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

Дмитрий Косарев
Dmitrii.Kosarev@protonmail.ch

Санкт-Петербургский государственный университет
Математико-механический факультет
27 января 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, он привел к понижению производительности на стадии унификации (unification) про сравнению с реализацией μKanren 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 их которых были сделаны на различных диалектах LISP, либо на типизируемых языках в бестиповой манере. OCanren является типизированной реализацией μKanren 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: 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. Рассмотрим список целых чисел, содержащий один элемент: он состоит из трех блоков памяти (число, *cons*-блок и блок для пустого списка), которые образуют дерево высотой два. Переход к логическим значениям типа int добавит блок для конструктора $Value$ между *cons*-блоком и блоком числа. Затем, переход к логическому эквиваленту списков, во-первых, добавит $Value$-блок, который будет ссылаться на *cons*-блок и, во-вторых, добавит $Value$-блок между *cons*-блоком и хвостом списка, так как второй потомок *cons*-блока должен указывать на значения типа "логического" списка. Таким образом, переход к логическому представлению превращает дерево размером 3 и высотой 2 в дерево размером 6 и высотой 4, что сказывается на производительности унификации. Таким образом, общая производительность OCanren оказывается в разы меньше чем изначальный вариант.

Другой испробованный подход не использует конструкторы алгебраических типов данных при объявлении типа логических значений. Таким образом, не появляется дополнительных блоков памяти в представлении данных, следовательно размер деревьев не увеличивается, а производительность унификации увеличивается по сравнению с первым случаем. 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.