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

Partial Deduction: a Method to Improve Logic Programs

input program

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

Partial Deduction: a Method to Improve Logic Programs

input program

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

known argument

```
evalᵒ fm s true
```

Partial Deduction: a Method to Improve Logic Programs

input program

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

known argument

```
evalᵒ fm s true
```

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

Partial Deduction: a Method to Improve Logic Programs

input program

```
let rec eval° fm s r =
  fm ≡ neg x & not° a r & eval° x s a |
  ...
```

known argument

```
eval° fm s true
```

```
fm ≡ neg x & not° a true & eval° x s a |
...
```

```
fm ≡ neg x & eval° x s false |
...
```

Partial Deduction: a Method to Improve Logic Programs

input program

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

known argument

```
evalᵒ fm s true
```

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

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

...

Partial Deduction: a Method to Improve Logic Programs

input program

```
let rec eval^o fm s r =
  fm ≡ neg x & not^o a r & eval^o x s a |
  ...
```

```
eval^o fm s true
```

known argument

```
fm ≡ neg x & not^o a true & eval^o x s a |
...
```

```
fm ≡ neg x & eval^o x s false |
...
```

...

output

```
let rec eval_true^o fm s =
  fm ≡ neg x & eval_false^o x s |
  ...

let rec eval_false^o fm s =
  fm ≡ neg x & eval_true^o x s |
  ...
```

Partial Deduction for miniKanren: Bird's-eye View

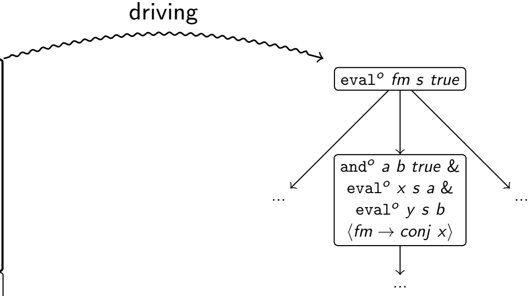
```
let rec evalᵒ fm s r =
  ...
  fm ≡ conj x y &
  andᵒ a b r &
  evalᵒ x s a &
  evalᵒ y s b |
  ...
```

```
evalᵒ fm s true
```

Partial Deduction for miniKanren: Bird's-eye View

```
let rec eval^o fm s r =
  ...
  fm ≡ conj x y &
  and^o a b r &
  eval^o x s a &
  eval^o y s b |
  ...
```

