Simply-Typed Lambda Calculus

(Slides mostly follow Dan Grossman's teaching materials)

Review of untyped λ-calculus

- Syntax: notation for defining functions

(Terms) M, N ::= x | λx. M | M N

- Semantics: reduction rules

$$\frac{}{(\lambda x. M)N \;\rightarrow\; M[N/x]}\;(\beta)$$

$$\frac{M \rightarrow M'}{M\ N \rightarrow M'N} \qquad \frac{N \rightarrow N'}{M\ N \rightarrow M\ N'} \qquad \frac{M \rightarrow M'}{\lambda x. M \rightarrow \lambda x. M'}$$

$$\frac{}{(\lambda x.M)N \ \rightarrow \ M[N/x]} \ (\beta)$$

$$\frac{M \rightarrow M'}{M\ N \rightarrow M'N}$$

$$\frac{N \rightarrow N'}{M\ N \rightarrow M\ N'}$$

$$\frac{M \rightarrow M'}{\lambda x.M \rightarrow \lambda x.M'}$$

(λf. λz. f (f z)) (λy. y+x)

→ λz. (λy. y+x) ((λy. y+x) z)

→ λz. (λy. y+x) (z+x)

→ λz. z+x+x

Review of untyped λ-calculus

$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$

$\rightarrow (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$

$\rightarrow$ …

This class: adding a type system

(We will see that well-typed terms in STLC always terminate.)

Why types

- Type checking catches “simple” mistakes early
  - Example: 2 + true + “a”
- (**Type safety**) Well-typed programs will not go wrong
  - Ensure execution never reach a “meaningless” state
  - But “meaningless” depends on the semantics (each language typically defines some as type errors and others run-time errors)
- Typed programs are easier to analyze and optimize
  - Compilers can generate better code (e.g. access components of structures by known offset)

Cons: impose constraints on the programmer

- Some valid programs might be rejected

Why formal type systems

- Many typed languages have informal descriptions of the type systems (e.g., in language reference manuals)
- A fair amount of careful analysis is required to avoid false claims of type safety
- A formal presentation of a type system is a precise specification of the type checker
- And allows formal proofs of type safety

What we will study about types

- Type system
  - Typing rules: assign types to terms
  - Type safety (soundness of typing rules): well-typed terms cannot go wrong
- Connection to constructive propositional logic
  - Curry-Howard isomorphism: “Propositions are Types”, “Proofs are Programs”

Adding types to λ-calculus – wrong attempt

base type
(e.g. int, bool)

function type

(Types) τ, σ ::= T | **fun**

Adding types to λ-calculus – wrong attempt

- Typing judgment (to assign types to terms)

$$\vdash M : \tau$$

M is of type τ

Judgment

- A statement J about certain formal properties
- Has a derivation $\vdash J$ (i.e. "a proof")
- Has a meaning ("judgment semantics") $\vDash J$

- Typing rules (to derive the typing judgment)

Adding types to λ-calculus – wrong attempt

Typing rules

$$\frac{}{\vdash (\lambda x. M) : \mathbf{fun}}$$

$$\frac{\vdash M : \mathbf{fun} \qquad \vdash N : \mathrm{T}}{\vdash M\ N : \mathrm{T}}$$

Not type safe, since well-typed terms may go wrong (reduce to a "meaningless" state)

e.g. ((λf. f 1) 3) will go "wrong", though ⊢ (λf. f 1) 3 $:$ int

Adding types to λ-calculus – getting it right

- Classify functions using argument and result types
  - (λx. x) and (λf. f 1) should be of different types: ((λx. x) 3) is acceptable, but ((λf. f 1) 3) is not
  - Explicitly specify argument types in function syntax
- Type-check function bodies, which have free variables
  - Types of free variables are the context: type of (f 1) depends on the type of f

Simply-typed λ-calculus (STLC)

base type (e.g. int, bool)

function type

(Types) τ, σ ::= T | $\sigma \to \tau$

An infinite number of types:

int → int, int → (int → int), (int → int) → int, …

→ is right-associative: $\tau \to \tau \to \tau$ is $\tau \to (\tau \to \tau)$

Simply-typed λ-calculus (STLC)

(Types) $\tau, \sigma \quad ::= \; T \; | \; \sigma \to \tau$

(Terms) $M, N \; ::= \; x \; | \; \lambda x : \tau . \; M \; | \; M \; N$

Reduction rules

$$\frac{}{(\lambda x{:}\,\tau.\,M)N \;\rightarrow\; M[N/x]}\;(\beta)$$

$$\frac{M \rightarrow M'}{M\;N \rightarrow M'N}$$

$$\frac{N \rightarrow N'}{M\;N \;\rightarrow M\;N'}$$

$$\frac{M \rightarrow M'}{\lambda x{:}\,\tau.\,M \rightarrow \lambda x{:}\,\tau.\,M'}$$

