The LF Logical Framework

Robert J. Simmons

Microsoft Research - August 10, 2007

The Twelf Metalogical Framework

Robert J. Simmons
Microsoft Research - August 10, 2007

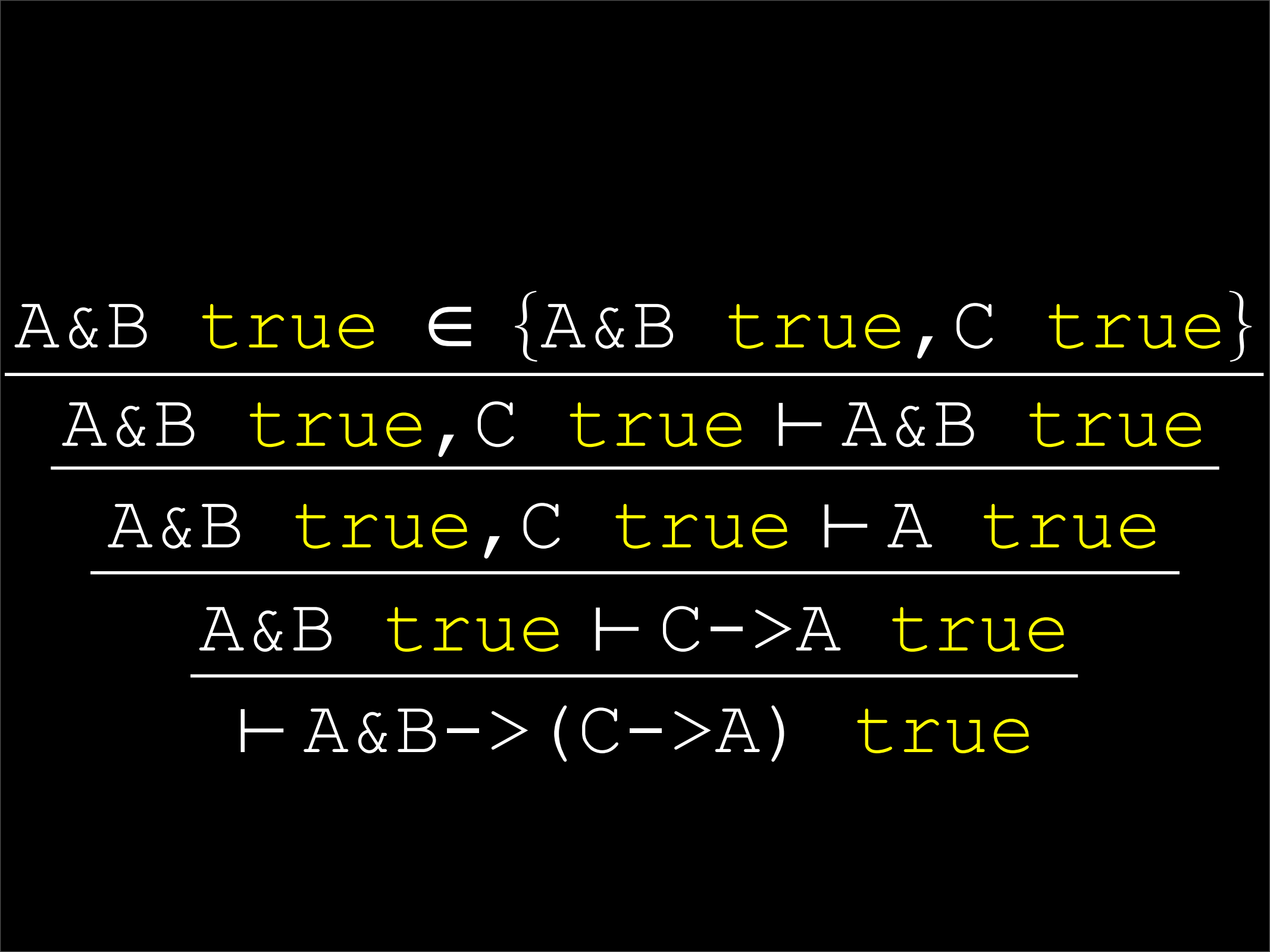
PART 0:
last time

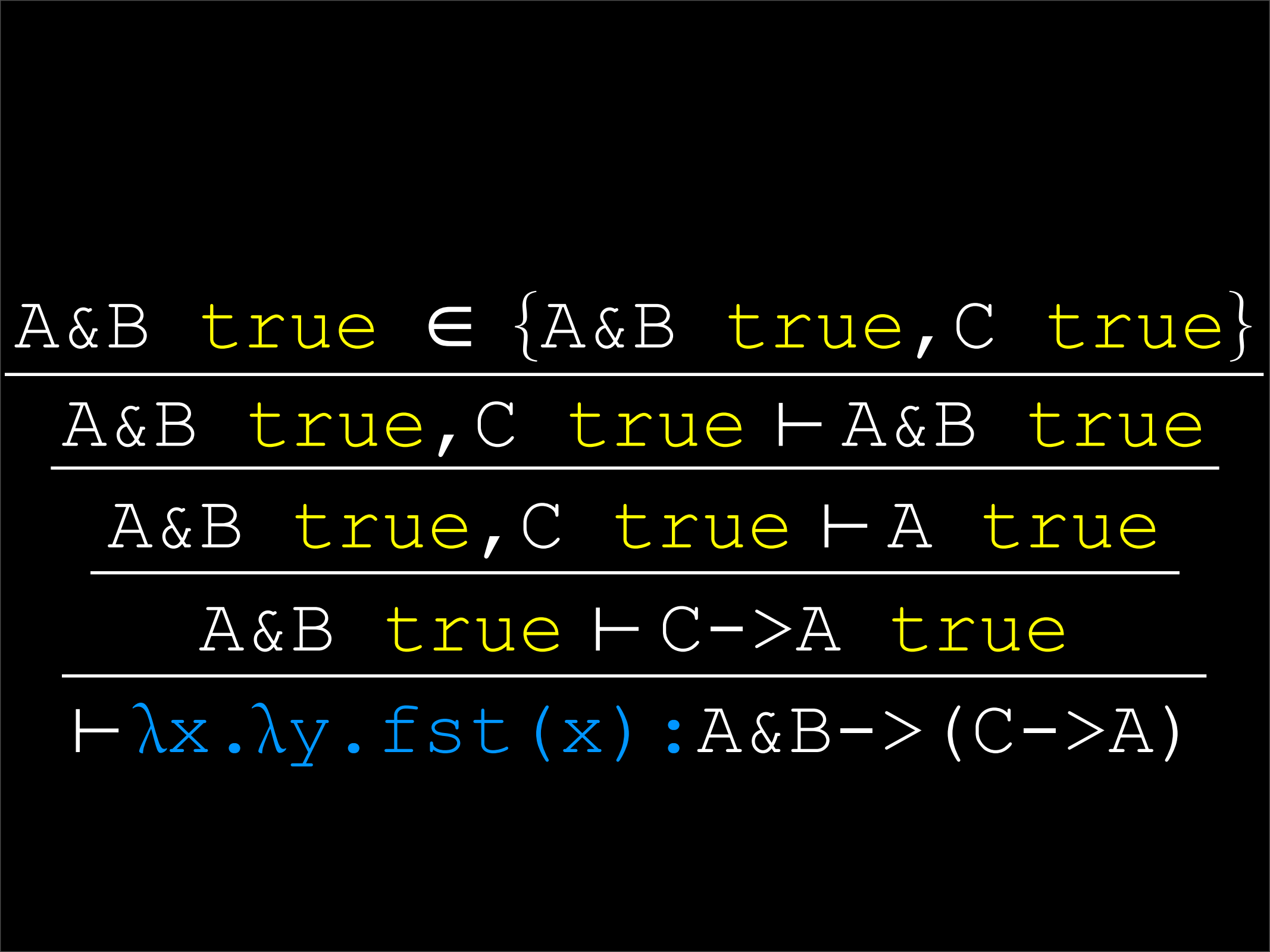
curry-howard

```
λx:A*B.λy:C.fst(x)
```

