Введение в лямбда-процессоры

Николай Пономарев

Математико-механический факультет СПбГУ

18 марта 2024 г.

Самый главный слайд

Дисклеймер

Строгости изложения не будет!

План

Главная цель

Посмотреть на лямбда-процессоры взглядом системного программиста

По пути посмотрим на то

- Как работает «обычный» процессор
- Откуда вырастает идея лямбда-процессора
- «Ассемблеры» для лямбда-процессора
  - Их много!
- Что люди в мире уже делали

Мотивировка

- Будем считать, что ЯП делятся только на две парадигмы:
  - Императивную
  - Функциональную
- На самом деле больше, иначе непонятно куда девать Prolog, APL, miniKanren и иже с ними
- Но разберемся как исполнять программы с точки зрения процессора только в этих двух парадигмах

Исполнение императивных языков I

- Про языки с виртуальной машиной говорить сложно
- Возьмём «простой» язык — Си — и посмотрим как его будет исполнять процессор

Исполнение императивных языков II

```c
uint64_t factorial(int n)
{
    uint64_t result = 1;
    for (uint64_t i = 1; i <= n; ++i)
        result *= i;
    return result;
}
```

Листинг: Факториал на языке Си

```asm
factorial:
        beqz    a0, .LBB0_6
        addi    a1, a0, 1
        li      a0, 2
        bltu    a0, a1, .LBB0_3
        li      a1, 2
.LBB0_3:
        li      a2, 1
        li      a0, 1
.LBB0_4:
        mul     a0, a2, a0
        addi    a2, a2, 1
        bne     a1, a2, .LBB0_4
        ret
.LBB0_6:
        li      a0, 1
        ret
```

Листинг: Факториал на Ассемблере RISC-V

Исполнение императивных языков III

- Дальше только машинные коды, но нам ни к чему
- Исполнять такой код очень просто!

Исполнение императивных языков III

- Дальше только машинные коды, но нам ни к чему
- Исполнять такой код очень просто!
- **А что будет с функциональными языками?**

«Ассемблер» для функциональных языков

«The purpose of the present paper is to propose a definition of effective calculability which is thought to correspond satisfactorily to the somewhat vague intuitive notion in terms of which problems of this class are often stated, and to show, by means of an example, that not every problem of this class is solvable.»

(Alonzo Church, 1936)

«Ассемблер» для функциональных языков

«The purpose of the present paper is to propose a definition of effective calculability which is thought to correspond satisfactorily to the somewhat vague intuitive notion in terms of which problems of this class are often stated, and to show, by means of an example, that not every problem of this class is solvable.»

(Alonzo Church, 1936)

«The lambda calculus is not only simple, it is also sufficiently expressive to allow us to translate any high-level functional language into it.»

(Simon L. Peyton Jones, 1987)

Бестиповое лямбда исчисление

- Лямбда исчисление задается примерно так:

$$
\begin{array}{rcll}
e & ::= & c & \text{константы} \\
 & | & v & \text{переменные} \\
 & | & \lambda v.e & \text{лямбда абстракции} \\
 & | & e\ e & \text{применение}
\end{array}
$$

- Пара важных формальностей
  - Абстракция распространяется максимально вправо

$$\lambda x.\lambda y.x\ y \leftrightarrow \lambda x.\lambda y.(x\ y)$$

  - Применение левоассоциативная операция

$$e_1\ e_2\ e_3 \leftrightarrow (e_1\ e_2)\ e_3$$

  - Есть встроенные функции (например, +)

Как на этом программировать?

- α-конверсия: переименование связанных переменных

$$\lambda x.+\ x\ 1 \underset{\alpha}{\leftrightarrow} \lambda y.+\ y\ 1$$

- η-конверсия: обеспечение экстенсиональности

$$\lambda x.+\ 1\ x \underset{\eta}{\leftrightarrow} +\ 1$$

- β-редукция: вычисление

$$\begin{aligned}(\lambda f.f\ 3)\ (\lambda x.+\ x\ 1) &\underset{\beta}{\to} (\lambda x.+\ x\ 1)\ 3\\ &\underset{\beta}{\to} +\ 3\ 1\\ &\underset{\beta}{\to} 4\end{aligned}$$

Редексы и нормальная форма

Очевидно, что способов редукции множество, поэтому формализуем этот процесс

Definition

Терм вида $(\lambda v.F)\ G$ называется редексом, а терм $F[v \coloneqq G]$ его сокращением.

