Relative Hilbert-Post completeness for exceptions

Dominique Duval, with
Jean-Guillaume Dumas, Burak Ekici,
Damien Pous, Jean-Claude Reynaud

[arXiv:1503.00948]

Géocalisation à Chambéry, 10 juin 2015

Outline

Reasoning about programs involving exceptions...

... is difficult:

- exceptions are computational effects:
  a program $X \to Y$
  is interpreted as a function $X \to Y + E$
  (where E is the set of exceptions)
- the handling mechanism is encapsulated
  in a single `try-catch` block
  which propagates exceptions: $X \to Y + E$
  BUT it relies on the `catch` part
  which recovers from exceptions: $X + E \to Y + E$

Logics for programs involving exceptions

- effects: no type of exceptions E
  but decorations:

term	decoration	interpretation
pure term	$f^{(0)} : X \to Y$	$f : X \to Y$
thrower/propagator	$f^{(1)} : X \to Y$	$f : X \to Y + E$
catcher	$f^{(2)} : X \to Y$	$f : X + E \to Y + E$

- encapsulation: 2 related languages:
  - programmers' language: with $\mathtt{throw}^{(1)}$ and $\mathtt{try\text{-}catch}^{(1)}$
    and rather sophisticated equations
  - core language: with $\mathtt{tag}^{(1)}$ and $\mathtt{untag}^{(2)}$
    and a single weak equation: $\mathtt{untag} \circ \mathtt{tag} \sim id$

Weak equations

$$\texttt{untag} \circ \texttt{tag} \sim id$$

Both members coincide on non-exceptional arguments
but they may differ on exceptional arguments.

$$\begin{array}{ccccccccc}
 & \texttt{tag} & & & (propagation) & & & \texttt{untag} & \\
p & \mapsto & \boxed{p} & \mapsto & \dots & \mapsto & \boxed{p} & \mapsto & p \\
\boxed{p} & \mapsto & \boxed{p} & \mapsto & \dots & \mapsto & \boxed{p} & \mapsto & p
\end{array}$$

Thus, equations are decorated, as well:

equation	decoration	interpretation
strong equation	$f \equiv g$	$\forall x\ f(x) = g(x)$
weak equation	$f \sim g$	$\forall x \notin E\ f(x) = g(x)$

"Strong" and "Weak" differ only for catchers:

$$f^{(2)} \equiv g^{(2)} \implies f^{(2)} \sim g^{(2)}$$
$$f^{(1)} \equiv g^{(1)} \iff f^{(1)} \sim g^{(1)}$$

Two languages for exceptions

The core language ($\mathbb{0}$ is the empty type):

- $\mathtt{tag}^{(1)} : P \to \mathbb{0}$
- $\mathtt{untag}^{(2)} : \mathbb{0} \to P$
- $\mathtt{untag} \circ \mathtt{tag} \sim id_P$

is extended with:

- $(\mathtt{CATCH}(b^{(1)}))^{(2)} : Y \to Y$ such that
  $\mathtt{CATCH}(b) \circ [\,]_Y \equiv b \circ \mathtt{untag}$ and $\mathtt{CATCH}(b) \sim id_Y$
- $(\mathtt{TRY}(a^{(1)}, k^{(2)}))^{(1)} : X \to Y$ such that
  $\mathtt{TRY}(a, k) \sim k \circ a$

The translation is defined as:

- $\mathtt{throw}_Y^{(1)} \mapsto [\,]_Y \circ \mathtt{tag} : P \to Y$
- $(\mathtt{try}(a)\mathtt{catch}(b))^{(1)} \mapsto \mathtt{TRY}(a, \mathtt{CATCH}(b)) : X \to Y$

Proposition. The translation from the programmers' language to the core language for exceptions is correct.

Some related work

- About effects: monads [Moggi 1991], effect systems [Lucassen&Gifford 1988], Lawvere theories [Plotkin&Power 2002], algebraic handlers [Plotkin&Pretnar 2009], comonads [Uustalu&Vene 2008] [Petricek&Orchard&Mycroft 2014], dynamic logic [Mossakowski&Schröder&Goncharov 2010],...
- Implementations: Haskell, Idris, Eff, Ynot,...
- About completeness properties of effects: (global) states [Pretnar 2010], local states [Staton 2010],...