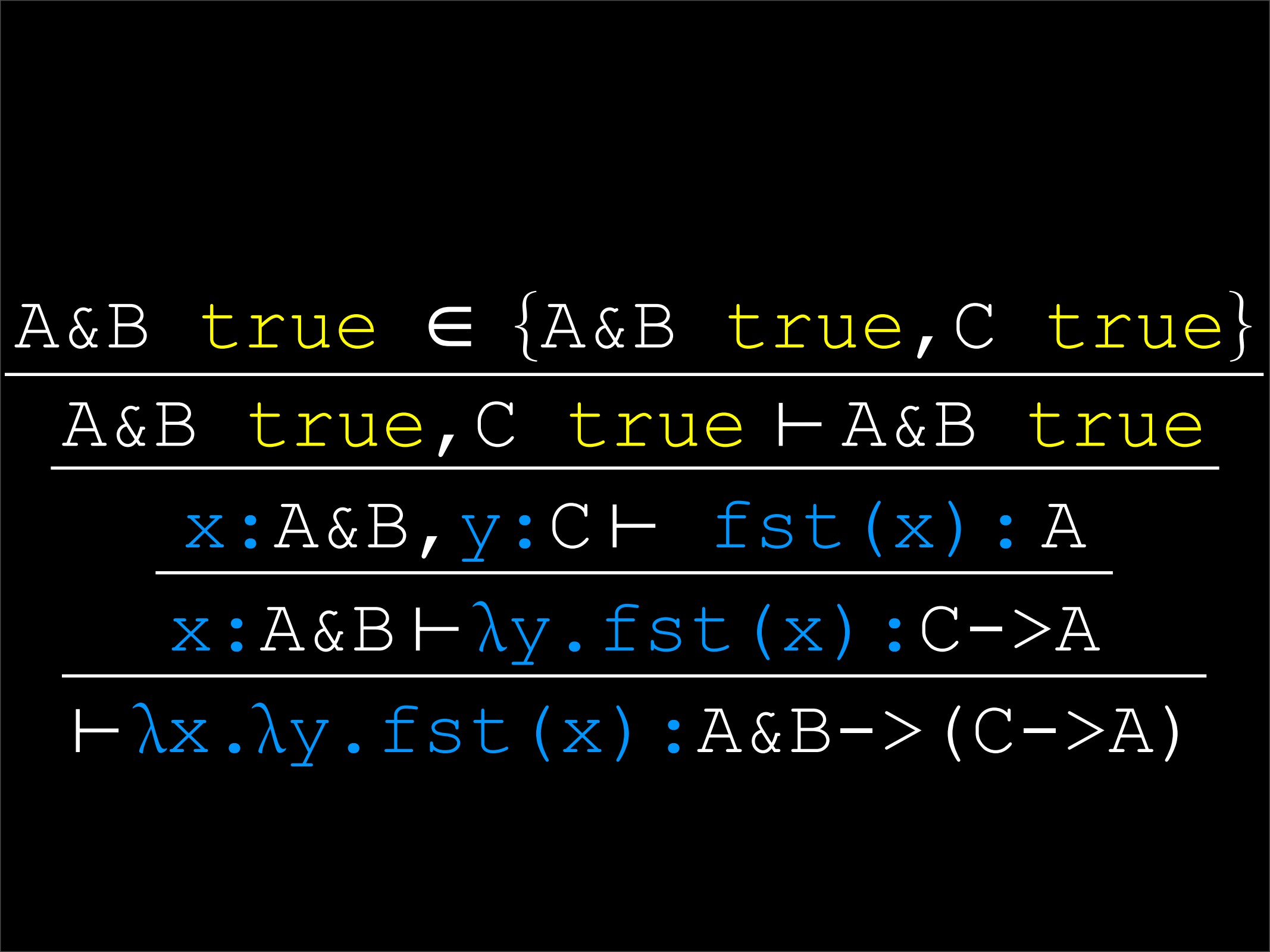
$$\lambda x{:}A*B.\lambda y{:}C.\mathrm{fst}(x)$$

