The LF Logical Framework

Robert J. Simmons
Microsoft Research - August 3, 2007

PART 0:
the framework

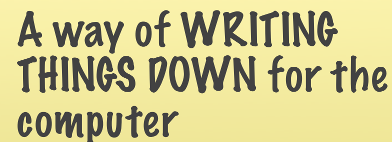
the name of the game is "abstraction"

a framework

is an

abstraction...

a framework is an abstraction language



a framework
is an
abstraction
machine

We want to RUN our abstractions!

Computational thinking

Process algebras?

a framework
is an
abstraction
thing?

a framework
is an
abstraction
landscape

landscape

landscape
suggests
directions

landscape
determines
difficulty

landscape
determines
difficulty

Closures in C

landscape
determines
difficulty

BDDs in Pure Haskell

landscape
determines
difficulty

a framework
is an
abstraction
landscape

in a logical framework,

we build the landscape using logic

ML

ML

abstract state machines

ML

abstract state machines

Maude

ML

abstract state machines

Maude

Coq

ML

abstract state machines

Maude

Coq

Isabelle/HOL

PART 1: overview

Proofs and types in ML

Proofs and types in ML

Proofs and types in LF

Proofs and types in ML

Proofs and types in LF

Representing systems in LF/Twelf

Proofs and types in ML

Proofs and types in LF

Representing systems in LF/Twelf

What Twelf is good at

Understand a Twelf proof

Understand a Twelf proof

Believe a Twelf proof

Understand a Twelf proof

Believe a Twelf proof

Start writing them yourself

Understand a Twelf proof

Believe a Twelf proof

Start writing them yourself

http://twelf.plparty.org/

PART 2:
proofs & types

PART 2:
proofs & types
or
(whoo functional programming!)

PART 2:
proofs & types

First I wished to construct a formalism that comes as close as possible to actual reasoning. Thus arose a "calculus of natural deduction".

Genzen, 1935

As opposed to propositional calculus!

Natural Deduction

A1&A2

A1&A2 true

A1&A2 true

$$\frac{\texttt{A1 true}}{\texttt{A1\&A2 true}}$$

$$\frac{\texttt{A1 true} \qquad \texttt{A2 true}}{\texttt{A1\&A2 true}}$$

$$\frac{A1\ \text{true} \qquad A2\ \text{true}}{A1\&A2\ \text{true}}$$

A1&A2 true

A1 true A2 true

___

A1&A2 true

A1&A2 true

___

$$\frac{A1\ \texttt{true} \qquad A2\ \texttt{true}}{A1\&A2\ \texttt{true}}$$

$$\frac{A1\&A2\ \texttt{true}}{A1\ \texttt{true}}$$

$$\frac{\texttt{A1 true} \qquad \texttt{A2 true}}{\texttt{A1\&A2 true}}$$

$$\frac{\texttt{A1\&A2 true}}{\texttt{A1 true}}$$

$$\frac{\texttt{A1\&A2 true}}{\texttt{A2 true}}$$

$$\frac{\Gamma \vdash A1 \ \textsf{true} \qquad \Gamma \vdash A2 \ \textsf{true}}{\Gamma \vdash A1 \& A2 \ \textsf{true}}$$

$$\frac{\Gamma \vdash A1 \& A2 \ \textsf{true}}{\Gamma \vdash A1 \ \textsf{true}}$$

$$\frac{\Gamma \vdash A1 \& A2 \ \textsf{true}}{\Gamma \vdash A2 \ \textsf{true}}$$

Γ is a set of ass s

$\Gamma =$

{A true, B true, etc...}

Γ is a set of assumptions

$\Gamma =$

{A true, B true, etc...}

A true $\in \Gamma$

Γ is a set of assumptions

$$\Gamma = \{A \text{ true}, B \text{ true}, \text{etc...}\}$$

"If you're assuming something,
then you can prove it!"

