Towards typed repositories of proofs

MIPS 2010

Matthias Puech[1,2] Yann Régis-Gianas[2]
puech@cs.unibo.it yrg@pps.jussieu.fr

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

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

July 10, 2010

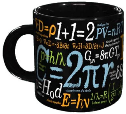
How are *constructed* formal mathematics?

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

- ((edit; compile)*; commit)* loop does not scale to proofs
- Concept freeze inhibits the discovery process
- No room for alternate definitions

Motivations

Rigidity of linear edition

- ((edit; compile)*; 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

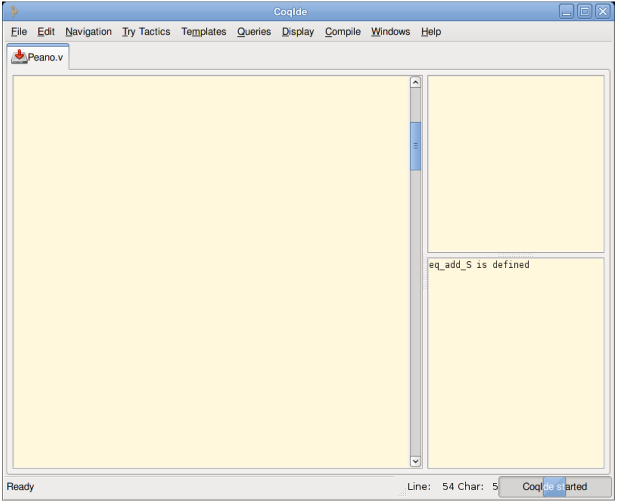
- ((edit; compile)*; commit)* loop does not scale to proofs
- Concept freeze inhibits the discovery process
- No room for alternate definitions

Laxity of textual representation

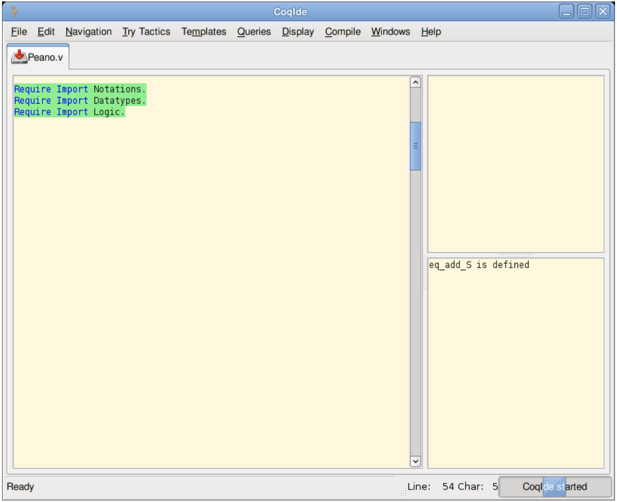
- Textual scripts diffs do not reflect the semantics
- Not even the syntax

