Formal verification in Coq of program properties involving the global state effect

Burak Ekici*
with J.-G. Dumas*, D. Duval*, D. Pous**

* LJK Grenoble, France
** ENS Lyon, France

December 5, 2013

Outline

1. Motivation
2. The States Effect
3. Coq Basics
4. States Effect in Coq
5. A Proof in Coq

Motivation

- Verifying properties of programs involving computational (side) effects such as:
  - State
  - Exceptions
  - IO
  - Partiality
  - ...
- Developing related `Coq libraries` for each effect and composing them.

Motivation

- Verifying properties of programs involving computational (side) effects such as:
  - State
  - Exceptions
  - IO
  - Partiality
  - ...
- Developing related `Coq libraries` for each effect and composing them.

The State

State of a program:

- the snapshot of the memory locations (variables) at any point during execution
- not syntactically mentioned
  - `int x = 3;`
- can be viewed as `set` or `array` of locations denoted by `S`

x	y	z	t	u	v
1	2	3	4	5	6

- provides an access to the memory via an interface:
  - $\texttt{update}_{\texttt{x}}(3)$; $\texttt{lookup}_{\texttt{x}}$;
  - `x = 3; !x`

N.B. Any access (for any reason: `update` or `lookup`) to the memory is defined as a computational effect.

The State

State of a program:

- the snapshot of the memory locations (variables) at any point during execution
- not syntactically mentioned
  - `int x = 3;`
- can be viewed as `set` or `array` of locations denoted by `S`

x	y	z	t	u	v
1	2	3	4	5	6

- provides an access to the memory via an interface:
  - $\texttt{update}_{\texttt{x}}(3)$; $\texttt{lookup}_{\texttt{x}}$;
  - `x = 3; !x`

N.B. Any access (for any reason: `update` or `lookup`) to the memory is defined as a computational effect.

The mismatch between syntax and interpretation

$\texttt{update}_x$:
`x = 3;`

- in syntax: $\texttt{int} \to \texttt{void}$
- in an interpretation: $\texttt{S} \times \texttt{int} \to \texttt{S}$

$\texttt{lookup}_x$:
`!x`

- in syntax: $\texttt{void} \to \texttt{int}$
- in an interpretation: $\texttt{S} \to \texttt{int}$

The mismatch between syntax and interpretation

```
int x, y;
void f (void) {
  x = 3;
  y = 4;
}
```

In syntax:

$$f : \quad \texttt{void} \xrightarrow{\texttt{three}} \texttt{int} \xrightarrow{\texttt{update}_x} \texttt{void} \xrightarrow{\texttt{four}} \texttt{int} \xrightarrow{\texttt{update}_y} \texttt{void}$$
$$* \longmapsto 3 \longmapsto * \longmapsto 4 \longmapsto *$$

In an interpretation:

$$f : \quad S \xrightarrow{\texttt{three}} \texttt{int} \times S \xrightarrow{\texttt{update}_x} S \xrightarrow{\texttt{four}} \texttt{int} \times S \xrightarrow{\texttt{update}_y} S$$
$$s \longmapsto (3, s) \longmapsto s_1 \longmapsto (4, s_1) \longmapsto s_2$$

where $s_1 := s[x \leftarrow 3]$ and $s_2 := s_1[y \leftarrow 4]$

The mismatch between syntax and interpretation

```
int x, y;
void f (void) {
  x = 3;
  y = 4;
}
```

In syntax:

$$\mathtt{f}: \quad \mathtt{void} \xrightarrow{\mathtt{three}} \mathtt{int} \xrightarrow{\mathtt{update_x}} \mathtt{void} \xrightarrow{\mathtt{four}} \mathtt{int} \xrightarrow{\mathtt{update_y}} \mathtt{void}$$
$$* \longmapsto 3 \longmapsto * \longmapsto 4 \longmapsto *$$

In an interpretation:

$$\mathtt{f}: \quad S \xrightarrow{\mathtt{three}} \mathtt{int} \times S \xrightarrow{\mathtt{update_x}} S \xrightarrow{\mathtt{four}} \mathtt{int} \times S \xrightarrow{\mathtt{update_y}} S$$
$$s \longmapsto (3, s) \longmapsto s_1 \longmapsto (4, s_1) \longmapsto s_2$$

where $s_1 := s[x \leftarrow 3]$ and $s_2 := s_1[y \leftarrow 4]$

The mismatch between syntax and interpretation

```
int x, y;
void f (void) {
  x = 3;
  y = 4;
}
```

In syntax:

$$\texttt{f}: \quad \texttt{void} \xrightarrow{\texttt{three}} \texttt{int} \xrightarrow{\texttt{update}_x} \texttt{void} \xrightarrow{\texttt{four}} \texttt{int} \xrightarrow{\texttt{update}_y} \texttt{void}$$
$$* \longmapsto 3 \longmapsto * \longmapsto 4 \longmapsto *$$

In an interpretation:

$$\texttt{f}: \quad \texttt{S} \xrightarrow{\texttt{three}} \texttt{int} \times \texttt{S} \xrightarrow{\texttt{update}_x} \texttt{S} \xrightarrow{\texttt{four}} \texttt{int} \times \texttt{S} \xrightarrow{\texttt{update}_y} \texttt{S}$$
$$\texttt{s} \longmapsto (3, \texttt{s}) \longmapsto \texttt{s}_1 \longmapsto (4, \texttt{s}_1) \longmapsto \texttt{s}_2$$

where $\texttt{s}_1 := \texttt{s}[\texttt{x} \leftarrow 3]$ and $\texttt{s}_2 := \texttt{s}_1[\texttt{y} \leftarrow 4]$

How to prove program equivalences

```
int x, y;
void f (void) {
  x = 3;
  y = 4;
}
```

```
int x, y;
void g (void) {
  y = 4;
  x = 3;
}
```

In syntax:

$$f, g : \quad \texttt{void} \longrightarrow \texttt{void}$$
$$* \longmapsto *$$

In an interpretation:

$$f = g : \quad S \longrightarrow S$$
$$s \longmapsto s_2$$

How to prove the equivalence of f and g without mentioning the state?

How to prove program equivalences

```
int x, y;
void f (void) {
  x = 3;
  y = 4;
}
```

```
int x, y;
void g (void) {
  y = 4;
  x = 3;
}
```

In syntax:

$$f, g : \quad \texttt{void} \longrightarrow \texttt{void}$$

$$* \longmapsto *$$

In an interpretation:

$$f = g : \quad S \longrightarrow S$$

$$s \longmapsto s_2$$

How to prove the equivalence of f and g without mentioning the state?