Our specificity lies in:

- the use of decorated logic for keeping close to the syntax: decorations often correspond to keywords of the languages
- the use of relative completeness: useful for combining effects

Outline

Categorical view of computation

Various syntactic and semantic notions are treated uniformly

- Syntax: a theory is a (...)-category, generated by some kind of presentation (signature, axioms,...)
- Semantics: a domain of interpretation is a (...)-category, and a model of a theory in a domain is a (...)-functor

Most famous example:
(...)-category = cartesian closed category
for simply typed lambda-calculus

Most simple example

(...)-category = category
for monadic equational logic

Example:

- Syntax: theory generated by:
  sorts U, Z
  operations $z : U \to Z$, $s, p : Z \to Z$
  equations $p \circ s = id_Z$, $s \circ p = id_Z$
- Semantics: model "of integers" in **Set**:

Theory	$\to$	*Domain*
U		$\{*\}$
Z		$\mathbb{Z}$
z		0
s		$x \mapsto x+1$
p		$x \mapsto x-1$

Decorations

(...)-category = decorated category
here for the core language for exceptions:
Example:

- Syntax: the theory generated by a pure part
  sorts U, Z, operations $z^{(0)}, s^{(0)}, p^{(0)}$, equations..., and:
  propagator: $\mathtt{tag}^{(1)} : Z \to \mathbb{0}$
  catcher: $\mathtt{untag}^{(2)} : \mathbb{0} \to Z$
  weak equation: $\mathtt{untag} \circ \mathtt{tag} \sim id$
- Semantics:
  the model "of integers" in **Set** and:

Theory	$\to$	*Domain*	
$\mathtt{tag}^{(1)} : Z \to \mathbb{0}$		$\mathtt{tag} : \mathbb{Z} \to E$	$p \mapsto \boxed{p}$
$\mathtt{untag}^{(2)} : \mathbb{0} \to Z$		$\mathtt{untag} : E \to \mathbb{Z} + E$	$\boxed{p} \mapsto p$

Soundness and completeness

- In this framework, soundness of equational semantics with respect to denotational semantics is granted:
  Provable $\Longrightarrow$ Valid
- But completeness is not satisfied, in general, whatever the notion of completeness:

* Semantic completeness:
  Valid $\Longrightarrow$ Provable
* Syntactic completeness:
  Every added unprovable sentence introduces an inconsistency, where inconsistency means:
  - either negation inconsistency:
    there is a sentence φ such that φ and $\neg\varphi$ are provable
  - or Hilbert-Post inconsistency:
    every sentence is provable

Hilbert-Post completeness

- (Absolute) H-P completeness (wrt to a logic L)
  A theory T is H-P complete if:
  - at least one sentence is unprovable from T
  - and every theory containing T
    either is T or is made of all sentences

  i.e., T is maximally consistent

- Relative H-P completeness (wrt to two logics $L_0 \subseteq L$)
  A theory T is relatively H-P complete wrt L_0 if:
  - at least one sentence is unprovable from T
  - and every theory containing T
    can be generated from T and some sentences in L_0

  i.e., T is maximally consistent "up to L_0"

Main results

Theorems (Completeness)
Both languages for exceptions are relatively Hilbert-Post complete with respect to their pure part

Proofs (Burak Ekici's thesis)
Done with the decorated logic, and checked in Coq

Outline

1. For each (non-pure) decoration,
   find canonical forms for terms
2. For each combination of decorations,
   prove that each equation between terms in canonical form
   is equivalent to a set of equations between pure terms

Canonical forms for terms

- Programmer's language, propagator $a^{(1)}$:
  $a^{(1)} \equiv \mathtt{throw}_Y^{(1)} \circ u^{(0)}$
- Core language, propagator $a^{(1)}$:
  $a^{(1)} \equiv [\,]_Y^{(0)} \circ \mathtt{tag}^{(1)} \circ u^{(0)}$
- Core language, catcher $f^{(2)}$:
  $f^{(2)} \equiv a^{(1)} \circ \mathtt{untag}^{(2)} \circ \mathtt{tag}^{(1)} \circ u^{(0)}$
  ("keep the first `untag` only")

Outline

- We have introduced the notion of relative Hilbert-Post completeness.
- This notion looks well-suited to effects: they are built on top of some “arbitrary” pure part, which is often incomplete.
- We have proved, and checked in Coq, that both decorated languages for exceptions are relatively H-P complete.
- We have proved, and checked in Coq, that a decorated language for states is relatively H-P complete.

Towards “structured” decorated categories

$$\begin{tikzcd}
\text{categories} \arrow[r] \arrow[d] & (\ldots)\text{-categories} \arrow[d, dashed] \\
\text{decorated categories} \arrow[r, dashed] & \text{decorated } (\ldots)\text{-categories}
\end{tikzcd}$$

THANKS FOR YOUR ATTENTION!