Анализ времени связывания для реляционных программ

Артемьева Ирина
Вербицкая Екатерина

SEIM-2020

15 мая 2020 г.

Реляционное программирование

- Программы реляционного программирования — математические отношения
- Можно исполнять в различных направлениях: зафиксировав часть аргументов, находить значение остальных
- $add^o \subseteq Int \times Int \times Int$ — отношение, третий аргумент которого является суммой первых двух ($?$ — выходной аргумент):
  - $add^o\ x\ y\ ?$ — сумма x и y
  - $add^o\ ?\ y\ z$ — разность z и y
  - $add^o\ ?\ ?\ z$ — пары чисел, сумма которых равна z
  - $add^o\ ?\ ?\ ?$ — все тройки отношения

Преимущество реляционного программирования

- Выбор направления: написав одну программу, получаем несколько целевых функций
- Следствие — решение задачи поиска посредством решения задачи распознавания
  - по интерпретатору языка и набору тестов синтезируем программу[1]
  - по предикату ”последовательность вершин в графе формирует путь желаемый путь” синтезируем генератор таких путей[2]

---

[1] W. E. Byrd, U. Ballantyne, U. Rosenblatt and M. Might, ”A unified approach to solving seven programming problems (functional pearl)”

[2] P. Lozov, E. Verbitskaia and D. Boulytchev, ”Relational interpreters for search problems”

Язык miniKanren

- Основной представитель парадигмы реляционного программирования
- Встраивается в языки общего назначения
  - проще использовать в своих проектах
  - для встраивания — реализовать небольшой интерпретатор: ядро языка на Scheme менее 40 строк[3]
- Все языковые конструкции обратимы
  - стратегия поиска решений interleaving — программа найдет все существующие решения
  - Prolog использует другую — операция cut необратима и делает невозможным выполнение по направлениям

[3] J. Hemann and D. P. Friedman, ”ukanren: A minimal functionalcore for relational programming”

Абстрактный синтаксис miniKanren

- Программа — определения отношений и цель
- Вызов другого отношения — по имени с аргументами-термами, используя $\underline{call}$.
- Свободные переменные вводятся $\underline{fresh}$.

$$
\begin{aligned}
Goal : & \; Goal \vee Goal \\
& \mid Goal \wedge Goal \\
& \mid Term \equiv Term \\
& \mid \underline{call}\ Name\ [Term] \\
& \mid \underline{fresh}\ [Var]\ Goal \\
Term : & \; Var \\
& \mid \underline{cons}\ Name\ [Term]
\end{aligned}
$$

Решение задачи поиска на miniKanren

- Реализовать на любом языке программирования N-местную функцию-распознаватель
- Автоматически транслировать её на miniKanren (получим $N+1$-местное отношение, связывающее аргументы функции с булевым значением).[4]
- Зафиксировав значение $N+1$-ого булевого аргумента, выполнить поиск.

[4] P. Lozov, E. Verbitskaia and D. Boulytchev, ”Relationalinterpreters for search problems”

Проблема скорости вычисления

- При написании программы подразумевается конкретное направление, называемое прямым; все другие — обратные
- Выполнение в обратном направлении обычно крайне неэффективно, а решение задачи поиска происходит именно так

Подходы к ускорению поиска

- Специализация[5]
  - Специализатор должен быть Джонс-оптимальным, чтобы избавиться от накладных расходов интерпретации.[6]
  - Его реализация — нетривиальная задача
- ***Трансляция***
  - По отношению с фиксированным направлением генерируется функция на функциональном языке программирования Haskell
  - Позволяет избежать затрат на интерпретацию

---

[5] P. Lozov, E. Verbitskaia and D. Boulytchev, ”Relational interpreters for search problems”

[6] N. D. Jones, C. K. Gomard, and P. Sestoft, ”Partial evaluationand automatic program generation”

Нормальная форма программы на miniKanren

Нормальной формой будем называть дизъюнкцию конъюнкций вызовов отношений или унификаций термов, в которой все свободные переменные введены в область видимости в самом начале

$$Goal : \underline{fresh}\ [Name]\ (\bigvee \bigwedge Goal')$$
$$Goal' : \underline{call}\ Name\ [Term]$$
$$\mid Term \equiv Term$$

- Любое отношение можно преобразовать в *нормальную форму*

Пример трансляции

Программа на miniKanren:

```
:: appendo x y xy = [h t r:
  (x === [] /\ xy === y) \/
  (x === h % t /\
   xy === h % r /\
   {appendo t y r})]
```

В обратном направлении выглядит на Haskell так:

```haskell
appendo xy = appendo1 xy ++ appendo2 xy
  where
    appendo1 y = do
      let x = []
      return (x, y)
    appendo1 _ = []
    appendo2 (h : r) = do
      (t, y) <- appendo r
      let x = h : t
      return (x, y)
    appendo2 _ = []
```

Проблемы трансляции

- Программа на miniKanren обладает свойствами, которые при трансляции в функциональный язык необходимо имитировать. Рассмотрим несколько таких свойств и способы их реализации.

