CIS 500
Software Foundations
Fall 2004
6 October
Midterm 1 is next Wednesday

- Today’s lecture will not be covered by the midterm.
- Next Monday, review class.
- Old exams and review questions on webpage.
- No recitation sections next week.
- New office hours next week, watch newsgroup for details.
Plans

Where we’ve been:

- Inductive definitions
  - abstract syntax
  - inference rules
- Proofs by structural induction
- Operational semantics
- The lambda-calculus
- Typing rules and type soundness

CIS 500, 6 October
Plans

Where we’ve been:
- Inductive definitions
  - abstract syntax
  - inference rules
- Proofs by structural induction
- Operational semantics
- The lambda-calculus
- Typing rules and type soundness

Where we’re going:
- “Simple types” for the lambda-calculus
- Formalizing more features of real-world languages (records, datatypes, references, exceptions, etc.)
- Subtyping
- Objects
The Simply Typed Lambda-Calculus
Lambda-calculus with booleans

\[ t ::= \]
\[
  \text{terms} \\
  x \\
  \lambda x.t \\
  t \ t \\
  \text{true} \\
  \text{false} \\
  \text{if} \ t \ \text{then} \ t \ \text{else} \ t \\
\]

\[ v ::= \]
\[
  \text{values} \\
  \lambda x.t \\
  \text{true} \\
  \text{false} \\
\]

\[ \text{variable} \]
\[ \text{abstraction} \]
\[ \text{application} \]
\[ \text{constant true} \]
\[ \text{constant false} \]
\[ \text{conditional} \]
\[ \text{abstraction value} \]
\[ \text{true value} \]
\[ \text{false value} \]
“Simple Types”

\[
T ::= \quad \text{types}
\]

\[
\begin{align*}
\text{Bool} & \quad \text{type of booleans} \\
T \to T & \quad \text{types of functions}
\end{align*}
\]
Typing rules

true : Bool \hspace{1cm} (T-\text{TRUE})

false : Bool \hspace{1cm} (T-\text{FALSE})

\[
\frac{t_1 : \text{Bool} \quad t_2 : \text{T} \quad t_3 : \text{T}}{	ext{if } t_1 \text{ then } t_2 \text{ else } t_3 : \text{T}} \quad (T-\text{IF})
\]
Typing rules

true : Bool \hspace{1cm} (T-TRUE)

false : Bool \hspace{1cm} (T-FALSE)

\[
\begin{array}{c}
t_1 : \text{Bool} \\
t_2 : T \\
t_3 : T
\end{array}
\]

\[\frac{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T}{x : T} \hspace{1cm} (T-\text{IF})\]

\[\frac{t_1 : T_{11} \neq T_{12}}{t_2 : T_{12}} \hspace{1cm} (T-\text{APP})\]

\[\frac{t_1 \neq t_2}{t_1 : T_{11} \neq T_{12}} \hspace{1cm} (T-\text{ABS})\]
Typing rules

true : Bool \hspace{1cm} (T-TRUE)
false : Bool \hspace{1cm} (T-FALSE)

\[ \frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \hspace{1cm} (T-IF) \]

\[ \frac{x : T \in \Gamma}{\Gamma \vdash x : T} \hspace{1cm} (T-VAR) \]
Typing rules

\( \Gamma \vdash \text{true} : \text{Bool} \) \hspace{1cm} (T-TRUE)

\( \Gamma \vdash \text{false} : \text{Bool} \) \hspace{1cm} (T-FALSE)

\[
\Gamma \vdash t_1 : \text{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T
\]

\( \Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T \) \hspace{1cm} (T-IF)

\[
\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T
\]

