To refutational completeness through non-termination test

Dmitri Rozplokhas

Saint-Petersburg Academic University

Relational programming

Relational programming is a technic of writing programs as relations. It allows using relational programs in unexpected ways.

List sorting function	as	List permutations generator
Type checker	as	Type inferencer or Type inhabitation problem solver
Interpreter	as	Generator of programs by tests

MiniKanren is a family of embedded relational DSLs.
Host languages include:

- **Scheme (original implementation)**
- Closure
- Haskell
- Go
- **OCaml (used implementation)**

Problem

Complete interleaving search: all existing solutions will be found.
But search can diverge, when no solution exist.

Goal

Make it easier to write refutationally complete relations, i.e. programs that always terminate whenever there are finitely many answers.

Possible approach

1. Advanced technics of writing relations, such as bounding the sizes of terms
2. Simulation of commutative conjunction evaluation
3. **Reordering of conjuncts during evaluation**

Our approach

Idea: testing current search proccess for divergence and try different order if it was detected.
This optimization is

- online
- non-intrusive
- conservative

Language

Relations in MiniKanren are represented by **goals** that can be built with the aid of a few simple combinators.

- $t_1 \equiv t_2$ unification of two terms
- $g_1 \wedge g_2$ conjunction of two goals
- $g_1 \vee g_2$ disjunction of two goals
- `fresh` (x) g introduction of fresh logic variable
- $r^n\ t_1 \ldots t_n$ call for previously defined relation

Examples

Good example to feel the problem is relational list sorting

```
let minmax^o a b min max =
  (min ≡ a ∧ max ≡ b ∧ le^o a b) ∨
  (min ≡ b ∧ max ≡ a ∧ gt^o a b)

let rec smallest^o l s l' =
  (l ≡ [s] ∧ l' ≡ []) ∨
  (fresh (h t s' t' max) (
    (l ≡ h :: t) ∧
    (l' ≡ max :: t') ∧
    (minmax^o h s' s max) ∧
    (smallest^o t s' t')
  ))

let rec sort^o =
  (xs ≡ [] ∧ ys ≡ []) ∨
  (fresh (s xs' ys') (
    (ys ≡ s :: ys') ∧
    (smallest^o xs s xs') ∧
    (sort^o xs' ys')
      ))
```

We can use this definition to actually sort a list

```
run 1 (λ q → sort^o [3; 2; 1] q) ⇝ {q=[1; 2; 3]}
```

or to find permutations of given sorted list

```
run 2 (λ q → sort^o q [1; 2]) ⇝ {q=[1; 2]; q=[2; 1]}
```

We also can request all answers, not fixed number of them.

```
run * (λ q → sort^o [3; 2; 1] q) ⇝ {q=[1; 2; 3]}

run * (λ q → sort^o q [1; 2])
```

But for the last query the program will diverge.
We can swap last two conjuncts in definition to fix it, but then previous query will lead to divergence.
The reason is **non-comutativity of conjunction**.
Fuction call get only information recieved by this point, wich can lead to divergence.
In this case, non-termination test will signal divergence in less informative call and optimized search will pull out more informative call ahead, making this sorting relation refutationally complete.
This automatic order choice gives opportunity to write easier definitions, as if conjunction was commutative, and still get terminating programs.
For example, consider the definition of division with reminder for binary numbers from MiniKanren standart library.

```
let rec div^o n m q r =
  (r ≡ n ∧ [] ≡ q ∧ plus^o r m n ∧lt^o r m) ∨
  ([1] ≡ q ∧ eql^o n m ∧plus^o r m n ∧lt^o r m) ∨
  ((ltl^o m n) ∧(lt^o r m) ∧(pos^o q) ∧
   (fresh (nh nl qh ql qlm qlmr rr rh) (
     (split^o n r nl nh) ∧
     (split^o q r ql qh) ∧
     ((([] ≡ nh) ∧ ([] ≡ qh) ∧
       (minus^o nl r qlm) ∧(mult^o ql m qlm)) ∨
      ((pos^o nh) ∧(mult^o ql m qlm) ∧
       (plus^o qlm r qlmr) ∧(minus^o qlmr nl rr) ∧
       (split^o rr r [] rh) ∧(div^o nh m qh rh))
     )
   ))
  )
```

The technic of bounding term sizes is used here to achieve refutational completeness. It's quite tricky and takes a bit of inspiration to write.
In contrast, optimized search provides much more straightforward variant.

```
let div^o n m q r =
  fresh (qm) (
    (pos^o m) ∧(pos^o qm) ∧
    (lt^o r m) ∧(mult^o q m qm) ∧
    (plus^o qm r n)
  )
```

MiniKanren semantics

Non-termination test

Implementation details?

Results?

References?

acknowledgements?