

Decorated semantics for an imperative language with exceptions

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

Work in progress

GdT Plume, ENS Lyon, 21 mars 2016

The language IMP-EX

Syntax

Arithmetic expressions:

$$a ::= 0 \mid 1 \mid -1 \mid 2 \mid -2 \mid \dots \mid \ell_1 \mid \ell_2 \mid \dots \mid a + a \mid a - a \mid a \times a$$

Boolean expressions:

$$b ::= \text{true} \mid \text{false} \mid \neg b \mid b \wedge b \mid b \vee b \mid a = a \mid a > a$$

Commands:

$$c ::= \begin{cases} \text{skip} \mid c; c \mid \ell_i := a \mid \\ \text{if}(b)\text{then}(c)\text{else}(c) \mid \text{while}(b)\text{do}(c) \mid \\ \text{throw}(\text{exn}_i) \mid \text{try}(c)\text{catch}(\text{exn}_i \Rightarrow c) \end{cases}$$

Programs:

$$pg ::= c; \text{return}(a) \mid c; \text{return}(b)$$

Semantics

Denotational: in the category of sets and partial functions

Operational: small-step, big-step

Predicate transformer semantics, ...

Theorem *"All semantics for IMP-EX coincide".*

Aims and tools

Aims.

- ▶ **Design** a “kind of” equational logic \mathcal{L} , close to the syntax, for reasoning about imperative programs with exceptions.
- ▶ **Translate** the syntax of IMP-EX into the logic \mathcal{L} .
- ▶ **Prove** properties of programs of IMP-EX in the logic \mathcal{L} .
- ▶ **Implement** this proof system in Coq.

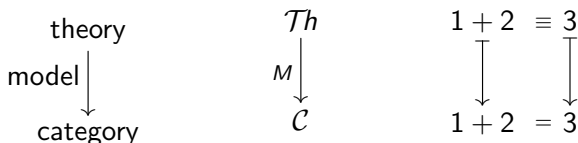
Tools.

- ▶ [Moggi 1989] “**effects as monads**” .
Terms of type B with a parameter of type A
are not interpreted by morphisms from A to B
but by morphisms from A to $T(B)$ for some monad T .
- ▶ Here, more generally, “**effects as functors**” .
Terms of type B with a parameter of type A
are not interpreted by morphisms from A to B
but by morphisms from $H(A)$ to $H(B)$ for some functor H .

Outline

Logic and categories

- **Syntax** and **equational semantics**: a **theory** Th
(a category with a congruence \equiv)
generated by a signature and equations.
- **Denotational semantics**: a **model** $M : Th \rightarrow \mathcal{C}$
(a functor mapping \equiv to $=$)
where \mathcal{C} is “given by mathematics”
(e.g., $\mathcal{C} = \mathcal{Set}$ or $\mathcal{C} = \mathcal{Part}$).

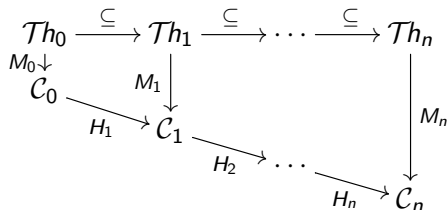


Soundness: granted

Remark: usually *structured* categories and functors

Decorated logic: theories and models

Simply “enlarge” the previous diagram



where the functor $\mathcal{Th}_{i-1} \subseteq \mathcal{Th}_i$

- ▶ is the identity on objects
- ▶ preserves \equiv and is “ \equiv -faithful”: for all $f, g : X \rightarrow Y$ in \mathcal{Th}_{i-1}

$$f \equiv g \text{ in } \mathcal{Th}_{i-1} \iff f \equiv g \text{ in } \mathcal{Th}_i$$

Decoration of terms (notation): $f^{(d)}$ iff $f \in \mathcal{Th}_d$

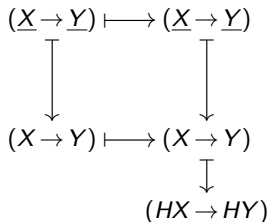
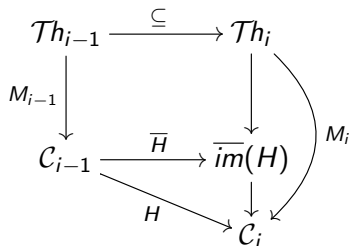
conversions: $f^{(d)} \implies f^{(d+1)}$

Soundness: if each H_i is faithful

Full image

The **full image** of a functor $H : \mathcal{C}_{i-1} \rightarrow \mathcal{C}_i$ is the category $\overline{im}(H)$ with:

- ▶ the same objects as \mathcal{C}_{i-1}
- ▶ an arrow $f : X \rightarrow Y$ for each $f : H(X) \rightarrow H(Y)$ in \mathcal{C}_i .



