Decorated proofs for computational effects: States

Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse, Jean-Claude Reynaud

LJK, University of Grenoble

April 1., 2012 – ACCAT 2012 – Tallinn
Outline
From computer algebra to effects

About the history of the authors:

- **Computer algebra**: exact computations on large integers, matrices, polynomials, field extensions, . . .

- Sophisticated *programmation* in several kinds of languages: C, C++, Axiom, . . .

- Questions about the languages: semantics of computational *effects*? (e.g., states, exceptions, . . .)
Effects and monads

Breaking a taboo:

\[ \text{effect} \neq \text{monad} \]
Effects and monads

Breaking a taboo:

\[ \text{effect} \neq \text{monad} \]

[Moggi’91]: When there is an effect:

1. a term \( f : X \to Y \) should not always be interpreted as a function \([f] : \llbracket X \rrbracket \to \llbracket Y \rrbracket\)
2. it should often be interpreted as a function \([f] : \llbracket X \rrbracket \to T[\llbracket Y \rrbracket] \) for some monad \( T \)
Effects and monads

Breaking a taboo:

\[ \text{effect} \neq \text{monad} \]

[Moggi’91]: When there is an effect:

1. a term \( f : X \rightarrow Y \) should not always be interpreted as a function \( [[f]] : [[X]] \rightarrow [[Y]] \)

2. it should often be interpreted as a function \( [[f]] : [[X]] \rightarrow T[[Y]] \) for some monad \( T \)

[Plotkin & Power 2002]: The operations and equations associated with the effect are described by a Lawvere theory.
Effects and monads

Breaking a taboo:

\[\text{effect} \neq \text{monad}\]

[Moggi’91]: When there is an effect:

1. a term \( f : X \to Y \) should not always be interpreted as a function \([[f]] : [[X]] \to [[Y]]\)

2. it should often be interpreted as a function \([[f]] : [[X]] \to T[[Y]]\) for some monad \(T\)

[Plotkin & Power 2002]: The operations and equations associated with the effect are described by a Lawvere theory.

Example. In an imperative language

\[T[[Y]] = (S \times [[Y]])^S\]
Breaking a taboo:

\[ \text{effect} \neq \text{monad} \]

[Moggi’91]: When there is an effect:

1. a term \( f : X \to Y \) should not always be interpreted as a function \( [[f]] : [[X]] \to [[Y]] \)

2. it should often be interpreted as a function \( [[f]] : [[X]] \to T[[Y]] \) for some monad \( T \)

[Plotkin & Power 2002]: The operations and equations associated with the effect are described by a Lawvere theory.

Example. In an imperative language

\[ T[[Y]] = (S \times [[Y]])^S \]

We agree with (1), not always with (2).
And we get operations and equations in a different way.
What is an effect?

Informally:

*An effect is an apparent lack of soundness.*
What is an effect?

Informally:

*An effect is an apparent lack of soundness.*

A lack of soundness:

\[
\text{syntax} \quad [ \times ] \quad \text{semantics}
\]

which can be “repaired”: 

\[
\text{syntax} \quad \bullet \quad \text{semantics}
\]
What is an effect?

Informally:

*An effect is an apparent lack of soundness.*

A lack of soundness:

```
syntax ——— X ——— ——— semantics
```

which can be “repaired”:

```
syntax ——— ——— ——— ——— semantics
```

Formally: [Domínguez & Duval MSCS’10]
Outline
A property of imperative languages

The **annihilation lookup-update (ALU)** property:

\[ X := X \text{ does not modify the state} \]
A property of imperative languages

The annihilation lookup-update (ALU) property:

\[ X := X \text{ does not modify the state} \]

Proof.
Let \( n \) be the value of \( X \) in the current state.

- First "\( X \)" (on the right) is evaluated as \( n \).
- Then "\( X := \)" (on the left) puts the value of \( X \) to \( n \), without modifying the value of other locations.

Hence the state is not modified. \( \square \)
Towards a formalization: a specification for states

Locations (or identifiers, or variables) $X, Y, \ldots$. The unit (or void, or singleton) type $\mathbb{1}$, with $\langle \rangle_A : A \to \mathbb{1}$ for each $A$. 

