An Empirical Study of Partial Deduction for miniKanren

Kate Verbitskaia, Daniil Berezun, Dmitry Boulytchev

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg State University

27.08.2020

Specialization: a Method to Improve Programs

input program

```
let rec evalᵒ fm s r =
  fm ≡ neg x & evalᵒ x s a & notᵒ a r |
  ...
```

Specialization: a Method to Improve Programs

input program

```
let rec evalᵒ fm s r =
  fm ≡ neg x & evalᵒ x s a & notᵒ a r |
  ...
```

known argument

```
evalᵒ fm s true
```

Specialization: a Method to Improve Programs

input program

let rec $\texttt{eval}^o$ `fm s r` =
  `fm` $\equiv$ `neg x` & $\texttt{eval}^o$ `x s a` & $\texttt{not}^o$ `a r` |
  ...

known argument

$\texttt{eval}^o$ `fm s` **true**

`fm` $\equiv$ `neg x` & $\texttt{eval}^o$ `x s a` & $\texttt{not}^o$ `a` **true** |
...

Specialization: a Method to Improve Programs

input program

let rec $\texttt{eval}^o$ fm s r =
  fm $\equiv$ neg x & $\texttt{eval}^o$ x s a & $\texttt{not}^o$ a r |
  ...

known argument

$\texttt{eval}^o$ fm s **true**

fm $\equiv$ neg x & $\texttt{eval}^o$ x s a & $\texttt{not}^o$ a **true** |
...

fm $\equiv$ neg x & $\texttt{eval}^o$ x s **false** |
...

Specialization: a Method to Improve Programs

input program

let rec $\texttt{eval}^o$ fm s r =
  fm ≡ neg x & $\texttt{eval}^o$ x s a & $\texttt{not}^o$ a r |
  . . .

known argument

$\texttt{eval}^o$ fm s **true**

fm ≡ neg x & $\texttt{eval}^o$ x s a & $\texttt{not}^o$ a **true** |
. . .

fm ≡ neg x & $\texttt{eval}^o$ x s **false** |
. . .

. . .

Specialization: a Method to Improve Programs

input program

```
let rec evalᵒ fm s r =
  fm ≡ neg x & evalᵒ x s a & notᵒ a r |
  ...
```

known argument

```
evalᵒ fm s true
```

```
fm ≡ neg x & evalᵒ x s a & notᵒ a true |
...
```

```
fm ≡ neg x & evalᵒ x s false |
...
```

...

output

```
let rec eval_trueᵒ fm s =
  fm ≡ neg x & eval_falseᵒ x s |
  ...

let rec eval_falseᵒ fm s =
  fm ≡ neg x & eval_trueᵒ x s |
  ...
```

Partial Deduction: Specialization for Logic Programming

input

```
let double_appendᵒ x y z r =
  ocanren {
    fresh t in
      appendᵒ x y t &
      appendᵒ t z r}

let rec appendᵒ x y r =
  ocanren {
    (x ≡ [] & y ≡ r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      appendᵒ x' y r' &
      r ≡ h :: r')}
```

```
double_appendᵒ x y z r
```

output

```
let double_appendᵒ x y z r =
  ocanren {
    (x ≡ [] & appendᵒ y z r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      double_appendᵒ x' y z r' &
      r ≡ h :: r')}

let rec appendᵒ x y r =
  ocanren {
    (x ≡ [] & y ≡ r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      appendᵒ x' y r' &
      r ≡ h :: r')}
```

Partial Deduction for miniKanren: Bird's-eye View

```
let rec append^o x y r =
  ocanren {
    (x ≡ [] & y ≡ r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      append^o x' y r' &
      r ≡ h :: r')}
```