How to prove program equivalences

```
int x, y;
void f (void) {
  x = 3;
  y = 4;
}
```

```
int x, y;
void g (void) {
  y = 4;
  x = 3;
}
```

In syntax:

$$\mathtt{f}, \mathtt{g} : \quad \mathtt{void} \longrightarrow \mathtt{void}$$
$$* \longmapsto *$$

In an interpretation:

$$\mathtt{f} = \mathtt{g} : \quad \mathtt{S} \longrightarrow \mathtt{S}$$
$$\mathtt{s} \longmapsto \mathtt{s}_2$$

How to prove the equivalence of f and g without mentioning the state?

How to prove program equivalences

```
int x, y;
void f (void) {
  x = 3;
  y = 4;
}
```

```
int x, y;
void g (void) {
  y = 4;
  x = 3;
}
```

In syntax:

$$\mathtt{f}, \mathtt{g} : \quad \mathtt{void} \longrightarrow \mathtt{void}$$
$$* \longmapsto *$$

In an interpretation:

$$\mathtt{f} = \mathtt{g} : \quad \mathtt{S} \longrightarrow \mathtt{S}$$
$$\mathtt{s} \longmapsto \mathtt{s}_2$$

How to prove the equivalence of f and g without mentioning the state?

How to prove program equivalences

- Decorating the interface (approach by Dumas et al.'[12])
  - to be able to deal with more interpretations of the state structure
  - keep interface functions closer to syntax

Decorations for States Effect

Let $\mathtt{x}$ be a location and $\mathtt{Val_x}$ be the set of values that could be stored in $\mathtt{x}$. E.g., $\mathtt{int}$

Functions are classified and decorated:

- pure: e.g., $\mathtt{id}_{\mathtt{x}}^{pure} : \mathtt{Val_x} \to \mathtt{Val_x}$, $\mathtt{forget}_{\mathtt{x}}^{pure} : \mathtt{Val_x} \to \mathtt{void}$
- accessors: e.g., $\mathtt{lookup}_{\mathtt{x}}^{ro} : \mathtt{void} \to \mathtt{Val_x}$
- modifiers: e.g., $\mathtt{update}_{\mathtt{x}}^{rw} : \mathtt{Val_x} \to \mathtt{void}$
- Hierarchy rules among functions: $\dfrac{\mathtt{f}^{pure}}{\mathtt{f}^{ro}}$, $\dfrac{\mathtt{f}^{ro}}{\mathtt{f}^{rw}}$

N.B.

Decorations specify the effects of the functions on the state.

Signatures explain what functions do.

Decorations for States Effect

Let $\mathtt{x}$ be a location and $\mathtt{Val_x}$ be the set of values that could be stored in $\mathtt{x}$. E.g., $\mathtt{int}$

Functions are classified and decorated:

- pure: e.g., $\mathtt{id}_{\mathtt{x}}^{pure} : \mathtt{Val_x} \to \mathtt{Val_x}$, $\mathtt{forget}_{\mathtt{x}}^{pure} : \mathtt{Val_x} \to \mathtt{void}$
- accessors: e.g., $\mathtt{lookup}_{\mathtt{x}}^{ro} : \mathtt{void} \to \mathtt{Val_x}$
- modifiers: e.g., $\mathtt{update}_{\mathtt{x}}^{rw} : \mathtt{Val_x} \to \mathtt{void}$
- Hierarchy rules among functions: $\dfrac{\mathtt{f}^{pure}}{\mathtt{f}^{ro}}$, $\dfrac{\mathtt{f}^{ro}}{\mathtt{f}^{rw}}$

N.B.
`Decorations` specify the effects of the functions on the state.

`Signatures` explain what functions do.

The "Luring" Trick for States: Operations

$$\begin{array}{lcl} \mathtt{f}^{pure} & : & \mathtt{X} \to \mathtt{Y} \\ \mathtt{f}^{ro} & : & \mathtt{X} \to \mathtt{Y} \\ \mathtt{f}^{rw} & : & \mathtt{X} \to \mathtt{Y} \end{array}$$

specify the decoration ↗

explain the decoration ↘

$$\mathtt{f} \ : \ \mathtt{X} \to \mathtt{Y}$$

$$\begin{array}{lcl} \mathtt{f} & : & \mathtt{X} \to \mathtt{Y} \\ \mathtt{f} & : & \mathtt{X} \times \mathtt{S} \to \mathtt{Y} \\ \mathtt{f} & : & \mathtt{X} \times \mathtt{S} \to \mathtt{Y} \times \mathtt{S} \end{array}$$

States: Equations

Equations: $\mathtt{f} = \mathtt{g} : \mathtt{X} \to \mathtt{Y}$

Decorations on equations:

- $\mathtt{f}^{rw} == \mathtt{g}^{rw}$ if the equation is strong (result + effect equivalence)
- $\mathtt{f}^{rw} \sim \mathtt{g}^{rw}$ if the equation is weak (result equivalence)

Hierarchy Rules:

- $\mathtt{f}^{rw} == \mathtt{g}^{rw} \Longrightarrow \mathtt{f}^{rw} \sim \mathtt{g}^{rw}$
- if $\mathtt{f}^{ro}$ and $\mathtt{g}^{ro}$, then $\mathtt{f}^{ro} == \mathtt{g}^{ro} \Longleftrightarrow \mathtt{f}^{ro} \sim \mathtt{g}^{ro}$

States: Equations

Equations: $\mathtt{f} = \mathtt{g} : \mathtt{X} \to \mathtt{Y}$

Decorations on equations:

- $\mathtt{f}^{rw} == \mathtt{g}^{rw}$ if the equation is strong (result + effect equivalence)
- $\mathtt{f}^{rw} \sim \mathtt{g}^{rw}$ if the equation is weak (result equivalence)

Hierarchy Rules:

- $\mathtt{f}^{rw} == \mathtt{g}^{rw} \Longrightarrow \mathtt{f}^{rw} \sim \mathtt{g}^{rw}$
- if $\mathtt{f}^{ro}$ and $\mathtt{g}^{ro}$, then $\mathtt{f}^{ro} == \mathtt{g}^{ro} \Longleftrightarrow \mathtt{f}^{ro} \sim \mathtt{g}^{ro}$

