A logical framework for incremental type-checking

Matthias Puech[1,2] Yann Régis-Gianas[2]

[1]Dept. of Computer Science, University of Bologna

[2]University Paris 7, CNRS, and INRIA, PPS, team πr^2

⋆ January 2011 ⋆

PPS – Groupe de travail théorie des types et réalisabilité

A paradoxical situation

Observation

We have powerful tools to mechanize the metatheory of (proof) languages

A paradoxical situation

Observation
We have powerful tools to mechanize the metatheory of (proof) languages

... And yet,

Workflow of programming and formal mathematics is still largely inspired by legacy software development (emacs, make, svn, diffs...)

A paradoxical situation

Observation

We have powerful tools to mechanize the metatheory of (proof) languages

. . . And yet,

Workflow of programming and formal mathematics is still largely inspired by legacy software development (emacs, make, svn, diffs. . .)

Isn't it time to make these tools metatheory-aware?

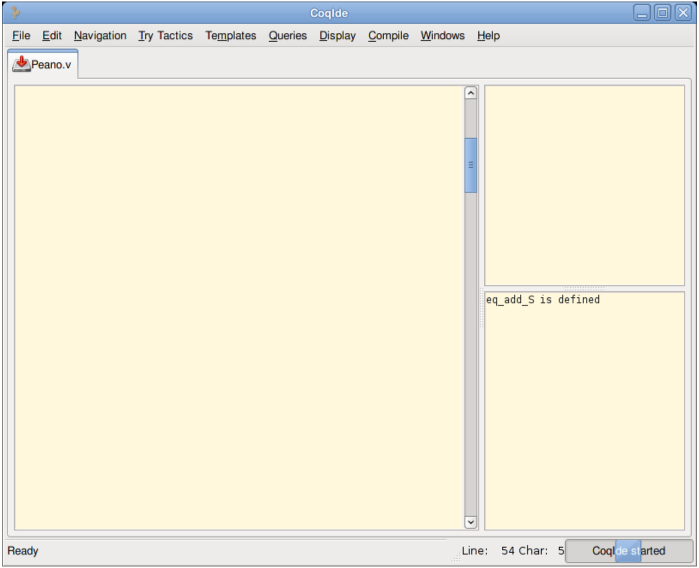
Incrementality in programming & proof languages

Q : Do you spend more time *writing* code or *editing* code?

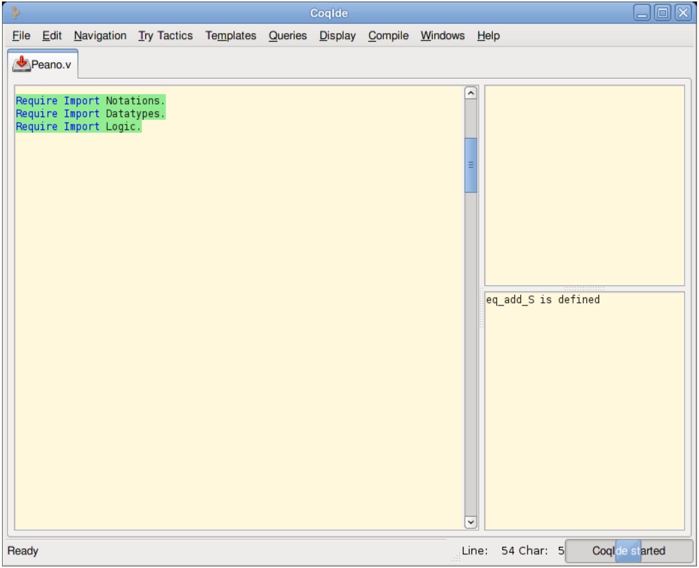
Today, we use:

- separate compilation
- dependency management
- version control on the scripts
- interactive toplevel with rollback (Coq)

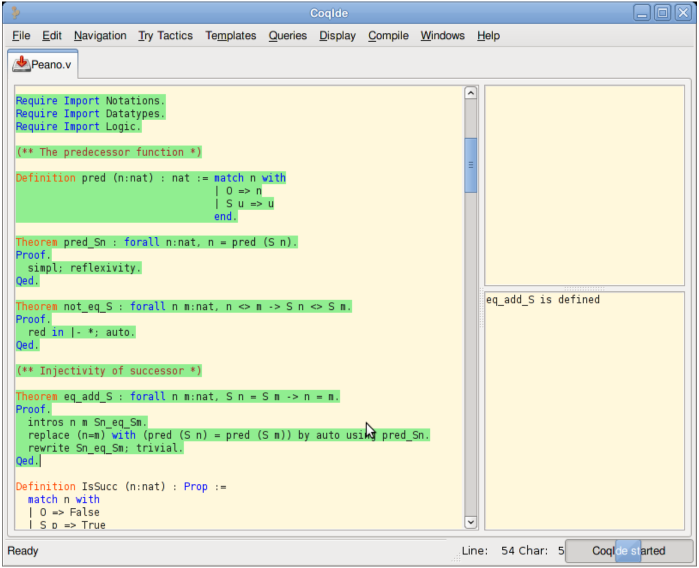
Incrementality in programming & proof languages



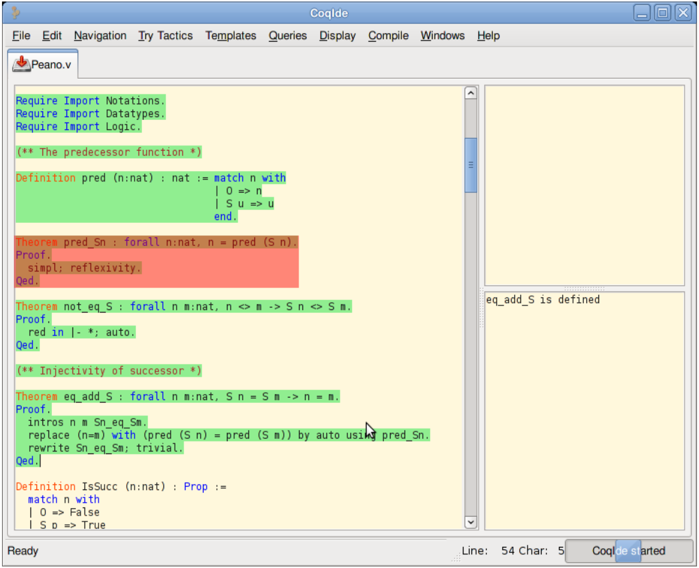
Incrementality in programming & proof languages



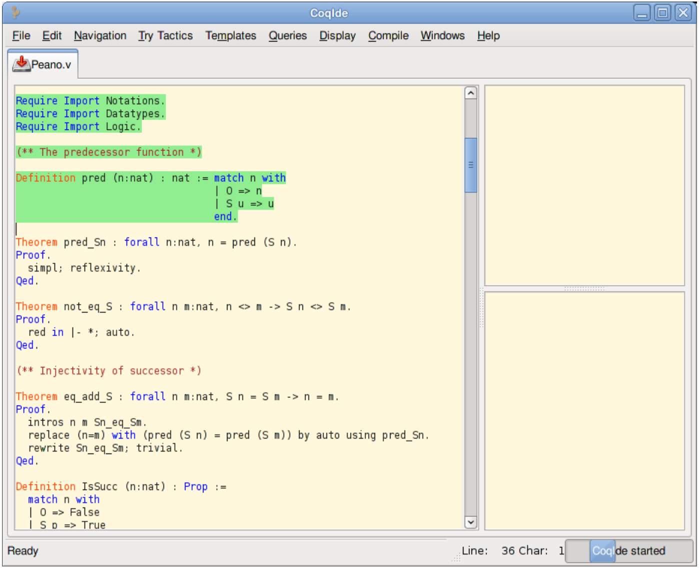
Incrementality in programming & proof languages



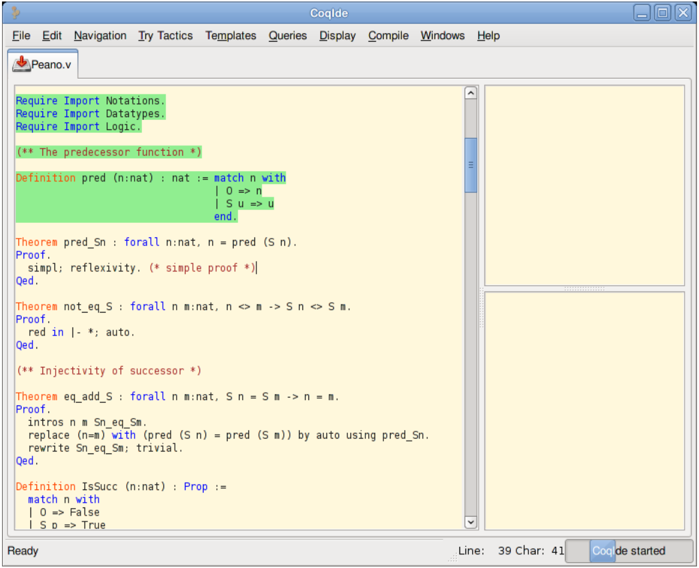
Incrementality in programming & proof languages