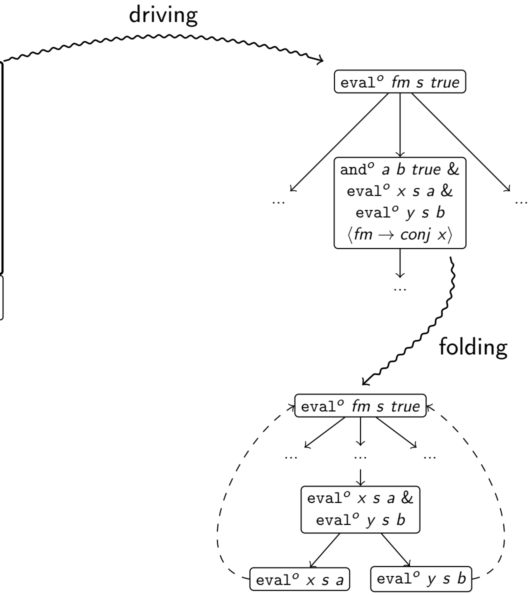
```
eval^o fm s true
```



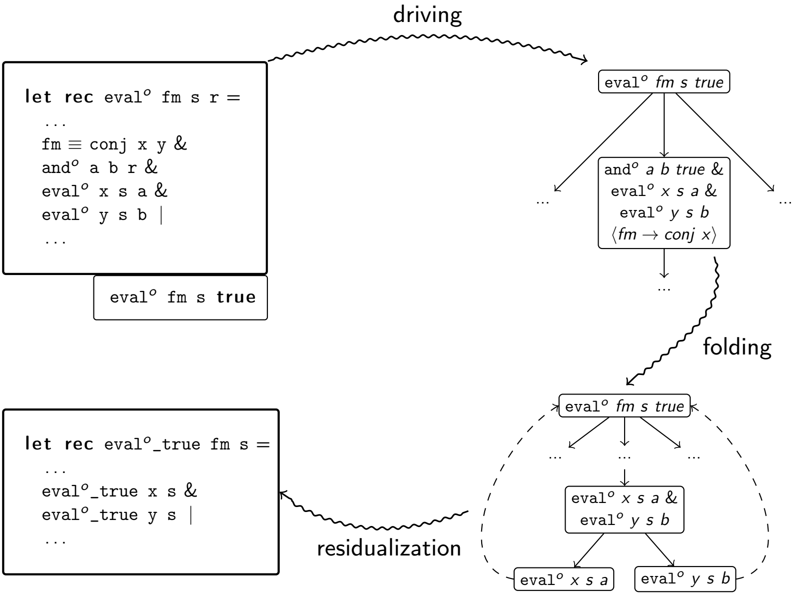
Partial Deduction for miniKanren: Bird's-eye View

```
let rec evalᵒ fm s r =
  ...
  fm ≡ conj x y &
  andᵒ a b r &
  evalᵒ x s a &
  evalᵒ y s b |
  ...
```

```
evalᵒ fm s true
```



Partial Deduction for miniKanren: Bird's-eye View



Driving: Unfolding

```
let rec evalᵒ fm s r =
  ...
  fm ≡ conj x y & andᵒ a b r &
  evalᵒ x s a & evalᵒ y s b |
  ...

let andᵒ x y r =
  ocanren {
    fresh xy in
      (nandᵒ x y xy & nandᵒ xy xy r) }

let rec nandᵒ x y r =
  ocanren {
    (x ≡ true & y ≡ true & r ≡ false) |
    (x ≡ true & y ≡ false & r ≡ true) |
    (x ≡ false & y ≡ true & r ≡ true) |
    (x ≡ false & y ≡ false & r ≡ true) }
```

```
evalᵒ fm s true
```

Driving: Unfolding

```
let rec evalᵒ fm s r =
  ...
  fm ≡ conj x y & andᵒ a b r &
  evalᵒ x s a & evalᵒ y s b |
  ...

let andᵒ x y r =
  ocanren {
    fresh xy in
      (nandᵒ x y xy & nandᵒ xy xy r) }

let rec nandᵒ x y r =
  ocanren {
    (x ≡ true & y ≡ true & r ≡ false) |
    (x ≡ true & y ≡ false & r ≡ true) |
    (x ≡ false & y ≡ true & r ≡ true) |
    (x ≡ false & y ≡ false & r ≡ true) }
```

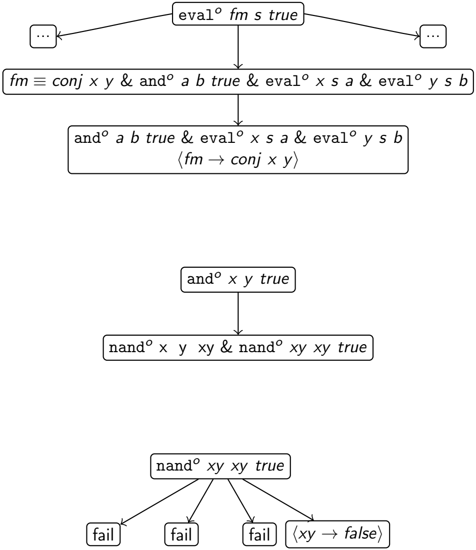
evalᵒ fm s true

goal

$\text{and}^o\ a\ b\ \mathit{true}\ \&\ \text{eval}^o\ x\ s\ a\ \&\ \text{eval}^o\ y\ s\ b$