The "Luring" Trick for States: Equations

$$\begin{array}{lcl} \mathtt{f} == \mathtt{g} & : & \mathtt{X} \to \mathtt{Y} \\ \mathtt{f} \sim \mathtt{g} & : & \mathtt{X} \to \mathtt{Y} \end{array}$$

specify the decoration ↗

↘ explain the decoration

$$\mathtt{f} = \mathtt{g} \quad : \quad \mathtt{X} \to \mathtt{Y}$$

$$\begin{array}{lcl} \mathtt{f} = \mathtt{g} & : & \mathtt{X} \times \mathtt{S} \to \mathtt{Y} \times \mathtt{S} \\ \pi_{\mathtt{Y}} \circ \mathtt{f} = \pi_{\mathtt{Y}} \circ \mathtt{g} & : & \mathtt{X} \times \mathtt{S} \to \mathtt{Y} \end{array}$$

Basic Operations: Update & Lookup

For each location x and y where $x \neq y$, there are two main operations and equations:

$$\mathtt{lookup}_x : \mathtt{void} \to \mathtt{int}$$
$$\mathtt{update}_x : \mathtt{int} \to \mathtt{void}$$

$$\mathtt{lookup}_x \circ \mathtt{update}_x \sim \mathtt{id}_x \quad \text{(axiom-1)}$$

$$\mathtt{lookup}_y \circ \mathtt{update}_x \sim \mathtt{lookup}_y \circ \mathtt{forget}_x \quad \text{(axiom-2)}$$

$\mapsto$

$$\mathtt{lookup}_x : \mathtt{S} \to \mathtt{int}$$
$$\mathtt{update}_x : \mathtt{int} \times \mathtt{S} \to \mathtt{S}$$

$$(3, \begin{array}{|c|}\hline x \\ \hline * \\ \hline \end{array}) \mapsto \begin{array}{|c|}\hline x \\ \hline 3 \\ \hline \end{array} \mapsto 3 \sim (3, \begin{array}{|c|}\hline x \\ \hline * \\ \hline \end{array}) \mapsto 3$$

$$(3, \begin{array}{|c|c|}\hline x & y \\ \hline * & 5 \\ \hline \end{array}) \mapsto \begin{array}{|c|c|}\hline x & y \\ \hline 3 & 5 \\ \hline \end{array} \mapsto 5 \sim$$

$$(3, \begin{array}{|c|c|}\hline x & y \\ \hline * & 5 \\ \hline \end{array}) \mapsto \begin{array}{|c|c|}\hline x & y \\ \hline * & 5 \\ \hline \end{array} \mapsto 5$$

Strong Equality: Compatibility w.r.t. composition

$$(s\text{-subs})\ \frac{f^{rw} \quad g_1^{rw} == g_2^{rw}}{g_1 \circ f == g_2 \circ f}$$

$$(s\text{-repl})\ \frac{f_1^{rw} == f_2^{rw} \quad g^{rw}}{g \circ f_1 == g \circ f_2}$$

Weak Equality: Compatibility w.r.t. composition

$$(w\text{-subs})\ \frac{f^{rw} \quad g_1^{rw} \sim g_2^{rw}}{g_1 \circ f \sim g_2 \circ f}$$

$$(\text{pure-}w\text{-repl})\ \frac{f_1^{rw} \sim f_2^{rw} \quad g^{pure}}{g \circ f_1 \sim g \circ f_2}$$

Weak Equality: Compatibility w.r.t. composition

$$(w\text{-subs})\ \frac{f^{rw} \quad g_1^{rw} \sim g_2^{rw}}{g_1 \circ f \sim g_2 \circ f}$$

$$(\text{pure-}w\text{-repl})\ \frac{f_1^{rw} \sim f_2^{rw} \quad g^{pure}}{g \circ f_1 \sim g \circ f_2}$$

Weak Equality: Compatibility w.r.t. composition

$$(w\text{-subs})\ \frac{f^{rw} \quad g_1^{rw} \sim g_2^{rw}}{g_1 \circ f \sim g_2 \circ f}$$

$$(\text{pure-}w\text{-repl})\ \frac{f_1^{rw} \sim f_2^{rw} \quad g^{pure}}{g \circ f_1 \sim g \circ f_2}$$

`the state before` f_1^{rw} `and` f_2^{rw}

x	y	z	t	u	v
1	2	3	4	5	6

Weak Equality: Compatibility w.r.t. composition

$$(w\text{-subs})\ \frac{f^{rw} \quad g_1^{rw} \sim g_2^{rw}}{g_1 \circ f \sim g_2 \circ f}$$

$$(\text{pure-}w\text{-repl})\ \frac{f_1^{rw} \sim f_2^{rw} \quad g^{pure}}{g \circ f_1 \sim g \circ f_2}$$

after $f_1^{rw} := \texttt{update}_{\texttt{u}}(9)$

x	y	z	t	u	v
1	2	3	4	9	6

returns void

Weak Equality: Compatibility w.r.t. composition

$$(w\text{-subs})\ \frac{f^{rw} \quad g_1^{rw} \sim g_2^{rw}}{g_1 \circ f \sim g_2 \circ f}$$

$$(\text{pure-}w\text{-repl})\ \frac{f_1^{rw} \sim f_2^{rw} \quad g^{pure}}{g \circ f_1 \sim g \circ f_2}$$

after $f_1^{rw} := \texttt{update}_\texttt{u}(9)$

x	y	z	t	u	v
1	2	3	4	9	6

returns void

after $f_2^{rw} := \texttt{forget}_\texttt{u}(9)$

x	y	z	t	u	v
1	2	3	4	5	6

returns void

Weak Equality: Compatibility w.r.t. composition

$$(w\text{-subs})\ \frac{f^{rw} \quad g_1^{rw} \sim g_2^{rw}}{g_1 \circ f \sim g_2 \circ f}$$

$$(\text{pure-}w\text{-repl})\ \frac{f_1^{rw} \sim f_2^{rw} \quad g^{pure}}{g \circ f_1 \sim g \circ f_2}$$

