САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МАТЕМАТИКО-МЕХАНИЧЕСКИЙ ФАКУЛЬТЕТ
Кафедра

РЕФЕРАТ ПО ИСТОРИИ ИНФОРМАТИКИ
на тему: История языков ML

Выполнил:
Проверил: профессор кафедры параллельных алгоритмов Прозорова Э.В.

Оценка:

Краткая рецензия:

Основные языки семейства ML: Standart ML и OCaml. В реферате был рассмотрен контекст и цели, для которых эти языки создавались, указаны родственные связи этих языков с современными императивными языками программирования. Во второй половине XXго века велись исследования функциональных языков программирования: языков, в которых абстракция достигается с помощью использования функций. Автор остановился на основных работах, посвященных развитию языков функционального программирования. Цели исследования выполнены.

Научный руководитель диссертационной работы: ???

Санкт-Петербург – 2020

Оглавление

1. Введение

В попытке классифицировать языки программирования можно выделить два типа: декларативные и императивные. Современные императивные языки однозначно классифицировать не так просто, потому что почти все успешные языки сочетают в себе особенности различных стилей программирования. Например: объектно-ориентированный подход; поддержка функций как сущностей первого порядка; поддержка встроенных языков, специфичных для конкретной предметной области, и т.д. Однако, если у $C++$ и $Java$ искать общего предшественника, то в нём инструкции должны выполняться последовательно друг за другом и должна быть поддержка изменяемых значений. Именно языки с такими свойствами принято называть императивными.

Своей популярности императивное программирование обязано в первую очередь устройству современного компьютера. Язык C предоставляет прямой доступ к памяти и регистрам современных процессоров, позволяя писать максимально эффективный код под данную архитектуру. Однако, программисту на C приходится самостоятельно беспокоиться о правильном выделении памяти и о корректном доступе к ней. В программах на C большинство низкоуровневых особенностей компьютера на виду, а компиляторы C не позволяют находить ошибки доступа к памяти или позволять программисту использовать высокий уровень абстракции. Поэтому язык C считается языком низкого уровня.

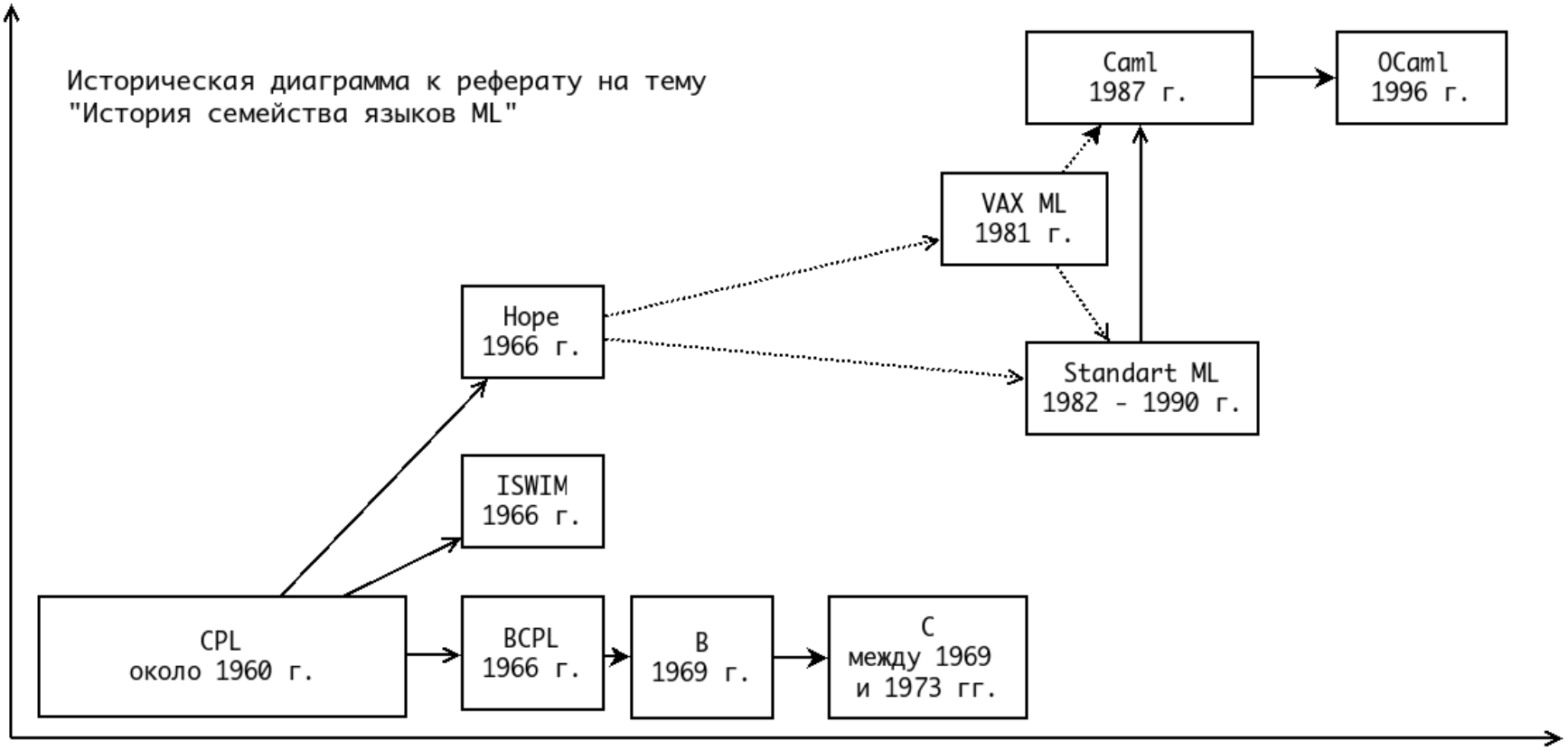
Идейные наследники языка C (например, $C++$ или $Java$) позволяют программисту меньше беспокоиться об управлении памятью и писать программы на более высоком уровне абстракции. Однако, из-за удаления языка от архитектуры компьютера программы на высокоуровневых языках в ограничены по производительности. При этом современ-