$\langle fm \rightarrow conj\ x\ y \rangle$

substitution



Partial Deduction

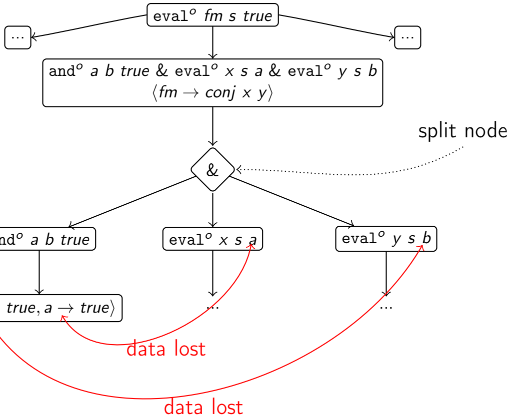
```
let rec evalᵒ fm s r =
  ...
  fm ≡ conj x y & andᵒ a b r &
  evalᵒ x s a & evalᵒ y s b |
  ...
```

```
evalᵒ fm s true
```

Partial Deduction

```
let rec eval^o fm s r =
  ...
  fm ≡ conj x y & and^o a b r &
  eval^o x s a & eval^o y s b |
  ...
```

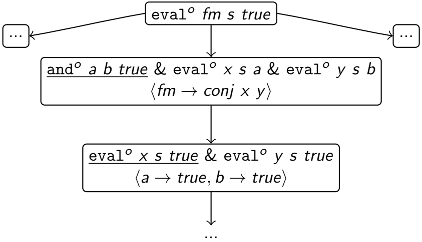
`eval^o fm s true`



Conjunctive Partial Deduction: Left-to-right Unfolding

```
let rec evalᵒ fm s r =
  ...
  fm ≡ conj x y & andᵒ a b r &
  evalᵒ x s a & evalᵒ y s b |
  ...
```

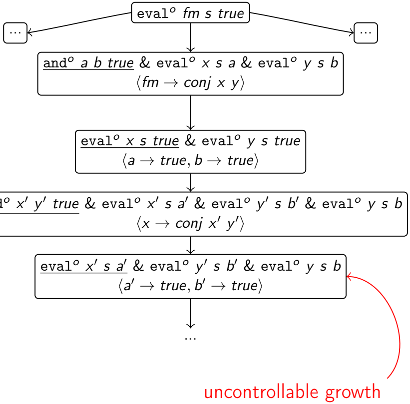
```
evalᵒ fm s true
```



CPD: Split is Necessary

```
let rec evalᵒ fm s r =
  ...
  fm ≡ conj x y & andᵒ a b r &
  evalᵒ x s a & evalᵒ y s b |
  ...
```

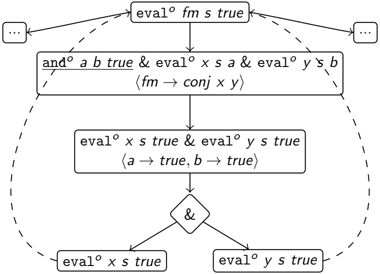
`evalᵒ fm s true`

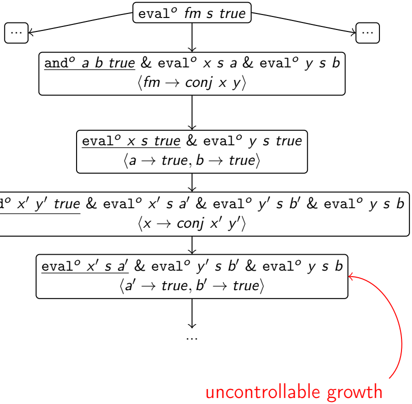


CPD: Split is Necessary

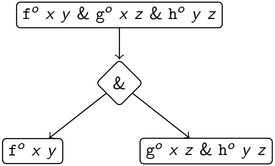
```
let rec evalᵒ fm s r =
  ...
  fm ≡ conj x y & andᵒ a b r &
  evalᵒ x s a & evalᵒ y s b |
  ...
```

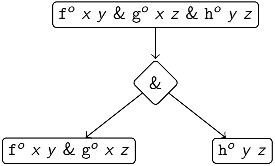
```
evalᵒ fm s true
```

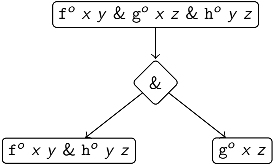


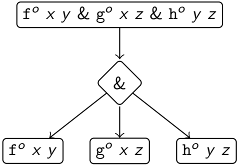


Split: Which Way is the Right Way?