Definition

Лямбда терм вида $\lambda v.E$ или $v\ E_1 \dots E_n$ находится в нормальной форме, если E или $E_1, \dots, E_n$ при $n \geq 0$, также находятся в нормальной форме.

Стратегии редукции

Строгие стратегии:

Applicative order вычисляет слева направо, изнутри наружу

Call-by-value вычисляет слева направо, изнутри наружу, не заходя в лямбда-абстракцию

Ленивые стратегии:

Normal order вычисляет слева направо, снаружи внутрь

Call-by-need вычисляет слева направо, снаружи внутрь, не заходя в лямбда-абстракцию

Call-by-value и call-by-need могут не привести к нормальной форме!

Комбинаторы

Definition

Комбинатором называют замкнутый лямбда терм

Канонические примеры комбинаторов:

$$S \leftrightarrow \lambda f.\lambda g.\lambda x.f\ x\ (g\ x)$$

$$K \leftrightarrow \lambda x.\lambda y.x \qquad \omega \leftrightarrow \lambda x.x\ x$$

$$I \leftrightarrow \lambda x.x \qquad \Omega \leftrightarrow \omega\ \omega$$

Рекурсия в λ исчислении

У нас отсутствуют имена для термов, но как тогда выразить рекурсию?
Есть красивые теоремы о комбинаторе неподвижной точки, но мы обойдемся только определением:

$$Y\ F \rightarrow F\ (Y\ F)$$

А можно и без рекурсии:

$$Y \leftrightarrow \lambda f.(\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x))$$

AST представление лямбда терма

- Строковое представление удобно для человека
- Работать удобнее с древовидным представлением
- @ — применение
- λx — абстракция
- NOT — встроенная функция
- TRUE — константа
- x — переменная

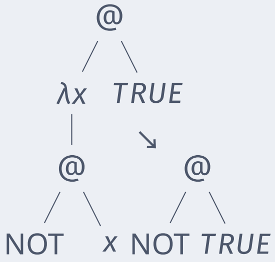
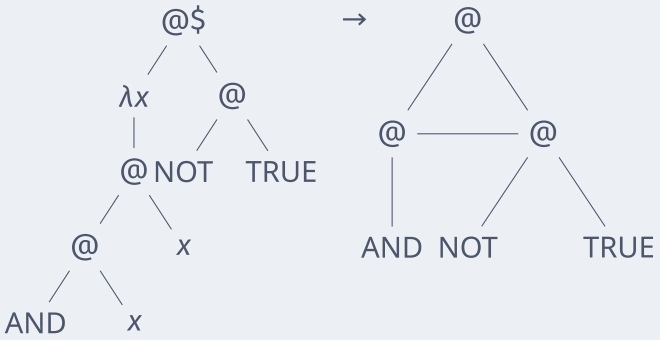


Рис.: AST представление терма (λx. NOT x) TRUE до и после редукции

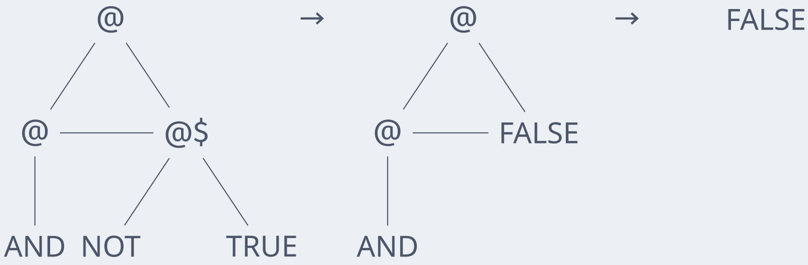
Красота редукции графов I

$$(\lambda x.\mathrm{AND}\ x\ x)\ (\mathrm{NOT\ TRUE}) \to \mathrm{AND}\ (\mathrm{NOT\ TRUE})\ (\mathrm{NOT\ TRUE})$$



