JETBRAINS RESEARCH

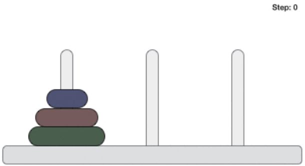
Integration of Offline Partial Deduction and Functional Conversion for miniKanren

Aleksandr Shefer, Ekaterina Verbitskaia

JetBrains Research, Programming Lanuages and Program Analysis Lab

April 7, 2024

Search is More Complicated than Verification



The Tower of Hanoi Puzzle

verify $[1 \to 2,\ 1 \to 3,\ 2 \to 3]$ ⇒ True
verify $[1 \to 2,\ 1 \to 2]$ ⇒ False

solve True ⇒ $[1 \to 2,\ 1 \to 3,\ 2 \to 3,\ \ldots]$

Search is Dual to Verification

$$\text{solve} \approx \mathit{verify}^{-1}$$

Logic Programming Highlights the Duality

$\texttt{hanoi}^o\ \texttt{candidate result}$

$\texttt{verify} \ = \textsf{run}\ \ \texttt{q}\ \ (\texttt{hanoi}^o\ [1 \to 2{,}1 \to 3{,}2 \to 3]\ \ \texttt{q}) \ \Rightarrow \texttt{q} = \texttt{True}$

$\texttt{solve} \ \ = \textsf{run}\ \ \texttt{q}\ \ (\texttt{hanoi}^o\ \texttt{q}\ \texttt{True}) \Rightarrow \texttt{q} = [1 \to 2{,}1 \to 3{,}2 \to \ 3{,}\ldots]$

Program Interpretation and Synthesis

Verifier

```
eval st (Conj x y) =
  eval st x && eval st y
 ...
```

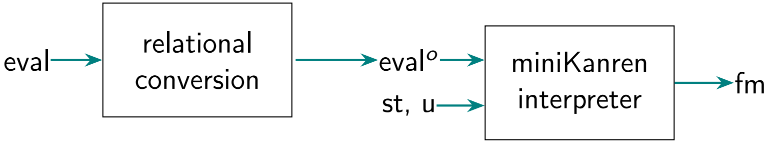
Solver

```
synth st res = do
  (u, v) ← [(u, v) |
    u ← [False, True],
    v ← [False, True],
    u && v == res]
  x ← synth st u
  y ← synth st v
  return (Conj x y)
  ...
```

Relational Interpreters for Search

```
eval st (Conj x y) =
  eval st x && eval st y
 ...
```

```
eval^o st fm u = fresh (x y v w)
    (fm ≡ Conj x y ∧
     eval^o st x v ∧
     eval^o st y w ∧
     and^o v w u);
     ...
```



Relational Interpreters for Search: the Issue

It is slow

Relational Programming in miniKanren

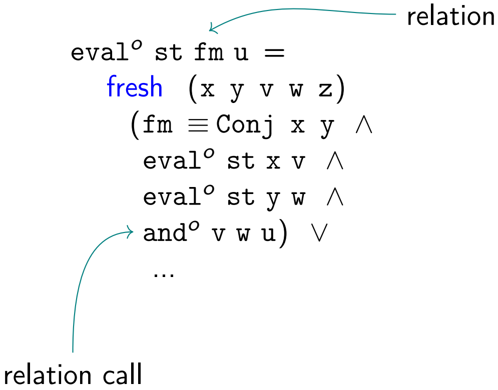
- Pure logic programming
- Complete search: interleaving

miniKanren Syntax

relation

$$
\begin{array}{l}
\texttt{eval}^{o}\ \texttt{st fm u} = \\
\quad \textsf{fresh}\ \ (\texttt{x y v w z}) \\
\qquad (\texttt{fm} \equiv \texttt{Conj x y} \ \wedge \\
\qquad\ \ \texttt{eval}^{o}\ \texttt{st x v} \ \wedge \\
\qquad\ \ \texttt{eval}^{o}\ \texttt{st y w} \ \wedge \\
\qquad\ \ \texttt{and}^{o}\ \texttt{v w u}) \ \vee \\
\qquad\ \ \ldots
\end{array}
$$

