Foundations of Software
Fall 2023

Week 6

Plan

PREVIOUSLY:

1. type safety as *progress* and *preservation*
2. typed arithmetic expressions
3. simply typed lambda calculus (STLC)

TODAY:

1. Equivalence of lambda terms
2. Preservation for STLC
3. Extensions to STLC

NEXT: state, exceptions
NEXT: polymorphic (not so simple) typing

Preservation for STLC

Preservation for STLC

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

Proof: By induction

Preservation for STLC

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??

Preservation for STLC

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

Proof: By induction on typing derivations.

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

Preservation for STLC

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

Proof: By induction on typing derivations.

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

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

Preservation for STLC

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

Proof: By induction on typing derivations.

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

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

Preservation for STLC

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

Proof: By induction on typing derivations.

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.

That is, if $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.

The "Substitution Lemma"

Lemma: Types are preserved under substitition.

That is, if $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.

Proof: ...

Weakening and Permutation

Two other lemmas will be useful.

Weakening tells us that we can *add assumptions* to the context without losing any true typing statements.

Lemma: If $\Gamma \vdash t : T$ and $x \notin dom(\Gamma)$, then $\Gamma, x{:}S \vdash t : T$.

Weakening and Permutation

Two other lemmas will be useful.

Weakening tells us that we can *add assumptions* to the context without losing any true typing statements.

Lemma: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{x} \notin dom(\Gamma)$, then $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$.

Permutation tells us that the order of assumptions in (the list) Γ does not matter.

Lemma: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and Δ is a permutation of Γ, then $\Delta \vdash \mathtt{t} : \mathtt{T}$.

Weakening and Permutation

Two other lemmas will be useful.

Weakening tells us that we can *add assumptions* to the context without losing any true typing statements.

Lemma: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{x} \notin dom(\Gamma)$, then $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$.

Moreover, the latter derivation has the same depth as the former.

Permutation tells us that the order of assumptions in (the list) Γ does not matter.

Lemma: If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and Δ is a permutation of Γ, then $\Delta \vdash \mathtt{t} : \mathtt{T}$.

Moreover, the latter derivation has the same depth as the former.

The "Substitution Lemma"

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

I.e., "Types are preserved under substitition."

The "Substitution Lemma"

Lemma: 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: By induction on the derivation of $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$. Proceed by cases on the final typing rule used in the derivation.

The "Substitution Lemma"

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

Proof: By induction on the derivation of $\Gamma, x{:}S \vdash t : T$. Proceed by cases on the final typing rule used in the derivation.

The "Substitution Lemma"

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

Proof: By induction on the derivation of $\Gamma, x{:}S \vdash t : T$. Proceed by cases on the final typing rule used in the derivation.

Case T-App: $t = t_1\ t_2$
$\Gamma, x{:}S \vdash t_1 : T_2 \to T_1$
$\Gamma, x{:}S \vdash t_2 : T_2$
$T = T_1$

By the induction hypothesis, $\Gamma \vdash [x \mapsto s]t_1 : T_2 \to T_1$ and $\Gamma \vdash [x \mapsto s]t_2 : T_2$. By T-App, $\Gamma \vdash [x \mapsto s]t_1\ [x \mapsto s]t_2 : T$, i.e., $\Gamma \vdash [x \mapsto s](t_1\ t_2) : T$.

The "Substitution Lemma"

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

Proof: By induction on the derivation of $\Gamma, x{:}S \vdash t : T$. Proceed by cases on the final typing rule used in the derivation.

Case T-Var: $t = z$
with $z{:}T \in (\Gamma, x{:}S)$

There are two sub-cases to consider, depending on whether z is x or another variable. If $z = x$, then $[x \mapsto s]z = s$. The required result is then $\Gamma \vdash s : S$, which is among the assumptions of the lemma. Otherwise, $[x \mapsto s]z = z$, and the desired result is immediate.

The "Substitution Lemma"

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

Proof: By induction on the derivation of $\Gamma, x{:}S \vdash t : T$. Proceed by cases on the final typing rule used in the derivation.

Case T-Abs: $t = \lambda y{:}T_2.t_1$ $\quad T = T_2 \to T_1$
$\Gamma, x{:}S, y{:}T_2 \vdash t_1 : T_1$

By our conventions on choice of bound variable names, we may assume $x \neq y$ and $y \notin FV(s)$. Using *permutation* on the given subderivation, we obtain $\Gamma, y{:}T_2, x{:}S \vdash t_1 : T_1$. Using *weakening* on the other given derivation ($\Gamma \vdash s : S$), we obtain $\Gamma, y{:}T_2 \vdash s : S$. Now, by the induction hypothesis, $\Gamma, y{:}T_2 \vdash [x \mapsto s]t_1 : T_1$. By T-Abs, $\Gamma \vdash \lambda y{:}T_2.\ [x \mapsto s]t_1 : T_2 \to T_1$, i.e. (by the definition of substitution), $\Gamma \vdash [x \mapsto s]\lambda y{:}T_2.\ t_1 : T_2 \to T_1$.

