Санкт-Петербургский государственный университет

Кафедра Системного программирования

Группа 21.Б07-мм

Визуализация встроенного в OCaml языка miniKanren

Каримов Тимур Рифатович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры системного программирования Д. С. Косарев

Санкт-Петербург
2023

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор **5**

2.1. miniKanren 5

2.2. Обзор решений 6

3. Используемые инструменты **7**

3.1. OCaml 7

3.2. Qt 7

4. Реализация **8**

4.1. Недостатки первоначальной версии 8

4.2. Архитектура решения 8

4.3. Вид дерева 9

5. Дальнейшие планы **10**

5.1. Тестирование 10

5.2. Построение по ходу выполнения 10

5.3. Изменение вида построения дерева 10

6. Результаты **11**

Список литературы **12**

Введение

Реляционное программирование – это вид декларативного программирования, в котором программы представляются как набор отношений. Отношения не делают различий между входными и выходными параметрами. В силу этой особенности одна и та же реляционная программа может использоваться для решения нескольких связанных проблем. Одним из представителей реляционной парадигмы является язык программирования miniKanren [1]. Он представляет собой небольшое число примитивов. Благодаря этому его можно легко реализовать в виде встраиваемого предметно-ориентированного языка. Примером этого является unicanren[1] — минимальная реализация miniKanren, написанная на функциональном языке OCaml [8].

Язык miniKanren относится к семейству реляционных языков программирования, использующему стратегию поиска с чередованием (interleaving search). Её суть заключается в том, что процесс поиска перемещается с одного узла дерева на другой и тем самым углубляет посещенную часть его, производя поиск. Из преимуществ можно выделить то, что это поиск полон, в отличие от поиска в глубину, и работает быстрее, чем банальный поиск в ширину. Из недостатков: сложно следить за исполнением программы с помощью трассировки, потому что контекст текущего исполнения постоянно меняется.

Решением этой проблемы может выступить визуализация дерева поиска по мере выполнения программы. С помощью него явно можно будет отследить текущий шаг и понять контекст исполнения. Оно так же позволит начинающим программистам познакомиться со стратегией поиска с чередованием, путём наглядной демонстрации хода работы программы.

В рамках данной работы предлагается разработать решение, визуализирующее miniKanren с целью повышения удобства отладки и демонстрации стратегии поиска с чередованием.

[1] https://github.com/Kakadu/unicanren (Дата доступа: 2022-12-11)
Дата сборки: 4 января 2023 г.

1. Постановка задачи

Целью работы является создание визуализатора минимальной реализации miniKanren. Для её выполнения были поставлены следующие задачи:

1. выбрать платформу визуализации;
2. реализовать решение, визуализирующее miniKanren.

2. Обзор

В данном разделе будут рассмотрены существующие наработки в области визуализации miniKanren.

2.1. miniKanren

Программы на miniKanren пишутся с помощью реляций [6]. В данной работе используется unicanren — минимальная реализацию miniKanren, состоящую из 4 базовых конструкций: Fresh — объявления новой переменной, Unify — присваивания, Cond^e — дизъюнкции и Conj — конъюнкции. Так выглядит каноническая реализация append^o, конкатенации списков, на Scheme [2]:

Листинг 1: **Каноническая реализация append^o на Scheme**

```
(define (appendo l s out)
  (conde
    [(== l '()) (== s out)]
    [(fresh (a d res)
       (== `(,a . ,d) l)
       (== `(,a . ,res) out)
       (appendo d s res))]))
```

Далее приведу примеры использования базовых реляций в unicanren на примере дерева абстрактного синтаксиса отношения append^o:

Листинг 2: **Дерево абстрактного синтаксиса отношения append^o**

```
  Conde
    [ Conj [ Unify (Var "xs", Nil); Unify (Var "ys", Var "xys") ]
    ; Fresh
      ( "h"
     , Fresh
         ( "tmp"
        , Fresh
          ( "tl"
```

```
        , Conj
          [ Unify (Cons (Var "h", Var "tl"), Var "xs")
          ; Unify (Cons (Var "h", Var "tmp"), Var "xys")
          ; Call ("appendo", [ Var "tl"; Var "ys"; Var "tmp" ])
          ] ) ) )
  ]
```

2.2. Обзор решений

Существует профайлер для miniKanren [7]. Он выполняет программу, подсчитывая количество выполненных унификаций, и раскрашивает строчки кода в зависимости от того, как часто они исполняются. Моё решение не считает унификации, а строит дерево поиска, но идеологически реализации схожи. Более явных аналогов не существует.

