Символические вычисления для реляционного программирования

Автор: Екатерина Вербицкая

Лаборатория языковых инструментов JetBrains
Санкт-Петербургский государственный университет

16 декабря 2017

Реляционное программирование

Программа — отношение

$$append^o \subseteq [A] \times [A] \times [A]$$

$$\begin{aligned} append^o \quad = \quad \{ \quad & ([\,], [\,], [\,]); \\ & ([0], [\,], [0]); \\ & ([1], [\,], [1]); \\ & \ldots \\ & ([\,], [0], [0]); \\ & \ldots \\ & ([4], [2], [4, 2]); \\ & \ldots \\ & ([4, 2], [13], [4, 2, 13]); \\ & \ldots \\ \} & \end{aligned}$$

Пример программы

В нотации Пролога

$$
\begin{aligned}
& append^o\ [\,]\ y\ y\,. \\
& append^o\ (h : t)\ y\ (h : ty) \quad \leftarrow \quad append^o\ t\ y\ ty\,.
\end{aligned}
$$

В нотации miniKanren

$$
\begin{aligned}
append^o\ x\ y\ xy \quad = \quad & x \equiv [\,] \wedge xy \equiv y \\
\vee \quad & \exists\, h\ t\ ty : \\
& x \equiv (h : t) \wedge xy \equiv (h : ty) \wedge append^o\ t\ y\ ty
\end{aligned}
$$

Вычисление в реляционном программировании

$$
\begin{array}{llllll}
append^o & [1] & [2,3] & q & \rightarrow \{ & [1,2,3] \} \\
append^o & [1] & q & [1,2,3] & \rightarrow \{ & [2,3] \} \\
append^o & q & [1] & [2,3] & \rightarrow \{ & \}
\end{array}
$$

Вычисление в реляционном программировании

$$append^o \quad q \quad p \quad [1,2] \quad \rightarrow \quad \{ \quad ([\,], [1,2]), \; ([1], [2]), \; ([1,2], [\,]) \quad \}$$

$$append^o \quad q \quad q \quad [2,4,2,4] \quad \rightarrow \quad \{ \quad [2,4] \quad \}$$

$$append^o \quad q \quad p \quad r \quad \rightarrow \quad \{ \quad ([\,], \; \__0, \; \__0), \; ([\__0], \; \__1, \; (\__0 : \__1)), \; ((\__0 : \__1), \; \__2, \; (\__0 : \__1 : \__2)) \; \ldots \quad \}$$

Направление вычислений

$$foo^o \subseteq A \times B$$

- $foo^o\ \alpha\ q : A \to [B]$
- $foo^o\ q\ \beta : B \to [A]$ — в “обратную” сторону
- $foo^o\ q\ p : () \to [(A \times B)]$

Направление вычислений

$$foo^o \subseteq A \times B$$

- $foo^o\ \alpha\ q : A \to [B]$
- $foo^o\ q\ \beta : B \to [A]$ — в “обратную” сторону
- $foo^o\ q\ p : () \to [(A \times B)]$

Время вычисления в разные стороны часто существенно отличается

Направление вычислений

$$foo^o \subseteq A \times B$$

- $foo^o\ \alpha\ q : A \to [B]$
- $foo^o\ q\ \beta : B \to [A]$ — в “обратную” сторону
- $foo^o\ q\ p : () \to [(A \times B)]$

Время вычисления в разные стороны часто существенно отличается

$$factorize\ num = mult^o\ [p, q]\ num$$

Порождение функций

$$
\begin{array}{lcl}
append^o\ x\ y\ xy & = & x \equiv [\,] \wedge xy \equiv y \\
& \vee & \exists\, h\ t\ ty : \\
& & x \equiv (h : t) \wedge xy \equiv (h : ty) \wedge append^o\ t\ y\ ty
\end{array}
$$

```
let append x y =
  match x with
  | [] → y
  | (h:t) → (h : append t y)
```

Порождение функций

$$
\begin{array}{lcl}
append^o\ x\ y\ xy & = & x \equiv [\,] \wedge xy \equiv y \\
& \vee & \exists\, h\ t\ ty : \\
& & x \equiv (h : t) \wedge xy \equiv (h : ty) \wedge append^o\ t\ y\ ty
\end{array}
$$

```
let append x y =
  match x with
  | [] → y
  | (h:t) → (h : append t y)
```

```
let suffix x xy =
  match x, xy with
  | [], y → y
  | (h:t), (h':t') | h = h' → suffix t t'
  | _ → error "x is not a prefix of xy"
```

Цель

Порождать функции из отношений

- Для заданного направления
- С максимально адекватной производительностью

Что нужно для порождения функций из отношения?

- Исследовать поведение программы
- Вычислить все, что возможно, зная направление (“известные” аргументы)

Что нужно для порождения функций из отношения?