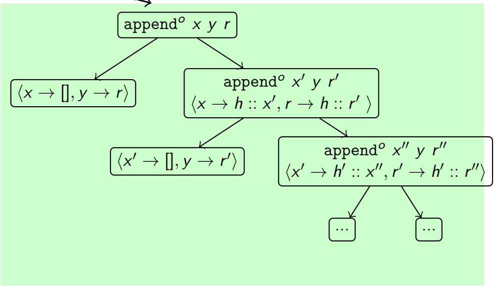
```
append^o x y r
```

Partial Deduction for miniKanren: Bird's-eye View

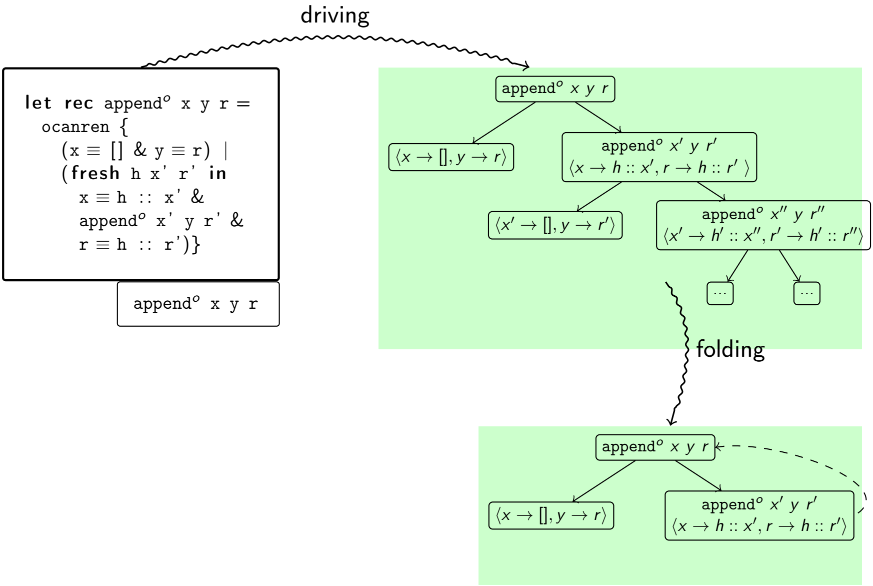
driving

```
let rec appendᵒ x y r =
  ocanren {
    (x ≡ [] & y ≡ r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      appendᵒ x' y r' &
      r ≡ h :: r')}
```

```
appendᵒ x y r
```



Partial Deduction for miniKanren: Bird's-eye View

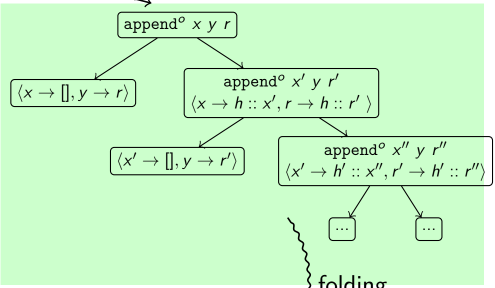


Partial Deduction for miniKanren: Bird's-eye View

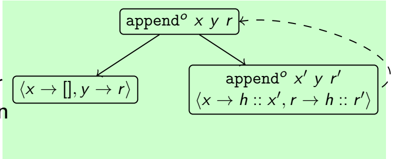
driving

```
let rec appendᵒ x y r =
  ocanren {
    (x ≡ [] & y ≡ r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      appendᵒ x' y r' &
      r ≡ h :: r')}
```

```
appendᵒ x y r
```



folding




```
let rec appendᵒ x y r =
  ocanren { fresh h t r' in
    (x ≡ [] & y ≡ r) |
    ( x ≡ h :: x' &
      appendᵒ x' y r' &
      t ≡ h :: r')}
```