$$\frac{A \text{ true} \in \Gamma}{\Gamma \vdash A \text{ true}}$$

$\Gamma \vdash$ A->B true

$$\frac{}{\Gamma \vdash A \to B \ \texttt{true}}$$

$$\frac{B \ \texttt{true}}{\Gamma \vdash A \texttt{->} B \ \texttt{true}}$$

$$\frac{\Gamma, A\ \texttt{true} \vdash B\ \texttt{true}}{\Gamma \vdash A\texttt{->}B\ \texttt{true}}$$

$$\frac{\Gamma, A\ \texttt{true} \vdash B\ \texttt{true}}{\Gamma \vdash A\texttt{->}B\ \texttt{true}}$$

$$\frac{\Gamma \vdash A\ \texttt{true} \quad \Gamma \vdash A\texttt{->}B\ \texttt{true}}{\Gamma \vdash B\ \texttt{true}}$$

Example

$\vdash$ `A&B->(C->A)` true

Example

$$\frac{}{\vdash A\&B \to (C \to A)\ \text{true}}$$

Example

$$\frac{\texttt{A\&B true} \vdash \texttt{C->A true}}{\vdash \texttt{A\&B->(C->A) true}}$$

Example

$$\dfrac{\dfrac{\texttt{A\&B true, C true} \vdash \texttt{A true}}{\texttt{A\&B true} \vdash \texttt{C->A true}}}{\vdash \texttt{A\&B->(C->A) true}}$$

Example

$$\dfrac{\dfrac{\dfrac{\texttt{A\&B true, C true} \vdash \texttt{A\&B true}}{\texttt{A\&B true, C true} \vdash \texttt{A true}}}{\texttt{A\&B true} \vdash \texttt{C->A true}}}{\vdash \texttt{A\&B->(C->A) true}}$$

Example

$$\dfrac{\dfrac{\dfrac{\dfrac{\texttt{A\&B true} \in \{\texttt{A\&B true}, \texttt{C true}\}}{\texttt{A\&B true}, \texttt{C true} \vdash \texttt{A\&B true}}}{\texttt{A\&B true}, \texttt{C true} \vdash \texttt{A true}}}{\texttt{A\&B true} \vdash \texttt{C->A true}}}{\vdash \texttt{A\&B->(C->A) true}}$$

Example

$$\dfrac{\dfrac{\dfrac{\dfrac{A\&B\ \mathtt{true} \in \{A\&B\ \mathtt{true}, C\ \mathtt{true}\}}{A\&B\ \mathtt{true}, C\ \mathtt{true} \vdash A\&B\ \mathtt{true}}}{A\&B\ \mathtt{true}, C\ \mathtt{true} \vdash A\ \mathtt{true}}}{A\&B\ \mathtt{true} \vdash C\text{->}A\ \mathtt{true}}}{\vdash A\&B\text{->}(C\text{->}A)\ \mathtt{true}}$$

Natural Deduction

Natural Sleep Aid?

Natural Deduction

Natural Deduction

(Aliens kidnap Harry Potter on page 573!)

Natural Deduction

Natural Deduction
gives us
Type Systems

Proof

$$\frac{\Gamma \vdash \mathtt{A1}\ \mathtt{true} \qquad \Gamma \vdash \mathtt{A2}\ \mathtt{true}}{\Gamma \vdash \mathtt{A1\&A2}\ \mathtt{true}}$$

$$\frac{\Gamma \vdash m{:}A1\ \text{true} \qquad \Gamma \vdash A2\ \text{true}}{\Gamma \vdash A1\&A2\ \text{true}}$$

$$\frac{\Gamma \vdash \mathtt{m:A1\ true} \qquad \Gamma \vdash \mathtt{n:A2\ true}}{\Gamma \vdash \mathtt{A1\&A2\ true}}$$

$$\frac{\Gamma \vdash m : A1\ \mathtt{true} \qquad \Gamma \vdash n : A2\ \mathtt{true}}{\Gamma \vdash (m,n) : A1 \& A2\ \mathtt{true}}$$

