

# Towards typed repositories of proofs

MIPS 2010

Matthias Puech<sup>1,2</sup>    Yann Régis-Gianas<sup>2</sup>  
puech@cs.unibo.it    yrg@pps.jussieu.fr

<sup>1</sup>Dept. of Computer Science, University of Bologna

<sup>2</sup>University Paris 7, CNRS, and INRIA, PPS, team  $\pi r^2$

July 10, 2010

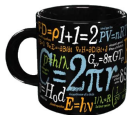
## How are *constructed* formal mathematics?

Q : What is the common point between the working mathematician and the working programmer?

## How are *constructed* formal mathematics?

Q : What is the common point between the working mathematician and the working programmer?

A : They both spend more time *editing* than *writing*



# 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 formal mathematics is still largely inspired by legacy software development:

- ▶ File-based scripts (`emacs`)
- ▶ Separate compilation (`make`)
- ▶ Text-based versioning (`svn`, `diffs`...)

# A paradoxical situation

## Observation

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

... And yet,

Workflow of formal mathematics is still largely inspired by legacy software development:

- ▶ File-based scripts (`emacs`)
- ▶ Separate compilation (`make`)
- ▶ Text-based versioning (`svn`, `diffs`...)

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

# Motivations

## Rigidity of linear edition

- ▶  $((\text{edit}; \text{compile})^*; \text{commit})^*$  loop does not scale to proofs
- ▶ Concept freeze inhibits the discovery process
- ▶ No room for alternate definitions

# Motivations

## Rigidity of linear edition

- ▶  $((\text{edit}; \text{compile})^*; \text{commit})^*$  loop does not scale to proofs
- ▶ Concept freeze inhibits the discovery process
- ▶ No room for alternate definitions

## Laxity of textual representation

- ▶ Textual scripts diffs do not reflect the semantics
- ▶ Not even the syntax



# Motivations

## Rigidity of linear edition

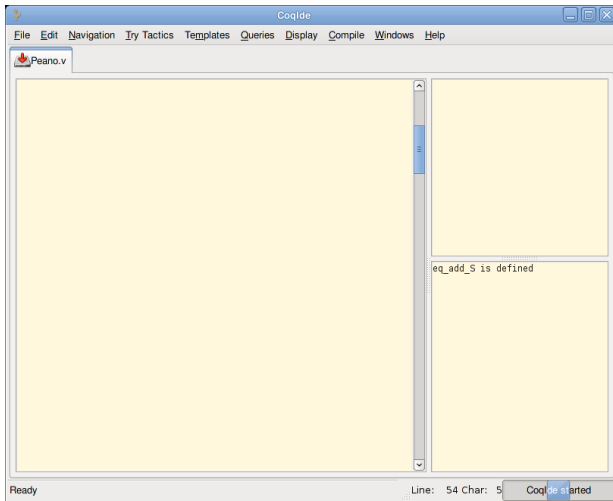
- ▶  $((\text{edit}; \text{compile})^*; \text{commit})^*$  loop does not scale to proofs
- ▶ Concept freeze inhibits the discovery process
- ▶ No room for alternate definitions

## Laxity of textual representation

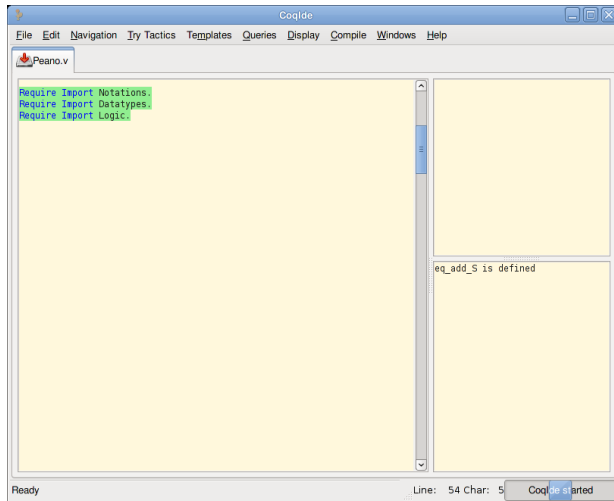
- ▶ Textual scripts diffs do not reflect the semantics
- ▶ Not even the syntax

