§ 2.3. ЛЯМБДА-ИСЧИСЛЕНИЕ – Λ – ИСЧИСЛЕНИЕ (автор Алонсо Черч)

Λ – исчисление – это безтиповая теория, которая рассматривает функции, как правила, а не как графики. В традиционном подходе: два представления (x^2-4) $(x-2)(x+2)$ выражают одну и ту же функцию, а с точки зрения Λ – исчисления они выражают разные функции, так как правила вычисления различны. **Λ – исчисление – это прикладное исчисление предикатов 1-го порядка (ПИП 1-го порядка).**

ОСОБЕННОСТИ:

1) Безтиповость: объекты могут являться функциями и аргументами, т.е. функции могут быть заданы программами (функции или процедуры Pascal), которые могут передаваться через формальные параметры другим программам.
2) Λ – исчисление представляет класс частичных функций – «λ – определимые функции», которые характеризуют неформальное понятие «эффективной вычислимости», тем самым связано с понятием «алгоритма вычисления».
3) Λ – исчисление является **теоретической моделью** современного функционального программирования (например, язык ЛИСП, который использует Λ – исчисление в качестве промежуточного кода после 1-го этапа трансляции).

2.3.1. Λ – выражения и их вычисления.

Пример: $f(x)=5x^3+2$. В Λ – исчислении различают (устраняют различное понимание): символьную запись $\lambda x.5x^3+2$ и ее вызов (вычисление) $(\lambda x.5x^3+2)x$. Λ – исчисление изучает функции и их аппликативное поведение (поведение относительно применения к аргументу). Выражение $f(x,y)=x+2y$ можно считать как функцию от x (это записывается как $\lambda x.x+2y$) и как функцию от y (это записывается как $\lambda y.x+2y$). Вызовы могут быть следующими:

а) $(\lambda x.x+2y)a=a+2y$ и $(\lambda y.x+2y)a=x+2a$;

б) $((\lambda y.\lambda x.x+2y)a)b=(\lambda x.x+2a)b=b+2a$;

в) $((\lambda x.\lambda y.x+2y)a)b=(\lambda y.a+2y)b=a+2b$.

2.3.2. Определение Λ – термов и Λ – выражений

ОПРЕДЕЛЕНИЕ:

1. Каждая переменная или константа есть **терм.**
2. По любой переменной x и любому $\lambda-$терму M строится новый $\lambda-$терм: $\lambda x.M$ – функция от x, которую называют $\lambda-$абстракцией.

Замечание 1: Аналогичен в Pascal селектор записи «$M.x$».

Замечание 2: **Константы:** целые числа, булевы константы, арифметические операции (функторы), булевы функции и т.п.

3. По любым $\lambda-$термам M и N строится новый $\lambda-$терм MN, обозначающий применение (аппликацию) оператора M к аргументу N, то есть подстановка $[N/x]M$ позволяет получать $\Lambda-$выражения (предикаты).

Замечание 3:Это определение $\lambda-$термов (выражений) является правилом конструирования формул «по индукции».

Замечание 4: Правила вывода определяют алгоритм вычисления $\Lambda-$выражений

$\delta-$СВОРАЧИВАНИЕ, $\beta-$СВОРАЧИВАНИЕ И РЕДУКЦИЯ

Определение 1: Константа, обозначающая функтор, применяемый к операндам, определяет подтерм, называемый $\delta-$редексом, а процесс применения называется $\delta-$сворачиванием, в результате которого получается новое $\Lambda-$выражение.

ПРИМЕР: $+13 \rightarrow_{\delta} 4$. То есть, «$+13$» – $\delta-$редекс, $\delta-$сворачивание: $+13 \rightarrow_{\delta} 4$.

Определение 2: Терм вида $(\lambda x.M)N$ называется $\beta-$редексом.

Определение 3: Если $\beta-$редекс содержится в терме P и одно из его вхождений заменяется термом (подстановкой) $[N/x]M$, то этот процесс $\beta-$сворачивания (свертывания) обозначается $P \rightarrow_{\beta} Q$ и означает, что терм P $\beta-$сворачивается к $\Lambda-$**в**ыражению Q.

Определение 4: Конечная последовательность $\delta-$ или $\beta-$свертываний называется «**редукцией**». Один шаг $\delta-$ или $\beta-$свертывания обозначается стрелкой без индекса, просто «$\rightarrow$».

Замечание 5: Использование констант и $\delta-$правил излишне, так как константы можно реализовать только атомарными термами в виде переменных (т.е., **«чистое $\Lambda-$исчисление»** - $\Lambda-$исчисление без констант и $\delta-$правил).