residualization

Driving: Unfolding

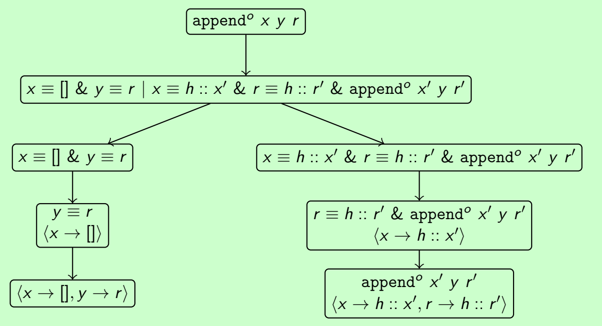
```
let rec append^o x y r =
  ocanren {
    (x ≡ [] & y ≡ r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      append^o x' y r' &
      r ≡ h :: r')}
```

```
append^o x y r
```

Driving: Unfolding

```
let rec appendᵒ x y r =
  ocanren {
    (x ≡ [] & y ≡ r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      appendᵒ x' y r' &
      r ≡ h :: r')}
```

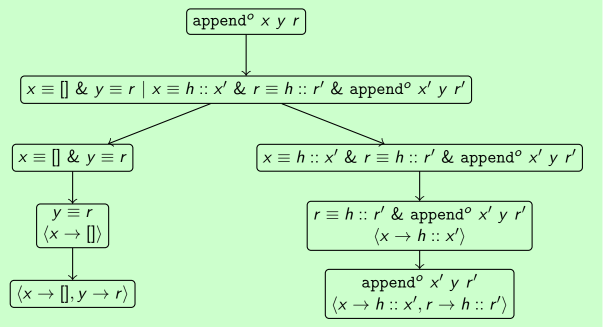
```
appendᵒ x y r
```

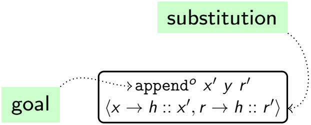


Driving: Unfolding

```
let rec appendᵒ x y r =
  ocanren {
    (x ≡ [] & y ≡ r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      appendᵒ x' y r' &
      r ≡ h :: r')}
```

```
appendᵒ x y r
```

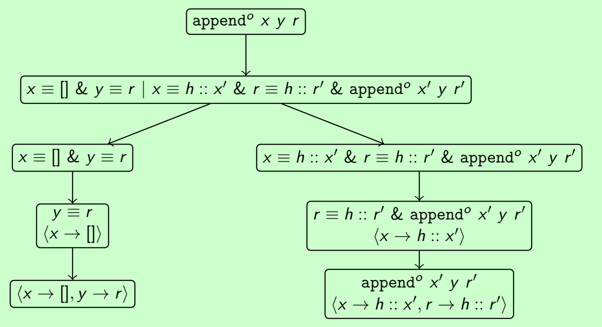


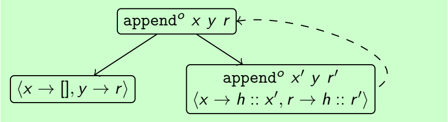


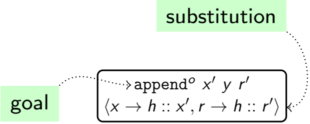
Driving: Unfolding

```
let rec appendᵒ x y r =
  ocanren {
    (x ≡ [] & y ≡ r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      appendᵒ x' y r' &
      r ≡ h :: r')}
```

