{IMP + Exc} over Decorated Logic

Burak Ekici*
with J.-G. Dumas*, D. Duval*, D. Pous**

* LJK Grenoble, France
** ENS Lyon, France

February 18, 2014

The Overview

What we have in hand:

1. A generic library to deal with `states` effect
2. A generic library to deal with `exceptions` effect (dual to `states`)
3. Another library: `IMP_STATES` involving $\rightarrow$
   - A simple language (`no name`) in `Coq syntax` but in `IMP semantics`
   - Prove program equivalencies.
4. A generic library to deal with `combined states` and `exceptions effects`
5. Another library: `IMP_STATES + EXCEPTIONS` involving $\rightarrow$
   - A simple language (`no name`) in `Coq syntax` but in `IMP semantics` with `exceptions`
   - Prove program equivalencies including combined effects.

How it looks like: IMP_STATES

```
Inductive ArithExp : Type :=
    | const : Z → ArithExp
    | loc : Loc → ArithExp
    | add : ArithExp → ArithExp → ArithExp
    | multiply : ArithExp → ArithExp → ArithExp.

Fixpoint defArithExp (a: ArithExp) : term Z unit :=
 match a with
    | const n ⇒ (@constant Z n)
    | loc x ⇒ (lookup x)
    | add t1 t2 ⇒ plus o (pair (defArithExp t1) (defArithExp t2))
    | multiply t3 t4 ⇒ mult o (pair (defArithExp t3) (defArithExp t4))
 end.
```

How it looks like: IMP_STATES

```
Inductive BoolExp : Type :=
  | constT : term boolT unit → BoolExp
  | constF : term boolT unit → BoolExp
  | gtT : ArithExp → ArithExp → BoolExp
  | ltT : ArithExp → ArithExp → BoolExp
  | eqT : ArithExp → ArithExp → BoolExp
  | andT : BoolExp → BoolExp → BoolExp
  | orT : BoolExp → BoolExp → BoolExp.

Fixpoint defBoolExp (b: BoolExp) : term boolT unit :=
 match b with
   | constT ttrue ⇒ ttrue
   | constF ffalse ⇒ ffalse
   | gtT a1 a2 ⇒ (gt o (pair (defArithExp a1) (defArithExp a2)))
   | ltT a1 a2 ⇒ (lt o (pair (defArithExp a1) (defArithExp a2)))
   | eqT a1 a2 ⇒ (eq o (pair (defArithExp a1) (defArithExp a2)))
   | andT b1 b2 ⇒ (andB o (pair (defBoolExp b1) (defBoolExp b2)))
   | orT b3 b4 ⇒ (orB o (pair (defBoolExp b3) (defBoolExp b4)))
 end.
```

How it looks like: IMP_STATES

```
Inductive Command : Type :=
    | skip : Command
    | sequence : Command → Command → Command
    | assign : Loc → ArithExp → Command
    | ifthenelse : BoolExp → Command → Command → Command
    | loops : BoolExp → Command → Command

 Fixpoint defCommand (c: Command): (term unit unit) :=
  match c with
    | skip ⇒ (@id unit)
    | sequence c0 c1 ⇒ (defCommand c1) o (defCommand c0)
    | assign i e0 ⇒ (update i) o (defArithExp e0)
    | ifthenelse b c2 c3 ⇒ (copair (defCommand c2) (defCommand c3)) o
                           (defBoolExp b)
    | loops b c4 ⇒ (copair (loopdec o defCommand c4) (@id unit)) o
                   (defBoolExp b)
    end.
```

How it looks like: IMP_STATES

```
Notation "x '::=' a" := (assign x a) (at level 60).
Notation "c1 ';;' c2" := (sequence c1 c2) (at level 65).
Notation "'SKIP'" := (skip) (at level 60).
Notation "'IFB' b 'THEN' t1 'ELSE' t2 'ENDIF'" := (ifthenelse b t1 t2)
                                                    (at level 60).
Notation "'WHILE' b 'DO' c 'ENDWHILE'" := (loops b c) (at level 60).
Notation " x '+++' y" := (add x y) (at level 60).
Notation " x '***' y" := (multiply x y) (at level 60).
Notation " x '≫' y" := (gtT x y) (at level 63).
Notation " x '≪' y" := (ltT x y) (at level 63).
Notation " x '?==' y" := (eqT x y) (at level 63).
Notation " x '&&&' y" := (andT x y) (at level 64).
Notation " x '|||' y" := (orT x y) (at level 64).
Notation "'{{' c '}}'" := (defCommand c) (at level 67).
```