Красота редукции графов II

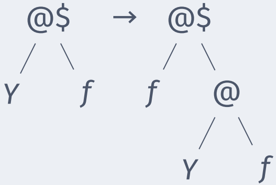
AND (NOT TRUE) (NOT TRUE) → AND FALSE FALSE → FALSE



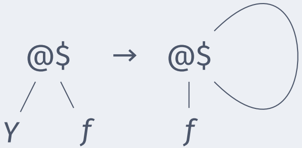
Рекурсия в редукции графов

$$Y\,f \rightarrow f\,(Y\,f)$$

Вариант 1:



Вариант 2:



Реализация редукции графов

- Идея красивая
- На привычных ЯП может быть красиво реализована

Реализация редукции графов

- Идея красивая
- На привычных ЯП может быть красиво реализована
- **Но как воспроизвести её в железе?**
  - Что делать со свободными переменными?
  - Достатоточно ли одношаговой редукции?

Суперкомбинаторы I

Definition

Суперкомбинатор $S, арности n, — это лямбда терм вида

$$\lambda x_1.\lambda x_2 \ldots \lambda x_n.E,$$

где E не является абстракцией, и выполняются условия

- В $S нет свободных переменных
- Любая лямбда абстракция в E есть суперкомбинатор
- $n \geq 0$, то есть абстракций может вообще не быть

Суперкомбинаторы II

Definition

Суперкомибнаторным редексом называется применение суперкомбинатора арности n к n аргументам.
Редукция суперкомбинатора заменяет суперкомибнаторный редекс на тело суперкомбинатора, подставляя аргументы на место соответствующих им переменных.

Примеры суперкомбинаторов:

$$3 \qquad \lambda f.f\,(\lambda x. + \; x\,x)$$
$$\lambda x. + \; x\,1 \qquad \lambda x.\lambda y. - \; y\,x$$

Примеры не суперкомбинаторов:

$$\lambda y. - \; y\,x$$
$$\lambda f.f\,(\lambda x.f\,x\,2)$$

Редукция суперкомбинаторов

$$\frac{\$Y\ w\ y = +\ y\ w \quad \$X\ x = \$Y\ x\ x \quad \$Prog = \$X\ 4}{\$Prog}$$

$$
\begin{aligned}
& \$Prog \\
\to\ & \$X\ 4 \\
\to\ & \$Y\ 4\ 4 \\
\to\ & +\ 4\ 4 \\
\to\ & 8
\end{aligned}
$$

Рекурсия в суперкомбинаторах

Использование Y комбинатора в суперкомбинаторах нерационально:

- Суперкомбинаторы могут быть рекурсивными, в отличие от лямбда термов, у них есть имена

$$\$F\ x = \$G\ (\$F\ (-\ x\ 1))\ 0$$

- Использование Y комбинатора потребует дополнительное определение

$$\$F = Y\ \$F1$$
$$\$F1\ F\ x = \$G\ (F\ (-\ x\ 1))\ 0$$

Конвертация лямбда терма в суперкомбинатор

ПОКА есть лямбда абстракции в терме:

1. Выбрать лямбда абстракцию, у которой нет лямбда абстракций в теле
2. Абстрагироваться по всем свободным переменным
3. Дать имя лямбда абстракции
4. Заменить лямбда абстракцию на её имя, примененное ко всем свободным переменным
5. Переместить лямбда абстракцию в список «скомпилированных»

SKI комбинаторы

А что если мы хотим ещё более примитивный способ представления термов? Тут нам помогут комбинаторы SKI:

$$I \leftrightarrow \lambda x.x$$

$$K \leftrightarrow \lambda x.\lambda y.x$$

$$S \leftrightarrow \lambda f.\lambda g.\lambda x.f\ x\ (g\ x)$$

Оказывается через них выражаются любые лямбда термы!