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

The impact of changes

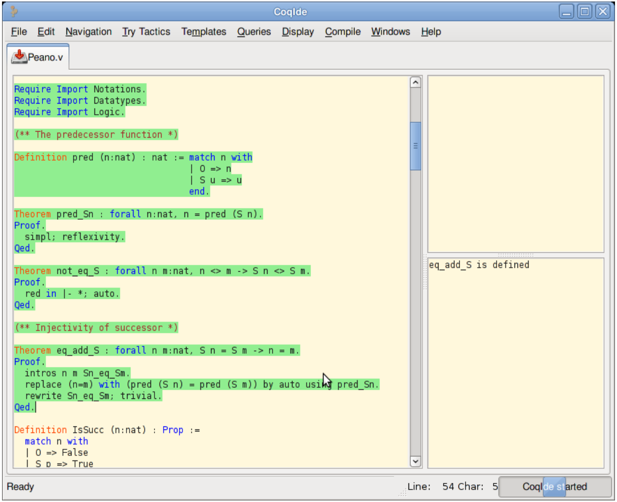


The impact of changes



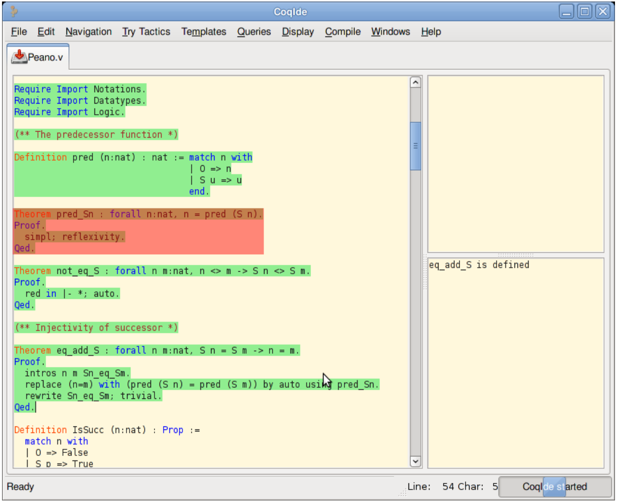
- File-based separate compilation

The impact of changes



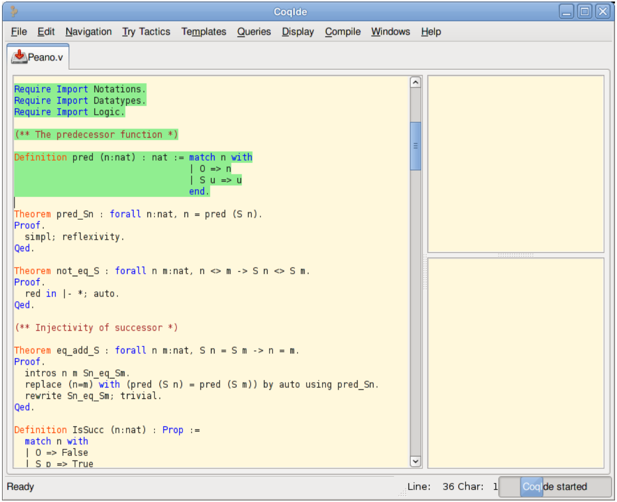
- File-based separate compilation
- Interaction loop with global undo

The impact of changes



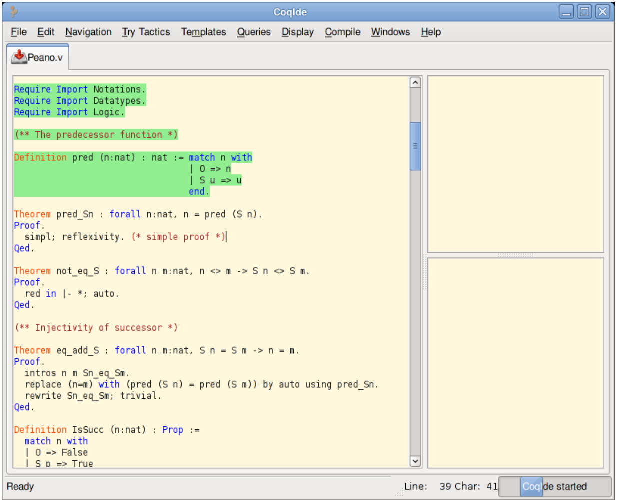
- File-based separate compilation
- Interaction loop with global undo

The impact of changes



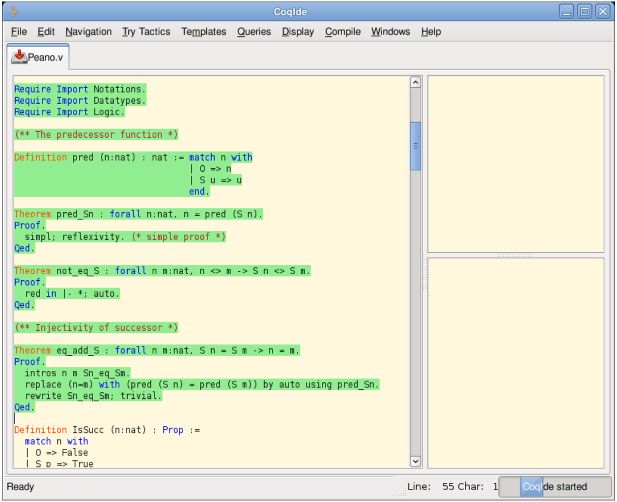
- File-based separate compilation
- Interaction loop with global undo