ные языки программирования унаследовали концепцию изменяемых переменных, а она очень сильно осложняет создание формальной семантики языков программирования, а этот вопрос в последнее время приобретает всё большую популярность. Изменяемые значения также осложняет статическое исследование программ: поведение кода зависит не только от него самого, но и от всего окружения, в котором он выполняется.

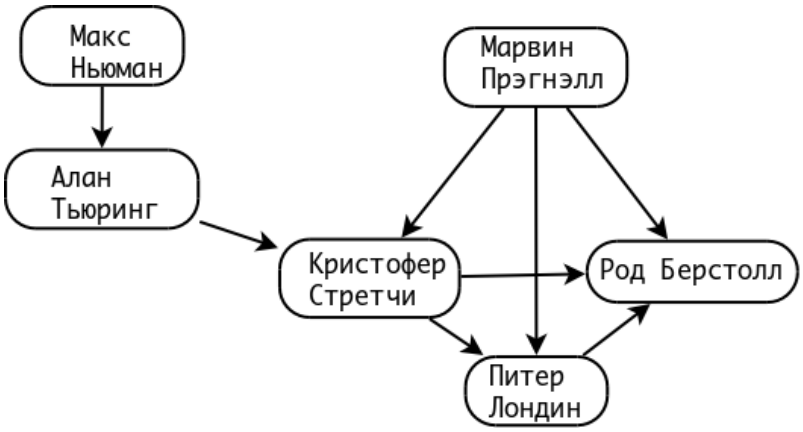
Во второй половине XXго века также велись исследования в области функциональных языков программирования: языков, в которых абстракция достигается с помощью использования функций, и вопрос необходимости использования изменяемого состояния стоит не так остро. Все они основаны на λ-исчислении Алонзо Чёрча (Alonzo Church), для некоторых построены формальные модели, другие основаны на формальных математических моделях. Долгое время они находились в тени императивных языков программирования, потому что для эффективной работы с ними требуется наличие автоматического управления памятью, что сказывается на эффективности программ. Долгое время функциональные языки компилировались в код на языке $LISP$, который интерпретировался, при этом весьма медленно. С ростом производительности и объёмов памяти у компьютеров функциональные языки получили возможность соревноваться по производительности с императивными языками, а современные императивные языки начали перенимать языковые конструкции, характерные для функционального программирования.

Целью данного реферата является рассмотрение пути развития наиболее известного статически типизируемого языка ML, а также родственных с ним языков. Будут рассмотрены работы учёных, которые

оказали на его развитие наибольшее влияние, а также найден общий предок ML и современных императивных языков программирования. Кроме этого, будут рассмотрены причины и контекст, в которых эти языки появились. Будут указаны ошибки при развитии языков, а также подчеркнуты цели, к которым стоит стремиться при разработке нового языка программирования.


Историческая диаграмма к реферату на тему
"История семейства языков ML"
Caml
1987 г.
OCaml
1996 г.
VAX ML
1981 г.
Hope
1966 г.
Standart ML
1982 - 1990 г.
ISWIM
1966 г.
CPL
около 1960 г.
BCPL
1966 г.
B
1969 г.
C
между 1969
и 1973 гг.

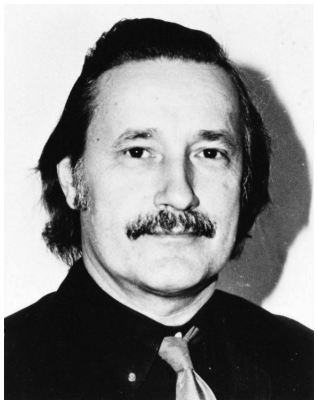
2. Предыстория развития ML в Британии



Говоря об основоположниках *ML* стоит упомянуть таких британских ученых как Макс Ньюман (Max Newman), Алан Тьюринг (Alan Turing), Кристофер Стрэйчи (Cristopher Strachey), Петер Лондин (Peter Laundin), Род Берстолл (Rod Berstall) и Марвин Прэгнелл (Marvin Pragnell). Среди них идейным лидером можно назвать Кристофера Стрейчи, а Марвина Прэгнелла – ”катализатором”. Однажды, Марвин Прэгнелл увидел в кафе человека, который читает книгу Principia Mathematica [1] и пригласил его на семинар, этим человеком оказался Питер Лондин. С Родом Берстоллом случилась аналогичная история, но в библиотеке. Сам семинар проводился в колледже Бёркберк без разрешения руководителей колледжа. Марвин Прэгнелл нашел человека с ключом, который их впустит аудиторию, где они заседали до позднего вечера. Вместе с Кристофером Стрэйчи, Петером Лондиным и Родом Берстолом семинар несколько раз посещал и Робин Милнер. Там они обсуждали вопросы логики, теории категорий и компьютеров. Вероятно, с этого любительского семинара и стоит отсчитывать историю ML. Случайные посетили вспоминали, что на нём всё было устроено несколько по-любительски, в отличие от того как такие вещи были организованы в США. При этом стоит заметить, что участники не явля-

лись формально учеными, а были работающими программистами.