`after g` $[:= \texttt{lookup}_{\texttt{u}}^{ro}] \circ f_1^{rw}$

x	y	z	t	u	v
1	2	3	4	9	6

`returns 9`

Weak Equality: Compatibility w.r.t. composition

$$(w\text{-subs})\ \frac{f^{rw} \quad g_1^{rw} \sim g_2^{rw}}{g_1 \circ f \sim g_2 \circ f}$$

$$(\text{pure-}w\text{-repl})\ \frac{f_1^{rw} \sim f_2^{rw} \quad g^{pure}}{g \circ f_1 \sim g \circ f_2}$$

after g $[:= \text{lookup}_{\text{u}}^{ro}] \circ f_1^{rw}$

x	y	z	t	u	v
1	2	3	4	9	6

returns 9

after g $[:= \text{lookup}_{\text{u}}^{ro}] \circ f_2^{rw}$

x	y	z	t	u	v
1	2	3	4	5	6

returns 5

Weak Equality: Compatibility w.r.t. composition

$$(w\text{-subs})\ \frac{f^{rw} \quad g_1^{rw} \sim g_2^{rw}}{g_1 \circ f \sim g_2 \circ f}$$

$$(\text{pure-}w\text{-repl})\ \frac{f_1^{rw} \sim f_2^{rw} \quad g^{pure}}{g \circ f_1 \sim g \circ f_2}$$

$$\mathbf{g}^{ro} \circ f_1^{rw} \nsim \mathbf{g}^{ro} \circ f_2^{rw}$$

The Rule of Observation

Two `weakly` equal functions are `strongly equal` if the state looks the same after their evaluations.

$$\frac{\begin{array}{l}(\text{observation}) \\ f^{rw}, g^{rw} : X \to \texttt{void} \quad \forall\ \texttt{k},\ (\texttt{lookup}^{ro}_{\texttt{k}} \circ f^{rw}) \sim (\texttt{lookup}^{ro}_{\texttt{k}} \circ g^{rw})\end{array}}{f == g}$$

The Rule of Observation

Two `weakly` equal functions are `strongly equal` if the state looks the same after their evaluations.

$$\frac{\text{(observation)}\quad f^{rw}, g^{rw} : X \to \texttt{void} \quad \forall\ \texttt{k},\ (\texttt{lookup}^{ro}_{\texttt{k}} \circ f^{rw}) \sim (\texttt{lookup}^{ro}_{\texttt{k}} \circ g^{rw})}{f == g}$$

after f

x	y	z	t	u	v
1	2	3	4	5	6

after g

x	y	z	t	u	v
1	2	3	4	5	6

The Rule of Observation

Two `weakly` equal functions are `strongly equal` if the state looks the same after their evaluations.

$$\frac{\text{(observation)}\quad f^{rw}, g^{rw} : X \to \texttt{void} \quad \forall\ \texttt{k},\ (\texttt{lookup}^{ro}_{\texttt{k}} \circ f^{rw}) \sim (\texttt{lookup}^{ro}_{\texttt{k}} \circ g^{rw})}{f == g}$$

after f

x	y	z	t	u	v
1	2	3	4	5	6

after g

x	y	z	t	u	v
1	2	3	4	5	6

The Rule of Observation

Two `weakly` equal functions are `strongly equal` if the state looks the same after their evaluations.

$$\frac{\text{(observation)}\quad f^{rw}, g^{rw} : X \to \texttt{void} \quad \forall\ \texttt{k},\ (\texttt{lookup}^{ro}_{\texttt{k}} \circ f^{rw}) \sim (\texttt{lookup}^{ro}_{\texttt{k}} \circ g^{rw})}{f == g}$$

after f

x	y	z	t	u	v
1	2	3	4	5	6

after g

x	y	z	t	u	v
1	2	3	4	5	6

The Rule of Observation

Two `weakly` equal functions are `strongly equal` if the state looks the same after their evaluations.

$$\frac{\text{(observation)}\quad f^{rw}, g^{rw} : X \to \texttt{void} \quad \forall\ \texttt{k},\ (\texttt{lookup}^{ro}_{\texttt{k}} \circ f^{rw}) \sim (\texttt{lookup}^{ro}_{\texttt{k}} \circ g^{rw})}{f == g}$$

after f

x	y	z	t	u	v
1	2	3	4	5	6

after g

x	y	z	t	u	v
1	2	3	4	5	6

The Rule of Observation

Two `weakly` equal functions are `strongly equal` if the state looks the same after their evaluations.

$$\frac{\text{(observation)}\quad f^{rw}, g^{rw} : X \to \texttt{void} \quad \forall\ \texttt{k},\ (\texttt{lookup}^{ro}_{\texttt{k}} \circ f^{rw}) \sim (\texttt{lookup}^{ro}_{\texttt{k}} \circ g^{rw})}{f == g}$$

after f

x	y	z	t	u	v
1	2	3	4	5	6

after g

x	y	z	t	u	v
1	2	3	4	5	6

The Rule of Observation

Two `weakly` equal functions are `strongly equal` if the state looks the same after their evaluations.

$$\frac{\text{(observation)}\quad f^{rw}, g^{rw} : X \to \texttt{void} \quad \forall\ \texttt{k},\ (\texttt{lookup}^{ro}_{\texttt{k}} \circ f^{rw}) \sim (\texttt{lookup}^{ro}_{\texttt{k}} \circ g^{rw})}{f == g}$$

after f

x	y	z	t	u	v
1	2	3	4	5	6

after g

x	y	z	t	u	v
1	2	3	4	5	6

The Rule of Observation

Two `weakly` equal functions are `strongly equal` if the state looks the same after their evaluations.

$$\frac{\text{(observation)}\quad f^{rw}, g^{rw} : X \to \texttt{void} \quad \forall\ \texttt{k},\ (\texttt{lookup}^{ro}_{\texttt{k}} \circ f^{rw}) \sim (\texttt{lookup}^{ro}_{\texttt{k}} \circ g^{rw})}{f == g}$$

after f

x	y	z	t	u	v
1	2	3	4	5	6

after g

x	y	z	t	u	v
1	2	3	4	5	6

The Rule of Observation

Two `weakly` equal functions are `strongly` equal if the state looks the same after their evaluations.

$$\frac{\text{(observation)}\quad f^{rw}, g^{rw} : X \to \texttt{void} \quad \forall\ \texttt{k},\ (\texttt{lookup}^{ro}_{\texttt{k}} \circ f^{rw}) \sim (\texttt{lookup}^{ro}_{\texttt{k}} \circ g^{rw})}{f == g}$$

after f

x	y	z	t	u	v
1	2	3	4	5	6

+ returns void

after g

x	y	z	t	u	v
1	2	3	4	5	6

+ returns void

The Rule of Observation

Two `weakly` equal functions are `strongly` equal if the state looks the same after their evaluations.

$$\frac{\text{(observation)}\quad f^{rw}, g^{rw} : X \to \texttt{void} \quad \forall\ \texttt{k},\ (\texttt{lookup}^{ro}_{\texttt{k}} \circ f^{rw}) \sim (\texttt{lookup}^{ro}_{\texttt{k}} \circ g^{rw})}{f == g}$$

after f

x	y	z	t	u	v
1	2	3	4	5	6

+ returns void

after g

x	y	z	t	u	v
1	2	3	4	5	6

+ returns void

done: $f == g$

A Naïve Introduction to Coq

What is Coq?