The impact of changes



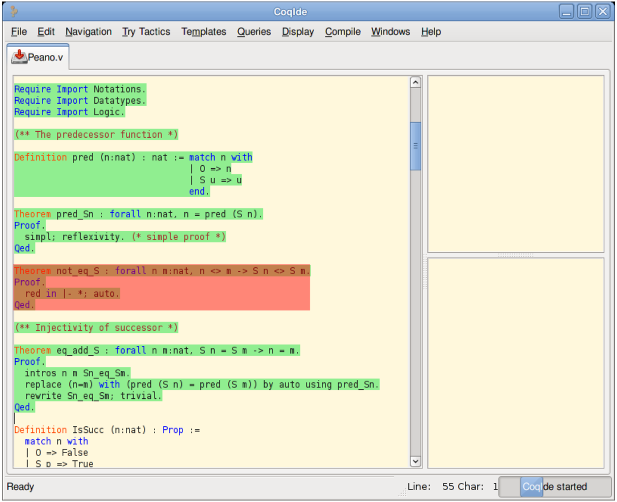
- File-based separate compilation
- Interaction loop with global undo

The impact of changes



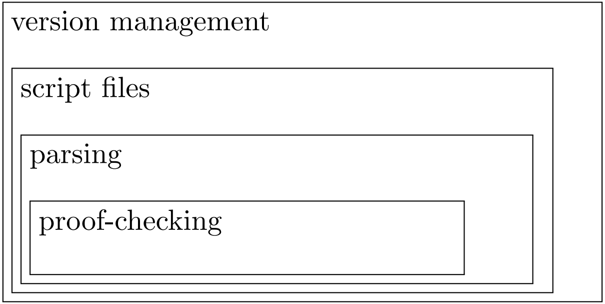
- File-based separate compilation
- Interaction loop with global undo