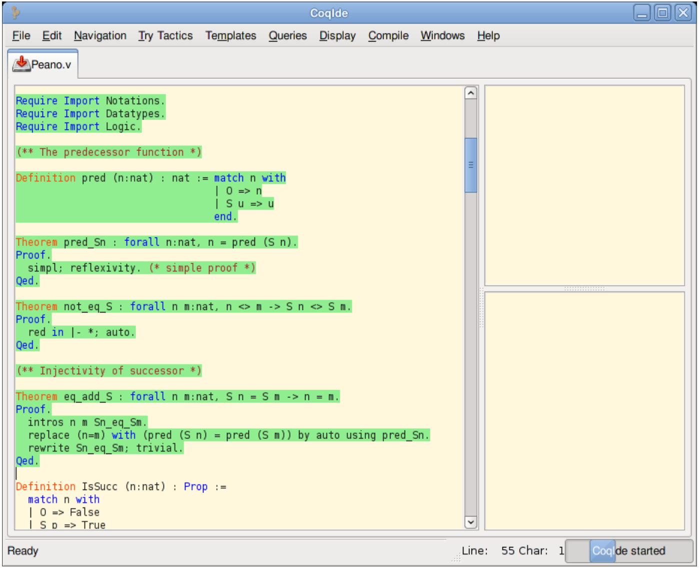
Incrementality in programming & proof languages



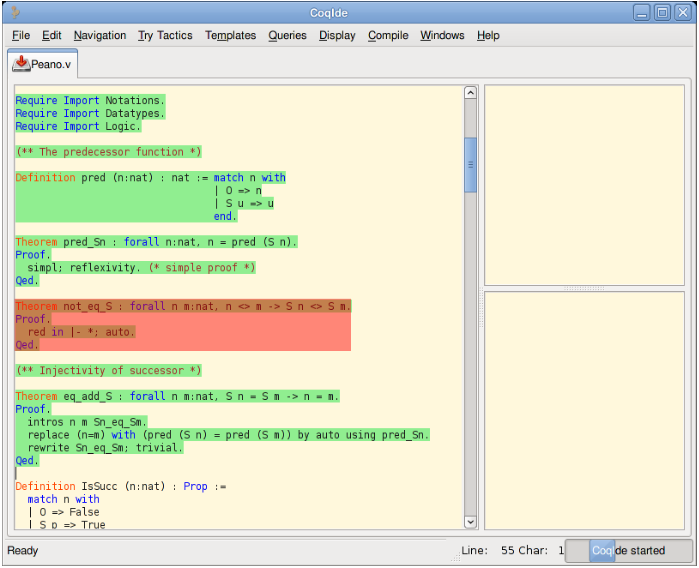
Incrementality in programming & proof languages



Incrementality in programming & proof languages



Incrementality in programming & proof languages



In an ideal world…

- Edition should be possible anywhere
- The impact of changes visible “in real time”
- No need for separate compilation, dependency management

In an ideal world…

- Edition should be possible anywhere
- The impact of changes visible “in real time”
- No need for separate compilation, dependency management

Types are good witnesses of this impact

In an ideal world…

- Edition should be possible anywhere
- The impact of changes visible "in real time"
- No need for separate compilation, dependency management

Types are good witnesses of this impact

Applications

- non-linear user interaction
- tactic languages
- type-directed programming
- typed version control systems

Menu

Menu

A logical framework for incremental type-checking

Yes, we're speaking about (any) typed language.

A type-checker

$$\textbf{val}\ \mathsf{check} : \mathsf{env} \to \mathsf{term} \to \mathsf{types} \to \mathsf{bool}$$

- builds and checks the derivation (on the stack)
- conscientiously discards it

Yes, we're speaking about (any) typed language.

A type-checker

```
val check : env → term → types → bool
```

- builds and checks the derivation (on the stack)
- conscientiously discards it

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \to B, B \to C, A \vdash B \to C}\,Ax \qquad \dfrac{\dfrac{}{A \to B, B \to C, A \vdash A \to B}\,Ax \qquad \dfrac{}{A \to B, B \to C, A \vdash A}\,Ax}{A \to B, B \to C, A \vdash B}\,\to_e}{A \to B, B \to C, A \vdash C}\,\to_e}{A \to B, B \to C \vdash A \to C}\,\to_i}{A \to B \vdash (B \to C) \to A \to C}\,\to_i}{\vdash (A \to B) \to (B \to C) \to A \to C}\,\to_i
$$

A logical framework for incremental type-checking

Yes, we're speaking about (any) typed language.

A type-checker

val check : env $\to$ term $\to$ types $\to$ bool

- builds and checks the derivation (on the stack)
- conscientiously discards it

true

A logical framework for incremental type-checking

Goal Type-check a large derivation taking advantage of the knowledge from type-checking previous versions

Idea Remember all derivations!

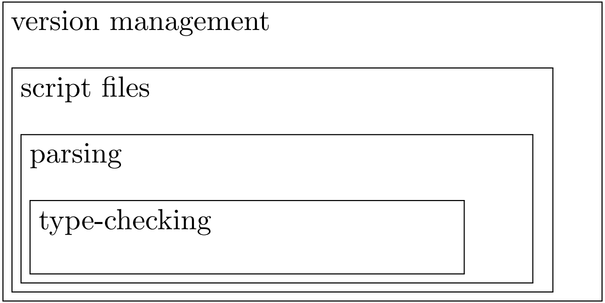
A logical framework for incremental type-checking

Goal Type-check a large derivation taking advantage of the knowledge from type-checking previous versions

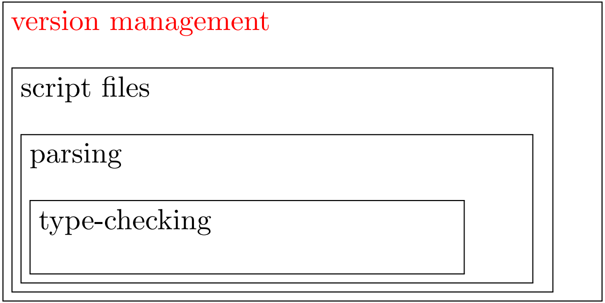
Idea Remember all derivations!

Q Do we really need faster type-checkers?
A Yes, since we implemented these ad-hoc fixes.