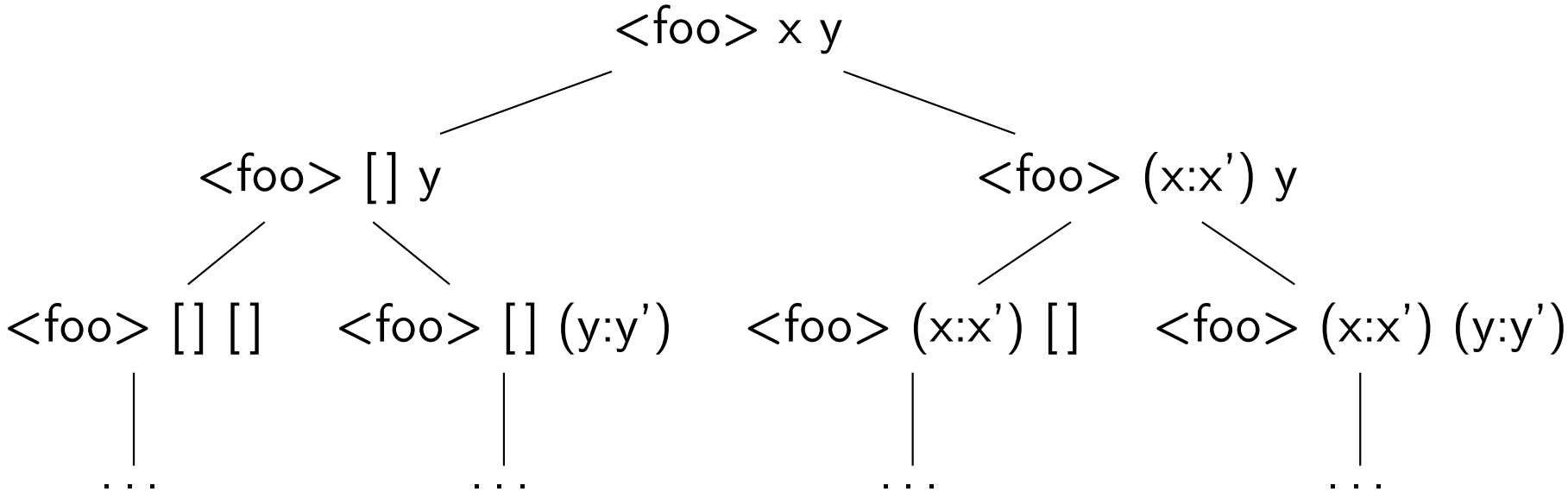
- Исследовать поведение программы
- Вычислить все, что возможно, зная направление (“известные” аргументы)

- Суперкомпиляция — средство получения информации о поведении программы
- Частичная дедукция — суперкомпиляция для логических языков
- Адаптируем суперкомпиляцию для miniKanren

Суперкомпиляция

```
foo: [A] → [B]
foo x y =
  ...
```

Строим дерево процессов,
обеспечивая его конечность



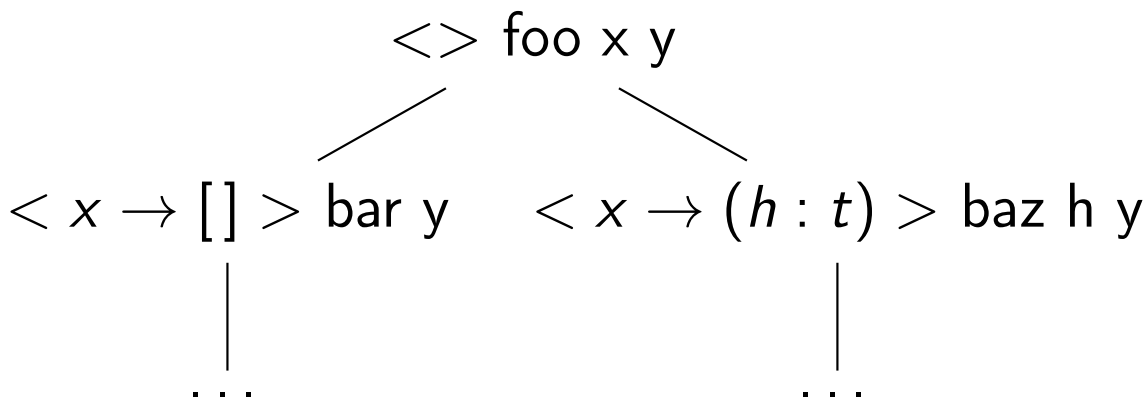
Строим остаточную программу

```
better_foo x y =
  ...
```