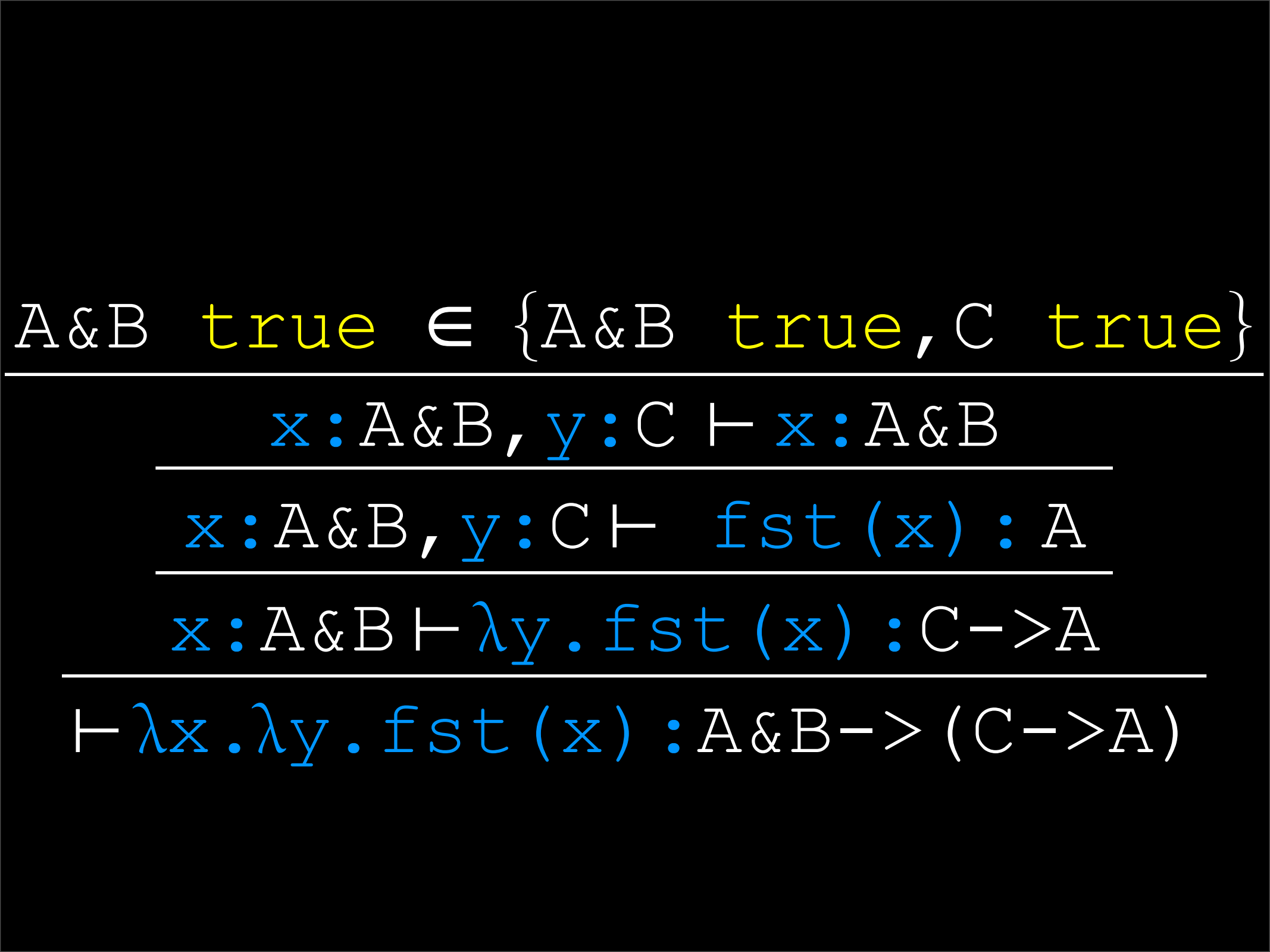
$$\dfrac{\dfrac{\dfrac{\dfrac{A\&B\ \mathrm{true} \in \{A\&B\ \mathrm{true}, C\ \mathrm{true}\}}{A\&B\ \mathrm{true}, C\ \mathrm{true} \vdash A\&B\ \mathrm{true}}}{A\&B\ \mathrm{true}, C\ \mathrm{true} \vdash A\ \mathrm{true}}}{A\&B\ \mathrm{true} \vdash C{->}A\ \mathrm{true}}}{\vdash A\&B{->}(C{->}A)\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