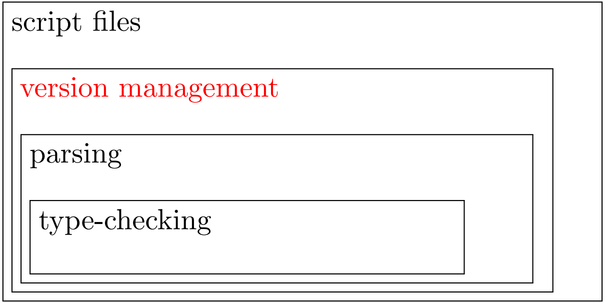
The big picture



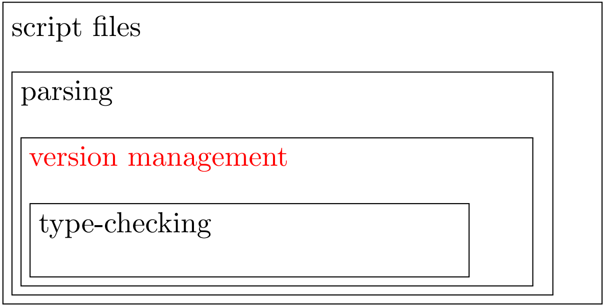
The big picture



The big picture

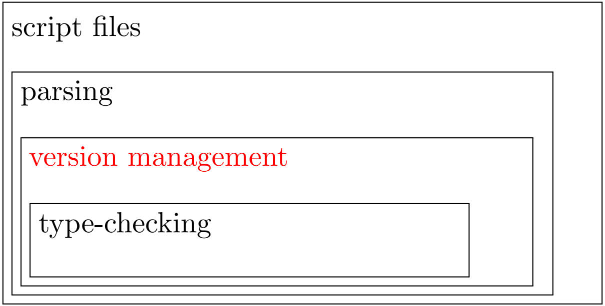


The big picture



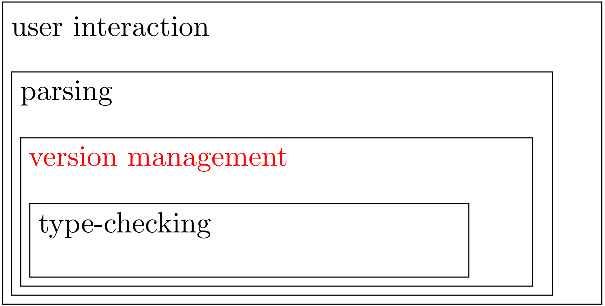
- AST representation

The big picture



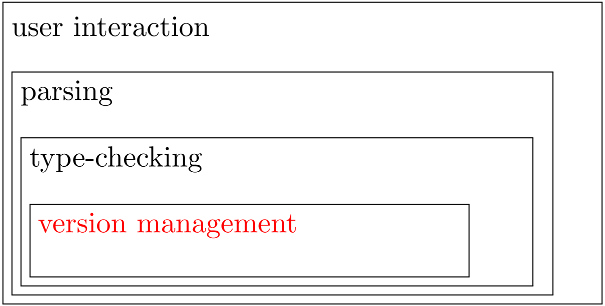
- AST representation

The big picture



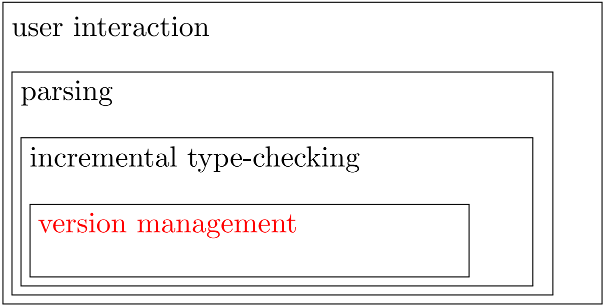
- AST representation

The big picture



- AST representation
- Typing annotations

The big picture



- AST representation
- Typing annotations

Menu

Memoization maybe?

```
let rec check env t a =
  match t with
  | ... → ... false
  | ... → ... true

and infer env t =
  match t with
  | ... → ... None
  | ... → ... Some a
```

Memoization maybe?

```
let table = ref ([] : environ × term × types) in
let rec check env t a =
  if List.mem (env,t,a) !table then true else
    match t with
    | ... → ... false
    | ... → ... table := (env,t,a)::!table; true
and infer env t =
  try List.assoc (env,t) !table with Not_found →
    match t with
    | ... → ... None
    | ... → ... table := (env,t,a)::!table; Some a
```

Memoization maybe?

+ lightweight

Memoization maybe?

+ lightweight
+ efficient implementation

Memoization maybe?

+ lightweight
+ efficient implementation
− imperative

Memoization maybe?

- \+ lightweight
- \+ efficient implementation
- − imperative

  What does it mean logically?

$$\frac{J \in \Gamma}{\Gamma \vdash J \ \mathsf{wf} \Rightarrow \Gamma}$$

$$\frac{\Gamma_1 \vdash J_1 \ \mathsf{wf} \Rightarrow \Gamma_2 \qquad \ldots \qquad \Gamma_{n-1}[J_{n-1}] \vdash J_n \ \mathsf{wf} \Rightarrow \Gamma_n}{\Gamma_1 \vdash J \ \mathsf{wf} \Rightarrow \Gamma_n[J_n][J]}$$

Memoization maybe?

- \+ lightweight
- \+ efficient implementation
- − imperative

  What does it mean logically?

$$\frac{J \in \Gamma}{\Gamma \vdash J \ \mathsf{wf} \Rightarrow \Gamma}$$

$$\frac{\Gamma_1 \vdash J_1 \ \mathsf{wf} \Rightarrow \Gamma_2 \qquad \ldots \qquad \Gamma_{n-1}[J_{n-1}] \vdash J_n \ \mathsf{wf} \Rightarrow \Gamma_n}{\Gamma_1 \vdash J \ \mathsf{wf} \Rightarrow \Gamma_n[J_n][J]}$$

- − external to the logic (meta-cut)

Memoization maybe?

- \+ lightweight
- \+ efficient implementation
- − imperative

  What does it mean logically?

$$\frac{J \in \Gamma}{\Gamma \vdash J \ \mathsf{wf} \Rightarrow \Gamma}$$

$$\frac{\Gamma_1 \vdash J_1 \ \mathsf{wf} \Rightarrow \Gamma_2 \qquad \dots \qquad \Gamma_{n-1}[J_{n-1}] \vdash J_n \ \mathsf{wf} \Rightarrow \Gamma_n}{\Gamma_1 \vdash J \ \mathsf{wf} \Rightarrow \Gamma_n[J_n][J]}$$

- − external to the logic (meta-cut)
- − introduces a dissymmetry

Memoization maybe?

- \+ lightweight
- \+ efficient implementation
- − imperative

  What does it mean logically?

$$\frac{J \in \Gamma}{\Gamma \vdash J \ \mathsf{wf} \Rightarrow \Gamma}$$

$$\frac{\Gamma_1 \vdash J_1 \ \mathsf{wf} \Rightarrow \Gamma_2 \qquad \ldots \qquad \Gamma_{n-1}[J_{n-1}] \vdash J_n \ \mathsf{wf} \Rightarrow \Gamma_n}{\Gamma_1 \vdash J \ \mathsf{wf} \Rightarrow \Gamma_n[J_n][J]}$$

- − external to the logic (meta-cut)
- − introduces a dissymmetry

  What if I want *e.g.* the weakening property to be taken into account?

Memoization maybe?

- \+ lightweight
- \+ efficient implementation
- − imperative

  What does it mean logically?

$$\frac{J \in \Gamma}{\Gamma \vdash J \ \mathsf{wf} \Rightarrow \Gamma}$$

$$\frac{\Gamma_1 \vdash J_1 \ \mathsf{wf} \Rightarrow \Gamma_2 \qquad \dots \qquad \Gamma_{n-1}[J_{n-1}] \vdash J_n \ \mathsf{wf} \Rightarrow \Gamma_n}{\Gamma_1 \vdash J \ \mathsf{wf} \Rightarrow \Gamma_n[J_n][J]}$$

- − external to the logic (meta-cut)
- − introduces a dissymmetry

  What if I want *e.g.* the weakening property to be taken into account?
- − syntactic comparison

Memoization maybe?

- \+ lightweight
- \+ efficient implementation
- – imperative

  What does it mean logically?

$$\frac{J \in \Gamma}{\Gamma \vdash J \ \mathsf{wf} \Rightarrow \Gamma}$$

$$\frac{\Gamma_1 \vdash J_1 \ \mathsf{wf} \Rightarrow \Gamma_2 \qquad \dots \qquad \Gamma_{n-1}[J_{n-1}] \vdash J_n \ \mathsf{wf} \Rightarrow \Gamma_n}{\Gamma_1 \vdash J \ \mathsf{wf} \Rightarrow \Gamma_n[J_n][J]}$$

- – external to the logic (meta-cut)
- – introduces a dissymmetry

  What if I want *e.g.* the weakening property to be taken into account?
- – syntactic comparison
- – still no trace of the derivation

Memoization maybe?

- \+ lightweight
- \+ efficient implementation
- − imperative

  What does it mean logically?

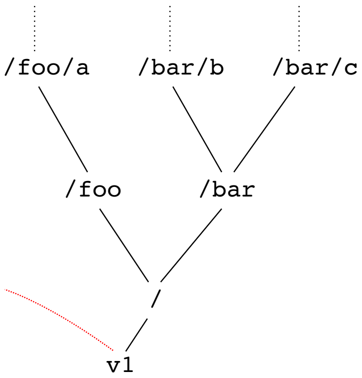
$$\frac{J \in \Gamma}{\Gamma \vdash J\ \mathsf{wf} \Rightarrow \Gamma}$$

$$\frac{\Gamma_1 \vdash J_1\ \mathsf{wf} \Rightarrow \Gamma_2 \qquad \dots \qquad \Gamma_{n-1}[J_{n-1}] \vdash J_n\ \mathsf{wf} \Rightarrow \Gamma_n}{\Gamma_1 \vdash J\ \mathsf{wf} \Rightarrow \Gamma_n[J_n][J]}$$

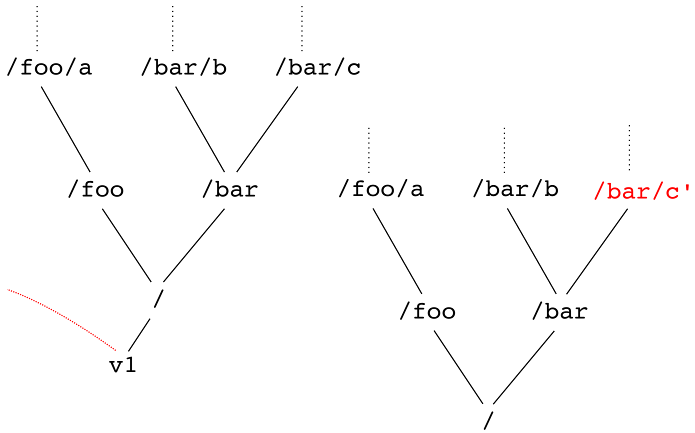
- − external to the logic (meta-cut)
- − introduces a dissymmetry