Same as untyped λ-calculus

Typing judgment

$$\Gamma \vdash M : \tau$$

M is of type τ in context Γ

- Typing context (a set of typing assumptions)

$$\Gamma ::= \cdot \mid \Gamma, x : \tau$$

  - Include types of all the free variables in M (each free variable x is of type τ)
  - Empty context $\cdot$ is for closed terms

- Under Γ, M is a well-typed term of type τ

Typing rules

$$\frac{}{\Gamma,\ x : \tau \vdash x : \tau}\ \text{(var)}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}\ \text{(app)}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x{:}\sigma. M) : \sigma \to \tau}\ \text{(abs)}$$

Typing derivation examples

$$\frac{}{\Gamma,\ x : \tau \vdash x : \tau}\ \text{(var)}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}\ \text{(app)}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x{:}\sigma.M) : \sigma \to \tau}\ \text{(abs)}$$

$$\cfrac{\cfrac{}{x{:}\tau \vdash x{:}\tau}\ \text{(var)}}{\cdot \vdash (\lambda x{:}\tau.x) : \tau \to \tau}\ \text{(abs)}$$

Typing derivation examples

$$\frac{}{\Gamma,\ x : \tau \vdash x : \tau}\ \text{(var)}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}\ \text{(app)}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x{:}\sigma. M) : \sigma \to \tau}\ \text{(abs)}$$

$$\cfrac{\cfrac{\cfrac{}{x{:}\tau, y{:}\sigma \vdash x : \tau}\ \text{(var)}}{x{:}\tau \vdash (\lambda y{:}\sigma. x) : \sigma \to \tau}\ \text{(abs)}}{\cdot \vdash (\lambda x{:}\tau. \lambda y{:}\sigma. x) : \tau \to \sigma \to \tau}\ \text{(abs)}$$

Typing derivation examples

$$\frac{}{\Gamma,\ x : \tau \ \vdash\ x : \tau}\ \text{(var)}$$

$$\frac{\Gamma \vdash M :\ \sigma \to \tau \qquad \Gamma \vdash N :\ \sigma}{\Gamma \vdash M\ N : \tau}\ \text{(app)}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x{:}\,\sigma. M) : \sigma \to \tau}\ \text{(abs)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x{:}\,\tau \to \tau, y{:}\,\tau\ \vdash x{:}\,\tau \to \tau}\ \text{(var)} \qquad \dfrac{}{x{:}\,\tau \to \tau, y{:}\,\tau\ \vdash y{:}\,\tau}\ \text{(var)}}{x{:}\,\tau \to \tau, y{:}\,\tau\ \vdash\ x\ y : \tau}\ \text{(app)}}{x{:}\,\tau \to \tau\ \vdash\ (\lambda y{:}\,\tau. x\ y) : \tau \to \tau}\ \text{(abs)}}{\cdot \vdash (\lambda x{:}\,\tau \to \tau. \lambda y{:}\,\tau. x\ y) : (\tau \to \tau) \to \tau \to \tau}\ \text{(abs)}$$

Soundness and completeness

- A sound type system never accepts a program that can go wrong
  - No false negatives
  - The language is type-safe
- A complete type system never rejects a program that can't go wrong
  - No false positives
- However, for any Turing-complete PL, the set of programs that may go wrong is undecidable
  - Type system cannot be sound and complete
  - Choose soundness, try to reduce false positives in practice

Soundness – well-typed terms in STLC never go wrong

Theorem (Type Safety):

If $\cdot \vdash M : \tau$ and $M \to^* M'$, then

$\cdot \vdash M' : \tau$, and either $M' \in$ Values or $\exists M''. M' \to M''$

Defined in language semantics (e.g. λ-abstraction, constants)

That is, the reduction of a well-typed term either diverges, or terminates in a value of the expected type.

Follows from two key lemmas (next page).