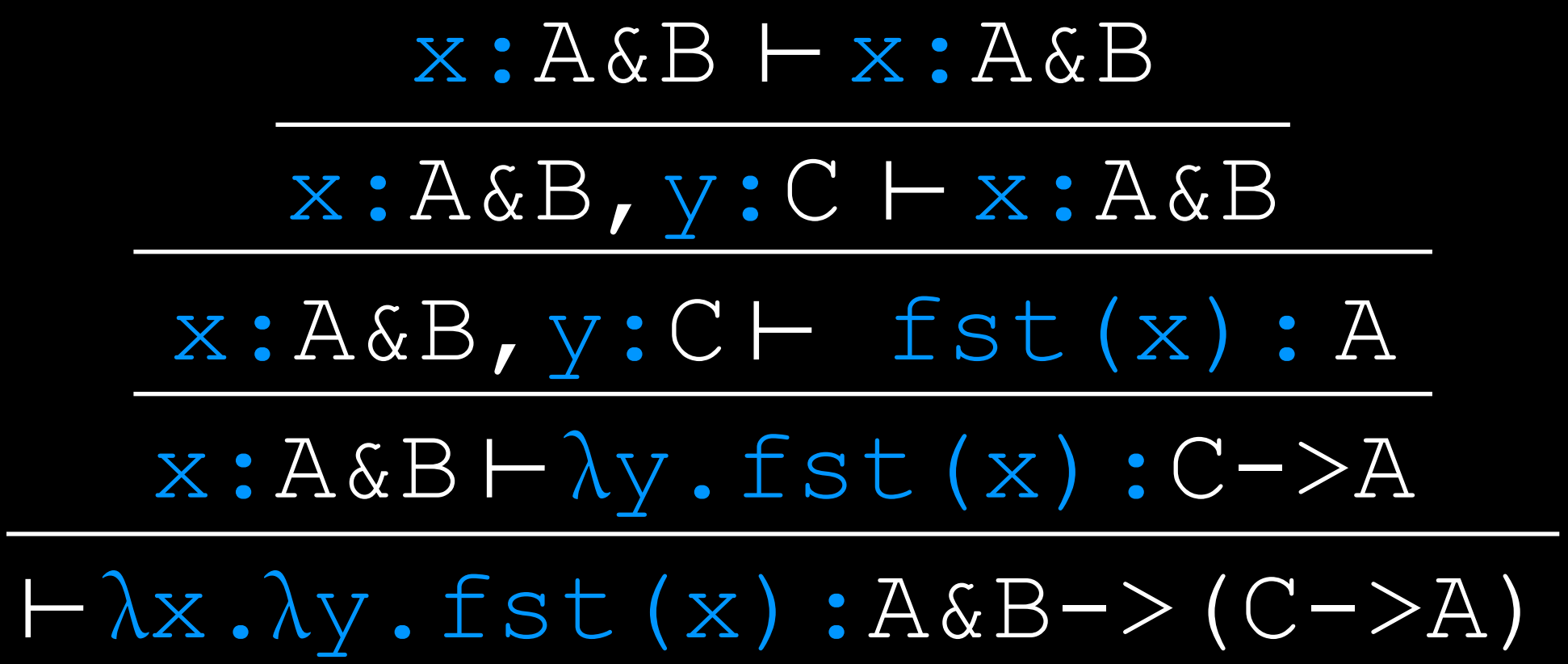

A&B true ∈ {A&B true,C true}
A&B true,C true ⊢ A&B true
A&B true,C true ⊢ A true
A&B true ⊢ C->A true
⊢ A&B->(C->A) true


