



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

(Группа)

\_\_\_\_\_  
(Подпись, дата)

В.К. Залыгин

(И.О. Фамилия)

Руководитель

\_\_\_\_\_  
(Подпись, дата)

Б.И. Бычков

(И.О. Фамилия)

Оценка \_\_\_\_\_

2025 г.

\_\_\_\_\_

по теме **Анализ синтаксиса и семантики стековых языков программирования**

Залыгин Вячеслав Константинович  
(Фамилия, имя, отчество)

Источник тематики (кафедра, предприятие, НИР) \_\_\_\_\_ кафедра \_\_\_\_\_

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

- 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.

## Руководитель

---

 (Подпись, дата)

---

 В.К. Залыгин  
(И.О. Фамилия)

**Студент**

---

 (Подпись, дата)

---

 Б.И. Бычков  
(И.О. Фамилия)

## **РЕФЕРАТ**

РПЗ 27 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

ВВЕДЕНИЕ .....	6
1 Определение модели исполнения .....	8
2 Аналитический обзор существующих решений .....	10
2.1 ЯП Forth .....	10
2.2 ЯП Cat .....	11
2.3 ЯП Factor .....	13
2.4 ЯП Wasm .....	14
2.5 Выводы .....	15
3 Анализ синтаксиса .....	17
4 Анализ семантики .....	20
4.1 Типизация .....	20
4.2 Статический и динамический анализ .....	21
4.3 Оптимизации .....	21
4.4 Управление памятью .....	23
4.5 Стандартная библиотека .....	24
4.6 Выводы .....	24
ЗАКЛЮЧЕНИЕ .....	25
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	26

## **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-ориентированность 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 относительно 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 системы), 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 линейен: 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.

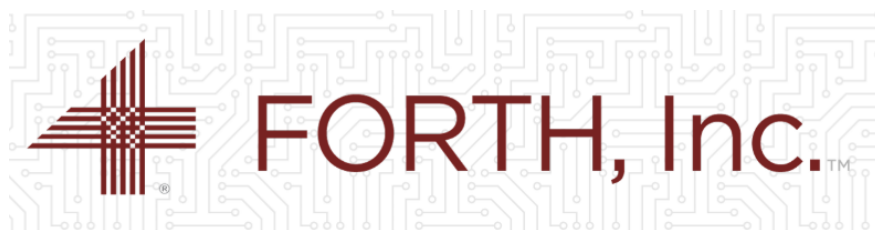


Рисунок 1 — Логотип Forth

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 )
```

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].

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 [5]. Каждая программа в Joy обозначает унарную функцию вида «stack  $\rightarrow$  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  $\rightarrow$  stack» – семантический моноид по операции композиции и тождественной функции в роли нейтрального элемента [5]. 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 [6].

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

## **2.2 ЯП Cat**

Еще одним логическим продолжением Joy является экспериментальный язык Cat (логотип языка показан на рисунке 2), который с 2006 года разрабатывается Кристофером Диггинсом. Язык используется для верификации и оптимизации программ под платформу .NET CLR [7]. 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 [7]. Таким образом при помощи статического анализа возможно предотвратить фатальные ошибки времени выполнения, связанные с несовпадением типов, опустошением или переполнением стека. Также статический анализ открывает дороги агрессивным оптимизациям, которые могут менять тексты программ и промежуточных представлений для повышения производительности.



Рисунок 2 — Логотип Cat

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  $stack \rightarrow stack$ , конкатенация как композиция» соединяется с сильной статической системой типов, ориентированной на анализ и оптимизацию.

### 2.3 ЯП Factor

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

- конкатенативный синтаксис;
- использование комбинаторов для управления потока исполнения;
- аналогичный способ создания собственных комбинаторов при помощи цитат.



Рисунок 3 — Логотип Factor

Прикладным язык считается за счет большой стандартной библиотеки, которая позволяет решать широкий спектр прикладных задач. Благодаря статическому анализу (так называемый `stack-checker`), а также динамическим проверкам времени исполнения компилятор языка обеспечивает верификацию программ, а также высокоуровневые и низкоуровневые оптимизации, что также делает язык более применимым для большого количества продуктовых задач. Пример определения нового слова вычисления факториала

Листинг 2 — Определение слово вычисления факториала