miniKanren Syntax

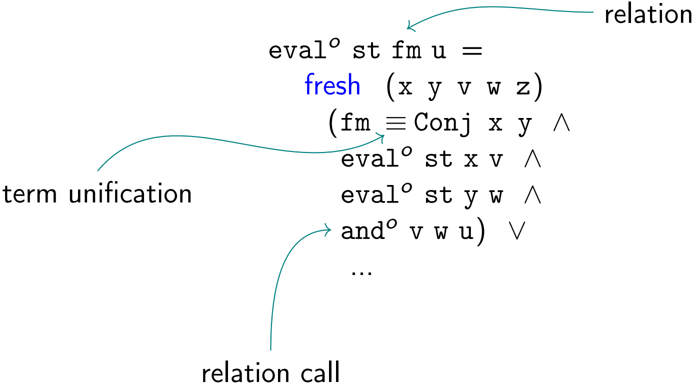


relation

```
eval^o st fm u =
  fresh (x y v w z)
    (fm ≡ Conj x y ∧
     eval^o st x v ∧
     eval^o st y w ∧
     and^o v w u) ∨
      ...
```

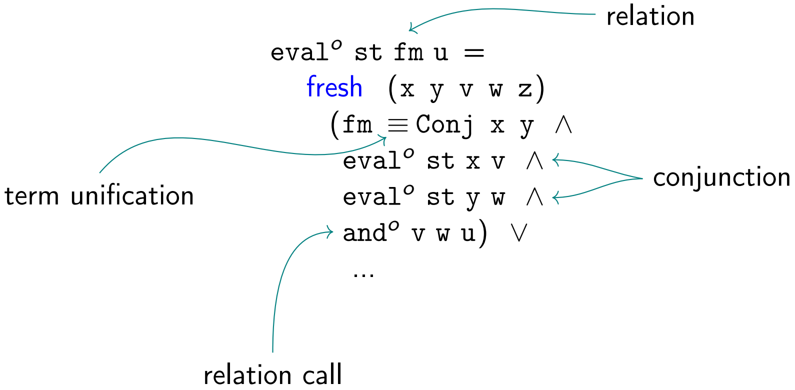
relation call

miniKanren Syntax



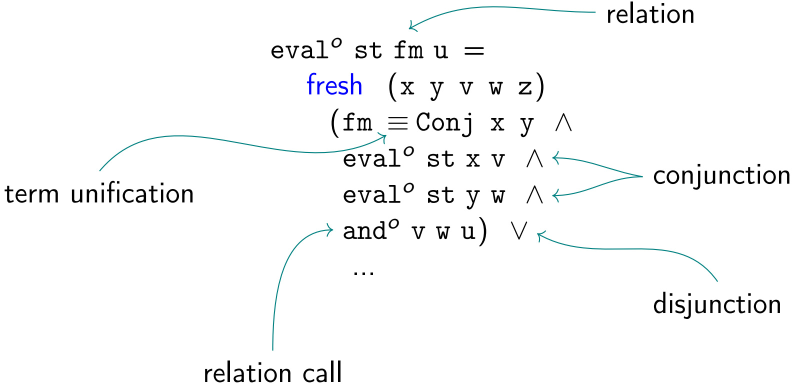
```
eval^o st fm u =
  fresh  (x y v w z)
   (fm ≡ Conj x y  ∧
    eval^o st x v  ∧
    eval^o st y w  ∧
    and^o v w u)  ∨
     ...
```