Правила преобразования

$$\begin{aligned} \lambda x.x &\Rightarrow I \\ \lambda x.c &\Rightarrow K\ c \quad (c \neq x) \\ \lambda x.e_1\ e_2 &\Rightarrow S\ (\lambda x.e_1)\ (\lambda x.e_2) \end{aligned}$$

Вычисление в SKI

$$
\begin{aligned}
& (\lambda x.+\ x\ x)\ 5 \\
S \Rightarrow\ & S\,(\lambda x.+\ x)\,(\lambda x.x)\ 5 \\
S \Rightarrow\ & S\,(S\,(\lambda x.+)\,(\lambda x.x))\,(\lambda x.x)\ 5 \\
I \Rightarrow\ & S\,(S\,(\lambda x.+)\,I)\,(\lambda x.x)\ 5 \\
I \Rightarrow\ & S\,(S\,(\lambda x.+)\,I)\,I\ 5 \\
K \Rightarrow\ & S\,(S\,(K\,+)\,I)\,I\ 5
\end{aligned}
$$

$$
\begin{aligned}
& S\,(S\,(K\,+)\,I)\,I\ 5 \\
\rightarrow\ & S\,(K\,+)\,I\ 5\,(I\ 5) \\
\rightarrow\ & K\ +\ 5\,(I\ 5)\,(I\ 5) \\
\rightarrow\ & +\ (I\ 5)\,(I\ 5) \\
\rightarrow\ & +\ 5\,(I\ 5) \\
\rightarrow\ & +\ 5\ 5 \\
\rightarrow\ & 10
\end{aligned}
$$

Замечание I (теоретическое)

SKI даже не самый минимальный набор комбинаторов

$$\begin{aligned} & S\;K\;K\;x \\ \to\; & K\;x\;(K\;x) \\ \to\; & x \end{aligned} \qquad\qquad \begin{aligned} & I\;x \\ \to\; & x \end{aligned}$$

Замечание II (практическое)

Очевидно, что при трансляции в SKI происходит взрыв размера терма, поэтому, обычно, вводят дополнительные «оптимизирующие» комбинаторы

$$
\begin{aligned}
B\ f\ g\ x &\to f\ (g\ x) \\
C\ f\ g\ x &\to f\ x\ g \\
S'\ c\ f\ g\ x &\to c\ (f\ x)\ (g\ x) \\
B^*\ c\ f\ g\ x &\to c\ (f\ (g\ x)) \\
C'\ c\ f\ g\ x &\to c\ (f\ x)\ g
\end{aligned}
$$

Замечание III (рекурсивное)

Для рекурсии необходим Y комбинатор. Если думать об Y комбинаторе ровно так, как в разделе про редукцию графов, то Y комбинатор при трансляции стоит считать встроенное функцией

Историческая справка

Попыток создать лямбда-процессор было немало. Две важные вехи:

1980-е Активное развитие функциональных языков программирования и теории вокруг них, конференция The Functional Programming Languages and Computer Architecture $\implies$ множество идей о создании функциональных машин

2010+ Массовость FPGA, проникновение ФП в массовые ЯП, идея специализированных ускорителей $\implies$ возрождение идеи лямбда-процессора

Проекты 1980-х

1975 Язык SASL Дэвида Тернера

1980 SKIM — реализация машины на *SKI* для SASL авторства Томаса Кларка и его команды из Кембриджа

1982 Суперкомбинаторы Джона Хьюза

1986 NORMA — параллельная *SKI* машина авторства Марка Шивеля из Исследовательского центра Остина

1987 GRIP — созданная командой Саймона Пейтона Джонса в Университетском колледже Лондона машина, основанная на суперкомбинаторах. Была реализована на процессорах Motorola 68020

Reduceron

Matthew Naylor & Colin Runciman, 2008–2012

- Процессор, основанный на суперкомбинаторах
- Реализован на DSL для Haskell — York Lava
- Для программирования используется своё подмножества Haskell — F-lite
- В последней версии выполнял один шаг редукции за один цикл
- А также поддерживал большое количество оптимизаций: спекулятивное исполнение, параллельность, анализ зависимостей

PilGRIM

Arjan Boeijink, Philip K. F. Hölzenspies & Jan Kuper, 2011

- RISC-style набор инструкций
- Наличие конвейера
- Цель — реализация в кремнии
- Идея реализации с помощью Clash
- Так никогда и не был реализован

fun arch

Cecil Accetti, Peilin Liu, et al., 2020+

- Целое семейство процессоров
- Использует свои RISC-style инструкции, которые описывают особые комбинаторы:
$$\lambda f.\lambda g.\lambda h.\lambda x.\lambda y.f\ (g\ (h\ x\ y)) \Rightarrow C_{5[1,2,3,4,5]}^{x(x(xxx))}$$
- Кажется, проект умер

