# 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

*JFLA 2014, Fréjus, France*

January 9, 2014

# Outline

# 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
- 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
- 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

$\texttt{update}_\texttt{x}$:

`x = 3;`

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

$\texttt{lookup}_\texttt{x}$:

`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:

$$f : \quad \mathtt{void} \xrightarrow{\mathtt{three}} \mathtt{int} \xrightarrow{\mathtt{update}_x} \mathtt{void} \xrightarrow{\mathtt{four}} \mathtt{int} \xrightarrow{\mathtt{update}_y} \mathtt{void}$$

$$* \longmapsto 3 \longmapsto * \longmapsto 4 \longmapsto *$$

In an interpretation:

$$f : \quad S \xrightarrow{\mathtt{three}} \mathtt{int} \times S \xrightarrow{\mathtt{update}_x} S \xrightarrow{\mathtt{four}} \mathtt{int} \times S \xrightarrow{\mathtt{update}_y} S$$

$$s \longmapsto (3, s) \longmapsto s_1 \longmapsto (4, s_1) \longmapsto s_2$$

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:

$$f : \quad \mathtt{void} \xrightarrow{\mathtt{three}} \mathtt{int} \xrightarrow{\mathtt{update}_x} \mathtt{void} \xrightarrow{\mathtt{four}} \mathtt{int} \xrightarrow{\mathtt{update}_y} \mathtt{void}$$

$$* \longmapsto 3 \longmapsto * \longmapsto 4 \longmapsto *$$

In an interpretation:

$$f : \quad S \xrightarrow{\mathtt{three}} \mathtt{int} \times S \xrightarrow{\mathtt{update}_x} S \xrightarrow{\mathtt{four}} \mathtt{int} \times S \xrightarrow{\mathtt{update}_y} S$$

$$s \longmapsto (3, s) \longmapsto s_1 \longmapsto (4, s_1) \longmapsto s_2$$

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:

$$f: \quad \mathtt{void} \xrightarrow{\mathtt{three}} \mathtt{int} \xrightarrow{\mathtt{update}_x} \mathtt{void} \xrightarrow{\mathtt{four}} \mathtt{int} \xrightarrow{\mathtt{update}_y} \mathtt{void}$$

$$* \longmapsto 3 \longmapsto * \longmapsto 4 \longmapsto *$$

In an interpretation:

$$f: \quad S \xrightarrow{\mathtt{three}} \mathtt{int} \times S \xrightarrow{\mathtt{update}_x} S \xrightarrow{\mathtt{four}} \mathtt{int} \times S \xrightarrow{\mathtt{update}_y} S$$

$$s \longmapsto (3, s) \longmapsto s_1 \longmapsto (4, s_1) \longmapsto s_2$$

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

# How to prove program equivalences

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

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

In syntax:

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

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 type of 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 type of 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 type of 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:

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

In an interpretation:

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

How to prove the equivalence of f and g without mentioning the type of 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 x be a location and $\mathtt{Val}_\mathtt{x}$ be the set of values that can be stored in x.

E.g., $\mathtt{Val}_\mathtt{x} = \mathtt{int}$

Functions are classified and decorated:

- pure: e.g., $\mathtt{id}_\mathtt{x}^{pure} : \mathtt{Val}_\mathtt{x} \to \mathtt{Val}_\mathtt{x}$, $\mathtt{forget}_\mathtt{x}^{pure} : \mathtt{Val}_\mathtt{x} \to \mathtt{void}$
- accessors: e.g., $\mathtt{lookup}_\mathtt{x}^{ro} : \mathtt{void} \to \mathtt{Val}_\mathtt{x}$
- modifiers: e.g., $\mathtt{update}_\mathtt{x}^{rw} : \mathtt{Val}_\mathtt{x} \to \mathtt{void}$
- 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.

# Decorations for States Effect

Let $\mathtt{x}$ be a location and $\mathtt{Val_x}$ be the set of values that can be stored in $\mathtt{x}$.

E.g., $\mathtt{Val_x} = \mathtt{int}$

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.

