Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

ОТЧЕТ

О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Анализ синтаксиса и семантики стековых языков программирования

Студент ИУ6-73Б _____ В.К. Залыгин
(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель _____ Б.И. Бычков
(Подпись, дата) (И.О. Фамилия)

Оценка _____

2025 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме ___Анализ синтаксиса и семантики стековых языков программирования

Студент группы ИУ6-73Б

Залыгин Вячеслав Константинович
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)
исследовательская

Источник тематики (кафедра, предприятие, НИР) ___кафедра

График выполнения НИР: 25% _4_ нед., 50% _7_ нед., 75% _11_ нед., 100% 14 нед.

Техническое задание: *___выполнить анализ способов представления данных в распределенных системах, осуществить выбор способов для хранения и обработки данных об успеваемости студентов в электронном университете*

Оформление научно-исследовательской работы:

1) Отчет на 25-30 листах формата А4.
2) Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
___Необходимый иллюстративный графический материал включить в качестве рисунков в расчетно-пояснительную записку
3) Приложение А. Техническое задание на ВКРБ на 5-8 листах формата А4.

Дата выдачи задания « 1 » сентября 2025 г.

Руководитель _____ (Подпись, дата) В.К. Залыгин (И.О. Фамилия)

Студент _____ (Подпись, дата) Б.И. Бычков (И.О. Фамилия)

РЕФЕРАТ

РПЗ 34 с., 5 рис., 1 табл., 13 источн., 0 прил.

СТЕК, КОМПИЛЯТОР, СТЕКОВЫЙ ЯЗЫК, ЯЗЫК ПРОГРАММИРОВАНИЯ, ОБРАТНАЯ ПОЛЬСКАЯ ЗАПИСЬ

Объектом анализа являются стековые языки программирования.

Цель работы – проанализировать существующие подходы к построению стековых языков программирования, сделать анализ синтаксиса и семантики языков прграммирования, выявить идеи, которые лежат в основе построения компиляторов для данных языков.

В результате работы выполнен аналитический обзор таких аспектов стековых языков как: область применения, модель исполнения, используемые синтаксические конструкции и их семантика, типизация, статический и динамический анализ программ, возможные оптимизации, работа с памятью и подходы к построению стандартной библиотеки.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Компилятор

Стек

Виртуальная машина

Обратная польская запись

Нитевой код

Лексема

ЯП

Моноид

Гомоморфизм

Комбинатор

.NET CIL

Алгоритма Хиндли–Милнера

Копирующий сборщик мусора

Mark-sweep-compact

Терминал

Нетерминал

ВВЕДЕНИЕ

Стековые (или стек-ориентированные) языки программирования характеризуются применением стека данных в качестве основного механизма передачи информации и хранения результатов вычислений. Стек-оринетированность позволяет программам на таких языках выглядеть компактно и эффективно исполняться. Исторически первым стековым языком стал Forth, разработанный Чарльзом Муром в начале 1970-х годов. Язык Forth изначально создавался для системного и низкоуровневого программирования [1], но в целом позволяет писать достаточно выразительный и понятный высокоуровневый код. Forth наиболее часто используется именно при разработке встроенных устройств. Например, этот язык применялся в ряде космических миссий NASA (включая проекты Voyager и Deep Impact), где программные системы работали на специализированных процессорах со стековой архитектурой (Harris RTX2000/2010) [1].

В начале 2000-х годов возрос интерес исследователей к более высокоуровневым стековым языкам. Был предложен термин «конкатенативность» для обозначения семейства стековых языков, в которых программа воспринимается как функция, преобразующая последовательность аргументов в последовательность результатов, а конкатенация функций в тексте эквивалентна их композиции во время выполнения (отсюда и название – конкатенативные языки). Язык Joy, разработанный Манфредом фон Туном и выпущенный в 2001 году, полностью избегает переменных, предлагает фиксированный набор комбинаторов для работы со стеком [2]. Joy позволил заложить в теорию стековых языков строгие формальные основы. Дальнейшим развитием идей Joy стал язык Factor, созданный Славой Пестовым и впервые опубликованный в 2003 году. В отличие от Forth и ранних стековых языков, Factor позиционируется как современный высокоуровневый язык общего назначения: он динамически типизирован, поддерживает объектно-ориентированные конструкции и автоматическое управление памятью, что делает его пригодным для создания как скриптов, так и крупных приложений [3]. Несмотря на относительную узость области применения по

сравнению с наиболее популярными языками, стековые языки продолжают эволюционировать. Их принципы легли в основу ряда виртуальных машин. Так, современный байткод WebAssembly представляет собой стековую виртуальную машину, предназначенную для эффективной работы в Web-среде [4]. Виртуальные машины языков Java (JVM) и C# (.NET) также используют стековые языки для близкого к машине представления программ.

В целом, благодаря экономичности и простоте реализации (а значит, и простоте портирования на разнообразные архитектуры) стековые языки нашли своё применение в низкоуровневых системах (системы реального времени, встраиваемые системах), различных виртуальных машинах (WebAssembly, JVM, .NET), графических процессорах и других специализированных областях.

1 Определение модели исполнения

Модель исполнения стекового языка – это некая (чаще всего виртуальная) стековая машина, имеющая хотя бы один стек данных и набор инструкций для работы со стеком. Например, реализация Forth оперирует двумя стеками: стеком данных для хранения аргументов и результатов и стеком возвратов для адресов возврата при вызове подпрограмм [1]. Программа представляет собой последовательность слов, которые могут быть либо встроенными примитивами (например, арифметические операции, операции над стеком), либо определенными пользователем субпрограммами. Каждое слово либо модифицирует состояние стека, либо изменяет поток исполнения программы. Большинство инструкций в такой машине берёт необходимые операнды с вершины стека, выполняет вычисление и кладёт полученный результат обратно на стек. Когда выполнение программы завершено, результат обычно считается находящимся на вершине стека, откуда его можно извлечь для дальнейшего использования.