miniKanren Syntax



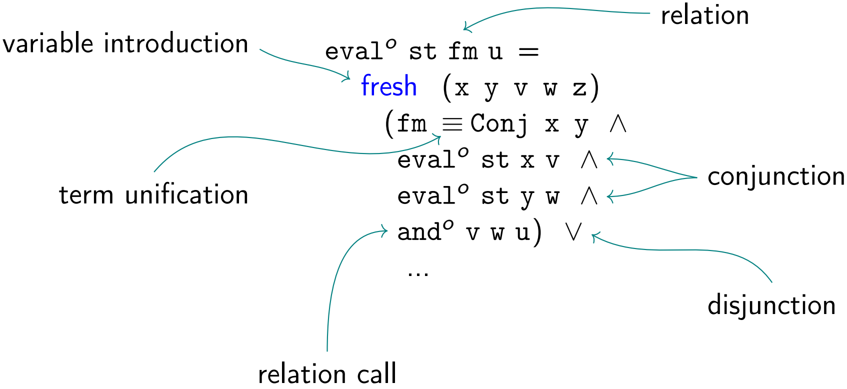
```
$\texttt{eval}^o$ st fm u =
  fresh (x y v w z)
   (fm ≡ Conj x y ∧
    $\texttt{eval}^o$ st x v ∧
    $\texttt{eval}^o$ st y w ∧
    $\texttt{and}^o$ v w u) ∨
    ...
```

miniKanren Syntax



$$
\begin{array}{l}
\texttt{eval}^{o}\ \texttt{st fm u} =\\
\quad \textsf{fresh}\ \ (\texttt{x y v w z})\\
\quad\ (\texttt{fm} \equiv \texttt{Conj x y} \ \wedge\\
\qquad \texttt{eval}^{o}\ \texttt{st x v} \ \wedge\\
\qquad \texttt{eval}^{o}\ \texttt{st y w} \ \wedge\\
\qquad \texttt{and}^{o}\ \texttt{v w u}) \ \vee\\
\qquad \dots
\end{array}
$$

miniKanren Syntax



```
eval^o st fm u =
  fresh (x y v w z)
   (fm ≡ Conj x y ∧
    eval^o st x v ∧
    eval^o st y w ∧
    and^o v w u) ∨
    ...
```