Soundness: if \overline{H} is faithful

Kleisli category

The **Kleisli category** of a monad $T : \mathcal{C} \rightarrow \mathcal{C}$ is the category \mathcal{C}_T with:

- ▶ the same objects as \mathcal{C}
- ▶ an arrow $f : X \rightarrow Y$ for each $f : X \rightarrow T(Y)$ in \mathcal{C} .

$$\begin{array}{ccccc}
 \mathcal{C}h_{i-1} & \xrightarrow{\subseteq} & \mathcal{C}h_i & \xrightarrow{\subseteq} & \mathcal{C}h_{i+1} \\
 M_{i-1} \downarrow & & M_i \downarrow & & \downarrow M_{i+1} \\
 \mathcal{C} & \xrightarrow{\quad} & \mathcal{C}_T & \xrightarrow{\quad} & \overline{im}(T) \\
 & \searrow T & & & \downarrow \\
 & & & & \mathcal{C}
 \end{array}$$

M_{i+1}

$$\begin{array}{ccccc}
 (\underline{X} \rightarrow \underline{Y}) \vdash (\underline{X} \rightarrow \underline{Y}) \vdash (\underline{X} \rightarrow \underline{Y}) \\
 \downarrow & & \downarrow & & \downarrow \\
 (X \rightarrow Y) \vdash (X \rightarrow Y) \vdash (X \rightarrow Y) \\
 & & \vdots & & \downarrow \\
 (X \rightarrow TY) \vdash (TX \rightarrow TY)
 \end{array}$$

Soundness: if each component of the unit $\eta : Id \Rightarrow T$ is mono

Decorated logic: decorated equations

Notation: $\boxed{f \bullet g = g \circ f}$ when $\bullet \xrightarrow{f} \bullet \xrightarrow{g} \bullet$

In each theory:

► a **congruence** \equiv :

- equivalence relation between parallel terms
- compatible with composition

$$g_1 \equiv g_2 \implies f \bullet g_1 \bullet h \equiv f \bullet g_2 \bullet h$$

► a **weak congruence** (or several):

- extends \equiv
- preorder relation between parallel terms
- “sometimes” symmetry
- “sometimes” **substitution**

$$g_1 \equiv g_2 \implies f \bullet g_1 \equiv f \bullet g_2$$

- “sometimes” **replacement**

$$g_1 \equiv g_2 \implies g_1 \bullet h \equiv g_2 \bullet h$$

Outline

The language XS-IMP

Syntax

Expressions:

$$a ::= 0 \mid 1 \mid -1 \mid 2 \mid -2 \mid \dots \mid \ell \mid s(a) \mid p(a)$$
$$b ::= \text{true} \mid \text{false} \mid \neg b \mid a = 0 \mid a > 0$$
$$e ::= a \mid b$$

Commands:

$$c ::= \text{skip} \mid c; c \mid \ell := a$$

Programs:

$$pg ::= c; \text{return}(e)$$

Restrictions (easy to remove):

- ▶ only one location ℓ
- ▶ no binary operation on expressions

Later:

- ▶ exceptions, conditionals, loops

Decorated logic for states

Comonad $D(X) = S \times X$

$$\mathcal{Th}_0 \xrightarrow{\subseteq} \mathcal{Th}_1 \xrightarrow{\subseteq} \mathcal{Th}_2$$

$$\begin{array}{ccccc} (X \rightarrow Y) \vdash & \longrightarrow & (X \rightarrow Y) \vdash & \longrightarrow & (X \rightarrow Y) \\ \Downarrow & & \Downarrow & & \Downarrow \\ (X \rightarrow Y) \vdash & \longrightarrow & (X \rightarrow Y) \vdash & \longrightarrow & (X \rightarrow Y) \\ & & \Downarrow & & \Downarrow \\ & & (S \times X \rightarrow Y) \vdash & \cdots \longrightarrow & (S \times X \rightarrow S \times Y) \end{array}$$