For each $X$, a type $V_X$ for values, two operations:

- $\ell_X : 1 \to V_X$ (lookup)
- $u_X : V_X \to 1$ (update)

and equations:

$$\ell_X \circ u_X \equiv \text{id}$$

$$\ell_Y \circ u_X \equiv \ell_Y \circ \langle \rangle$$

formalizing the intended semantics:

- $\ell_X$ returns the value of $X$ in the current state
- $u_X(n)$ modifies the current state: the value of $X$ becomes $n$, and the value of $Y$ is not modified, for every $Y \neq X$.
Towards a formalization: a specification for states

**Locations** (or identifiers, or variables) $X, Y, \ldots$

The **unit** (or void, or singleton) type $\mathbb{1}$, with $\langle \rangle_A : A \to \mathbb{1}$ for each $A$.

For each $X$, a type $V_X$ for **values**, two operations:

$$
\ell_X : \mathbb{1} \to V_X \quad \text{(lookup)} \\
u_X : V_X \to \mathbb{1} \quad \text{(update)}
$$

and equations:

$$
\ell_X \circ u_X \equiv id
\quad \ell_Y \circ u_X \equiv \ell_Y \circ \langle \rangle \quad \text{when } Y \neq X
$$

formalizing the intended semantics:

- $\ell_X$ returns the value of $X$ in the current state
- $u_X(n)$ modifies the current state: the value of $X$ becomes $n$, and the value of $Y$ is not modified, for every $Y \neq X$
A property of imperative languages: proof # 1

Let $\Sigma$ be the specification made of $\ell_X : \mathbb{1} \to V_X$ and $u_X : V_X \to \mathbb{1}$ such that $\ell_X \circ u_X \equiv id$ and $\ell_Y \circ u_X \equiv \ell_Y \circ \langle \rangle$ when $Y \neq X$.

Then $\Sigma$ satisfies the **annihilation lookup-update (ALU) property:**

$$u_X \circ \ell_X \equiv id$$
A property of imperative languages: proof # 1

Let Σ be the specification made of \( \ell_X : 1 \rightarrow V_X \) and \( u_X : V_X \rightarrow 1 \) such that \( \ell_X \circ u_X \equiv id \) and \( \ell_Y \circ u_X \equiv \ell_Y \circ \langle \rangle \) when \( Y \neq X \).

Then Σ satisfies the annihilation lookup-update (ALU) property:

\[
\boxed{u_X \circ \ell_X \equiv id}
\]

Proof. By observation: prove that \( \ell_Y \circ u_X \circ \ell_X \equiv \ell_Y \) for each \( Y \).

When \( Y = X \):

\[
\begin{align*}
\text{(subst) } & \quad \ell_X \circ u_X \equiv id \\
\hline
\ell_X \circ u_X \circ \ell_X & \equiv \ell_X \quad \text{(unit)} \\
\ell_X \circ u_X \circ \ell_X & \equiv \ell_Y \quad \text{(trans)}
\end{align*}
\]

Hence the state is not modified. \( \square \)
A property of imperative languages: proof # 1

Let \( \Sigma \) be the specification made of \( \ell_X : 1 \to V_X \) and \( u_X : V_X \to 1 \) such that \( \ell_X \circ u_X \equiv id \) and \( \ell_Y \circ u_X \equiv \ell_Y \circ \langle \rangle \) when \( Y \neq X \). Then \( \Sigma \) satisfies the **annihilation lookup-update (ALU) property**:  

\[
\boxed{u_X \circ \ell_X \equiv id}
\]

**Proof.** By observation: prove that \( \ell_Y \circ u_X \circ \ell_X \equiv \ell_Y \) for each \( Y \).

When \( Y = X \):

\[
\ell_X \circ u_X \equiv id \quad \text{(subst)} \quad \frac{\ell_X \circ u_X \equiv id}{\ell_X \circ u_X \circ \ell_X \equiv \ell_X}
\]

When \( Y \neq X \):

\[
\ell_Y \circ u_X \equiv \ell_Y \circ \langle \rangle \quad \text{(trans)} \quad \frac{\ell_Y \circ u_X \circ \ell_X \equiv \ell_Y \circ \langle \rangle \circ \ell_X}{\ell_X \circ \ell_X \equiv \ell_Y}
\]

\[
(\text{unit}) \quad \frac{\langle \rangle \circ \ell_X \equiv id}{\ell_Y \circ \langle \rangle \circ \ell_X \equiv \ell_Y}
\]

\[
(\text{repl}) \quad \frac{\ell_Y \circ \langle \rangle \circ \ell_X \equiv \ell_Y}{\ell_Y \circ u_X \circ \ell_X \equiv \ell_Y}
\]