2.1. Кристофер Стрейчи



Кристофер Стрейчи (Cristopher Stratchey, 1916-1975) был другом Алана Тьюринга, когда тот работал в Кембридже и Манчестере. Он известен тем, что написал программу для игры в шашки в 1951 году в Национальной Физической Лаборатории (National Physical Laboratory, NPL). Это, возможно, была первая играющая программа для компьютера. Эта программа запускалась на машине Марк-1 манчестерского университета, так как эта машина имела намного больше памяти чем, например, машина *ACE*, которую спроектировал Алан Тьюринг. Для Марк-1 Кристофер Стрэйчи также написал первую программу для проигрывания простых мелодий, а именно детской песенки *Baa Baa Black Sheep*.

Что касается языков программирования, Кристофер Стрейчи спроектировал *CPL* (*Combined Programming Language*), который назывался изначально *Cambridge Programming language*, а после переезда в Лондон был переименован в *Combined*. Также его называли в шутку *Cristopher's programming language*. В этом языке были функции как сущности первого класса, а также было впервые введено понятие L-values — выражений, которые вычисляются в место, где они хранятся. *CPL*

никогда не был успешно реализован, были прототипы и в Лондоне, и в Кембридже, но язык был очень амбициозен и сложен для того времени, так что его компиляторы никогда не появились и пользователей у него не было. Однако, Мартин Ричардс написал выпускную работу про компилятор этого языка, и вскоре решил, что было бы хорошо несколько упростить язык CPL. Так появился $Basic\ CPL$ ($BCPL$) [2], компилятор которого был реализован. Это был очень успешный язык для реализации больших систем в конце 60х – начале 70х, который использовался, например, в компании Xerox Park. А в Bell Labs Кен Томпсон (Kenneth Thompson) также начал использовать его и улучшать. Так появился язык B, который потом эволюционировал в $C++$.

```
CPL  =>  BCPL  =>  B  =>  C  =>  C++
```

Также Кристофер Стрэйчи ввёл понятие каррирования (currying). В середине 1960х он написал очень важную работу, представленную на конференции в Копенгагене в 1967, которая называлась “Фундаментальные понятия в языках программирования“ [3]. В ней он описывал различные виды полиморфизма и первым описал параметрический полиморфизм. Так же он известен как создатель денотационной семантики Скотта-Стрейчи (совместно с Дана Скотт (Dana Scott) в 1969 [5], который предоставил математическое обоснование этой семантики, основанной на лямбда-исчислении). Также Кристофер Стрейчи первым дал формальное определение продолжений (continuations). Он пришел к идее разделения времени (time-sharing, 1958, [4]) правда в несколько урезанном виде. В 1960-64 годах он нанял Питера Лондина (Peter Lundin) как своего помощника, который работал в национальной Лаборатории физики и в National Research and Develpment Corporation в Лондоне, а с 1959 Питер Лондин стал работать как независимый кон-

сультант.

> Вот что пишет ученик Стрейчи Крис Водсворт:
>
> “К.Стрейчи имел особое чутьё в тех случаях, когда что-то было сделано “правильно” – в основном когда было достаточно просто и достаточно элегантно, что выглядеть интуитивно правильным – и он презирал чересчур проработанные и умудренные методы которые “кое-как работали”. Его любимым принципом был такой: “Ты можешь затолкать на гору горошину носом, но это не будет являться правильным способом осуществления этого”. Я это называл “тестом Стрейчи”.”

2.2. Питер Лондин

Питер Лондин (Peter Landin) написал множество статей в 1960х годах.

В статье “Механическое вычисление выражений” [6] он ввел понятие SECD-машины. Это не была первая абстрактная машина в мире, он использовал идеи сообщества программистов на языке $ALGOL$ из города Мюнхен.

В статье ”Соответсвие между языком $ALGOL60$ и лямбда-нотацией

Чёрча: части 1 и 2” [7] он ввел понятие потоков (streams), которые являются особого рода методом для осуществления ввода-вывода.

В статье ”Обобщение переходов и меток” [8] он ввел понятие J-оператора, который является предшественником термина продолжение (continuation).

Статья ”The next 700 programming languages” [9] 1966 года описывает $ISWIM$ язык, который оказал влияние на ML.

В соавторстве с Родом Берстоллом в 1969 году была опубликована статья [10], в которой Питер Лондин предвосхитил появление алгебраических типов данных. В предыдущих работах, например про $ISWIM$, использовались некоторые дополнительные лингвистические конструкции для описания структур данных, которые не являлись частью языка.

В течение 1960х в MIT шла разработка Pedagogical Algorithmic Language (PAL) — реализации $ISWIM$ Артура Эванса. Кристофер Стрейчи приезжал пару раз в MIT в начале 60х, но дело не сдвигалось пока в 1967 году Питер Лондин отправился работать в MIT, где в течение года он закончил PAL.

2.3. Робин Милнер

Как и Кристофер Стрейчи, и Питер Лэндин, Робин Милнер не начинал как ученый – вначале он работал школьным учителем, а перед тем как попасть в научную среду – программистом на компанию Ferranti.

> Из его тьюринговской лекции:
>
> “Идея того, чтобы машина доказывала теоремы используя логику, и идея того, чтобы используя логику понимать, что машина делала... Эта двойственность вдохновляла меня в том числе и потому,

что это было неочевидно”

Кристофер Стрейчи, Питер Лондин, Род Берстолл, Робин Милнер (и другие, как Тони Хоар) основали британскую традицию в исследовании языков программирования и руководствовались в своей научной деятельности следующими принципами:

- Осмысление важности фундаментальных оснований и семантики при изучении вычислений и программирования.
- Поиск ясности, строгости и элегантности путём использования математических идей и подходов, в частности из логики и алгебры.