Поток исполнения программы контролируется при помощи счетчика команд (program counter), который указывает на текущую команду. Для большинства команд поток исполнения линеен: после выполнения одной команды машина берет на исполнение следующую команду (то есть инкрементирует значение счетчика до следующей команды). Исключение составляют команды условного и безусловного переходов, в некоторых реализациях стековых машин также присутствуют команды циклов. Для таких команд машина может менять значение счетчика неким особым образом согласно семантике конкретной команды. Следует обратить внимание, что отсутствие команд циклов не означает невозможность итерирования: в таком случае цикличность исполнения может быть достигнута при помощи команд ветвления и рекурсивных вызовов.

Таким образом модель исполнения (машина) содержит в себе следующие компоненты:

– набор стеков данных (один или более);

– стек возвратов;

– счетчик команд;

– словарь поддерживаемых команд (в число которых могут входить команды модификации стека данных, а также команды модификации стека возвратов и счетчика команд), который может быть расширен в процессе исполнения программы.

2 Аналитический обзор существующих решений

Начиная с семидесятых годов прошлого века было создано достаточно большое количество стековых языков. Далее рассмотрены несколько значимых языков, каждый из которых привнес важные нововведения в группу языков. Для языков рассматриваются аспекты синтаксиса, типизации, работы с памятью и стандартной библиотекой, особенности каждого языка.

2.1 ЯП Forth

Forth представляет собой один из первых стековых конкатенативных языков программирования. Логотип языка представлен на рисунке 1. Forth создан Чарльзом Муром в начале 1970-х годов как сочетание расширяемого языка и интерактивной методологии разработки. Основная идея состоит в минимальном ядре и словаре «слов» (подпрограмм), через которые реализуется как высокоуровневая логика, так и низкоуровневое аппаратное управление; новые слова добавляются прямо во время работы системы, что позволяет подстраивать язык под конкретную предметную область. Синтаксис несложен: используется обратная польская нотация, программа – это последовательность слов, разделённых пробелами. Интерпретатор читает токены, ищет их в словаре и либо выполняет связанный код, либо интерпретирует токен как число и кладёт его на стек.

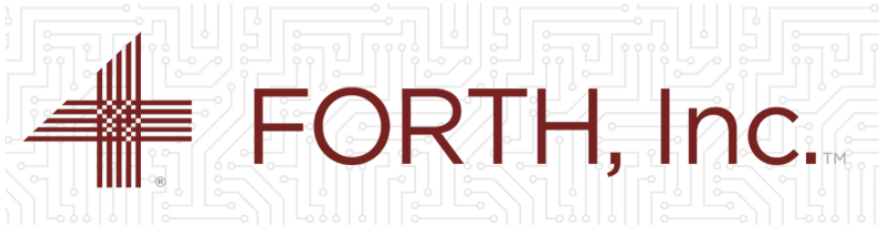

Рисунок 1 — Логотип Forth

С точки зрения семантики типов Forth рассматривается как «безтиповый» [5]: данные представляются машинными словами фиксированной разрядности, а язык не навязывает проверку согласованности типов – ответственность за корректность интерпретации содержимого стека возлагается на разработчика. Такой подход обеспечивает максимальную гибкость и предсказуемое временное поведение, но затрудняет статический анализ и контроль. Forth применяется в встроенных и ресурсно-ограниченных

системах, прошивках, системах реального времени и космической технике: он используется в контроллерах космических аппаратов и приборов NASA, а также в реализациях Open Firmware (стандарт, регламентирующий принцип описания аппаратной конфигурации устройств) для платформ Apple, IBM и Sun. К языка особенностям относятся малый размер полной среды (компилятор, интерпретатор и редактор умещаются в память 8-битных систем), компиляция в нитевой код (представление программы, полностью состоящее из последовательности вызовов подпрограмм) для ускоренной интерпретации, а также единство языка и среды: Forth одновременно служит командной оболочкой, компилятором и минимальной операционной системой, а его стандарт (ANS Forth) описывает лишь набор слов и модель стека, оставляя пользователю свободу строить поверх ядра собственные DSL и диалекты [1].

Стандартной библиотеки у языка в привычном понимании нет – расширение возможностей происходит за счет включения в среду дополнительных наборов слов.

Определения новых слов задаются конструкцией вида «: NAME … ;», причём большинство лексем – от переменных до управляющих конструкций – являются словами. Стандартный «core word set» ANS Forth [5] включает базовые стековые операции («DUP», «DROP», «SWAP», «OVER»), арифметику («+ - * /»), сравнения, примитивы работы с памятью («@» – чтение по адресу, «!» – запись по адресу), конструкции ветвления и циклов («IF … ELSE … THEN», «DO … LOOP»), а также определяющие слова «CREATE», «VARIABLE», «CONSTANT» и др.; остальные word set’ы стандарта являются опциональными расширениями. Почти все управляющие слова в Forth реализуются при помощи понятия слов времени компиляции – это конструкции, которые во время компиляции компилятор разворачивает в другие слова. Например, конструкция с ветвлением «… DUP 6 < IF DROP 5 ELSE 1 - THEN …» разворачивается в последовательность слов «… DUP LIT 6 < ?BRANCH 5 DROP LIT 5 BRANCH 3 LIT 1 - …» – конструкция условного перехода заменена на слова условного и безусловного переходов [1].

Поскольку Forth является интерактивной средой, процесс исполнения программы можно разделить на 2 составляющие: состояние интерпретации и состояние компиляции. В состоянии интерпретации среда занимается исполнением примитивных слов. В то же время, если среда в процессе встречает некоторый набор слов (например, «:» – создание нового слова), то она переходит в состояние компиляции и осуществляет разбор подпрограмм, встречаемых синтаксических конструкций. Выполнить переключение состояния также можно при помощи специальных слов «[« – переход в состояние интерпретации и «]» – переход в состояние компиляции.

В листинге 1 определяется слово «HELLO», которое затем вызывается и выводит в консоль текст. Круглые кавычки используются для комментариев.