... Maybe it wasn't adapted to software development

# The impact of changes

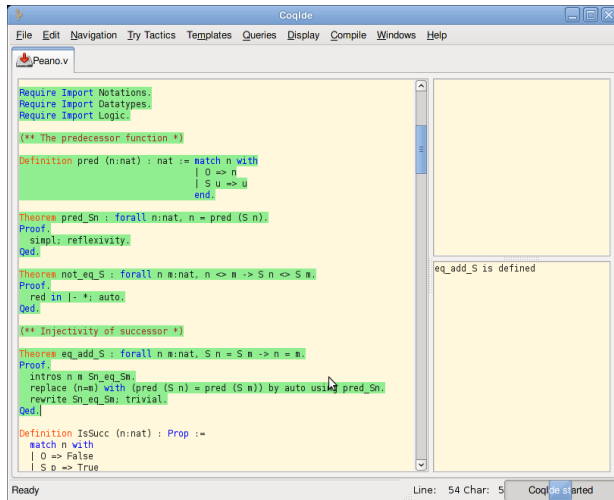


# The impact of changes



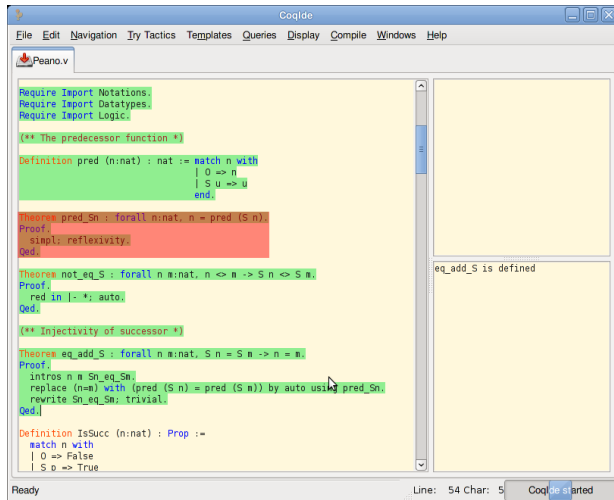
- File-based separate compilation

# The impact of changes



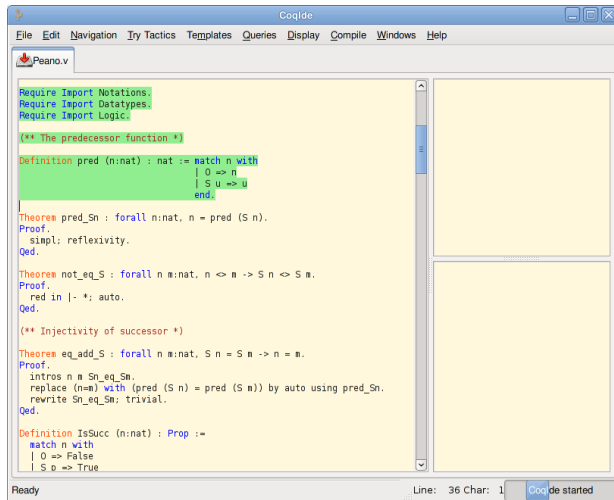
- File-based separate compilation
- Interaction loop with global undo

# The impact of changes



- File-based separate compilation
- Interaction loop with global undo

# The impact of changes



The screenshot shows the CoqIDE application window. The title bar says "CoqIDE". The menu bar includes "File", "Edit", "Navigation", "Try Tactics", "Templates", "Queries", "Display", "Compile", "Windows", and "Help". The main editor area displays the file "Peano.v" with the following Coq code:

```
Require Import Notations.
Require Import Datatypes.
Require Import Logic.

(** The predecessor function *)
Definition pred (n:nat) : nat := match n with
| 0 => n
| S u => u
end.

Theorem pred_Sn : forall n:nat, n = pred (S n).
Proof.
  simpl; reflexivity.
Qed.

Theorem not_eq_S : forall n m:nat, n <> m -> S n <> S m.
Proof.
  red in |- *. auto.
Qed.

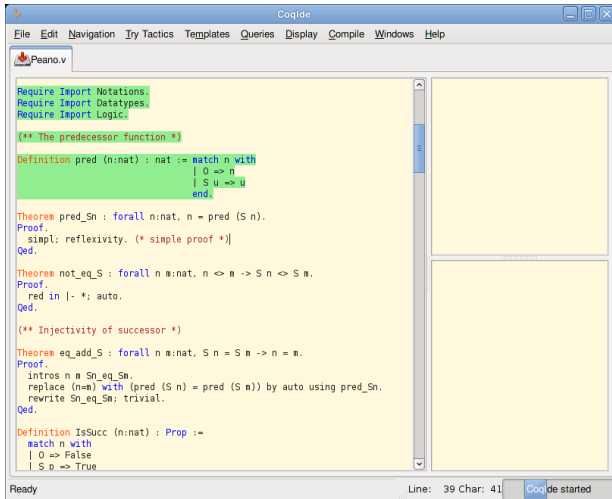
(** Injectivity of successor *)
Theorem eq_add_S : forall n m:nat, S n = S m -> n = m.
Proof.
  intros n m Sn_eq_Sm.
  replace (n=m) with (pred (S n) = pred (S m)) by auto using pred_Sn.
  rewrite Sn_eq_Sm; trivial.
Qed.

Definition IsSucc (n:nat) : Prop :=
  match n with
  | 0 => False
  | S o => True
```

The status bar at the bottom indicates "Ready" and "Line: 36 Char: 1". A small button labeled "CoqIDE started" is visible on the right.

