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 with exceptions

Relative Hilbert-Post completeness

Conclusion

Reasoning about programs involving exceptions...

... is difficult:

exceptions are computational effects:
 a program X → Y
 is interpreted as a function X → Y + E
 (where E is the set of exceptions)

► the handling mechanism is encapsulated in a single try-catch block which propagates exceptions: X → Y + E BUT it relies on the catch part which recovers from exceptions: X + E → Y + E

Logics for programs involving exceptions

effects: no type of exceptions E

but decorations:

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

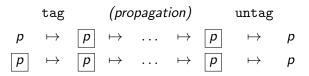
- encapsulation: 2 related languages:
 - programmers' language: with throw⁽¹⁾ and try-catch⁽¹⁾ and rather sophisticated equations

► core language: with tag⁽¹⁾ and untag⁽²⁾ and a single weak equation: untag ◦ tag ~ id

Weak equations

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

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



Thus, equations are decorated, as well:

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

"Strong" and "Weak" differ only for catchers: $\begin{array}{l}
f^{(2)} \equiv g^{(2)} \implies f^{(2)} \sim g^{(2)} \\
f^{(1)} \equiv g^{(1)} \iff f^{(1)} \sim g^{(1)}
\end{array}$

Two languages for exceptions

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

- ▶ $tag^{(1)}: P \rightarrow 0$
- untag⁽²⁾ : $\mathbb{O} \rightarrow P$
- untag \circ tag \sim *id*_P

is extended with:

► $(CATCH(b^{(1)}))^{(2)} : Y \to Y$ such that $CATCH(b) \circ []_Y \equiv b \circ untag \text{ and } CATCH(b) \sim id_Y$ $(TDV((1) + (2)))^{(1)} = Y = Y$

•
$$(\operatorname{TRY}(a^{(1)}, k^{(2)}))^{(1)} : X \to Y$$
 such that $\operatorname{TRY}(a, k) \sim k \circ a$

The translation is defined as:

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

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

Reasoning with exceptions

Relative Hilbert-Post completeness

Conclusion

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

 (\dots) -category = category for monadic equational logic

Example:

- Syntax: theory generated by: sorts U, Z operations z : U → Z, s, p : Z → Z equations p ∘ s = id_Z, s ∘ p = id_Z
- Semantics: model "of integers" in Set:

Theory	\rightarrow	Domain
U		{*}
Z		\mathbb{Z}
z		0
5		$x \mapsto x + 1$
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: tag⁽¹⁾ : Z → 0 catcher: untag⁽²⁾ : 0 → Z weak equation: untag ∘ tag ~ id
- Semantics:

the model "of integers" in Set and:

Theory
$$\rightarrow$$
Domain $tag^{(1)}: Z \rightarrow \mathbb{O}$ $tag: \mathbb{Z} \rightarrow E$ $p \mapsto p$ $untag^{(2)}: \mathbb{O} \rightarrow Z$ $untag: E \rightarrow \mathbb{Z} + E$ $p \mapsto p$

Soundness and completeness

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

 $\mathsf{Valid} \implies \mathsf{Provable}$

* Syntactic completeness:

Every added unprovable sentence introduces an inconsistency, where inconsistency means:

- either negation inconsistency:
 there is a sentence φ such that φ and ¬φ 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 L₀ ⊆ L) A theory T is relatively H-P complete wrt L₀ if:
 - at least one sentence is unprovable from T
 - and every theory containing T
 can be generated from T and some sentences in L₀
 - 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
- 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
$$a^{(1)}$$
:
 $a^{(1)} \equiv \operatorname{throw}_Y^{(1)} \circ u^{(0)}$

► Core language, propagator $a^{(1)}$: $a^{(1)} \equiv []^{(0)}_{Y} \circ tag^{(1)} \circ u^{(0)}$

► Core language, catcher
$$f^{(2)}$$
:
 $f^{(2)} \equiv a^{(1)} \circ \text{untag}^{(2)} \circ \text{tag}^{(1)} \circ u^{(0)}$
("keep the first untag only")

◆□▶ ◆□▶ ◆三▶ ◆三▶ 三三 のへぐ

Outline

Reasoning with exceptions

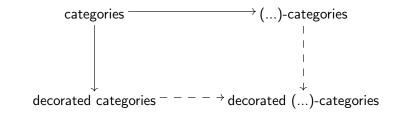
Relative Hilbert-Post completeness

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ 臣 のへぐ

Conclusion

- 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



▲□▶ ▲□▶ ▲目▶ ▲目▶ 目 のへで

THANKS FOR YOUR ATTENTION!