Листинг 1 — Определение и использование слова «HELLO»

```
: HELLO  CR ." Hello, World!" ;
HELLO ( выводит Hello, World! в консоль на следующей строке
после вызова слова )
```

В итоге язык Forth дает ряд возможностей при программировании систем с ограниченными ресурсами. Множество низкоуровневых слов, ряд оптимизаций, точность и гибкость исполнения позволяют писать маленькие и быстрые программы [1].

2.2 ЯП Joy

В 2001 году Манфред фон Тун в La Trobe University представил язык Joy, как попытку формализации логики стековых языков. Joy – функциональный стековый конкатенативный язык программирования [2].

В отличие от обычных функциональных языков, которые строятся вокруг операции применения функции к аргументу, Joy использует операцию композиции функций [6]. Каждая программа в Joy обозначает унарную функцию вида «stack -> stack»: на вход подаётся состояние (стек данных), в процессе применения команды происходит некая модификация стека и его передача следующей команде, значения и подпрограммы в свою очередь передаются через стек.

С математической точки зрения это формализуется следующим образом: множество всех программ образует синтаксический моноид по операции

конкатенации (ассоциативная операция «склейки» программ и пустая программа как нейтральный элемент), а множество функций «stack -> stack» – семантический моноид по операции композиции и тождественной функции в роли нейтрального элемента [6]. Отображение, которое каждой программе сопоставляет её «смысл» (как некоторую функцию над стеком), является гомоморфизмом моноидов, то есть сохраняет операцию: смысл «P Q» равен композиции смыслов «P» и «Q», а смысл пустой программы – тождественная функция [7].

Синтаксис Joy минималистичен и основан на постфиксной записи [2]. Программа – это последовательность слов, разделённых пробелами, исполнение идёт слева направо. Каждое слово выполняет некоторое преобразование над стеком данных. Операторы (арифметические, логические и другие) снимают одно или несколько значений с вершины стека и помещают результат обратно. Для структур данных используются литералы: списки и цитаты программ записываются в квадратных скобках «[…]», множества – в фигурных, строки – в кавычках. Цитата – это значение, содержащее в себе фрагмент программы как данные, который можно затем анализировать или исполнить [2]. Определения новых слов записываются как равенства: «square == dup * .» определяет слово «square», которое дублирует вершину стека («dup») и перемножает два верхних элемента («*»). Joy при этом остаётся функционально чистым: стандартные операции не изменяют скрытое глобальное состояние, а только преобразуют стек, так что одну и ту же программу можно рассматривать как математическую функцию «stack -> stack» без побочных эффектов [6].

При построении идиоматических программ на Joy активно используются комбинаторы. В контексте языка это стековые функции, которые принимают одну или несколько цитат программ (списков слов) и управляют их исполнением различными способами [2]. Например, функция-комбинатор «i» («interpret») берет вершину стека и исполняет список слов, которые лежали внутри вершины. С точки зрения синтаксиса это эквивалентно опусканию скобочек: «[1 print] i» значит то же самое, что и «1 print». Для ветвлений

используется комбинатор «ifte», для циклических алгоритмов – комбинаторы различных схем рекурсии «primrec», «linrec», «binrec». Программист волен создавать и свои функции-комбинаторы на основе уже существующих через механизм определения новых слов.

Joy является динамически типизированным языком. В язык встроены числовые (целые и вещественные числа), агрегатные типы (строки, списки и неупорядоченные множества) и тип цитат [2]. Типизация стека на этапе компиляции программы никак не контролируется и проверка типов происходит по факту в момент применения той или иной функции. В случае несовпадения ожидаемого и фактического типов выдается ошибка времени выполнения [8] (в отличие от поведения программ на Forth, где произойдет недопустимая реинтерпертация данных и вследствие нее уход неопределенной поведение). Для борьбы с ошибками несовпадения типов рекомендуется к каждому определению в языке приписывать комментарий в следующей нотации: пишется название функции, затем после двоеточия ее эффект на стек – какой стек функция принимает и какой стек отдает. Например, нотация функции «dup : A -> A A» и функции «+ : Int Int -> Int». Нотация помогает рассмотреть программу как формулу в рамках алгебры стеков (то есть алгебры, которая строится над вычислениями со стеками в качестве переменных) [7]. Такая формализация помогает применять к программам на Joy многие методы теории вычислимости, что является предтечей статического анализа программ.

Благодаря возможности определять новые слова, которые могут быть комбинаторами или другими функциями, Joy имеет достаточно разветвленную стандартную библиотеку. В стандартной поставке есть несколько библиотек, разбитых по областям применения – например, «agglib» и «seqlib» содержат обобщенные операции над агрегатами (в контексте языка – неупорядоченные множества, строки и списки) и последовательностями, «numlib» – числовые функции и численные методы, «mtrlib» – функции для работы с матрицами [8].

Язык Joy имеет в основном академическое и эксперементальное применением, так как служит для демонстрации применения идей функционального программирования к стековой модели исполнения. Базовая

реализация – интерпретатор на C с автоматической сборкой мусора и набором библиотек. В итоге Joy можно рассматривать как компактную и хорошо формализованную модель стекового языка высокого уровня, где математические понятия моноида и гомоморфизма используются не как абстрактные термины, а как точное описание связи между текстом программы и её поведением при выполнении.

2.3 ЯП Factor

Среди стековых языков Factor занимают нишу языков общего назначения [3]. Первая версия языка была выпущена в 2003 году Славой Пестовым. Логотип языка показан на рисунке 2. Factor заимствует многие элементы из языка Joy:

– конкатенативный синтаксис;

– использование комбинаторов для управления потока исполнения;

– аналогичный способ создания собственный комбинаторов при помощи цитат.