3. Используемые инструменты

3.1. OCaml

Для данного решения была выбрана реализация miniKanren, встроенная в OCaml версии 4.14.0 [4]. Выбор OCaml так же обусловлен существованием Js_of_ocaml[2] — компилятора байт-кода OCaml для JavaScript. Он позволяет запускать программы OCaml в среде JavaScript.

3.2. Qt

В качестве платформы визуализации был выбран Qt[3] — фреймворк для разработки кроссплатформенного программного обеспечения [5]. Входящий в его состав QML предоставляет полную свободу действий при создании пользовательского интерфейса. QML — декларативный язык программирования, использующийся для разработки приложений, делающих основной упор на дизайн графической части. Язык QML использует синтаксис, подобный JSON, и позволяет определять различные выражения и методы как функции JavaScript. Это также позволяет пользователям импортировать файлы JavaScript и использовать функциональность, предоставляемую этим импортом [3].

[2] https://ocaml.org/p/js_of_ocaml/4.0.0 (Дата доступа: 22-12-14)
[3] https://www.qt.io/ (Дата доступа: 2022-12-12)

4. Реализация

Данная работа является развитием предыдущей попытки сделать визуализатор miniKanren и основывается на её наработках[4]. Подробнее об исправлениях, внесённых в первоначальную версию, будет сказано ниже.

4.1. Недостатки первоначальной версии

В первоначальной реализации некорректно работали кнопки вывода дерева как по шагам, так и в целом.

Программа неверно отображала переменные, выводя их имена, а не значения.

В некоторых функциях интерпретатора, передающих информацию для построения дерева, некорректно были заданы аргументы, из-за чего помимо имён переменных или названий реляций в узлы дерева передавалась ненужная информация.

4.2. Архитектура решения

4.2.1. Компиляция в JS

OCaml компилируется в JavaScript с помощью компилятора байт-кода для Js_of_ocaml.

4.2.2. Qt вызывает miniKanren

Отдельное приложение на Qt во время выполнения вызывает miniKanren, используя скомпилированный файл, для чего пришлось модифицировать unicanren.

4.2.3. Функции построения

В функциях интерпретатора вызываются функции для построения дерева, использующие код на JavaScript.

[4] https://github.com/Kakadu/miniKanren-vizualizer (Дата доступа: 22-12-14)

```
main.bc.js                                caml_ba_create_buffer(kind, size)
911        h ^= h >>> 16;
912        return h}
913      function caml_is_ml_bytes(s){return s instanceof MlBytes}
914      function caml_is_ml_string(s){return caml_is_ml_bytes(s)}
915      function caml_hash(count,limit,seed,obj)
916       {var queue,rd,wr,sz,num,h,v,i,len;
917        sz = limit;
918        if(sz < 0 || sz > 256)sz = 256;
919        num = count;
920        h = seed;
921        queue = [obj];
922        rd = 0;
923        wr = 1;
```

Рис. 1: main.bc.js — скомпилированный в JavaScript OCaml

Листинг 3: **Одна из модификаций интерпретатора — последние две строчки здесь специфичны для визуализатора**

```
| Unify (l, r) →
  let* l = eval_term l in
  let* r = eval_term r in
  Js.Unsafe.global##.unifyf
    (Js.string (asprintf "%a and %a" Value.pp l Value.pp r));
```

4.3. Вид дерева

Приведу объяснение прикреплённого рисунка 2.

- Корень дерева находится в левом верхнем углу.
- Вызвалась реляция append^o.
- За ней следует дизъюнкция, за ней — конъюнкция.
- Отдельная ветка — попытка унификации.
- Отдельная ветка — введение новых переменных.