Hence the state is not modified. \( \square \)
A property of imperative languages: proof #1

Let $\Sigma$ be the specification made of $\ell_X : 1 \to V_X$ and $u_X : V_X \to 1$ such that $\ell_X \circ u_X \equiv id$ and $\ell_Y \circ u_X \equiv \ell_Y \circ \langle \rangle$ when $Y \neq X$. Then $\Sigma$ satisfies the annihilation lookup-update (ALU) property:

$$u_X \circ \ell_X \equiv id$$

Proof. By observation: prove that $\ell_Y \circ u_X \circ \ell_X \equiv \ell_Y$ for each $Y$.

When $Y = X$:

$$\text{(subst)} \quad \frac{\ell_X \circ u_X \equiv id}{\ell_X \circ u_X \circ \ell_X \equiv \ell_X}$$

When $Y \neq X$:

$$\text{(subst)} \quad \frac{\ell_Y \circ u_X \equiv \ell_Y \circ \langle \rangle}{\ell_Y \circ u_X \circ \ell_X \equiv \ell_Y \circ \langle \rangle \circ \ell_X} \quad \text{(unit)} \quad \frac{\langle \rangle \circ \ell_X \equiv id}{\ell_Y \circ \langle \rangle \circ \ell_X \equiv \ell_Y} \quad \text{(repl)} \quad \frac{\langle \rangle \circ \ell_X \equiv id}{\ell_Y \circ \langle \rangle \circ \ell_X \equiv \ell_Y}$$

Hence the state is not modified. □
A property of imperative languages, proof # 2

The **annihilation lookup-update (ALU)** property:

\[ u_X \circ \ell_X \equiv id \]
A property of imperative languages, proof # 2

The annihilation lookup-update (ALU) property:

\[ u_X \circ \ell_X \equiv id \]

Another proof.
The (unit) rule states that \( id \) is the unique \( f : 1 \rightarrow 1 \).

\[
\begin{array}{c}
\text{(unit)} \quad \frac{u_X \circ \ell_X : 1 \rightarrow 1}{u_X \circ \ell_X \equiv id}
\end{array}
\]

\[ \square \]
A property of imperative languages, proof # 2

The annihilation lookup-update (ALU) property:

\[ u_X \circ \ell_X \equiv id \]

Another proof.
The (unit) rule states that \( id \) is the unique \( f : 1 \rightarrow 1 \).

\[
\begin{align*}
(\text{unit}) & \quad \frac{u_X \circ \ell_X : 1 \rightarrow 1}{u_X \circ \ell_X \equiv id}
\end{align*}
\]

BUT in the same way, we could prove for all \( Y \):

\[
\begin{align*}
(\text{unit}) & \quad \frac{u_X \circ \ell_Y : 1 \rightarrow 1}{u_X \circ \ell_Y \equiv id}
\end{align*}
\]

which obviously is FALSE!
Two proofs of (ALU). Proof #1 is right, proof #2 is wrong.

WHY?
Two proofs of (ALU). Proof #1 is right, proof #2 is wrong.

**WHY?**

The (unit) rule should state that \( id \) is the unique \( f : \mathbb{1} \rightarrow \mathbb{1} \) under the assumption that \( f \) cannot modify the state, and it should be impossible to apply this rule to \( u_X \circ \ell_Y \).

How can we formalize this fact?
Questions

Two proofs of (ALU). Proof #1 is right, proof #2 is wrong.

WHY?

The (unit) rule should state that \( \text{id} \) is the unique \( f : 1 \to 1 \) under the assumption that \( f \) cannot modify the state, and it should be impossible to apply this rule to \( u_X \circ l_Y \).

How can we formalize this fact?

By decorating terms and equations.
Decorations: terms and equations

Terms are classified:

- $f^{(0)}$: $f$ is pure if it cannot use nor modify the state.
- $f^{(1)}$: $f$ is an accessor if it can use the state, not modify it.
- $f^{(2)}$: $f$ is a modifier if it can use and modify the state.

Hierarchy rules: $f^{(0)} \leq f^{(1)}$, $f^{(1)} \leq f^{(2)}$.

Equations are classified:

- $f \equiv g$: strong equation: $f$ and $g$ return the same value and they have the same effect on the state.
- $f \sim g$: weak equation: $f$ and $g$ return the same value but they may have different effects on the state.