Рисунок 2 — Логотип Factor

Как и Joy, Factor является динамически-типизированным языком [3]. Документация и сообщество языка активно используют нотацию «stack effects», при помощи которой описывается, как то или иное слово при применении модифицирует стек. Таким образом оптимизирующий компилятор может проверить объявленные эффекты и отклоняет определения, для которых эффект не удаётся вывести, трактуя это как ошибку компиляции. Внутренний механизм этой проверки описывается как абстрактная интерпретация программы: стек-чекер симулирует эффекты слов на абстрактном стеке, при встрече ветвлений анализирует обе ветви и унифицирует состояния, а несовместимость высоты/формы стека превращается в ошибку времени

компиляции. Такая «проверка стека» играет своего рода статический анализатор, который ловит классы ошибок, типичные для стековых языков, и одновременно создаёт фундамент для агрессивных оптимизаций [3].

В целом язык предлагает множество высокоуровневых и низкоуровневых оптимизаций. Оптимизирующий компилятор состоит оптимизирующего фронтенда, который строит промежуточное представление языка, использующееся для работы стек-чекера и высокоуровневых оптимизаций, а также бекенда, который строит непосредственно исполняемый код для машины, занимается низкоуровневыми оптимизациями [3].

В отличие от Joy стандратная библиотека Factor обладает гораздо большими размерами и количеством возможностей. Именно благодаря обширной стандартной библиотеке, за счет которой решается множество прикладных задач, язык и носит название прикладного. Также для языка существует большое количество различных инструментов разработки, сопряженные в IDE: интерактивный отладчик, браузер документации, инспектов объектов, механизм модификации программы без ее перезапуска [9].

Для работы с памятью язык использует сборщик мусора, встроенный в виртуальную стекову машину языка. Для молодых объектов используется копирующий сборщик, для старшего поколения используется алгоритм «mark-sweep-compact» [3]. Для работы с неуправляемыми ресурсами используются библиотека «destructors» и комбинатор «with-destructors».

В листинге 2 показан пример определения слово для вычисления факториала. Наглядно видно использование нотации «stack -> stack» после определения названия слово, а также использование комбинатора «if» и цитат.

Листинг 2 — Определение слово вычисления факториала

```
: factorial ( n -- n! )
dup 0 =
[ drop 1 ]
[ dup 1 - factorial * ]
if ;
```

2.4 ЯП Cat

Еще одним логическим продолжением Joy является экспериментальный язык Cat (логотип языка показан на рисунке 3), который с 2006 года разрабатывается Кристофером Диггинсом. Язык используется для верификации и оптимизации программ под платформу .NET CIL [10]. Язык наследует идею Joy: любая программа рассматривается как функция вида «stack -> stack», а основная операция — композиция таких функций, реализуемая простой конкатенацией лексем в исходном тексте программы. Cat вводит статическую систему типов поверх этой модели. Это означает, что в момент компиляции компилятор может вычислить размер стека данных и его типизацию для любого момента времени исполнения программы и верифицировать программу на соответствие ожидаемым типам на стеке и фактическим [10]. Таким образом при помощи статического анализа возможно предовратить фатальные ошибки времени выполнения, связанные с несовпадением типов, опустошением или переполнением стека. Также статический анализ открывает дороги агрессивным оптимизациями, которые могут менять тексты программ и промежуточных представлен для повышения производительности.

Рисунок 3 — Логотип Cat

Аналогично Joy, программы на языке Cat опираются на обратную польскую запись с последовательным перечислением слов программы слева направо [10]. Работа с поток исполнения устроена при помощи комбинаторов «if» и «while», соответствующих ветвлению и циклу по условию: комбинаторы принимают набор цитат (которые аналогично Joy записываются в квадратных скобках; еще цитаты называют замыканиями) и огранизуют поток управления должным образом. Определение новых слов возможно при помощи слова

«define» – определение слов глобально и единственно (множественные определения запрещены).

В множестве команд языка доступны арифметические, логические операции, операции сравнения, операции манипуляции стеком и наконец комбинаторы (или же, как привычнее для Cat, – функции высшего порядка). Также существуют примиты работы с последовательностями, рекурсивными алгоритмами.

Наибольшего внимания заслуживает система типов Cat, благодаря которой язык обзавелся статическим анализом. Система типов Cat описывает поведение функций в терминах потребления и производства элементов стека. Тип функции определяется тем, что функция ожидает на стеке перед началом выполнения, и тем, что она оставляется на стеке после выполнения. Тип записывается как стрелка между конфигурациями стека [11], например «(int int -> int)» для операции сложения или «(’a ’S -> ’a ’a ’S)« для дублирования вершины стека. Слева указываются типы, которые должны находиться на вершине входного стека, справа — типы элементов на вершине выходного стека. Типы с апострофом (’a, ’b) обозначают типовые переменные (для которых точный тип не специализирован из-за ненадобности), а специальные «типовые векторы», обозначаемые заглавными буквами (’A, ’B), представляют произвольные последовательности типов, то есть «хвост» стека. Такое представление соответствует row-полиморфизму: каждая функция не только определяет, какие значения она снимает и кладёт на вершину стека, но и неявно пропускает через себя остальную часть стека, обозначаемую переменной «ряда» типа; это позволяет типизировать локальные преобразования независимо от длины и состава «фона» стека. Диггинс формализует эту систему через различие между «родами» (kinds) для обычных типов и для стеков значений: типы значений и типы стеков образуют две категории, между которыми определяются функции «stack -> stack», а типизация выражается в виде правил натурального вывода (правила, которые задают соответствие между набором посылок – входной конфигурацей стека и выводом из этого – выходной конфигурацией стека;

для каждого слова определен свой набор таких правил) и полиморфной системы родов, позволяющей описывать как отдельные значения, так и целые стековые конфигурации [11]. Вывод типов на основе правил в Cat базируется на вариации алгоритма Хиндли–Милнера, обобщённой на стековые типы и row-полиморфизм, причём языковые конструкции допускают полиморфизм более высоких рангов (то есть возможен логический вывод для программ, где используются функции высшего порядка), а аннотации типов (так называемые «stack diagrams») используются как документация и как данные для статического анализа [11].