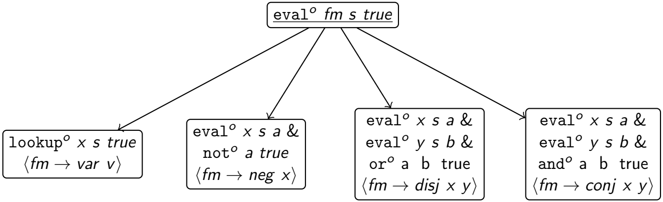
Decisions in Partial Deduction

- What to unfold: which calls, how many calls?
  - CPD: the leftmost call, which does not have a predecessor *embedded* into it
- How to unfold: to what depth a call should be unfolded?
  - CPD: unfold once
- When to stop driving?
  - When a goal is an instance of some goal in the process tree
- When to split?
  - When there is a predecessor embedded into the goal

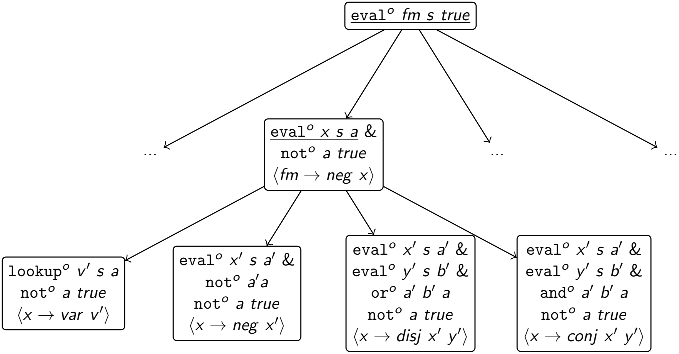
Evaluator of Logic Formulas: Unfolding Step 1

```
let rec evalᵒ fm s r =
  ocanren { fresh v x y a b in
    (fm ≡ var v & lookupᵒ v s r) |
    (fm ≡ neg x & evalᵒ x s a & notᵒ a r) |
    (fm ≡ conj x y & evalᵒ x s a & evalᵒ y s b & andᵒ a b r) |
    (fm ≡ disj x y & evalᵒ x s a & evalᵒ y s b & oroᵒ a b r) }
```

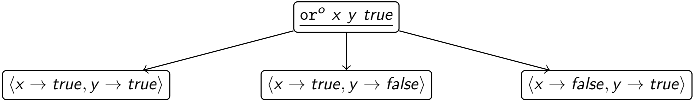
```
evalᵒ fm s true
```

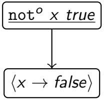


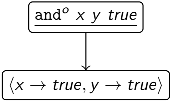
Evaluator of Logic Formulas: Unfolding Step 2