What can we prove?

```
Lemma IMP_ex1: ∀ (x y: Loc), x ≠ y →
       {{x ::= (const 10) ;;
         y ::= (const 3) ;;
        IFB ((loc x) >>= (loc y) *** (const 3))
          THEN (x ::= (loc y))
        ELSE (SKIP)
        ENDIF}}
        ===
       {{x ::= (const 10) ;;
         y ::= (const 3) ;;
        IFB ((loc x) >>= (const 4) *** (loc y))
          THEN (SKIP)
        ELSE (x ::= (loc y))
        ENDIF}}.
```

What can we prove?

```
Lemma IMP_ex2: ∀ (x: Loc),
       {{x ::= (const 10) ;;
        IFB ((loc x) >>= (const 0))
          THEN (IFB ((loc x) ?== (const 10))
                  THEN (x ::= (const 20))
                ELSE (SKIP)
                ENDIF)
        ELSE (x ::= (const 30))
        ENDIF}}
        ===
       {{x ::= (const 20)}}.
```

What can we prove?

```
Lemma IMP_ex3: ∀ (x: Loc),
        {{x ::= (const 2) ;;
          WHILE ((loc x) << (const 11))
            DO (x ::= ((loc x) +++ (const 4)))
          ENDWHILE}}
        ===
        {{x ::= (const 14)}}.
```

How it looks like: IMP_STATES + EXCEPTIONS

The same arithmetic and Boolean expressions with additional throw and try-catch commands.

```
| throw : EName → Command
| trycatch : EName → Command → Command → Command
```

```
| throwexc t0 ⇒ (throw unit t0)
| trycatch t1 c5 c6 ⇒ (@TRY_CATCH _ _ t1 (defCommand c5)
                                          (defCommand c6))
```

```
Notation "'THROW' en" := (throwexc en) (at level 60).
Notation "'TRY' s0 'CATCH' e '⇒' s1" := (trycatch e s0 s1) (at level 60).
```

What can we prove?

```
Lemma IMP_ex4: ∀ (x: Loc), ∀ (e: EName),
        {{IFB (constF ffalse)
             THEN (x ::= (const 5))
           ELSE (x ::= (const 10))
           ENDIF}}
         ===
         {{TRY(IFB (constT ttrue)
                  THEN (THROW e)
                ELSE(x ::= (const 5))
                ENDIF)
          CATCH e ⇒ (x ::= (const 10))}}.
```

What can we prove?

```
Lemma IMP_ex5: ∀ (x y: Loc), ∀ (e: EName),
        {{x ::= (const 1) ;;
          y ::= (const 23) ;;
          TRY(WHILE (constT ttrue)
                DO(IFB ((loc x) ≪= (const 0))
                     THEN (THROW e)
                   ELSE(x ::= ((loc x) +++ (const (-1))))
                   ENDIF)
              ENDWHILE)
          CATCH e ⇒ (y ::= (const 7)) ;;
          y ::= (const 45)}}
       ===
       {{x ::= (const 0) ;;
          y ::= (const 45)}}.
```

To do list:

- Having more generic proofs. E.g.,

```
Lemma IMP_ex6: ∀ (x: Loc), ∀ (p: Z),
          {{ IFB ((const p) ≪ (const 11))
             THEN (x ::= (const 11))
          ELSE (x ::= (const p))
          ENDIF}}
          ===
          {{ IFB ((const p) ≫ (const 11))
             THEN (x ::= (const p))
             ELSE (x ::= (const 11))
          ENDIF}}.
```

- A simple `IMP parser`
- Shorten the proof lengths: using `Ltac` to make `Coq` search the proper way!

The End!

Un grand merci de votre attention !

Questions ?