Soundness – well-typed terms in STLC never go wrong

- Preservation (subject reduction): well-typed terms reduce only to well-typed terms of the same type

$$\text{If } \cdot \vdash M : \tau \text{ and } M \to M', \text{then } \cdot \vdash M' : \tau$$

- Progress: a well-typed term is either a value or can be reduced

$$\text{If } \cdot \vdash M : \tau, \text{then either } M \in \text{Values or } \exists M'. M \to M'$$

Not complete – the type system may reject terms that do not go wrong

- (λx. (x (λy. y)) (x 3)) (λz. z)

  Cannot find σ, τ such that

  $$x{:}\,\sigma \;\vdash\; \big(x\;(\lambda y.\,y)\big)(x\;3) \,:\; \tau$$

  because we have to pick one type for x

- But (λx. (x (λy. y)) (x 3)) (λz. z)

  $\to$ ((λz. z) (λy. y)) ((λz. z) 3)

  $\to$ (λy. y) 3 $\to$ 3

Well-typed terms in STLC always terminate (strong normalization theorem)

- Recall $(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$
  $\rightarrow (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$
  $\rightarrow \ldots$

- $(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$ cannot be assigned a type

Expect σ to be in the form of $\sigma \rightarrow \tau$, which is impossible!

$$\frac{x{:}\,\sigma \ \vdash x{:}\,? \qquad x{:}\,\sigma \ \vdash x{:}\,\sigma}{x{:}\,\sigma \ \vdash \ x\ x : ?}$$

Main points of STLC

(Types) $\tau, \sigma \quad ::= \ T \ | \ \sigma \to \tau$

(Terms) $M, N \ ::= \ x \ | \ \lambda x : \tau. M \ | \ M \, N$

Reduction rules

$$\frac{}{(\lambda x{:}\tau. M)N \ \to \ M[N/x]} \ (\beta)$$

Typing rules

$$\frac{}{\Gamma, \ x : \tau \ \vdash \ x : \tau} \ (\text{var}) \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x{:}\sigma. M) : \sigma \to \tau} \ (\text{abs})$$

$$\frac{\Gamma \vdash M : \ \sigma \to \tau \qquad \Gamma \vdash N : \ \sigma}{\Gamma \vdash M \ N : \tau} \ (\text{app})$$

Soundness (type safety)

Adding stuff

Use STLC as a foundation for understanding other common language constructs

- Extend the syntax (types & terms)
- Extend the operational semantics (reduction rules)
- Extend the type system (typing rules)
- Extend the soundness proof (new proof cases)

Adding product type

(Types) τ, σ ::= … | $\sigma \times \tau$

(Terms) M, N ::= … | <M, N> | proj1 M | proj2 M

Consider structures in C:

```
struct date{
    int year;
    int month;
    int day;
}
```

Product type

(Types) $\tau, \sigma \; ::= \; \ldots \; | \; \sigma \times \tau$

(Terms) $M, N \; ::= \; \ldots \; | \; \langle M, N \rangle \; | \; \text{proj1}\ M \; | \; \text{proj2}\ M$

Reduction rules

$$\frac{}{\text{proj1}\ \langle M, N \rangle \to M} \qquad \frac{}{\text{proj2}\ \langle M, N \rangle \to N}$$

$$\frac{M \to M'}{\langle M, N \rangle \to \langle M', N \rangle} \qquad \frac{N \to N'}{\langle M, N \rangle \to \langle M, N' \rangle}$$

$$\frac{M \to M'}{\text{proj1}\ M \to \text{proj1}\ M'} \qquad \frac{M \to M'}{\text{proj2}\ M \to \text{proj2}\ M'}$$

Product type

(Types) τ, σ ::= … | $\sigma \times \tau$

(Terms) M, N ::= … | <M, N> | proj1 M | proj2 M

Typing rules

$$\frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau} \text{ (pair)}$$

$$\frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \text{proj1 } M : \sigma} \text{ (proj1)} \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \text{proj2 } M : \tau} \text{ (proj2)}$$

Typing derivation example

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x:\sigma\times\tau\ \vdash x:\sigma\times\tau}\,(\text{var})}{x:\sigma\times\tau\ \vdash \text{proj2}\ x\ :\ \tau}\,(\text{proj2}) \qquad \dfrac{\dfrac{}{x:\sigma\times\tau\ \vdash x:\sigma\times\tau}\,(\text{var})}{x:\sigma\times\tau\ \vdash \text{proj1}\ x\ :\ \sigma}\,(\text{proj1})}{x:\sigma\times\tau\ \vdash <\text{proj2}\ x, \text{proj1}\ x>\ :\ \tau\times\sigma}\,(\text{pair})}{\cdot \vdash (\lambda x:\sigma\times\tau.<\text{proj2}\ x, \text{proj1}\ x>)\ :\ (\sigma\times\tau)\rightarrow(\tau\times\sigma)}\,(\text{abs})$$

Soundness theorem (type safety)

- Preservation:

$$\text{If } \cdot \vdash M : \tau \text{ and } M \to M', \text{then } \cdot \vdash M' : \tau$$

- Progress:

$$\text{If } \cdot \vdash M : \tau, \text{then either } M \in \text{Values or } \exists M'. M \to M'$$

Include <v1, v2> now

Adding sum type

(Types) τ, σ ::= … | $\sigma + \tau$

(Terms) M, N ::= … | left M | right M | case M do M1 M2

Consider unions in C:

```
union data{
    int i;
    float f;
    char c;
}
```

Using the same location for multiple data.
Can contain only one value at any given time.