  What if I want *e.g.* the weakening property to be taken into account?
- − syntactic comparison
- − still no trace of the derivation
- \+ gives good reasons to go on

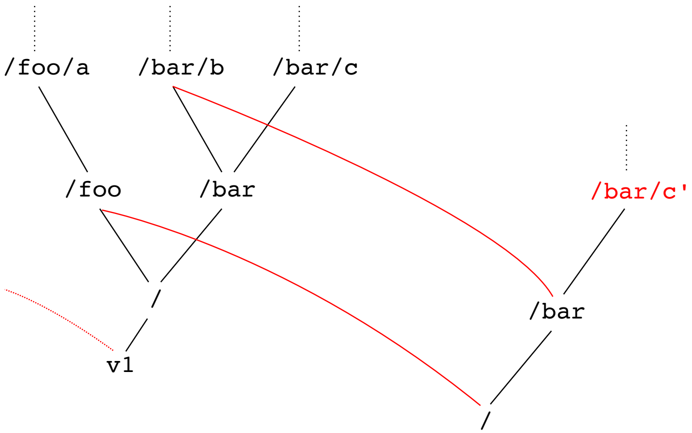
A popular storage model for repositories



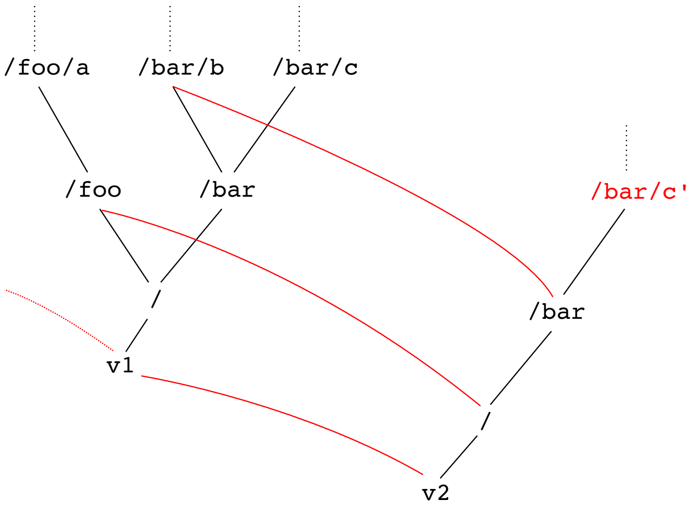
A popular storage model for repositories



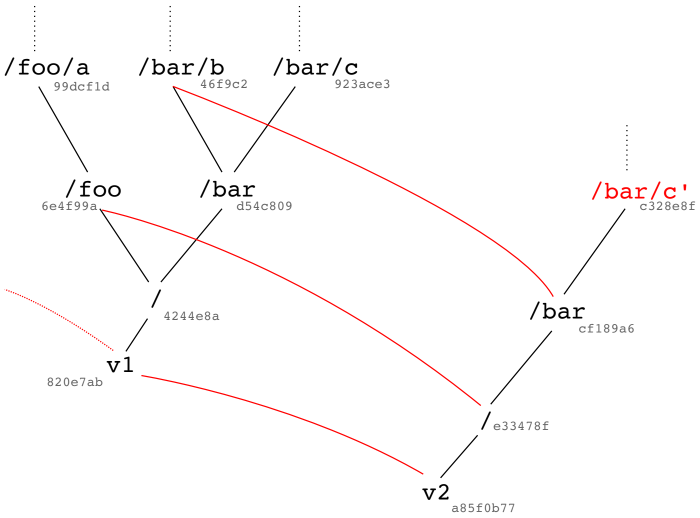
A popular storage model for repositories



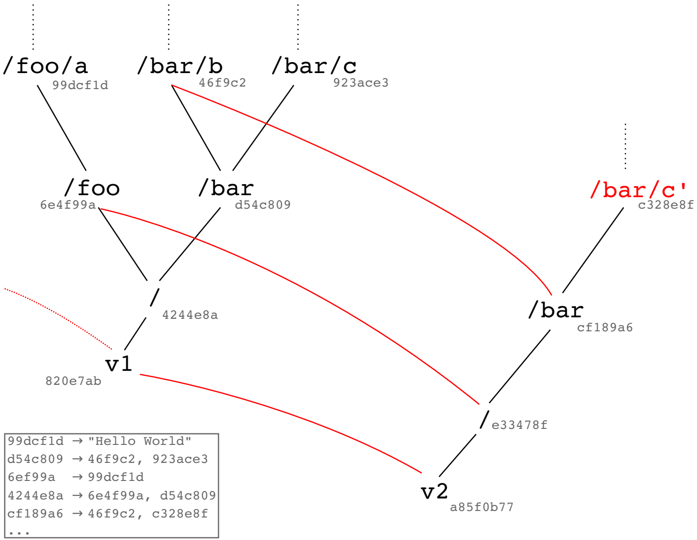
A popular storage model for repositories



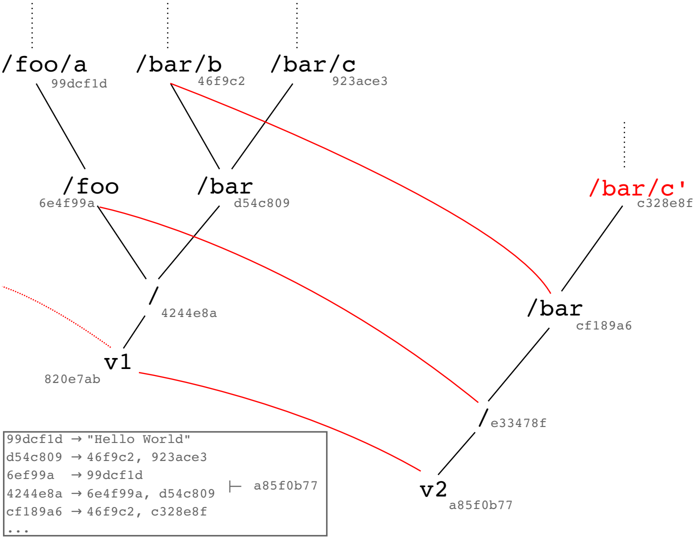
A popular storage model for repositories



A popular storage model for repositories



A popular storage model for repositories



A popular storage model for repositories

The repository R is a pair (Δ, x):

$$\Delta : x \mapsto (\mathsf{Commit}\ (x \times y) \mid \mathsf{Tree}\ \vec{x} \mid \mathsf{Blob}\ string)$$

with the invariants:

- if $(x, \mathsf{Commit}\ (y, z)) \in \Delta$ then
  - $(y, \mathsf{Tree}\ t) \in \Delta$
  - $(z, \mathsf{Commit}\ (t, v)) \in \Delta$
- if $(x, \mathsf{Tree}(\vec{y})) \in \Delta$ then
  for all y_i, either $(y_i, \mathsf{Tree}(\vec{z}))$ or $(y_i, \mathsf{Blob}(s)) \in \Delta$

A popular storage model for repositories

The repository R is a pair (Δ, x):

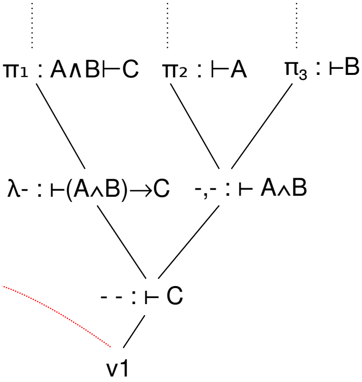
$$\Delta : x \mapsto (\mathsf{Commit}\ (x \times y) \mid \mathsf{Tree}\ \vec{x} \mid \mathsf{Blob}\ \mathit{string})$$

with the invariants:

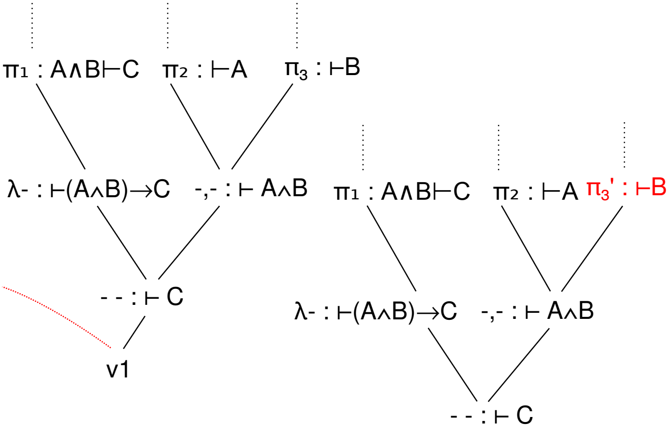
- if $(x, \mathsf{Commit}\ (y, z)) \in \Delta$ then
  - $(y, \mathsf{Tree}\ t) \in \Delta$
  - $(z, \mathsf{Commit}\ (t, v)) \in \Delta$
- if $(x, \mathsf{Tree}(\vec{y})) \in \Delta$ then
  for all y_i, either $(y_i, \mathsf{Tree}(\vec{z}))$ or $(y_i, \mathsf{Blob}(s)) \in \Delta$

