Суперкомпиляция для miniKanren

Автор: Екатерина Вербицкая

Лаборатория языковых инструментов JetBrains
Санкт-Петербургский государственный университет

15 декабря 2018

Реляционное программирование

Программа — отношение

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

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

Пример программы на miniKanren

```
appendo x y z =
  (x ≡ [] ∧ z ≡ y)
  ∨ (∃ h t r
        ( x ≡ h : t
        ∧ z ≡ h : r
        ∧ appendo t y r))
```

Вычисление в реляционном программировании

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

Вычисление в реляционном программировании

$$
\begin{aligned}
append^o\ q\ p\,[1,2] \quad &\to \quad \{ \\
&\qquad\qquad ([\,],[1,2]), \\
&\qquad\qquad ([1],[2]), \\
&\qquad\qquad ([1,2],[\,]) \\
&\qquad \}
\end{aligned}
$$

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

$$
\begin{aligned}
append^o\ q\ p\ r \quad &\to \quad \{ \\
&\qquad\qquad ([\,],\ \alpha,\ \alpha), \\
&\qquad\qquad ([\alpha],\ \beta,\ (\alpha:\beta)) \\
&\qquad\qquad ((\alpha:\beta),\ \gamma,\ (\alpha:(\beta:\gamma))) \\
&\qquad\qquad \ldots \\
&\qquad \}
\end{aligned}
$$

Направление вычислений

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

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

Направление вычислений

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

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

Время вычисления в разных направлениях часто отличается

Направление вычислений

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

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

Время вычисления в разных направлениях часто отличается

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

Цель

Улучшать производительность реляционных программ, не уменьшая декларативности подхода

- Для заданного направления
- Учитывая частично определенные входные данные

Оптимизации: известные данные

```
foo p q ∧ repeat x p
```

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

repeat ⊆ A × [A]
repeat x xs =
  ∃ r (xs ≡ x : r ∧ repeat x r)
```

Оптимизации: известные данные

```
foo p q ∧ repeat x p
```

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

repeat ⊆ A × [A]
repeat x xs =
  ∃ r (xs ≡ x : r ∧ repeat x r)
```

```
foo_r q
```

```
foo_r ⊆ [A] × [B]
foo_r y = light y
```

Оптимизации: промежуточные структуры данных

```
map f p q ∧ map g q r
```

```
map f ⊆ [A] × [B]
map f x y =
 (x ≡ [] ∧ y ≡ [])
 ∨ (∃ h t r ( x ≡ h : t
             ∧ y ≡ f h : r
             ∧ map f t r))
```

Оптимизации: промежуточные структуры данных

```
map f p q ∧ map g q r
```

```
map f ⊆ [A] × [B]
map f x y =
 (x ≡ [] ∧ y ≡ [])
 ∨ (∃ h t r ( x ≡ h : t
            ∧ y ≡ f h : r
            ∧ map f t r))
```

```
map_fg f g p r
```

```
map_fg f g ⊆ [A] × [B]
map_fg f g x y =
 (x ≡ [] ∧ y ≡ [])
 ∨ (∃ h t r ( x ≡ h : t
            ∧ y ≡ g (f h) : r
            ∧ map_fg f g t r))
```

Оптимизации: группировка вычислений

```
sum p s ∧ len p l
```

```
sum ⊆ [A] × Int
sum x s = (x ≡ [] ∧ s ≡ 0)
        ∨ (∃ h t r
            ( x ≡ h : t
            ∧ sum t r
            ∧ s = r + h))

len ⊆ [A] × Int
len x l = (x ≡ [] ∧ l ≡ 0)
        ∨ (∃ h t m
            ( x ≡ h : t
            ∧ len t m
            ∧ l = m + 1 ))
```

Оптимизации: группировка вычислений

```
sum p s ∧ len p l
```

```
sum ⊆ [A] × Int
sum x s = (x ≡ [] ∧ s ≡ 0)
        ∨ (∃ h t r
              ( x ≡ h : t
              ∧ sum t r
              ∧ s = r + h))

len ⊆ [A] × Int
len x l = (x ≡ [] ∧ l ≡ 0)
        ∨ (∃ h t m
              ( x ≡ h : t
              ∧ len t m
              ∧ l = m + 1 ))
```

```
sum_len p s l
```

```
sum_len ⊆ [A] × Int × Int
sum_len x s l =
  (x ≡ [] ∧ s ≡ 0 ∧ l ≡ 0)
  ∨ (∃ h t r m
        ( x ≡ h : t
        ∧ sum_len t r m
        ∧ s = r + h
        ∧ l = m + 1))
```

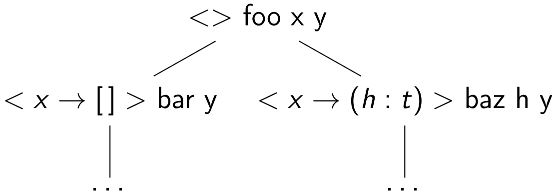
За счет чего можно улучшать производительность

- Статически вычислить все, что можно: специализация
  - Если известны некоторые аргументы
  - Если известно направление вычисления
- Не делать одну и ту же работу дважды
  - Избегать промежуточные значения: deforestation
  - Группировать вычисления, выполняемые во время обхода одной структуры данных: tupling

Все это может делать конъюнктивная частичная дедукция (суперкомпиляция) для логических языков

Суперкомпиляция для miniKanren: дерево процессов

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



В какой момент завершать символические вычисления?