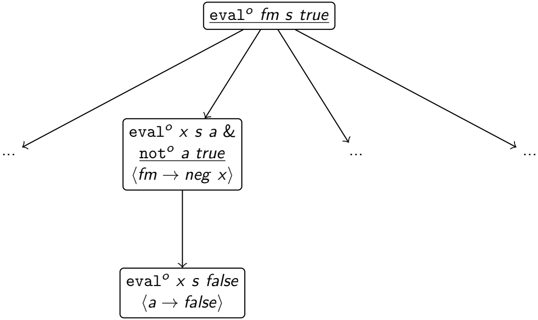
Unfolding of Boolean Connectives



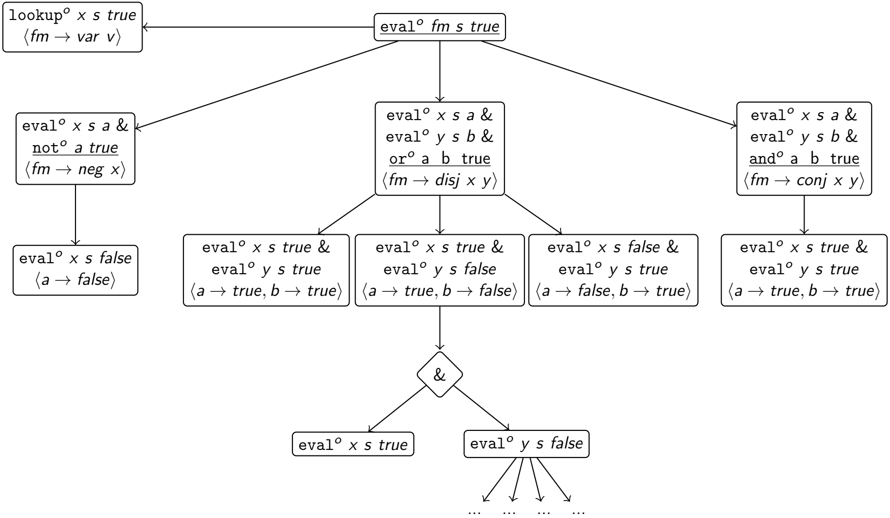




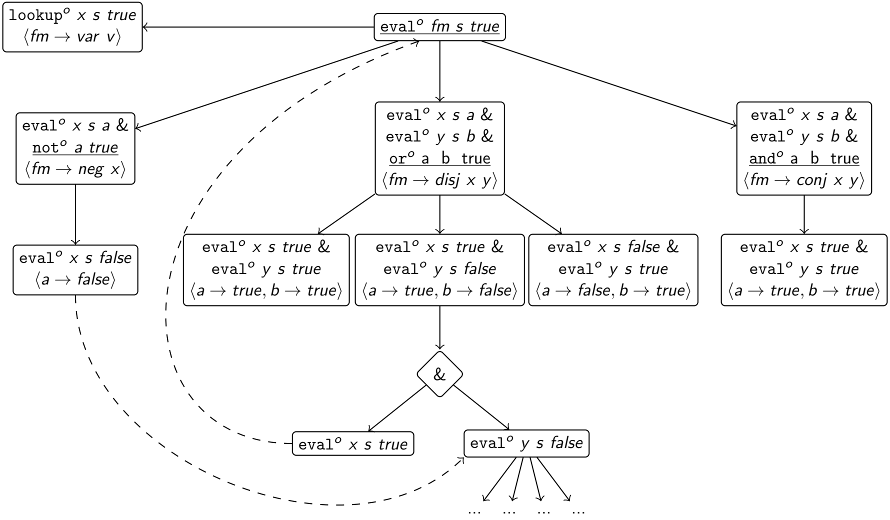
Unfolding Boolean Connectives First



Evaluator of Logic Formulas: Conservative PD



Evaluator of Logic Formulas: Conservative PD



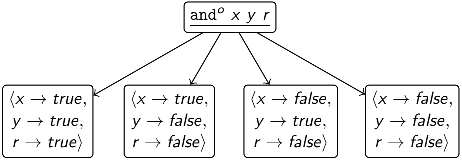
Conservative Partial Deduction

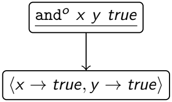
- Split conjunction into individual calls
- Unfold each call in isolation
- Unfold until embedding is encountered
- Find a call which narrows the search space (less-branching heuristics)
- Join the result of unfolding the selected call with the other calls not unfolded
- Continue driving the constucted conjunction

Less-branching Heuristics

Less-branching heuristics is used to select a call to unfold

If a call in the context unfolds into less branches than it does in isolation, select it