Weak equations $f_1 \sim_{st} f_2 : X \rightarrow Y$

interpreted as: $f_1 \bullet \varepsilon_Y = f_2 \bullet \varepsilon_Y : S \times X \rightarrow Y$

$$S \times X \xrightarrow{f_1} S \times Y \xrightarrow{\varepsilon_Y} Y$$

$$\xrightarrow{f_2} S \times Y \xrightarrow{\varepsilon_Y} Y$$

\sim_{st} satisfies substitution and **pure** replacement:

$$g_1 \sim_{st} g_2 \implies f \bullet g_1 \bullet h^{(0)} \sim_{st} f \bullet g_2 \bullet h^{(0)}$$

Pure operations and equations

The **pure** theory Th_0 contains:

- ▶ sorts $\mathbb{1}, A, B$
- ▶ operations $0, 1, -1, \dots : \mathbb{1} \rightarrow A, s, p : A \rightarrow A,$
 $\text{true}, \text{false} : \mathbb{1} \rightarrow B, \text{not} : B \rightarrow B, \text{null?}, \text{pos?} : A \rightarrow B$
- ▶ equations $s(0) \equiv 1, p(0) \equiv -1, \dots, s \bullet p \equiv id_A, p \bullet s \equiv id_A,$
 $\text{true} \bullet \text{not} \equiv \text{false}, \dots$

$M_0 : Th_0 \rightarrow Set$ interprets A as the set A of integers,
 B as the set B of truth values, etc

Operations and equations for states

In *Set*: a **set of states** S with (here) $S \cong A$, denoted $\boxed{x} \leftrightarrow x$

Then \mathcal{Th}_1 and \mathcal{Th}_2 are generated from \mathcal{Th}_0 by two operations:

$\text{lookup}^{(1)} : \mathbb{1} \rightarrow A$	$\text{update}^{(2)} : A \rightarrow \mathbb{1}$
$\text{lookup} : S \rightarrow A$	$\text{update} : S \times A \rightarrow S$
$\text{lookup} : \boxed{x} \mapsto x$	$\text{update} : (\boxed{x}, y) \mapsto \boxed{y}$

one weak equation:

$\text{update} \bullet \text{lookup} \sim_{st} id_A$
$\text{update} \bullet \text{lookup} = \varepsilon_A$
$(\boxed{x}, y) \mapsto \boxed{y} \mapsto y$

and decorated rules...

Translation

Expressions: $e \mapsto e^{(1)} : \mathbb{1} \rightarrow Expr$ (where $Expr$ is A or B)

- ▶ $0, 1, \dots \mapsto 0^{(0)}, 1^{(0)}, \dots$, $\text{true}, \text{false} \mapsto \text{true}^{(0)}, \text{false}^{(0)}$
- ▶ $s(a) \mapsto a \bullet s^{(0)}$, $p(a) \mapsto a \bullet p^{(0)}$, $\neg b \mapsto b \bullet \text{not}^{(0)}$, ...
- ▶ $\ell \mapsto \text{lookup}^{(1)}$

Commands: $c \mapsto c^{(2)} : \mathbb{1} \rightarrow \mathbb{1}$

- ▶ $\text{skip} \mapsto id_{\mathbb{1}}^{(0)}$
- ▶ $c_1; c_2 \mapsto c_1 \bullet c_2$
- ▶ $\ell := a \mapsto a \bullet \text{update}^{(2)}$

Programs: $pg \mapsto pg^{(2)} : \mathbb{1} \rightarrow Expr$

- ▶ $c; \text{return}(e) \mapsto c \bullet e$

Forward semantics

Given a **program** $pg^{(2)} : \mathbb{1} \rightarrow Expr$,

find a **result** $rs^{(0)} : \mathbb{1} \rightarrow Expr$ such that $\boxed{pg \sim_{st} rs}$

This means that $pg : S \rightarrow S \times Expr$ and $rs : \mathbb{1} \rightarrow Expr$ satisfy:

$$pg(s) = (s', rs(x)) \text{ for some } s'$$

$$\begin{array}{ccc} S & \xrightarrow{pg} & S \times Expr \\ \varepsilon_{\mathbb{1}} \downarrow & = & \downarrow \varepsilon_{Expr} \\ \mathbb{1} & \xrightarrow{u} & Expr \end{array}$$

This requires an **initialization** of the state
and the derived **strong equation**: for each $u^{(0)} : \mathbb{1} \rightarrow A$

$$\boxed{u \bullet \text{update} \bullet \text{lookup} \equiv u \bullet \text{update} \bullet u}$$

Method:

- ▶ first \equiv is used inductively, by **replacement**
- ▶ until finally \sim_{st} can be used, by **pure replacement**

This corresponds to an **operational semantics**.