- An interactive theorem prover
- A programming language
  - not Turing complete
- everything (a program or a type)
  - has a name
  - has a type

A Naïve Introduction to Coq

- A *program* has a type
  - `0`: `nat`
  - `Definition f (x: nat) := x^2.` `f`: `nat` → `nat`
- A *type* has a `Type`
  - `nat`: `Type`
  - `nat` → `nat`: `Type`
  - `Prop`: `Type`
  - `Type`: $\texttt{Type}_1$
  - $\texttt{Type}_i$: $\texttt{Type}_{i+1}$
- A *proof* is a *program* of type `Prop`
  (`Curry-Howard Correspondence`)

A Naïve Introduction to Coq

The strength:

- Dependent types: the `type of output instance` depends on the `type of input instance`
- `Inductive` **term**: `Type` → `Type` → `Type` :=
  | `id`: ∀ {X: `Type`}, **term** X X
  | `comp`: ∀ {X Y Z: `Type`}, **term** X Y → **term** Y Z → **term** X Z

N.B. **term** defines a function type depending on its input types.

A Simple Proof by Coq

Coq < Parameters A B : Prop.

Coq < Lemma neg_or_and : $\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$

1 subgoal

==========(1/1)

$\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$

Coq < unfold not.

A Simple Proof by Coq

Coq < Parameters A B : Prop.

Coq < Lemma neg_or_and : $\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$

1 subgoal

==========(1/1)

$\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$

Coq < unfold not.

==========(1/1)

$A \vee B \to False \leftrightarrow (A \to False) \wedge (B \to False)$

A Simple Proof by Coq

Coq < Parameters A B : Prop.

Coq < Lemma neg_or_and : $\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$

1 subgoal

==========(1/1)

$A \vee B \rightarrow False \leftrightarrow (A \rightarrow False) \wedge (B \rightarrow False)$

Coq < split.

A Simple Proof by Coq

Coq < Lemma neg_or_and : $\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$

1 subgoal

==========(1/1)

$A \vee B \to False \leftrightarrow (A \to False) \wedge (B \to False)$

Coq < split.

2 subgoals

==========(1/2)

$(A \vee B \to False) \to (A \to False) \wedge (B \to False)$

==========(2/2)

$(A \to False) \wedge (B \to False) \to A \vee B \to False$

A Simple Proof by Coq

2 subgoals

$$==========(1/2)$$
$$(A \vee B \to False) \to (A \to False) \wedge (B \to False)$$

$$==========(2/2)$$
$$(A \to False) \wedge (B \to False) \to A \vee B \to False$$

Coq < intro H.

A Simple Proof by Coq

2 subgoals

H: $A \vee B \to False$

==========(1/2)

$(A \to False) \wedge (B \to False)$

==========(2/2)

$(A \to False) \wedge (B \to False) \to A \vee B \to False$

Coq < split.

A Simple Proof by Coq

3 subgoals

H: $A \vee B \to False$

==========(1/3)

$(A \to False)$

==========(2/3)

$(B \to False)$

==========(3/3)

$(A \to False) \wedge (B \to False) \to A \vee B \to False$

Coq < intro H0.

A Simple Proof by Coq

3 subgoals

H: $A \vee B \rightarrow False$
H0: A
==========(1/3)
$False$

==========(2/3)
$(B \rightarrow False)$

==========(3/3)
$(A \rightarrow False) \wedge (B \rightarrow False) \rightarrow A \vee B \rightarrow False$

Coq < apply H.

A Simple Proof by Coq

3 subgoals

H: $A \vee B \to False$
H0: A
==========(1/3)
$A \vee B$

==========(2/3)
$(B \to False)$

==========(3/3)
$(A \to False) \wedge (B \to False) \to A \vee B \to False$

Coq < left.

A Simple Proof by Coq

3 subgoals

H: $A \vee B \rightarrow False$
H0: A
==========(1/3)
A

==========(2/3)
$(B \rightarrow False)$

==========(3/3)
$(A \rightarrow False) \wedge (B \rightarrow False) \rightarrow A \vee B \rightarrow False$

Coq < apply H0.

A Simple Proof by Coq

2 subgoals

H: $A \vee B \to False$

==========(1/2)

$(B \to False)$

==========(2/2)

$(A \to False) \wedge (B \to False) \to A \vee B \to False$

Coq < intro H0.

A Simple Proof by Coq

2 subgoals

H: $A \vee B \rightarrow False$
H0: B

==========(1/2)
$False$

==========(2/2)
$(A \rightarrow False) \wedge (B \rightarrow False) \rightarrow A \vee B \rightarrow False$

Coq < apply H.

A Simple Proof by Coq

2 subgoals

H: $A \vee B \to False$
H0: B

==========(1/2)
$A \vee B$

==========(2/2)
$(A \to False) \wedge (B \to False) \to A \vee B \to False$

Coq < right.

A Simple Proof by Coq

2 subgoals

H: $A \vee B \rightarrow False$
H0: B

==========(1/2)
B

==========(2/2)
$(A \rightarrow False) \wedge (B \rightarrow False) \rightarrow A \vee B \rightarrow False$

Coq < apply H0.

A Simple Proof by Coq

1 subgoal

$$==========(1/1)$$
$$(A \to False) \wedge (B \to False) \to A \vee B \to False$$

Coq < intro H.

A Simple Proof by Coq

1 subgoal

H: $(A \to False) \wedge (B \to False)$

==========(1/1)

$A \vee B \to False$

Coq < destruct H as (H0&H1).

A Simple Proof by Coq

1 subgoal

H0: $(A \to False)$
H1: $(B \to False)$

==========(1/1)
$A \vee B \to False$

Coq < intro H.

A Simple Proof by Coq

1 subgoal

H0: $(A \to False)$
H1: $(B \to False)$
H: $A \vee B$

==========(1/1)
$False$

Coq < destruct H as [H2|H3].

A Simple Proof by Coq

2 subgoals

H0: $(A \to False)$
H1: $(B \to False)$
H2: A

==========(1/2)
$False$

==========(2/2)
$False$

Coq < apply H0.