The impact of changes

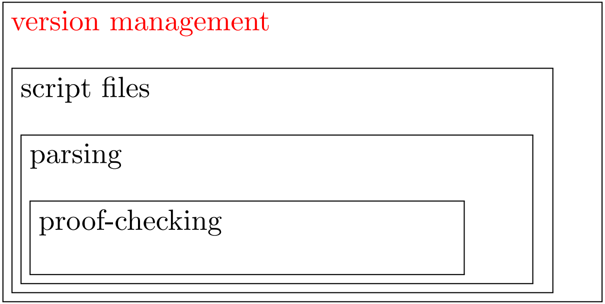


- File-based separate compilation
- Interaction loop with global undo

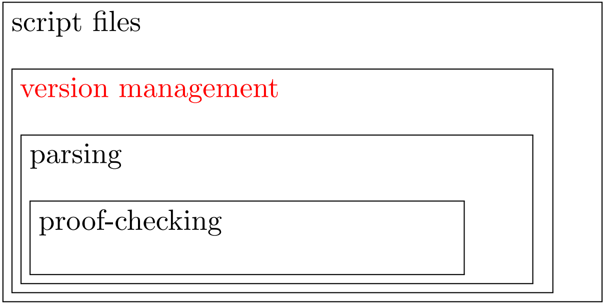
Methodology



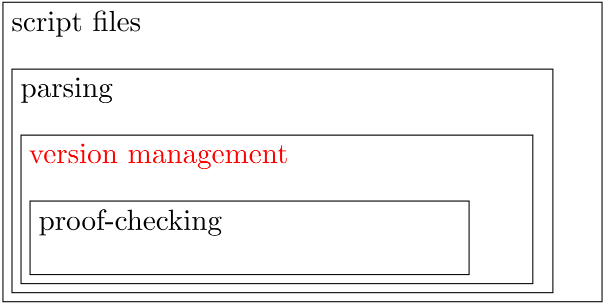
Methodology



Methodology

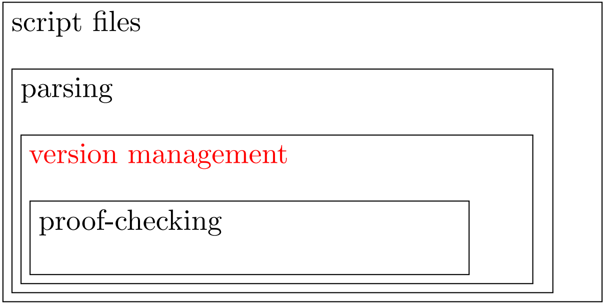


Methodology



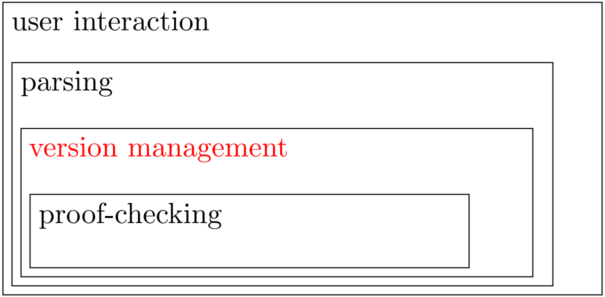
- AST representation

Methodology



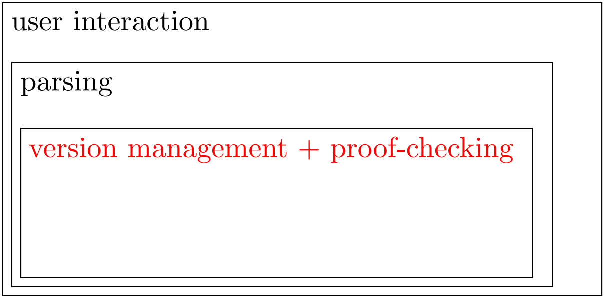
- AST representation
- Explicit dependency DAG

Methodology



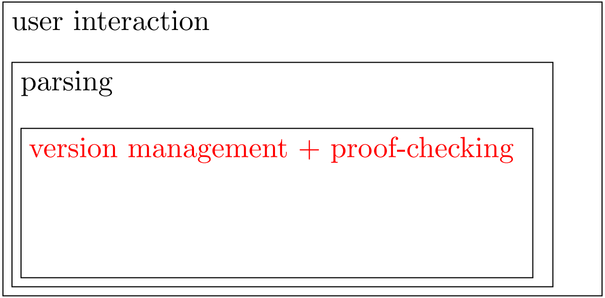
- AST representation
- Explicit dependency DAG

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)

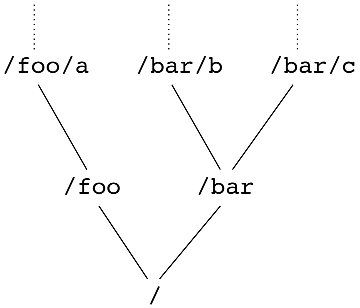
Features

- Typing
- Incrementality
- Dependency

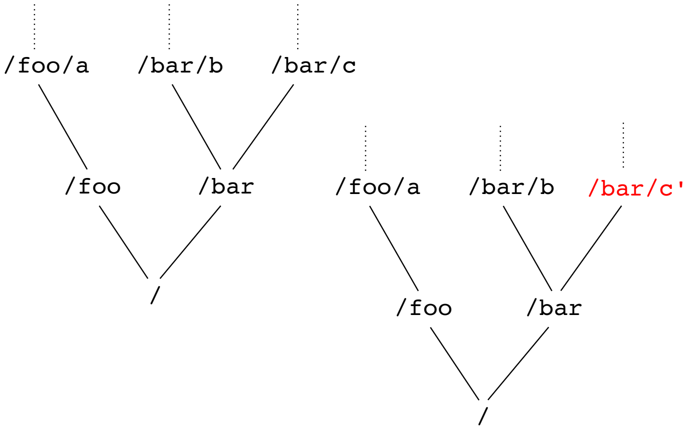
A kernel for a typed version control system?