relation

variable introduction

term unification

conjunction

relation call

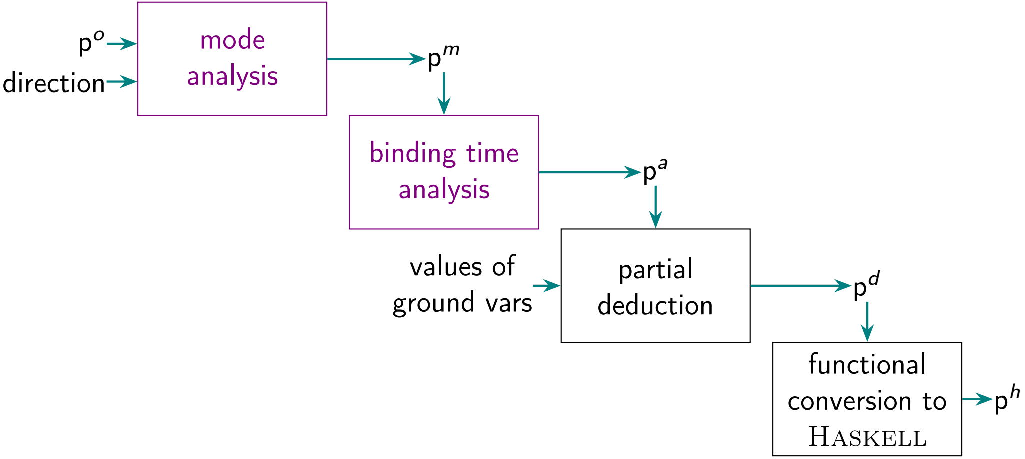
disjunction

Sources of Inefficiency

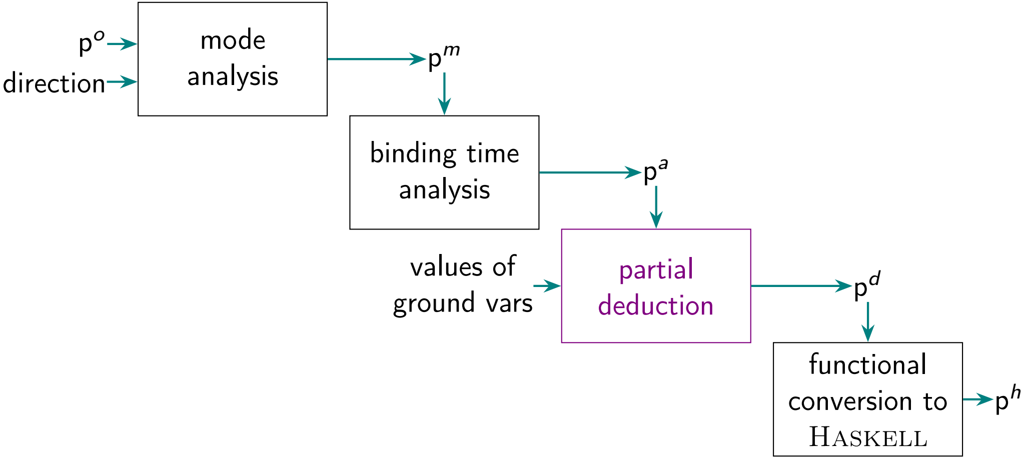
- Running backwards is slow
- Order of clauses influences performance
- Constant arguments (`eval`^{o} `[] q True`)

Solution: specialization

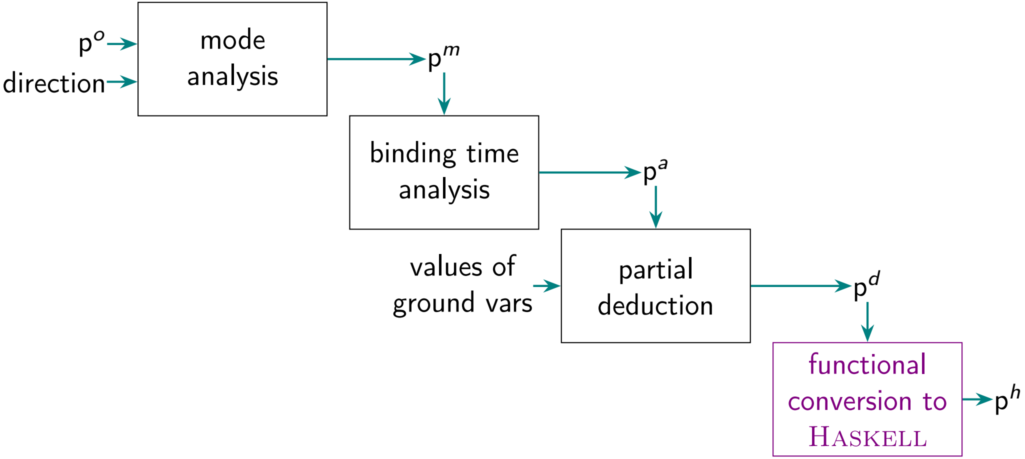
Specialization Scheme



Specialization Scheme



Specialization Scheme



Mode Analysis

Variable modes:

- Ground term: True
- Free variable: x

$$\texttt{eval}^o\ []\ \texttt{fm}\ \texttt{True} \rightarrow \texttt{eval}^o\ g\ f\ g$$

```
eval^o st fm u =
  fresh (x y v w z)
    (fm ≡ Conj x y ∧
     eval^o st x v ∧
     eval^o st y w ∧
     and^o v w u) ∨   u = True
     ...
```

```
eval^o st fm u =
  fresh (x y v w z)
    (and^o v w u ∧      u = True
     eval^o st x v ∧    v = True
     eval^o st y w ∧    w = True
     fm ≡ Conj x y) ∨
     ...
```