A&B true ∈ {A&B true,C true}
A&B true,C true ⊢ A&B true
A&B true,C true ⊢ A true
A&B true ⊢ C->A true
⊢ λx.λy.fst(x) : A&B->(C->A)

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{A\&B\ true} \in \{\mathtt{A\&B\ true}, \mathtt{C\ true}\}}{\mathtt{A\&B\ true}, \mathtt{C\ true} \vdash \mathtt{A\&B\ true}}}{\mathtt{A\&B\ true}, \mathtt{C\ true} \vdash \mathtt{A\ true}}}{\mathtt{x}:\mathtt{A\&B} \vdash \lambda\mathtt{y}.\mathtt{fst}(\mathtt{x}):\mathtt{C}\text{->}\mathtt{A}}}{\vdash \lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{fst}(\mathtt{x}):\mathtt{A\&B}\text{->}(\mathtt{C}\text{->}\mathtt{A})}$$


A&B true ∈ {A&B true,C true}
A&B true,C true ⊢ A&B true
x:A&B,y:C⊢ fst(x):A
x:A&B⊢λy.fst(x):C->A
⊢λx.λy.fst(x):A&B->(C->A)


A&B true ∈ {A&B true, C true}
x:A&B, y:C ⊢ x:A&B
x:A&B, y:C ⊢ fst(x):A
x:A&B ⊢ λy.fst(x):C->A
⊢ λx.λy.fst(x):A&B->(C->A)


x:A&B ⊢ x:A&B
x:A&B, y:C ⊢ x:A&B
x:A&B, y:C ⊢ fst(x):A
x:A&B ⊢ λy.fst(x):C->A
⊢ λx.λy.fst(x):A&B->(C->A)

propositions as types

propositions
are
true

propositions
are
true if
the type is
inhabited

proofs in ML

```
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
```

```
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...
```

proofs in ML

proofs in ML
= bad methodology

proofs in 'ML'

proofs in 'ML'
too weak

proofs in 'ML'
too weak

(propositional logic)

dependent types

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

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

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

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

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

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

plus:
```

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

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

plus: nat -> nat -> nat?
```

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

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

plus: nat -> nat -> nat -> bool?
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
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).
```

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

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

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

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

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

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

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

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

$$\frac{\texttt{plus N M P}}{\texttt{plus (s N) M (s P)}} \qquad \frac{}{\texttt{pz:plus z N N}}$$

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

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

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

$$\frac{D : \mathrm{plus}\ N\ M\ P}{ps(D) : \mathrm{plus}\ (s\ N)\ M\ (s\ P)}$$

```
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).
```

```
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).

rel:
```

```
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).

rel:
   plus N M P -> plus M N P -> type.
```

PART 1:
a representation language

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

this is all you get

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

useful for encoding certain systems

warning: some handwaving ahead

```
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.


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

```
exp: type.
tp: type.


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

binding in the framework

```
exp: type.
tp: 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.


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

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

```
exp: type.
tp: 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.


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 T E1) E2) (E1 E2).
```

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

```
exp: type.
tp: 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 T E1) E2) (E1 E2).
```

substitution in the framework

```
exp: type.
tp: 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 T E1) E2) (E1 E2).
```

substitution in the framework

substitution in the language

substitution
is fundamental

the framework provides substitution

```
exp: type.
tp: type.


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

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

```
exp: type.
tp: type.


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

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

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

```
exp: type.
tp: type.


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

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

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

```
exp: type.
tp: type.


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

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

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

```
exp: type.
tp: type.


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

of: exp -> tp -> type.

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

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

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

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

```
exp: type.
tp: 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.


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.