Terms: Decorated Logic

```
Variable coproductT: Type → Type → Type.
Notation "y '/+/' z" := (coproductT y z) (at level 70).

Variable productT: Type → Type → Type.
Notation "y '/*/' z" := (productT y z) (at level 70).

Definition boolT: Type := (unit/+/unit).

Inductive term: Type → Type → Type :=
 | id: ∀ {X}, term X X
 | comp: ∀ {X Y Z}, term X Y → term Y Z → term X Z
 | downcast: ∀ {X Y} (f: term X Y), term X Y
 | forget: ∀ {X}, term unit X
 | empty: ∀ {X}, term X empty_set
 | pair: ∀ {X Y Z}, term X Z → term Y Z → term (X/*/Y) Z
 | copair: ∀ {X Y Z}, term Z X → term Z Y → term Z (X/+/Y)
 | pi1: ∀ {X Y}, term X (X/*/Y)
 | pi2: ∀ {X Y}, term Y (X/*/Y)
 | copi1: ∀ {X Y}, term (X/+/Y) X
 | copi2: ∀ {X Y}, term (X/+/Y) Y
 | constant: ∀ (X: Set), X → term X unit
```

Terms: Decorated Logic

```
| constant_exc: ∀ (X: Set), X → term X unit
| loopdec: term unit unit
| plus: ∀ {X}, term Z (Z/*/Z)
| mult: ∀ {X}, term Z (Z/*/Z)
| gt: term boolT (Z/*/Z)
| lt: term boolT (Z/*/Z)
| ge: term boolT (Z/*/Z)
| le: term boolT (Z/*/Z)
| eq: term boolT (Z/*/Z)
| neq: term boolT (Z/*/Z)
| andB: term boolT (boolT/*/boolT)
| orB: term boolT (boolT/*/boolT)
| lookup: Loc → term Z unit
| update: Loc → term unit Z
| tag: EName → term empty_set unit
| untag: EName → term unit empty_set.

Infix "o" := comp (at level 70).
```

Some Derived Terms: Decorated Logic

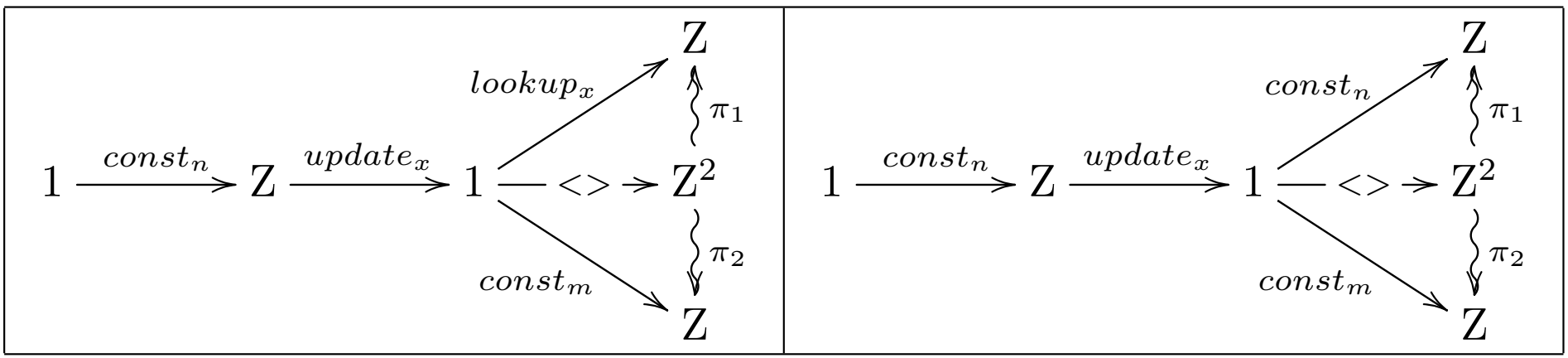
```
Definition ttrue: term boolT unit := (@copi1 unit unit).
Definition ffalse: term boolT unit := (@copi2 unit unit).

Definition throw (t1: Type) (t2: EName) := (@empty t1) o tag t2.

Definition TRY_CATCH (X Y: Type) (t: EName) (f: term Y X)
                                 (g: term Y unit)
 := downcast((copair (@id Y) (g o (untag t))) o iso_exc o f).
```

