Уменьшение цены абстракции при типобезопасном встраивании реляционного языка программирования в OCaml

Дмитрий Косарев
Dmitrii.Kosarev@protonmail.ch

Санкт-Петербургский государственный университет
Математико-механический факультет
16 марта 2017 г.

Аннотация

В данной работе затронуты детали OCanren — типобезопасной реализации реляционного языка программирования из семейства miniKanren на OCaml, а именно, два подхода для организации типов логических значений. Первый, наивный, подход позволил получить типобезопасную реализацию, однако привел к понижению производительности на стадии унификации по сравнению с реализацией miniKanren на Racket. Второй подход не страдает этим недостатком.

Ключевые слова: OCaml, miniKanren, унификация, цена абстракции.

Реляционный язык программирования miniKanren позволяет записывать программы как формулы, состоящие из отношений (relations). Относительная простота miniKanren привела к появлению целого семейства реализаций, большинство из которых были сделаны либо для различных диалектов LISP, либо на типизируемых языках в бестиповой манере. OCanren является типизированной реализацией miniKanren на OCaml — языке программирования со строгой статической типизацией.

В OCanren разрешается унифицировать между собой только те логические переменные и значения, которые инкапсулируют значения одинакового типа. Это позволяет находить на стадии компиляции случаи, при которых унифицируются значения разных типов, и, следовательно, унификация не может завершиться успешно ни при каких условиях. Также компилятор, теоретически, может генерировать вызов специализированной для конкретного типа унификации вместо обобщенной.

При создании новой реализации miniKanren необходимо обдумать два аспекта: в каком порядке будут вычисляться результаты (порядок поиска), и как именно будет происходить процесс унификации. В OCanren была реализована истинно полиморфная унификация: единая (для всех типов агу-

ментов) функция позволяет унифицировать любые значения одинаковых типов. Унификация производится путём сравнения ориентированных графов (в случае miniKanren — деревьев), т.к. все значения OCaml хранятся в памяти именно в таком виде.

Однако наивный подход, связанный с объявлением алгебраического типа логических значений `'a logic`, который содержит в себе либо логические переменные (конструктор `Var`), либо обычные значения (конструктор `Value`), приводит к тому, что размер представлений (деревьев) значений в памяти увеличивается. Например, целые числа типа `int` хранятся в виде одного блока памяти. Логическое представление для этих чисел будет состоять из двух блоков памяти: один для конструктора `Value` и один непосредственно для числа, тем самым увеличивая высоту дерева представления в два раза: с единицы до двух.

Для списков (и других *рекурсивных* структур данных) высота деревьев также увеличивается в два раза. Например, список чисел из одного элемента будет занимать в памяти три блока и образовывать дерево высотой два. Логический эквивалент такого же списка будет представляться в памяти деревом размером 6 и высотой 4, что будет сказываться на производительности унификации. Таким образом, общая производительность OCanren оказывается в разы меньше чем изначальный вариант.

Альтернативный подход не использует алгебраических типов данных при объявлении типа логических значений (если говорить упрощенно, то `type 'a injected = 'a`). Таким образом, не появляется дополнительных блоков памяти в представлении данных, размер деревьев не увеличивается, и производительность унификации увеличивается по сравнению с первым случаем. Данный подход требует некоторых преобразований получившихся значений, если они содержат в себе свободные логические переменные. Эти преобразования, однако, происходят один раз в конце вычислений и влияют на производительность существенно меньшим образом. Также подход требует специфической работы на уровне системы типов, а именно использования примитивов для конструирования логических значений.

```
val lift: 'a -> ('a,'a) injected
val inj: ('a, 'b) injected -> ('a, 'b logic) injected
val distribute: ('a,'b) injected t -> ('a t, 'b t) injected
```

Первые два примитива универсальны, последний вид примитивов не объявлен заранее для каждого вида `t`, его можно сгенерировать, если тип `t` представим как функтор.

Список литературы

[1] Jason Hemann, Daniel P. Friedman. μKanren: A Minimal Core for Relational Programming // Proceedings of the 2013 Workshop on Scheme and Functional Programming (Scheme '13).

[2] Dmitrii Kosarev, Dmitri Boulytchev. Typed Embedding of a Relational Language in OCaml // Workshop on ML, 2016.