Эти принципы особенно прижились в сообществе Эдинбурга.

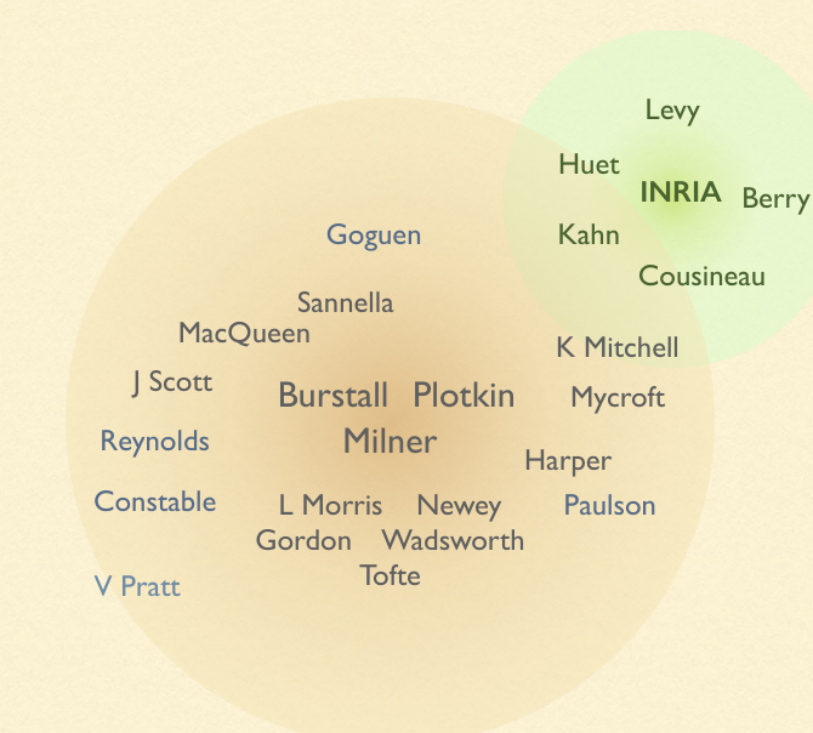
3. Эдинбург в конце 1970х

В 1970х годах Робин Милнер работал над LCF – системой доказательства теорем, которая заработала в 1978-79х годах. В ней ML использовался как метаязык. Осенью 1978 в Эдинбург приезжает Лука Карделли (Luca Cardelli) для получения Ph.D. Род Берстолл и Дэйв МакКуин работали в то время над языком программирования $Hope$ [11] – чистым языком программирования, а также первым языком программирования с алгебраическими типами данных. Хотя за год до этого Род Берстолл сделал ”игрушечный” язык программирования NPL [12], где абстрактные типы данных были, но в нём немного по-другому к типам добавлялись их конструкторы.

В Эдинбурге были две основные исследовательские группы. Группа под руководством Рода Берстолла располагалась в Школе Искусствен-

ного Интеллекта (School of Artificial Intelligence) , а группа Робина Милнера – в Королевском Доме Науки (King’s Science Building) на три мили южнее. Это расстояние создало существенный барьер между этими двумя группами – не так просто было сходить туда-сюда. Решение нашлось в виде сообщества Роберта Бойера, который в те времена получал степень в Школе Искусственного Интеллекта и тоже организовывал семинары. Когда Роберт Бойер после защиты покинул Эдинбург, его работу как организатора начали выполнять другие люди: слушатели стали собирался по вечерам дома у Робина Милнера или в гостиной у кого-нибудь другого. Это сблизило два сообщества и в конце концов слушатели образовали основную часть Лаборатории Основ Информатики (Laboratory for Foundations of Computer Science, LFCS), хотя эта лаборатория формально не существовала до 1986 года. Кроме Робина Милнера и Рода Берстолла и организационной деятельностью также занимался Мэьтю Хеннеси (Matthew Hennessy) и Гордон Плоткин (Gordon Plotkin).

В 70х Эдинбург являлся в некотором смысле центром исследования языков программирования в Европе: в этом городе работали несколько ключевых фигур, а также первое поколение их учеников. Другой центр располагался в Париже в INRIA (Institut national de recherche en informatique et en automatique, Государственный институт исследований в информатике и автоматике). Между ними были налажены прочные связи: командировки и совместные исследования финансировались государством.



Самый первый ML, также известный как $DEC10\ ML$, встраивался в систему LCF как метаязык. Логика Скотта в нём описывалась с помощью объектного языка $PPLAMBDA$. В ML поддерживались термы, формулы и теоремы, при этом теоремы являлись абстрактными типами и их конструирование из термов могло происходить только путём применения правил вывода, таким образом в языке не могло появиться неверных теорем. В LCF присутствовала возможность преобразования синтаксического дерева ML в код и наоборот (техники, называемые quotation/antiquotation). Функциональный язык программирования был необходим, чтобы с помощью комбинирования тактик высшего порядка строить новые тактики, а сильная типизация – чтобы не было обходных способов создать теоремы.

Основные свойства LCF/ML:

- Основан на $ICWIM$.
- Вывод типов через милнеровский let-полиморфизм.
- Абстрактные типы данных (через ключевое слово abstype).