```
: factorial ( n -- n! )  
dup 0 =
```

## Продолжение листинга 2

```
[ drop 1 ]  
[ dup 1 - factorial * ]  
if ;
```

### 2.4 ЯП Wasm

В 2017 году группа компаний, в которую входит W3C, Mozilla, Google, Microsoft, Apple, представила язык WebAssembly (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 является 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 [8] (еще одна черта, которая отличает этот язык от представленных ранее). Листинг 3 показывает определение функции сложения числа с самим собой на wasm в человеко-читаемом варианте.

### Листинг 3 — Определение функции сложения числа с самим собой

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

Язык и виртуальная 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).

## 2.5 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 свойствами стековых языков считаются 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 быстрее 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 и структурирует информацию об вышеописанных языках.

Таблица 1 — Сводная характеристика рассмотренных стековых языков

Язык	Год выпуска первой версии	Область применения	Вклад в развитие
Forth	1970	Встроенные системы	Первый стековый язык, задал прикладную область
Joy	2001	Экспериментальный язык	
Factor	2003	Язык общего назначения	
Cat	2006	Экспериментальный язык, верификация программ .NET CIL	
Wasm	2017	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 функций 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) [9]. Исключение составляют некоторые конструкции времени компиляции, которые затем преобразуются в соответствующую последовательность слов.

Граматику 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).

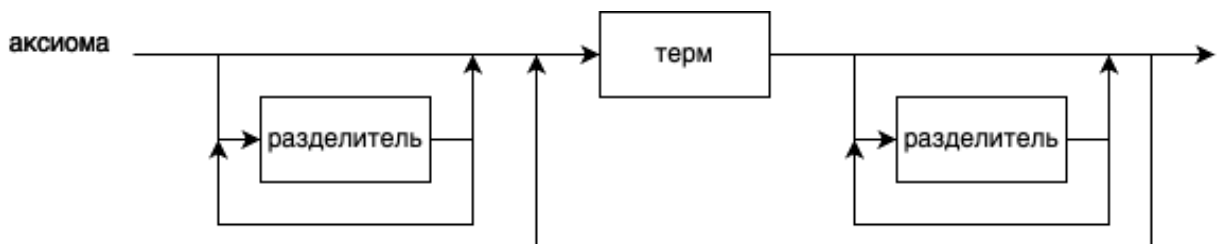


Рисунок 5 — Синтаксическая диаграмма аксиомы грамматики конкатенативных языков

В языке Forth особую 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 [10].

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 4 [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 4 — Аксиома грамматики и правило языка Cat (правила с терминалами не показаны)

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

Синтакс Wasm принципиально отличается. Спецификация определяет текстовый формат, основанный на S-выражениях, в котором модуль задаётся как иерархическая скобочная структура [8]. Вместе с тем тело функции сохраняет линейный характер и соответствует стековой виртуальной машине: инструкции потребляют и порождают значения на неявном стеке [4]. Листинг 5 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 5 — Аксиома и правила грамматики Wasm (правила с терминалами не показаны)

```
axiom ::= expr
expr  ::= (expr-type ops)
ops   ::= 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 [5,6].

### 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 [10]. На практике это повышает гибкость и предсказуемость исполнения, но усложняет 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 [12]. 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 [7,11].

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) [10,12].

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 [7].

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 [10].

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 [12].

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 [7].

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 [10].

Joу 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 [12].

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 [3,13].

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. Mathematical foundations of Joy [Электронный ресурс]. URL: <https://hypercubed.github.io/joy/html/j02maf.html> (дата обращения: 10.10.2025).
6. The Algebra of Joy [Электронный ресурс]. URL: <https://hypercubed.github.io/joy/html/j04alg.html> (дата обращения: 10.10.2025).
7. Typing Functional Stack-Based Languages [Электронный ресурс]. URL: <https://dcreager.net/remarkable/Diggins2008b.pdf> (дата обращения: 10.10.2025).
8. Understanding WebAssembly text format [Электронный ресурс]. URL: [https://developer.mozilla.org/en-US/docs/WebAssembly/Guides/Understanding\\_the\\_text\\_format](https://developer.mozilla.org/en-US/docs/WebAssembly/Guides/Understanding_the_text_format) (дата обращения: 10.10.2025).
9. A foundation for typed concatenative languages [Электронный ресурс]. URL: <https://www2.ccs.neu.edu/racket/pubs/dissertation-kleffner.pdf> (дата обращения: 10.10.2025).
10. ANS Forth standard [Электронный ресурс]. URL: <https://forth-standard.org/standard/words> (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. The prototype implementation of Joy [Электронный ресурс]. URL: <https://hypercubed.github.io/joy/html/j09imp.html> (дата обращения: 10.10.2025).
13. Factor wiki [Электронный ресурс]. URL: <https://concatenative.org/wiki/view/Factor> (дата обращения: 20.11.2025).

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