Hierarchy rule: $f \equiv g \Rightarrow f \sim g$. 

Decorations: terms and equations

Terms are classified:

- $f^{(0)}$: $f$ is pure if it cannot use nor modify the state.
- $f^{(1)}$: $f$ is an accessor if it can use the state, not modify it.
- $f^{(2)}$: $f$ is a modifier if it can use and modify the state.

Hierarchy rules: $f^{(0)} \Rightarrow f^{(1)}$, $f^{(1)} \Rightarrow f^{(2)}$.

Equations are classified:

- $f \equiv g$: strong equation: $f$ and $g$ return the same value and they have the same effect on the state.
- $f \sim g$: weak equation: $f$ and $g$ return the same value but they may have different effects on the state.

Hierarchy rule: $f \equiv g \Rightarrow f \sim g$. 
Decorated rules

The rules of the logic are also decorated, for instance:

\[
\begin{align*}
\text{(unit)} \quad & f : 1 \to 1 \\
\hline
& f \sim id
\end{align*}
\]
Decorated rules

The rules of the logic are also decorated, for instance:

\[(\text{unit}) \quad \frac{f : 1 \to 1}{f \sim id}\]

There are new rules (which become trivial without decorations):

\[(1-\sim\text{-to-}\equiv) \quad \frac{f(1) \ g(1) \ f \sim g}{f \equiv g}\]
Decorated rules

The rules of the logic are also decorated, for instance:

\[(\text{unit}) \quad \frac{f : \mathbb{1} \to \mathbb{1}}{f \sim \text{id}}\]

There are new rules (which become trivial without decorations):

\[(1\sim\text{-to-}\equiv) \quad \frac{f^{(1)} \ g^{(1)} \ f \sim g}{f \equiv g}\]

Hence there are new derived rules, like:

\[(1\text{-unit}) \quad \frac{f^{(1)} : \mathbb{1} \to \mathbb{1}}{f \equiv \text{id}}\]
Proof #2 is wrong: it cannot be properly decorated

Proof #2 of (ALU) can be decorated as follows:

\[
\begin{align*}
(\text{unit}) \quad & \frac{u_X \circ \ell_X : \mathbb{1} \to \mathbb{1}}{u_X \circ \ell_X \sim id}
\end{align*}
\]

which does not entail \( u_X \circ \ell_X \equiv id \).
Proof #2 is wrong: it cannot be properly decorated

Proof #2 of (ALU) can be decorated as follows:

\[
\begin{align*}
\text{(unit)} & \quad \frac{u_X \circ \ell_X : 1 \to 1}{u_X \circ \ell_X \sim id} \\
\end{align*}
\]

which does not entail \( u_X \circ \ell_X \equiv id \).

In fact for each \( Y \) there is a proof:

\[
\begin{align*}
\text{(unit)} & \quad \frac{u_X \circ \ell_Y : 1 \to 1}{u_X \circ \ell_Y \sim id} \\
\end{align*}
\]

which is right but without any interest.
Decorated rules for substitution and replacement

Strong equations form a congruence:

\[
\text{(≡-subs)} \quad \frac{g_1 \equiv g_2}{g_1 \circ f \equiv g_2 \circ f} \quad \text{(≡-repl)} \quad \frac{f_1 \equiv f_2}{g \circ f_1 \equiv g \circ f_2}
\]

Indeed:

\(f_1\) and \(f_2\) may modify the state in a different way, so that \(g \circ f_1\) and \(g \circ f_2\) may return different values if \(g\) is not pure.
Decorated rules for substitution and replacement

Strong equations form a congruence:

\[
\frac{g_1 \equiv g_2}{g_1 \circ f \equiv g_2 \circ f} \quad \text{(\(\equiv\)-subs)}
\]

\[
\frac{f_1 \equiv f_2}{g \circ f_1 \equiv g \circ f_2} \quad \text{(\(\equiv\)-repl)}
\]

Weak equations do not form a congruence:

\[
\frac{g_1 \sim g_2}{g_1 \circ f \sim g_2 \circ f} \quad \text{(\(\sim\)-subs)}
\]

\[
\frac{f_1 \sim f_2}{g \circ f_1 \sim g \circ f_2} \quad \text{(0-\(\sim\)-repl)}
\]

\[
\frac{g(0)}{g \circ f_1 \sim g \circ f_2 : X \rightarrow Z}
\]