- Тип-сумма и тип-произведение (`t1+t2`, `t1#t2`).
- Изменяемые значения через объявление letref (с весьма нетривиальными правилами типизации).
- Вложенные кортежи и шаблоны для байндинга списков (pattern-matching).
- Условные циклы.
- Поддержка ошибочных ситуаций путём передачи строк, которые назывались токенами. Различные виды ”ловушек” (обработчиков исключений-токенов): простые, условные и зацикливающиеся.

Компилятор для LCF/ML был изначально написан на $LISP$ (вначале на диалекте Standford, потом Rutgers). Код с ML транслировался в $LISP$ и интерпретировался, поэтому работал весьма медленно. Парсер[1] был основан на приоритетном парсере Вона Пратта (Vaugnan Pratt), который имел следующую особенность: функции-парсеры прикреплялись к отдельно взятым символам, в то время как было бы более естественно использовать разные парсеры для одного и того же символа в зависимости от контекста. Таким образом получалось, что если запятая использовалась для разделения элементов кортежа, то её нельзя было переиспользовать для разделения элементов списка, а если точка с запятой использовалась для разделения элементов списка, то для оператора следования нужно было использовать удвоенную точку с запятой, и т.д.

[1] В русскоязычной литературе также используется термин ”синтаксический анализатор”.

3.1. VAX ML (или Cardelli ML)

В 1980 Лука Карделли начинает работу на собственным диалектом ML и его компилятором. К 1981 году он реализовывает его полностью на паскале, включая и сборщик мусора. В данной реализации были следующие нововведения:

- Именованные структуры и варианты был основаны на плоткинских лекциях по теории доменов. До этого они не объявлялись явно перед использованием.
- Комбинаторы для управления вычислениями (declaration combinators):
  - and – одновременно;
  - enc – последовательно (enclosing) ⇒ `d1;d2`;
  - ins – локально ⇒ `local d1 in d2 end`;
  - rec – рекурсивно;
  - with – для особой формы абстрактных типов (объявления вида `with t ⇔ ty`);
- Оператор для ссылочного типа (ref type) вместо конструкции letref. А также правила вывода для него: ref, !, :=.
- Ввод-вывод с помощью потоков.
- Базовая поддержка модулей с раздельной компиляцией.

Компилятор $VAX\ ML$ был также реализован в Эдинбурге в 1980-1982 годах, где запускался под операционной системой VAX/VMS. Он также был написан на Паскале, включая библиотеки времени выполнения. Код компилировался в виртуальную машину FAM (Functional Abstract

Machine), которая генерировала код для VAX машины из FAM кода. В этой реализации впервые поддерживался импорт/экспорт модулей на лету. Данный компилятор вместе с операционной системой продавался пользователям начиная с 1981 года.

Развитие $VAX\ ML$ показало, что ML может использоваться как язык общего назначения и имеет эффективную реализацию. Это привело к росту количества диалектов ML и последующему предложению от Робина Милнера “стандартизовать” ML ($Standart\ ML$ или SML, апрель 1983). Таким образом, $VAX\ ML$ являлся непосредственным предшественником SML и тестовой площадкой для ранних экспериментов над дизайном SML.

Из самых важных встреч для обсуждения дизайна можно выделить три: состоявшиеся в апреле 1983 года, июне 1984 и мае 1985.

Так получилось, что большое количество людей собралось в Эдинбурге в апреле 1983, и Бернард Суфрин организовал встречу в гостиной Робина Милнера, где хозяин дома представил своё виденье дизайна языка [18]. К данной встрече бы написан первый черновик дизайна и включал в себя лучшие идеи из языков LCF/ML, $VAX\ ML$ и $Hope$.

Название для языка Робин Милнер выбрал сам, надеясь предотвратить длительные споры о названии и дизайне. Но это оказалось ошибкой, потому что такое амбициозное имя в некотором смысле ”приклеилось” к ML и вводит в заблуждение людей мало знакомых с этой темой.

Основные возможности, заявленные в первом дизайне языка:

- Формы объявления типов данных, конструкторы в паттернах.
- Без записей и вариантов (в смысле $VAX\ ML$).
- Функциональные выражения с клаузами: `fun v1. e1 | ... | vn. en`.

- Мономорфные ссылки и равенство.
- ”Локальные” объявления значений вместо специального оператора `ins` из $Cardelli\ ML$.
- Механизм обработки ошибок (escape) с токеном и единая форма обработки (trap) для этого: `e1 trap v1. e1 | ... | vn. En`. Но исключения могли в себе нести только строки.

В те времена ещё не были распространены $email$ и $ARPANET$, поэтому вся переписка велась через почтовое сообщение. Постепенно спецификация стандарта менялась, наращивая небольшие изменения. В июне 1985 года люди собрались во второй раз, называв эту встречу $ML\ Workshop$. Так состоялся первый $ML\ Workshop$, который проводится каждый год по сей день.

По мимо черновиков стандарта самого языка рассматривались предложения по системам ввода-вывода и модулей.

3.2. Standard ML (SML 90)

Работы над формальным описанием языка начались в районе 1986 года. Всего было в Эдинбурге было выпущено три издания стандарта языка. Работы велись преимущественно Робином Милнером, Робертом Харпером и учеником Робина Милнера Мэдсом Тофтэ.

- 8/87: The Semantics of Standard ML, издание 1.
- 8/88: The Definition of Standard ML, издание 2.
- 5/89: The Definition of Standard ML, издание 3.

Финальная версия спецификации языка была выпущена в 1990 году в издательстве MIT Press.