A Simple Proof by Coq

2 subgoals

H0: $(A \to False)$
H1: $(B \to False)$
H2: A

==========(1/2)
A

==========(2/2)
$False$

Coq < apply H2.

A Simple Proof by Coq

2 subgoals

H0: $(A \rightarrow False)$
H1: $(B \rightarrow False)$
H3: B

==========(1/1)
$False$

Coq < apply H1.

A Simple Proof by Coq

2 subgoals

H0: $(A \to False)$
H1: $(B \to False)$
H3: B

==========(1/1)
B

Coq < apply H3.

A Simple Proof by Coq

Coq < Qed.

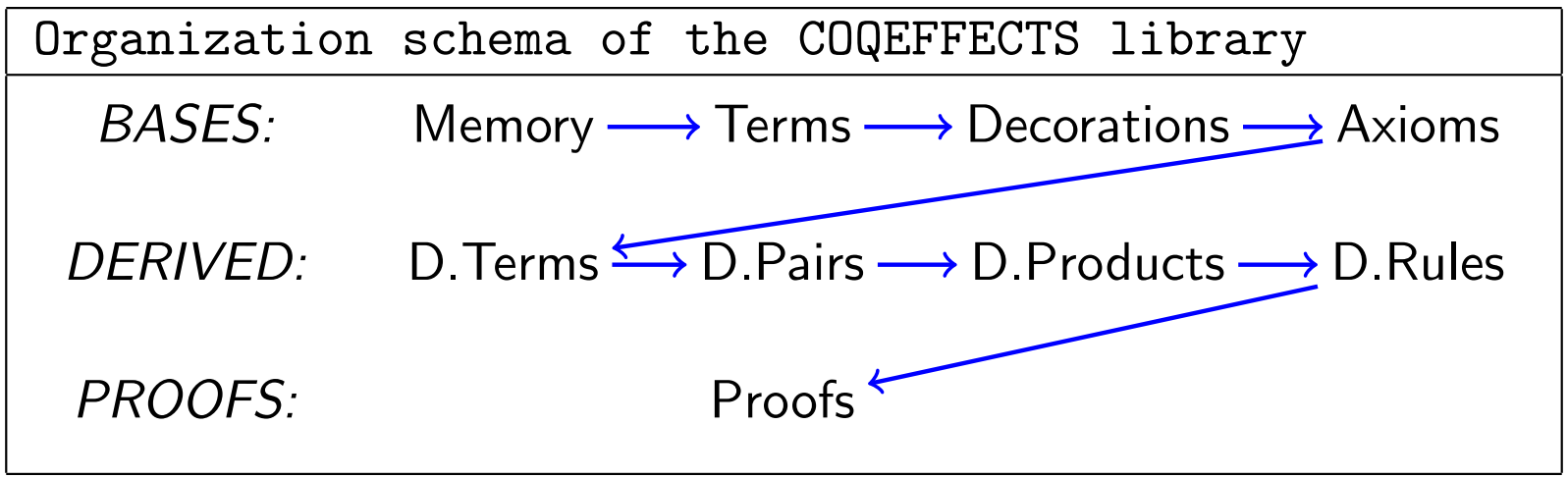
Lemma neg_or_and : $\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$

unfold not. split. intro H. split.
intro H0. apply H. left. apply H0.
intro H0. apply H. right. apply H0.
intro H. destruct H as (H0&H1). intro H.
destruct H as [H2|H3].
apply H0. apply H2.
apply H1. apply H3.

neg_or_and is defined

Coq <

Organization schema of the COQEFFECTS library



Memory & Terms

```
Parameter Loc: Type.

Parameter Val: Loc → Type.

Inductive term: Type → Type → Type :=
 | id: ∀ {X: Type}, term X X
 | comp: ∀ {X Y Z: Type}, term X Y → term Y Z → term X Z
 | forget: ∀ {X: Type}, term unit X
 | pair: ∀ {X Y Z: Type}, term X Z → term Y Z → term (X×Y) Z
 | pi1: ∀ {X Y: Type}, term X (X×Y)
 | pi2: ∀ {X Y: Type}, term Y (X×Y)
 | three {x: Loc}: term (Val x) unit
 | four {y: Loc}: term (Val y) unit
 | lookup: ∀ i: Loc, term (Val i) unit
 | update: ∀ i: Loc, term unit (Val i).

Infix "o" := comp (at level 70).
```

Decorations

```
Inductive kind := pure | ro | rw.

Inductive is: kind → ∀ X Y, term X Y → Prop :=
 | is_id: ∀ X, is pure (@id X)
 | is_comp: ∀ k X Y Z (f: term X Y) (g: term Y Z), is k f → is k g → is k
   (f ∘ g)
 | is_forget: ∀ X, is pure (@forget X)
 | is_pair: ∀ k X Y Z (f: term X Z) (g: term Y Z), is k f → is k g → is k
   (pair f g)
 | is_pi1: ∀ X Y, is pure (@pi1 X Y)
 | is_pi2: ∀ X Y, is pure (@pi2 X Y)
 | is_three {x: Loc}: is pure (@three x)
 | is_four {y: Loc}: is pure (@four y)
 | is_lookup: ∀ i, is ro (lookup i)
 | is_update: ∀ i, is rw (update i)
 | is_pure_ro: ∀ X Y (f: term X Y), is pure f → is ro f
 | is_ro_rw: ∀ X Y (f: term X Y), is ro f → is rw f
```