Let's do the same with *proofs*

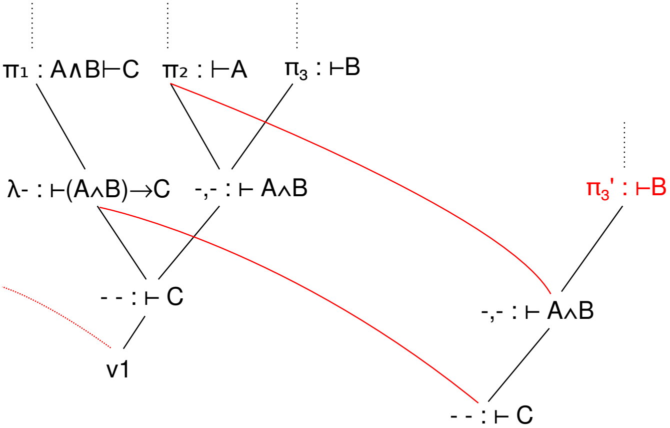
A *typed* repository of proofs



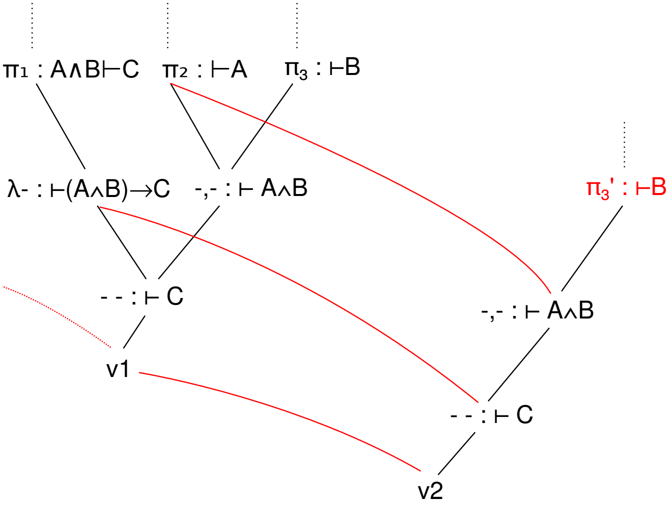
A *typed* repository of proofs



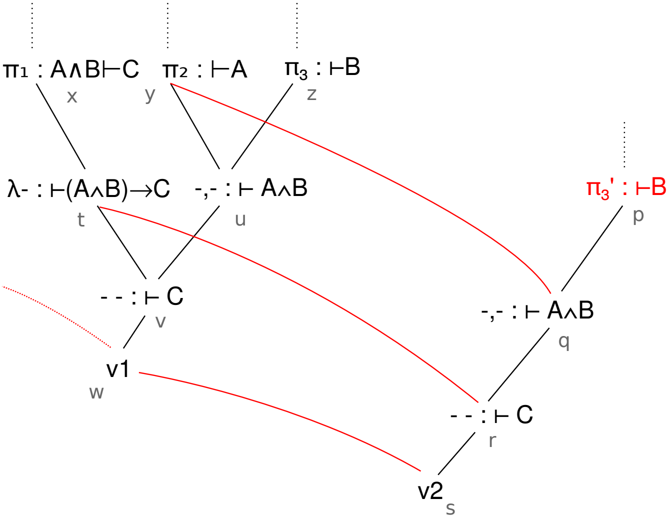
A *typed* repository of proofs



A *typed* repository of proofs



A *typed* repository of proofs



A *typed* repository of proofs

$$
\begin{aligned}
x &= \dots \ : A \wedge B \vdash C \\
y &= \dots \ : \vdash A \\
z &= \dots \ : \vdash B \\
t &= \lambda a^{A \wedge B} \cdot x : \vdash A \wedge B \to C \\
u &= (y, z) : \vdash A \wedge B \\
v &= t\ u : \vdash C \\
w &= \mathsf{Commit}(v, w1) : \mathsf{Version}
\end{aligned}
$$

A *typed* repository of proofs

$$
\begin{aligned}
x &= \ldots\ : A \wedge B \vdash C \\
y &= \ldots\ : \vdash A \\
z &= \ldots\ : \vdash B \\
t &= \lambda a^{A \wedge B} \cdot x : \vdash A \wedge B \to C \\
u &= (y, z) : \vdash A \wedge B \\
v &= t\ u : \vdash C \\
w &= \mathsf{Commit}(v, w1) : \mathsf{Version} \qquad , \quad w
\end{aligned}
$$

A *typed* repository of proofs

$$
\begin{aligned}
x &= \ldots \;: A \wedge B \vdash C \\
y &= \ldots \;: \vdash A \\
z &= \ldots \;: \vdash B \\
t &= \lambda a^{A \wedge B} \cdot x : \vdash A \wedge B \to C \\
u &= (y, z) : \vdash A \wedge B \\
v &= t\ u : \vdash C \\
w &= \mathsf{Commit}(v, w1) : \mathsf{Version} \\
p &= \ldots \;: \vdash B \\
q &= (y, p) : \vdash A \wedge B \\
r &= t\ q : \vdash C \\
s &= \mathsf{Commit}(r, w) : \mathsf{Version}
\end{aligned}
$$

A *typed* repository of proofs

$$
\begin{aligned}
x &= \ldots \ : A \wedge B \vdash C \\
y &= \ldots \ : \vdash A \\
z &= \ldots \ : \vdash B \\
t &= \lambda a^{A \wedge B} \cdot x : \vdash A \wedge B \to C \\
u &= (y, z) : \vdash A \wedge B \\
v &= t\ u : \vdash C \\
w &= \mathsf{Commit}(v, w1) : \mathsf{Version} \\
p &= \ldots \ : \vdash B \\
q &= (y, p) : \vdash A \wedge B \\
r &= t\ q : \vdash C \\
s &= \mathsf{Commit}(r, w) : \mathsf{Version} \qquad , \quad s
\end{aligned}
$$

A *typed* repository of proofs

```
let x = ... : is (cons (conj A B) nil) C in
  let y = ... : is nil A in
    let z = ... : is nil B in
      let t = lam (conj A B) x : is nil (arr (conj A B) C) in
        let u = pair y z : is nil (conj A B) in
          let v = app t u : is nil C in
            let w = commit v w1 : version in
              w
```

A *typed* repository of proofs

```
let x = ... : is (cons (conj A B) nil) C in
  let y = ... : is nil A in
    let z = ... : is nil B in
      let t = lam (conj A B) x : is nil (arr (conj A B) C) in
        let u = pair y z : is nil (conj A B) in
          let v = app t u : is nil C in
            let w = commit v w1 : version in
              let p = ... : is nil B
                let q = pair y p : is nil (conj A B) in
                  let r = t q : is nil C
                    let s = commit r w : version in
                      s
```

A *typed* repository of proofs

```
...
val is : env → prop → type
val conj : prop → prop → prop
val pair : is α β → is α γ → is α (conj β γ)
val version : type
val commit : is nil C → version → version
...
```

```
let x = ... : is (cons (conj A B) nil) C in
  let y = ... : is nil A in
    let z = ... : is nil B in
      let t = lam (conj A B) x : is nil (arr (conj A B) C) in
        let u = pair y z : is nil (conj A B) in
          let v = app t u : is nil C in
            let w = commit v w1 : version in
              let p = ... : is nil B
                let q = pair y p : is nil (conj A B) in
                  let r = t q : is nil C
                    let s = commit r w : version in
                      s
```

A logical framework for incremental type-checking

LF [Harper et al. 1992] provides a way to represent and validate syntax, rules and proofs by means of a typed λ-calculus. But we need a little bit more:

```
 ...
let u = pair y z : is nil (conj A B) in
  let v = app t u : is nil C in
...
```

A logical framework for incremental type-checking

LF [Harper et al. 1992] provides a way to represent and validate syntax, rules and proofs by means of a typed λ-calculus. But we need a little bit more:

```
 ...
let u = pair y z : is nil (conj A B) in
  let v = app t u : is nil C in
...
```