\( x : T \in \Gamma \)

\[
\Gamma \vdash x : T
\]
Typing rules

\[
\begin{align*}
\Gamma & \vdash \text{true} : \text{Bool} & \quad \text{(T-True)} \\
\Gamma & \vdash \text{false} : \text{Bool} & \quad \text{(T-False)} \\
\Gamma & \vdash t_1 : \text{Bool} \quad \Gamma & \vdash t_2 : T \quad \Gamma & \vdash t_3 : T \\
& \quad \Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T & \quad \text{(T-If)} \\
x & : T \in \Gamma \\
& \quad \Gamma \vdash x : T & \quad \text{(T-Var)} \\
\Gamma, x : T_1 & \vdash t_2 : T_2 \\
& \quad \Gamma \vdash \lambda x : T_1. t_2 : T_1 \rightarrow T_2 & \quad \text{(T-Abs)}
\end{align*}
\]
Typing rules

\[ \Gamma \vdash \text{true} : \text{Bool} \quad \text{(T-TRUE)} \]

\[ \Gamma \vdash \text{false} : \text{Bool} \quad \text{(T-FALSE)} \]

\[ \Gamma \vdash t_1 : \text{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T \]

\[ \Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T \quad \text{(T-IF)} \]

\[ x : T \in \Gamma \]

\[ \Gamma \vdash x : T \quad \text{(T-VAR)} \]

\[ \Gamma, x : T_1 \vdash t_2 : T_2 \]

\[ \Gamma \vdash \lambda x : T_1 . t_2 : T_1 \rightarrow T_2 \quad \text{(T-ABS)} \]

\[ \Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1 \]

\[ \Gamma \vdash t_1 t_2 : T_2 \quad \Gamma \vdash t_2 : T_1 \]

\[ \Gamma \vdash t_1 t_2 : T_1 \quad \text{(T-APP)} \]
Typing Derivations

What derivations justify the following typing statements?

¬ \( (\lambda x: \text{Bool}. x) \text{true} : \text{Bool} \)
¬ \( f: \text{Bool} \to \text{Bool} \vdash f (\text{if false then true else false}) : \text{Bool} \)
¬ \( f: \text{Bool} \to \text{Bool} \vdash (\lambda x: \text{Bool}. f (\text{if x then false else x})) : \text{Bool} \to \text{Bool} \)
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.
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 $\vdash t : T$, then either $t$ is a value or else $t \rightarrow t'$ for some $t'$.

2. **Preservation**: Types are preserved by one-step evaluation
   
   If $\Gamma \vdash t : T$ and $t \rightarrow t'$, then $\Gamma \vdash t' : T$. 
Proving progress

Same steps as before...
Proving progress

Same steps as before...

♦ inversion lemma for typing relation
♦ canonical forms lemma
♦ progress theorem
Typing rules again (for reference)

\[
\Gamma \vdash \text{true} : \text{Bool} \quad (\text{T-TRUE})
\]

\[
\Gamma \vdash \text{false} : \text{Bool} \quad (\text{T-FALSE})
\]

\[
\frac{\Gamma \vdash t_1 : \text{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad (\text{T-IF})
\]

\[
\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad (\text{T-VAR})
\]

\[
\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1. t_2 : T_1 \rightarrow T_2} \quad (\text{T-ABS})
\]

\[
\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 \ t_2 : T_{12}} \quad (\text{T-APP})
\]
Inversion

Lemma:

1. If $\Gamma \vdash \text{true} : R$, then $R = \text{Bool}$.
2. If $\Gamma \vdash \text{false} : R$, then $R = \text{Bool}$.
3. If $\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$, then $\Gamma \vdash t_1 : \text{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.
Inversion

Lemma:

1. If $\Gamma \vdash \text{true} : R$, then $R = \text{Bool}$.

2. If $\Gamma \vdash \text{false} : R$, then $R = \text{Bool}$.

3. If $\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$, then $\Gamma \vdash t_1 : \text{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.

4. If $\Gamma \vdash x : R$, then
Inversion

Lemma:

1. If $\Gamma \vdash \text{true} : R$, then $R = \text{Bool}$.
2. If $\Gamma \vdash \text{false} : R$, then $R = \text{Bool}$.
3. If $\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$, then $\Gamma \vdash t_1 : \text{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.
4. If $\Gamma \vdash x : R$, then $x : R \in \Gamma$. 
Inversion

Lemma:

1. If $\Gamma \vdash \text{true} : R$, then $R = \text{Bool}$.

2. If $\Gamma \vdash \text{false} : R$, then $R = \text{Bool}$.

3. If $\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$, then $\Gamma \vdash t_1 : \text{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.