В статье [10] описывается чистое подмножество языка Cat с правилами вывода, показанными в листинге 3. При выводе типов алгоритм проходится по словам и находит тип каждой композиции из слов. Затем проводится унификация типов согласно алгоритму Хинди-Милнера.

Листинг 3 — Правила вывода основных слов в Cat

```
pop : ('a -> )
dup : ('a -> 'a 'a)
swap: ('a 'b -> 'b 'a)
if  : ('A bool ('A -> 'B) ('A -> 'B) -> 'B)
eval: ('A ('A -> 'B) -> 'B)
while:('A ('A -> 'A) ('A -> 'A bool) -> 'A)
```

Статический анализ в Cat не ограничивается проверкой согласованности типов. Поскольку типы фиксируют не только типы элементов, но и форму стека до и после применения функции, система также гарантирует отсутствие недопустимых ситуаций, таких как недостаток аргументов на стеке или несовместимость конфигураций стека в ветвлениях [11]. Для слова условного оператора «if» тип требует, чтобы обе ветви, принимая на вход одну и ту же конфигурацию стека, возвращали стек одинаковой структуры, в противном случае выражение считается некорректным (например, одна ветвь возвращает строку, а другая — число). Для слова цикла «while» тип требует, чтобы после выполнения цикла конфигурация стека осталось такой же, как была на момент начала цикла. Дополнительно вводится различие между чистыми и побочными функциями при помощи двух видов стрелок (-> и ~>): функция считается имеющей побочный эффект, если в её реализации используется хотя бы одна

побочная операция [11]. Эта информация также фиксируется в типах и может использоваться оптимизатором или проверяющими инструментами.

Область применения Cat складывается из двух направлений. Во-первых, язык используется как исследовательская и учебная платформа для изучения конкатенативного программирования, типовых систем с выводом типов и row-полиморфизма в стековой модели. Работы по типизации функциональных стековых языков прямо опираются на Cat и рассматривают его как эталонный пример статически типизированного конкатенативного языка. Во-вторых, Cat разрабатывается как промежуточный язык для компиляторов и инструментов анализа. Хотя язык не получает широкого промышленного распространения, он влияет на дальнейшие разработки в области конкатенативных языков и типовых систем. В итоге Cat формирует пример полноценно типизированного стекового языка, в котором идея «программа как функция stack → stack, конкатенация как композиция» соединяется с сильной статической системой типов, ориентированной на анализ и оптимизацию.

2.5 ЯП Wasm

В 2017 году группа компаний, в которую входит W3C, Mozilla, Google, Microsoft, Apple, представила язык WebAssebmly (Wasm). Логотип Wasm показан на рисунке 4. Wasm представляет собой низкоуровневый переносимый байткод (набор под виртуальную стековую машину, запускаемую преимущественно (но не только) в веб-браузерах наравне с движками EcmaScript для исполнения клиентского кода [4]. Это первый из представленных в обзоре языков, который являются целью компиляции, а не сам компилируется во что-либо. Поддержка компиляции в Wasm имеется для множества высокоуровневых языков: в первую очередь для таких как C/C++/Rust, которые имеют маленький (или вообще не имеют) рантайм. В исходных целях проектирования Wasm фиксируются требования к компактности двоичного представления, быстрой однопроходной валидации и компиляции, а также к «песочнице» с предсказуемой семантикой, пригодной для запуска недоверенного кода [4].

Рисунок 4 — Логотип Wasm

Байткод на Wasm в основном поставляется в виде бинарного представления, которое в то же время возможно транслировать в текстовое человеко-читаемое представление. Текстовый вариант основан на записи S-выражений [12] (еще одна черта, которая отличает этот язык от представленных ранее). Листинг 4 показывает определение функции сложения числа с самим собой на wasm в человеко-читаемом варианте.

Листинг 4 — Определение функции сложения числа с самим собой

```
(func (param $p i32)
  (result i32)
  local.get $p
  local.get $p
  i32.add
)
```

Как стековый язык исполнения Wasm опирается на неявный операндный стек и структурированное управление потоком [4]. Инструкции потребляют значения с вершины стека и помещают результаты обратно, при этом реализация не обязана хранить «настоящий» стек: спецификация прямо описывает интерпретацию стека как набора анонимных регистров, а статическая типовая проверка делает высоту стека известной на этапах валидации и компиляции. Управление потоком задаётся структурными конструкциями (block, loop, if) и переходами к меткам (br, br_if, br_table). Единица трансляции Wasm называется модулем [4].

Типизация в Wasm является статической и рассчитана на проверку до исполнения. Функции имеют явные типы параметров и результатов, инструкции типизируются через эффекты над стеком (как и в языке Cat), а модуль проходит валидацию, гарантирующую корректность применения инструкций к операндам нужных типов и отсутствие underflow ошибок.

Верификация модуля происходит за один проход [4], что соответствует целям языка по скорости исполнения.

Язык и вирутальная машина используются в качестве безопасной песочницы с производительностью низкоуровневого кода, в основном исполняемой в веб-браузерах. По сравнению с традиционным для браузеров языком EcmaScript, низкоуровневый стековый байткод занимает меньше пространства и быстрее работает, так как требует меньше накладных расходов (по сравнению с движками EcmaScript, например, V8).

Поскольку язык является низкоуровневым, понятие стандартной библиотеки к нему не применяется. Вместо встроенных API используется модель импортов: окружение предоставляет функции и объекты, а модуль экспортирует точки входа.

Управление памятью осуществляется в ручном режиме: так как язык Wasm является целью компиляции, то ответственность за гарантии при работе с памятью ложится на средства исходного языка [4] (такие как статические проверки Rust, сборщики мусора или, в случае, например, C/C++ сами разработчики ПО).

