow t_1'.1} \qquad \text{(E-Proj1)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.2 \longrightarrow t_1'.2} \qquad \text{(E-Proj2)}$$

$$\frac{t_1 \longrightarrow t_1'}{\{t_1, t_2\} \longrightarrow \{t_1', t_2\}} \qquad \text{(E-Pair1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\{v_1, t_2\} \longrightarrow \{v_1, t_2'\}} \qquad \text{(E-Pair2)}$$

Typing rules for pairs

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \{\mathtt{t}_1,\mathtt{t}_2\} : \mathtt{T}_1 \times \mathtt{T}_2} \quad \text{(T-Pair)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \times \mathtt{T}_{12}}{\Gamma \vdash \mathtt{t}_1.1 : \mathtt{T}_{11}} \quad \text{(T-Proj1)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \times \mathtt{T}_{12}}{\Gamma \vdash \mathtt{t}_1.2 : \mathtt{T}_{12}} \quad \text{(T-Proj2)}$$

Tuples

t	::=	...	*terms*
		$\{t_i\ ^{i\in 1..n}\}$	*tuple*
		t.i	*projection*
v	::=	...	*values*
		$\{v_i\ ^{i\in 1..n}\}$	*tuple value*
T	::=	...	*types*
		$\{T_i\ ^{i\in 1..n}\}$	*tuple type*

Evaluation rules for tuples

$$\{v_i{}^{\,i\in 1..n}\}.j \longrightarrow v_j \qquad \text{(E-ProjTuple)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.i \longrightarrow t_1'.i} \qquad \text{(E-Proj)}$$

$$\frac{t_j \longrightarrow t_j'}{\{v_i{}^{\,i\in 1..j-1}, t_j, t_k{}^{\,k\in j+1..n}\} \longrightarrow \{v_i{}^{\,i\in 1..j-1}, t_j', t_k{}^{\,k\in j+1..n}\}} \qquad \text{(E-Tuple)}$$

Typing rules for tuples

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{t_i{}^{\,i\in 1..n}\} : \{T_i{}^{\,i\in 1..n}\}} \qquad \text{(T-Tuple)}$$

$$\frac{\Gamma \vdash t_1 : \{T_i{}^{\,i\in 1..n}\}}{\Gamma \vdash t_1.j : T_j} \qquad \text{(T-Proj)}$$

Records

t	::=	...	*terms*
		$\{l_i = t_i\ ^{i \in 1..n}\}$	*record*
		t.l	*projection*
v	::=	...	*values*
		$\{l_i = v_i\ ^{i \in 1..n}\}$	*record value*
T	::=	...	*types*
		$\{l_i : T_i\ ^{i \in 1..n}\}$	*type of records*

Evaluation rules for records

$$\{\mathtt{l}_i\mathtt{=v}_i\ ^{i\in 1..n}\}\mathtt{.l}_j \longrightarrow \mathtt{v}_j \qquad \text{(E-ProjRcd)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\mathtt{.l} \longrightarrow \mathtt{t}_1'\mathtt{.l}} \qquad \text{(E-Proj)}$$

$$\frac{\mathtt{t}_j \longrightarrow \mathtt{t}_j'}{\begin{array}{c}\{\mathtt{l}_i\mathtt{=v}_i\ ^{i\in 1..j-1}\mathtt{,l}_j\mathtt{=t}_j\mathtt{,l}_k\mathtt{=t}_k\ ^{k\in j+1..n}\} \\ \longrightarrow \{\mathtt{l}_i\mathtt{=v}_i\ ^{i\in 1..j-1}\mathtt{,l}_j\mathtt{=t}_j'\mathtt{,l}_k\mathtt{=t}_k\ ^{k\in j+1..n}\}\end{array}} \qquad \text{(E-Rcd)}$$

Typing rules for records

$$\frac{\text{for each } i \quad \Gamma \vdash \mathtt{t}_i : \mathtt{T}_i}{\Gamma \vdash \{\mathtt{l}_i\mathtt{=t}_i\ ^{i\in 1..n}\} : \{\mathtt{l}_i\mathtt{:T}_i\ ^{i\in 1..n}\}} \qquad \text{(T-Rcd)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \{\mathtt{l}_i\mathtt{:T}_i\ ^{i\in 1..n}\}}{\Gamma \vdash \mathtt{t}_1\mathtt{.l}_j : \mathtt{T}_j} \qquad \text{(T-Proj)}$$

Intro vs. elim forms

An introduction form for a given type gives us a way of constructing elements of this type.