Evaluation

We implemented the Conservative Partial Deduction and compared it with CPD for miniKanren and CPD with branching heuristics on the following relations

- Two implementations of an evaluator of logic formulas
- A program to compute a unifier of two terms
- A program to search for paths of a specific length in a graph

Evaluator of Logic Formulas: Order of Calls

boolean connective last

```
let rec evalᵒ fm s r =
  ocanren { fresh v x y a b in
    (fm ≡ var v & lookupᵒ v s r) |
    (fm ≡ neg x & evalᵒ x s a & notᵒ a r) |
    (fm ≡ conj x y & evalᵒ x s a & evalᵒ y s b & andᵒ a b r) |
    (fm ≡ disj x y & evalᵒ x s a & evalᵒ y s b & orᵒ a b r) }
```

Evaluator of Logic Formulas: Order of Calls

boolean connective last

```
let rec evalᵒ fm s r =
  ocanren { fresh v x y a b in
    (fm ≡ var v & lookupᵒ v s r) |
    (fm ≡ neg x & evalᵒ x s a & notᵒ a r) |
    (fm ≡ conj x y & evalᵒ x s a & evalᵒ y s b & andᵒ a b r) |
    (fm ≡ disj x y & evalᵒ x s a & evalᵒ y s b & oroᵒ a b r) }
```

boolean connective first

```
let rec evalᵒ fm s r =
  ocanren { fresh v x y a b in
    (fm ≡ var v & lookupᵒ v s r) |
    (fm ≡ neg x & notᵒ a r & evalᵒ x s a) |
    (fm ≡ conj x y & andᵒ a b r & evalᵒ x s a & evalᵒ y s b) |
    (fm ≡ disj x y & oroᵒ a b r & evalᵒ x s a & evalᵒ y s b) }
```

Evaluator of Logic Formulas: Compexity of Relations

table-based implementation

```
let rec or^o x y r =
  ocanren {
    (x ≡ true & y ≡ true & r ≡ true) |
    (x ≡ true & y ≡ false & r ≡ true) |
    (x ≡ false & y ≡ true & r ≡ true) |
    (x ≡ false & y ≡ false & r ≡ false) }
```

Evaluator of Logic Formulas: Compexity of Relations

table-based implementation

```
let rec orᵒ x y r =
  ocanren {
    (x ≡ true & y ≡ true & r ≡ true) |
    (x ≡ true & y ≡ false & r ≡ true) |
    (x ≡ false & y ≡ true & r ≡ true) |
    (x ≡ false & y ≡ false & r ≡ false) }
```

implementation via nand^o

```
let orᵒ x y r =
  ocanren {
    fresh a b in
      (nandᵒ x x a & nandᵒ y y b & nandᵒ a b r) }

let rec nandᵒ x y r =
  ocanren {
    (x ≡ true & y ≡ true & r ≡ false) |
    (x ≡ true & y ≡ false & r ≡ true) |
    (x ≡ false & y ≡ true & r ≡ true) |
    (x ≡ false & y ≡ false & r ≡ true) }
```

Evaluator of Logic Formulas: Evaluation

Implementations:

- *last*: boolean connectives last, implemented via $\texttt{nand}^o$
- *plain*: boolean connectives first, straightforward implementation

Query: find 1000 formulas which evaluate to true

	last	plain
Original	1.06s	1.84s
CPD	—	1.13s
Branching	3.11s	7.53s
ConsPD	0.93s	0.99s

Table: Evaluation results

Unification

Relation to find a unifier of two terms

Query: unification of terms $f(X, X, g(Z, t))$ and $f(g(p, L), Y, Y)$

Path Search

Relation to search for paths in a graph

Query: find 5 paths in a graph with 20 vertices and 30 edges

Evaluation Results

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

Table: Evaluation results

Conclusion

- We developed and implemented Conservative Partial Deduction
  - Less-branching heuristics
- Evaluation shows some improvement, but not for every query
- Future work:
  - Develop models to predict execution time
  - Develop specialization which is more predictable, stable and well-behaved