2.6 Выводы

Вышеописанные обзоры языков позволяют проследить эволюцию стековых языков. Начавшись с близкого к аппаратной части Forth, они и заложенные в них идеи развились и со временем нашли применение в самых разных системах: от встраиваемого ПО до современных виртуальных машин и форматов переносимого кода. Основными свойстваим стековых языков счиаются простота реализации и переносимости, возможность строгой верификации корректности программ, а также минимальные накладные расходы рантайма.

Рассмотренные языки со временем приобретают большие возможности статического анализа, что открывает дополнительные возможности верификации и оптимизации программ. Благодаря этому стековые языки становятся быстре и безопаснее для использования, что вместе с изначально им присущей высокой переносимостью и простой реализации стековой машины, дает

большие преимущества в развитии как низкоуровневых языков для различного рода машин.

Таблица 1 собирает структурирует информацию об вышеописанных языках.

Таблица 1 — Сводная характеристика рассмотренных стековых языков

Язык	Синтаксис	Типизация	Анализ и оптимизации	Стандартная библиотека	Область применения
Forth, 1970	Постфиксный конкатенативный	Безтиповой	Наличие базовых оптимизаций (нитевой код), статического анализа нет	Наборы слов (ANS core + расширения)	Встроенные системы
Joy, 2001	Постфиксный конкатенативный	Динамическая типизациция	Задана формальная модель языка	Базовые слова + библиотеки слов	Экспериментальный язык
Factor, 2003	Постфиксный конкатенативный	Динамическая типизация	Есть stack-checker и агрессивные оптимизации	Обширная стандратная библиотека и инструментарий разработки	Язык общего назначения
Cat, 2006	Постфиксный конкатенативный	Статическая типизация стека по форме и размеру	Статический вывод типов и эффектов стека, верификация программ	Минимальная стандратная библиотека	Эксперементальный язык, верификация программ .NET CIL
Wasm, 2017	S-выражения	Статическая типизация	Статический анализ за один проход, верификация, JIT/AOT оптимизации в рантайме	Нет, но есть импортируемые API	Цель компиляции для Web-среды и изолированных рантаймов

3 Анализ синтаксиса

Для большей части языков, рассмотренных в обзоре, характерен «конкатенативный» тип синтаксической организации: программа представляется линейной последовательностью лексем (или слов, комбинаторов, функций), а их конкатенация в тексте соответствует композиции набора преобразований неявного стека данных. В такой модели вызов подпрограммы, как правило, не имеет самостоятельной скобочной формы вида «f(x)», а выражается отдельным токеном. В конкатенативных языках у фукнций всегда есть один и только один неявный аргумент – стек данных. В формальных описаниях конкатенативных языков фиксируется именно списковая структура выражений, где единицей синтаксиса выступает слово, а выражение задаётся как список слов с возможными литералами и блоками (цитатами) [13]. Исключение составляют некоторые конструкции времени компиляции, которые затем преобразуются в соответствующую последовательность слов.

Грамматику синтаксиса конкатенативных языков можно описать при помощи синтаксической диаграммы на рисунке 5. На рисунке «термом» обозначаются те самые слова, комбинаторы, функции (конкретный термин зависит от контекста конкретного языка).

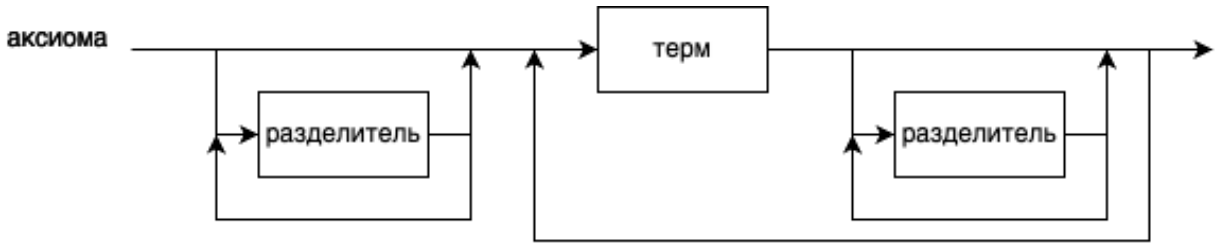

Рисунок 5 — Синтаксическая диаграмма аксиомы грамматики конкатенативных языков

В языке Forth особоую роль играет интерактивная модель «текстового интерпретатора»: входная строка разбивается на токены, которые ищутся в словаре и либо исполняются немедленно, либо компилируются в новое определение [1]. Базовая конструкция задания определения представляет собой так называемое «colon definition» вида «: name … ;»: слово «:» считывает имя и переводит систему в режим компиляции, а «;» завершает определение и возвращает режим интерпретации [5].

Языки семейства Joy сохраняют постфиксную запись и токенизацию по пробельным разделителям, но дополняют её явными литералами структур данных и механизмом «цитирования» программ. В Joy квадратные скобки «[…]» используются для записи списков и одновременно для представления «цитаты» как значения, которое может передаваться через стек и затем исполняться посредством комбинаторов [2]. Определения новых слов задаются в форме равенств «name == …». В языке распространены комментарии на основе stack-effect нотации.

Factor использует аналогичный Joy синтаксис. Важной особенностью является применение «stack effect notation»: сигнатуры в скобках с разделителем «–», включаемой непосредственно в текст определения слова как его формализованного интерфейса [3], которая затем используется компилятором для статического анализа.

Для языка Cat выделяется предельно компактное синтаксическое ядро, формируемое двумя операциями: цитированием и композицией. Это ядро может быть выражено абстрактной грамматикой из листинга 5 [11]. В прикладных описаниях Cat данная основа дополняется также набором правил, задающих определение слова (терма). В результате синтаксис Cat близок к языкам Joy-семейства по внешнему виду.

Листинг 5 — Аксиома грамматики и правило языка Cat (правила с терминалами не показаны)

```
axiom ::= term
term ::= [term] | term term | empty
```

Синтакт Wasm принциально отличается. Спецификация определяет текстовый формат, основанный на S-выражениях, в котором модуль задаётся как иерархическая скобочная структура [12]. Вместе с тем тело функции сохраняет линейный характер и соответствует стековой виртуальной машине: инструкции потребляют и порождают значения на неявном стеке [4]. Листинг 6 приводит правила для построения грамматики.