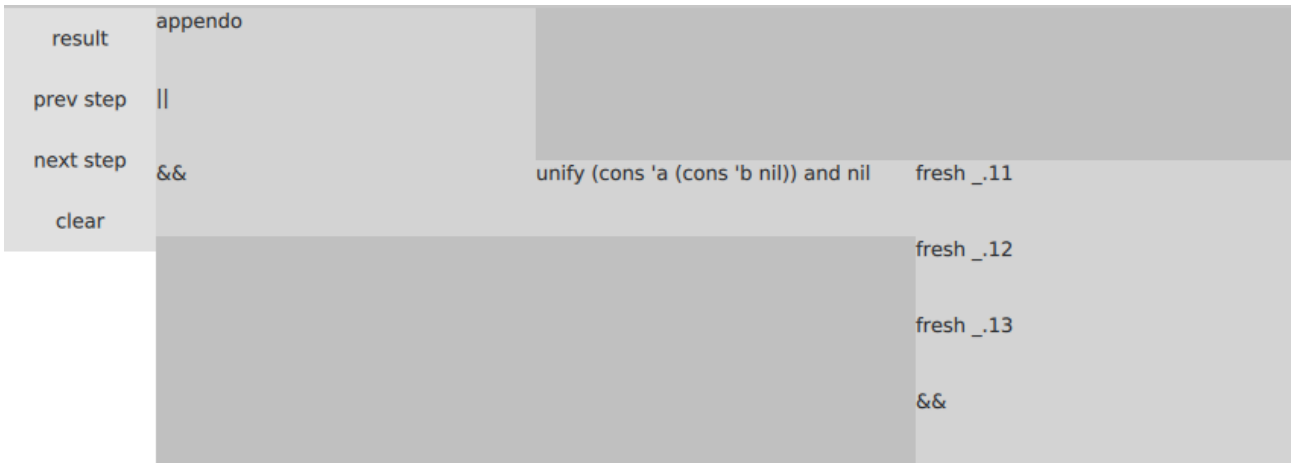

Рис. 2: Часть дерева в текущем виде

5. Дальнейшие планы

Также был составлен план дальнейших действий.

5.1. Тестирование

Пока программа тестировалась только для append^o — одной из самых простых содержательных реляции в miniKanren. В будущем нужно протестировать визуализатор на бóльшем количестве реляционных программ.

5.2. Построение по ходу выполнения

Сейчас дерево строится после завершения программы. Хочется реализовать построение по мере выполнения, чтобы в случае ошибки дерево бы построилось хотя бы до некоторого шага.

5.3. Изменение вида построения дерева

На данный момент дерево строится не в самом понятном виде. Не ясно, где какая ветка, что от чего наследуется. Ведётся работа по улучшению визуальной составляющей с целью повышения уровня понимания выводимой информации.

6. Результаты

На данный момент выполнены следующие задачи:

- выбран Qt в качестве платформы визуализации;
- начата работа над визуализатором miniKanren:[5]
  - реализован корректный вывод переменных и реляций;
  - исправлены кнопки, позволяющие выводить дерево целиком, по шагам и стирать содержимое;
  - исправлены функция вывода дерева и полоса прокрутки в приложении.

[5] https://github.com/Rifatovich/miniKanren-vizualizer (Дата доступа: 2022-12-12)

Список литературы

[1] Byrd William E., Holk Eric, Friedman Daniel P. miniKanren, Live and Untagged. — URL: http://webyrd.net/quines/quines.pdf (дата обращения: 2022-12-14).

[2] Friedman Daniel P., Byrd William E., Kiselyov Oleg. The Reasoned Schemer. — The MIT Press, 2005.

[3] Integrating QML and JavaScript. — URL: https://doc.qt.io/qt-6/qtqml-javascript-topic.html (дата обращения: 2022-13-22).

[4] Minsky Yaron, Madhavapeddy Anil, Hickey Jason. Real World OCaml - Functional Programming for the Masses. — O'Reilly, 2013.

[5] Neto Manoel C.M., Andrade Sandro S., Novais Renato L. Cross-Platform Multimedia Application Development: For Mobile, Web, Embedded and IoT with Qt / QML // Proceedings of the 23rd Brazilian Symposium on Multimedia and the Web. — WebMedia '17. — New York, NY, USA : Association for Computing Machinery, 2017. — P. 23–26. — URL: https://doi.org/10.1145/3126858.3131627 (дата обращения: 2022-12-14).

[6] A Unified Approach to Solving Seven Programming Problems (Functional Pearl) / William E. Byrd, Michael Ballantyne, Gregory Rosenblatt, Matthew Might // Proc. ACM Program. Lang. — 2017. — aug. — Vol. 1, no. ICFP. — 26 p. — URL: https://doi.org/10.1145/3110252 (дата обращения: 2022-12-14).

[7] miniKanren Profiler / Sloan Chochinov, Daksh Malhotra, Gregory Rosenblatt et al. — URL: https://icfp22.sigplan.org/details/minikanren-2022-papers/4/Fail-Fast-and-Profile-On-Towards-a-miniKanren-Profiler (дата обращения: 2022-12-11).

[8] Моисеенко Е. А., Подкопаев А. В. Реляционное программирование с мемоизацией и отрицанием // Научно-технические ведомо-

сти СПбГПУ. Информатика. Телекоммуникации. Управление. 2018. Т. 11. № 2. С. 35–46. DOI: 10.18721/JCSTCS.11203. — URL: https://infocom.spbstu.ru/userfiles/files/articles/2018/2/2018_2_03.pdf (дата обращения: 2022-12-11).