“Суперкомпиляция” для miniKanren

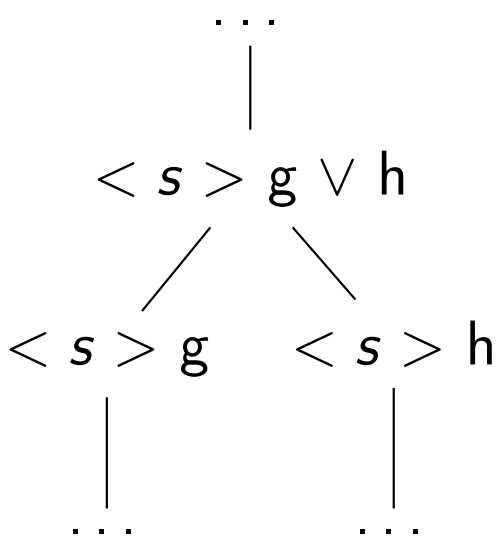
```
foo ⊆ [A] × [B]
foo x y =
 (x ≡ [] ∧ bar y)
 ∨ (∃ h t
     (x ≡ h : t
      ∧ baz h y))
```

В узлах дерева процессов
текущая цель и подстановка



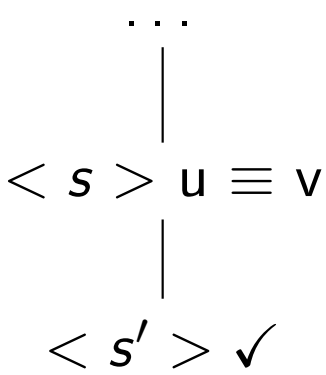
Строим функциональную остаточную программу

Дерево процессов: дизъюнкция

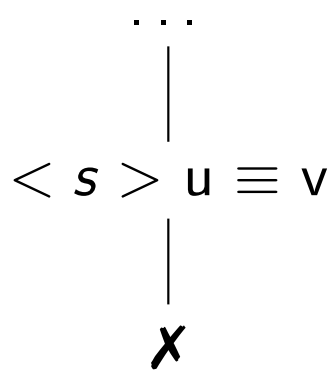


Дерево процессов: унификация

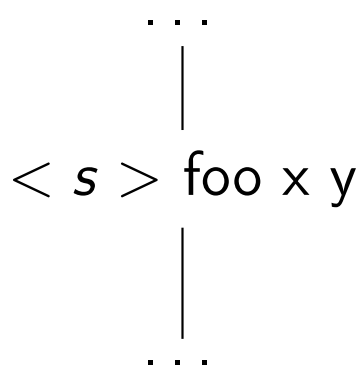
Если унификация успешна



Если унификация неуспешна

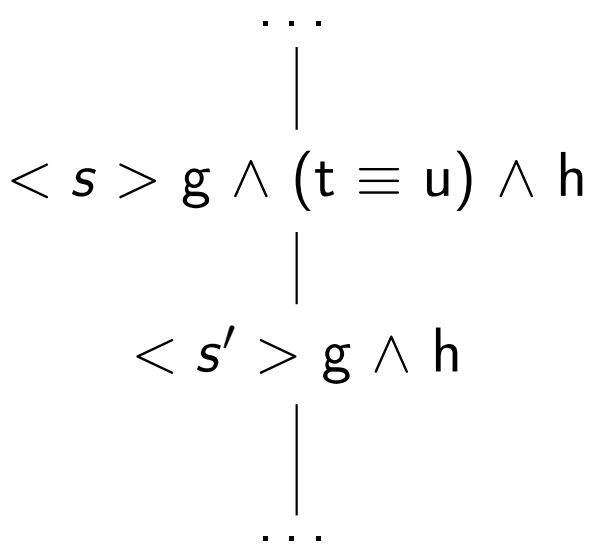


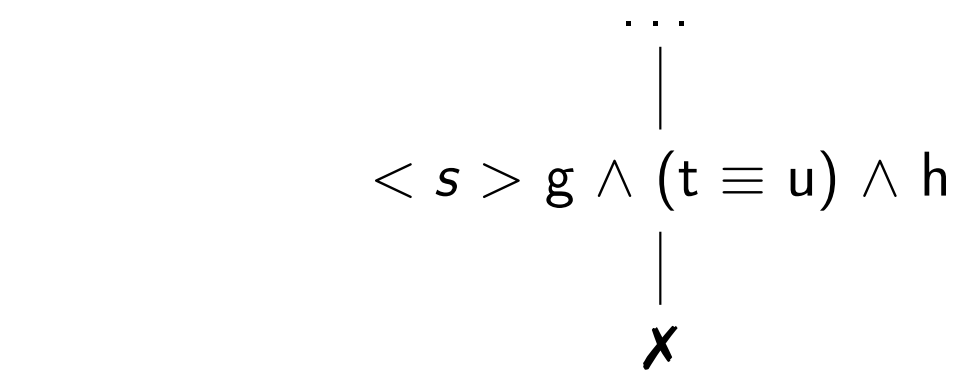
Дерево процессов: применение отношения



- Если цель (с точностью до переименования переменных) встречалась раньше, прекращаем строить дерево
- Если цель *похожа* на какую-то из ранее встретившихся, попробовать ее *абстрагировать* и продолжить строить дерево
- Если цель ни на что не похожа, продолжаем строить дерево для применения отношения