Таким образом, синтаксис рассмотренных языков образует спектр от «операционально задаваемого» (Forth, где значимую роль играют

слова-парсеры и режимы интерпретации/компиляции) до «формально фиксированного» (WebAssembly, где нормативно определена грамматика текстового формата). При этом сохраняется общий принцип: последовательность токенов интерпретируется как композиция стековых преобразований, а скобочные конструкции применяются локально – преимущественно для представления цитат или для древовидного описания модулей [2–4,11].

Листинг 6 — Акиома и правила грамматики Wasm (правила с терминалами не показаны)

```
axiom ::= expr
expr ::= (expr-type opers)
opers ::= term | expr
```

4 Анализ семантики

Семантика стековых языков удобно описывается через состояние вычислений. В простейшем случае это «операндный стек», а также (в зависимости от языка) память, словарь определений, ввод-вывод и другие компоненты среды. Выполнение программы сводится к последовательному применению слов (инструкций), каждое из которых берёт данные со стека и кладёт результаты обратно. Для языков семейства Joy/Cat/Factor распространена интерпретация «программа есть функция вида stack -> stack», а конкатенация фрагментов программы соответствует последовательному применению этих преобразований [6,7].

4.1 Типизация

Forth не задаёт строгой типовой дисциплины: элементы стека являются машинными словами, а их смысл (число, адрес, флаг) определяется соглашениями программиста. Стандарт описывает набор слов и модель стека, но не вводит обязательной проверки типов [5]. На практике это повышает гибкость и предсказуемость исполнения, но осложняет контроль корректности.

Joy относится к динамически типизированным языкам: тип значения проверяется в момент применения операции. Если операция ожидает, например, число, а на стеке находится список, возникает ошибка времени выполнения [8]. Для снижения числа таких ошибок часто используют «stack effect» как комментарий к определениям, фиксируя, что слово ожидает и что возвращает.

Factor также динамически типизирован, однако дополняет это обязательной проверкой «формы стека» при компиляции. Для слов обычно задаётся эффект на стек, и компилятор проверяет, что тело определения действительно согласовано по высоте и структуре стека во всех ветвлениях. Таким образом, часть типовых ошибок устраняется ещё до запуска программы [3].

Cat вводит статическую типизацию поверх стековой модели. Тип задаётся как преобразование конфигурации стека в другую конфигурацию, включая «хвост» стека через типовые переменные (row-полиморфизм). Это

позволяет типизировать композиции слов и проверять корректность ветвлений и циклов на уровне типов [10,11].

WebAssembly изначально строится как формат с нормативной статической типизацией и валидацией: у функций есть типы параметров и результатов, а инструкции имеют определённые эффекты над стеком. Модуль перед исполнением проходит валидацию, которая гарантирует согласованность типов и отсутствие ошибок вида «недостаточно операндов на стеке» [4].

4.2 Статический и динамический анализ

Динамический контроль в стековых языках проявляется как проверки во время выполнения: операции снимают значения со стека и либо корректно работают, либо завершаются ошибкой (Joy), либо могут привести к некорректной интерпретации данных при отсутствии проверок (типичный сценарий для Forth) [5,8].

Статический анализ использует информацию о том, как слова изменяют стек, и пытается проверить программу до исполнения. В Factor это реализовано как «проверка стека»: анализатор проходит по коду, моделирует изменения абстрактного стека и требует, чтобы в точках слияния потока управления состояние стека было совместимо. Несовместимость превращается в ошибку компиляции [3].

В Cat статический анализ встроен в систему вывода типов. Поскольку тип слова фиксирует входную и выходную конфигурации стека, система может гарантировать корректность композиции и ограничения для управляющих комбинаторов: у «if» обе ветви должны возвращать одинаковую структуру стека, у «while» конфигурация стека сохраняется как инвариант цикла [11].

В WebAssembly статический анализ выражен в процедуре валидации: она проверяет типы и структуру управления (block/loop/if) и тем самым обеспечивает безопасную основу для дальнейшей компиляции и оптимизаций в рантайме [4].

4.3 Оптимизации

Оптимизации в стековых системах обычно направлены на две основные проблемы: накладные расходы диспетчеризации (переходы между

инструкциями интерпретатора) и накладные расходы доступа к стеку (загрузки/сохранения значений в памяти).

Для Forth характерны компактные представления программы и быстрый вызов «слов». Широко используется нитевой код, где программа хранится как последовательность ссылок на подпрограммы, что ускоряет интерпретацию и снижает размер представления [1]. Кроме того, управляющие конструкции часто разворачиваются на этапе компиляции в более примитивные переходы, уменьшая стоимость интерпретации.

Для виртуальных стековых машин существенный вклад в оптимизацию внёс подход «кэширования стека» (stack caching). Его идея проста: держать верхние элементы операндного стека не в памяти, а в регистрах реальной машины, чтобы сократить число загрузок и выгрузок. В работах М. А. Эрткля систематизированы модели «одно- и многосостояльного» стек-кэша и рассмотрены как динамический, так и статический варианты отслеживания состояния кэша . В статье «Stack Caching for Interpreters» показано, что интерпретатор может тратить значительную долю времени именно на обмен между стеком в памяти и регистрами, и предложены две техники: динамическое кэширование (несколько специализированных версий интерпретатора для разных состояний кэша) и статическое кэширование (компилятор отслеживает состояние кэша как конечный автомат и выбирает нужные варианты примитивов) .

Отдельное направление оптимизаций — «суперинструкции», когда часто встречающиеся последовательности байткод-инструкций объединяются в одну более крупную операцию, уменьшая число шагов диспетчеризации. Работа «Combining Stack Caching with Dynamic Superinstructions» показывает, что совмещение суперинструкций с кэшированием стека даёт больший выигрыш, чем каждое из решений по отдельности: суперинструкции уменьшают стоимость диспетчеризации, а stack caching — стоимость стек-доступов .