Несколько выходных переменных

- Решение: объединим их в кортеж
- Пример: строка 8 в результате трансляции $append^o$ в обратном направлении

```
1  appendo xy = appendo1 xy ++ appendo2 xy
2    where
3      appendo1 y = do
4        let x = []
5        return (x, y)
6      appendo1 _ = []
7      appendo2 (h : r) = do
8        (t, y) <- appendo r
9        let x = h : t
10       return (x, y)
11     appendo2 _ = []
```

Пересечение результатов дизъюнктов

- Решение: каждый дизъюнкт транслируется во вспомогательную функцию; целевая функция — конкатенация их вызовов
- Пример: функции $appendo1$ и $appendo2$ в результате трансляции $append^o$ в обратном направлении

```
1  appendo xy = appendo1 xy ++ appendo2 xy
2    where
3      appendo1 y = do
4        let x = []
5        return (x, y)
6      appendo1 _ = []
7      appendo2 (h : r) = do
8        (t, y) <- appendo r
9        let x = h : t
10       return (x, y)
11     appendo2 _ = []
```

Недетеминированность результатов

- Решение: список как несколько вариантов одной переменной
- Пример: в результате трансляции $append^o$ в обратном направлении все вычисления происходят в монаде списка

```
1  appendo xy = appendo1 xy ++ appendo2 xy
2    where
3      appendo1 y = do
4        let x = []
5        return (x, y)
6      appendo1 _ = []
7      appendo2 (h : r) = do
8        (t, y) <- appendo r
9        let x = h : t
10       return (x, y)
11     appendo2 _ = []
```

Недетеминированность порядка вычислений

- Порядок исполнения отношений внутри целевого может отличаться для разных направлений
- Решение: анализ времени связывания $\rightarrow$ алгоритм аннотирования
- Пример: при трансляции второго дизъюнкта $append^o$ в обратном направлении порядок конъюнктов изменился

```
1 (x === h % t /\
2  xy === h % r /\
3  {appendo t y r})
```

```
1 appendo2 (h : r) = do
2   (t, y) <- appendo r
3   let x = h : t
4   return (x, y)
5 appendo2 _ = []
```

Анализ времени связывания

- Цель — указать порядок, в котором имена переменных связываются со значениями
- Позволит выявлять направления и порядок вычислений определений

Аналоги анализа времени связывания

- Используется при offline-специализации.[7]
  - какие данные известны статически и могут быть учтены при специализации
- Mercury.[8]
  - используется для эффективной компиляции
  - два типа аннотаций (статические и динамические) — нам недостаточно
  - выполняется с учётом графа типов — в miniKanren нет типов
- ***Лямбда-исчисление с функциями высшего порядка***.[9]
  - определяется порядок связывания переменных
  - типы аннотаций — $\{0, 1, \ldots, N\}$

[7] N. D. Jones, C. K. Gomard, and P. Sestoft, ”Partial evaluation and automatic program generation”

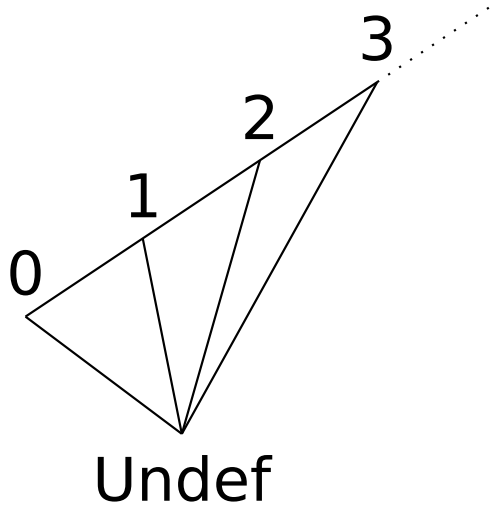
[8] W. Vanhoof, M. Bruynooghe and M. Leuschel, ”Binding-time analysis for mercury”

[9] P. Thiemann, ”Binding-time analysis for lambda calculus”

Аннотирование

- Алгоритм принимает на вход программу на miniKanren и имена входных переменных
- Каждой переменной ставится в соответствие целое положительное число, обозначающее время её связывания
- Процесс подбора чисел назовем *аннотированием*
- Изначально входные аннотируются помечаются 0, остальные — $Undef$ (время связывания неизвестно)