A repository of directories

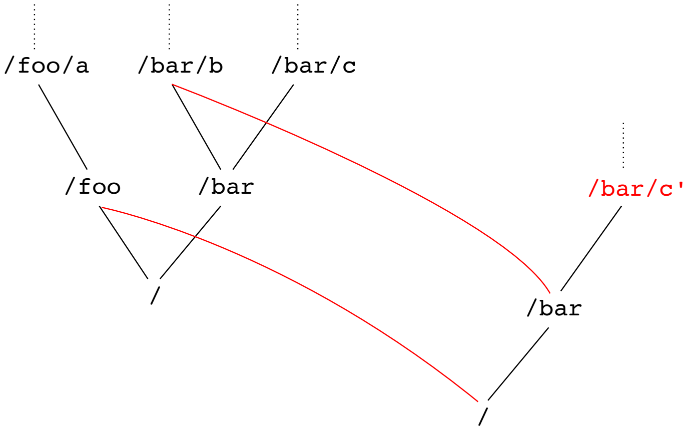
A repository of directories



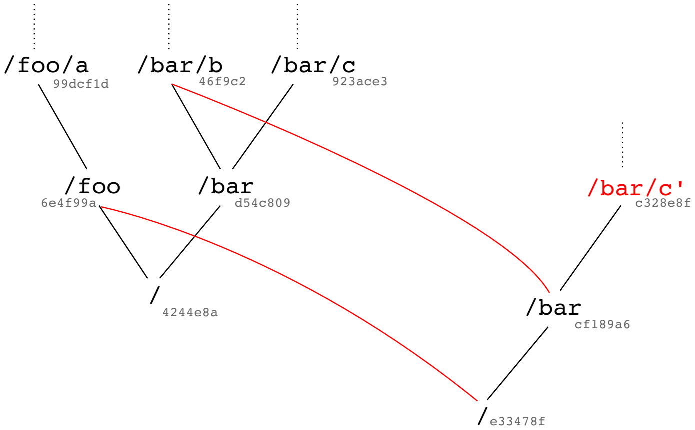
A repository of directories



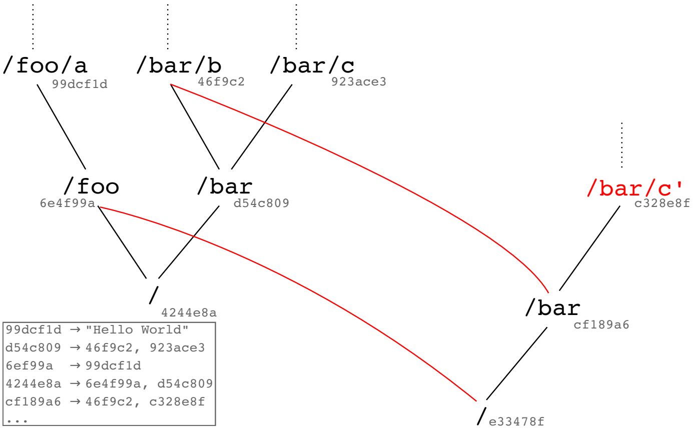
A repository of directories



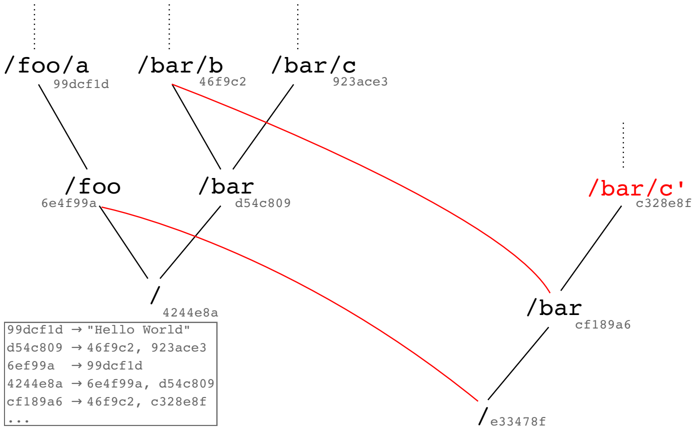
A repository of directories



A repository of directories

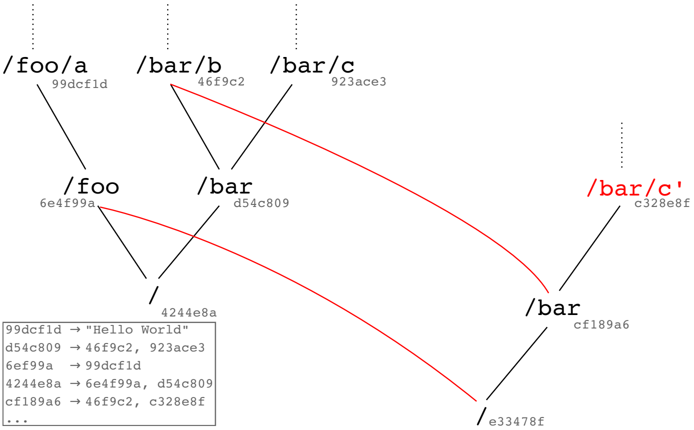


A repository of directories



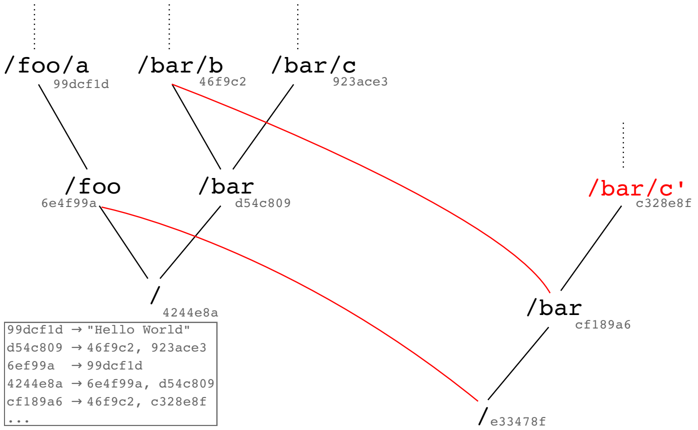
- “Content-adressable”