Одним из самых важных нововведений $SML90$, которое было предложено не в самом начале, была поддержка исключений как расширяемых типов данных: язык был расширен конструкторами исключений и сопоставлением с образцом (pattern matching) исключений в их обработчике.

Три ранние реализации $Standart\ ML$:

- $Cardelli\ ML$ (или $VAX\ ML$) можно назвать ”полустандартным” ML – в нём были поддержанны особенности языка, предложенные в 1983-84 годах.
- $Edinburgh\ ML$ => $Edinburgh\ SML$.
- Кэвин Митчелл (Kevin Mitchell), Алан Майкрофт (Alan Mycroft), Джон Скотт (John Scott) и Роберт Харпер (Robert Harper) переписали на $Cardelli\ ML$ компилятор $Cardelli\ ML$.
- Дэйв Мэтьюс (Dave Matthews) реализовал на SML синтаксический анализ для своего языка $PolyML$.

Более поздние реализации:

- Standart ML of New Jersey (SML/NJ);
- MLKit;
- Moscow ML;
- MLton.

3.3. Главные идеи в Standard ML ‘90

- Let-полиморфизм, вывод типов, наиболее общие (principal) типы: статьи Макса Ньюмана (1940е), Хаскелла Карри (1969) [16], Рождера Хиндли (1969) [15], Робина Милнера [17].

- Алгебраические типы данных и функции с клаузами, разбор случаев с помощью сопоставления с образцом (заимстовано из языка $Hope$).

- Модули и их сигнатуры, функторы, спецификации разделения данных и генеративные структуры (“strong structure sharing”): структуры с некоторой статической уникальной идентичностью, которые можно было сравнивать, т.е. на этапе компиляции проверять, что структуры равны. У этой особенности языка получилась очень сложная семантика, потому она не стала особо популярной.

- Исключения как расширяемый тип данных.

- Поддержка изменяемых значений (а именно ссылочных типов – ref types) используя понятие ”императивных переменных типа” (imperative type variables) привела к некоторому количеству сложностей в языке. Было написано большое количество статей на тему того как улучшить данных подход и почему от него стоит отказаться.

Эволюция алгебраических типов данных:

1. Неформальные описания данных, использованные в $ISWIM$.

2. Формальное развитие этих методов в работе Питера Лондина и Рода Берстолла “Programs and Their Proofs: An Algebraic Approach” [10] (полуформально описаны).

3. “Игрушечный” язык Берстаола NPL.

4. В языке $Hope$ более-менее полно сформировались и были заимствованы оттуда в $Standart\ ML$.

3.4. Ошибки в процессе дизайна

”Замораживание” формального описания языка в виде книги оказалось плохой идеей. Если язык программирования обретает реализацию и начинает использоваться, то его спецификацию необходимо поддерживать, а также (весьма осторожно) позволять языку изменяться. Описание должно быть открытым, но при этом очень аккуратно поддерживаемым. Возникновение этой проблемы связано со степенью участия Робина Милнера в проекте. Он был весьма занятым человеком, его основными проектами в то время были CCS (исчисление взаимодействующих систем, Calculus of Communicating Systems) и пи-исчисление. Ему хотелось дойти до состояния, когда он может передать рукопись в издательство MIT и перестать думать об этом проекте. Для своих целей он не планировал реализовывать, поддерживать и даже использовать этот язык. Следует помнить, что в те времена не существовало WWW, так что не было возможности выложить описание в онлайн.

Выпущенная книга про SML'90 имела очень короткое (2-3 страницы) приложение про “основы”, что по сути являлось стандартной библиотекой типов и функций. Список был очень короткий и состоял только из 43х пунктов, таким образом, каждый кто реализовывал SML должен был расширять этот перечень, и каждая реализация расширяла этот список несовместимым друг с другом способом. Процесс ”устаканивания” занял довольно длительное время, до тех пор пока Джон Реппи (John Reppy) не взялся за эту проблему и не создал “SML Basis Library”. Однако, эта работа вышла в свет лишь спустя несколько лет после SML'97.

3.5. Развитие языка в 1990е годы

В 1997 году институт Ньютона (Isaac Newton Institute for Mathematical Sciences), а именно программа по исследованию семантики вычислений собрала Робина Милнера, Роберта Харпера, Макса Тофтэ и Дэйва МакКуина в Кембридже, где они начали работу по переработке описания языка $Standart\ ML$ [19]. Вот значимые изменения:

- Сокращения типов в сигнатурах (реализовано в SML/NJ 0.93 и в $Caml\ Special\ Light$ [14]).
- Непрозрачное сравнение сигнатур.
- Слабое разделение сигнатур (подразумевает собой только равенство типов).
- Полиморфизм значений: уход от императивных типовых переменных, переход к ограничению полиморфизма значений (value restriction).
- Клонирование типов данных.

3.6. ML2000

Серия встреч между 1993 и 2000 годами была посвящена идее создания ML “следующего поколения”, но консенсус так и не был найден в основном из-за не согласия участников по поводу добавления объектно-ориентированных возможностей в язык. В то время уже существовали языки со своей собственной реализацией объектов такие как $OCaml$, а также язык $Moby$ за авторством Джона Рэппи (John Reppy) и Кэтлин Фишер (Kathleen Fisher). Основные предлагаемые изменения в языке были опубликованы в Principles and Preliminary Design of $ML2000$ [13].

4. OCaml

Изначально, “Caml” являлся акронимом для Categorical Abstract Machine Language (CAM) – языком для Категориальной Абстрактной Машины. Имя языка является слиянием названия машины и семейства языков, которому он принадлежит. Название $Caml$ сохранилось на протяжении всей эволюции языка, несмотря на то, что современная реализация не имеет никакого отношения к CAM.