appendᵒ x y r

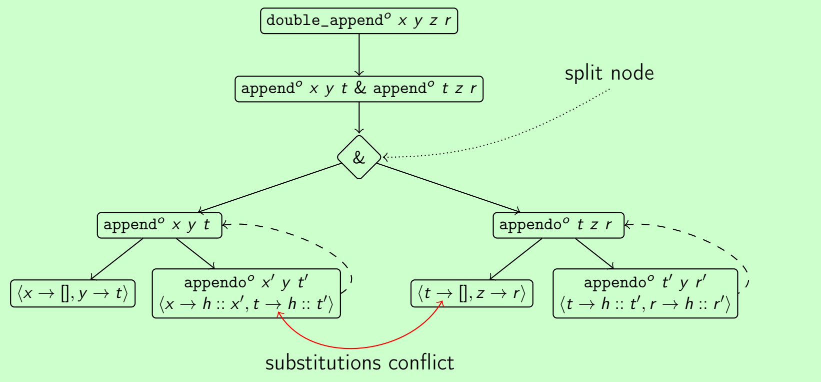






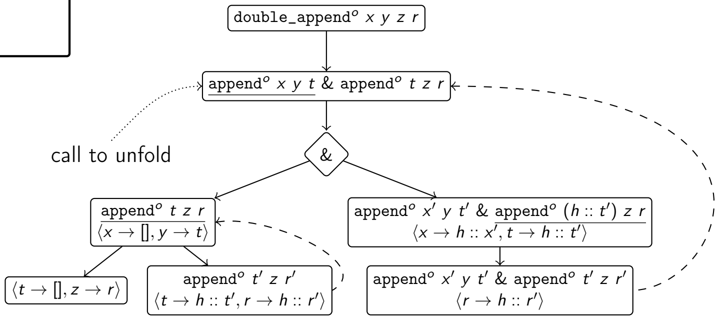
Partial Deduction

```
let double_appendᵒ x y z r =
  ocanren {
    fresh t in
      appendᵒ x y t &
      appendᵒ t z r}
```



Conjunctive Partial Deduction

```
let double_appendᵒ x y z r =
  ocanren {
    fresh t in
      appendᵒ x y t &
      appendᵒ t z r}
```

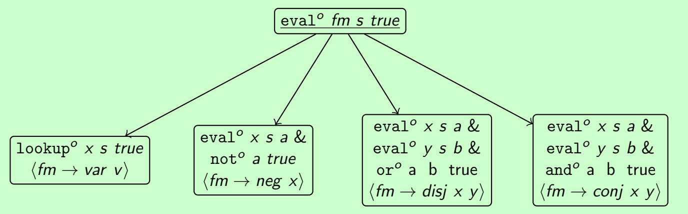



```
let double_appendᵒ x y z r =
  ocanren {
    (x ≡ [] & appendᵒ y z r) |
    (fresh h x' r' in
      x ≡ h :: x' &
      double_appendᵒ x' y z r' &
      r ≡ h :: r')}
```

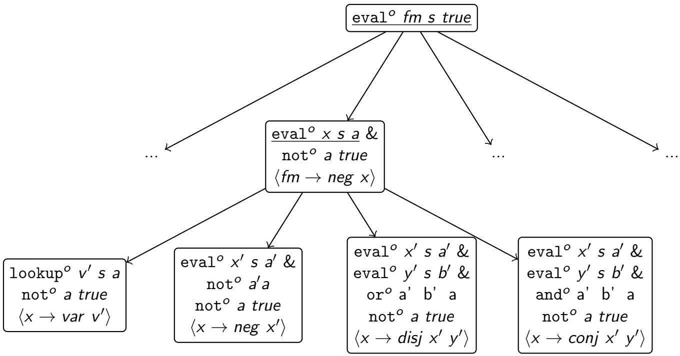
Evaluator of Logic Formulas

```
let rec eval^o fm s r =
  ocanren { fresh v x y a b in
    (fm ≡ var v & lookup^o v s r) |
    (fm ≡ neg x & eval^o x s a & not^o a r) |
    (fm ≡ conj x y & eval^o x s a & eval^o y s b & and^o a b r) |
    (fm ≡ disj x y & eval^o x s a & eval^o y s b & oro^o a b r)
```