Adding sum type

(Types) τ, σ ::= … | $\sigma + \tau$

(Terms) M, N ::= … | left M | right M | case M do M1 M2

Subclasses in Java:

```
abstract class t {abstract t' m();}
class A extends t { t1 x; t' m(){...}}
class B extends t { t2 x; t' m(){...}}
...
e.m();
```

case e do A.m B.m

Adding sum type

(Types) τ, σ ::= … | $\sigma + \tau$

(Terms) M, N ::= … | left M | right M | case M do M1 M2

In Coq:

```
Inductive bool : Set :=
  | true : bool
  | false : bool.
Definition not (b : bool) : bool :=
  match b with
    | true => false
    | false => true
  end.
```

Sum type: reduction rules

$$\frac{}{\text{case (left M) do M1 M2} \rightarrow \text{M1 M}}$$

$$\frac{}{\text{case (right M) do M1 M2} \rightarrow \text{M2 M}}$$

$$\frac{\text{M} \rightarrow \text{M'}}{\text{case M do M1 M2} \rightarrow \text{case M' do M1 M2}}$$

$$\frac{\text{M1} \rightarrow \text{M1'}}{\text{case M do M1 M2} \rightarrow \text{case M do M1' M2}}$$

$$\frac{\text{M2} \rightarrow \text{M2'}}{\text{case M do M1 M2} \rightarrow \text{case M do M1 M2'}}$$

$$\frac{\text{M} \rightarrow \text{M'}}{\text{left M} \rightarrow \text{left M'}}$$

$$\frac{\text{M} \rightarrow \text{M'}}{\text{right M} \rightarrow \text{right M'}}$$

Sum type: typing rules

$$\frac{\Gamma \vdash \mathrm{M}\!: \sigma}{\Gamma \vdash \mathrm{left\ M} : \sigma + \tau}\ \text{(left)}$$

$$\frac{\Gamma \vdash \mathrm{M}\!: \tau}{\Gamma \vdash \mathrm{right\ M} : \sigma + \tau}\ \text{(right)}$$

$$\frac{\Gamma \vdash \mathrm{M}\!: \sigma + \tau \qquad \Gamma \vdash \mathrm{M1}\!: \sigma \to \rho \qquad \Gamma \vdash \mathrm{M2}\!: \tau \to \rho}{\Gamma \vdash \mathrm{case\ M\ do}\ M1\ M2\!: \rho}\ \text{(case)}$$

Typing derivation examples

$$\dfrac{\dfrac{\dfrac{}{x:\tau \vdash x:\tau}\ \text{(var)}}{x:\tau \vdash \text{left}\ x : \tau + \sigma}\ \text{(proj2)} \qquad \dfrac{\dfrac{}{x:\tau \vdash x:\tau}\ \text{(var)}}{x:\tau \vdash \text{left}\ x : \tau + \rho}\ \text{(proj1)}}{\dfrac{x:\tau \vdash <\text{left}\ x, \text{left}\ x> : (\tau + \sigma) \times (\tau + \rho)}{\cdot \vdash (\lambda x:\tau. <\text{left}\ x, \text{left}\ x>) : \tau \to (\tau + \sigma) \times (\tau + \rho)}\ \text{(abs)}}\ \text{(pair)}$$

other side can be anything

Soundness theorem (type safety)

- Preservation:

$$\text{If } \cdot \vdash M : \tau \text{ and } M \to M', \text{then } \cdot \vdash M' : \tau$$

- Progress:

$$\text{If } \cdot \vdash M : \tau, \text{then either } M \in \text{Values or } \exists M'.\, M \to M'$$

Include “left v” and “right v” now

Products vs. sums

- “logical duals” (more on this later)
  - To make a $\sigma \times \tau$, we need a σ and a τ
  - To make a $\sigma + \tau$, we need a σ or a τ
  - Given a $\sigma \times \tau$, we can get a σ or a τ or both (our “choice”)
  - Given a $\sigma + \tau$, we must be prepared for either a σ or a τ (the value’s “choice”)

Main points till now

- STLC extended with products and sums:

(Types) τ, σ ::= T | $\sigma \to \tau$ | $\sigma \times \tau$ | $\sigma + \tau$

(Terms) M, N ::= x | λx : τ. M | M N
| <M, N> | proj1 M | proj2 M
| left M | right M | case M do M1 M2