1. definitions / explicit substitutions

A logical framework for incremental type-checking

LF [Harper et al. 1992] provides a way to represent and validate syntax, rules and proofs by means of a typed λ-calculus. But we need a little bit more:

```
 ...
let u = pair y z : is nil (conj A B) in
  let v = app t u : is nil C in
...
```

1. definitions / explicit substitutions
2. type annotations on application spines

A logical framework for incremental type-checking

LF [Harper et al. 1992] provides a way to represent and validate syntax, rules and proofs by means of a typed λ-calculus. But we need a little bit more:

```
 ...
let u = pair y z : is nil (conj A B) in
  let v = app t u : is nil C in
...
```

1. definitions / explicit substitutions
2. type annotations on application spines
3. fully applied constants / η-long NF

A logical framework for incremental type-checking

LF [Harper et al. 1992] provides a way to represent and validate syntax, rules and proofs by means of a typed λ-calculus. But we need a little bit more:

```
 ...
let u = pair y z : is nil (conj A B) in
  let v = app t u : is nil C in
...
```

1. definitions / explicit substitutions
2. type annotations on application spines
3. fully applied constants / η-long NF
4. Naming of all application spines / A-normal form
   (= construction of syntax/proofs)

Positionality

$R =$

```
let x = ... : is (cons (conj A B) nil) C in
  let y = ... : is nil A in
    let z = ... : is nil B in
      let t = lam (conj A B) x : is nil (arr (conj A B) C) in
        let u = pair y z : is nil (conj A B) in
          let v = app t u : is nil C in
            let w = commit v w1 : version in
              w
```

Positionality

$R =$

```
let x = ... : is (cons (conj A B) nil) C in
  let y = ... : is nil A in
    let z = ... : is nil B in
      let t = lam (conj A B) x : is nil (arr (conj A B) C) in
        let u = pair y z : is nil (conj A B) in
          let v = app t u : is nil C in
            let w = commit v w1 : version in
              w
```

- Expose the *head* of the term

Positionality

$R =$

```
let x = ... : is (cons (conj A B) nil) C in
  let y = ... : is nil A in
    let z = ... : is nil B in
      let t = lam (conj A B) x : is nil (arr (conj A B) C) in
        let u = pair y z : is nil (conj A B) in
          let v = app t u : is nil C in
            let w = commit v w1 : version in
              w
```

- Expose the *head* of the term

$$(\lambda x.\lambda y.T)\ U\ V$$

Positionality

$R =$

```
let x = ... : is (cons (conj A B) nil) C in
  let y = ... : is nil A in
    let z = ... : is nil B in
      let t = lam (conj A B) x : is nil (arr (conj A B) C) in
        let u = pair y z : is nil (conj A B) in
          let v = app t u : is nil C in
            let w = commit v w1 : version in
              w
```

- Expose the *head* of the term

$$(\lambda x.\lambda y.T)\ U\ V$$

- Abstract from the *positions* of the binders
  (from inside and from outside)

Menu

Presentation (of the ongoing formalization)

- alternative syntax for LF
- a datastructure of LF derivations
- the repository storage model

Motto: *Take control of the environment*

$$\lambda_{\text{LF}} \xrightarrow[\text{annotation}]{\text{LJ-style}} \text{XLF} \xrightarrow[\text{forget positions}]{\text{heads exposed}} \text{NLF}$$

From LF to XLF

$$
\begin{aligned}
K &::= \Pi x^A \cdot K \mid * \\
A &::= \Pi x^A \cdot A \mid A\ t \mid a \\
t &::= \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid t\ t \mid x \mid c \\
\Gamma &::= \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma &::= \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{aligned}
$$

- start from standard λ_{LF} with definitions

From LF to XLF

$$
\begin{aligned}
K &::= \Pi x^A \cdot K \mid * \\
A &::= \Pi x^A \cdot A \mid A[l] \mid a[l] \\
t &::= \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid t[l] \mid x[l] \mid c[l] \\
l &::= \cdot \mid t; l \\
\Gamma &::= \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma &::= \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{aligned}
$$

- start from standard λ_{LF} with definitions
- sequent calculus-like applications ($\bar{\lambda}$)

From LF to XLF

$$
\begin{aligned}
K &::= \Pi x^A \cdot K \mid * \\
A &::= \Pi x^A \cdot A \mid A[l] : K \mid a[l] : K \\
t &::= \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid t[l] : A \mid x[l] : A \mid c[l] : A \\
l &::= \cdot \mid t; l \\
\Gamma &::= \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma &::= \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{aligned}
$$

- start from standard λ_{LF} with definitions
- sequent calculus-like applications ($\bar{\lambda}$)
- type annotations on application spines

From LF to XLF

$$
\begin{array}{rcl}
K & ::= & \Pi x^A \cdot K \mid * \\
A & ::= & \Pi x^A \cdot A \mid A[l] : K \mid a[l] : K \\
t & ::= & \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid t[l] : A \mid x[l] : A \mid c[l] : A \\
l & ::= & \cdot \mid t; l \\
\Gamma & ::= & \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma & ::= & \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{array}
$$

- start from standard λ_{LF} with definitions
- sequent calculus-like applications ($\bar{\lambda}$)
- type annotations on application spines

$$
\frac{\Gamma \vdash t : A \qquad \Gamma, B\{x/t\} \vdash l : C}{\Gamma, \Pi x^A \cdot B \vdash t; l : C}
$$

From LF to XLF

$$
\begin{array}{rcl}
K & ::= & \Pi x^A \cdot K \mid * \\
A & ::= & \Pi x^A \cdot A \mid A[l] : K \mid a[l] : K \\
t & ::= & \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid t[l] : A \mid x[l] : A \mid c[l] : A \\
l & ::= & \cdot \mid t; l \\
\Gamma & ::= & \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma & ::= & \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{array}
$$

- start from standard λ_{LF} with definitions
- sequent calculus-like applications ($\bar{\lambda}$)
- type annotations on application spines

$$
\frac{\Gamma \vdash t : A \qquad \Gamma\,[x = t]\,, B \vdash l : C}{\Gamma, \Pi x^A \cdot B \vdash t; l : C}
$$

From LF to XLF

$$
\begin{aligned}
K &::= \Pi x^A \cdot K \mid * \\
A &::= \Pi x^A \cdot A \mid A[l] : K \mid a[l] : K \\
t &::= \lambda x^A \cdot t \mid \textsf{let } x = t \textsf{ in } t \mid t[l] : A \mid x[l] : A \mid c[l] : A \\
l &::= \cdot \mid x = t; l \\
\Gamma &::= \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma &::= \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{aligned}
$$

- start from standard λ_{LF} with definitions
- sequent calculus-like applications ($\bar{\lambda}$)
- type annotations on application spines
- named arguments

$$
\frac{\Gamma \vdash t : A \qquad \Gamma\,[x = t]\,, B \vdash l : C}{\Gamma, \Pi x^A \cdot B \vdash x = t; l : C}
$$

From LF to XLF

$$
\begin{array}{rcl}
K & ::= & \Pi x^A \cdot K \mid * \\
A & ::= & \Pi x^A \cdot A \mid A[l] : K \mid a[l] : K \\
t & ::= & \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid t[l] : A \mid x[l] : A \mid c[l] : A \\
l & ::= & \cdot \mid x = t; l \\
\Gamma & ::= & \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma & ::= & \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{array}
$$

- start from standard λ_{LF} with definitions
- sequent calculus-like applications ($\bar{\lambda}$)
- type annotations on application spines
- named arguments

$$
\mathsf{FV}(t[l] : A) = \mathsf{FV}(t) \cup \mathsf{FV}(l) \cup (\mathsf{FV}(A) - \mathsf{FV}(l))
$$
$$
\mathsf{FV}(x = t; l) = \mathsf{FV}(t) \cup (\mathsf{FV}(l) - \{x\})
$$

XLF: Properties

- LJ-style application
- type annotation on application spines
- named arguments (labels)

Lemma (Conservativity)

- $\Gamma \vdash_{\mathrm{LF}} K \ \mathsf{kind} \quad \textit{iff} \quad |\Gamma| \vdash_{\mathrm{XLF}} |K| \ \mathsf{kind}$
- $\Gamma \vdash_{\mathrm{LF}} A \ \mathsf{type} \quad \textit{iff} \quad |\Gamma| \vdash_{\mathrm{XLF}} |A| \ \mathsf{type}$
- $\Gamma \vdash_{\mathrm{LF}} t : A \quad \textit{iff} \quad |\Gamma| \vdash_{\mathrm{XLF}} |t| : |A|$

From XLF to NLF

$$
\begin{aligned}
K &::= \Pi x^A \cdot K \mid * \\
A &::= \Pi x^A \cdot A \mid A[l] : K \mid a[l] : K \\
t &::= \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid t[l] : A \mid x[l] : A \mid c[l] : A \\
l &::= \ \cdot \ \mid x = t; l \\
\Gamma &::= \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma &::= \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{aligned}
$$