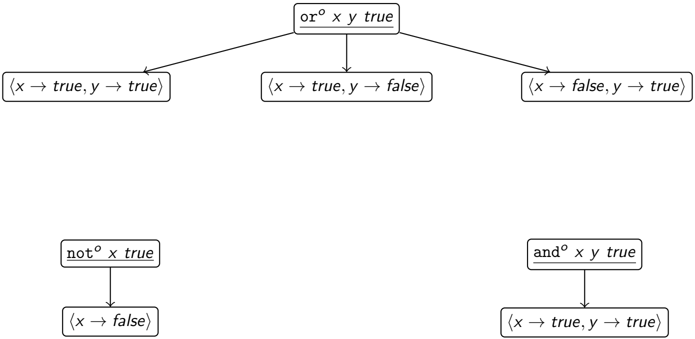
$\texttt{eval}^o\ \texttt{fm s}\ \textbf{true}$



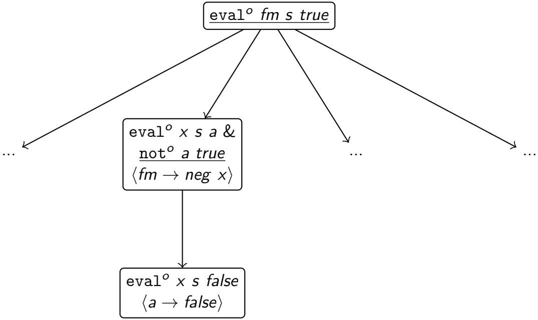
Evaluator of Logic Formulas: Unfolding 2



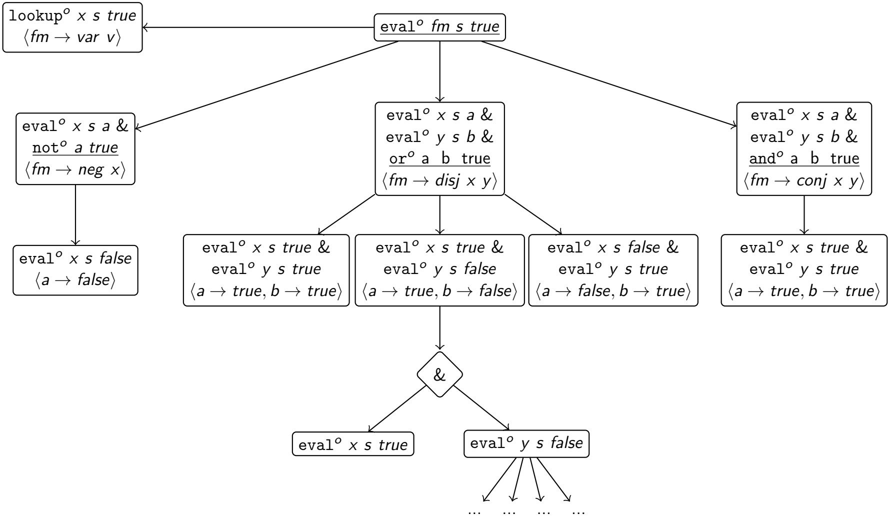
Boolean Connectives



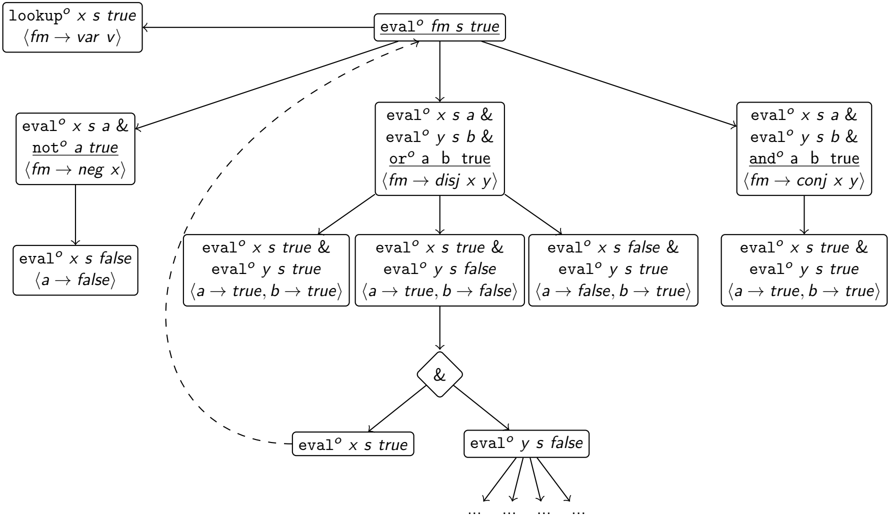
Evaluator of Logic Formulas: Unfolding 3