$$\frac{\Gamma \vdash m : A1\ \mathtt{true} \qquad \Gamma \vdash n : A2\ \mathtt{true}}{\Gamma \vdash (m,n) : A1 \& A2\ \mathtt{true}}$$

$$\Gamma \vdash m : A1 \& A2\ \mathtt{true}$$

$$\frac{\Gamma \vdash m : A1 \ \text{true} \qquad \Gamma \vdash n : A2 \ \text{true}}{\Gamma \vdash (m, n) : A1 \& A2 \ \text{true}}$$

$$\frac{\Gamma \vdash m : A1 \& A2 \ \text{true}}{}$$

$$\frac{\Gamma \vdash \texttt{m}:\texttt{A1}\ \texttt{true} \qquad \Gamma \vdash \texttt{n}:\texttt{A2}\ \texttt{true}}{\Gamma \vdash \texttt{(m,n)}:\texttt{A1\&A2}\ \texttt{true}}$$

$$\frac{\Gamma \vdash \texttt{m}:\texttt{A1\&A2}\ \texttt{true}}{\Gamma \vdash \texttt{fst m}:\texttt{A1}\ \texttt{true}}$$

$$\frac{\Gamma \vdash \mathtt{m}:\mathtt{A1}\ \mathtt{true} \qquad \Gamma \vdash \mathtt{n}:\mathtt{A2}\ \mathtt{true}}{\Gamma \vdash (\mathtt{m},\mathtt{n}):\mathtt{A1\&A2}\ \mathtt{true}}$$

$$\frac{\Gamma \vdash \mathtt{m}:\mathtt{A1\&A2}\ \mathtt{true}}{\Gamma \vdash \mathtt{fst\ m}:\mathtt{A1}\ \mathtt{true}}$$

$$\frac{\Gamma \vdash \mathtt{m}:\mathtt{A1\&A2}\ \mathtt{true}}{\Gamma \vdash \mathtt{snd\ m}:\mathtt{A2}\ \mathtt{true}}$$

$$\frac{\Gamma \vdash m : A1 \qquad \Gamma \vdash n : A2}{\Gamma \vdash (m, n) : A1 \& A2}$$

$$\frac{\Gamma \vdash m : A1 \& A2}{\Gamma \vdash \mathtt{fst}\ m : A1}$$

$$\frac{\Gamma \vdash m : A1 \& A2}{\Gamma \vdash \mathtt{snd}\ m : A2}$$

$$\frac{\Gamma \vdash m : A1 \qquad \Gamma \vdash n : A2}{\Gamma \vdash (m, n) : A1 * A2}$$

$$\frac{\Gamma \vdash m : A1 * A2}{\Gamma \vdash \mathtt{fst}\ m : A1}$$

$$\frac{\Gamma \vdash m : A1 * A2}{\Gamma \vdash \mathtt{snd}\ m : A2}$$

$$\frac{\Gamma, A \ \texttt{true} \vdash B \ \texttt{true}}{\Gamma \vdash A\texttt{->}B \ \texttt{true}}$$

$$\frac{\Gamma \vdash A \ \texttt{true} \qquad \Gamma \vdash A\texttt{->}B \ \texttt{true}}{\Gamma \vdash B \ \texttt{true}}$$

$$\frac{\Gamma, A \ \texttt{true} \vdash \texttt{m:B}}{\Gamma \vdash \texttt{A->B} \ \texttt{true}}$$

$$\frac{\Gamma \vdash \texttt{A} \ \texttt{true} \qquad \Gamma \vdash \texttt{A->B} \ \texttt{true}}{\Gamma \vdash \texttt{B} \ \texttt{true}}$$

$$\frac{\Gamma, \mathtt{x:A} \vdash \mathtt{m:B}}{\Gamma \vdash \mathtt{A\text{->}B\ true}}$$

$$\frac{\Gamma \vdash \mathtt{A\ true} \qquad \Gamma \vdash \mathtt{A\text{->}B\ true}}{\Gamma \vdash \mathtt{B\ true}}$$