$Caml$ был создан в исследовательской группой $Formel$ из $INRIA$, которой руководил Жерар Уэ (Gérard Huet). Его развитие продолжалось в исследовательских группах $Cristal$, и современной $Gallium$.

4.1. Истоки

Лаборатория $Formel$ начала интересоваться языком ML в начале 80х годов. В те времена ML использовался в LCF, а сам LCF был написан частично на $LISP$, частично на ML. В $Formel$ использовались несколько операционных систем ($Multics$, $Berkeley\ Unix\ on\ Vax$, $Symbolics$), поэтому Жерар Уэ решил создать реализацию ML, которая будет совместима одновременно с несколькими компиляторами $LISP$ ($MacLisp$, $FranzLisp$, $LeLisp$, $ZetaLisp$). В работу были вовлечены Гай Кузино (Guy Cousineau) и Ларри Паульсон (Larry Paulson). В придачу к созданию компилятора была существенна увеличена производительность языка.

Гай Кузино, следуя идеям Робина Милнера, также добавил в язык сопоставление с образцом и алгебраические типы данных, которые он заимствовал из языка $Hope$, спроектированного Родом Берстоллом и Дэйвом МакКуином. Некоторое время эта реализация называлась ”Le ML”, но это название не прижилось.

Около 1984 года, когда Робин Милнер предложил формальную спе-

цификацию ML, а Лука Карделли уже имел эффективную реализацию ML для FAM, Пьер-Луи Курье разработал исчисление категориальных комбинаторов, а также соответствие между ними и лямбда-исчислением. Это наблюдение использовал Гай Кузино для реализации эффективного способа компиляции в ML. Этот способ был близок к тому, что был использован в $Edinburgh\ ML$, но мог быть формально описан, доказан и по-простому оптимизирован. Это привело к появлению Категориальной Абстрактной Машины.

Таким образом Гаю Кузино была необходима новая реализация, основанная на CAM. Однако, в итоге получился не $StandartML$, а $Caml$. Почему? Главной причиной для разработки $Caml$ было то, что он использовался в лаборатории $Formel$ как основной язык для внутренней разработки. Также он использовался для разработки системы Coq, которая после появления выпускной работы Тьери Кокана (Thierry Coquand) стала основным направлений исследований в $Formel$. Группа не приняла ”стандартного” синтаксиса, потому что он мог помешать адаптации языка под текущие нужды. В частности, Филипп Лё Шенадэ (Philippe Le Chenadec) и Мишель Мони (Michel Mauny) разработали утилиты для манипуляции синтаксисом, которые оказались весьма полезными и были интегрированы в $Caml$. Обсуждение изменений, которые выглядели полезными, с разработчиками $StandartML$ внесло бы чересчур большую задержку в работу. Более того, философия ведения разработки противоречила ”стандартному” языку, где не предполагалось слишком быстрых изменений. Таким образом, все существенные улучшения, которые появлялись в $Standart\ ML$ и $Edinburgh\ ML$ вручную переносились в $Caml$.

4.2. Первая реализация

Первая реализация $Caml$ появилась в 1987 году и продолжала своё развитие до 1992 года. Она были сделана в основном Аскандером Суарезом (Ascander Suarez). После того, как он покинул лабораторию в 1988 году, Пьер Вей (Pierre Weis) и Мишель Мони (Michel Mauny) занимались развитием и поддержкой до 1992 года. Эта реализация компилировала $Caml$ в $LLM3$, виртуальную машину системы $Le\ Lisp$. Гай Кузино скромно вспоминает: ”Я должен признать, что когда начиналась разработка $Caml$, я имел весьма небольшой опыт в языках программирования. То, что мы взяли за основу систему $Le\ Lisp$ и её механизмы по выделению памяти и сборке мусора, сэкономило нам много времени, но не позволяло получить высокой производительности. Модель CAM позволяла быстро конструировать замыкания и разделять области видимости, но с доступом к окружению и применением оптимизаций были сложности. Также она могла порождать утечки памяти, так как ненужные значения сохранялись в замыканиях. Также, я не сразу осознал, что более важным является хорошая производительность не-функциональных программ, чем полностью написанных в функциональном стиле. Также я недооценил важности портируемости языка на другие платформы и открытости модели разработки. Несмотря на эти неувязки, ответственность за которых я беру на себя, Аскандер, Пьер и Мишель выполнили великолепную работу.”

4.3. Caml Light

В 1990 и 1991 годах Завье Леруа (Xavier Leroy) спроектировал абсолютно новую реализацию $Caml$, в основе которой лежал интерпретатор байт кода, написанный на C. А Дамьен Долигэ (Damien Doligez) спроекти-

ровал великолепную систему управления памятью. Эта новая реализация, получившая название *Caml Light*, была легко портируемой и запускалась на таких небольших домашних компьютерах как Macintosh и PC. Она заменила старую реализацию *Caml* и привела к росту популярности *Caml* в системах высшего образования и научно-исследовательских лабораториях. Поддержка потоков данных и технологии синтаксического анализа (благодаря Мишелю Мони) позволила команде *Formel* создать необычные системы управления синтаксисом.

4.4. Caml Special Light

В 1995 году Завье Леруа анонсировал *Caml Special Light*, который улучшил *Caml Light* в нескольких аспектах. Во-первых, это оптимизирующий компилятор в машинный код, который был сравним или превосходил производительность лучших на тот момент компиляторов функциональных языков и позволил *Caml* соревноваться в производительности с такими языками как *C++*. Во-вторых, в *Caml Special Light* была высокоуровневая система модулей, спроектированная Завье Леруа на основе системы модулей в *SML*. Эта система предлагала богатые возможности абстракции и программирования в целом.