- File-based separate compilation
- Interaction loop with global undo

# The impact of changes



The screenshot shows the CoqIDE application window. The title bar says "CoqIde". The menu bar includes "File", "Edit", "Navigation", "Try Tactics", "Templates", "Queries", "Display", "Compile", "Windows", and "Help". The main editor area displays the file "Peano.v" with the following code:

```
Require Import Notations.
Require Import Datatypes.
Require Import Logic.

(** The predecessor function *)
Definition pred (n:nat) : nat := match n with
| 0 => n
| S u => u
end.

Theorem pred_Sn : forall n:nat, n = pred (S n).
Proof.
  simpl; reflexivity. (* simple proof *)
Qed.

Theorem not_eq_S : forall n m:nat, n <> m -> S n <> S m.
Proof.
  red in |- *. auto.
Qed.

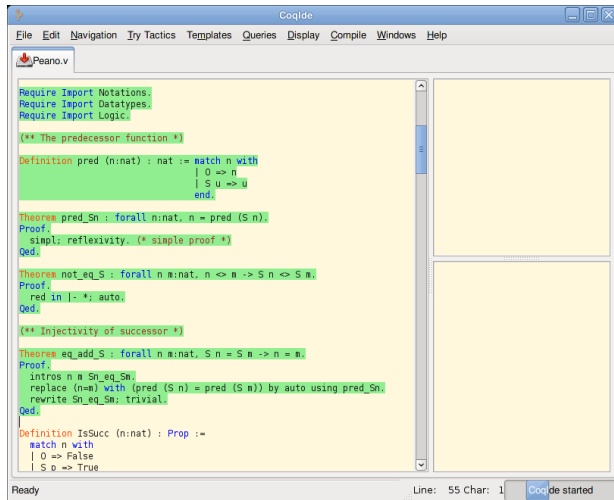
(** Injectivity of successor *)
Theorem eq_add_S : forall n m:nat, S n = S m -> n = m.
Proof.
  intros n m Sn_eq_Sm.
  replace (n=m) with (pred (S n) = pred (S m)) by auto using pred_Sn.
  rewrite Sn_eq_Sm; trivial.
Qed.

Definition IsSucc (n:nat) : Prop :=
  match n with
  | 0 => False
  | S o => True
```

The status bar at the bottom indicates "Ready" and "Line: 39 Char: 41". A small button labeled "CoqIde started" is visible on the right.

- ▶ File-based separate compilation
- ▶ Interaction loop with global undo

# The impact of changes



```
Require Import Notations.
Require Import Datatypes.
Require Import Logic.

(** The predecessor function *)
Definition pred (n:nat) : nat := match n with
| 0 => n
| S u => u
end.

Theorem pred_Sn : forall n:nat, n = pred (S n).
Proof.
  simpl; reflexivity. (* simple proof *)
Qed.

Theorem not_eq_S : forall n m:nat, n <> m -> S n <> S m.
Proof.
  red in |- *. auto.
Qed.

(** Injectivity of successor *)
Theorem eq_add_S : forall n m:nat, S n = S m -> n = m.
Proof.
  intros n m Sn_eq_Sm.
  replace (n=m) with (pred (S n) = pred (S m)) by auto using pred_Sn.
  rewrite Sn_eq_Sm; trivial.
Qed.

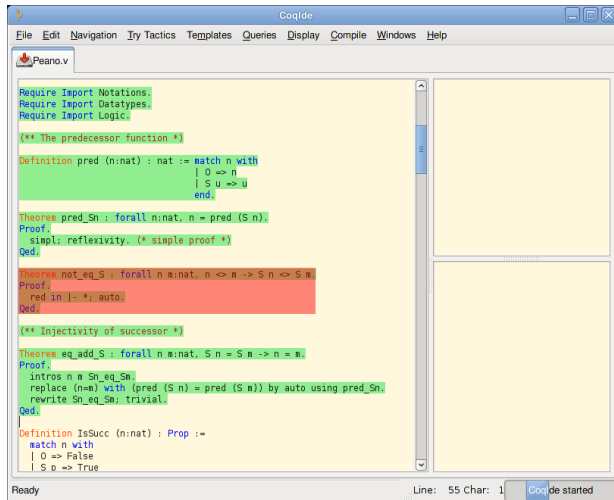
Definition IsSucc (n:nat) : Prop :=
  match n with
  | 0 => False
  | S o => True
```

Ready Line: 55 Char: 1 CoqIDE started

- ▶ File-based separate compilation
- ▶ Interaction loop with global undo



# The impact of changes



```
Require Import Notations.
Require Import Datatypes.
Require Import Logic.

(** The predecessor function *)
Definition pred (n:nat) : nat := match n with
| 0 => n
| S u => u
end.

Theorem pred_Sn : forall n:nat, n = pred (S n).
Proof.
  simpl; reflexivity. (* simple proof *)
Qed.

Theorem not_eq_S : forall n m:nat, n <> m -> S n <> S m.
Proof.
  red in |- *. auto.
Qed.

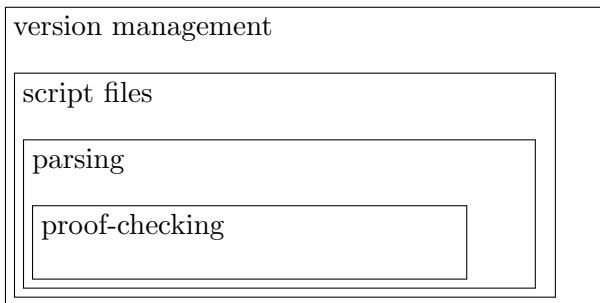
(** Injectivity of successor *)
Theorem eq_add_S : forall n m:nat, S n = S m -> n = m.
Proof.
  intros n m Sn_eq_Sm.
  replace (n=m) with (pred (S n) = pred (S m)) by auto using pred_Sn.
  rewrite Sn_eq_Sm; trivial.
Qed.

Definition IsSucc (n:nat) : Prop :=
  match n with
  | 0 => False
  | S o => True
```