# 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 $\mathtt{f}^{ro} == \mathtt{g}^{ro} \Longleftrightarrow \mathtt{f}^{ro} \sim \mathtt{g}^{ro}$

# Basic Operations: Update & Lookup

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

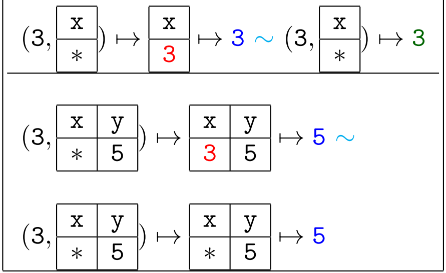
$\mathtt{lookup}_x : \mathtt{void} \to \mathtt{int}$
$\mathtt{update}_x : \mathtt{int} \to \mathtt{void}$

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

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

$\mapsto$

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



# 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 $f_1^{rw} := \mathtt{update}_{\mathtt{u}}(9)$

| 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 $f_1^{rw} := \mathtt{update}_\mathtt{u}(9)$

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

returns void

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

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

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

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

returns 9

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

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

returns 5

# Weak Equality: Compatibility w.r.t. composition

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

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

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

# The Rule of Observation

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

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

# The Rule of Observation

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

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

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.

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

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.

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

`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 $\mathtt{weakly}$ equal functions are $\mathtt{strongly}$ $\mathtt{equal}$ if the state looks the same after their evaluations.

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

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.

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

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.

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

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.

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

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.

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

after $f$

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

+ returns void

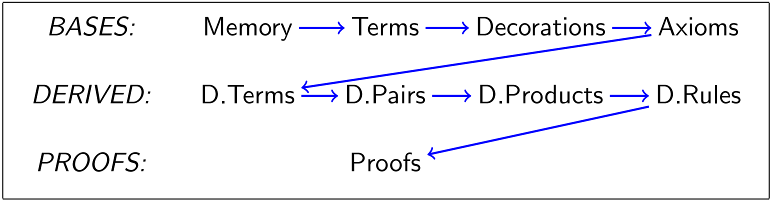
after $g$

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

+ returns void

done: $f == g$

# Organization schema of the COQEFFECTS library



# Locations & 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)
 | constant: ∀ i: Loc, (Val i) → term (Val i) 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 o 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_constant: ∀ i: Loc, ∀ c: (Val i), is pure (@constant i c)
 | 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_forget_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

In the paper; we have explained the proof of `commutation update-lookup`.

Requirements:

1. categorical pairs
2. categorical products
3. semi-pure products
4. sequential products

Due to lack of time we present the proof of `commutation update-update`.

# A Simple Example: Commutation update-update

`Commutation update-update`:

| | |
|---|---|
| $x = 3;\ y = 4;$ | $* \xrightarrow{\texttt{three}} 3 \xrightarrow{\texttt{update}_x} * \xrightarrow{\texttt{four}} 4 \xrightarrow{\texttt{update}_y} *$ |
| == | == |
| $y = 4;\ x = 3;$ | $* \xrightarrow{\texttt{four}} 4 \xrightarrow{\texttt{update}_y} * \xrightarrow{\texttt{three}} 3 \xrightarrow{\texttt{update}_x} *$ |

in Coq:

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

# Coq Implementation: Commutation update-update

1 subgoal

==================================(1/1)
forall {x y} : Loc (three: (Val x)) (four: (Val y)), x $\neq$ y $\rightarrow$
((update y $\circ$ (constant four)) $\circ$ update x) $\circ$ (constant three) ==
((update x $\circ$ (constant three)) $\circ$ update y) $\circ$ (constant four).

Coq < intros.

# Coq Implementation: Commutation update-update

1 subgoal

x : Loc
y : Loc
H : $x \neq y$
three : (Val x)
four : (Val y)

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

Coq < apply observation.

# Coq Implementation: Commutation update-update

1 subgoal

x : Loc
y : Loc
H : $x \neq y$
three : (Val x)
four : (Val y)

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

Coq < intros.