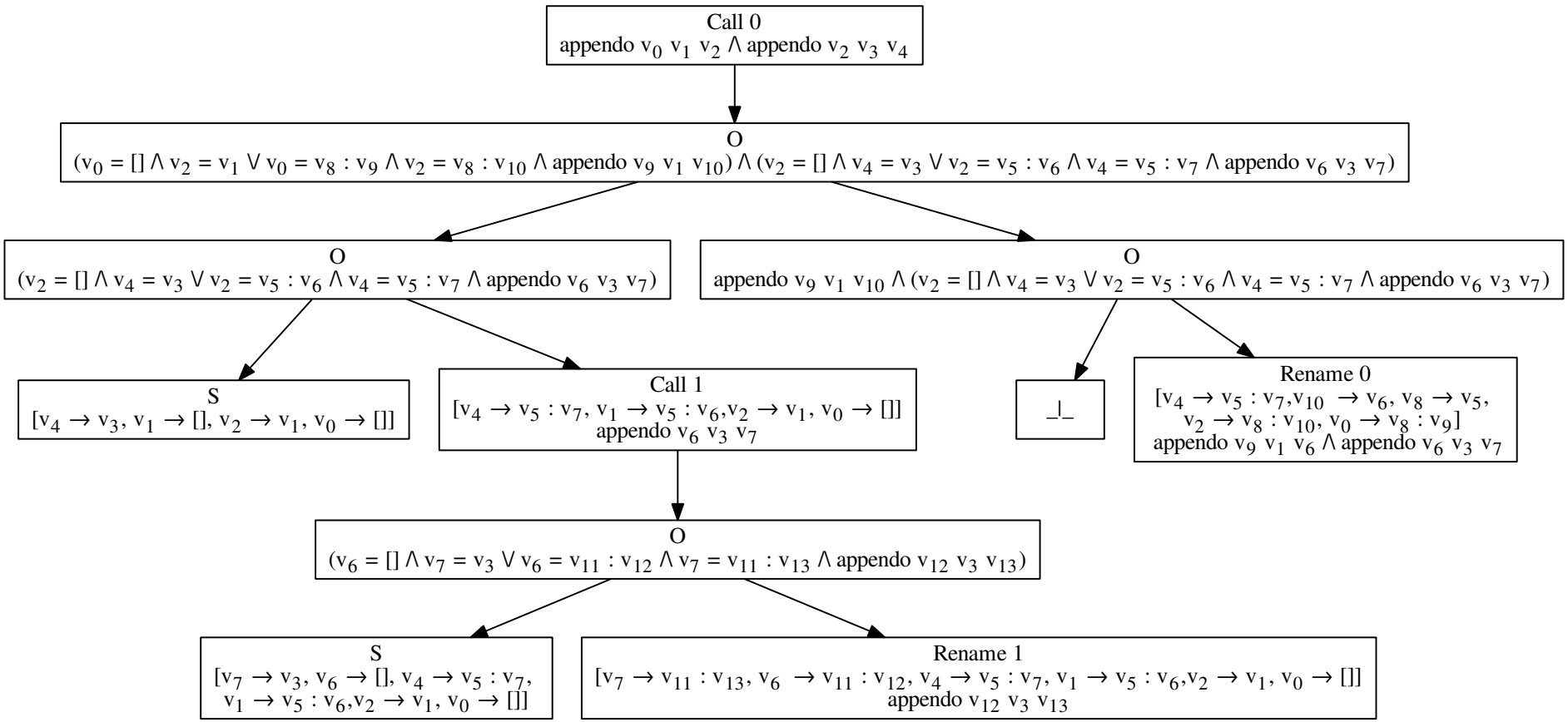
Дерево процессов: конъюнкция





- Все унификации проталкиваем вверх и вычисляем в подстановки
- Конъюнкция применений обрабатывается похоже на единичное применение

Дерево процессов (упрощенное)



Сложности

- Что значит, что цель *похожа* на другую цель?
- Как *абстрагировать*?
- Как учитывается связь между переменными?

Текущее положение дел

Реализовано несколько вариантов “суперкомпиляции”

- Пока все они не вполне устраивают
- Ищу вариант получше в литературе по частичной дедукции

Дальнейшие планы

- Построение остаточной программы
- “Негативная” суперкомпиляция
- Адаптация более мощных техник символьных вычислений