Axioms

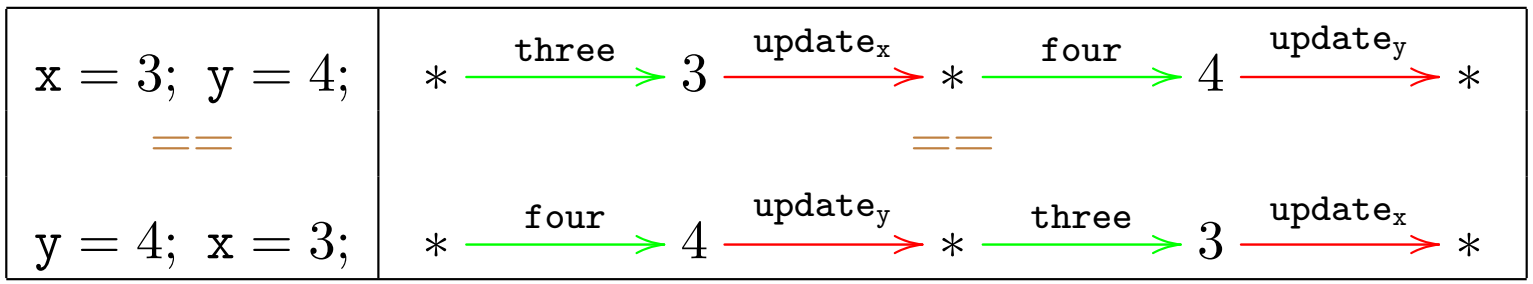
```
Reserved Notation " x == y" (at level 80).
Reserved Notation "x ∼ y" (at level 80).

Inductive strong: ∀ X Y, relation (term X Y) :=
 | strong_refl: ∀ X Y (f: term X Y), f == f
 | id_src: ∀ X Y (f: term X Y), f o id == f
 | id_tgt: ∀ X Y (f: term X Y), id o f == f
 | strong_subs: ∀ X Y Z (g1 g2: term X Y) (f: term Y Z), g1
   == g2 → g1 o f == g2 o f
 | strong_repl: ∀ X Y Z (g1 g2: term X Y) (f: term Z X), g1
   == g2 → f o g1 == f o g2
 | ro_weak_to_strong: ∀ X Y (f g: term X Y),
   is ro f → is ro g → f ∼ g → f == g
 | strong_sym: ∀ X Y, Symmetric (@strong X Y)
 | strong_trans: ∀ X Y, Transitive (@strong X Y)
```

Axioms

```
with weak: ∀ X Y, relation (term X Y) :=
 | weak_subs: ∀ X Y Z (g1 g2: term X Y) (f: term Y Z),
   g1 ∼ g2 → g1 o f ∼ g2 o f
 | pure_weak_repl: ∀ X Y Z (g: term X Y) (f1 f2: term Y Z),
   is pure g → f1 ∼ f2 → g o f1 ∼ g o f2
 | axiom_1: ∀ i, lookup i o update i ∼ id
 | axiom_2: ∀ i j, i≠j → lookup j o update i
   ∼ lookup j o forget
 | strong_to_weak: ∀ X Y (f g: term X Y), f == g → f ∼ g
 | weak_sym: ∀ X Y, Symmetric (@weak X Y)
 | weak_trans: ∀ X Y, Transitive (@weak X Y)
 | weak_final_unique: ∀ X (f g: term unit X), f ∼ g
 | observation: ∀ X (f g: term unit X),
    (∀ i, lookup i o f ∼ lookup i o g) → f == g
```

A Simple Example: Commutation update-update

Commutation update-update:



in Coq:

```
(update y) ∘ four ∘ (update x) ∘ three
==
(update x) ∘ three ∘ (update y) ∘ four.
```

Coq Implementation: Commutation update-update

1 subgoal

====================================(1/1)

forall x y : Loc, $x \neq y \rightarrow$ ((update y $\circ$ four) $\circ$ update x) $\circ$ three $==$ ((update x $\circ$ three) $\circ$ update y) $\circ$ four.

Coq < intros.

Coq Implementation: Commutation update-update

1 subgoal

x : Loc
y : Loc
H : x ≠ y

==========(1/1)
((update y ∘ four) ∘ update x) ∘ three == ((update x ∘ three) ∘ update y) ∘ four.

Coq < apply observation.

Coq Implementation: Commutation update-update

1 subgoal

x : Loc
y : Loc
H : x ≠ y

==========(1/1)
forall i : Loc, lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼
lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < intros.

Coq Implementation: Commutation update-update

1 subgoal

x : Loc
y : Loc
H : x ≠ y
i : Loc

==========(1/1)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < destruct (Loc_dec i y).

Coq Implementation: Commutation update-update

2 subgoals

x : Loc
y : Loc
H : x ≠ y
i : Loc
e : i = y

==========(1/2)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(2/2)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < rewrite e.

Coq Implementation: Commutation update-update

2 subgoals

x : Loc
y : Loc
H : x ≠ y
i : Loc
e : i = y

==========(1/2)
lookup y ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(2/2)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < transitivity(id ∘ four ∘ update x ∘ three).

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$

==========(1/3)
lookup y ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ id ∘ four ∘ update x ∘ three

==========(2/3)
id ∘ four ∘ update x ∘ three ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < rewrite assoc.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
(lookup y ∘ ((update y ∘ four) ∘ update x)) ∘ three ∼ id ∘ four ∘ update x ∘ three

==========(2/3)
id ∘ four ∘ update x ∘ three ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < apply weak_subs.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
lookup y ∘ ((update y ∘ four) ∘ update x) ∼ id ∘ four ∘ update x

==========(2/3)
id ∘ four ∘ update x ∘ three ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < rewrite assoc.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
(lookup y ∘ (update y ∘ four)) ∘ update x ∼ id ∘ four ∘ update x

==========(2/3)
id ∘ four ∘ update x ∘ three ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < apply weak_subs.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
lookup y ∘ (update y ∘ four) ∼ id ∘ four

==========(2/3)
id ∘ four ∘ update x ∘ three ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < rewrite assoc.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
(lookup y ∘ update y) ∘ four ∼ id ∘ four

==========(2/3)
id ∘ four ∘ update x ∘ three ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < apply weak_subs.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
(lookup y ∘ update y) ∼ id

==========(2/3)
id ∘ four ∘ update x ∘ three ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < apply axiom_1.

Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/2)
id ∘ four ∘ update x ∘ three ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(2/2)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < transitivity(id ∘ four ∘ id).

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$

==========(1/3)
id ∘ four ∘ update x ∘ three ∼ id ∘ four ∘ id

==========(2/3)
id ∘ four ∘ id ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < apply zero_weak_repl.

Coq Implementation: Commutation update-update

4 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/4)
is pure id
==========(2/4)
four ∘ update x ∘ three ∼ four ∘ id
==========(3/4)
id ∘ four ∘ id ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)
==========(4/4)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < apply is_id.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
four ∘ update x ∘ three ∼ four ∘ id

==========(2/3)
id ∘ four ∘ id ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < apply zero_weak_repl.

Coq Implementation: Commutation update-update

4 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/4)
is pure four
==========(2/4)
update x ∘ three ∼ id
==========(3/4)
id ∘ four ∘ id ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)
==========(4/4)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < apply is_four.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
update x ∘ three ∼ id

==========(2/3)
id ∘ four ∘ id ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < apply weak_final_unique.

Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/2)
id ∘ four ∘ id ∼ lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four)

==========(2/2)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < apply weak_sym.

Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/2)
lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four) ∼ id ∘ four ∘ id

==========(2/2)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < transitivity((lookup y) ∘ zero_final ∘ three ∘ (update y) ∘ four).

