A pattern matcher for miniKanren

or

How to get into trouble with CPS macros

Andrew W. Keep, Michael D. Adams, Lindsey Kuper,
William E. Byrd, Daniel P. Friedman

INDIANA UNIVERSITY

SCHOOL OF INFORMATICS AND COMPUTING

Bloomington

Introduction

- Writing a pattern matcher for miniKanren
  - Automatically bind variables from pattern
- A CPS macro implementation
  - Delayed binding and conditional expansion
  - Auxiliary keyword comparison
- Two fixes for the CPS macro
  - `bound-identifier=?` with **`syntax-case`**
  - Eager binding

INDIANA UNIVERSITY
SCHOOL OF INFORMATICS AND COMPUTING
Bloomington

2

miniKanren

- A relational programming language
- Embedded in Scheme
- Similar to Prolog, different approach
  - Interleaved search
  - Lexical scoping of variables

3

CPS Macros

- Stay within **`syntax-rules`**
  - Simpler semantics
  - Provides powerful macro facility
- Control expansion order
- Allows for conditional expansion
  - Some caution may be required

4

Pattern matcher goals

- Simplify miniKanren relations
  - Provide simple pattern matching
  - Automatically create variables in pattern
- Generate relations with good performance
  - "Fail fast" philosophy, tight variable scope
  - Avoid creating unnecessary binding forms

5

An Example: append

```
append([],Y,Y).
append([A|D],Y2,[A|R])
  :- append(D,Y2,R).
```


6

An Example: append

```
(define append
  (λ (x y z)
    (conde
      ((≡ `() x) (≡ y z))
      ((exist (a d r)
         (≡ `(,a . ,d) x)
         (≡ `(,a . ,r) z)
         (append d y r))))))
```


6

An Example: append

```
(define append
  (λᵉ (x y z)
    ((() __ ,y))
    (((,a . ,d) __ (,a . ,r))
     (append d y r))))
```


6

An Example: append

```
(define append
  (λᵉ (x y z)
    ((() __ ,y))
    (((,a . ,d) __ (,a . ,r))
     (append d y r))))
```

```
append([],Y,Y).
append([A|D],Y2,[A|R])
  :- append(D,Y2,R).
```


6

Original Algorithm

- λᵉ and matchᵉ
- handle-clauses
- handle-pattern
- build-cons
- build-goal
- build-var
- build-clauses-part
- build-clause
- make-clauses-cont
- make-pattern-cont
- case-id

7

Original algorithm

- CPS macro controls expansion order
- Binding delayed until end of expansion
- Unbound identifiers compared
  - free, symbolically-equal identifiers appear equal
  - Otherwise equal identifiers introduced in different lexical scopes left unbound

8

Original algorithm

```
(define append
  (λᵉ (x y z)
    ((() __ ,y))
     (((,a . ,d) __ (,a . ,r))
      (append d y r)))))
```


9

Original algorithm

```
(define append
  (λe (x y z)
    ((() ___ ,y))
    (((,a . ,d) __ (,a . ,r))
     (append d y r))))
```

```
((ex a d) (≡ (cons a d) x))
```


9

Original algorithm

```
(define append
  (λe (x y z)
    ((() __ ,y))
    (((,a . ,d) ,d) (,a . ,r))
     (append d y r))))
```

```
((ex a d) (≡ (cons a d) x))
```


9

Original algorithm

```
(define append
  (λᵉ (x y z)
    ((() __ ,y))
    (((,a . ,d) __ (,a . ,r))
     (append d y r))))
```

```
((ex a d) (≡ (cons a d) x)
 (ex r) (≡ (cons a r) z))
```


9

Original algorithm

```
(define append
  (λᵉ (x y z)
    ((() __ ,y))
    (((,a . ,d) __ (,a . ,r))
     (append d y r))))
```

```
(exist (a d)
  (≡ (cons a d) x)
  (exist (r)
    (≡ (cons a r) z)
    (append d y r)))
```


9

Original Algorithm

```
(define append
  (λ (x y z)
    (conde
      ((≡ '() x) (≡ y z))
      ((exist (a d)
         (≡ (cons a d) x)
         (exist (r)
           (≡ (cons a r) z)
           (append d y r)))))))
```


10

Original Algorithm

```
(define append
  (λ (x y z)
    (conde
      ((≡ `() x) (≡ y z))
      ((exist (a d r)
         (≡ `(,a . ,d) x)
         (≡ `(,a . ,r) z)
         (append d y r))))))
```

```
(define append
  (λ (x y z)
    (conde
      ((≡ '() x) (≡ y z))
      ((exist (a d)
         (≡ (cons a d) x)
         (exist (r)
           (≡ (cons a r) z)
           (append d y r)))))))
```


10

Exposing our bug

```
(define-syntax break-λe
  (syntax-rules ()
    ((_ v)
     (λe (x y)(((,w . ,v) ,v))))))

(break-λe z) ⇒
(λe (x y) (((,w . ,z) ,z))) ⇒
(λ (x y)
 (conde
   ((exist (z w)
      (== (cons w z) x)
      (== z y)))))
```