Summary: Preservation

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

Lemmas to prove:

- Weakening
- Permutation
- Substitution preserves types
- Reduction preserves types (i.e., preservation)

Review: Type Systems

To define and verify a type system, you must:

1. Define types
2. Specify typing rules
3. Prove soundness: *progress* and *preservation*

Two Typing Topics

Erasure

$$
\begin{array}{lcl}
\mathit{erase}(\mathtt{x}) & = & \mathtt{x} \\
\mathit{erase}(\lambda\mathtt{x}{:}\mathtt{T}_1.\ \mathtt{t}_2) & = & \lambda\mathtt{x}.\ \mathit{erase}(\mathtt{t}_2) \\
\mathit{erase}(\mathtt{t}_1\ \mathtt{t}_2) & = & \mathit{erase}(\mathtt{t}_1)\ \mathit{erase}(\mathtt{t}_2)
\end{array}
$$

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.

The Curry-Howard Correspondence

In *constructive logics*, a proof of P must provide *evidence* for P.

- "law of the excluded middle" — $P \vee \neg P$ — not recognized.

A proof of $P \wedge Q$ is a *pair* of evidence for P and evidence for Q.

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

Propositions as Types

Logic	Programming languages
propositions	types
proposition $P \supset Q$	type `P->Q`
proposition $P \wedge Q$	type `P` $\times$ `Q`
proof of proposition P	term `t` of type `P`
proposition P is provable	type `P` is inhabited (by some term)

Propositions as Types

Logic	Programming languages
propositions	types
proposition $P \supset Q$	type `P->Q`
proposition $P \wedge Q$	type `P` $\times$ `Q`
proof of proposition P	term `t` of type `P`
proposition P is provable	type `P` is inhabited (by some term)
proof simplification (a.k.a. "cut elimination")	evaluation

Proving things with Curry-Howard (1)

Given the logic formula

$$(A \wedge B) \supset A$$

Clicker question: Prove that it is true by giving a well-typed term of the appropriate type.

Use \ to note a λ and * to note a $\times$.

URL: `ttpoll.eu`
Session ID: `cs452`

Proving things with Curry-Howard (2)

Given the logic formula

$$((A \wedge B) \supset C) \supset (A \supset (B \supset C))$$

Clicker question: Prove that it is true by giving a well-typed term of the appropriate type.

Use \ to note a λ and * to note a $\times$.

URL: `ttpoll.eu`
Session ID: `cs452`

Extensions to STLC

Base types

Up to now, we've formulated "base types" (e.g. `Nat`) by adding them to the syntax of types, extending the syntax of terms with associated constants (`0`) and operators (`succ`, etc.) and adding appropriate typing and evaluation rules. We can do this for as many base types as we like.

For more theoretical discussions (as opposed to programming) we can often ignore the term-level inhabitants of base types, and just treat these types as uninterpreted constants.
E.g., suppose `B` and `C` are some base types. Then we can ask (without knowing anything more about `B` or `C`) whether there are any types `S` and `T` such that the term

```
(λf:S. λg:T. f g) (λx:B. x)
```

is well typed.

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

t	::=	...	*terms*
		$t_1;t_2$	

Sequencing

t	::=	...	*terms*
		$t_1;t_2$	

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

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

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

Derived forms

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

Sequencing as a derived form

$$\mathtt{t_1;t_2} \stackrel{\text{def}}{=} (\lambda \mathtt{x}{:}\mathtt{Unit.t_2})\ \mathtt{t_1} \quad \text{where } \mathtt{x} \notin FV(\mathtt{t_2})$$

Equivalence of the two definitions

[board]

Ascription

New syntactic forms

$\mathtt{t}$ ::= ... *terms*

$\mathtt{t\ as\ T}$ *ascription*

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

$$\mathtt{v_1\ as\ T} \longrightarrow \mathtt{v_1} \qquad \text{(E-Ascribe)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1\ as\ T} \longrightarrow \mathtt{t_1'\ as\ T}} \qquad \text{(E-Ascribe1)}$$

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

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T}}{\Gamma \vdash \mathtt{t_1\ as\ T} : \mathtt{T}} \qquad \text{(T-Ascribe)}$$

Ascription as a derived form

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

Let-bindings

New syntactic forms

`t`	::=	...	*terms*
		`let x=t in t`	*let binding*

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

$$\texttt{let x=}v_1\texttt{ in }t_2 \longrightarrow [x \mapsto v_1]t_2 \qquad \text{(E-LetV)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{let x=}t_1\texttt{ in }t_2 \longrightarrow \texttt{let x=}t_1'\texttt{ in }t_2} \qquad \text{(E-Let)}$$

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

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \texttt{let x=}t_1\texttt{ in }t_2 : T_2} \qquad \text{(T-Let)}$$

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

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2} \qquad \text{(T-Pair)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.1 : T_{11}} \qquad \text{(T-Proj1)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.2 : T_{12}} \qquad \text{(T-Proj2)}$$

Tuples

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

Evaluation rules for tuples

$$\{\texttt{v}_i\ ^{i\in 1..n}\}\texttt{.j} \longrightarrow \texttt{v}_j \qquad \text{(E-ProjTuple)}$$

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

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

Typing rules for tuples

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

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