Partial Deduction

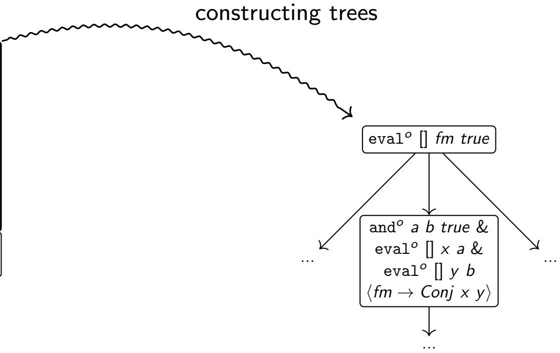
```
eval^o st fm u =
  safe  and^o a b u &
  unsafe  eval^o st x a &
  unsafe  eval^o st y b &
  fm ≡ Conj  x  y  |
   ...
```

```
eval^o [] fm  true
```

Partial Deduction

```
eval^o st fm u =
   safe  and^o a b u &
   unsafe  eval^o st x a &
   unsafe  eval^o st y b &
  fm ≡ Conj x y |
   ...
```

eval^o [] fm true



constructing trees

eval^o [] *fm true*

...

and^o *a b true* &
eval^o [] *x a* &
eval^o [] *y b*
$\langle fm \rightarrow Conj\ x\ y\rangle$

...

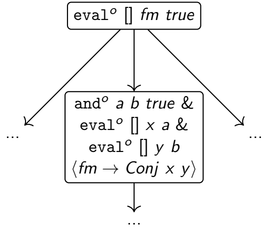
...

Partial Deduction

constructing trees

```
eval^o st fm u =
  safe  and^o a b u &
  unsafe  eval^o st x a &
  unsafe  eval^o st y b &
  fm ≡ Conj x y |
   ...
```

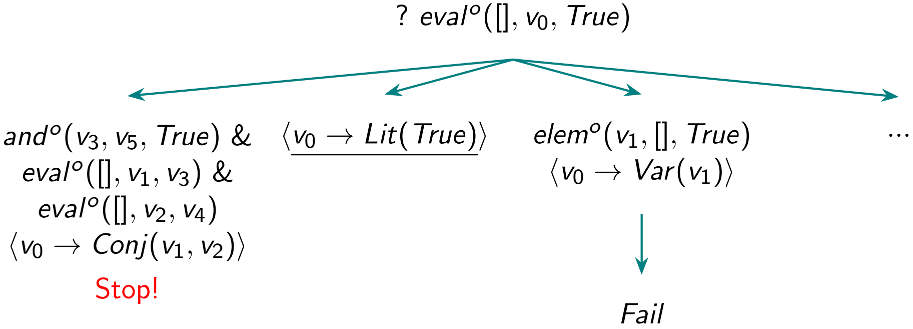
```
eval^o [] fm true
```



```
eval^o_true fm =
  eval^o_true x &
  eval^o_true y &
  fm ≡ conj x y |
   ...
```

residualization

Partial Deduction: Trees Construction



$? \ eval^o([], v_0, True)$

$and^o(v_3, v_5, True) \ \& \ eval^o([], v_1, v_3) \ \& \ eval^o([], v_2, v_4)$
$\langle v_0 \rightarrow Conj(v_1, v_2)\rangle$

Stop!