Ready Line: 55 Char: 1 CoqIDE started

- File-based separate compilation
- Interaction loop with global undo

# Methodology



# Methodology

version management

script files

parsing

proof-checking

# Methodology

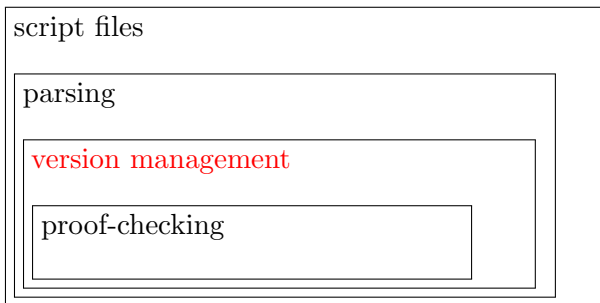
script files

version management

parsing

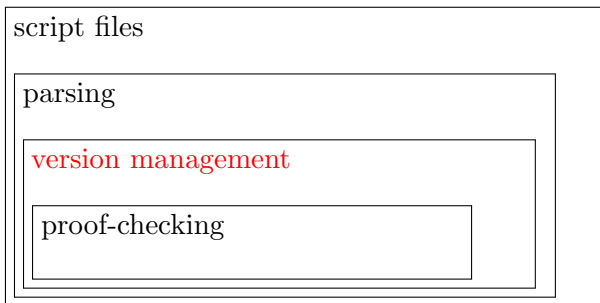
proof-checking

# Methodology



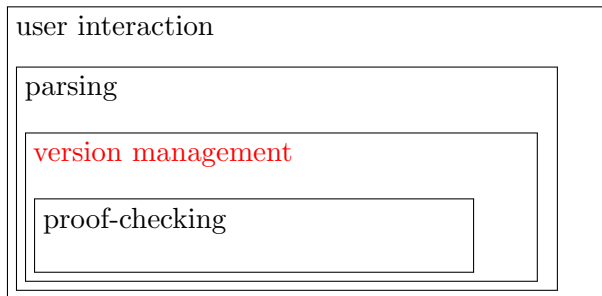
- ▶ AST representation

# Methodology



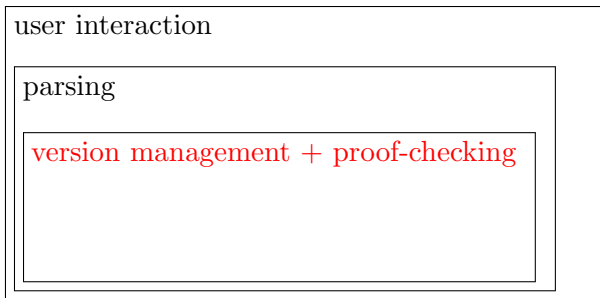
- ▶ AST representation
- ▶ Explicit dependency DAG

# Methodology



- ▶ AST representation
- ▶ Explicit dependency DAG

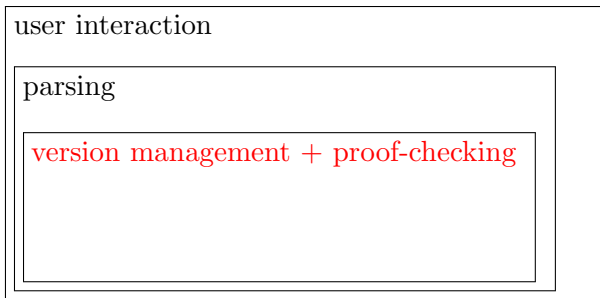
# Methodology



- ▶ AST representation
- ▶ Explicit dependency DAG
- ▶ Typing annotations



# Methodology



- ▶ AST representation
- ▶ Explicit dependency DAG
- ▶ Typing annotations
- ▶ Incremental type-checking

# A core meta-language for incremental type-checking

## Expresses

- ▶ (abstract) Syntax
- ▶ (object-) Logics
- ▶ Proofs (-terms)

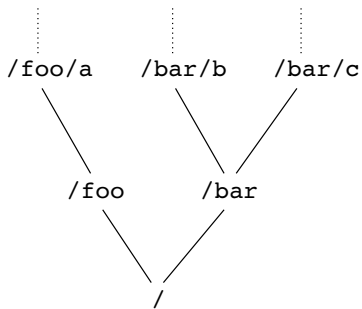
## Features

- ▶ Typing
- ▶ Incrementality
- ▶ Dependency

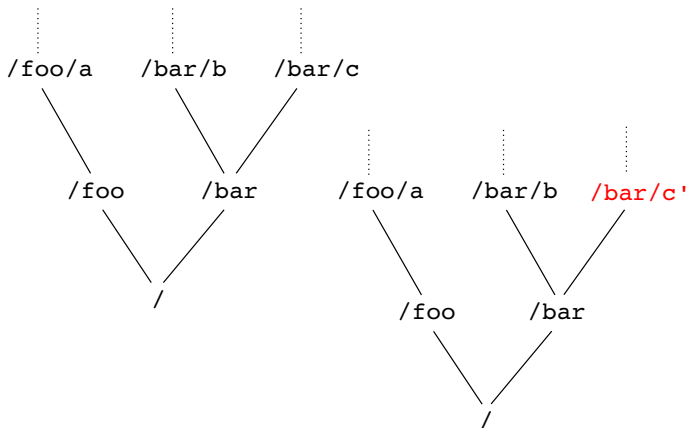
*A kernel for a typed version control system?*