Примеры редукций (используемые редексы подчеркнуты):

1.

$$(\lambda f.\lambda x.f\ 3x)(\lambda y.\lambda x.*\ x\ y)0 \rightarrow (\lambda x.(\lambda y.\lambda x.*\ x\ y)3x)0 \rightarrow (\lambda y.\lambda x.*\ x\ y)30 \rightarrow$$

$$(\lambda x.*\ x3)\ 0 \rightarrow *03 \rightarrow 0$$

2.

$$(\lambda f.\lambda x.f\ 3x)(\lambda y.\lambda x.*\ x\ y)0 \rightarrow (\lambda x.(\lambda y.\lambda x.*\ x\ y)3x)0 \rightarrow (\lambda x.*\ x3)\ x)0 \rightarrow$$

$$(\lambda x.*\ x3)\ 0 \rightarrow *03 \rightarrow 0$$

2.3.3. Нормальные формы $\Lambda-$выражений

Определение: Если для Λ – выражения нельзя применить никакого правила редукции, т.е. Λ – выражение не содержит редексов и находится в **«нормальной форме».**

Замечание 1: В программировании этому понятию соответствует понятие конца вычислений по определенной синтаксической схеме, например:

While-(существует хотя бы один редекс) – do – (преобразовывать один из редексов) – end – (выражение теперь в нормальной форме).

Замечание 2: β **– свертывания не всегда могут давать желаемый результат – может происходить зацикливание.**

ПРИМЕР:

а) $(\lambda x.\lambda y.y)((\lambda z.z\ z)(\lambda z.z\ z)) \to (\lambda x.\lambda y.y)((\lambda z.z\ z)(\lambda z.z\ z)) \to$ *зацикливание*

б) $(\lambda x.\lambda y.y)((\lambda z.z\ z)(\lambda z.z\ z)) \to \lambda y.y \to$ *редукция закончилась.*

ПОРЯДОК РЕДУКЦИЙ (стратегия выбора редексов)

1. **АППЛИКАТИВНЫЙ порядок редукции (АПР):** вначале преобразовывается самый левый из самых внутренних редексов, не содержащих других редексов: $(\lambda x.\lambda y.y)((\lambda z.z\ z)(\lambda z.z\ z)) \to$ **такой порядок редукции в данном примере привел к зацикливанию.**
2. **НОРМАЛЬНЫЙ порядок редукции (НПР):** вначале преобразовывается самый левый из самых внешних редексов, которые не содержатся в других редексах: $(\lambda x.\lambda y.y)((\lambda z.z\ z)(\lambda z.z\ z)) \to \lambda y.y$ **такой порядок редукции в данном примере привел за один шаг к окончанию вычислений.**

ЗАМЕЧАНИЕ: Функция $\lambda x.\lambda y.y$ в случае НПР отбрасывает свой аргумент x.

ВЫВОДЫ:

1. НПР в таких случаях эффективно откладывает вычисление любых редексов внутри выражения аргумента до тех пор, пока это возможно: **«не делай ничего, пока это не потребуется»**. Это пример **«ленивых вычислений» - стратегия НПР,** которая встречается в некоторых функциональных языках, например, в алгоритмическом языке **Clean.**
2. Стратегия АПР соответствует **“энергичным вычислениям» -«делай все, что можешь».** Эта стратегия применяется в языке **ЛИСП,** которая может приводить иногда **к зацикливанию**.

Следствие из теоремы Черча-Россера: Если Λ – выражение может быть приведено двумя различными способами к двум нормальным формам, то эти формы совпадают или могут быть получены одна из другой с помощью замены связанных переменных ($\lambda x.x \equiv \lambda y.y$).

Теорема «стандартизации»: Если Λ – выражение имеет нормальную форму (НФ), то НПР гарантирует достижение этой НФ (с точностью до замены связанных переменных).

Рекурсивные функции

Так как в $\Lambda-$исчислении все функции анонимны (без имен), то необходимо их вызов производить не по имени, а другим способом. Например, рассмотрим рекурсивную функцию, которая в качестве одного из аргументов имеет ссылку на себя:

а) $Sum(x)$ - сложение всех целых чисел:

$sum(n) = (IF(= n\ 0)\ 0\ (+n(sum(1-n))))$ – если $n = 0$, то $sum(0)$, иначе складываем (+) число n с суммой для числа, меньшего на 1, чем n, т.е. (1–) обозначает уменьшение на 1. Это была форма записи рекурсивной функции на языке Гильберта-Клини, а в форме $\lambda-$абстракции: $sum = \lambda s.\lambda n.IF(= 0\ n)\ 0\ (+n(sum(1-n)))$.

Осталось связать переменную s со значением функции sum, чтобы сделать функцию анонимной. Это можно сделать с помощью специальной функции «$Y-$комбинатор»:

$$Yf = f(Yf).$$

Например, функция «$\lambda x.x$» имеет бесконечное число «неподвижных» точек.