11

Exposing our bug

```
(define-syntax break-λe
  (syntax-rules ()
    ((_ v)
     (λe (x y) (((,w . ,v) ,v))))))

(break-λe x) ⇒
(λe (x y) (((,w . ,x) ,x)))  ⇒
(λ (x y)
  (condᵉ
    ((exist (x w)
       (== (cons w x) x)
       (== x y)))))
```

Exposing our bug

```
(define-syntax break-λe
  (syntax-rules ()
    ((_ v)
     (λe (x y)(((,w . ,v) ,v))))))

(break-λe w) ⇒
(λe (x y) (((,w . ,w) ,w)))⇒
(λ (x y)
 (conde
   ((exist (w)
      (== (cons w w) x)
      (== w y))))))
```

Source of our bug

```
(define-syntax case-id
  (syntax-rules (else)
    ((_ x ((x** ...) act*) ... (else e-act))
     (letrec-syntax
       ((helper
          (syntax-rules (x else)
            ((_ (else a)) a)
            ((_ (() a) c . c*) (helper c . c*))
            ((_ ((x . z*) a) c . c*) a)
            ((_ ((y z* (... ...)) a) c . c*)
             (helper ((z* (... ...)) a) c . c*)))))
       (helper
         ((x** ...) act*) ... (else e-act))))))
```


12

Source of our bug

```
(define-syntax case-id
  (syntax-rules (else)
    ((_ x ((x** ...) act*) ... (else e-act))
     (letrec-syntax
       ((helper
          (syntax-rules (x else)
            ((_ (else a)) a)
            ((_ (() a) c . c*) (helper c . c*))
            ((_ ((x . z*) a) c . c*) a)
            ((_ ((y z* (... ...)) a) c . c*)
             (helper ((z* (... ...)) a) c . c*)))))
       (helper
         ((x** ...) act*) ... (else e-act))))))
```


12

Source of our bug

```
(define-syntax case-id
  (syntax-rules (else)
    ((_ x ((x** ...) act*) ... (else e-act))
     (letrec-syntax
       ((helper
          (syntax-rules (x else)
            ((_ (else a)) a)
            ((_ (() a) c . c*) (helper c . c*))
            ((_ ((x . z*) a) c . c*) a)
            ((_ ((y z* (... ...)) a) c . c*)
             (helper ((z* (... ...)) a) c . c*)))))
       (helper
         ((x** ...) act*) ... (else e-act))))))
```


12

Source of our bug

- Auxiliary keyword check is similar to `free-identifier=?`
  - Delayed binding leaves variables free
- Free, symbolically-equal identifiers may be conflated

13

`bound-identifier=?` **to the rescue**

- Identifiers compared as if they were bound
  - Free, symbolically-equal variables introduced in different lexical scopes can be distinguished
- Needs **`syntax-case`**
  - More then term-rewriting
  - Needs Scheme conditionals to be useful

14

case-id redux

```
(define-syntax case-id
  (λ (exp)
    (syntax-case exp (else)
      ((_ x (else e-act)) #'e-act)
      ((_ x ((y x* ...) act) ((x** ...) act*) ...
            (else e-act))
       (bound-identifier=? #'x #'y)
       #'act)
      ((_ x ((y x* ...) act) ((x** ...) act*) ...
            (else e-act))
       #'(case-id x
           ((x* ...) act) ((x** ...) act*) ...
           (else e-act)))
      ((_ x (() act) ((x** ...) act*) ...
            (else e-act))
       #'(case-id x
           ((x** ...) act*) ... (else e-act))))))
```


15

Eager binding to the rescue

- Binds variables as they are encountered
  - Ensures at most one variable free in comparison
  - `syntax-rules` based `case-id` sufficient
- Requires rewrite of code handling identifier
  - Expression built as expansion happens
  - Binding forms can be recognized by expander

16

Eager binding to the rescue

- `λ^e` and `match^e`
- `handle-clauses`
- `do-clause`
- `do-pattern-opt`
- `do-pattern`
- `handle-pattern`
- `handle-pattern-cont`
- `exist-helper`

17

Comparing Output

```
(define append
  (λ (x y z)
    (conde
      ((≡ '() x) (≡ y z))
      ((exist (a d)
        (≡ (cons a d) x)
        (exist (r)
          (≡ (cons a r) z)
          (append d y r)))))))
```


18

Comparing Output

```
(define append
  (λ (x y z)
    (condᵉ
      ((exist () (≡ () x) (≡ y z)))
      ((exist (a)
         (exist (d)
           (exist ()
             (≡ (cons a d) x)
             (exist (r)
               (exist ()
                 (≡ (cons a r) z)
                 (append d y r)))))))))
```


18

Summary

- CPS macros powerful, require care
  - Limited to `free-identifier=?`
  - Conditional expansion on identifiers to be bound should be done with caution
- `bound-identifier=?` is a simple fix
- Possible to stay within **`syntax-rules`**
  - Larger job to rewrite for eager binding

19

Thanks

- My co-authors
  - Michael D. Adams
  - Lindsey Kuper
  - William E. Byrd
  - Daniel P. Friedman
- The anonymous reviewers

INDIANA UNIVERSITY
SCHOOL OF INFORMATICS AND COMPUTING
Bloomington

20

Questions?

21