# A repository of directories

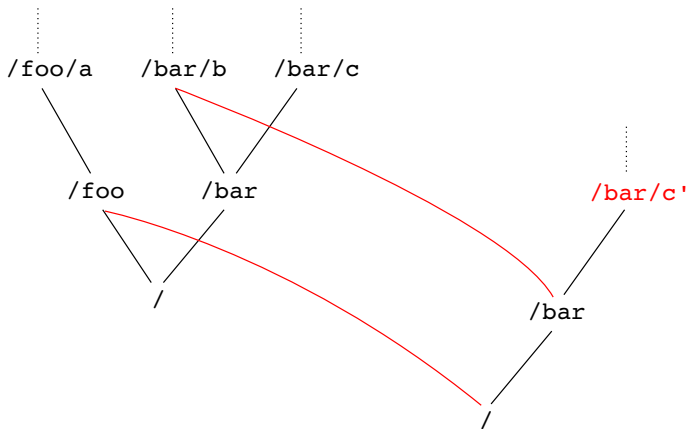
# A repository of directories



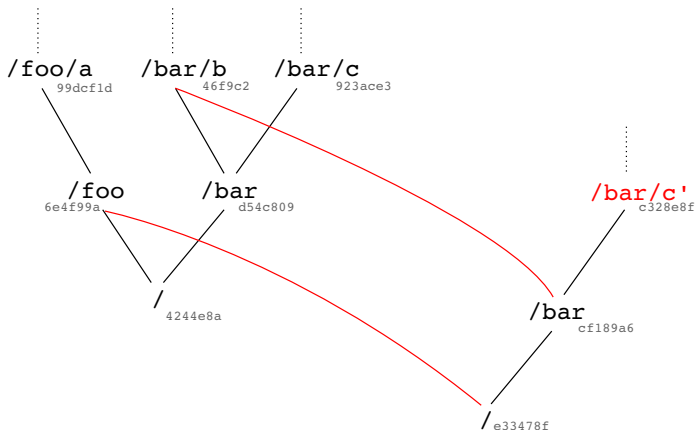
# A repository of directories



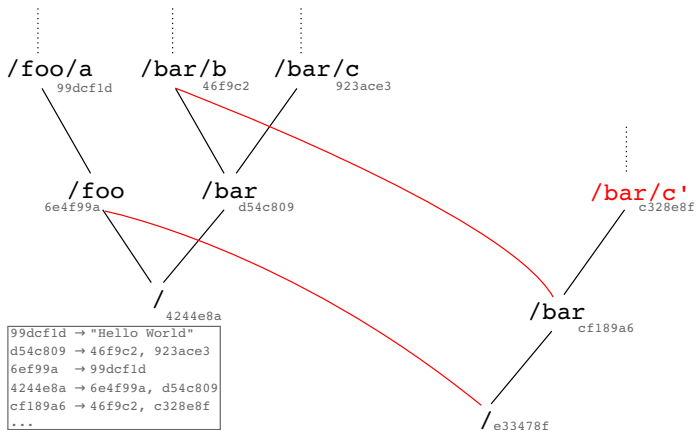
# A repository of directories



# A repository of directories

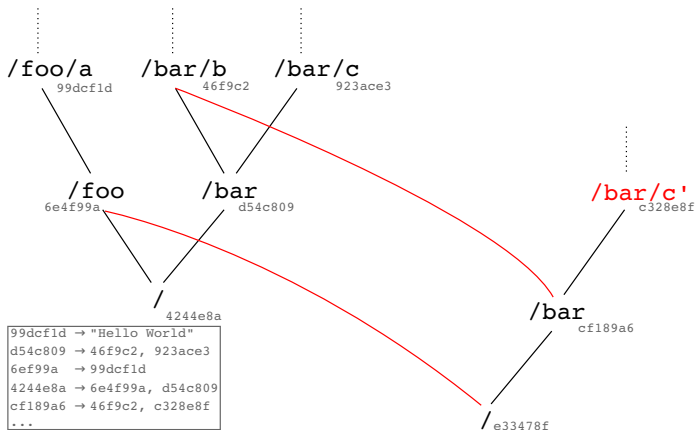


# A repository of directories



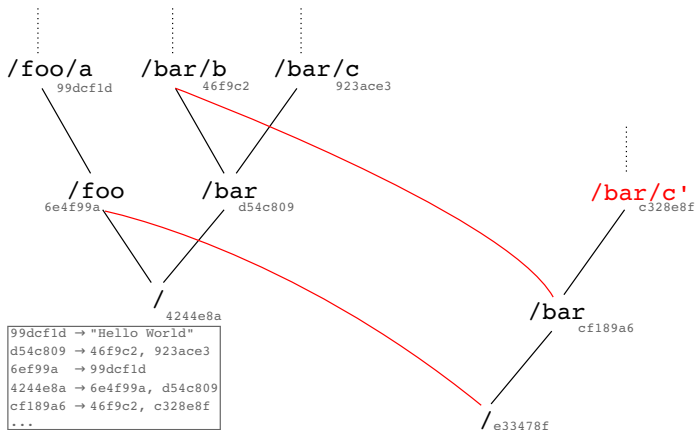


# A repository of directories



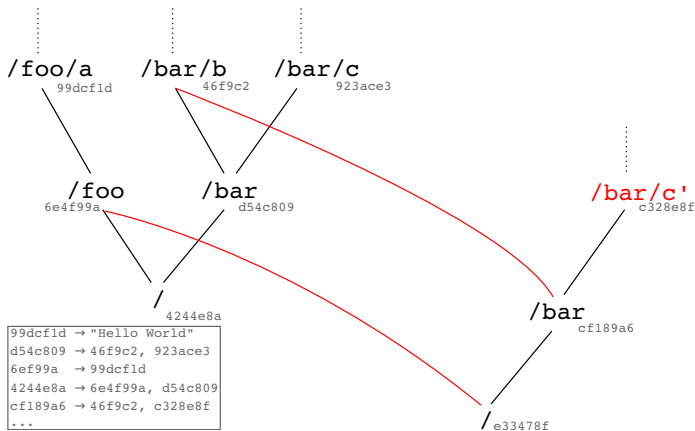
- “Content-adressable”