$$\frac{\Gamma, x : A \vdash m : B}{\Gamma \vdash \lambda x.m : A \to B}$$

$$\frac{\Gamma \vdash A \text{ true} \quad \Gamma \vdash A \to B \text{ true}}{\Gamma \vdash B \text{ true}}$$

$$\frac{\Gamma,\ x : A \vdash m : B}{\Gamma \vdash \lambda x . m : A \to B}$$

$$\frac{\Gamma \vdash m : A \qquad \Gamma \vdash n : A \to B}{\Gamma \vdash B \text{ true}}$$

$$\frac{\Gamma,\ \mathtt{x : A} \vdash \mathtt{m : B}}{\Gamma \vdash \lambda\mathtt{x.m : A \text{->} B}}$$

$$\frac{\Gamma \vdash \mathtt{m : A} \qquad\qquad \Gamma \vdash \mathtt{n : A \text{->} B}}{\Gamma \vdash \mathtt{n\ m : B}}$$

PART 3:
The limit of ML

Goal: if types are propositions, then we want types to say as much as possible

```
module rob : sig




end
```

```
module rob : sig
  type clear
  type day
  type rob_walks
  type rob_flies




end
```

```
module rob : sig
  type clear
  type day
  type rob_walks
  type rob_flies

  val f1 : clear
  val f2 : day
  val f3 : clear -> day -> rob_walks
end
```

```
module rob : sig
  type clear
  type day
  type rob_walks
  type rob_flies

  val f1 : clear
  val f2 : day
  val f3 : clear -> day -> rob_walks
end

let pf : rob_walks
```

```
module rob : sig
  type clear
  type day
  type rob_walks
  type rob_flies

  val f1 : clear
  val f2 : day
  val f3 : clear -> day -> rob_walks
end

let pf : rob_walks = f3 f1 f2
   // I believe I'll walk today...
```

```
module rob : sig
  type clear
  type day
  type rob_walks
  type rob_flies

  val f1 : clear
  val f2 : day
  val f3 : clear -> day -> rob_walks
end

let pf : rob_flies
```

```
module rob : sig
   type clear
   type day
   type rob_walks
   type rob_flies

   val f1 : clear
   val f2 : day
   val f3 : clear -> day -> rob_walks
end

let pf : rob_flies =     //I believe I
    let rec f() = f() in f() // can fly
```

Problem 1: Unsound logic!

Problem 2: Propositional logic!

ML is doing

ML is doing "syntax"

ML is doing
"syntax"
not proofs...

```
module rob : sig
   type clear
   type day
   type rob_walks
   type rob_flies

   val f1 : clear
   val f2 : day
   val f3 : clear -> day -> rob_walks
end

let pf : rob_walks
```

```
module natural_numbers : sig
  type nat




end
```

```
module natural_numbers : sig
  type nat

  val z : nat
  val s : nat -> nat




end
```

```
module natural_numbers : sig
  type nat

  val z : nat
  val s : nat -> nat




end

let pf : (s(s z) + (s z) = (s(s(s z))))
```

```
module logic : sig
  type prop









end
```

```
module logic : sig
  type prop

  val true  : prop
  val false : prop
  val and   : prop -> prop -> prop
  val or    : prop -> prop -> prop
  val not   : prop -> prop

end
```

```
module logic : sig
  type prop

  val true  : prop
  val false : prop
  val and   : prop -> prop -> prop
  val or    : prop -> prop -> prop
  val not   : prop -> prop

end

let pf : ((not true) or true) is_true
```

We need types

```
let pf1: rob_walks
let pf2: (s(s z) + (s z) = (s(s(s z))))
let pf3: ((not true) or true) is_true
```

We need rules

```
let pf1: rob_walks
let pf2: (s(s z) + (s z) = (s(s(s z))))
let pf3: ((not true) or true) is_true
```

We need dependent types

PART 4:
dependent types

creating types

example:
unary addition

$$\frac{}{z : nat}$$

```
type nat
val  z : nat
val  s : nat -> nat
```

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{z : nat}$$

$$\frac{N : nat}{s(N) : nat}$$

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{\texttt{nat}} \qquad \frac{\texttt{nat}}{\texttt{nat}}$$