4. If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.

5. If $\Gamma \vdash \lambda x : T_1 . t_2 : R$, then

---

CIS 500, 6 October
Inversion

Lemma:

1. If $\Gamma \vdash \text{true} : R$, then $R = \text{Bool}$.

2. If $\Gamma \vdash \text{false} : R$, then $R = \text{Bool}$.

3. If $\Gamma \vdash \text{if} \, t_1 \, \text{then} \, t_2 \, \text{else} \, t_3 : R$, then $\Gamma \vdash t_1 : \text{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.

4. If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.

5. If $\Gamma \vdash \lambda x : T_1 . t_2 : R$, then $R = T_1 \rightarrow R_2$ for some $R_2$ with $\Gamma, \, x : T_1 \vdash t_2 : R_2$. 
Inversion

Lemma:

1. If $\Gamma \vdash \text{true} : R$, then $R = \text{Bool}$.

2. If $\Gamma \vdash \text{false} : R$, then $R = \text{Bool}$.

3. If $\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$, then $\Gamma \vdash t_1 : \text{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.

4. If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.

5. If $\Gamma \vdash \lambda x : T_1 . t_2 : R$, then $R = T_1 \rightarrow R_2$ for some $R_2$ with $\Gamma, x : T_1 \vdash t_2 : R_2$.

6. If $\Gamma \vdash t_1 \ t_2 : R$, then
Inversion

Lemma:

1. If $\Gamma \vdash \text{true} : R$, then $R = \text{Bool}$.

2. If $\Gamma \vdash \text{false} : R$, then $R = \text{Bool}$.

3. If $\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$, then $\Gamma \vdash t_1 : \text{Bool}$ and $\Gamma \vdash t_2, t_3 : R$.

4. If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.

5. If $\Gamma \vdash \lambda x : T_1 . t_2 : R$, then $R = T_1 \rightarrow R_2$ for some $R_2$ with $\Gamma, x : T_1 \vdash t_2 : R_2$.

6. If $\Gamma \vdash t_1 \ t_2 : R$, then there is some type $T_{11}$ such that $\Gamma \vdash t_1 : T_{11} \rightarrow R$ and $\Gamma \vdash t_2 : T_{11}$. 
Canonical Forms

Lemma:
Canonical Forms

Lemma:

1. If \( v \) is a value of type \texttt{Bool}, then
Canonical Forms

Lemma:

1. If $v$ is a value of type $\text{Bool}$, then $v$ is either $\text{true}$ or $\text{false}$.
Canonical Forms

Lemma:

1. If $v$ is a value of type $\text{Bool}$, then $v$ is either $\text{true}$ or $\text{false}$.

2. If $v$ is a value of type $T_1 \rightarrow T_2$, then
Canonical Forms

Lemma:

1. If $v$ is a value of type $\text{Bool}$, then $v$ is either $\text{true}$ or $\text{false}$.

2. If $v$ is a value of type $T_1 \rightarrow T_2$, then $v$ has the form $\lambda x : T_1 . t_2$. 
**Progress**

**Theorem:** Suppose $t$ is a closed, well-typed term (that is, $\vdash t : T$ for some $T$). Then either $t$ is a value or else there is some $t'$ with $t \rightarrow t'$.

**Proof:** By induction
Progress

**Theorem:** Suppose $t$ is a closed, well-typed term (that is, $\vdash t : T$ for some $T$). Then either $t$ is a value or else there is some $t'$ with $t \rightarrow t'$.

**Proof:** By induction on typing derivations.
Progress

**Theorem:** Suppose $t$ is a closed, well-typed term (that is, $\vdash t : T$ for some $T$). Then either $t$ is a value or else there is some $t'$ with $t \rightarrow t'$.

**Proof:** By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because $t$ is closed). The abstraction case is immediate, since abstractions are values.
**Progress**

**Theorem:** Suppose \( t \) is a closed, well-typed term (that is, \( \vdash t : T \) for some \( T \)). Then either \( t \) is a value or else there is some \( t' \) with \( t \rightarrow t' \).

**Proof:** By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because \( t \) is closed). The abstraction case is immediate, since abstractions are values.

Consider the case for application, where \( t = t_1 \ _t_2 \) with \( \vdash t_1 : T_{11} \rightarrow T_{12} \) and \( \vdash t_2 : T_{11} \).
Progress

**Theorem:** Suppose $t$ is a closed, well-typed term (that is, $\vdash t : T$ for some $T$). Then either $t$ is a value or else there is some $t'$ with $t \rightarrow t'$.

