Typed Embedding of a Relational Language in OCaml

Dmitrii Kosarev, Dmitrii Boulytchev

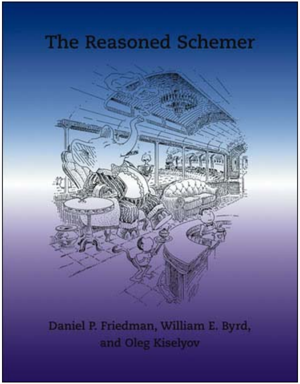
Saint-Petersburg State University
JetBrains Research

ML Family Workshop
September 22, 2016
Nara, Japan

Relational Programming in miniKanren

From programs as *functions* to programs as *relations*:

$$f : X \to Y \rightsquigarrow f^o \subseteq X \times Y$$



- Daniel P. Friedman, William Byrd and Oleg Kiselyov. *The Reasoned Schemer*, The MIT Press, Cambridge, MA, 2005
- A DSL for Scheme/Racket with rather simple minimal implementation
- A family of languages (μKanren, α-Kanren, cKanren etc.)
- Implemented as DSL for a wide range of host languages (including OCaml, Haskell, Scala etc.)

An Example: Relational List Append

$\texttt{append}: \alpha\,\texttt{list} \rightarrow \alpha\,\texttt{list} \rightarrow \alpha\,\texttt{list}$

$\texttt{append}^o \subseteq \alpha\,\texttt{list} \times \alpha\,\texttt{list} \times \alpha\,\texttt{list}$

An Example: Relational List Append

$\texttt{append}: \alpha\,\texttt{list} \to \alpha\,\texttt{list} \to \alpha\,\texttt{list}$

$\texttt{append}^o \subseteq \alpha\,\texttt{list} \times \alpha\,\texttt{list} \times \alpha\,\texttt{list}$

```
let rec append xs ys
```

An Example: Relational List Append

$\texttt{append}: \alpha\,\texttt{list} \to \alpha\,\texttt{list} \to \alpha\,\texttt{list}$

$\texttt{append}^o \subseteq \alpha\,\texttt{list} \times \alpha\,\texttt{list} \times \alpha\,\texttt{list}$

```
let rec append xs ys =
  match xs with
```

An Example: Relational List Append

$\texttt{append}: \alpha\,\texttt{list} \to \alpha\,\texttt{list} \to \alpha\,\texttt{list}$

$\texttt{append}^o \subseteq \alpha\,\texttt{list} \times \alpha\,\texttt{list} \times \alpha\,\texttt{list}$

```
let rec append xs ys =
  match xs with
  | []    → ys
```

An Example: Relational List Append

$\texttt{append}: \alpha\, \texttt{list} \to \alpha\, \texttt{list} \to \alpha\, \texttt{list}$

$\texttt{append}^o \subseteq \alpha\, \texttt{list} \times \alpha\, \texttt{list} \times \alpha\, \texttt{list}$

```
let rec append xs ys =
  match xs with
  | []     → ys
  | h::tl → h :: (append tl ys)
```

An Example: Relational List Append

append: α list $\to$ α list $\to$ α list

append^o $\subseteq$ α list $\times$ α list $\times$ α list

```
let rec append^o xs ys xys
```

```
let rec append xs ys =
  match xs with
  | []     → ys
  | h::tl → h :: (append tl ys)
```

An Example: Relational List Append

$\texttt{append}: \alpha\,\texttt{list} \to \alpha\,\texttt{list} \to \alpha\,\texttt{list}$

```
let rec append xs ys =
  match xs with
  | []    → ys
  | h::tl → h :: (append tl ys)
```

$\texttt{append}^o \subseteq \alpha\,\texttt{list} \times \alpha\,\texttt{list} \times \alpha\,\texttt{list}$

```
let rec appendᵒ xs ys xys =
  ((xs ≡ nil) &&& (xys ≡ ys))
```

An Example: Relational List Append

$\texttt{append}: \alpha\,\texttt{list} \rightarrow \alpha\,\texttt{list} \rightarrow \alpha\,\texttt{list}$

```
let rec append xs ys =
  match xs with
  | []    → ys
  | h::tl → h :: (append tl ys)
```