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{z} \qquad \frac{N}{s(N)}$$

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{z:nat}$$

$$\frac{N:nat}{s(N):nat}$$

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{z:nat}$$

$$\frac{N:nat}{s(N):nat}$$

$$\frac{}{z+N=N}$$

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{z : nat}$$

$$\frac{N : nat}{s(N) : nat}$$

$$\frac{}{z + N = N}$$

$$\frac{N + M = P}{(s\ N) + M = (s\ P)}$$

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{z:nat}$$

$$\frac{N:nat}{s(N):nat}$$

$$\frac{}{\text{plus } z\ N\ N}$$

$$\frac{N+M=P}{(s\ N)+M=(s\ P)}$$

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{z{:}nat} \qquad \frac{N{:}nat}{s(N){:}nat}$$

$$\frac{}{\texttt{plus}\ z\ N\ N}$$

$$\frac{\texttt{plus}\ N\ M\ P}{\texttt{plus}\ (s\ N)\ M\ (s\ P)}$$

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{z{:}nat}$$

$$\frac{N{:}nat}{s(N){:}nat}$$

```
plus z z z
```

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{z{:}nat}$$

$$\frac{N{:}nat}{s(N){:}nat}$$

plus z z z
is
something I can
prove w/ these rules

$$\frac{}{z : nat}$$

$$\frac{N : nat}{s(N) : nat}$$

```
type nat
val  z : nat
val  s : nat -> nat
```

plus z z z
is
a type!

```
type nat
val  z : nat
val  s : nat -> nat
```

$$\frac{}{z : nat}$$

$$\frac{N : nat}{s(N) : nat}$$

plus z z z
is
a type
with terms in it!

```
type nat
val  z : nat
val  s : nat -> nat
```

```
type nat
val  z : nat
val  s : nat -> nat

type {nat}{nat}{nat}plus
```

```
nat : type.
z : nat.
s : nat -> nat.

type {nat}{nat}{nat}plus

//Too close for missiles...
```

```
nat : type.
z : nat.
s : nat -> nat.

type {nat}{nat}{nat}plus

//Too close for missiles...
//I'm switching to Twelf
```

```
nat : type.
z : nat.
s : nat -> nat.

plus:
 {a:nat}{b:nat}{c:nat} type.
```

```
nat : type.
z : nat.
s : nat -> nat.

plus:
 {a:nat}{b:nat}{c:nat} type.
 Πa:nat. Πb:nat. Πc:nat. type
```

creating types

creating rules

$$\frac{\Gamma, x : A \vdash m : B}{\Gamma \vdash \lambda x . m : A \rightarrow B}$$

$$\frac{\Gamma \vdash n : A \rightarrow B \qquad \Gamma \vdash m : A}{\Gamma \vdash n\, m : B}$$

$$\frac{\Gamma,\ x : A \vdash m : B}{\Gamma \vdash \lambda x . m : \{x : A\} B}$$

$$\frac{\Gamma \vdash n : \quad A \texttt{->} B \qquad \Gamma \vdash m : A}{\Gamma \vdash n\ m : B}$$

$$\frac{\Gamma,\ x : A \vdash m : B}{\Gamma \vdash \lambda x\,.\,m : \{x : A\}B}$$

$$\frac{\Gamma \vdash n : \{x : A\}B \qquad \Gamma \vdash m : A}{\Gamma \vdash n\ m : B}$$

$$\frac{\Gamma,\ \mathtt{x:}\mathrm{A} \vdash \mathtt{m:}\mathrm{B}}{\Gamma \vdash \lambda\mathtt{x.m:}\{\mathtt{x:}\mathrm{A}\}\mathrm{B}}$$