# A repository of directories



- “Content-adressable”
- Name reflects content

# A repository of directories



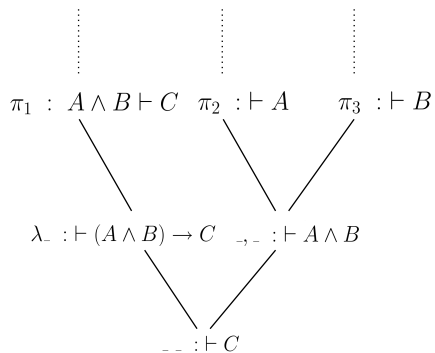
- ▶ “Content-adressable”
- ▶ Name reflects content
- ▶ Maximal sharing (or hash-consing)

# A repository of directories

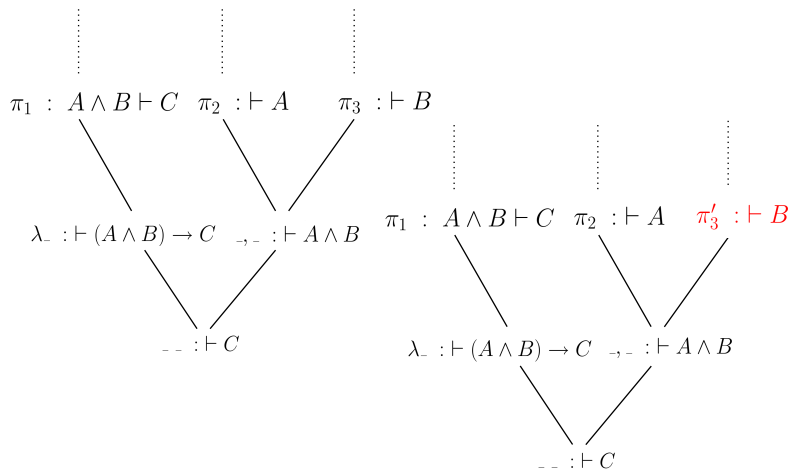
Let's do the same with *proofs*

- ▶ “Content-adressable”
- ▶ Name reflects content
- ▶ Maximal sharing (or hash-consing)