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
lookup y ∘ (((update x ∘ three) ∘ update y) ∘ four) ∼ (lookup y) ∘ zero_final ∘ three ∘ (update y) ∘ four

==========(2/3)
(lookup y) ∘ zero_final ∘ three ∘ (update y) ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < rewrite assoc; apply weak_subs.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$

==========(1/3)
lookup y ∘ ((update x ∘ three) ∘ update y) ∼ (lookup y) ∘ zero_final ∘ three ∘ (update y)

==========(2/3)
(lookup y) ∘ zero_final ∘ three ∘ (update y) ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < rewrite assoc; apply weak_subs.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$

==========(1/3)
lookup y ∘ (update x ∘ three) ∼ (lookup y) ∘ zero_final ∘ three

==========(2/3)
(lookup y) ∘ zero_final ∘ three ∘ (update y) ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < rewrite assoc; apply weak_subs.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
lookup y ∘ update x ∼ (lookup y) ∘ zero_final

==========(2/3)
(lookup y) ∘ zero_final ∘ three ∘ (update y) ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < apply axiom_2

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
x ≠ y

==========(2/3)
(lookup y) ∘ zero_final ∘ three ∘ (update y) ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < assumption.

Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/2)
(lookup y) ∘ zero_final ∘ three ∘ (update y) ∘ four ∼ id ∘ four ∘ id

==========(2/2)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < transitivity((lookup y) ∘ (update y) ∘ four).

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
(lookup y) ∘ zero_final ∘ three ∘ (update y) ∘ four ∼ (lookup y) ∘(update y) ∘ four
==========(2/3)

lookup y ∘update y ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < apply weak_subs; apply weak_subs.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
(lookup y) ∘ zero_final ∘ three ∼ (lookup y)
==========(2/3)

lookup y ∘update y ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < apply strong_to_weak.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
(lookup y) ∘ zero_final ∘ three == (lookup y)
==========(2/3)

lookup y ∘ update y ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < rewrite id_src at 6.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
(lookup y) ∘ zero_final ∘ three == (lookup y) ∘ id
==========(2/3)

lookup y ∘ update y ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < apply strong_repl.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
s : forall (X Y : Type) (g : term () Y) (h : term () X) (f : term Y X),
is pure g → is pure h → is pure f → h == g ∘ f
Heqs : s = E_0_3

==========(1/3)
zero_final ∘ three == id
==========(2/3)
lookup y ∘ update y ∘ four ∼ id ∘ four ∘ id
==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < apply E_0_3.

Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/2)
lookup y ∘ update y ∘ four ∼ id ∘ four ∘ id

==========(2/2)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < transitivity(id ∘ four).

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
lookup y ∘ update y ∘ four ∼ id ∘ four
==========(2/3)

id ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < apply weak_subs.

Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/3)
lookup y ∘ update y ∼ id
==========(2/3)

id ∘ four ∼ id ∘ four ∘ id

==========(3/3)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < apply axiom_1.

Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/2)
id ∘ four ∼ id ∘ four ∘ id
==========(2/2)

lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < apply weak_sym.

Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/2)
id ∘ four ∘ id ∼ id ∘ four
==========(2/2)

lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < apply pure_weak_repl.

Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/2)
four ∘ id ∼ four
==========(2/2)

lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < rewrite ← id_src at 3.

Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y

==========(1/2)
four ∘ id ∼ four ∘ id
==========(2/2)

lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘ (((update x ∘ three) ∘ update y) ∘ four)

Coq < apply weak_refl.

Coq Implementation: Commutation update-update

1 subgoal
x, y, i : Loc
H : x ≠ y
e : i = x

==========(1/1)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < ...

Coq Implementation: Commutation update-update

1 subgoal
x, y, i : Loc
H : $x \neq y$
e : $i \neq x \wedge i \neq y$

==========(1/1)
lookup i ∘ (((update y ∘ four) ∘ update x) ∘ three) ∼ lookup i ∘
(((update x ∘ three) ∘ update y) ∘ four)

Coq < ...

So Far:

1. A Coq library for the global states effect:
   - $\approx$ 1700 LoC
   - available: `http://coqeffects.forge.imag.fr/`
   - used to prove the *Hilbert-Post Completeness of the global state structure*
2. A paper for the conference JFLA14.

What's Next?

Future work :

- developing the framework for local state (allocation)
- developing the library for exceptions
- developing the concepts/Coq for combining effects (monad transformers [Haskell])
- generalization to the other effects

References


J.-G.Dumas, D. Duval, B. Ekici, D. Pous, *Formal verification in Coq of program properties involving the global state effect*. JFLA, 2014.


J.-G.Dumas, D. Duval, J.-C. Reynaud, *Patterns for computational effects arising from a monad or a comonad*. Rapport de Recherche, 2013.


J.-G.Dumas, D. Duval, L. Fousse, J.-C. Reynaud, *Decorated proofs for computational effects: States*. ACCAT 2012. EPTCS 93 p.45-59, 2012.


J.-G.Dumas, D. Duval, J.-C. Reynaud, *Cartesian effect categories are Freyd-categories*. JSC 46 p. 272-293, 2011.

The End!

Many thanks for your attention!

Questions?

The End!

Many thanks for your attention!

Questions?

Some Coq Tactics

Tactics are generalized rules in Coq environment!

$$(\texttt{intro H})\frac{\Gamma, H : A \vdash ? : B}{\Gamma \vdash ? : A \to B} \quad (\texttt{apply H})\frac{\Gamma \vdash H : A \to B \quad \Gamma \vdash ? : A}{\Gamma \vdash ? : B}$$

$$(\texttt{split})\frac{\Gamma \vdash ? : A \quad \Gamma \vdash ? : B}{\Gamma \vdash ? : A \wedge B}$$

$$(\texttt{left})\frac{\Gamma \vdash ? : A}{\Gamma \vdash ? : A \vee B} \quad (\texttt{right})\frac{\Gamma \vdash ? : B}{\Gamma \vdash ? : A \vee B}$$

$$(\texttt{destruct H})\frac{\Gamma \vdash H : A \wedge B \quad \Gamma, H0 : A, H1 : B \vdash ? : C}{\Gamma \vdash ? : C}$$

$$(\texttt{destruct H})\frac{\Gamma \vdash H : A \vee B \quad \Gamma, H0 : A \vdash ? : C \quad \Gamma, H1 : B \vdash ? : C}{\Gamma \vdash ? : C}$$