$$\frac{\Gamma \vdash \mathtt{n:}\{\mathtt{x:}\mathrm{A}\}\mathrm{B} \qquad \Gamma \vdash \mathtt{m:}\mathrm{A}}{\Gamma \vdash \mathtt{n\ m:}\mathrm{B}[\mathtt{m/x}]}$$

```
nat : type.
z : nat.
s : nat -> nat.

plus: {a:nat}{b:nat}{c:nat} type.
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
```

```
______________
 plus z N N
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat}
```

```
plus z N N
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
```

$$\frac{}{\texttt{plus z N N}}$$

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:
```

$$\frac{\texttt{plus N M P}}{\texttt{plus (s N) M (s P)}}$$

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
```

```
         plus N M P
   ---------------------------
   plus (s N) M (s P)
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
      plus N M P
```

$$\frac{\texttt{plus N M P}}{\texttt{plus (s N) M (s P)}}$$

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
     plus N M P
       -> plus (s N) M (s P).
```

$$\frac{\texttt{plus N M P}}{\texttt{plus (s N) M (s P)}}$$

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
     plus N M P
       -> plus (s N) M (s P).
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
     plus N M P
       -> plus (s N) M (s P).
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
     plus N M P
       -> plus (s N) M (s P).
```

```
p0: plus z (s z) (s z)
      = pz (s z).
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
     plus N M P
       -> plus (s N) M (s P).

p0: plus z (s z) (s z) = pz (s z).
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
     plus N M P
       -> plus (s N) M (s P).

p0: plus z (s z) (s z) = pz (s z).
```

```
p1: plus (s z) (s z) (s(s z))
        = ps z (s z) (s z) p0.
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
     plus N M P
       -> plus (s N) M (s P).

p0: plus z (s z) (s z) = pz (s z).
p1: plus (s z) (s z) (s(s z)) = ps z (s z) (s z) p0.
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
     plus N M P
       -> plus (s N) M (s P).

p0: plus z (s z) (s z) = pz (s z).
p1: plus (s z) (s z) (s(s z)) = ps z (s z) (s z) p0.
```

```
p2: plus (s(s z)) (s z) (s(s(s z)))
   = ps (s z) (s z) (s(s z)) p1.
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz:{N:nat} plus z N N.
ps:{N:nat}{M:nat}{P:nat}
     plus N M P
       -> plus (s N) M (s P).

p0: plus z (s z) (s z) = pz (s z).
p1: plus (s z) (s z) (s(s z)) = ps z (s z) (s z) p0.
p2: plus (s(s z)) (s z) (s(s(s z)))
     = ps (s z) (s z) (s(s z)) p1.
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz: plus z N N.
ps: plus N M P -> plus (s N) M (s P).


p0: plus z (s z) (s z) = pz (s z).
p1: plus (s z) (s z) (s(s z)) = ps z (s z) (s z) p0.
p2: plus (s(s z)) (s z) (s(s(s z)))
     = ps (s z) (s z) (s(s z)) p1.
```

```
nat : type.
z : nat.
s : nat -> nat.

plus: nat -> nat -> nat -> type.
pz: plus z N N.
ps: plus N M P -> plus (s N) M (s P).


p0: plus z (s z) (s z) = pz.
p1: plus (s z) (s z) (s(s z)) = ps p0.
p2: plus (s(s z)) (s z) (s(s(s z))) = ps p1.
```

PART 5:
more twelf

twelf can do

twelf is good at

twelf is good at higher-order abstract syntax

```
exp: type.
tp: type.
```

```
exp: type.
tp: type.


unit: tp.
arrow:tp -> tp -> tp.
```

```
exp: type.
tp: type.


unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:
```

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:
```

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> var -> exp -> exp.
```

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.
```

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.
```

binding in the framework

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.
```

binding in the framework
binding in the language

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

steps-to: exp -> exp -> type.
```

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

steps-to: exp -> exp -> type.
```

$$\frac{}{((\lambda x{:}t.e1)\ e2)\ \texttt{steps-to}\ ([e2/x]e1)}$$

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

steps-to: exp -> exp -> type.

s-lam-app: steps-to (app (lam E1) E2) (E1 E2).
```

$$\frac{}{((\lambda x{:}t.e1)\ e2)\ \texttt{steps-to}\ ([e2/x]e1)}$$

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

steps-to: exp -> exp -> type.