- Next: recursion

Recursion

- Recall in untyped λ-calculus, every term has a fixpoint
  - Fixpoint combinator is a higher-order function h satisfying

    for all f, (h f) gives a fixpoint of f

    i.e. h f = f (h f)

  - Turing's fixpoint combinator Θ

    Let A = $\lambda x. \lambda y.\ y\ (x\ x\ y)$ and Θ = A A

  - Church's fixpoint combinator **Y**

    Let **Y** = $\lambda f. (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$

Recursion

- Recall "strong normalization theorem": well-typed terms in STLC always terminate
  - Extensions so far (products & sums) preserve termination
- Recursion is not allowed by the typing rules: it is impossible to find types for fixed-point combinators
- So we add an explicit construct for recursion

(Terms) M, N ::= … | **fix** M

(Types) τ, σ ::= … (no new types)

Reduction rules for fix

$$\frac{}{\mathbf{fix}\,\lambda x.\,M \rightarrow\ M[\mathbf{fix}\,\lambda x.\,M/x]}$$

$$\frac{M \rightarrow M'}{\mathbf{fix}\,M \rightarrow \mathbf{fix}\,M'}$$

(**fix** λf. λn. if (n == 0) then 1 else n * f(n-1)) 3

→ (λn. if (n == 0) then 1 else n*((**fix** λf. λn. if (n == 0) then 1 else n * f(n-1))(n-1))) 3

→ if (3 == 0) then 1 else 3 * ((**fix** λf. λn. if (n == 0) then 1 else n * f(n-1))(3-1))

→ 3 * ((**fix** λf. λn. if (n == 0) then 1 else n * f(n-1))(3-1))

→ …

Typing fix

$$\frac{\Gamma \vdash M : \tau \rightarrow \tau}{\Gamma \vdash \mathbf{fix}\, M : \tau} \text{(fix)}$$

- Math explanation: If M is a function from τ to τ , then **fix** M, the fixed-point of M, is some τ with the fixed-point property
- Operational explanation: **fix** λx.M' reduces to M'[**fix** λx.M'/x].
    - The substitution means x and **fix** λx.M' need the same type
    - The result means M' and **fix** λx.M' need the same type
- Soundness (type safety) is straightforward
- But strong normalization is eliminated

Main points till now

- STLC with products and sums:

(Types) τ, σ ::= T | $\sigma \rightarrow \tau$ | $\sigma \times \tau$ | $\sigma + \tau$

(Terms) M, N ::= x | λx : τ. M | M N
| <M, N> | proj1 M | proj2 M
| left M | right M | case M do M1 M2

- We can also add recursion

- Next: Curry-Howard isomorphism

Curry-Howard Isomorphism

- What we did:
  - Define a programming language
  - Define a type system to rule out “bad” programs
- What logicians do:
  - Define logic propositions
    - E.g. $p, q ::= B \mid p \wedge q \mid p \vee q \mid p \Rightarrow q$
  - Define a proof system to prove “good” propositions
- Turn out to be related
  - Propositions are Types
  - Proofs are Programs

Curry-Howard Isomorphism

- Slogans
  - Propositions are Types
  - Proofs are Programs

In this class, we will show correspondence between formulas of constructive propositional logic

$$\text{(Prop)} \quad p, q \quad ::= \ B \mid p \Rightarrow q \mid p \wedge q \mid p \vee q$$

and types of STLC with products and sums

$$\text{(Types)} \quad \tau, \sigma \quad ::= \ T \mid \sigma \rightarrow \tau \mid \sigma \times \tau \mid \sigma + \tau$$

Examples of terms and types

$\lambda x : \tau.\ x$

has type

$\tau \to \tau$

Examples of terms and types

$$\lambda x{:}\ \tau.\ \lambda f{:}\ \tau \to \sigma.\ f\ x$$

has type

$$\tau \to (\tau \to \sigma) \to \sigma$$

Examples of terms and types

$$\lambda f\!: \tau \to \sigma \to \rho.\ \lambda x\!: \sigma.\ \lambda y\!: \tau.\ f\ y\ x$$

has type

$$(\tau \to \sigma \to \rho) \to \sigma \to \tau \to \rho$$

Examples of terms and types

λx: τ. <left x, left x>

has type

$$\tau \rightarrow ((\tau + \sigma) \times (\tau + \rho))$$

Examples of terms and types

$$\lambda f{:}\ \tau \rightarrow \rho.\ \lambda g{:}\ \sigma \rightarrow \rho.\ \lambda x{:}\ \tau + \sigma.\ (\text{case } x \text{ do } f\ g)$$

has type

$$(\tau \rightarrow \rho) \rightarrow (\sigma \rightarrow \rho) \rightarrow (\tau + \sigma) \rightarrow \rho$$