**Proof:** By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because $t$ is closed). The abstraction case is immediate, since abstractions are values.

Consider the case for application, where $t = t_1 \ t_2$ with $\vdash t_1 : T_{11} \rightarrow T_{12}$ and $\vdash t_2 : T_{11}$. By the induction hypothesis, either $t_1$ is a value or else it can make a step of evaluation, and likewise $t_2$. 
Progress

**Theorem:** Suppose $t$ is a closed, well-typed term (that is, $\vdash t : T$ for some $T$). Then either $t$ is a value or else there is some $t'$ with $t \rightarrow t'$.

**Proof:** By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because $t$ is closed). The abstraction case is immediate, since abstractions are values.

Consider the case for application, where $t = t_1 \ t_2$ with $\vdash t_1 : T_{11} \rightarrow T_{12}$ and $\vdash t_2 : T_{11}$. By the induction hypothesis, either $t_1$ is a value or else it can make a step of evaluation, and likewise $t_2$. If $t_1$ can take a step, then rule E-App1 applies to $t$. If $t_1$ is a value and $t_2$ can take a step, then rule E-App2 applies. Finally, if both $t_1$ and $t_2$ are values, then the canonical forms lemma tells us that $t_1$ has the form $\lambda x : T_{11}. t_{12}$, and so rule E-AppAbs applies to $t$. 
Proving Preservation

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

Proof: By induction
Proving Preservation

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

Proof: By induction on typing derivations.

[Which case is the hard one?]
Proving Preservation

Theorem: If $\Gamma \vdash t : T$ and $t \rightarrow t'$, 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 $\Gamma \vdash t : T$ and $t \rightarrow t'$, 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 $\Gamma \vdash t' : T_{12}$

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

Theorem: If $\Gamma \vdash t : T$ and $t \rightarrow t'$, 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 $\Gamma \vdash t' : T_{12}$

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

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

- $t_2$ a value $v_2$
- $t' = [x \mapsto v_2] t_{12}$
Theorem: If $\Gamma \vdash t : T$ and $t \rightarrow t'$, 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 $\Gamma \vdash t' : T_{12}$

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 substitution.

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 substitution.

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

Proof: ...
On to real programming languages...
The **Unit** type

\[ t ::= ... \quad \text{terms} \]

\[ \text{unit} \quad \text{constant unit} \]

\[ v ::= ... \quad \text{values} \]

\[ \text{unit} \quad \text{constant unit} \]

\[ T ::= ... \quad \text{types} \]

\[ \text{Unit} \quad \text{unit type} \]

**New typing rules**

\[ \Gamma \vdash t : T \]

\[ \Gamma \vdash \text{unit} : \text{Unit} \quad (T-\text{UNIT}) \]
Sequencing

\[ t ::= \ldots \]
\[ t_1; t_2 \]
Sequencing

t ::= ... terms

\[ t_1; t_2 \]

\[
\frac{t_1 \rightarrow t_1'}{t_1; t_2 \rightarrow t_1'; t_2}
\] \hspace{1cm} (E-SEQ)

\[
\frac{\text{unit}; t_2 \rightarrow t_2}{\text{(E-SEQNEXT)}}
\]

\[
\frac{\Gamma \vdash t_1 : \text{Unit} \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1; t_2 : T_2}
\] \hspace{1cm} (T-SEQ)
Derived forms

- Syntactic sugar
- Internal language vs. external (surface) language
Sequencing as a derived form

\[ t_1 ; t_2 \overset{\text{def}}{=} (\lambda x: \text{Unit}. t_2) \; t_1 \]

where \( x \not\in FV(t_2) \)
Equivalence of the two definitions

[board]
Ascription

New syntactic forms
\[ t ::= \ldots \quad \text{terms} \]
\[ t \text{ as } T \quad \text{ascription} \]

New evaluation rules
\[ v_1 \text{ as } T \rightarrow v_1 \quad (E\text{-ASCRIBE}) \]
\[ t_1 \rightarrow t_1' \quad (E\text{-ASCRIBE1}) \]
\[ t_1 \text{ as } T \rightarrow t_1' \text{ as } T \]