# Coq Implementation: Commutation update-update

1 subgoal

x : Loc
y : Loc
H : $x \neq y$
i : Loc
three : (Val x)
four : (Val y)

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

Coq < destruct (Loc_dec i y).

# Coq Implementation: Commutation update-update

2 subgoals

x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x)
four : (Val y)

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

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

Coq <   rewrite e.

# Coq Implementation: Commutation update-update

2 subgoals

x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)

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

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

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

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x $\neq$ y
e : i = y
three : (Val x), four : (Val y)
====================================(1/3)
lookup y ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
id ∘ (constant four) ∘ update x ∘ (constant three)
====================================(2/3)
id ∘ (constant four) ∘ update x ∘ (constant three) ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
====================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply weak_subs.

# Coq Implementation: Commutation update-update

```
3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x), four : (Val y)
====================================(1/3)
lookup y ∘ (update y ∘ (constant four) ∘ update x) ∼
id ∘ (constant four) ∘ update x
====================================(2/3)
id ∘ (constant four) ∘ update x ∘ (constant three) ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
====================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <   rewrite assoc.
```

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x), four : (Val y)
==================================(1/3)
lookup y ∘ (update y ∘ (constant four)) ∘ update x ∼
id ∘ (constant four) ∘ update x
==================================(2/3)
id ∘ (constant four) ∘ update x ∘ (constant three) ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
==================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply weak_subs.

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x), four : (Val y)
==================================(1/3)
lookup y ∘ (update y ∘ (constant four)) ∼ id ∘ (constant four)

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

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

Coq < rewrite assoc.

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
====================================(1/3)
lookup y ∘ (update y) ∘ (constant four) ∼ id ∘ (constant four)
====================================(2/3)
id ∘ (constant four) ∘ update x ∘ (constant three) ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
====================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply weak_subs.

# Coq Implementation: Commutation update-update

```
3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x)
four : (Val y)
==================================(1/3)
lookup y ∘ update y ∼ id
==================================(2/3)
id ∘ (constant four) ∘ update x ∘ (constant three) ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
==================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <   apply axiom_1.
```

# Coq Implementation: Commutation update-update

```
2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x)
four : (Val y)
====================================(1/2)
id ∘ (constant four) ∘ update x ∘ (constant three) ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
====================================(2/2)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <   transitivity(id ∘ (constant four) ∘ id).
```

# Coq Implementation: Commutation update-update

```
3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x), four : (Val y)
=================================(1/3)
id ∘ (constant four) ∘ update x ∘ (constant three) ∼
id ∘ (constant four) ∘ id
=================================(2/3)
id ∘ (constant four) ∘ id ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
=================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <  apply zero_weak_repl.
```

# Coq Implementation: Commutation update-update

4 subgoals
x, y, i : Loc
H : x ≠ y, e : i = y
three : (Val x), four : (Val y)
==================================(1/4)
is pure id
==================================(2/4)
(constant four) ∘ update x ∘ (constant three) ∼ (constant four) ∘ id
==================================(3/4)
id ∘ (constant four) ∘ id ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
==================================(4/4)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply is_id.

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
==================================(1/3)
(constant four) ∘ update x ∘ (constant three) ∼ (constant four) ∘ id
==================================(2/3)
id ∘ (constant four) ∘ id ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
==================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply zero_weak_repl.

# Coq Implementation: Commutation update-update

4 subgoals
x, y, i : Loc
H : x $\neq$ y, e : i = y
three : (Val x), four : (Val y)
==================================(1/4)
is pure (constant four)
==================================(2/4)
update x ∘ (constant three) ∼ id
==================================(3/4)
id ∘ (constant four) ∘ id ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
==================================(4/4)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply is_constant.

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
=================================(1/3)
update x ∘ (constant three) ∼ id
=================================(2/3)
id ∘ (constant four) ∘ id ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
=================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply weak_forget_unique.

# Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x $\neq$ y
e : i = y
three : (Val x)
four : (Val y)
====================================(1/2)
id ∘ (constant four) ∘ id ∼
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)
====================================(2/2)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply weak_sym.

# Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : x $\neq$ y
e : i = y
three : (Val x)
four : (Val y)
======================================(1/2)
lookup y $\circ$ (update x $\circ$ (constant three) $\circ$ update y) $\circ$ (constant four) $\sim$ id $\circ$ (constant four) $\circ$ id
======================================(2/2)
lookup i $\circ$ (update y $\circ$ (constant four) $\circ$ update x) $\circ$ (constant three) $\sim$
lookup i $\circ$ (update x $\circ$ (constant three) $\circ$ update y) $\circ$ (constant four)

Coq < transitivity(lookup y $\circ$ forget $\circ$ (constant three) $\circ$ (update y) $\circ$ (constant four)).

# Coq Implementation: Commutation update-update

```
3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x), four : (Val y)
=================================(1/3)
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four) ∼
lookup y ∘ forget ∘ (constant three) ∘ (update y) ∘ (constant four)
=================================(2/3)
(lookup y) ∘ forget ∘ (constant three) ∘ (update y) ∘ (constant four) ∼
id ∘ (constant four) ∘ id
=================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <  apply weak_subs.
```

# Coq Implementation: Commutation update-update

```
3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x), four : (Val y)
===================================(1/3)
lookup y ∘ (update x ∘ (constant three) ∘ update y) ∼
lookup y ∘ forget ∘ (constant three) ∘ (update y)
===================================(2/3)
(lookup y) ∘ forget ∘ (constant three) ∘ (update y) ∘ (constant four) ∼
id ∘ (constant four) ∘ id
===================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <   rewrite assoc; apply weak_subs.
```

# Coq Implementation: Commutation update-update

```
3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x), four : (Val y)
===================================(1/3)
lookup y ∘ update x ∘ (constant three) ∼
lookup y ∘ forget ∘ (constant three)
===================================(2/3)
(lookup y) ∘ forget ∘ (constant three) ∘ (update y) ∘ (constant four) ∼
id ∘ (constant four) ∘ id
===================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <    apply weak_subs.
```

# Coq Implementation: Commutation update-update

```
3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x)
four : (Val y)
==================================(1/3)
lookup y ∘ update x ∼ lookup y ∘ forget
==================================(2/3)
(lookup y) ∘ forget ∘ (constant three) ∘ (update y) ∘ (constant four) ∼
id ∘ (constant four) ∘ id
==================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <  apply axiom_2
```

# Coq Implementation: Commutation update-update

```
3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x)
four : (Val y)
===================================(1/3)
x ≠ y
===================================(2/3)
(lookup y) ∘ forget ∘ (constant three) ∘ (update y) ∘ (constant four) ∼
id ∘ (constant four) ∘ id
===================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <  apply assumption.
```

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x $\neq$ y
e : i = y
three : (Val x)
four : (Val y)
====================================(1/2)
(lookup y) ∘ forget ∘ (constant three) ∘ (update y) ∘ (constant four) ∼
id ∘ (constant four) ∘ id
====================================(2/2)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

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

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
==================================(1/3)
(lookup y) ∘ forget ∘ (constant three) ∘ (update y) ∘ (constant four) ∼
(lookup y) ∘(update y) ∘ (constant four)
==================================(2/3)
lookup y ∘ update y ∘ (constant four) ∼ id ∘ (constant four) ∘ id
==================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant 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
three : (Val x)
four : (Val y)
====================================(1/3)
(lookup y) ∘ forget ∘ (constant three) ∼ (lookup y)
====================================(2/3)
lookup y ∘ update y ∘ (constant four) ∼ id ∘ (constant four) ∘ id
====================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <   apply strong_to_weak.
```

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x)
four : (Val y)
======================================(1/3)
(lookup y) ∘ forget ∘ (constant three) == (lookup y)
======================================(2/3)
lookup y ∘ update y ∘ (constant four) ∼ id ∘ (constant four) ∘ id
======================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < rewrite id_src at 6.

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
==================================(1/3)
(lookup y) ∘ forget ∘ (constant three) == (lookup y) ∘ id
==================================(2/3)
lookup y ∘ update y ∘ (constant four) ∼ id ∘ (constant four) ∘ id
==================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply strong_repl.

# Coq Implementation: Commutation update-update