A repository of directories



- "Content-adressable"
- Name reflects content

A repository of directories



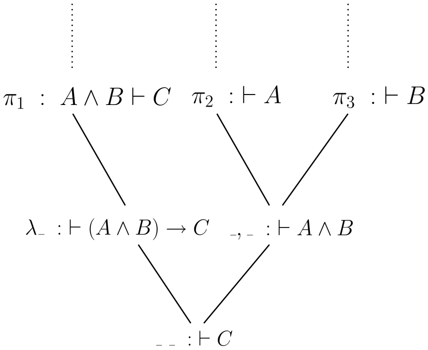
- "Content-adressable"
- Name reflects content
- Maximal sharing (or hash-consing)

A repository of directories

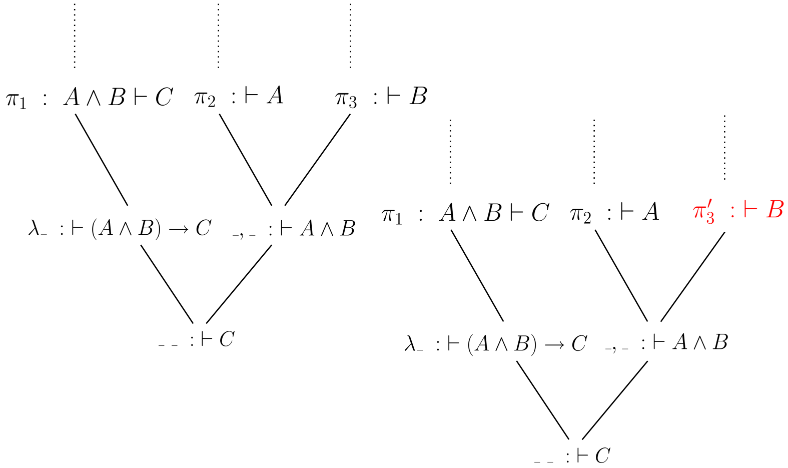
Let's do the same with *proofs*

- "Content-adressable"
- Name reflects content
- Maximal sharing (or hash-consing)