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

this is a little complicated

theory of adequacy

theory of adequacy
(how to know it's
right)

the hard part is
the context

closed worlds

regular worlds

regular worlds (patterns describing the context)

PART 2:
programming in LF

nobody programs in S.T.L.C.

essentially
nobody programs in
LF directly

(almost)
nobody programs in
normalizing
languages

LF is a representation language

LF types
are the basis of a
language

LF types
are the basis of a
logic programming
language

logic programming

logic programming

= logic +

logic programming
= logic +
fixed search
strategy

prolog
= logic +
fixed search
strategy

prolog
= F.O.L. +
fixed search
strategy

prolog
= F.O.L. +
depth first
search

prolog

```
plus(z,N,N)
plus(s(N),M,s(P)) :- plus(N,M,P)
```

prolog

```
plus(z,N,N)
plus(s(N),M,s(P)) :- plus(N,M,P)

∀N.plus(z,N,N)
∀N.∀M.∀P.plus(N,M,P)
    -> plus(s(N),M,s(P))
```

prolog

```
plus(z,N,N)
plus(s(N),M,s(P)) :- plus(N,M,P)
?
```

prolog

```
plus(z,N,N)
plus(s(N),M,s(P)) :- plus(N,M,P)
? plus(s(s(z)),s(z),X)
```

prolog

```
plus(z,N,N)
plus(s(N),M,s(P)) :- plus(N,M,P)
? plus(s(s(z)),s(z),X)
- X = s(s(s(z)))
```

prolog

```
plus(z,N,N)
plus(s(N),M,s(P)) :- plus(N,M,P)
? plus(s(s(z)),q,X)
- X = s(s(q))
```

```
prolog

nat(z)
nat(s(N)) :- nat(N)

plus(z,N,N)
plus(s(N),M,s(P)) :- plus(N,M,P)
? plus(s(s(z)),q,X)
- X = s(s(q))
```

```
prolog

nat(z)
nat(s(N)) :- nat(N)

plus(z,N,N) :- nat(N)
plus(s(N),M,s(P)) :- plus(N,M,P)
? plus(s(s(z)),q,X)
No solution
```

```
twelf

nat(z)
nat(s(N)) :- nat(N)

plus(z,N,N)
plus(s(N),M,s(P)) :- plus(N,M,P)
```

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

plus(z,N,N)
plus(s(N),M,s(P)) :- plus(N,M,P)
```

```
twelf
nat: type.
z: nat.
s: nat -> nat.
plus: nat -> nat -> nat -> type.
plus(z,N,N)
plus(s(N),M,s(P)) :- plus(N,M,P)
```

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

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

%solve P: plus (s(s z)) (s z) X.
```

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

%solve P: plus (s(s z)) (s z) X.
P:plus (s(s z)) (s z) (s(s(s z)))=
    ps(ps(pz)).
```

PART 3:
program analysis

logic programming
in Twelf is
"pure"

ease of analysis emphasized

ease of analysis emphasized

(occasionally over expressiveness)

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

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

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

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

well-moded
(no unification vars)

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

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

%mode plus + + -.
```

well-moded
(no unification vars)

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

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

%mode plus + - +.
```

well-moded
(no unification vars)

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

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

%mode plus + - -.
```

well-moded
(no unification vars)

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

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

%mode plus + - -.
Constant pz.
Occurrence of variable N in output not
necessarily ground.
```

well-moded

(no unification vars)

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

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

%covers plus + + -.
```

coverage
(won't get stuck)

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

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

%covers plus + - +.
```

coverage
(won't get stuck)

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

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

%covers plus + - +.
Coverage error --- missing cases:
{X1:nat}{X2:nat} |- plus (s X1) X2 z.
```

coverage (won't get stuck)

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

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

termination
(won't run forever)

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

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

%terminates T (plus T _ _).
```

termination
(won't run forever)

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

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

%terminates T (plus _ T _).
```

termination
(won't run forever)

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

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

%terminates T (plus _ T _).
Termination violation:
 ---> (M) < (M)
```

termination
(won't run forever)

add it all up

add it all up:
totality
assertions

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

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

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

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

%mode plus +A +B -C.
```

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

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

%mode plus +A +B -C.
%worlds () (plus _ _ _).
```

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

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

%mode plus +A +B -C.
%worlds () (plus _ _ _).
%total T (plus T _ _).
```

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

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

%mode plus +A +B -C.
%worlds () (plus _ _ _).
%total T (plus T _ _).

rel:
   plus N M P -> plus M N P -> type.
```

PART 3:
proof by induction

addition
is commutative

For any natural numbers N,
there exists a derivation D:(plus N z N).