New typing rules
\[ \Gamma \vdash t_1 : T \quad (T\text{-ASCRIBE}) \]
\[ \Gamma \vdash t_1 \text{ as } T : T \]
Ascription as a derived form

\[ t \text{ as } T \overset{\text{def}}{=} (\lambda x:T. \ x) \ t \]
Let-bindings

New syntactic forms
\[ t ::= \ldots \]
\[ \text{let } x = t \text{ in } t \]

terms

let binding

New evaluation rules
\[ \text{let } x = v_1 \text{ in } t_2 \rightarrow [x \mapsto v_1]t_2 \]
\[ t_1 \rightarrow t'_1 \]
\[ \frac{}{\text{let } x = t_1 \text{ in } t_2 \rightarrow \text{let } x = t'_1 \text{ in } t_2} \]
\[ (E-LetV) \]
\[ (E-Let) \]

New typing rules
\[ \frac{\Gamma \vdash t : T}{\Gamma \vdash \text{let } x = t \text{ in } t_2 : T} \]
\[ (T-Let) \]
Pairs

\[ t ::= \ldots \quad \text{terms} \]
\[ \{t, t\} \quad \text{pair} \]
\[ t.1 \quad \text{first projection} \]
\[ t.2 \quad \text{second projection} \]

\[ v ::= \ldots \quad \text{values} \]
\[ \{v, v\} \quad \text{pair value} \]

\[ T ::= \ldots \quad \text{types} \]
\[ T_1 \times T_2 \quad \text{product type} \]
Evaluation rules for pairs

\[
\begin{align*}
\{v_1, v_2\}.1 & \rightarrow v_1 \\
\{v_1, v_2\}.2 & \rightarrow v_2 \\
\{t_1, t_2\} & \rightarrow \{t'_1, t_2\} \\
\{v_1, t_2\} & \rightarrow \{v_1, t'_2\}
\end{align*}
\]

\(\text{(E-PairBeta1)}\)

\(\text{(E-PairBeta2)}\)

\(\text{(E-Proj1)}\)

\(\text{(E-Proj2)}\)

\(\text{(E-Pair1)}\)

\(\text{(E-Pair2)}\)
Typing rules for pairs

\[ \Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : T_2 \]
\[ \Gamma \vdash \{t_1, t_2\} : T_1 \times T_2 \]  \hspace{1cm} (T-PAIR)

\[ \Gamma \vdash t_1 : T_{11} \times T_{12} \]
\[ \Gamma \vdash t_{1.1} : T_{11} \]  \hspace{1cm} (T-PROJ1)

\[ \Gamma \vdash t_1 : T_{11} \times T_{12} \]
\[ \Gamma \vdash t_{1.2} : T_{12} \]  \hspace{1cm} (T-PROJ2)
Tuples

\[ t ::= \ldots \]
\[ \{ t_i \}_{i \in 1..n} \]
\[ t.i \]

\[ v ::= \ldots \]
\[ \{ v_i \}_{i \in 1..n} \]

\[ T ::= \ldots \]
\[ \{ T_i \}_{i \in 1..n} \]

terms

tuple

projection

values
tuple value

types
tuple type
Evaluation rules for tuples

\[
\{v_i^{i \in 1..n}\}.j \rightarrow v_j \quad \text{(E-PROJ\_TUPLE)}
\]

\[
\begin{align*}
t_1 & \rightarrow t'_1 \\
t_1.i & \rightarrow t'_1.i
\end{align*}
\quad \text{(E-PROJ)}
\]

\[
\begin{align*}
t_j & \rightarrow t'_j \\
\{v_i^{i \in 1..j-1}, t_j, t_k^{k \in j+1..n}\} & \rightarrow \{v_i^{i \in 1..j-1}, t'_j, t_k^{k \in j+1..n}\}
\end{align*}
\quad \text{(E-TUPLE)}
\]
Typing rules for tuples

\begin{align*}
\text{for each } i & \quad \Gamma \vdash t_i : T_i \\
\Gamma & \vdash \{t_i \; \mid i \in 1..n\} : \{T_i \; \mid i \in 1..n\} \\
\hline \\
\Gamma & \vdash t_1 : \{T_i \; \mid i \in 1..n\} \\
\Gamma & \vdash t_1.j : T_j
\end{align*}

