Российская Академия наук
Санкт-Петербургский академический университет —
Научно-образовательный центр нанотехнологий

Санкт-Петербургская кафедра философии РАН

История и методология механизмов элиминации в языках с зависимыми типами

Реферат аспиранта СПБАУ-НОЦНТ
Шабалина Александра Леонидовича

Научный руководитель
к.ф.-м.н., доцент
Москвин Денис Николаевич

Руководитель аспирантской группы
д.ф.н., проф.
Мангасарян Владимир Николаевич

Санкт-Петербург
2015

Содержание

1. Лямбда-исчисление с зависимыми типами

1.1. Нетипизированное лямбда-исчисление

Лямбда-исчисление Чёрча, наряду с машинами Тьюринга и общерекурсивными функциями Гёделя, задает множество эффективно вычислимых функций [1]. То есть функций, для вычисления которых существует алгоритм. Это, в свою очередь, означает, что любая программа, исполняемая на современных вычислительных устройствах, должна быть выразима на языке лямбда-исчисления. Формально этот язык задается следующим образом [1]:

- Есть некоторое множество *переменных* V.
- Есть множество *термов* T, задаваемых индуктивно:

$$T = \begin{cases} x, & x \in V & \text{переменная} \\ M\ N, & M, N \in T & \text{применение} \\ \lambda x.M, & x \in V, M \in T & \text{лямбда-абстракция} \end{cases}$$

- Множеством свободных переменных терма t называется подмножество V, в которое входят все переменные в терме t, кроме тех, что связаны лямбда-абстракцией (определение аналогично для других связывающих конструкций в математике, к примеру, кванторов существования и всеобщности):

$$\begin{aligned} FV(x) &= \{x\} \\ FV(MN) &= FV(M) \cup FV(N) \\ FV(\lambda x.M) &= FV(M) \setminus \{x\} \end{aligned}$$

- Вычисление выполняется с помощью правила бэта-редукции:

$$(\lambda x.M)N \Rightarrow_\beta M[x := N]$$

  Где нотация $M[x := N]$ означает замену в M всех свободных вхождений переменной x на терм N.

1.2. Просто типизированное лямбда-исчисление

Недостаток нетипизированного лямбда-исчисления лежит во «вседозволенности» операции применения MN: ничто не гарантирует, что M

будет функцией. Чтобы решить эту проблему, Чёрч разработал просто типизированное лямбда-исчисление [2]:

- Множество переменных на уровне термов V и на уровне типов TyV.
- Множество типов Ty:

$$Ty = \begin{cases} x, & x \in TyV \\ \sigma \to \tau, & \sigma, \tau \in Ty \end{cases}$$

- Множество термов T:

$$T = \begin{cases} x, & x \in V \\ M\ N, & M, N \in T \\ \lambda x : \sigma.M, & x \in V, \sigma \in Ty, M \in T \end{cases}$$

- Правила присваивания типов:

$$\begin{aligned} &\Gamma \vdash x : \sigma, \quad (x : \sigma) \in \Gamma \\ &\Gamma \vdash MN : \sigma, \quad \Gamma \vdash M : \tau \to \sigma, \Gamma \vdash N : \tau \\ &\Gamma \vdash (\lambda x : \sigma.M) : \sigma \to \tau, \quad (\Gamma \cup (x : \sigma)) \vdash M : \tau \end{aligned}$$

- Вычисление выполняется также с помощью бэта-редукции, имеющему аналогичный вид нетипизированному случаю. Но определяется только для термов, которым можно присвоить тип.

1.3. Другие способы типизирования

Недостаток просто типизированного лямбда-исчисления обратный: множество термов, которые хотелось бы написать будут отвергнуты системой типов. Поэтому было придумано множество способов расширить систему типов, чтобы с одной стороны позволить писать больше программ, которые хочется, а с другой — отвергать некорректные. Одной из таких систем типов является System F Жирара [6]:

- Типы

$$Ty = \begin{cases} x, & x \in TyV \\ \sigma \to \tau, & \sigma \in Ty, \tau \in Ty \\ \forall x.\sigma, & x \in TyV, \sigma \in Ty \end{cases}$$

- Термы

$$T = \begin{cases} x, & x \in V \\ M\ N, & M, N \in T \\ \lambda x : \sigma.M, & x \in V, \sigma \in Ty, M \in T \\ \Lambda x.M, & X \in TyV, M \in T \quad \text{абстракция по типу} \\ M[\sigma], & M \in T, \sigma \in Ty \quad \text{применение типа} \end{cases}$$

- Типизация. Все правила из просто типизированного плюс:

$$\Gamma \vdash \Lambda x.M : \forall X.\sigma, \quad \Gamma \cup \{x\} \vdash M : \sigma$$
$$\Gamma \vdash M[\sigma] : \tau[x := \sigma], \quad \Gamma \vdash M : \forall x.\tau$$

Введение абстракции по типам позволяет строить утверждения о программах [8]. Например, существует ровно один терм с типом $\forall a.\forall b.a \to b \to a$ — $\Lambda a.\Lambda b.\lambda x.\lambda y.x$.

1.4. Зависимые типы

Развивая идею построения утверждения по программам, можно придти к соответствию Карри-Говарда между типами и теоремами в математической логике [7]. В частности, System F соответствует интуиционистской (конструктивной) пропозициональной логике второго порядка (но без квантора существования).

Если теперь вместо пропозициональной логики взять логику предикатов и построить по ней систему типов, то получатся так называемые зависимые типы [4]. Ключевая идея в том, что мы объединяем множества типов и термов в одно. Позволяя таким образом писать термы на уровне типов. Это открывает возможность для проведения формальной верификации программ с помощью одной лишь системы типов.

2. Индуктивные типы данных

2.1. Расширение лямбда-исчисления индуктивными типами данных

TODO: Из [6] ?

2.2. Сопоставление с образцом как альтернатива элиминаторам

TODO: Из [3]

2.3. Зависимые индуктивные типы данных

TODO: Calculus of Inductive Constructions
TODO: UTT

3. Механизмы элиминации для зависимых индуктивных типов данных

TODO: Начиная с [5].

4. Список литературы

[1] H. P. Barendregt. *The Lambda Calculus Its Syntax and Semantics*, volume 103. North Holland, revised edition, 1984.

[2] Henk Barendregt, S. Abramsky, D. M. Gabbay, T. S. E. Maibaum, and H. P. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1992.

[3] Rod M Burstall. Proving properties of programs by structural induction. *The Computer Journal*, 12(1):41–48, 1969.

[4] P. Martin-Löf. An intuitionistic theory of types: predicative part. In *Logic Colloquium*, 1973.

[5] CONOR MCBRIDE and JAMES MCKINNA. The view from the left. *Journal of Functional Programming*, 14:69–111, 1 2004.

[6] Benjamin C. Pierce. *Types and Programming Languages.* MIT Press, Cambridge, MA, USA, 2002.

[7] Philip Wadler. Propositions as types.

[8] Philip Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture*, pages 347–359. ACM Press, 1989.