Forward semantics: an example

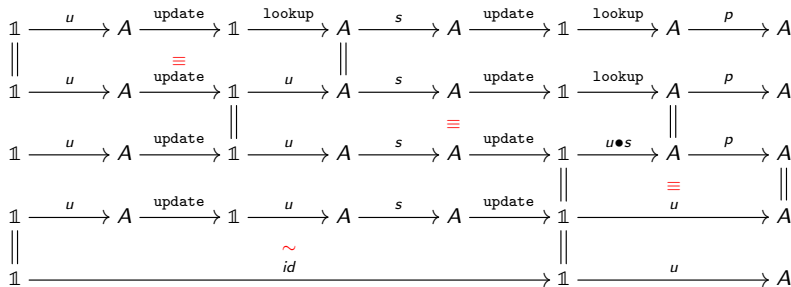
Initialization: $\ell := u^{(0)}$ for any $u^{(0)} : \mathbb{1} \rightarrow A$

The given **program** is

$\ell := u; \ell := s(\ell); \text{return}(p(\ell))$

translated as:

$pg^{(2)} = u^{(0)} \bullet \text{update}^{(2)} \bullet \text{lookup}^{(1)} \bullet s^{(0)} \bullet \text{update}^{(2)} \bullet \text{lookup}^{(1)} \bullet p^{(0)}$



Conclusion: $pg^{(2)} \sim_{st} rs^{(0)}$ where $rs^{(0)} = u$. The **result** is u

Backward semantics

Given a **program** $pg = c; \text{return}(post) : \mathbb{1} \rightarrow Expr$,

find an expression $pre : \mathbb{1} \rightarrow Expr$ such that $pg \sim_{st} \text{return}(pre)$

This means that c , $post$ and pre satisfy:

$$post(c(s)) = pre(s)$$

$$\begin{array}{ccc} S & \xrightarrow{c^{(2)}} & S \\ pre^{(1)} \downarrow & = & \downarrow post^{(1)} \\ Expr & \xrightarrow{id^{(0)}} & Expr \end{array}$$

This requires only the **weak equation**:

$$\text{update} \bullet \text{lookup} \sim_{st} id_A$$

Method:

- ▶ \sim_{st} is used inductively, by **substitution** and **pure replacement**
- ▶ until finally \equiv is used for simplifying pure terms

When $Expr = B$ this corresponds to a **weakest precondition semantics** (here with a restricted language for conditions)

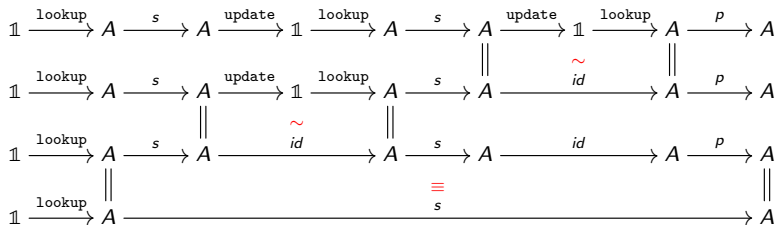
Backward semantics: an example

The given **program** is

$\ell := s(\ell); \ell := s(\ell); \text{return}(p(\ell))$

translated as:

$$pg^{(2)} = \text{lookup}^{(1)} \bullet s^{(0)} \bullet \text{update}^{(2)} \bullet \text{lookup}^{(1)} \bullet s^{(0)} \bullet \text{update}^{(2)} \bullet \text{lookup}^{(1)} \bullet p^{(0)}$$



Conclusion: $pg^{(2)} \sim_{st} \text{lookup} \bullet s^{(0)}$. The “**pre-expression**” is $s(\ell)$