Indeed: \(f_1\) and \(f_2\) may modify the state in a different way, so that \(g \circ f_1\) and \(g \circ f_2\) may return different values if \(g\) is not pure.
A decorated specification for states

For each $X$, a type $V_X$ for values, two operations:

- $\ell_X^{(1)} : 1 \to V_X$ (lookup) : an accessor
- $u_X^{(2)} : V_X \to 1$ (update) : a modifier

and weak equations:

- $\ell_X \circ u_X \sim id$
- $\ell_Y \circ u_X \sim \ell_Y \circ \langle \rangle$ when $Y \neq X$
Proof #1 is right: it can be properly decorated

The annihilation lookup-update (ALU) property:

\[ u_X \circ \ell_X \equiv id \]

Proof. By observation: prove that \( \ell_Y \circ u_X \circ \ell_X \sim \ell_Y \) for each \( Y \).

When \( Y = X \):

\[
(\sim\text{-subs}) \quad \frac{\ell_X \circ u_X \sim id}{\ell_X \circ u_X \circ \ell_X \sim \ell_X}
\]
Proof #1 is right: it can be properly decorated

The annihilation lookup-update (ALU) property:

\[
\begin{align*}
\boxed{u_X \circ \ell_X & \equiv id} \\
\end{align*}
\]

Proof. By observation: prove that \( \ell_Y \circ u_X \circ \ell_X \sim \ell_Y \) for each \( Y \).

When \( Y = X \):

\[
(\sim\text{-subs}) \quad \ell_X \circ u_X \sim id \\
\ell_X \circ u_X \circ \ell_X \sim \ell_X
\]

When \( Y \neq X \):

\[
(\sim\text{-subs}) \quad \ell_Y \circ u_X \sim \ell_Y \circ \langle \rangle \\
\ell_Y \circ u_X \circ \ell_X \sim \ell_Y \circ \langle \rangle \circ \ell_X
\]

(1-unit) \( \ell_X^{(1)} \)

\[
\langle \rangle \circ \ell_X \equiv id
\]

(\equiv\text{-repl}) \( \ell_Y \circ \langle \rangle \circ \ell_X \equiv \ell_Y \)

(\equiv\text{-to}\sim) \( \ell_Y \circ \langle \rangle \circ \ell_X \sim \ell_Y \)

\[
\ell_Y \circ u_X \circ \ell_X \sim \ell_Y
\]
Other properties of imperative languages

The 7 properties in [Plotkin&Power 02] can be proved similarly. For instance the **commutation update-update (CUU)** property, is proved in the paper.
Other properties of imperative languages

The 7 properties in [Plotkin\&Power 02] can be proved similarly. For instance the \textit{commutation update-update (CUU)} property, is proved in the paper. When $X \neq Y$:

The order of storing values in $X$ and $Y$ does not matter
Other properties of imperative languages

The 7 properties in [Plotkin&Power 02] can be proved similarly. For instance the commutation update-update (CUU) property, is proved in the paper. When $X \neq Y$:

The order of storing values in $X$ and $Y$ does not matter

which is formalized as:

$$u_Y \circ (u_X \times id) \equiv u_X \circ (id \times u_Y) : V_X \times V_Y \rightarrow 1$$

where $\times$ is the semi-pure product from [Dumas&Duval&Reynaud]

*Cartesian effect categories are Freyd-categories* JSC 2011. ACCAT’09.

\[
\begin{array}{c}
\text{\begin{tikzpicture}[baseline=(current bounding box.center)]
\node (A) at (0,0) {$V_X$};
\node (B) at (1,0) {$1$};
\node (C) at (0,-1) {$V_Y$};
\node (D) at (1,-1) {$V_Y$};
\path[->] (A) edge node [above] {$u_X^{(2)}$} (B);
\path[->] (A) edge node [left] {$\equiv$} (C);
\path[->] (B) edge node [left] {$\equiv$} (D);
\path[->] (C) edge node [below] {$id^{(0)}$} (D);
\end{tikzpicture}}
\end{array}
\]
Explicit proofs

Another way to prove results about states:

1. introduce explicitly a type of states $S$
Explicit proofs

Another way to prove results about states:

1. introduce explicitly a type of states $S$
2. expand (translate) the decorations