IMP Comparison & Conditionals over Decorated Logic - I

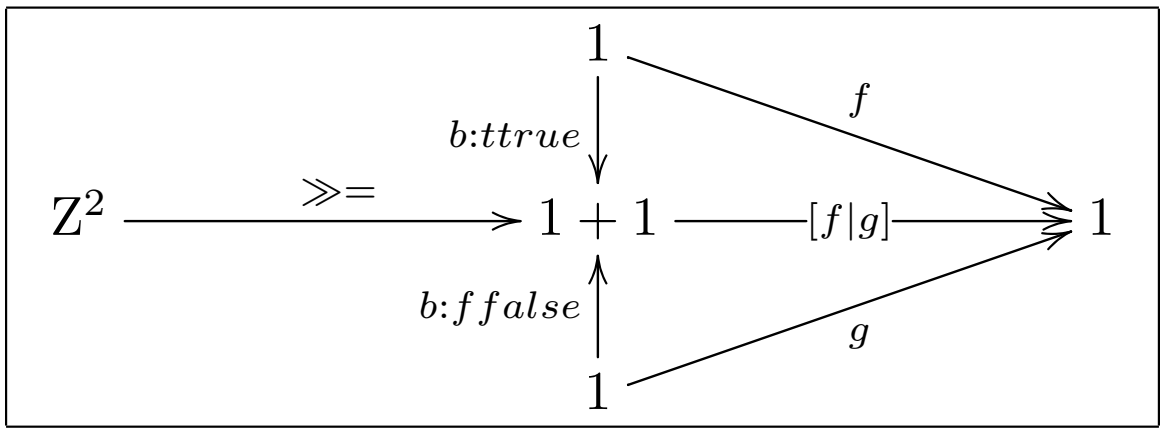
```
{{x ::= (const n) ;;
  IFB ((loc x) >>= (const m)) ...}}
```



Left hand side diagram is **strongly equal** to the right hand side one.

IMP Comparison & Conditionals over Decorated Logic - II

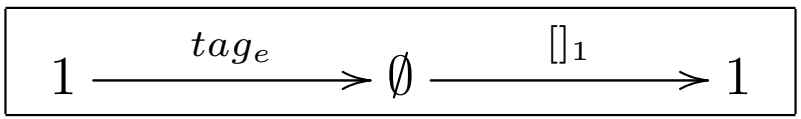
```
{{x ::= (const n) ;;
  IFB ((loc x) >>= (const m)) THEN (f) ELSE (g) ENDIF}}
```



If `b` is `ttrue` then `f` else `g` is executed.

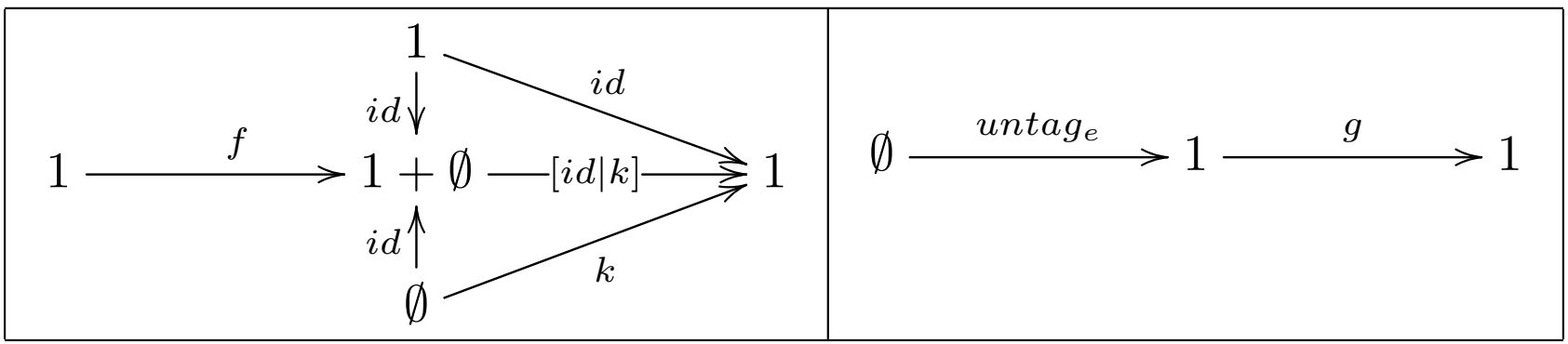
{IMP + Exc}: `throw` structure & `try-catch` block

{{THROW e}}.



`throw`ing an exception of name `e`.

{{TRY(f) CATCH e ⇒ g}}.



`try f`, if it `throw`s an exception of name `e` then it is caught inside the `catch` block and `g` is executed.