Examples of terms and types

λx: $\tau \times \sigma$. λy: ρ. < <y, proj1 x>, proj2 x >

has type

$$(\tau \times \sigma) \to \rho \to ((\rho \times \tau) \times \sigma)$$

Empty and nonempty types

Have seen several "nonempty" types (closed terms of that type)

$\tau \to \tau$

$\tau \to (\tau \to \sigma) \to \sigma$

$(\tau \to \sigma \to \rho) \to \sigma \to \tau \to \rho$

$\tau \to ((\tau + \sigma) \times (\tau + \rho))$

$(\tau \to \rho) \to (\sigma \to \rho) \to (\tau + \sigma) \to \rho$

$(\tau \times \sigma) \to \rho \to ((\rho \times \tau) \times \sigma)$

There're also lots of "empty" types (no closed terms of that type)

$\tau \qquad \tau \to \sigma \qquad \tau + (\tau \to \sigma) \qquad \tau \to (\sigma \to \tau) \to \sigma$

How to know whether a type is nonempty?

How to know whether a type is nonempty?

Let's replace $\rightarrow$ with $\Rightarrow$, $\times$ with $\wedge$, + with $\vee$:

$\tau \Rightarrow \tau$
$\tau \Rightarrow (\tau \Rightarrow \sigma) \Rightarrow \sigma$
$(\tau \Rightarrow \sigma \Rightarrow \rho) \Rightarrow \sigma \Rightarrow \tau \Rightarrow \rho$
$\tau \Rightarrow ((\tau \vee \sigma) \wedge (\tau \vee \rho))$
$(\tau \Rightarrow \rho) \Rightarrow (\sigma \Rightarrow \rho) \Rightarrow (\tau \vee \sigma) \Rightarrow \rho$
$(\tau \wedge \sigma) \Rightarrow \rho \Rightarrow ((\rho \wedge \tau) \wedge \sigma)$

Can be proved in propositional logic

(corresponding to nonempty types – have closed terms)

τ
$\tau \Rightarrow \sigma$
$\tau \vee (\tau \Rightarrow \sigma)$
$\tau \Rightarrow (\sigma \Rightarrow \tau) \Rightarrow \sigma$

Cannot be proved in propositional logic

(corresponding to empty types – no closed terms)

Example – propositional-logic proof

$\Gamma \vdash p$

assumptions

$$\dfrac{\dfrac{\dfrac{\tau, \tau \Rightarrow \sigma \vdash \tau \Rightarrow \sigma \qquad \tau, \tau \Rightarrow \sigma \vdash \tau}{\tau, \tau \Rightarrow \sigma \vdash \sigma}}{\tau \vdash (\tau \Rightarrow \sigma) \Rightarrow \sigma}}{\cdot \vdash \tau \Rightarrow (\tau \Rightarrow \sigma) \Rightarrow \sigma}$$

Propositional logic (natural deduction)

(Prop) p, q ::= B | $p \Rightarrow q$ | $p \wedge q$ | $p \vee q$

(Ctxt) Γ ::= $\cdot$ | Γ, p

$$\frac{}{\Gamma, p \vdash p}\ \text{(axiom)} \qquad \frac{\Gamma, p \vdash q}{\Gamma \vdash p \Rightarrow q}\ (\Rightarrow\text{-intro}) \qquad \frac{\Gamma \vdash p \Rightarrow q \quad \Gamma \vdash p}{\Gamma \vdash q}\ (\Rightarrow\text{-elim})$$

$$\frac{\Gamma \vdash p \quad \Gamma \vdash q}{\Gamma \vdash p \wedge q}\ (\wedge\text{-intro}) \qquad \frac{\Gamma \vdash p \wedge q}{\Gamma \vdash p}\ (\wedge\text{-elim-l}) \qquad \frac{\Gamma \vdash p \wedge q}{\Gamma \vdash q}\ (\wedge\text{-elim-r})$$

$$\frac{\Gamma \vdash p}{\Gamma \vdash p \vee q}\ (\vee\text{-intro-l}) \qquad \frac{\Gamma \vdash q}{\Gamma \vdash p \vee q}\ (\vee\text{-intro-r})$$

$$\frac{\Gamma \vdash p \vee q \quad \Gamma \vdash p \Rightarrow r \quad \Gamma \vdash q \Rightarrow r}{\Gamma \vdash r}\ (\vee\text{-elim})$$

