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

Отчет 25 с., 4 рис., 0 табл., 12 источн., 0 прил.

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

Объектом анализа являются стековые языки программирования.

Цель работы – проанализировать и сравнить

разделы в ведении дать постановку задачи (цель работы является … аналитический обзор, выделение и сравнение свойств различных стековых языков, в конец или между ним и про модель) – ЭТО ТЗ Выполнить ана обзор Выполнить сравнительный Анализ Сделать ввывдо о тенденции обзор результаты сравнительного анализа стековых языков комментировать таблицы линия развития посмотреть в будущее

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-ориентированность позволяет программам на таких языках компактно выглядеть 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.

Цель работы – выполнить аналитический обзор различных стековых языков программирования, выделить и сравнить свойста представленных языков, сделать выводы о существующих тенденциях в области стековых языков.

Модель исполнения стекового языка – это некая (чаще всего виртуальная) стековая машина, имеющая стек данных, операционный стек 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 команд, которые могут быть либо встроенными примитивами (например, арифметические операции, операции над стеком; интринсиками), либо определенными пользователем субпрограммами. В терминах стековых языков команды принято называть словами, или комбинаторами, или функиями. 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.

1 Аналитический обзор существующих языков

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.

1.1 Язык программирования Forth

Forth является одним 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.

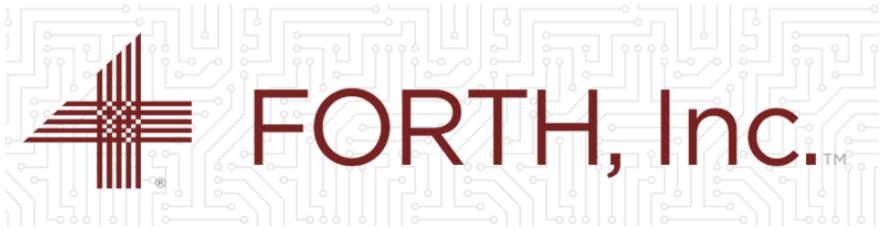

Рисунок 1 — Логотип Forth

Синтаксис Forth основан на обратной польской нотации: программа представляет собой последовательность слов, разделённых пробелами. Интерпретатор последовательно читает токены, ищет их в словаре и либо выполняет связанный код, либо интерпретирует токен как число и помещает его на стек. Определения новых слов задаются конструкцией вида «: NAME … ;», при этом большинство лексем — от переменных до управляющих конструкций — оформляются как слова.

Стандартной библиотеки в привычном понимании в Forth нет: расширение возможностей осуществляется за счёт подключения и использования дополнительных наборов слов (word sets), входящих в конкретную реализацию среды.

С точки зрения семантики типов Forth рассматривается как «безтиповый» [2]: 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 является интерактивной средой, в которой 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. Определения новых слов задаются конструкцией вида «: NAME … ;», причём большинство лексем – от переменных до управляющих конструкций – являются словами. Стандартный «core word set» ANS Forth [2] 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, то есть конструкций, которые во время компиляции разворачиваются в более примитивные последовательности слов. 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].

