???

???

???

Relational programming

- Construction of functions as relations.
- Allows execution in various directions (e.g. get arguments by result value).
- Provides elegant solutions to non-trivial problems.

Sorting function	for	**generation of permutations**
Type checker	for	**type inhabitation problem**
Interpreter	for	**generation of quines**

MiniKanren

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

- Scheme (original implementation)
- Closure
- Haskell
- Go
- OCaml (implementation we use)

Specification constructors:

- unification ($\equiv$) of two terms
- conjunction ($\wedge$) and disjunction ($\vee$)
- fresh variable introduction

Example 1

Relational specification of list concatenation.

```
let rec append^o x y xy =
  (x ≡ [] ∧ y ≡ xy) ∨
  (fresh (h t ty) (
    (x ≡ h :: t) ∧
    (append^o t y ty) ∧
    (xy ≡ h :: ty)
  ))
```

Problem

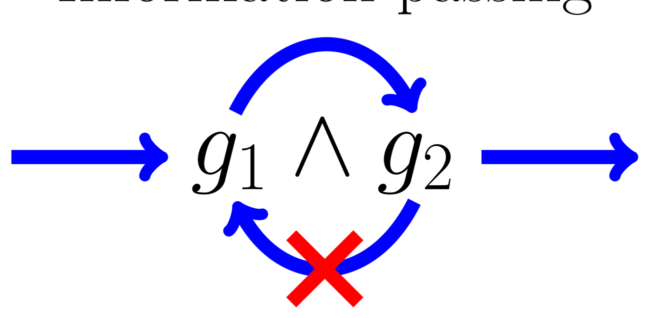
☺ Interleaving search is guaranteed to find all answers.
☹ But it can diverge when no answers left.

```
(λ q → append^o [1; 2] [3] q) ⤳ {q=[1; 2; 3]}

(λ q r → append^o q r [1; 2; 3]) ⤳ ⊥
```

Reason: non-commutativity of conjunction.

Information passing



Search in g_1 diverges $\Rightarrow$ search in conjunction diverges

Specification is **Refutationally Complete**,
if search by it always terminates when no answers left.
Shift of recursive call to the end makes **append**^{o} RC.
It doesn't work in more complex cases.

Goal

Make it easier to write refutationally complete specifications.

Possible solutions

1. Advanced technics of writing specifications, such as bounding the sizes of terms
2. Simulation of commutative conjunction
3. Reordering of conjuncts during execution ⟵ our approach

Conjuncts reordering approach

Idea:

1. testing current search proccess for divergence
2. try different order if it was detected

This search extension is:

- online
- non-intrusive
- conservative

Examples

Example 2

In relational sorting

```
let rec sort^o xs ys=
  (xs ≡ [] ∧ ys ≡ []) ∨
  (fresh (s xst yst) (
    (ys ≡ s :: yst) ∧
    (smallest^o xs s xst) ∧  (*1*)
    (sort^o xst yst)          (*2*)
  ))
```

for termination we need

- order $1 \rightarrow 2$ for direct execution
- order $2 \rightarrow 1$ for reverse execution

Permutation relstion based on it

```
let rec perm^o xs ys =
  (fresh (ts) (
    (sort^o xs ts) ∧ (sort^o ys ts)
  ))
```

requires execution in both directions.

Original search diverges with any order
+ search is too inefficient on lengths more than 3.

Extended search reconstructs (experimentally) the way of information propagation for each call.
Therefore multidirectional calls (like in **perm**^{o}) converge.

This aproach allows to write RC specifications naively.

Example 3

For division with remainder in binary arithmetics:

$$n = m \cdot q + r, \quad r < m$$

instead of this sophisticated solution

```
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))
     )
   ))
  )
```

we can just write down definition

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

and it will be RC under extended search.

MiniKanren semantics

Non-termination test

Implementation details?

References?

Acknowledgements?