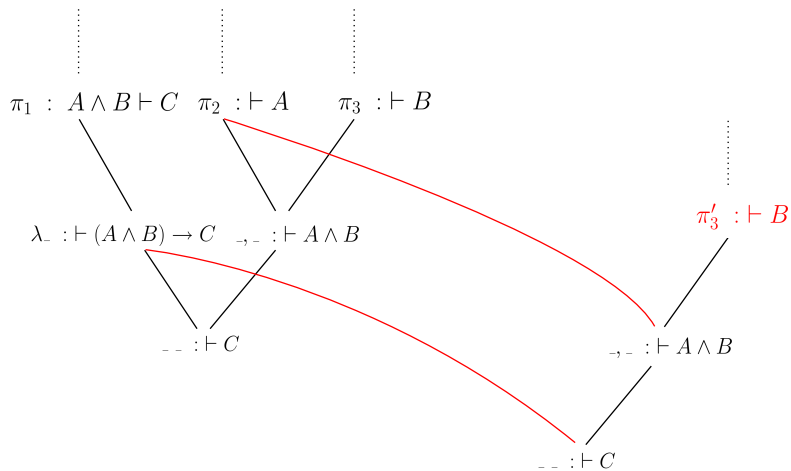
# A typed repository of proofs



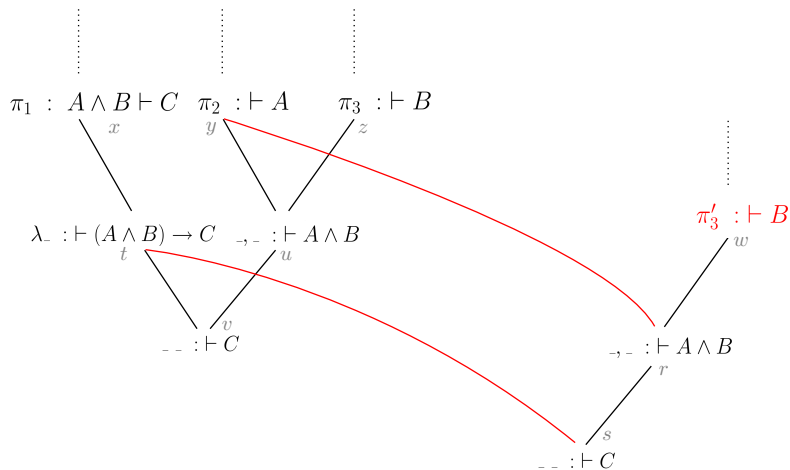
# A typed repository of proofs



# A typed repository of proofs

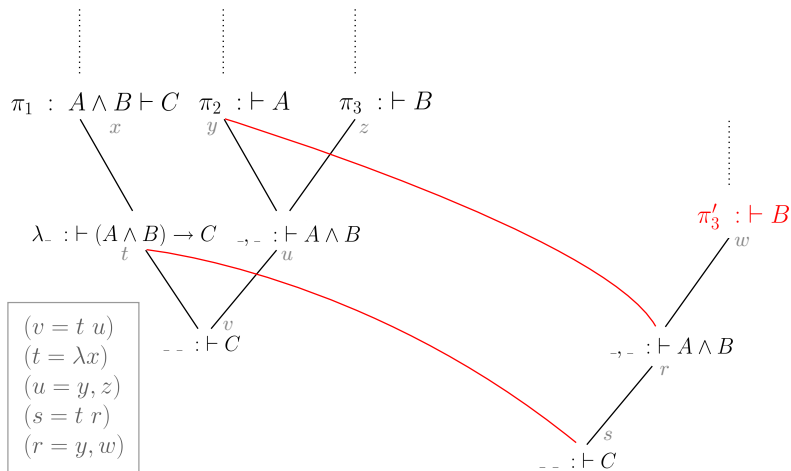


# A typed repository of proofs





# A typed repository of proofs



# Incremental type-checking, incrementally

## Syntax

$$t ::= [x : t] \cdot t \mid (x : t) \cdot t \mid x \mid t \ t \mid *$$

## Environments

$$\Gamma ::= \cdot \mid \Gamma[x : t]$$

## Judgement

$$\Gamma \vdash t : u$$

“ In environment  $\Gamma$ , term  $t$  has type  $u$  ”

# Incremental type-checking, incrementally

## Syntax

$$t ::= [x : t] \cdot t \mid (x : t) \cdot t \mid \textcolor{red}{x} \mid \textcolor{red}{t} \textcolor{red}{t} \mid *$$

## Environments

$$\Gamma ::= \cdot \mid \Gamma[x : t]$$

## Judgement

$$\Gamma \vdash t : u$$

“ In environment  $\Gamma$ , term  $t$  has type  $u$  ”

# Incremental type-checking, incrementally

## Syntax

$$t ::= [x : t] \cdot t \mid (x : t) \cdot t \mid \textcolor{red}{a} \mid *$$

$$a ::= x \mid a \ x$$

## Environments

$$\Gamma ::= \cdot \mid \Gamma[x : t]$$

## Judgement

$$\Gamma \vdash t : u$$

“ In environment  $\Gamma$ , term  $t$  has type  $u$  ”

# Incremental type-checking, incrementally

## Syntax

$$t ::= [x : t] \cdot t \mid (x : t) \cdot t \mid a \mid * \mid (x = a) \cdot t$$

$$a ::= x \mid a \ x$$

## Environments

$$\Gamma ::= \cdot \mid \Gamma[x : t] \mid \Gamma[x = a : t]$$

## Judgement

$$\Gamma \vdash t : u$$

“ In environment  $\Gamma$ , term  $t$  has type  $u$  ”

# Incremental type-checking, incrementally

## Syntax

$$t ::= [x : t] \cdot t \mid (x : t) \cdot t \mid a \mid * \mid (x = a) \cdot t$$
$$a ::= x \mid a \ x$$

## Environments

$$\Gamma ::= \cdot \mid \Gamma[x : t] \mid \Gamma[x = a : t]$$

## Judgement

$$\Gamma \vdash t : u \quad \text{“ In environment } \Gamma, \text{ term } t \text{ has type } u \text{ ”}$$

# Incremental type-checking, incrementally

## Syntax

$$t ::= (x : t) \cdot t \mid a \mid * \mid (x = a) \cdot t$$

$$a ::= x \mid a \ x$$

## Environments

$$\Gamma ::= \cdot \mid \Gamma[x : t] \mid \Gamma[x = a : t]$$

## Judgement

$$\Gamma \vdash t : u$$

“ In environment  $\Gamma$ , term  $t$  has type  $u$  ”

# Incremental type-checking, incrementally

## Syntax

$$t ::= (x : t) \cdot t \mid a \mid * \mid (x = a) \cdot t$$

$$a ::= x \mid a \ x$$

## Environments

$$\Gamma ::= \cdot \mid \Gamma[x : t] \mid \Gamma[x = a : t]$$

## Judgement

$\Gamma \vdash t : u \Rightarrow \Delta$     “From repository  $\Gamma$ , term  $t$  of type  $u$  leads to the new repository  $\Delta$ ”



# Typing rules

## Product

$$\frac{\Gamma \vdash t : * \qquad \Gamma[x : t] \vdash u : *}{\Gamma \vdash (x : t) \cdot u : *}$$

# Typing rules

## Product

$$\frac{\Gamma \vdash t : * \Rightarrow \_ \quad \Gamma[x : t] \vdash u : * \Rightarrow \Delta}{\Gamma \vdash (x : t) \cdot u : * \Rightarrow \Delta}$$

# Typing rules

## Product

$$\frac{\Gamma \vdash t : * \Rightarrow \_ \quad \Gamma[x : t] \vdash u : * \Rightarrow \Delta}{\Gamma \vdash (x : t) \cdot u : * \Rightarrow \Delta}$$

## Init

$$\frac{}{\Gamma \vdash x : t} \quad [x : t] \in \Gamma$$

# Typing rules

## Product

$$\frac{\Gamma \vdash t : * \Rightarrow \_ \quad \Gamma[x : t] \vdash u : * \Rightarrow \Delta}{\Gamma \vdash (x : t) \cdot u : * \Rightarrow \Delta}$$

