Certification of programs with computational effects

Burak Ekici*

j.w.w. Jean-Guillaume Dumas*, Dominique Duval*, Damien Pous†

*LJK, University Joseph Fourier, France
†LIP, ENS-Lyon, France

Casys-Meff Seminars’14
Grenoble, France

November 20, 2014
Motivation

• proving program properties involving computational (side) effects:
  • State
  • Exceptions

through decorated logic
Motivation

- proving program properties involving computational (side) effects:
  - State
  - Exceptions
  through decorated logic
- with related completeness proofs
Motivation

- proving program properties involving computational (side) effects:
  - State
  - Exceptions
  through decorated logic
- with related completeness proofs
- Coq certification of mentioned proofs
1991 Eugenio Moggi
  • Monads to model effects

2008 Dominguez and Duval
  • Decorated Logic: Cartesian Effect Categories

⇒ equivalence proofs among programs with effects
Decorated logic: exceptions - syntax

\[ f^{(0)} : X \rightarrow Y \]
\[ f^{(1)} : X \rightarrow Y \quad \text{pure} \]
\[ f^{(2)} : X \rightarrow Y \quad \text{thrower/propagator} \]
\[ f : X \rightarrow Y \quad \text{catcher} \]

⇒ Ease of composition: exceptional behaviors are kept implicit.

I.e.,

Given \( f^{(2)} : X \rightarrow Y \) and \( g^{(1)} : Y \rightarrow Z \), \((g \circ f)^{(2)} : X \rightarrow Z\)
Decorated logic: exceptions - syntax

- strong equality (on ordinary and exceptional arguments) \( f \equiv g \)
- weak equality (on ordinary arguments only) \( f \sim g \)

\[
\begin{align*}
&f \equiv g : X \rightarrow Y \\
&f \sim g : X \rightarrow Y
\end{align*}
\]

\[
\begin{align*}
f = g & : X \rightarrow Y \\
f \circ \text{inl}_X &= g \circ \text{inl}_X & : X \rightarrow Y + E
\end{align*}
\]

\(\text{inl}_X\) is the inclusion of \(X\) into \(X+E\)

\(\Rightarrow\) More precise equational proofs of programs: w.r.t. effects and ordinary cases.
The given logic is enriched with some number of rules:

- **Conversion rules**

\[
\begin{align*}
\frac{f(0)}{f(1)} & \quad \frac{f(1)}{f(2)} & \quad \frac{f(d) \equiv g(d')}{f \sim g} & \quad \frac{f(d) \sim g(d')}{f \equiv g} \quad \text{if } \max(d, d') \leq 1
\end{align*}
\]

- **Equivalence rules**
- **Rules on monadic equational logic**
- **Categorical coproduct rules**
- **Observational properties of core exceptional operations**
Soundness of the inference system

Such a formalization allows us to prove primitive properties of programs with exceptions:

- commutation catch-catch: given \( s \neq t \)

\[
\text{try}\{f\} \text{ catch}(t \Rightarrow g \mid s \Rightarrow h) \equiv \text{try}\{f\} \text{ catch}(s \Rightarrow h \mid t \Rightarrow g)
\]
Automatizing mathematics: \( \lambda C \) [Coquand et al’85]

1960 Curry-Howard Correspondence
   • Proofs as programs

1985 Thierry Coquand & Gérard Huet
   • A formal language to write proofs: Calculus of Constructions (\( \lambda C \))
Calculus of Constructions (\(\lambda C\)):

- extension to \(\lambda \rightarrow\) with
  - polymorphism, type operators and dependent types
Calculus of Constructions ($\lambda C$):

- extension to $\lambda \rightarrow$ with
  - polymorphism, type operators and dependent types

1988 Thierry Coquand & Christine Paulin

- $\lambda C +$ inductive definitions $:=$ Calculus of Inductive Constructions (CIC or Coq)
Coq

[Coquand et al’88]

Calculus of Constructions ($\lambda C$):

- extension to $\lambda \to$ with
  - polymorphism, type operators and dependent types

1988 Thierry Coquand & Christine Paulin

- $\lambda C +$ inductive definitions $\Rightarrow$ Calculus of Inductive Constructions (CIC or Coq)

