Реализация сравнения различных представлений λ-термов

студент: Жаворонков Эдгар

научный руководитель: В.И. Исаев

13 июня 2017 г.

Введение

1. Программирование использует имена для того, что бы идентифицировать сущности
2. Как следствие, в программах зачастую возникает проблема коллизий имен
3. Чтобы доказывать свойства программ нужна формальная система, которая бы умела решать эту проблему
4. Пример такой системы – λ-исчисление

Введение(2)

1. Термы λ-исчисления имеют несколько способов записи – представлений
2. Какие-то удобнее для компьютерной реализации, какие-то для рассуждений «на бумаге»
3. В целом, задача формализации лямбда-исчисления довольно интересна и популярна
4. Но существующие работы в этой области, как правило, формализуют какое-то одно представление

Введение(3)

1. Тонкость заключается в том, что именованные термы рассматриваются с точностью до α-эквивалентности
2. То есть мы рассматриваем фактор-множество и нам хотелось бы уметь удобно описывать их с помощью языка программирования
3. Возникает вопрос, есть ли язык(теория типов), который позволяет просто конструировать фактор-типы?

Введение(3)

1. Тонкость заключается в том, что именованные термы рассматриваются с точностью до α-эквивалентности
2. То есть мы рассматриваем фактор-множество и нам хотелось бы уметь удобно описывать их с помощью языка программирования
3. Возникает вопрос, есть ли язык(теория типов), который позволяет просто конструировать фактор-типы?
4. Да, есть. Он называется Vclang[1] и его теоретическая основа – гомотопическая теория типов с типом интервала

[1] https://github.com/valis/vclang

Цель и задачи

Цель работы – показать равенство между различными представлениями λ-термов.

Задачи:

1. Реализовать типы данных для термов в интересующих нас представлениях:
    1. Именованном
    2. Неименованным
    3. Монадическом
2. Для каждого представления реализовать операцию подстановки
3. Формализовать её свойства:
    1. Унитальность
    2. Ассоциативность
4. Построить эквивалентности между описанными в п.1 представлениями

Существующие решения

1. Задача формализации λ-исчисления довольно популярна
2. Очень много работ посвящено формализации типизированных вариаций λ-исчисления
3. Во многих работах авторы формализуют неименованное представление
4. Есть работы, в которых авторы сравнивают именованное и неименованное представление, но не устанавливают, что они равны

Решение. Именованное представление

Тип данных для термов:

```haskell
data Term =
    Var Name
  | App Term Term
  | Lam Name Term
```

1. Нужно разрешимое равенство на *Name*
2. α-эквивалентность определяется индукцией по структуре терма плюс некоторый трюк в случае абстракции
3. Подстановка определяется через более общий случай – параллельную подстановку

Решение. Именованное представление(2)

1. Все утверждения о подстановке рассматривают термы с точностью до α-эквивалентности
2. Свойства подстановки выглядят следующим образом:

$$
\begin{aligned}
x[x \mapsto t] &=_\alpha t \\
t[x \mapsto x] &=_\alpha t \\
t[x \mapsto N][y \mapsto M] &=_\alpha t[y \mapsto M][x \mapsto N[y \mapsto M]](x \notin FV(M))
\end{aligned}
$$

3. Доказываются очень нудной индукцией по структуре терма t

Решение. Неименованное представление

Термы:

```
data Term (n : Nat) =
    Var (i : Fin n)
    | App (Term n) (Term n)
    | Lam (Term (suc n))
```

1. Здесь *n* – длина контекста, в котором определен терм, а *i* – индекс переменной в нем
2. Подстановка в таком представлении – полная(во все переменные)
3. Так как нет имен переменных, то и определяется она намного проще

Решение. Монадическое представление

Термы:

```
data Term (V : Set) : Set =
     Var V
     | App (Term V) (Term V)
     | Lam (Term (Maybe V))
```

1. Нетрудно заметить, что это функтор
2. Чуть менее очевидно, но это монада
3. fmap – переименовывает переменные в терме, а $(>>=)$ – это, внезапно, подстановка
4. Монадические законы в точности описывают свойства подстановки

Решение. Монадическое представление(2)

1. Чтобы доказать, что это монада заметим, что есть два способа определить $>>=$
   1. По стрелке Клейсли $k : V \to Term\ W$ построить стрелку Клейсли $k : (V + 1) \to Term\ (W + 1)$
   2. Обобщить сигнатуру $>>= : Term\ V \to (V \to Term\ W) \to Term\ W$ до $>>=' : Term\ (V + n) \to (V \to Term\ W) \to Term\ (W + n)$
2. Второй менее удобен, так как всплывают взаимно-рекурсивные определения, с которыми неудобно работать.

Решение. Эквивалентности между представлениями

1. Чтобы преобразовать именованный терм в неименованный нам не обойтись без контекста и доказательства, что терм определен в контексте
2. Обратно, нам не очень важно, в каком контексте будет определен результат, поэтому его можно сгенерировать
3. Для монадических и неименованных термов эквивалентность показывается практически «в лоб»
4. Имея эквивалентность, мы получаем буквальное равенство между типами, которым можно пользоваться при дальнейших доказательствах

Результаты

1. Построены биекции:
    1. Между именованными термами и неименованными
    2. Между неименованными термами и монадическими
2. Для именованных термов описана α-эквивалентность
3. Доказано, что преобразование именованного терма в неименованный уважает α-эквивалентность
4. Для каждого представления определена операция подстановки
5. Для неименованного и монадического представления полностью формализованы свойства подстановки
    1. Унитальность
    2. Ассоциативность
6. Для именованного представления мы столкнулись с некоторыми проблемами

fin

Спасибо за внимание!
Вопросы?

Github repo:

https://github.com/edgarzhavoronkov/vclang-lib/tree/lambda_calculus/test/LC