```
3 subgoals
x, y, i : Loc
H : x ≠ y, e : i = y
three : (Val x), four : (Val 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)
forget ∘ (constant three) == id
======================================(2/3)
lookup y ∘ update y ∘ (constant four) ∼ id ∘ (constant four) ∘ id
======================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <   apply E_0_3.
```

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
==================================(1/2)
lookup y ∘ update y ∘ (constant four) ∼ id ∘ (constant four) ∘ id
==================================(2/2)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < transitivity(id ∘ (constant four)).

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
==================================(1/3)
lookup y ∘ update y ∘ (constant four) ∼ id ∘ (constant four)
==================================(2/3)
id ∘ (constant four) ∼ id ∘ (constant four) ∘ id
==================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply weak_subs.

# Coq Implementation: Commutation update-update

3 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
===================================(1/3)
lookup y $\circ$ update y $\sim$ id
===================================(2/3)
id $\circ$ (constant four) $\sim$ id $\circ$ (constant four) $\circ$ id
===================================(3/3)
lookup i $\circ$ (update y $\circ$ (constant four) $\circ$ update x) $\circ$ (constant three) $\sim$
lookup i $\circ$ (update x $\circ$ (constant three) $\circ$ update y) $\circ$ (constant four)

Coq < apply axiom_1.

# Coq Implementation: Commutation update-update

```
2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x)
four : (Val y)
====================================(1/2)
id ∘ (constant four) ∼ id ∘ (constant four) ∘ id
====================================(2/2)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <   apply weak_sym.
```

# Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
==================================(1/2)
id ∘ (constant four) ∘ id ∼ id ∘ (constant four)
==================================(2/2)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply pure_weak_repl.

# Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
====================================(1/3)
is pure id
====================================(2/3)
id ∘ (constant four) ∘ id ∼ id ∘ (constant four)
====================================(3/3)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq < apply is_id.

# Coq Implementation: Commutation update-update

2 subgoals
x, y, i : Loc
H : $x \neq y$
e : $i = y$
three : (Val x)
four : (Val y)
==================================(1/2)
(constant four) $\circ$ id $\sim$ (constant four)
==================================(2/2)
lookup i $\circ$ (update y $\circ$ (constant four) $\circ$ update x) $\circ$ (constant three) $\sim$
lookup i $\circ$ (update x $\circ$ (constant three) $\circ$ update y) $\circ$ (constant four)

Coq < rewrite $\leftarrow$ id_src at 3.

# Coq Implementation: Commutation update-update

```
2 subgoals
x, y, i : Loc
H : x ≠ y
e : i = y
three : (Val x)
four : (Val y)
====================================(1/2)
(constant four) ∘ id ∼ (constant four) ∘ id
====================================(2/2)
lookup i ∘ (update y ∘ (constant four) ∘ update x) ∘ (constant three) ∼
lookup i ∘ (update x ∘ (constant three) ∘ update y) ∘ (constant four)