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 )
```

Forth применяется во встраиваемых и ресурсно-ограниченных системах, прошивках, системах реального времени и космической технике. В частности, он используется в контроллерах космических аппаратов и приборов NASA, а также в реализациях Open Firmware для платформ Apple, IBM и Sun. К характерным особенностям практического использования относятся малый размер полной среды (компилятор, интерпретатор и редактор могут размещаться в памяти 8-битных систем), а также единство языка и среды исполнения: Forth одновременно выступает командной оболочкой, компилятором и минимальной операционной системой, при этом стандарт описывает набор слов и модель стека, оставляя свободу построения предметно-ориентированных расширений и диалектов [1].

1.2 Язык программирования Joy

В 2001 году Манфред фон Тун из университета Ла Троба представил язык Joy как попытку формализации логики стековых языков. Joy представляет собой функциональный стековый конкатенативный язык программирования [3]. В отличие от традиционных функциональных языков, которые строятся вокруг операции применения функции к аргументу, Joy использует операцию композиции функций [4]. Автор вводит формальную основу: каждая команда в Joy обозначает унарную функцию вида «stack -> stack»: на вход подаётся состояние (стек данных), в процессе применения команды происходит его модификация и передача следующей команде; значения и подпрограммы при этом также передаются через стек.

Язык с точки зрения математики 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 [4]. 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 [5].

Как в случае с Forth, синтаксис Joy основан на постфиксной записи [3]. Программа представляет собой последовательность слов, разделённых пробелами, исполнение происходит слева направо. Каждое слово выполняет некоторое преобразование над стеком данных. Операторы снимают одно или несколько значений с вершины стека и помещают результат обратно. Язык определяет способ записи отложенных вычислений при помощи механизма цитат, которая содержащит фрагмент программы как данные [3]. Для представления структур данных используются литералы: цитаты программ записываются в квадратных скобках «[…]», множества — в фигурных скобках, строки — в кавычках. Определения новых слов записываются в виде равенств, например «square == dup * .». Язык при этом остаётся функционально чистым: стандартные операции не изменяют скрытое глобальное состояние, а только преобразуют стек, что позволяет рассматривать программу как математическую функцию «stack -> stack» без побочных эффектов [4].

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 [3]. 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.

Благодаря возможности определять новые слова, которые могут выступать как обычные функции или как комбинаторы, Joy располагает достаточно разветвлённой стандартной библиотекой. В стандартной поставке присутствует несколько библиотек, сгруппированных по областям применения. Так, библиотеки «agglib» и «seqlib» содержат обобщённые операции над агрегатами (неупорядоченными множествами, строками и списками) и последовательностями, «numlib» включает числовые функции и численные методы, а «mtrlib» предназначена для работы с матрицами [6].

Joy является динамически типизированным языком. В язык встроены числовые типы (целые и вещественные числа), агрегатные типы (строки, списки и неупорядоченные множества), а также тип цитат [3]. Типизация стека на этапе компиляции не контролируется, и проверка типов выполняется в момент применения операции. При несовпадении ожидаемого и фактического типов возникает ошибка времени выполнения [6]. Для уменьшения числа подобных ошибок рекомендуется сопровождать определения функций комментариями в нотации эффектов стека, где указывается, какой стек функция принимает и какой возвращает, например «dup : A -> A A» или «+ : Int Int -> Int». Такая нотация позволяет рассматривать программы как формулы в рамках алгебры стеков и служит основой для рассуждений о корректности и эквивалентности программ [5], что можно рассматривать как предпосылку к статическому анализу.

Работа с памятью в Joy в значительной степени скрыта от программиста. Базовая реализация языка представляет собой интерпретатор на языке C с автоматической сборкой мусора, что позволяет безопасно работать с динамическими структурами данных, такими как списки, строки и цитаты, без явного управления временем их жизни [6].

Язык Joy имеет преимущественно академическое и экспериментальное применение, так как используется для демонстрации и исследования идей функционального программирования в стековой модели исполнения.

В результате Joy можно рассматривать как компактную и хорошо формализованную модель стекового языка высокого уровня, в которой математические понятия моноида и гомоморфизма служат точным описанием связи между текстом программы и её поведением при выполнении.

1.3 Язык программирования Cat

Логическим продолжением Joy является экспериментальный язык Cat (логотип языка показан на рисунке 2), по которому в 2006 году Кристофером Диггинсом опубликовано несколько статей [7,8]. Cat 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, что позволяет компилятору на этапе компиляции вычислить размер стека данных и его типизацию для любого момента времени исполнения программы и верифицировать программу на соответствие ожидаемым и фактическим типам [8]. Таким образом, за счёт статического анализа возможно предотвратить фатальные ошибки времени выполнения, связанные с несовпадением типов, опустошением или переполнением стека; кроме того, статический анализ открывает путь к агрессивным оптимизациям, затрагивающим текст программы и промежуточные представления.

Рисунок 2 — Логотип Cat

Аналогично Joy, программы на языке Cat опираются на обратную польскую запись с последовательным перечислением слов слева направо [8]. Работа с потоком исполнения организована при помощи комбинаторов «if» и «while», соответствующих ветвлению и циклу по условию: комбинаторы принимают набор цитат (которые, аналогично Joy, записываются в квадратных

скобках; также цитаты называют замыканиями) и организуют поток управления. Определение новых слов возможно при помощи слова «define», определения являются глобальными и единственными (множественные определения запрещены).

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