Свойства аннотаций



- На аннотациях выполняется естественный порядок на целыйх положительных числах ($Undef$ меньше любой числовой аннотации).
- Аннотация никогда не заменяется на меньшую

Пример аннотирования $append^o$

- $append^o$ в обратном направлении
- Число около переменной — её аннотация

```
:: append^o x^1 y^1 z^0 = [h, t, r:
  (x^1 === [] /\ y^1 === z^0) \/
  (x^3 === h^1 % t^2 /\
   z^0 === h^1 % r^1 /\
   {append^o t^2 y^2 r^1})]
```

- При трансляции в Haskell аннотации второго дизъюнкта дадут следующие определения:
  - $x = h : t$
  - $(h : r) = z$
  - $(t, y) = appendo\ r$

Аннотирование вызова

- Для избежания повторного аннотирования вызовов отношений в конкретных направлениях сохраним их в “стеке вызовов”
- Все термы вызова — $Undef$ или все термы — числа $\rightarrow$ вернуть исходную цель
- Существует ли вызов с таким именем и направлением в стеке вызовов?
  - Да $\rightarrow$ заменить $Undef$-аннотации аргументов вызова на $n + 1$, где n — максимальная аннотация аргументов
  - Нет $\rightarrow$ добавить его и направление в стек; проаннотировать тело вызываемого отношения; обновить стек

Алгоритм аннотирования

- Аннотировать дизъюнкт — аннотировать все его конъюнкты
- Переменная в конъюнктах одного дизъюнкта должна иметь одну аннотацию — согласованность аннотирования
- Аннотировать конъюнкт — аннотировать унификацию или вызов отношения

Аннотирование унификации

Случаи в зависимости от типа термов-участников унификации (x — переменная, $t[\dots]$ — терм):

- $x^{Undef} \equiv t[y_0^{i_0}, \dots, y_k^{i_k}] \rightarrow$ аннотация x стоановится $n+1$, где $n = max\{i_0, \dots i_k\}$.
- $x^n \equiv t[y_0^{i_0}, \dots, y_k^{i_k}]$, некоторые свободные переменные терма t проаннотированы $Undef \rightarrow$ переменным y_j^{Undef} присваивается аннотация $n+1$.
- $\underline{cons}\ Name\ [t_0^{i_0}, \dots, t_k^{i_k}] \equiv \underline{cons}\ Name\ [s_0^{j_0}, \dots, s_k^{j_k}] \rightarrow$ унификация эквивалентна конъюнкции унификаций вида $t_l^{i_l} \equiv s_l^{j_l}$, каждую из которых следует анализировать в соответствии с одним из перечисленных случаев
- Остальные случаи симметричны

Несколько вызовов в дизъюнкте (1)

Последовательность нескольких вызовов влияет на направления их трансляции.
Пример: пусть y — входная переменная; порядок вычисления f^o и h^o не зависит друг от друга, но зависит от направления g^o:

```
1   rel^o x y z =
2     f^o x y ∧
3     h^o z y ∧
4     g^o x z
```

- g^o не может вычисляться до вычисления f^o и h^o (неизвестны входные переменные)
- может вычисляться между ними (в прямом или обратном порядке)
- может вычисляться после (и выполнять роль предиката)

Несколько вызовов в дизъюнкте (2)

- При получении такого отношения алгоритм аннотирования возвращает частично проаннотированную цель (содержит $Undef$-аннотацию)
- Запустим алгоритм еще раз, изменив порядок вызовов
- Если и при другом порядке в аннотированной цели останутся $Undef$-переменные, цель считается неаннотируемой

Терминируемость алгоритма аннотирования

- Повторное аннотирование отношений не производится
- Имеющиеся в стеке вызовов отношения не аннотируются снова, а в каждом отношении используется конечное количество уникальных переменных
- В итоге — каждому отношению можно сопоставить конечное количество уникальных аннотаций
- Для случая нескольких вызовов в дизъюнкте — количество перестановок вызовов конечно $\rightarrow$ конечно количество запусков алгоритма

Результаты

- В работе представен алгоритм анализа времени связывания для miniKanren — он определяет порядок, в котором связываются переменные данного отношения с учётом направления его вычисления
- Его недостаток — при существовании нескольких вызовов отношений в одном дизъюнкте необходимо осуществлять полный перебор возможных направлений вычислений отношений
- По проаннотированной программе можно получить направления и порядок вычисления отношений — алгоритм аннотирования успешно используется в алгоритме трансляции