Coq <   apply weak_refl.
```

# Coq Implementation: Commutation update-update

1 subgoal
x, y, i : Loc
H : x ≠ y
e : i = x
three : (Val x)
four : (Val y)

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

Coq < ...

# Coq Implementation: Commutation update-update

1 subgoal
x, y, i : Loc
H : $x \neq y$
e : $i \neq x \wedge i \neq y$
three : (Val x)
four : (Val y)

==================================(1/1)
lookup i $\circ$ (update y $\circ$ (constant four) $\circ$ update x) $\circ$ (constant three) $\sim$
lookup i $\circ$ (update x $\circ$ (constant three) $\circ$ update y) $\circ$ (constant 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 Coq library for exceptions effect (not yet released).

# What's Next?

Future work :

- developing the framework for local state (allocation)
- developing the concepts/Coq for combining effects (monad transformers [Haskell])
- generalization to the other effects
- verification of a real-life `C` code with `effects`.

# What's Next?

Future work :

- developing the framework for local state (allocation)
- developing the concepts/Coq for combining effects (monad transformers [Haskell])
- generalization to the other effects
- verification of a real-life `C code with effects`.

# The End!

Un grand merci de votre attention !

Questions ?

# The End!

Un grand merci de votre attention !

Questions ?

# References


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.

# Some Coq Tactics

Tactics are generalized rules in Coq environment!

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

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

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

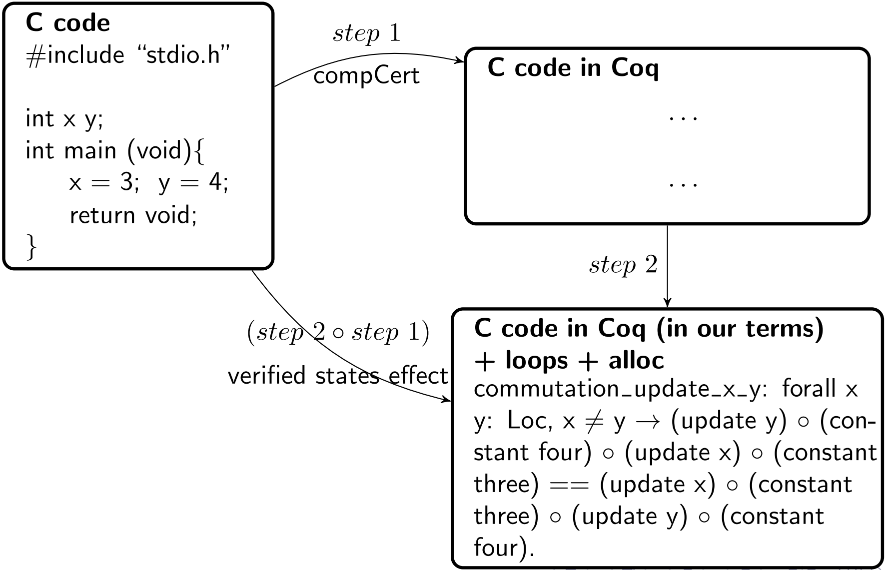
$$(\texttt{destruct H})\frac{\Gamma \vdash H : A \wedge B \quad \Gamma, H0 : A, H1 : B \vdash ? : C}{\Gamma \vdash ? : C}$$

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

# Properties of the State Structure by Plotkin et al.

1. `Annihilation lookup-update` $\forall i \in Loc,\ u_i \circ l_i == id_{\mathbb{1}} : \mathbb{1} \to \mathbb{1}$
2. `Interaction lookup-lookup`
   $\forall i \in Loc,\ l_i \circ \langle\,\rangle_i \circ l_i == l_i : \mathbb{1} \to V_i$
3. `Interaction update-update`
   $\forall i \in Loc,\ u_i \circ \pi_2 \circ (u_i \rtimes id_i) == u_i \circ \pi_2 : V_i \times V_i \to \mathbb{1}$
4. `Interaction update-lookup` $\forall i \in Loc,\ l_i \circ u_i \sim\ id_i : V_i \to V_i$
5. `Commutation lookup-lookup`
   $\forall i \neq j \in Loc,\ (id_i \ltimes l_j) \circ l_i == perm_{j,i} \circ (id_j \ltimes l_i) \circ l_j : \mathbb{1} \to V_i \times V_j$
6. `Commutation update-update`
   $\forall i \neq j \in Loc,\ u_j \circ \pi_2 \circ (u_i \rtimes id_j) == u_i \circ \pi_1 \circ (id_i \ltimes u_j) : V_i \times V_j \to \mathbb{1}$
7. `Commutation update-lookup`
   $\forall i \neq j \in Loc,\ l_j \circ u_i == \pi_2 \circ (u_i \rtimes id_j) \circ (id_i \ltimes l_j) \circ (\pi_1)^{-1} : V_i \to V_j$

