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 23 с., 0 рис., 0 табл., 0 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

Java Bytecode

.NET CIL

алгоритма Хиндли–Милнера

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, 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).

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. 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. 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. 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 (зачастую 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. 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 наиболее значимых языков, каждый из которых привнес важные нововведения в группу языков.

2.1 ЯП Forth

Forth представляет собой стековый конкатенативный язык программирования, созданный 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.

С точки зрения семантики типов Forth рассматривается как «безтиповый»: 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.

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 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.

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.

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.

В отличие от обычных функциональных языков, которые строятся вокруг операции применения функции к аргументу, Joy использует операцию композиции функций. 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. 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.

Синтаксис Joy минималистичен и основан на постфиксной записи. 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. 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.

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. 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. 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 (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). 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 богатую 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.

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 Cat

Логическим продолжением ЯП Joy является язык Cat, который с 2006 года разрабатывается Кристофером Диггинсом. Автор характеризует свой язык как «Joy с типами». Язык используется для верификации и оптимизации программ под платформу .NET CIL. 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 программы и верифицировать программу на соответствие ожидаемым типам на стеке и фактическим. 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.

Аналогично Joy, программы на языке Cat опираются на обратную польскую запись с последовательным перечислением слов программы слева направо. 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, 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 важен), 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. Статья о типизации функциональных стековых языков 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. 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 даже для программ, где используются функции высшего порядка), а аннотации

типов (так называемые «stack diagrams») используются как документация и как данные для статического анализа.

В статье описывается чистое подмножество языка Cat с правилами вывода, показанными в листинге 2. При выводе типов алгоритм проходится по словам и находит тип каждой композиции из слов. Затем проводится унификация типов согласно алгоритму Хинди-Милнера.

Листинг 2 — Правила вывода некоторых 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. 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 момнет 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; эта информация также фиксируется в типах и может использоваться оптимизатором или проверяющими инструментами. Формальная работа по типизации функциональных стековых языков уточняет эту модель, рассматривая подмножество Cat без побочных эффектов, с целыми, булевыми и функциональными типами, и задаёт малошаговую операционную семантику, на фоне которой доказываются свойства корректной типизации и безопасности исполнения.

Управление памятью в Cat опирается на одну из особенностей семантики языка: для каждой функции доступно только то состояние стека, которое существовало на момент вызова этой функции. То есть функция не может ссылаться на предыдущие состояния стека, а значит и использовать данные оттуда. Таким образом можно четко определить время жизни каждого фрагмента данных исходя из знания «в каких конфигурациях стека этот фрагмент данных был, а в каких уже был недоступен».

Благодаря статическому анализу Cat активно работает с оптимизациями и переписыванием программ. На этапе компиляции возможно провести оптимизации константной свертки (вычисление значений, которые зависят только от статического контекста), устранение избыточных стековых операции, композицию цитат (то есть объединение нескольких цитат с целью уменьшения количество операций) и другие.

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: статьи и доклады подчеркивают его пригодность для формальной верификации, оптимизации и анализа свойств программ на традиционных языках, транслируемых в Cat, особенно в контексте .NET и CIL. Хотя язык не получает широкого промышленного распространения, он влияет на дальнейшие разработки в области конкатенативных языков и типовых систем: упоминания Cat и его системы типов встречаются в описаниях последующих языков (например, Statick и Kitten). В итоге Cat формирует пример полноценно типизированного стекового языка, в котором идея «программа как функция stack → stack, конкатенация как композиция» соединяется с мощной статической системой типов, ориентированной на анализ и оптимизацию.

2.4 ЯП Factor

2.5 ЯП Wasm

2.6 Выводы

3 Анализ синтаксиса

4 Анализ семантики

4.1 Типизация

4.2 Статический и динамический анализ

4.3 Оптимизации

4.4 Управление памятью

4.5 Стандартная библиотека

4.6 Выводы

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