This is exactly our type system, erasing terms, replacing → with ⇒, × with ∧, + with ∨

$$\frac{}{\Gamma,\ x : \tau \vdash x : \tau}\ \text{(var)}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x{:}\sigma. M) : \sigma \to \tau}\ \text{(abs)}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}\ \text{(app)}$$

$$\frac{\Gamma \vdash \mathrm{M}: \sigma \qquad \Gamma \vdash \mathrm{N}: \tau}{\Gamma \vdash \langle \mathrm{M}, \mathrm{N} \rangle : \sigma \times \tau}\ \text{(pair)}$$

$$\frac{\Gamma \vdash \mathrm{M}: \sigma \times \tau}{\Gamma \vdash \text{proj1 M} : \sigma}\ \text{(proj1)}$$

$$\frac{\Gamma \vdash \mathrm{M}: \sigma \times \tau}{\Gamma \vdash \text{proj2 M} : \tau}\ \text{(proj2)}$$

$$\frac{\Gamma \vdash \mathrm{M}: \sigma}{\Gamma \vdash \text{left}\ M : \sigma + \tau}\ \text{(left)}$$

$$\frac{\Gamma \vdash \mathrm{M}: \tau}{\Gamma \vdash \text{right}\ M : \sigma + \tau}\ \text{(right)}$$

$$\frac{\Gamma \vdash \mathrm{M}: \sigma + \tau \qquad \Gamma \vdash \mathrm{M1}: \sigma \to \rho \qquad \Gamma \vdash \mathrm{M2}: \tau \to \rho}{\Gamma \vdash \text{case}\ M\ \text{do}\ M1\ M2 : \rho}\ \text{(case)}$$

Curry-Howard isomorphism

- Given a well-typed closed term, take the typing derivation, erase the terms, and have a propositional-logic proof
- Given a propositional-logic proof, there exists a closed term with that type
- A term that type-checks is a proof — it tells you exactly how to derive the logic formula corresponding to its type
- Constructive *(hold that thought)* propositional logic and simply-typed lambda-calculus with pairs and sums are the same thing.
  - Computation and logic are deeply connected
  - λ is no more or less made up than implication

Revisit our examples: “terms are proofs”

$\lambda x{:}\ \tau.\ x$

is a proof that

$\tau \Rightarrow \tau$

Revisit our examples: "terms are proofs"

$$\lambda x{:}\ \tau.\ \lambda f{:}\ \tau \rightarrow \sigma.\ f\ x$$

is a proof that

$$\tau \Rightarrow (\tau \Rightarrow \sigma) \Rightarrow \sigma$$

Revisit our examples: “terms are proofs”

λf: $\tau \to \sigma \to \rho$. λx: σ. λy: τ. f y x

is a proof that

$(\tau \Rightarrow \sigma \Rightarrow \rho) \Rightarrow \sigma \Rightarrow \tau \Rightarrow \rho$

Revisit our examples: “terms are proofs”

λx: τ. <left x, left x>

is a proof that

$$\tau \Rightarrow ((\tau \vee \sigma) \wedge (\tau \vee \rho))$$

Revisit our examples: "terms are proofs"

$$\lambda f\!: \tau \to \rho.\ \lambda g\!: \sigma \to \rho.\ \lambda x\!: \tau + \sigma.\ (\text{case x do f g})$$

is a proof that

$$(\tau \Rightarrow \rho) \Rightarrow (\sigma \Rightarrow \rho) \Rightarrow (\tau \vee \sigma) \Rightarrow \rho$$