```
For any natural numbers N,
there exists a derivation D:(plus N z N).

Structural induction
on the natural number N.
```

For any natural numbers N,
there exists a derivation D:(plus N z N).

Structural induction
on the natural number N.

Case N = z
By rule pz, we have:

$$\frac{}{\text{pz: plus z z z}}$$

For any natural numbers N,
there exists a derivation D:(plus N z N).

Structural induction
on the natural number N.

Case N = z
By rule pz, we have:

$$\frac{}{\text{pz: plus z z z}}$$

Case N = s N2
By i.h. we have D:(plus N2 z N2)
By rule ps, we have:

$$\frac{\text{plus N2 z N2}}{\text{plus (s N2) z (s N2)}}$$

For any natural numbers N,
there exists a derivation D:(plus N z N).

Structural induction
on the natural number N.

Case N = z
By rule pz, we have:

$$\frac{}{\texttt{pz: plus z z z}}$$

Case N = s N2
By i.h. we have D:(plus N2 z N2)
By rule ps, we have:

$$\frac{\texttt{plus N2 z N2}}{\texttt{plus (s N2) z (s N2)}}$$

This covers all
cases. QED.

For any natural numbers N, M, and P
and derivations D1 of (plus N M P),
there exists
a derivation D2 of (plus N (s M) (s P))

For any natural numbers N, M, and P
and derivations D1 of (plus N M P),
there exists
a derivation D2 of (plus N (s M) (s P))

Structural induction
on the derivation D1.

For any natural numbers N, M, and P
and derivations D1 of (plus N M P),
there exists
a derivation D2 of (plus N (s M) (s P))

Structural induction
on the derivation D1.

Case D1 = $\dfrac{}{\text{plus } z\ N\ N}$

For any natural numbers N, M, and P
and derivations D1 of (plus N M P),
there exists
a derivation D2 of (plus N (s M) (s P))

Structural induction
on the derivation D1.

Case D1 = $\dfrac{}{\text{plus z N N}}$

By rule pz, we have: $\dfrac{}{\text{plus z (s N) (s N)}}$

For any natural numbers N, M, and P
and derivations D1 of (plus N M P),
there exists
a derivation D2 of (plus N (s M) (s P))

Structural induction
on the derivation D1.

Case D1 = $\dfrac{\text{plus } N\ M\ P}{\text{plus } (s\ N)\ M\ (s\ P)}$

For any natural numbers N, M, and P
and derivations D1 of (plus N M P),
there exists
a derivation D2 of (plus N (s M) (s P))

Structural induction
on the derivation D1.

$$\text{Case D1} = \frac{}{\text{plus N M P}}$$

$$\text{plus (s N) M (s P)}$$

By the i.h we have D2:(plus N (s M) (s P))
By rule ps we have:

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

For any natural numbers N, M, and P
and derivations D1 of (plus N M P),
there exists
a derivation D2 of (plus M N P)

For any natural numbers N, M, and P
and derivations D1 of (plus N M P),
there exists
a derivation D2 of (plus M N P)

Structural induction
on the natural number N.

For any natural numbers N, M, and P
and derivations D1 of (plus N M P),
there exists
a derivation D2 of (plus M N P)

Structural induction
on the natural number N.

Case N = z

By hypothesis we have D1:(plus z M P)
By rule inversion M = P.
By lemma we have D2:(plus M z M)

For any natural numbers N, M, and P
and derivations D1 of (plus N M P),
there exists
a derivation D2 of (plus M N P)

Structural induction
on the natural number N.

Case N = s N2

By hypothesis we have D1:(plus (s N2) M P)
By rule inversion P = (s P2) and
D1 = $\dfrac{\text{plus N2 M P2}}{\text{plus (s N2) M (s P2)}}$
By i.h. we have D2:(plus M N2 P2)
By lemma we have (plus M (s N2) (s P2))