Источники I

A. Church
An Unsolvable Problem of Elementary Number Theory
American Journal of Mathematics, 58 (1936) 345
https://doi.org/10.2307/2371045

S. L. Peyton Jones
The implementation of functional programming languages
Prentice Hall Interna1tional (UK) Ltd., 1987

P. Sestoft
Demonstrating Lambda Calculus Reduction
Electronic Notes in Theoretical Computer Science, 45 (2003)
https://doi.org/10.1016/S1571-0661(04)80973-3

Источники II

R. Stewart
HAFLANG - A History of Functional Hardware
https://haflang.github.io/history.html (accessed March 18, 2024).

D. A. Turner
A new implementation technique for applicative languages
Software: Practice and Experience, 9 (1979) 31–49
https://doi.org/10.1002/spe.4380090105

T. J. W. Clarke, P. J. S. Gladstone, C. D. MacLean, & A. C. Norman
SKIM - The S, K, I reduction machine
Proceedings of the 1980 ACM conference on LISP and functional programming (New York, NY, USA: Association for Computing Machinery, 1980), pp. 128–135
https://doi.org/10.1145/800087.802798

Источники III

R. J. M. Hughes
Super-combinators a new implementation method for applicative languages
Proceedings of the 1982 ACM symposium on LISP and functional programming (New York, NY, USA: Association for Computing Machinery, 1982), pp. 1–10
https://doi.org/10.1145/800068.802129

M. Scheevel
NORMA: a graph reduction processor
Proceedings of the 1986 ACM conference on LISP and functional programming (New York, NY, USA: Association for Computing Machinery, 1986), pp. 212–219
https://doi.org/10.1145/319838.319864

Источники IV

S. L. Peyton Jones
Parallel Implementations of Functional Programming Languages
The Computer Journal, 32 (1989) 175–186
https://doi.org/10.1093/comjnl/32.2.175

M. Naylor & C. Runciman
The Reduceron: Widening the von Neumann Bottleneck for Graph Reduction Using an FPGA
In O. Chitil, Z. Horváth, & V. Zsók,eds., Implementation and Application of Functional Languages (Berlin, Heidelberg: Springer, 2008), pp. 129–146
https://doi.org/10.1007/978-3-540-85373-2_8

Источники V

M. Naylor & C. Runciman
The Reduceron reconfigured and re-evaluated
Journal of Functional Programming, 22 (2012) 574–613
https://doi.org/10.1017/S0956796812000214

A. Boeijink, P. K. F. Hölzenspies, & J. Kuper
Introducing the PilGRIM: A Processor for Executing Lazy Functional Languages
In J. Hage, & M.T. Morazán,eds., Implementation and Application of Functional Languages (Berlin, Heidelberg: Springer, 2011), pp. 54–71
https://doi.org/10.1007/978-3-642-24276-2_4

Источники VI

C. A. R. A. Melo, P. Liu, & R. Ying
A Platform for Full-Stack Functional Programming
2020 IEEE International Symposium on Circuits and Systems (ISCAS) (2020), pp. 1–5
https://doi.org/10.1109/ISCAS45731.2020.9180772

C. Accetti, R. Ying, & P. Liu
Structured Combinators for Efficient Graph Reduction
IEEE Computer Architecture Letters, 21 (2022) 73–76
https://doi.org/10.1109/LCA.2022.3198844

Источники VII

- C. Accetti & P. Liu
  Architectural Support for Functional Programming
  2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC) (2022), pp. 1–2
  https://doi.org/10.1109/VLSI-SoC54400.2022.9939644
- Д. Москвин
  Системы типизации лямбда-исчисления
  https://www.lektorium.tv/course/22797 (accessed March 18, 2024)
- Ю. Литвинов
  Лекция про лямбда исчисление
  https://github.com/yurii-litvinov/courses/tree/master/structures-and-algorithms (accessed March 18, 2024)