An elimination form for a type gives us a way of using elements of this type.

What typing rules are introduction forms? What are elimination forms?

Connection with untyped lambda calculus

Erasure

$$
\begin{aligned}
erase(\mathtt{x}) &= \mathtt{x} \\
erase(\lambda\mathtt{x}{:}\mathtt{T_1}.\ \mathtt{t_2}) &= \lambda\mathtt{x}.\ erase(\mathtt{t_2}) \\
erase(\mathtt{t_1}\ \mathtt{t_2}) &= erase(\mathtt{t_1})\ erase(\mathtt{t_2})
\end{aligned}
$$

Theorem:

1. If $\mathtt{t} \longrightarrow \mathtt{t'}$ then $erase(\mathtt{t}) \longrightarrow erase(\mathtt{t'})$.
2. If $erase(\mathtt{t}) \longrightarrow \mathtt{m'}$, then there is a simply typed term $\mathtt{t'}$ such that $\mathtt{t} \longrightarrow \mathtt{t'}$ and $erase(\mathtt{t'}) = \mathtt{m'}$.

Typability

An untyped λ-term $\mathtt{m}$ is said to be typable if there is some term $\mathtt{t}$ in the simply typed lambda-calculus, some type $\mathtt{T}$, and some context Γ such that $erase(\mathtt{t}) = \mathtt{m}$ and $\Gamma \vdash \mathtt{t} : \mathtt{T}$.

Cf. type reconstruction in OCaml.

Connection to Logic

The Curry-Howard Correspondence

In constructive logics, a proof of $\mathbf{P}$ must provide evidence for $\mathbf{P}$.

- “law of the excluded middle” — $\mathbf{P} \vee \neg\mathbf{P}$ — not recognized.

A proof of $\mathbf{P} \wedge \mathbf{Q}$ is a pair of evidence for $\mathbf{P}$ and evidence for $\mathbf{Q}$.

A proof of $\mathbf{P} \supset \mathbf{Q}$ is a procedure for transforming evidence for $\mathbf{P}$ into evidence for $\mathbf{Q}$.

Propositions as Types

Logic	Programming languages
propositions	types
proposition $\mathbf{P} \supset \mathbf{Q}$	type `P→Q`
proposition $\mathbf{P} \wedge \mathbf{Q}$	type `P` $\times$ `Q`
proposition **true**	type `unit`
proof of proposition $\mathbf{P}$	term `t` of type `P`
proposition $\mathbf{P}$ is provable	type `P` is inhabited (by some term)

Propositions as Types

Logic	Programming languages
propositions	types
proposition $\mathbf{P} \supset \mathbf{Q}$	type `P→Q`
proposition $\mathbf{P} \wedge \mathbf{Q}$	type `P` $\times$ `Q`
proposition **true**	type **`unit`**
proof of proposition $\mathbf{P}$	term `t` of type `P`
proposition $\mathbf{P}$ is provable	type `P` is inhabited (by some term)
	evaluation

Propositions as Types

Logic	Programming languages
propositions	types
proposition $\mathbf{P} \supset \mathbf{Q}$	type P→Q
proposition $\mathbf{P} \wedge \mathbf{Q}$	type P × Q
proposition **true**	type **unit**
proof of proposition $\mathbf{P}$	term t of type P
proposition $\mathbf{P}$ is provable	type P is inhabited (by some term)
proof simplification (a.k.a. "cut elimination")	evaluation