Outline

The language XS-IMP-EX

Syntax

Expressions:

as in XS-IMP

Commands:

$$c ::= \text{skip} \mid c; c \mid \ell := a \mid \text{throw} \mid \text{try}(c)\text{catch}(c)$$

Programs:

$$pg ::= c; \text{return}(e)$$

Restriction (easy to remove):

- ▶ only one exception name (thus, omitted)

Decorated logic for exceptions (only)

Monad $T(X) = X + E$

$$Th_0 \xrightarrow{\subseteq} Th_1 \xrightarrow{\subseteq} Th_2$$

$$\begin{array}{ccccc}
 (X \rightarrow Y) & \vdash & (X \rightarrow Y) & \vdash & (X \rightarrow Y) \\
 \downarrow & & \downarrow & & \downarrow \\
 (X \rightarrow Y) & \vdash & (X \rightarrow Y) & \vdash & (X \rightarrow Y) \\
 & & \vdots & & \downarrow \\
 & & (X \rightarrow Y + E) & \vdash & (X + E \rightarrow Y + E)
 \end{array}$$

Weak equations $f_1 \sim_{\text{ex}} f_2 : X \rightarrow Y$

interpreted as: $\eta_X \bullet f_1 = \eta_X \bullet f_2 : X \rightarrow Y + E$

$$X \xrightarrow{\eta_X} X + E \xrightarrow{f_1} Y + E \quad \xrightarrow{f_2}$$

\sim_{ex} satisfies replacement and **pure** substitution:

$$g_1 \sim_{\text{ex}} g_2 \implies f^{(0)} \bullet g_1 \bullet h \sim_{\text{ex}} f^{(0)} \bullet g_2 \bullet h$$

Operations and equations for exceptions (only)

In *Set*: a **set of exceptions** E with (here) $\mathbb{1} \cong E$, denoted $\star \leftrightarrow \odot \star$

Then \mathcal{Th}_1 and \mathcal{Th}_2 are generated from \mathcal{Th}_0 by two operations:

$\text{tag}^{(1)} : \mathbb{1} \rightarrow \mathbb{0}$	$\text{untag}^{(2)} : \mathbb{0} \rightarrow \mathbb{1}$
$\text{tag} : \mathbb{1} \rightarrow E$	$\text{untag} : E \rightarrow \mathbb{1} + E$
$\text{tag} : \star \mapsto \odot \star$	$\text{untag} : \odot \star \mapsto \star$

one weak equation:

$\text{tag} \bullet \text{untag} \sim_{\text{ex}} \text{id}_{\mathbb{1}}$
$\text{tag} \bullet \text{untag} = \eta_{\mathbb{1}}$
$\star \mapsto \odot \star \mapsto \star$ and $\odot \star \mapsto \odot \star \mapsto \star$

and decorated rules...

Decorated logic for states and exceptions

Duality is broken!

Functor $T(D(X)) = S \times X + S \times E$

$$\mathcal{T}h_0 \xrightarrow{\subseteq} \mathcal{T}h_1 \xrightarrow{\subseteq} \mathcal{T}h_2 \xrightarrow{\subseteq} \mathcal{T}h_3 \xrightarrow{\subseteq} \mathcal{T}h_4$$

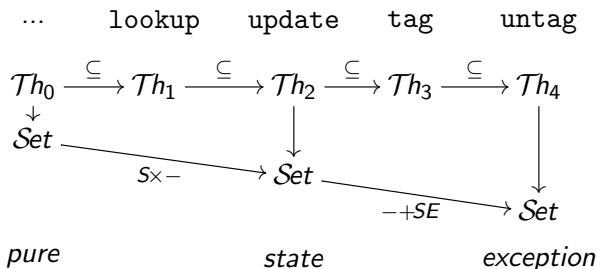
$$\begin{array}{ccccc} (X \rightarrow Y) \vdash & \longrightarrow & (X \rightarrow Y) \vdash & \longrightarrow & (X \rightarrow Y) \vdash \\ \downarrow & & \downarrow & & \downarrow \\ (X \rightarrow Y) \vdash & \longrightarrow & (X \rightarrow Y) \vdash & \longrightarrow & (X \rightarrow Y) \vdash \\ & & \downarrow & & \downarrow \\ & & (S \times X \rightarrow S \times Y) \vdash & \longrightarrow & (S \times X \rightarrow S \times Y) \vdash \\ & & & & \downarrow \\ & & & & (S \times X + S \times E \rightarrow S \times Y + S \times E) \end{array}$$

pure

state

exception

Operations for states and exceptions: summary



- ▶ $f^{(1)}$: may **use** the state
- ▶ $f^{(2)}$: may use and **modify** the state
- ▶ $f^{(3)}$: may use and modify the state, may **raise** exceptions and must **propagate** exceptions
- ▶ $f^{(4)}$: may use and modify the state, may raise exceptions and must propagate exceptions, may **recover** from exceptions