$\texttt{append}^o \subseteq \alpha\,\texttt{list} \times \alpha\,\texttt{list} \times \alpha\,\texttt{list}$

```
let rec appendᵒ xs ys xys =
  ((xs ≡ nil) &&& (xys ≡ ys)) |||
  (fresh (h t tys)
```

An Example: Relational List Append

$\texttt{append}: \alpha\,\texttt{list} \to \alpha\,\texttt{list} \to \alpha\,\texttt{list}$

```
let rec append xs ys =
  match xs with
  | []    → ys
  | h::tl → h :: (append tl ys)
```

$\texttt{append}^o \subseteq \alpha\,\texttt{list} \times \alpha\,\texttt{list} \times \alpha\,\texttt{list}$

```
let rec appendᵒ xs ys xys =
  ((xs ≡ nil) &&& (xys ≡ ys)) |||
  (fresh (h t tys)
     (xs  ≡ h % t)
```

An Example: Relational List Append

$\texttt{append}: \alpha\,\texttt{list} \to \alpha\,\texttt{list} \to \alpha\,\texttt{list}$

$\texttt{append}^o \subseteq \alpha\,\texttt{list} \times \alpha\,\texttt{list} \times \alpha\,\texttt{list}$

```
let rec append xs ys =
  match xs with
  | []    →  ys
  | h::tl →  h :: (append tl ys)
```

```
let rec appendᵒ xs ys xys =
  ((xs ≡ nil) &&& (xys ≡ ys))  |||
  (fresh (h t tys)
     (xs  ≡ h % t)
     (xys ≡ h % tys)
```

An Example: Relational List Append

$\texttt{append}: \alpha\,\texttt{list} \to \alpha\,\texttt{list} \to \alpha\,\texttt{list}$

```
let rec append xs ys =
  match xs with
  | []    → ys
  | h::tl → h :: (append tl ys)
```

$\texttt{append}^o \subseteq \alpha\,\texttt{list} \times \alpha\,\texttt{list} \times \alpha\,\texttt{list}$

```
let rec appendᵒ xs ys xys =
  ((xs ≡ nil) &&& (xys ≡ ys)) |||
  (fresh (h t tys)
     (xs  ≡ h % t)
     (xys ≡ h % tys)
     (appendᵒ t ys tys)
  )
```

An Example: Relational List Append

$\texttt{append}: \alpha\,\texttt{list} \to \alpha\,\texttt{list} \to \alpha\,\texttt{list}$

```
let rec append xs ys =
  match xs with
  | []    →  ys
  | h::tl →  h :: (append tl ys)
```

$\texttt{append}^o \subseteq \alpha\,\texttt{list} \times \alpha\,\texttt{list} \times \alpha\,\texttt{list}$

```
let rec appendᵒ xs ys xys =
  ((xs ≡ nil) &&& (xys ≡ ys))  |||
  (fresh (h t tys)
     (xs  ≡ h % t)
     (xys ≡ h % tys)
     (appendᵒ t ys tys)
  )
```

```
(define (appendᵒ xs ys xys)
   (conde
      [(≡ '() xs) (≡ ys xys)]
      [(fresh (h t tys)
         (≡ `(,h . ,t) xs)
         (≡ `(,h . ,tys) xys)
         (appendᵒ t ys tys))]))
```

Implementation Sketch

Jason Hemann, Daniel P. Friedman. *μKanren: A Minimal Functional Core for Relational Programming* // Scheme'13:

- Logic variables: $X = \{x_1, x_2, \ldots\}$;
- Symbols (constructors): $S = \{s_1, s_2, \ldots\}$;
- Terms: $T = X \cup \{s\ (t_1, \ldots, t_k) \mid s \in S,\ t_i \in T\}$;
- Substitutions: $\Sigma = T^X$;
- Unification: $(\equiv) : \Sigma \to T \to T \to \Sigma_\perp$;
- State: a substitution + some info to create fresh variables;
- Goal:

Current Implementation

- Repository: `https://github.com/dboulytchev/OCanren`
- Implements μKanren + disequality constraints
- Passes most of the orignal tests
- Outperforms μKanren on long queries