Compositional Non-termination without Fuel

Gregory Malecha
BedRock Systems

Outline

1. **Non-termination in Gallina**

2. Semantics with Interaction Trees
   a. Infinite loops
   b. Linking

Non-Termination in Gallina

```
(* computations that can diverge [1] *)
CoInductive thunk (A : Type) : Type :=
| Answer : A -> thunk A
| Think : thunk A -> thunk A.

(* loop forever *)
CoFixpoint loop : thunk Empty_set :=
  Think loop.

(* find an element in a stream *)
CoFixpoint find (p : nat -> bool)
                (ls : stream nat)
: thunk nat :=
  match ls with
  | scons l ls =>
    if p
    then l
    else Think (find p ls)
  end.
```

[1] A. Chlipala. Certified Programming with Dependent Types

Guarded Definitions & An Attempt at Writing Fix

```
CoFixpoint mfix_attempt1 {a b}
  (f : (a -> thunk b) -> a -> thunk b)
: a -> thunk b :=
  fun x => f (mfix f) x.
(* not guarded *)
```

```
CoFixpoint mfix_attempt2 {a b}
  (f : (a -> thunk b) -> a -> thunk b)
: a -> thunk b :=
  fun x => Think (f (mfix f) x).
(* not guarded *)
```

Outline

1. Non-termination in Gallina

2. **Semantics with Interaction Trees**
   a. **Infinite loops**
   b. **Linking**

Denotational Semantics

```
(* syntax of expressions *)
Inductive expr : Set :=
| Var (_ : var)
| Lit (_ : Z)
| Plus (_ _ : expr).


(* meaning of expressions *)
Fixpoint denoteExpr (e : expr)
: locals -> option Z :=
  fun l =>
    match e with
    | Var v => tryLookup v l
    | Lit v => Some v
    | Plus a b =>
      match denoteExpr a l
          , denoteExpr b l with
      | Some l , Some r => Some (l + r)
      | _ , _ => None
      end
    end.
```

Denotational Semantics with Interaction Trees

```
Inductive stmt : Set :=
| Assign (x : var) (e : expr)
| Seq    (a b : stmt)
| While  (t : expr) (b : stmt)

Fixpoint denoteStmt (s : stmt)
: locals -> optionT (itree empty) locals :=
  fun l =>
    match s with
    | Assign x e => v <- denoteExpr e l ;;
                    ret (set x v l)
    | Seq a b => l' <- denoteStmt a l ;;
                 denoteStmt b l'
    | While t b =>
      while (fun l =>
               t <- denoteExpr t l ;;
               if t
               then l' <- denoteStmt b l ;;
                    ret (true, l')
               else ret (false, l))
    end.
```

Current Work

- Reasoning principles
  - Equivalence up to *finite* stuttering
- Other semantics & uses
  - LLVM -- Steve Z.
  - DeepSpec Webserver -- Benjamin P.
- Co-inductive reasoning is not well supported in Coq. Some possibilities,
  - Paco
  - Only reason about finite approximations

References

[1] https://gmalecha.github.io/reflections/2018/compositional-coinductive-recursion-in-coq

[2] Interaction Trees Repository. https://github.com/DeepSpec/InteractionTrees

[3] C McBride. Turing Completeness Totally-Free.
https://personal.cis.strath.ac.uk/conor.mcbride/TotallyFree.pdf