Decorated equations for states and exceptions

Weak equations

$$SX \xrightarrow{\eta_{SX}} SX + SE \xrightarrow[f_2]{f_1} SY + SE \xrightarrow{\varepsilon_Y + SE} Y + SE$$

► $f_1 \sim_{\text{ex}} f_2 : X \rightarrow Y$

interpreted as: $\eta_{SX} \bullet f_1 = \eta_{SX} \bullet f_2$

$$g_1 \sim_{\text{ex}} g_2 \implies f^{(2)} \bullet g_1 \bullet h \sim_{\text{ex}} f^{(2)} \bullet g_2 \bullet h$$

► $f_1 \sim_{\text{st}} f_2 : X \rightarrow Y$

interpreted as: $f_1 \bullet (\varepsilon_Y + SE) = f_2 \bullet (\varepsilon_Y + SE)$

$$g_1 \sim_{\text{st}} g_2 \implies f \bullet g_1 \bullet h^{(0)} \sim_{\text{st}} f \bullet g_2 \bullet h^{(0)}$$

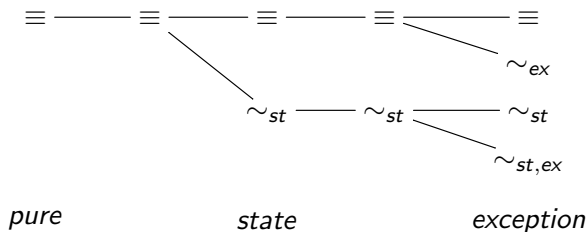
► $f_1 \sim_{\text{st},\text{ex}} f_2 : X \rightarrow Y$

interpreted as: $\eta_{SX} \bullet f_1 \bullet (\varepsilon_Y + SE) = \eta_{SX} \bullet f_2 \bullet (\varepsilon_Y + SE)$

$$g_1 \sim_{\text{st},\text{ex}} g_2 \implies f^{(2)} \bullet g_1 \bullet h^{(0)} \sim_{\text{st},\text{ex}} f^{(2)} \bullet g_2 \bullet h^{(0)}$$

Equations for states and exceptions: summary

$$\mathcal{Th}_0 \xrightarrow{\subseteq} \mathcal{Th}_1 \xrightarrow{\subseteq} \mathcal{Th}_2 \xrightarrow{\subseteq} \mathcal{Th}_3 \xrightarrow{\subseteq} \mathcal{Th}_4$$



Translation

Expressions: $e \mapsto e^{(1)} : \mathbb{1} \rightarrow Expr$

as for XS-IMP

Commands: $c \mapsto c^{(3)} : \mathbb{1} \rightarrow \mathbb{1}$ (really (3), not (4))

- ▶ skip, $c_1; c_2$, $\ell := a$:
as for XS-IMP
- ▶ throw $\mapsto \text{tag}^{(3)} \bullet []_{\mathbb{1}}^{(0)}$
pretends that the exception has type $\mathbb{1}$, instead of $\mathbb{0}$
- ▶ try(c_1)catch(c_2) $\mapsto (\downarrow (c_1 \bullet [id_{\mathbb{1}} | \text{untag}^{(4)} \bullet c_2])^{(3)})$
(next slide)

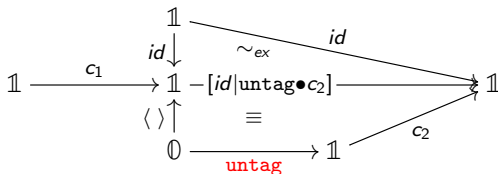
Programs: $pg \mapsto pg^{(3)} : \mathbb{1} \rightarrow Expr$ (really (3), not (4))

as for XS-IMP

Translation of try-catch

$$\text{try}(c_1)\text{catch}(c_2) \mapsto (\downarrow (c_1 \bullet [id_{\mathbb{1}} \mid \text{untag}^{(4)} \bullet c_2]))^{(3)}$$