## Init

$$\frac{}{\Gamma \vdash x : t \Rightarrow \Gamma} \quad [x : t] \in \Gamma$$

# Typing rules

## Equality binder

$$\frac{\Gamma \vdash a : u \Rightarrow \_ \quad \Gamma[x = a : u] \vdash t : * \Rightarrow \Delta}{\Gamma \vdash (x = a) \cdot t : * \Rightarrow \Delta} \quad [y = a : u] \notin \Gamma$$

# Typing rules

## Equality binder

$$\frac{\Gamma \vdash a : u \Rightarrow \_ \quad \Gamma[x = a : u] \vdash t : * \Rightarrow \Delta}{\Gamma \vdash (x = a) \cdot t : * \Rightarrow \Delta} \quad [y = a : u] \notin \Gamma$$

$$\frac{\Gamma \vdash t\{y/x\} : * \Rightarrow \Delta}{\Gamma \vdash (x = a) \cdot t : * \Rightarrow \Delta} \quad [y = a : u] \in \Gamma$$

# Typing rules

## Equality binder

$$\frac{\Gamma \vdash a : u \Rightarrow \_ \quad \Gamma[x = a : u] \vdash t : * \Rightarrow \Delta}{\Gamma \vdash (x = a) \cdot t : * \Rightarrow \Delta} \quad [y = a : u] \notin \Gamma$$

$$\frac{\Gamma \vdash t\{y/x\} : * \Rightarrow \Delta}{\Gamma \vdash (x = a) \cdot t : * \Rightarrow \Delta} \quad [y = a : u] \in \Gamma$$

## Application

$$\frac{\Gamma \vdash a : (y : u) \cdot t \Rightarrow \Delta}{\Gamma \vdash a \ x : t\{x/y\} \Rightarrow \Delta} \quad [x : u] \in \Gamma$$

## Content-aware names (implementation)

Given “ $a$ ”, how to decide efficiently “ $[y = a : u] \in \Gamma$ ”?



## Content-aware names (implementation)

Given “ $a$ ”, how to decide efficiently “ $[y = a : u] \in \Gamma$ ”?

$ \cdot $	:	$\vec{\kappa} \rightarrow \kappa$	Hash function
$\equiv$	:	$\kappa \rightarrow \kappa \rightarrow \mathbb{B}$	Efficient comparison
$\nu$	:	$\text{unit} \rightarrow \kappa$	Fresh key generator

## Content-aware names (implementation)

Given “ $a$ ”, how to decide efficiently “ $[y = a : u] \in \Gamma$ ”?

$ \cdot $	$:$	$\vec{\kappa} \rightarrow \kappa$	Hash function
$\equiv$	$:$	$\kappa \rightarrow \kappa \rightarrow \mathbb{B}$	Efficient comparison
$\nu$	$:$	$\text{unit} \rightarrow \kappa$	Fresh key generator

$$\frac{\Gamma \vdash a : u \Rightarrow \_ \quad \Gamma[|a| = a : u] \vdash t\{x/|a|\} : * \Rightarrow \Delta}{\Gamma \vdash (x = a) \cdot t : * \Rightarrow \Delta} \quad [y = a : u] \notin \Gamma$$

# Content-aware names (implementation)

Given “ $a$ ”, how to decide efficiently “ $[y = a : u] \in \Gamma$ ”?

$ \cdot $	$: \vec{\kappa} \rightarrow \kappa$	Hash function
$\equiv$	$: \kappa \rightarrow \kappa \rightarrow \mathbb{B}$	Efficient comparison
$\nu$	$: \text{unit} \rightarrow \kappa$	Fresh key generator

$$\frac{\Gamma \vdash a : u \Rightarrow \_ \quad \Gamma[|a| = a : u] \vdash t\{x/|a|\} : * \Rightarrow \Delta}{\Gamma \vdash (x = a) \cdot t : * \Rightarrow \Delta} \quad [y = a : u] \notin \Gamma$$

$$\frac{\Gamma \vdash t : * \Rightarrow \_ \quad \Gamma[k : t] \vdash u\{x/k\} : * \Rightarrow \Delta}{\Gamma \vdash (x : t) \cdot u : * \Rightarrow \Delta} \quad k = \nu()$$

# Content-aware names (implementation)

Given “ $a$ ”, how to decide efficiently “ $[y = a : u] \in \Gamma$ ”?

$ \cdot $	$: \vec{\kappa} \rightarrow \kappa$	Hash function
$\equiv$	$: \kappa \rightarrow \kappa \rightarrow \mathbb{B}$	Efficient comparison
$\nu$	$: \text{unit} \rightarrow \kappa$	Fresh key generator

$$\frac{\Gamma \vdash a : u \Rightarrow \_ \quad \Gamma[|a| = a : u] \vdash t\{x/|a|\} : * \Rightarrow \Delta}{\Gamma \vdash (x = a) \cdot t : * \Rightarrow \Delta} \quad [y = a : u] \notin \Gamma$$

$$\frac{\Gamma \vdash t : * \Rightarrow \_ \quad \Gamma[k : t] \vdash u\{x/k\} : * \Rightarrow \Delta}{\Gamma \vdash (x : t) \cdot u : * \Rightarrow \Delta} \quad k = \nu()$$

$$\Gamma : \kappa \rightarrow \vec{\kappa} * \tau$$

## Further Work

What if we reintroduce  $[x : t] \cdot t$  ?

## Further Work

What if we reintroduce  $[x : t] \cdot t$ ?

1. Constructive metatheory

## Further Work

What if we reintroduce  $[x : t] \cdot t$ ?

1. Constructive metatheory
2. A language to express patches?