<table>
<thead>
<tr>
<th>$f^{(0)} : X \rightarrow Y$</th>
<th>$f : X \rightarrow Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f^{(1)} : X \rightarrow Y$</td>
<td>$f : X \times S \rightarrow Y$</td>
</tr>
<tr>
<td>$f^{(2)} : X \rightarrow Y$</td>
<td>$f : X \times S \rightarrow Y \times S$</td>
</tr>
<tr>
<td>$f \equiv g : X \rightarrow Y$</td>
<td>$f \equiv g : X \times S \rightarrow Y \times S$</td>
</tr>
<tr>
<td>$f \sim g : X \rightarrow Y$</td>
<td>$\pi \circ f \equiv \pi \circ g : X \times S \rightarrow Y$</td>
</tr>
</tbody>
</table>
Explicit proofs

Another way to prove results about states:

1. introduce explicitly a type of states $S$
2. expand (translate) the decorations

<table>
<thead>
<tr>
<th>$f^{(0)} : X \to Y$</th>
<th>$f : X \to Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f^{(1)} : X \to Y$</td>
<td>$f : X \times S \to Y$</td>
</tr>
<tr>
<td>$f^{(2)} : X \to Y$</td>
<td>$f : X \times S \to Y \times S$</td>
</tr>
<tr>
<td>$f \equiv g : X \to Y$</td>
<td>$f \equiv g : X \times S \to Y \times S$</td>
</tr>
<tr>
<td>$f \sim g : X \to Y$</td>
<td>$\pi \circ f \equiv \pi \circ g : X \times S \to Y$</td>
</tr>
</tbody>
</table>

3. prove in the “usual” (not decorated) logic
Explicit proofs

Another way to prove results about states:

1. introduce explicitly a type of states $S$
2. expand (translate) the decorations

<table>
<thead>
<tr>
<th>$f^{(0)} : X \rightarrow Y$</th>
<th>$f : X \rightarrow Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f^{(1)} : X \rightarrow Y$</td>
<td>$f : X \times S \rightarrow Y$</td>
</tr>
<tr>
<td>$f^{(2)} : X \rightarrow Y$</td>
<td>$f : X \times S \rightarrow Y \times S$</td>
</tr>
</tbody>
</table>

$\quad f \equiv g : X \rightarrow Y$

$\quad f \sim g : X \rightarrow Y$

$\quad \pi \circ f \equiv \pi \circ g : X \times S \rightarrow Y$

3. prove in the “usual” (not decorated) logic

But the notion of effect is lost.
A span of “logics”

- decorations → syntax: forget the decorations
- decorations → semantics: expansion, with an explicit $S$ for states
From proofs to models

The expansion:

- maps decorated proofs to “usual” explicit proofs
From proofs to models

The expansion:

- maps decorated proofs to “usual” explicit proofs
- and provides a notion of decorated model

because it can be seen as a functor $F$ with a right adjoint:

\[
\begin{array}{c}
decorations \\ \downarrow \quad F \\semantics \\ \downarrow \quad G
\end{array}
\]

\[
\text{Mod}_{\text{deco}}(\Sigma, G\Theta) \cong \text{Mod}_{\text{expl}}(F\Sigma, \Theta)
\]

For instance:

- $\Sigma$ is the decorated specification for states
- $\Theta$ is $\textbf{Set}$ with the distinguished set $S = \prod_X V_X$
From states to exceptions

- We can prove properties of imperative languages in a logic which respects the syntax of the language.
From states to exceptions

- We can prove properties of imperative languages in a logic which respects the syntax of the language.

- THUS, we can prove properties of exceptions in a logic which respects the syntax of exceptions.
  [Dumas&Duval&Fousse&Reynaud] *Decorated proofs for computational effects: exceptions*. Submitted for publication.
From states to exceptions

- We can prove properties of imperative languages in a logic which respects the syntax of the language.
- THUS, we can prove properties of exceptions in a logic which respects the syntax of exceptions.
  [Dumas&Duval&Fousse&Reynaud]  *Decorated proofs for computational effects: exceptions*. Submitted for publication.

- This is due to the duality between states and the core part of exceptions.
Conclusion and future work

We have designed a framework for effects which provides a denotational semantics and a proof system.
Conclusion and future work

We have designed a framework for effects which provides a denotational semantics and a proof system.

Our projects include:

- Using a proof assistant for proving decorated properties.
- Extending our framework for combining effects by composing spans.
Thank you!