Uses: the decorated coproduct $\mathbb{1} = \mathbb{1} + \mathbb{0}$



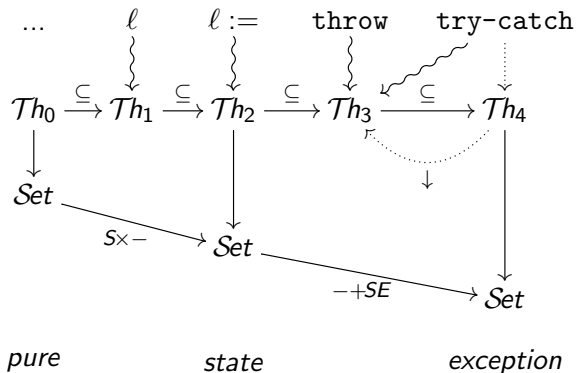
and the “downcast” operator \downarrow

$$(\downarrow (f^{(4)}))^{(3)} \text{ is such that } f \sim_{ex} \downarrow f$$

- ▶ $\downarrow f$ is the same as f on non-exceptional arguments
- ▶ $\downarrow f$ propagates exceptions while f may recover from exceptions

Rules for \downarrow include: $\downarrow (f_1) \equiv \downarrow (f_2) \iff f_1 \sim_{ex} f_2$

Translation of XS-IMP-EX: summary



Backward semantics

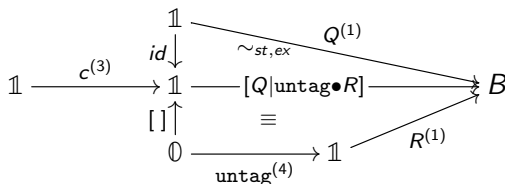
Predicate transformer semantics [Claude Marché, MPRI 2012]

Hoare triples: $\{P\} c \{Q|R\}$ is valid if:

if c is executed in a state satisfying P then:

- if c terminates normally in a state s' then s' satisfies Q
- if c terminates abruptly in a state s' then s' satisfies R

This means that $P \sim_{st,ex} c \bullet [Q \mid \text{untag} \bullet R]$



Outline

The language IMP-EX: syntax, revisited

Expressions:

$$a ::= 0 \mid 1 \mid -1 \mid 2 \mid -2 \mid \dots \mid \ell \mid s(a) \mid p(a)$$
$$b ::= \text{true} \mid \text{false} \mid \neg b \mid a = 0 \mid a > 0$$
$$e ::= a \mid b$$

Commands:

$$c ::= \begin{cases} \text{skip} \mid c; c \mid \ell := a \mid \\ \text{throw} \mid \text{try}(c)\text{catch}(c) \mid \\ \text{if}(b)\text{then}(c)\text{else}(c) \mid \text{repeat}(c) \end{cases}$$

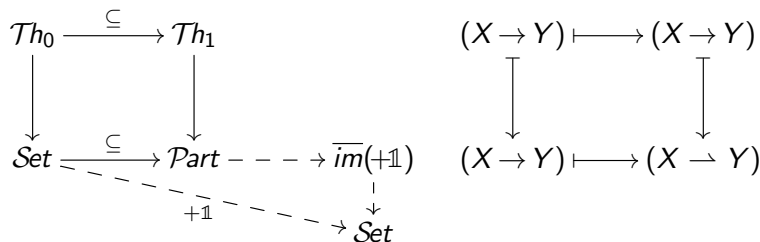
Programs:

$$pg ::= c; \text{return}(e)$$

- ▶ As before: only one location, no binary operation on expressions, only one exception name
- ▶ In addition: $\text{repeat}(c)$ “instead of” $\text{while}(b)\text{do}(c)$

Decorated logic for non-termination

Partiality



Weak equations are inequations $f_1 \succcurlyeq f_2 : X \rightarrow Y$

interpreted as: $f_1 \geq f_2 : X \rightarrow Y$ (as partial functions)

$Part$ with \geq is a 2-category

\succcurlyeq satisfies replacement and substitution:

$$g_1 \succcurlyeq g_2 \implies f \bullet g_1 \bullet h \succcurlyeq f \bullet g_2 \bullet h$$