- start from XLF

From XLF to NLF

$$
\begin{aligned}
K &::= \Pi x^A \cdot K \mid h_K \\
h_K &::= * \\
A &::= \Pi x^A \cdot A \mid A[l] : K \mid a[l] : K \\
t &::= \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid t[l] : A \mid x[l] : A \mid c[l] : A \\
l &::= \cdot \mid x = t; l \\
\Gamma &::= \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma &::= \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{aligned}
$$

- start from XLF
- isolate heads (non-binders)

From XLF to NLF

$$
\begin{aligned}
K &::= \Pi x^A \cdot K \mid h_K \\
h_K &::= * \\
A &::= \Pi x^A \cdot A \mid h_A \\
h_A &::= A[l] : K \mid a[l] : K \\
t &::= \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid t[l] : A \mid x[l] : A \mid c[l] : A \\
l &::= \cdot \mid x = t; l \\
\Gamma &::= \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma &::= \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{aligned}
$$

- start from XLF
- isolate heads (non-binders)

From XLF to NLF

$$
\begin{aligned}
K &::= \Pi x^A \cdot K \mid h_K \\
h_K &::= * \\
A &::= \Pi x^A \cdot A \mid h_A \\
h_A &::= A[l] : K \mid a[l] : K \\
t &::= \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid h_t \\
h_t &::= t[l] : A \mid x[l] : A \mid c[l] : A \\
l &::= \cdot \mid x = t; l \\
\Gamma &::= \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma &::= \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{aligned}
$$

- start from XLF
- isolate heads (non-binders)

From XLF to NLF

$$
\begin{array}{rcl}
K & ::= & \Pi x^{A} \cdot K \mid h_K \\
h_K & ::= & * \\
A & ::= & \Pi x^{A} \cdot A \mid h_A \\
h_A & ::= & A[l] : h_K \mid a[l] : h_K \\
t & ::= & \lambda x^{A} \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid h_t \\
h_t & ::= & t[l] : A \mid x[l] : A \mid c[l] : A \\
l & ::= & \cdot \mid x = t; l \\
\Gamma & ::= & \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma & ::= & \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{array}
$$

- start from XLF
- isolate heads (non-binders)
- enforce η-long forms by annotating with heads

From XLF to NLF

$$
\begin{array}{rcl}
K & ::= & \Pi x^A \cdot K \mid h_K \\
h_K & ::= & * \\
A & ::= & \Pi x^A \cdot A \mid h_A \\
h_A & ::= & A[l] : h_K \mid a[l] : h_K \\
t & ::= & \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid h_t \\
h_t & ::= & t[l] : h_A \mid x[l] : h_A \mid c[l] : h_A \\
l & ::= & \cdot \mid x = t; l \\
\Gamma & ::= & \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma & ::= & \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{array}
$$

- start from XLF
- isolate heads (non-binders)
- enforce η-long forms by annotating with heads

From XLF to NLF

$$
\begin{array}{rcl}
K & ::= & \Pi x^A \cdot K \mid h_K \\
h_K & ::= & * \\
A & ::= & \Pi x^A \cdot A \mid h_A \\
h_A & ::= & A[l] : h_K \mid a[l] : h_K \\
t & ::= & \lambda x^A \cdot t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \mid h_t \\
h_t & ::= & t[l] : h_A \mid x[l] : h_A \mid c[l] : h_A \\
l & ::= & \cdot \mid x = t; l \\
\Gamma & ::= & \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma & ::= & \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{array}
$$

- start from XLF
- isolate heads (non-binders)
- enforce η-long forms by annotating with heads

$$t[l] : \Pi x^A \cdot B \quad \longrightarrow_\eta \quad \lambda x^A \cdot (t[x = x; l] : B) \qquad x \notin \mathsf{FV}(t)$$

From XLF to NLF

$$
\begin{array}{rcl}
K & ::= & \Pi x^A \cdot K \mid h_K \\
h_K & ::= & * \\
A & ::= & \Pi x^A \cdot A \mid h_A \\
h_A & ::= & A[l] : h_K \mid a[l] : h_K \\
t & ::= & \Gamma \vdash h_t \\
h_t & ::= & t[l] : h_A \mid x[l] : h_A \mid c[l] : h_A \\
l & ::= & \cdot \mid x = t; l \\
\Gamma & ::= & \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma & ::= & \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{array}
$$

- start from XLF
- isolate heads (non-binders)
- enforce η-long forms by annotating with heads
- factorize binders and environments

From XLF to NLF

$$
\begin{array}{rcl}
K & ::= & \Pi x^A \cdot K \mid h_K \\
h_K & ::= & * \\
A & ::= & \Gamma \vdash h_A \\
h_A & ::= & A[l] : h_K \mid a[l] : h_K \\
t & ::= & \Gamma \vdash h_t \\
h_t & ::= & t[l] : h_A \mid x[l] : h_A \mid c[l] : h_A \\
l & ::= & \cdot \mid x = t; l \\
\Gamma & ::= & \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma & ::= & \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{array}
$$

- start from XLF
- isolate heads (non-binders)
- enforce η-long forms by annotating with heads
- factorize binders and environments

From XLF to NLF

$$
\begin{array}{rcl}
K & ::= & \Gamma \vdash h_k \\
h_K & ::= & * \\
A & ::= & \Gamma \vdash h_A \\
h_A & ::= & A[l] : h_K \mid a[l] : h_K \\
t & ::= & \Gamma \vdash h_t \\
h_t & ::= & t[l] : h_A \mid x[l] : h_A \mid c[l] : h_A \\
l & ::= & \cdot \mid x = t; l \\
\Gamma & ::= & \cdot \mid \Gamma\,[x : A] \mid \Gamma\,[x = t] \\
\Sigma & ::= & \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{array}
$$

- start from XLF
- isolate heads (non-binders)
- enforce η-long forms by annotating with heads
- factorize binders and environments

From XLF to NLF

$$
\begin{array}{rcl}
K & ::= & \Gamma \vdash h_k \\
h_K & ::= & * \\
A & ::= & \Gamma \vdash h_A \\
h_A & ::= & A[l] : h_K \mid a[l] : h_K \\
t & ::= & \Gamma \vdash h_t \\
h_t & ::= & t[l] : h_A \mid x[l] : h_A \mid c[l] : h_A \\
l & ::= & \Gamma \\
\Gamma & ::= & \cdot \mid \Gamma\,[x : A] \ \mid \Gamma\,[x = t] \\
\Sigma & ::= & \cdot \mid \Sigma\,[c : A] \ \mid \Sigma\,[a : K]
\end{array}
$$

- start from XLF
- isolate heads (non-binders)
- enforce η-long forms by annotating with heads
- factorize binders and environments

From XLF to NLF

$$
\begin{aligned}
K &::= \Gamma \vdash h_k \\
h_K &::= * \\
A &::= \Gamma \vdash h_A \\
h_A &::= A[l] : h_K \mid a[l] : h_K \\
t &::= \Gamma \vdash h_t \\
h_t &::= t[l] : h_A \mid x[l] : h_A \mid c[l] : h_A \\
l &::= \Gamma \\
\Gamma &: x \mapsto ([x : A] \mid [x = t]) \\
\Sigma &::= \cdot \mid \Sigma\,[c : A] \mid \Sigma\,[a : K]
\end{aligned}
$$

- start from XLF
- isolate heads (non-binders)
- enforce η-long forms by annotating with heads
- factorize binders and environments
- abstract over environment datastructure (maps)

NLF

Syntax

$$
\begin{array}{rcl}
K & ::= & \Gamma\ \mathsf{kind} \\
A & ::= & \Gamma \vdash h_A\ \mathsf{type} \\
h_A & ::= & a\ \Gamma \\
t & ::= & \Gamma \vdash h_t : h_A \\
h_t & ::= & t\ \Gamma \mid x\ \Gamma \mid c\ \Gamma \\
\Gamma & : & x \mapsto ([x : a] \mid [x = t])
\end{array}
$$

Judgements

- $\Gamma\ \mathsf{kind}$
- $\Gamma \vdash h_A\ \mathsf{type}$
- $\Gamma \vdash h_t : h_A$

NLF

Syntax

$$
\begin{array}{rcl}
K & ::= & \Gamma\ \textsf{kind} \\
A & ::= & \Gamma \vdash h_A\ \textsf{type} \\
h_A & ::= & a\ \Gamma \\
t & ::= & \Gamma \vdash h_t : h_A \\
h_t & ::= & t\ \Gamma \mid x\ \Gamma \mid c\ \Gamma \\
\Gamma & : & x \mapsto ([x : a] \mid [x = t])
\end{array}
$$