Таким образом, в $\Lambda-$исчислении функция sum записывается так:

$$Y(\lambda s.\lambda n.IF(= 0\ n)\ 0\ (+n(sum(1-n)))).$$

Проверим, например, $s(1)$ должно быть равно 1.:

$$Y(\lambda s.\lambda n.IF(= 0\ n)\ 0\ (+n(sum(1-n))))1 \rightarrow$$

$$\lambda n.IF(= 0\ n)\ 0\ (+n((Y(\lambda s.\lambda n.IF(= 0\ n)\ 0\ (+n(s(1-n)))))(1-n)))1 \rightarrow$$

$$IF(= 0\ n)\ 0\ (+1((Y(\lambda s.\lambda n.IF(= 0\ n)\ 0\ (+n(s(1-n)))))(1-1))) \rightarrow$$

$$(+1((Y(\lambda s.\lambda n.IF(= 0\ n)\ 0\ (+n(s(1-n)))))\ 0)) \rightarrow$$

$$(+1(\lambda n.IF(= 0\ n)\ 0\ (+n((Y(\lambda s.\lambda n....)))))0) \rightarrow$$

$$(+1(IF(= 0\ 0)\ 0\ (+0((Y(\lambda s.\lambda n....)))))) \rightarrow$$

$$(+1\ 0) \rightarrow 1.$$

Определим «комбинатор» Y как

$$\lambda h.(\lambda x.h(x\,x))(\lambda x.h(x\,x)).$$

Проверим выполнение формы записи $Yf = f(Yf)$:

$$Yf \equiv (\lambda h.(\lambda x.h(x\,x))(\lambda x.h(x\,x)))f \rightarrow (\lambda x.f(x\,x))(\lambda x.f(x\,x)) \rightarrow$$

$$f((\lambda x.f(x\,x))(\lambda x.f(x\,x))) \rightarrow f(Yf).$$

Чистое $\Lambda-$исчисление

Чистое $\Lambda-$исчисление получается при удалении констант и $\delta-$правил, так как в нем любые константы и функции можно построить атомарными термами, использующие только лишь переменные.

ПРИМЕРЫ:

а) $TRUE \equiv \lambda x.\lambda y.x$; б) $TRUE \equiv \lambda x.\lambda y.y$; в) $IF \equiv \lambda p.\lambda q.\lambda r.p\,q\,r$.

Легко проверить, что выполняются следующие редукции:

1. $TRUE\ A\ B \rightarrow A \equiv \quad ((\lambda x.\lambda y.x)A)B \rightarrow (\lambda y.A)B \rightarrow A$
2. $FALSE\ A\ B \rightarrow B \equiv \quad ((\lambda x.\lambda y.y)A)B \rightarrow (\lambda y.y)B \rightarrow B$

$IF\ TRUE\ A\ B \equiv$

$(\lambda p.\lambda q.\lambda r.p\,q\,r)(\lambda x.\lambda y.x)AB \to (\lambda q.\lambda r.(\lambda x.\lambda y.x)\,q\,r)A\,B \to$

3. $(\lambda q.\lambda r.(\lambda y.q)\,r)A\,B \to (\lambda q.\lambda r.q)A\,B \to (\lambda r.A)\,B \to A$

Нумерация Черча: n представлять композицией $x^n(y)$, т.е. $x(x(....(xy)...))$, т.е. x повторяется n раз. Для каждого натурального n положим: $<0> \equiv \lambda x.\lambda y.y$,

$<n> \equiv \lambda x.\lambda y.x^n(y)$. Тогда «сложение» чисел определяется $\Lambda-$выражением:

$$+\,x\,y \equiv \lambda p.\lambda q.x\,p(y\,p\,q).$$

Проверка:

$+\,1\,1 \equiv \lambda p.\lambda q.<1>p(<1>p\ q) \equiv$

$\equiv \lambda p.(\lambda q.((\lambda x.\lambda y.x\,y)\,p((\lambda x.\lambda y.x\,y)\,p\ q))) \to \lambda p.(\lambda q.((\lambda x.\lambda y.x\,y)\,p\,p\ q)) \to$

$\to \lambda p.(\lambda q.(p\,p\ q)) \equiv <2>.$

Определение: Говорят, что частичная функция φ с k аргументами $\lambda-$определима термом M, когда терм $M <n_1><n_2> .....<n_k>$ $\beta-$редуцируется к терму $<\varphi(n_1,n_2,.....,n_k)>$, если значение $\varphi(n_1,n_2,...,n_k)$ определено, и $M <n_1><n_2> .....<n_k>$ не имеет **нормальной формы,** если $\varphi(n_1,n_2,...,n_k)$ не определено.

Теорема Клини: Частичная функция частично-рекурсивна тогда и только тогда, когда она $\lambda-$определима.