Operations and equations for non-termination

\mathcal{Th}_1 is generated from \mathcal{Th}_0 by
one operation constructor:

$\text{loop}(c)^{(1)} : X \rightarrow X$ for each $c^{(1)} : X \rightarrow X$

$\text{loop}(c) : X \rightarrow X$ is the least fixed point of $f \mapsto c \bullet f$
--

one strong equation:

$\text{loop}(c) \equiv c \bullet \text{loop}(c)$
--

and decorated rules, including:

$f \equiv c \bullet f \implies f \succcurlyeq \text{loop}(c)$

A weak congruence

The “weakest” congruence for states, exceptions and non-termination is $\preceq_{st,ex}$. For instance:

$$f \preceq_{st,ex} u^{(0)} : X \rightarrow Y$$

is a concise way to express the following:

$f : S \times X + S \times E \rightarrow S \times Y + S \times E$ and $u : X \rightarrow Y$ are such that:
if $f(s, x)$ is defined, then it returns $(s', u(x))$ for some s' .

$$\begin{array}{ccccccc} SX & \xrightarrow{\epsilon_X} & SX + SE & \xrightarrow{f} & SY + SE & \xrightarrow{\epsilon_Y + SE} & Y + SE \\ \eta_{SX} \downarrow & & & \Downarrow & & & \uparrow \eta_Y \\ X & \xrightarrow{\quad\quad\quad} & & & & & Y \\ & & & u & & & \end{array}$$

This is the kind of relation required
between a “**program**” f and its “**result**” u

Translation

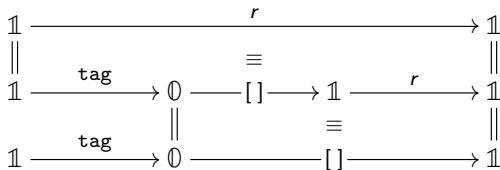
Translation is obvious:

- $\text{repeat}(c) \mapsto \text{loop}(c)$

Example

$\text{repeat}(\text{throw}) \equiv \text{throw}$

$\text{repeat}(\text{throw})$ is translated as $r = \text{loop}(\text{tag} \bullet []_{\perp})$



because r (like all commands) propagates exceptions

Decorated logic for conditions

Weak equations are conditional equations

$$f_1 \sim f_2 : X \rightarrow Y \text{ if } b$$

where \sim is any of the previous (strong or weak) congruence and b is a boolean expression.

For replacement, conditional \sim has the same properties as \sim .

\mathcal{Th}_1 is generated from \mathcal{Th}_0 by

two operation constructors (“conditional non-determinism”):

$$\begin{array}{l} \text{choose}(c_1, c_2)^{(1)} : X \rightarrow Y \text{ for each } c_1^{(1)}, c_2^{(1)} : X \rightarrow Y \\ \downarrow_b (c)^{(0)} : X \rightarrow Y \text{ for each } c^{(1)} : X \rightarrow Y \end{array}$$

$$\begin{array}{l} [c_1 | c_2] : X + X \rightarrow Y \\ \downarrow_b ([c_1 | c_2]) = b \bullet [c_1 | c_2] : X \rightarrow Y \end{array}$$

Decorated logic for IMP-EX

Combine the decorated logics for:

- ▶ states
- ▶ exceptions
- ▶ non-termination
- ▶ and “conditional non-determinism”

by composing the corresponding functors
and extending the corresponding weak congruences

Outline

Conclusion

Remark.

Effects as functors, with their weak congruences, can be seen as a kind of generalization of 2-categories, with **decorated** categorical notions as a generalization of **lax** categorical notions.

To do...

- ▶ “work in progress”:
 - which is the best order for composing the effects?
- ▶ Define **while loops** by:
$$\text{while}(b)\text{do}(c) =$$
$$\text{try}(\text{repeat}(\text{if}(b)\text{then}(c)\text{else}(\text{throw}))\text{ catch}(\text{skip})$$

Prove that indeed such a while loop is the least fixed point of $f \mapsto \text{if}(b)\text{then}(c; f)\text{else}(\text{skip})$
- ▶ Complete the implementation in **Coq**
- ▶ Towards richer languages (C, C++, Java,...)