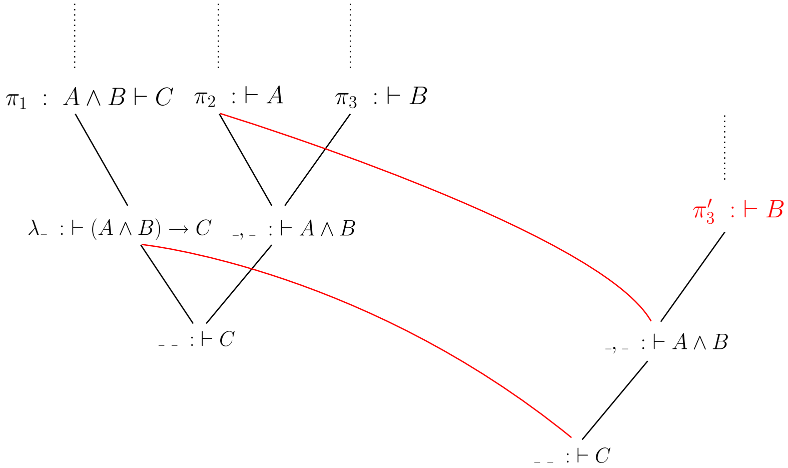
A typed repository of proofs



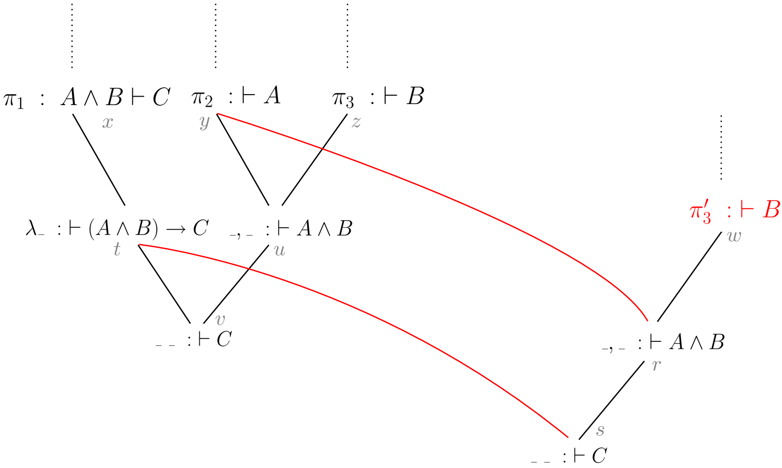
A typed repository of proofs



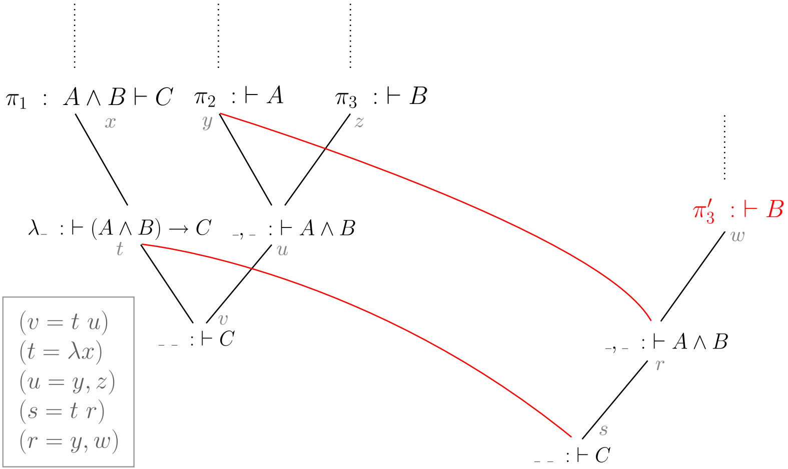
A typed repository of proofs



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 Γ, term t has type u "

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 Γ, term t has type u ”

Incremental type-checking, incrementally

Syntax

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

Environments

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

Judgement

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

Incremental type-checking, incrementally

Syntax

$$t \;::=\; (x : t) \cdot t \mid a \mid \ast \mid (x = a) \cdot t$$
$$a \;::=\; x \mid a\ x$$

Environments

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

Judgement

$\Gamma \vdash t : u$ “ In environment Γ, 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 Γ, term t of type u leads to the new repository Δ"

Typing rules

Product

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

Typing rules

Product

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

Typing rules

Product

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

Init

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

Typing rules

Product

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

Init

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

Typing rules

Equality binder

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

Typing rules

Equality binder

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

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

Typing rules

Equality binder

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

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

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 "$[y = a : u] \in \Gamma$"?

$$
\begin{array}{lcll}
|\cdot| & : & \vec{\kappa} \to \kappa & \text{Hash function} \\
\equiv & : & \kappa \to \kappa \to \mathbb{B} & \text{Efficient comparison} \\
\nu & : & \mathsf{unit} \to \kappa & \text{Fresh key generator}
\end{array}
$$

Content-aware names (implementation)

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

$$\begin{array}{lcll} |\cdot| & : & \vec{\kappa} \to \kappa & \text{Hash function} \\ \equiv & : & \kappa \to \kappa \to \mathbb{B} & \text{Efficient comparison} \\ \nu & : & \mathsf{unit} \to \kappa & \text{Fresh key generator} \end{array}$$

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

Content-aware names (implementation)

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

$$\begin{array}{lcll} |\cdot| & : & \vec{\kappa} \to \kappa & \text{Hash function} \\ \equiv & : & \kappa \to \kappa \to \mathbb{B} & \text{Efficient comparison} \\ \nu & : & \mathsf{unit} \to \kappa & \text{Fresh key generator} \end{array}$$

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

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

Content-aware names (implementation)

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

$$
\begin{array}{lcll}
|\cdot| & : & \vec{\kappa} \to \kappa & \text{Hash function} \\
\equiv & : & \kappa \to \kappa \to \mathbb{B} & \text{Efficient comparison} \\
\nu & : & \mathsf{unit} \to \kappa & \text{Fresh key generator}
\end{array}
$$

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

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

$$
\Gamma \;:\; \kappa \to \vec{\kappa} * \tau
$$

Further Work

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

Further Work

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

1. Constructive metatheory

Further Work

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

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