Factor использует собственный оптимизирующий компилятор и промежуточные представления. Наличие stack-checker даёт надёжную информацию о структуре стека, что помогает оптимизатору безопасно переписывать код,

устранять лишние операции со стеком и затем генерировать эффективный машинный код [3].

Cat интересен тем, что статическая типизация «эффектов на стек» делает возможными агрессивные преобразования при сохранении корректности: типы фактически выступают контрактом, который должен сохраниться после оптимизаций. Это полезно как для верификации, так и для применения языка в роли промежуточного представления [10].

В WebAssembly оптимизации в значительной степени выполняются в рантаймах (JIT/AOT). При этом стековая форма байткода удобна для компактности и быстрой валидации, а во время компиляции обычно преобразуется в более подходящее для машинного кода представление (например, регистровое) [4].

4.4 Управление памятью

В Forth работа с памятью является частью базовой модели: стандартные слова позволяют читать и записывать по адресу, а выделение и освобождение памяти реализуются либо явно, либо через библиотечные соглашения и слова конкретной системы [5].

В Joy активно используются динамические структуры данных (списки, строки, цитаты), поэтому реализация обычно опирается на сборку мусора, снимая с программиста необходимость вручную освобождать временные объекты [8].

В Factor управление памятью встроено в виртуальную машину и реализовано сборщиком мусора с разделением по поколениям; для неуправляемых ресурсов предусмотрены специальные библиотечные средства, позволяющие корректно освобождать внешние ресурсы [3].

В Cat модель памяти зависит от целевого окружения. В вариантах, ориентированных на .NET, управление памятью, как правило, делегируется среде выполнения, а ключевой вклад Cat связан именно с проверкой корректности стековых преобразований [10].

В WebAssembly память задана как линейный массив байт, доступный через инструкции загрузки и сохранения. В базовой спецификации управление

временем жизни объектов не стандартизовано: оно определяется исходным языком и его рантаймом либо предоставляется окружением через импорты [4].

4.5 Стандартная библиотека

Forth опирается на стандартные наборы слов («core» и опциональные расширения), а дальнейшее расширение обычно идёт через подключаемые словари и создание предметно-ориентированных надстроек [5].

Joy поставляется с набором библиотек по областям (операции над агрегатами и последовательностями, численные функции и т.п.), однако экосистема остаётся сравнительно компактной, что соответствует учебно-исследовательскому характеру языка [8].

Factor включает обширную стандартную библиотеку и развитые средства разработки (интерактивная среда, отладка, инспекция объектов и т.д.), что делает его пригодным для прикладного использования как языка общего назначения [3,9].

Cat ориентирован на минимальное ядро и исследовательские задачи; основной упор делается на примитивы, комбинаторы и систему типов, поэтому стандартная библиотека по объёму ограничена [11].

WebAssembly не имеет «стандартной библиотеки» в традиционном смысле: модуль взаимодействует с внешним миром через импорты, а набор доступных функций определяется окружением исполнения (например, браузером или WASI) [4].

4.6 Выводы

Сравнение семантических аспектов показывает общий тренд: по мере развития стековых языков возрастает доля информации, доступной до исполнения (эффекты на стек, правила для ветвлений и циклов, валидация модулей). Это повышает надёжность и упрощает автоматическую проверку корректности. Одновременно исследования по реализации стековых машин на регистровых архитектурах демонстрируют, что стековая модель может быть эффективной: кэширование стека и суперинструкции позволяют существенно сократить накладные расходы интерпретации без изменения внешней семантики программ .

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Forth documentation [Электронный ресурс]. URL: https://www.forth.com/resources/forth-programming-language/ (дата обращения: 10.10.2025).

2. An informal tutorial on Joy [Электронный ресурс]. URL: https://hypercubed.github.io/joy/html/j01tut.html (дата обращения: 10.10.2025).

3. Factor: a dynamic stack-based programming language [Электронный ресурс]. URL: https://factorcode.org/slava/dls.pdf (дата обращения: 20.11.2025).

4. WebAssembly Specification 1.0 [Электронный ресурс]. URL: https://webassembly.github.io/spec/versions/core/WebAssembly-1.0.pdf (дата обращения: 10.10.2025).

5. ANS Forth standard [Электронный ресурс]. URL: https://forth-standard.org/standard/words (дата обращения: 10.10.2025).

6. Mathematical foundations of Joy [Электронный ресурс]. URL: https://hypercubed.github.io/joy/html/j02maf.html (дата обращения: 10.10.2025).

7. The Algebra of Joy [Электронный ресурс]. URL: https://hypercubed.github.io/joy/html/j04alg.html (дата обращения: 10.10.2025).

8. The prototype implementation of Joy [Электронный ресурс]. URL: https://hypercubed.github.io/joy/html/j09imp.html (дата обращения: 10.10.2025).

9. Factor wiki [Электронный ресурс]. URL: https://concatenative.org/wiki/view/Factor (дата обращения: 20.11.2025).

10. Typing Functional Stack-Based Languages [Электронный ресурс]. URL: https://dcreager.net/remarkable/Diggins2008b.pdf (дата обращения: 10.10.2025).

11. Simple Type Inference for Higher-Order Stack-Oriented Languages [Электронный ресурс]. URL: https://dcreager.net/remarkable/Diggins2008a.pdf (дата обращения: 10.10.2025).

12. Understanding WebAssembly text format [Электронный ресурс]. URL: https://developer.mozilla.org/en-US/docs/WebAssembly/Guides/Understanding_the_text_format (дата обращения: 10.10.2025).

13. A foundation for typed concatenative languages [Электронный ресурс]. URL: https://www2.ccs.neu.edu/racket/pubs/dissertation-kleffner.pdf (дата обращения: 10.10.2025).

14. Cat language repository [Электронный ресурс]. URL: https://github.com/cdiggins/cat-language (дата обращения: 10.10.2025).