Judgements

- K
- A
- t

NLF

Syntax

$$
\begin{array}{rcl}
K & ::= & \Gamma\ \textsf{kind} \\
A & ::= & \Gamma \vdash h_A\ \textsf{type} \\
h_A & ::= & a\ \Gamma \\
t & ::= & \Gamma \vdash h_t : h_A \\
h_t & ::= & t\ \Gamma \mid x\ \Gamma \mid c\ \Gamma \\
\Gamma & : & x \mapsto ([x : a] \mid [x = t])
\end{array}
$$

Judgements

- K wf
- A wf
- t wf

NLF

Notations

- "h_A" for "$\vdash h_A$"
- "h_A" for "$\emptyset \vdash h_A \; \mathsf{type}$"
- "a" for "$a \; \emptyset$"

Example

$$\lambda f^{A \to B} \cdot \lambda x^A \cdot f \; x : (A \to B) \to A \to B \qquad \equiv$$

$$[f : [a : A] \vdash B \; \mathsf{type}] \, [x : A] \vdash f \; [a = x] : B$$

Some more examples

$$A : \emptyset \ \text{kind}$$
$$\equiv *$$

$$vec : [len : \mathbb{N}] \ \text{kind}$$
$$\equiv \mathbb{N} \to *$$

$$nil : \ \vdash vec \ [len = \ \vdash 0 : \mathbb{N}] \ \text{type}$$
$$\equiv vec \ 0 : *$$

$$cons : [l : \mathbb{N}] \, [hd : A] \, [tl : \ \vdash vec \ [len = \ \vdash l : \mathbb{N}] \ \text{type}] \vdash vec \ [len = \ \vdash s \ [n = \ \vdash l : \mathbb{N}] : \mathbb{N}] \ \text{type}$$
$$\equiv \Pi l^{\mathbb{N}} \cdot A \to \Pi tl^{vec \ l} \cdot vec \ (s \ l : \mathbb{N}) : *$$

$$fill : [n : \mathbb{N}] \vdash vec \ [len = \ \vdash n : \mathbb{N}] \ \text{type}$$
$$\equiv \Pi n^{\mathbb{N}} \cdot (vec \ n : *)$$

$$empty : [e : vec \ [len = 0]] \ \text{kind}$$
$$\equiv vec \ 0 \to *$$

$$- : \ \vdash empty \ [e = \ \vdash fill \ [n = 0] : vec \ [len = n]] \ \text{type}$$
$$\equiv empty \ (fill \ 0 : vec \ 0)$$

Environments

... double as labeled *directed acyclic graphs* of dependencies:

Definition (environment)

$\Gamma = (V, E)$ directed acyclic where:

- $V \subseteq \mathcal{X} \times (t \uplus A)$ and
- $(x, y) \in E$ (x depends on y) if $y \in \mathsf{FV}(\Gamma(x))$

Definition (lookup)

$$\Gamma(x) : A \quad \text{if} \quad (x, A) \in E$$
$$\Gamma(x) = t \quad \text{if} \quad (x, t) \in E$$

Definition (bind)

$$\Gamma\,[x : A] = (V \cup (x, A), E \cup \{(x, y) \mid y \in \mathsf{FV}(A)\})$$
$$\Gamma\,[x = t] = (V \cup (x, A), E \cup \{(x, y) \mid y \in \mathsf{FV}(t)\})$$

Environments

… double as labeled *directed acyclic graphs* of dependencies:

Definition (decls, defs)

$\mathsf{decls}(\Gamma) = [x_1, \ldots, x_n]$ s.t. $\Gamma(x_i) : A_i$ topologically sorted *wrt.* Γ
$\mathsf{defs}(\Gamma) = [x_1, \ldots, x_n]$ s.t. $\Gamma(x_i) = t_i$ topologically sorted *wrt.* Γ

Definition (merge)

$\Gamma \cdot \Delta = \Gamma \cup \Delta$ s.t.

- if $\Gamma(x) : A$ and $\Gamma(x) = t$ then $\Gamma \cdot \Delta(x) = t$
- undefined otherwise

Reduction

Definition (A-contexts)

$\Delta^* = \{\, [x = x] \mid x \in \mathsf{defs}(\Delta)\}$

$$
\begin{array}{rcll}
\Gamma \vdash (\Delta \vdash h_t : h_A)\ \Xi : \_ & \xrightarrow{\text{``}\beta\text{''}} & \Gamma \cdot \Delta \cdot \Xi \vdash h_t : h_A & \\
\Gamma \vdash c\ \Delta : h_A & \longrightarrow & \Gamma \cdot \Delta \vdash c\ \Delta^* : h_A & \text{if } \Delta \neq \Delta^* \\
\Gamma \vdash c\ \Xi^* : a\ \Delta & \longrightarrow & \Gamma \cdot \Delta \vdash c\ \Xi^* : a\ \Delta^* & \text{if } \Delta \neq \Delta^*
\end{array}
$$

Typing

$$\textsc{Fam}\quad \frac{\Sigma(a) : (\Xi\ \textsf{kind}) \qquad \Gamma \vdash \Delta : \Xi}{\Gamma \vdash a\ \Delta\ \textsf{type}}$$

$$\textsc{Objc}\quad \frac{\Sigma(c) : (\Xi \vdash h_A\ \textsf{type}) \qquad \Gamma \vdash \Delta : \Xi \qquad \Gamma \cdot \Xi \cdot \Delta \vdash h'_A \equiv h_A\ \textsf{type}}{\Gamma \vdash c\ \Delta : h'_A}$$

$$\textsc{Objx}\quad \frac{\Gamma(x) = (\Xi \vdash h_t : h_A) \qquad \Gamma \vdash \Delta : \Xi \qquad \Gamma \cdot \Xi \cdot \Delta \vdash h'_A \equiv h_A\ \textsf{type} \qquad \Gamma \cdot \Xi \cdot \Delta \vdash h_t : h_A}{\Gamma \vdash x\ \Delta : h'_A}$$

$$\textsc{Args}\quad \frac{\forall x \in \textsf{decls}(\Xi) \quad \Delta(x) = (\Delta' \vdash h_t : h_A) \qquad \Xi(x) : (\Xi' \vdash h'_A\ \textsf{type}) \quad \Gamma \cdot \Delta \cdot \Delta' \vdash h_t : h_A \qquad \Gamma \cdot \Xi \cdot \Delta \cdot \Xi' \cdot \Delta' \vdash h'_A \equiv h_A\ \textsf{type}}{\Gamma \vdash \Delta : \Xi}$$

Properties

Translation functions

- $|\cdot|_\Gamma \;:\; K_{\mathrm{LF}} \to \Gamma_{\mathrm{NLF}} \to K_{\mathrm{NLF}}$ option
- $|\cdot|_\Gamma \;:\; A_{\mathrm{LF}} \to \Gamma_{\mathrm{NLF}} \to A_{\mathrm{NLF}}$ option
- $|\cdot|_\Gamma \;:\; t_{\mathrm{LF}} \to \Gamma_{\mathrm{NLF}} \to t_{\mathrm{NLF}}$ option
- $\ldots$ and their inverses $|\cdot|^{-1}$

Conjecture (Conservativity)

- $\vdash_{\mathrm{LF}} K$ kind $\quad$ iff $\quad (|K|_\emptyset)$ wf
- $\vdash_{\mathrm{LF}} A$ type $\quad$ iff $\quad (|A|_\emptyset)$ wf
- $\vdash_{\mathrm{LF}} t : A$ $\quad$ iff $\quad (|t|_\emptyset)$ wf

Menu

Status

```
$ ./gasp init hol.elf
```

Status

```
$ ./gasp init hol.elf

[holtype : kind]
[i : holtype]
[o : holtype]
[arr : [x2 : holtype][x1 : holtype] ⊢ holtype type]

Fatal error: exception Assert_failure("src/NLF.ml", 61, 13)
```

Checkout

```
$ ./gasp checkout v42
```

if

$$t = \Gamma \vdash v_{52} : \mathsf{Version} \quad \text{and} \quad \Gamma(v_{42}) = \mathsf{Commit}(v_{41}, h)$$

then

$$|\Gamma(h)|^{-1}$$

is the LF term representing `v42`

Commit

```
$ ./gasp commit term.elf
```

if

$$t = \Gamma \vdash v_{52} : \mathsf{Version} \quad \text{and} \quad |\texttt{term.elf}|_\Gamma = \Delta \vdash h_t : h_A$$

then

$$\Delta\,[v_{53} = \mathsf{Commit}\ [prev = v_{52}]\,[this = h_t]] \vdash v_{53} : \mathsf{Version}$$

is the new repository

Further work

- still some technical & metatheoretical unknowns
- from derivations to terms (proof search? views?)
- diff on terms or derivations
- type errors handling and recovery
- mimick other operations from VCS (Merge)