(T-TUPLE) (T-PROJ)
Records

\[ t ::= \ldots \]
\[ \{ l_i = t_i \mid i \in \ldots \} \]
\[ t.l \]

\[ v ::= \ldots \]
\[ \{ l_i = v_i \mid i \in \ldots \} \]

\[ T ::= \ldots \]
\[ \{ l_i : T_i \mid i \in \ldots \} \]

**terms**

**record**

**projection**

**values**

**record value**

**types**

**type of records**
Evaluation rules for records

\{l_i=v_i \quad i \in \{1,\ldots,n\}\}.l_j \rightarrow v_j \quad (E-\text{PROJRCD})

\[ t_1 \rightarrow t_1' \]
\[ t_1.l \rightarrow t_1'.l \quad (E-\text{PROJ})\]

\[ t_j \rightarrow t_j' \]
\[ \{l_i=v_i \quad i \in \{1,\ldots,j-1\}, l_j=t_j, l_k=t_k \quad k \in \{j+1,\ldots,n\}\} \]
\[ \rightarrow \{l_i=v_i \quad i \in \{1,\ldots,j-1\}, l_j=t_j', l_k=t_k \quad k \in \{j+1,\ldots,n\}\} \quad (E-\text{RCD})\]
Typing rules for records

\[
\text{for each } i \quad \Gamma \vdash t_i : T_i
\]

\[
\Gamma \vdash \{l_i=t_i \quad i \in 1..n\} : \{l_i:T_i \quad i \in 1..n\}
\]

\[\Gamma \vdash t_1 : \{l_i:T_i \quad i \in 1..n\} \]

\[
\Gamma \vdash t_1.l_j : T_j
\]
Discussion
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?
The Curry-Howard Correspondence

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

- “law of the excluded middle” — $P \lor \neg P$ — not recognized.

A proof of $P \land Q$ 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

<table>
<thead>
<tr>
<th>Logic</th>
<th>Programming Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>propositions</td>
<td>types</td>
</tr>
<tr>
<td>proposition $P \supset Q$</td>
<td>type $P \rightarrow Q$</td>
</tr>
<tr>
<td>proposition $P \land Q$</td>
<td>type $P \times Q$</td>
</tr>
<tr>
<td>proof of proposition $P$</td>
<td>term $t$ of type $P$</td>
</tr>
<tr>
<td>proposition $P$ is provable</td>
<td>type $P$ is inhabited (by some term)</td>
</tr>
</tbody>
</table>
# Propositions as Types

<table>
<thead>
<tr>
<th>Logic</th>
<th>Programming Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>propositions</td>
<td>types</td>
</tr>
<tr>
<td>proposition $P \supset Q$</td>
<td>type $P \rightarrow Q$</td>
</tr>
<tr>
<td>proposition $P \land Q$</td>
<td>type $P \times Q$</td>
</tr>
<tr>
<td>proof of proposition $P$</td>
<td>term $t$ of type $P$</td>
</tr>
<tr>
<td>proposition $P$ is provable</td>
<td>type $P$ is inhabited (by some term)</td>
</tr>
<tr>
<td></td>
<td>evaluation</td>
</tr>
</tbody>
</table>
## Propositions as Types

<table>
<thead>
<tr>
<th>Logic</th>
<th>Programming Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>propositions</td>
<td>types</td>
</tr>
<tr>
<td>proposition $P \supset Q$</td>
<td>type $P \rightarrow Q$</td>
</tr>
<tr>
<td>proposition $P \land Q$</td>
<td>type $P \times Q$</td>
</tr>
<tr>
<td>proof of proposition $P$</td>
<td>term $t$ of type $P$</td>
</tr>
<tr>
<td>proposition $P$ is provable</td>
<td>type $P$ is inhabited (by some term)</td>
</tr>
<tr>
<td>proof simplification</td>
<td>evaluation</td>
</tr>
<tr>
<td>(a.k.a. “cut elimination”)</td>
<td></td>
</tr>
</tbody>
</table>
Erasure

\[
\begin{align*}
erase(x) & = x \\
erase(\lambda x : T_1 . \ t_2) & = \lambda x . \ erase(t_2) \\
erase(t_1 \ t_2) & = erase(t_1) \ erase(t_2)
\end{align*}
\]
Typability

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

Cf. type reconstruction in OCaml.