Дерево процессов: применение отношения

$\ldots$

$< s >$ foo x y

$\ldots$

- Если цель (с точностью до подстановки) встречалась раньше, перестаем исследовать эту ветвь
- Если цель *похожа* на какого-то ее предка, попробовать ее *абстрагировать* и продолжить строить дерево
- Если цель ни на что не похожа, продолжаем строить дерево для применения отношения

Дерево процессов: конъюнкция

$$\begin{array}{c} \dots \\ | \\ < s > \mathrm{g} \wedge (\mathrm{t} \equiv \mathrm{u}) \wedge \mathrm{h} \\ | \\ < s' > \mathrm{g} \wedge \mathrm{h} \\ | \\ \dots \end{array} \qquad\qquad \begin{array}{c} \dots \\ | \\ < s > \mathrm{g} \wedge (\mathrm{t} \equiv \mathrm{u}) \wedge \mathrm{h} \\ | \\ \text{✗} \end{array}$$

- Все унификации проталкиваем вверх и вычисляем в подстановки
- Остается конъюнкция применений отношений
  - Проверяем на совпадение с предками
  - Проверяем на *похожесть* с предками
  - Иначе разворачиваем применения отношений

Ошибка: слишком агрессивное разворачивание вызовов

Трудности

- Определить, что значит, что цели *похожи*
- Определить, как *абстрагировать*
- Определить, как учитывать связи между переменными
- Проверка на похожесть и абстракция связаны между собой существенно теснее, чем в функциональных языках

Ошибка: считать, что решение для функциональных языков применимо для трансформации реляционных программ

Примеры, которые уже работают

$$\begin{aligned}&\texttt{appendo x y z} =\\&\quad(\texttt{x} \equiv \texttt{[]} \wedge \texttt{y} \equiv \texttt{z})\\&\quad\vee (\exists\ \texttt{h t r}\ (\texttt{x} \equiv \texttt{h : t} \wedge \texttt{z} \equiv \texttt{h : r} \wedge \texttt{appendo t y r}))\end{aligned}$$

```
appendo [] y z
```

Примеры, которые уже работают

```
appendo x y z =
  (x ≡ [] ∧ y ≡ z)
  ∨ (∃ h t r (x ≡ h : t ∧ z ≡ h : r ∧ appendo t y r))
```

```
appendo [] y z
```

```
empty_appendo y z
```

```
empty_appendo y z = y ≡ z
```

Примеры, которые уже работают

```
appendo x y z =
  (x ≡ [] ∧ y ≡ z)
  ∨ (∃ h t r (x ≡ h : t ∧ z ≡ h : r ∧ appendo t y r))
```

```
appendo [1,2,3] y z
```

Примеры, которые уже работают

```
appendo x y z =
  (x ≡ [] ∧ y ≡ z)
  ∨ (∃ h t r (x ≡ h : t ∧ z ≡ h : r ∧ appendo t y r))
```

```
appendo [1,2,3] y z
```

```
app_123 y z
```

```
app_123 y z =
  z ≡ 1 : (2 : (3 : y))
```

Примеры, которые уже работают

```
appendo x y z =
  (x ≡ [] ∧ y ≡ z)
  ∨ (∃ h t r (x ≡ h : t ∧ z ≡ h : r ∧ appendo t y r))
```

```
appendo x [1,2,3] z
```

Примеры, которые уже работают

```
appendo x y z =
  (x ≡ [] ∧ y ≡ z)
  ∨ (∃ h t r (x ≡ h : t ∧ z ≡ h : r ∧ appendo t y r))
```

```
appendo x [1,2,3] z
```

```
app_123 x z
```

```
app_123 x z =
  (x ≡ [] ∧ z ≡ [1,2,3])
  ∨ (∃ h t r ( x ≡ h : t
              ∧ z ≡ h : r
              ∧ app_123 t r))
```

Примеры, которые уже работают

```
appendo x y z =
  (x ≡ [] ∧ y ≡ z)
  ∨ (∃ h t r (x ≡ h : t ∧ z ≡ h : r ∧ appendo t y r))
```

```
appendo x y i
 ∧  appendo i z r
```

Примеры, которые уже работают

```
appendo x y z =
  (x ≡ [] ∧ y ≡ z)
  ∨ (∃ h t r (x ≡ h : t ∧ z ≡ h : r ∧ appendo t y r))
```

```
appendo x y i
 ∧  appendo i z r
```

```
double_app x y z r
```

```
double_app x y z r = ∃ h t s
  (x ≡ [] ∧ y ≡ [] ∧ z ≡ r)
  ∨ ( x ≡ []
    ∧ y ≡ h : t
    ∧ r ≡ h : s
    ∧ appendo t z s)
  ∨ ( x ≡ h : t
    ∧ r ≡ h : s
    ∧ double_app t y z s)
```

Текущее положение дел

- Попробовали реализовать несколько версий суперкомпиляции
  - Некоторые программы успешно преобразовываются
  - Некоторые преобразовываются с ухудшением производительности
  - На некоторых суперкомпиляция не завершается
- Стало понятно, *что* мы делали неправильно
- Решено *буквально* адаптировать для miniKanren решения для логического программирования

Дальнейшие планы

- Доведение до работоспособности конъюнктивной частичной дедукции
- Поддержка отношений высшего порядка
- “Негативная” суперкомпиляция
- Адаптация более мощных техник символьных вычислений