

Уменьшение цены абстракции при типобезопасном встраивании реляционного языка программирования в OCaml.

Дмитрий Косарев
Dmitrii.Kosarev@protonmail.ch

Санкт-Петербургский государственный университет
Математико-механический факультет
28 января 2017 г.

Аннотация

В данной работе затронуты детали OCamlgen – 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, он привел к понижению производительности на стадии унификации (unification) по сравнению с реализацией μ Kanren на Racket. Второй подход не страдает этим недостатком.

Ключевые слова: OCaml, miniKanren, реляционное программирование, 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 на различных диалектах LISP, либо на типизируемых языках в бестиповой манере. OCamlgen[2] является типизированной реализацией μ Kanren[1] (одной из наиболее минималистичных реализаций miniKanren) на OCaml – языке программирования со строгой статической типизацией.

В OCamlgen 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 поис-

ка), и как именно будет происходить процесс унификации. В OCaml была реализована истинно полиморфная унификация: одна 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, 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: для конструктора *Value* и 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 OCaml оказывается в разы меньше чем изначальный вариант.

Другой испробованный подход не использует конструктор *Value* для создания логических значений, хотя это требует больше небезопасных приведений типов в ядре OCaml. Но при этом OCaml не требует небезопасных приведений типов в клиентском коде, что позволяет писать в целом типобезопасные программы. Данный подход требует некоторых преобразований получившихся значений, если они содержат в себе свободные логические переменные. Однако, эти преобразования происходят один раз в конце вычислений и влияют на производительность существенно меньшим образом. Таким образом, при втором подходе не появляется дополнительных блоков памяти в представлении данных, следовательно размер деревьев не увеличивается, производительность унификации увеличивается по сравнению с первым случаем, а цена абстракции при переписывании кода на типизированный язык уменьшается.

Список литературы

- [1] Jason Hemann, Daniel P. Friedman. μ Kanren: 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.