Evaluator of Logic Formulas: ConsPD



Evaluator of Logic Formulas: ConsPD



$\texttt{reverse}^o$

```
let rec reverseᵒ xs sx =
  ocanren {
    (xs ≡ [] & sx ≡ []) |
    (fresh h t t' in
      xs ≡ h :: t &
      reverseᵒ t t' &
      appendᵒ t' [h] sx}
```

$\texttt{reverse}^o\ \texttt{xs}\ \texttt{sx}$

$\underline{\texttt{reverse}^o\ xs\ sx}$

$\downarrow$

$\underline{\texttt{reverse}^o\ t\ t'}\ \&\ \texttt{append}^o\ t'\ [h]\ sx$
$\langle xs \rightarrow h :: t \rangle$

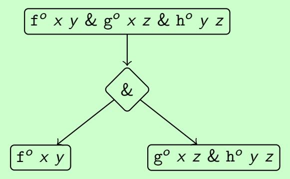
$\downarrow$

$\underline{\texttt{reverse}^o\ s\ s'}\ \&\ \texttt{append}^o\ s'\ [h']\ t'\ \&\ \texttt{append}^o\ t'\ [h]\ sx$
$\langle t \rightarrow h' :: s \rangle$

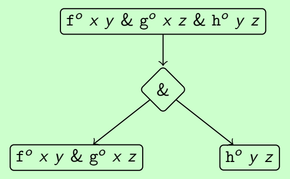
$\downarrow$

$\ldots$

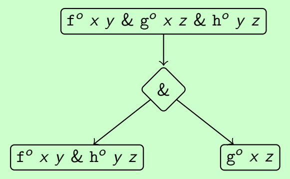
Split



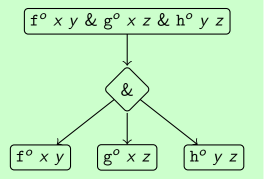
f^o x y & g^o x z & h^o y z
&
f^o x y
g^o x z & h^o y z



f^o x y & g^o x z & h^o y z
&
f^o x y & g^o x z
h^o y z



f^o x y & g^o x z & h^o y z
&
f^o x y & h^o y z
g^o x z

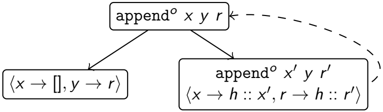


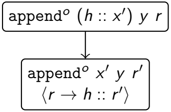
f^o x y & g^o x z & h^o y z
&
f^o x y
g^o x z
h^o y z

Conservative Partial Deduction

Branching heuristics is used to select a call to unfold

If the call has less branches in the process tree than the relation can possible have, unfold the call





Evaluation

Evaluator of Logic Formulas

Evaluator of Logic Formulas: Order of Calls

:

Evaluator of Logic Formulas: Compexity of Relations

Evaluator of Logic Formulas: Results

Unification

Path Search

Evaluation Results

	last	plain	unify	isPath
Original	1.06s	1.84s	—	—
CPD	—	1.13s	14.12s	3.62s
ConsPD	0.93s	0.99s	0.96s	2.51s
Branching	3.11s	7.53s	3.53s	0.54s

Table: Evaluation results

Conclusion

- Conservative Partial Deduction
  - Less-branching heuristics
- Evaluation shows some improvement, but not for every query
- Models to predict performance can help