2004 Bruno Barras & Hugo Herbelin

- Coq + Type predicativity $\rightarrow$ to avoid Russell-like paradoxes $\Rightarrow$ Coq 8.0.
Calculus of Constructions ($\lambda C$):

- extension to $\lambda \rightarrow$ with
  - polymorphism, type operators and dependent types

1988 Thierry Coquand & Christine Paulin
  - $\lambda C +$ inductive definitions $\Rightarrow$ Calculus of Inductive Constructions (CIC or Coq)

2004 Bruno Barras & Hugo Herbelin
  - Coq + Type predicativity $\rightarrow$ to avoid Russell-like paradoxes $\Rightarrow$ Coq 8.0.

2014 Bertot & Courtieu & Filliâtre & Marché & Sozeau & …
  - Coq 8.4:
    - proof assistant
    - strongly typed, purely functional programming language
      - not Turing complete: non-termination avoided
    - Co-inductive definitions: to cope with infinite data structures.
      I.e., streams
IMP+EXC is an imperative language enriched with exceptions:
IMP+EXC is an imperative language enriched with exceptions:

Syntax:

\[
\begin{align*}
\text{aexp} & : \quad a_1 \ a_2 \ ::= \ ... \\
\text{bexp} & : \quad b_1 \ b_2 \ ::= \ ... \\
\text{cmd} & : \quad c_1 \ c_2 \ ::= \ \text{skip} | \ x := e | \ c_1; c_2 \ | \ \text{if } b \ \text{then } c_1 \ \text{else } c_2 \ | \\
& \quad \text{while } b \ \text{do } c_1 \ | \ \text{throw } \text{exc} \ | \ \text{try } c_1 \ \text{catch } \text{exc} \Rightarrow c_2
\end{align*}
\]
IMP+EXC is an imperative language enriched with exceptions:

Syntax:

\[
\begin{align*}
\text{aexp: } & \quad a_1 \ a_2 \ ::= \ \ldots \\
\text{bexp: } & \quad b_1 \ b_2 \ ::= \ \ldots \\
\text{cmd : } & \quad c_1 \ c_2 \ ::= \ \text{skip} \mid x := e \mid c_1 ; c_2 \mid \text{if } b \ \text{then } c_1 \ \text{else } c_2 \mid \\
& \quad \text{while } b \ \text{do } c_1 \mid \text{throw exc} \mid \text{try } c_1 \ \text{catch exc} \ \Rightarrow \ c_2
\end{align*}
\]

⇒ Operational semantics of IMP+EXC: IMP-STATES-EXCEPTIONS

(source code) (IMP-STATES-EXCEPTIONS)
E.g.,

\[
\begin{align*}
\text{prog}_1 &= ( \\
&\text{var } x, y; \\
&x := 1; y := 23; \\
&\text{try} ( \\
&\quad \text{while}(tt) \text{ do } ( \\
&\qquad \text{if}(x \leq 0) \\
&\qquad \quad \text{then} (\text{throw } e) \\
&\qquad \quad \text{else} (x := x - 1) \\
&\qquad ) \\
&\quad ) \\
&\quad \text{catch } e => (y := 7); \\
&y := 45; \\
&\text{)}. \quad == \\
\end{align*}
\]

\[
\begin{align*}
\text{prog}_2 &= ( \\
&\text{var } x, y; \\
&x := 0; y := 45; \\
&\text{)}. \quad ==
\end{align*}
\]
So far & future work

So far:

- A Coq library for the global states:
  - with Hilbert-Post Completeness proof
- A Coq library for exceptions
- A Coq library for combined states and exceptions
- IMP specifications:
  - IMP-STATES
  - IMP-STATES-EXCEPTIONS
- All sources on http://coqeffects.forge.imag.fr
So far & future work

So far:

- A Coq library for the global states:
  - with Hilbert-Post Completeness proof
- A Coq library for exceptions
- A Coq library for combined states and exceptions
- IMP specifications:
  - IMP-STATES
  - IMP-STATES-EXCEPTIONS
- All sources on http://coqeffects.forge.imag.fr

Future:

- Hilbert-Post Completeness proof for exceptions
- systematic way to compose effects + generalization
The end!

Many thanks for your kind attention!

Questions?