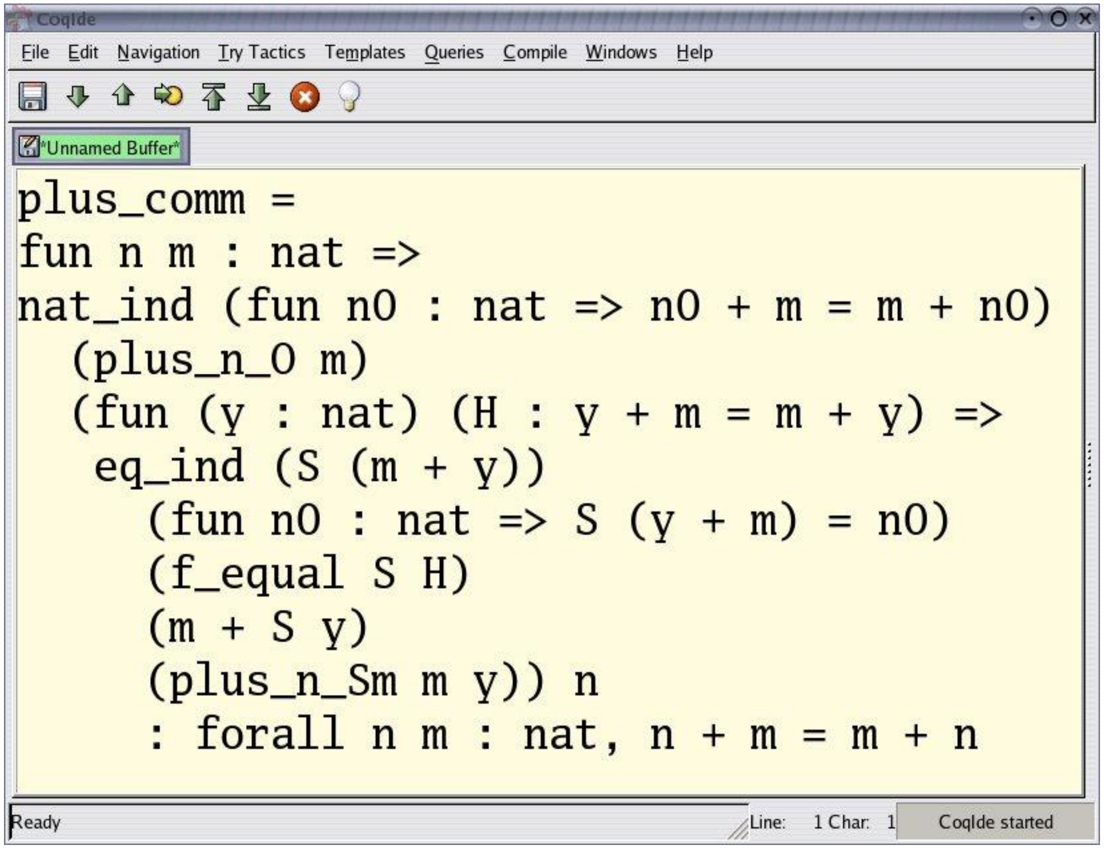
Revisit our examples: "terms are proofs"

λx: $\tau \times \sigma$. λy: ρ. < <y, proj1 x>, proj2 x >

is a proof that

$$(\tau \wedge \sigma) \Rightarrow \rho \Rightarrow ((\rho \wedge \tau) \wedge \sigma)$$

Coq example: proof can be written as functional program



Proof of commutativity of addition on nat in Coq.

Why care?

Because:

- This is just fascinating
- Don't think of logic and computing as distinct fields
- Thinking "the other way" can help you know what's possible/impossible
- Can form the basis for automated theorem provers
- Type systems should not be *ad hoc* piles of rules!

So, every typed λ-calculus is a proof system for some logic...

Is STLC with pairs and sums a *complete* proof system for propositional logic? Almost...

Classical vs. constructive

- Classical propositional logic has the "law of the excluded middle":

$$\overline{\Gamma \ \vdash p \lor (p \Rightarrow q)}$$

Think "$p \ \lor \neg p$"

- STLC does not support it: e.g. no closed term has type $\rho + (\rho \to \sigma)$

- Logics without this rule are called "constructive" or "intuitionistic".
  - Formulae are *only* considered "true" when we have direct evidence ("proofs produce examples")

Example classical proof

- Theorem: There exist two irrational numbers a and b such that a^b is rational.
- Can be proved using “the law of exclusive middle”.
  - It’s known that $\sqrt{2}$ is irrational.
  - Consider the number $\sqrt{2}^{\sqrt{2}}$.
    - If it is rational, the proof is complete, and $a = b = \sqrt{2}$.
    - If it is irrational, then let $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$. Then $a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\times\sqrt{2}} = (\sqrt{2})^2 = 2$, and the proof is complete.
- Constructive logics would not accept this argument

Classical vs. constructive

- In constructive logics, “branch on possibilities” by making “excluded middle” an explicit assumption:

$$(p \vee (p \Rightarrow q)) \wedge (p \Rightarrow r) \wedge \big((p \Rightarrow q) \Rightarrow r\big) \Rightarrow r$$

- “if any number is either rational or irrational, then there exist two irrational numbers a and b such that a^b is rational”

What about “fix”?

- A “non-terminating proof” is no proof at all
- Remember the typing rule

$$\frac{\Gamma \vdash M : \tau \to \tau}{\Gamma \vdash \mathbf{fix}\, M : \tau} \text{ (fix)}$$

- It lets us prove anything! E.g. **fix** $\lambda x{:}\tau.\ x$ has type τ
- So the “logic” is inconsistent

Last word on Curry-Howard

- Not just constructive propositional logic & STLC
- ***Every*** logic has a corresponding typed system
  - Classical logics
  - Inconsistent logics

- If you remember one thing:

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}\text{(app)} \quad \Leftrightarrow \quad \frac{\Gamma \vdash p \Rightarrow q \qquad \Gamma \vdash p}{\Gamma \vdash q}(\Rightarrow\text{-elim})$$