s-lam-app: steps-to (app (lam E1) E2) (E1 E2).
```

substitution in the framework

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

steps-to: exp -> exp -> type.

s-lam-app: steps-to (app (lam E1) E2) (E1 E2).
```

substitution in the framework

substitution in the language

substitution
is fundamental

the framework provides substitution

```
exp: type.
tp: type.
var: type.

unit:  tp.
arrow:tp -> tp -> tp.
app:   exp -> exp -> exp.
lam:   tp -> (exp -> exp) -> exp.

of: exp -> tp -> type.
```

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

of: exp -> tp -> type.
```

$\Gamma \vdash$ of (lam T(λx.E)) (arrow T T2)

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

of: exp -> tp -> type.
```

$$\frac{}{\Gamma \vdash \texttt{of (lam T(}\lambda\texttt{x.E)) (arrow T T2)}}$$

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

of: exp -> tp -> type.
```

$$\frac{\Gamma, \texttt{of x T} \vdash \texttt{of E T2}}{\Gamma \vdash \texttt{of (lam T(}\lambda\texttt{x.E)) (arrow T T2)}}$$

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

of: exp -> tp -> type.

of-lam: ({x: exp}                              )
```

$$\frac{\Gamma, \texttt{of x T} \vdash \texttt{of E T2}}{\Gamma \vdash \texttt{of (lam T(}\lambda\texttt{x.E)) (arrow T T2)}}$$

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

of: exp -> tp -> type.

of-lam: ({x: exp} of x T                    )
```

$$\frac{\Gamma, \texttt{of x T} \vdash \texttt{of E T2}}{\Gamma \vdash \texttt{of (lam T(}\lambda\texttt{x.E)) (arrow T T2)}}$$

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

of: exp -> tp -> type.

of-lam: ({x: exp} of x T -> of (E x) T2)
```

$$\frac{\Gamma, \texttt{of x T} \vdash \texttt{of E T2}}{\Gamma \vdash \texttt{of (lam T}(\lambda\texttt{x.E})\texttt{) (arrow T T2)}}$$

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

of: exp -> tp -> type.

of-lam: ({x: exp} of x T -> of (E x) T2)
          -> of (lam T E) (arrow T T2).
```

$$\frac{\Gamma, \texttt{of x T} \vdash \texttt{of E T2}}{\Gamma \vdash \texttt{of (lam T}(\lambda \texttt{x.E})\texttt{) (arrow T T2)}}$$

```
exp: type.
tp: type.
var: type.

unit: tp.
arrow:tp -> tp -> tp.
app:  exp -> exp -> exp.
lam:  tp -> (exp -> exp) -> exp.

of: exp -> tp -> type.

of-lam: ({x: exp} of x T -> of (E x) T2)
          -> of (lam T E) (arrow T T2).
```

context in the framework
context in the language

do ya’ trust me?

adequacy

adequacy

I mean what I say

PART 6: conclusion

LF can:

Define a system

Represent judgments
    of that system

LF can:

Define a system

Represent judgments of that system

“Do A and B add to C?”

LF can:
Define a system
Represent judgments
    of that system

"Do A and B add to C?"
"Does expression A have type T?"

LF can:
Define a system
Represent judgments
of that system

"Do A and B add to C?"
"Does expression A have type T?"
"Does expression A step to
expression B?"

Twelf can:

Define a system

Represent judgments
of that system

Prove properties
of that system

Twelf can:
Define a system
Represent judgments
    of that system
Prove properties
    of that system

“Is addition commutative?”

Twelf can:
Define a system
Represent judgments
of that system
Prove properties
of that system

“Is addition commutative?”
“Do all well-typed expressions
step to well-typed expressions?”

Twelf can:
Define a system
Represent judgments
of that system
Prove properties
of that system

"Is addition commutative?"
"Do all well-typed expressions
step to well-typed expressions?"
"Does my logic admit the 'cut' rule?"

Next Time