4.5. Objective Caml

Системы типов и системы вывода типов для объектно-ориентированных языков были популярной темой для исследования в начале 1990х годов. Дидье Рэми (Didier Rémy), к которому позднее присоединился Жером Вульон (Jérôme Vouillon), разработал элегантную и выразительную систему типов для объектов и классов. Реализация была интегрирована в *Caml Special Light*, что привело к появлению в 1996 году реализа-

ции языка $Objective\ Caml$, который был переименован в $OCaml$ в 2011 году. Он является первым языком, который сочетает выразительность объектно-ориентированного программирования со статической типизацией в стиле ML и выводом типов. Он поддерживает большинство современных идиом объектно-ориентированного программирования (параметрические классы, бинарные методы, специализация) статически и типобезопасно, в то время как в других языках ($C++$ и $Java$) это приводит к противоречивости системы типов или необходимости проверок во время выполнения.

В 2000 году Жак Гариг (Jacques Garrigue) расширил $Objective\ Caml$ некоторыми возможностями, над которыми экспериментировал в диалекте $Objective\ Label$. Среди них полиморфные методы, именованные и опциональные параметры функций, полиморфные варианты.

Расцвет OCaml пришелся на конец 1990х, когда он начал набирать популярность и привлекать существенное количество пользователей.

В 1996 году Гай Кузино писал: ”Конечно, история $Caml$ могла быть более линейной. Однако, методом проб и ошибок, во Франции появился функциональный язык программирования достаточно гибкий, портируемый и обладающий хорошей производительностью.”

5. Заключение

В рамках реферата была рассмотрена история появления основных языков семейства ML: $Standart\ ML$ и $OCaml$, был рассмотрен контекст и нужды, для которых эти языки создавались, очерчены родственные связи этих языков с современными императивными языками программирования. Также была предпринята попытка проанализировать ошибки, совершенные при проектировании языков. Были обозначены цели,

к которым надо стремиться при проектировании нового языка программирования.

Развитие языков функционального программирования и, в частности, *ML*, является перспективной темой для проведения исследований, так как наличие формальной семантики у этих языков, и, как следствие, появление систем-помощников в доказательстве теорем, позволяет создавать программы с формально доказуемой корректностью. А доказанная корректность является сама по себе очень полезным свойством для практического программирования, в частности, при разработке высоконадёжных систем (в ядреной энергертике, космической отрасли, медицине и т.д.).

6. Литература

Список литературы

[1] A.N. Whitehead, B. Russell Principia mathematica 1 (1 ed.), Cambridge: Cambridge University Press, 1910.

[2] M. Richards, The BCPL Reference Manual, Memorandum M-352, Project MAC, Cambridge, MA, USA, 1967.

[3] C. Strachey. Fundamental concepts in programming languages, Lecture Notes, International Summer School in Computer Programming, Copenhagen, 1967.

[4] C. Strachey. Time sharing in large fast computers, UNESCO Conference on Information Processing, Paris, 1959.

[5] D. Scott, C. Strachey. Toward a mathematical semantics for computer languages, Oxford Programming Research Group Technical Monograph, PRG-6, 1971.

[6] Peter J. Landin. The mechanical evaluation of expressions, The Computer Journal, British Computer Society 6 (4): pp. 308–320, 1964.

[7] Peter J. Landin. Correspondence between ALGOL 60 and Church's Lambda-notation: part I, Communications of the ACM 8 (2): pp. 89–101, 1965.

[8] Peter J. Landin. "A Generalization of Jumps and Labels, UNIVAC Systems Programming Research (technical report).

[9] Peter J. Landin. The next 700 programming languages, Communications of the ACM 9 (3): 157–166, 1966.

[10] R. M. Burnstall, Peter J. Landin. Programs and their proofs: an algebraic approach, Machine Intelligence 4, 1969.

[11] R. M. Burstall, D. B. MacQueen, D.T. Sannella. Hope: An Experimental Applicative Language, Conference Record of the 1980 LISP Conference, Stanford University, pp. 136-143, 1980.

[12] J. Darlington, R. Burnstall, . ”Program Transformation and Synthesis: Present Capabilities”. Research Report No. 77/43, Dept. of Computing and Control, Imperial College of Science and Technology, 1977.

[13] The ML2000 Working Group. Principles and a Preliminary Design for ML2000, 1999.

[14] X.Leroy. Manifest types, modules, and separate compilation, Proc. 21st Symp. Principles of Programming Languages, pp. 109–122, 1994.

[15] J. R. Hindley, The principal type scheme of an object in combinatory logic, Transactions of the American Mathematical Society (American Mathematical Society) 146: 29–60, 1969.

[16] H. B. Curry. A Theory of Formal Deducibility, J. Symbolic Logic Volume 34, Issue 1, 1969.

[17] R. Milner. A theory of type polymorphism in programming, Journal of Computer and System Sciences, 17:348-375, 1978.

[18] R. Milner. A proposal for standard ML, LFP ’84 Proceedings of the 1984 ACM Symposium on LISP and functional programming, 1984

[19] R. Milner, M. Tofte, R. Harper and D. MacQueen, The Definition of Standard ML (Revised), MIT Press, 1997.

[20] D. MacQueen, The history of Standart ML: ideas, principles, culture, ML Family Workshop, 2015.