# C code to verify w.r.t. states effect



# Categorical background for decorations

**Algorithm 1:** ACCESSORS explains the interpretations of decorated accessors.

**Input**: A $\texttt{category}$ $\mathbb{C}$ with a distinguished "object of states $\texttt{S}$", etc...
**Output**: the interpretations of accessors via $\texttt{states}$ comonad.

Prove $\Phi\colon \mathbb{C} \to \mathbb{C}$ as an endo-functor

$$\Phi(\texttt{X}) = \texttt{X} \times \texttt{S}$$

$$\Phi(\texttt{f}: \texttt{X} \to \texttt{Y}) = (\texttt{f} \times \texttt{id}_\texttt{S}): \texttt{X} \times \texttt{S} \to \texttt{Y} \times \texttt{S}$$

Prove $\texttt{cM}$ $(\Phi, \delta\colon \Phi \Rightarrow \Phi^2, \epsilon\colon \Phi \Rightarrow \texttt{id}_\mathbb{C})$ as the $\texttt{states}$ comonad.

Construct the $\texttt{coKleisli}$ category $\mathbb{C}_\mathbb{1}$ of $\texttt{cM}$ over $\mathbb{C}$

$$\texttt{Obj}(\mathbb{C}_\mathbb{1}) = \texttt{Obj}(\mathbb{C})$$

$$\texttt{Hom}_{(\mathbb{C}_\mathbb{1})}(\texttt{X}, \texttt{Y}) = \texttt{Hom}_{(\mathbb{C})}(\Phi\texttt{X}, \texttt{Y})$$

**return** *Any impure function* $\texttt{f}^{ro}: \texttt{X} \to \texttt{Y} \in \texttt{Hom}_{(\mathbb{C}_\mathbb{1})}$ *is interpreted as* $\texttt{f}_0: \texttt{X} \times \texttt{S} \to \texttt{Y} \in \texttt{Hom}_{(\mathbb{C})}$ *which represents an accessor.*

# Categorical background for decorations

**Algorithm 2:** MODIFIERS explains the interpretations of decorated modifiers.

**Input**: Categories $\mathbb{C}$ and $\mathbb{C}_{\mathbb{1}}$ as before.

**Output**: the interpretations of modifiers via `states` monad.

Prove $\Phi_1 \colon \mathbb{C}_{\mathbb{1}} \to \mathbb{C}_{\mathbb{1}}$ as an endo-functor

$$\Phi_1(\mathtt{X}) = \mathtt{X} \times \mathtt{S}$$

$$\Phi_1(\mathtt{f} \colon \mathtt{X} \to \mathtt{Y}) = (\mathtt{f} \times \mathtt{id}_\mathtt{S}) \colon \mathtt{X} \times \mathtt{S} \to \mathtt{Y} \times \mathtt{S}$$

Prove $\mathtt{M}\ (\Phi_1, \mu \colon \Phi_1^2 \Rightarrow \Phi_1, \eta \colon \mathtt{id}_{\mathbb{C}_{\mathbb{1}}} \Rightarrow \Phi_1)$ as the `states monad`.

Construct the `Kleisli` category $\mathbb{C}_{\mathbb{2}}$ of $\mathtt{M}$ over $\mathbb{C}_{\mathbb{1}}$

$$\mathtt{Obj}(\mathbb{C}_{\mathbb{2}}) = \mathtt{Obj}(\mathbb{C}_{\mathbb{1}}) = \mathtt{Obj}(\mathbb{C})$$

$$\mathtt{Hom}_{(\mathbb{C}_{\mathbb{2}})}(\mathtt{X}, \mathtt{Y}) = \mathtt{Hom}_{(\mathbb{C}_{\mathbb{1}})}(\mathtt{X}, \Phi_1\mathtt{Y}) = \mathtt{Hom}_{(\mathbb{C})}(\Phi\mathtt{X}, \Phi\mathtt{Y})$$

**return** *Any impure function* $\mathtt{f}^{rw} \colon \mathtt{X} \to \mathtt{Y} \in \mathtt{Hom}_{(\mathbb{C}_{\mathbb{2}})}$ *is interpreted as* $\mathtt{f}_0 \colon \mathtt{X} \times \mathtt{S} \to \mathtt{Y} \times \mathtt{S} \in \mathtt{Hom}_{(\mathbb{C})}$ *which represents a modifier.*