$\langle \underline{v_0 \rightarrow Lit(True)} \rangle$

$elem^o(v_1, [], True)$
$\langle v_0 \rightarrow Var(v_1)\rangle$

$Fail$

$\cdots$

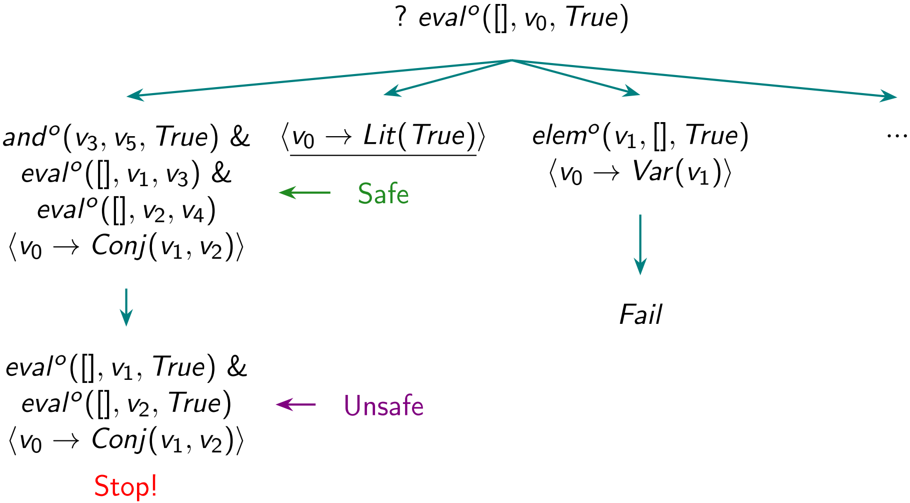
Binding Time Analysis

Call annotations:

- Safe: continue unfolding
- Unsafe: **possibly** stop unfolding

```
eval^o st fm u =
  fresh  (x y v w z)
   ( safe  and^o a b u &
     unsafe  eval^o st x a &
     unsafe  eval^o st y b &
    fm ≡ Conj  x  y) |
     ...
```

Partial Deduction with Annotations



$? \; eval^o([], v_0, True)$

$and^o(v_3, v_5, True) \; \&$
$eval^o([], v_1, v_3) \; \&$
$eval^o([], v_2, v_4)$
$\langle v_0 \to Conj(v_1, v_2) \rangle$

$\underline{\langle v_0 \to Lit(True) \rangle}$

$\leftarrow$ Safe

$elem^o(v_1, [], True)$
$\langle v_0 \to Var(v_1) \rangle$

$\cdots$

$Fail$

$eval^o([], v_1, True) \; \&$
$eval^o([], v_2, True)$
$\langle v_0 \to Conj(v_1, v_2) \rangle$

$\leftarrow$ Unsafe

Stop!

Functional Conversion

```
eval_true$^o$ fm =
  fresh (x y)
    (eval_true$^o$ x $\wedge$
     eval_true$^o$ y $\wedge$
     fm $\equiv$ Conj x y) $\vee$
    ...
```

⇒

```
eval_true = do
  x ← eval_true
  y ← eval_true
   return (Conj x y)
   ...
```

Evaluation

- Functional Conversion
- Online Partial Deduction and Functional Conversion
- Our approach: Offline Partial Deduction and Functional Conversion

Evaluation: Propositional Evaluator

	Functional conversion	Translation with partial deduction	
		Online	Offline
10 formulas	4.05 µs	0.33 µs	0.37 µs
100 formulas	56.00 µs	7.40 µs	7.73 µs
1000 formulas	645.00 µs	108.00 µs	108.00 µs

Evalutaion: the Tower of Hanoi puzzle

	Functional conversion	Translation with partial deduction	
		Online	Offline
3 disks	153 000.00 μs	125 000.00 μs	1.67 μs
4 disks	🕒timeout	🕒timeout	3.12 μs

Evalutaion: Other Benchmarks (DPPD[1])

	Functional conversion	Translation with partial deduction	
		Online	Offline
appLast	1.13 µs	0.08 µs	0.09 µs
contains	14.70 µs	0.78 µs	0.77 µs
doubleAppend	🕒timeout	2910.00 µs	2.00 µs
exDepth	🕒timeout	🕒timeout	3.09 µs
nthOpt	0.10 µs	0.08 µs	0.10 µs
...			

[1] Dozens of Problems for Partial Deduction benchmark: https://github.com/leuschel/DPPD

Conclusion

- We integrated the Functional Conversion and Offline Partial Deduction approaches
- We conducted a preliminary evaluation of the approach
- The effectiveness of the integration has been shown