Наибольшего внимания заслуживает система типов Cat, благодаря которой строится статический анализ. Система типов 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 оставляет на стеке после выполнения. Тип записывается как круглые скобки с стрелкой внутри между конфигурациями стека [7], например «(int int -> int)» для операции сложения или «(’S ’a -> ’S ’a ’a)« для дублирования вершины стека. Типы с апострофом (’a, ’b) обозначают типовые переменные, 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 разделяет хвост (типовой вектор) и голову (типовые переменные) стека 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 и полиморфной системы родов, позволяющей описывать как отдельные значения, так и целые стековые конфигурации [7]. Вывод типов в Cat

базируется на вариации алгоритма Хиндли–Милнера, обобщённой на стековые типы и row-полиморфизм. В статье [8] описывается пример вывода типов композиции нескольких комбинаторов (которая сама, конечно, также является комбинатором). Конструкции языка допускают полиморфизм более высоких рангов (в контексте Cat это означает, что позволяется выводить типы для функций, которые манипулируют другими полиморфными функциями сколь угодной степени вложенности), а аннотации типов («stack diagrams») используются как документация и как данные для статического анализа [7]. В статье [8] описывается чистое подмножество языка Cat с правилами вывода, показанными в листинге 2: алгоритм проходит по словам, находит тип каждой композиции, после чего выполняется унификация типов согласно алгоритму Хиндли–Милнера.

Листинг 2 — 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 тип функции фиксирует не только типы элементов, но и форму стека до и после применения функции, система гарантирует отсутствие недопустимых ситуаций, таких как недостаток аргументов на стеке или несовместимость конфигураций стека в ветвлениях [7]. Для 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. Для цикла «while» тип требует, чтобы после выполнения цикла конфигурация стека оставалась такой же, как и на момент начала цикла (как это и показано в листинге 2). 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 фиксируется в типах и может использоваться оптимизатором или проверяющими инструментами [7].

Статический анализ открывает путь к агрессивным оптимизациям, которые могут изменять текст программы и промежуточные представления для повышения производительности. Поскольку типы фиксируют форму и размер стека, оптимизатор может опираться на результаты вывода типов как на формализованный контракт корректности [7,8]: форма и размер стека должны остаться прежними после оптимизаций с точки зрения глобального наблюдателя.

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 опираются на Cat и рассматривают его как пример статически типизированного конкатенативного языка. Во-вторых, Cat разрабатывается как промежуточный язык для компиляторов и инструментов анализа, также используется для верификации и оптимизации программ под платформу .NET CIL [8]. Язык повлиял 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, что дает хорошие возможности для развития новых прикладных решений.

1.4 Язык программирования Factor

Среди стековых языков Factor занимает нишу языков общего назначения [9]. 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 потоком исполнения, а также аналогичный способ создания собственных комбинаторов при помощи цитат.

Рисунок 3 — Логотип Factor

Синтаксис Factor является конкатенативным: программа записывается как последовательность слов, применяемых слева направо, а для управления потоком исполнения используются комбинаторы и цитаты. Пример определения нового слова приведён в листинге 3; в нём показано использование нотации «stack effects», а также комбинатора «if» и цитат.

Листинг 3 — Определение слова вычисления факториала

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

Прикладным язык считается за счёт большой стандартной библиотеки, которая позволяет решать широкий спектр прикладных задач. В отличие от Joy, стандартная библиотека Factor обладает существенно большими размерами и количеством возможностей. Также для языка существует большое количество различных инструментов разработки, объединённых в IDE: интерактивный отладчик, браузер документации, инспектор объектов, механизм модификации программы без её перезапуска [10].

Как и Joy, Factor является динамически типизированным языком [9]. Документация и сообщество активно используют нотацию «stack effects», с помощью которой описывается, как то или иное слово при применении модифицирует стек. 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: stack-checker симулирует эффекты слов на абстрактном стеке, при встрече

ветвлений анализирует обе ветви и унифицирует состояния, а несовместимость высоты или формы стека превращается в ошибку времени компиляции. Такая «проверка стека» играет роль статического анализатора, который обнаруживает классы ошибок, типичные для стековых языков, и одновременно создаёт основу для агрессивных оптимизаций [9]. Прикладным язык считается также за счёт статического анализа (stack-checker) и динамических проверок времени исполнения, которые обеспечивают верификацию программ и повышают применимость языка для продуктовых задач.

В целом язык предлагает множество высокоуровневых и низкоуровневых оптимизаций. Оптимизирующий компилятор состоит из фронтенда, который строит промежуточное представление языка, используемое для работы stack-checker и высокоуровневых оптимизаций, а также бекенда, который генерирует исполняемый машинный код и выполняет низкоуровневые оптимизации [9].

Для работы с памятью язык использует сборщик мусора, встроенный в виртуальную стековую машину. Для молодых объектов применяется копирующий сборщик, для старшего поколения используется алгоритм «mark-sweep-compact» [9]. Для работы с неуправляемыми ресурсами используются библиотека «destructors» и комбинатор «with-destructors». Использование умного сборщика мусора также повышает степень пригодности языка к решению задач широкого спектра.

Язык Factor относится к языкам общего назначения и ориентирован на решение прикладных задач за счёт обширной стандартной библиотеки, развитого инструментария и оптимизирующей компиляции [9,10].

1.5 Язык программирования Wasm

В 2017 году группа компаний, в которую входят W3C, Mozilla, Google, Microsoft и Apple, представила язык WebAssembly (Wasm). Логотип Wasm показан на рисунке 4. Wasm представляет собой низкоуровневый переносимый байткод и архитектуру набора команд под виртуальную стековую машину, запускаемую преимущественно (но не только) в веб-браузерах наравне с движками ECMAScript для исполнения клиентского кода [11]. 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 C/C++/Rust, которые имеют небольшой (или вовсе не имеют) рантайм. Особенность области использования wasm – веб-браузер не может считать никакой получаеммый код доверенным, в результате чего к языку и машине Wasm предъявляются парадоксальные требования: виртуальная машина wasm должна дополнительно верифицировать код, порожденный тем или иным компилятором (и в общем случае считаемым компилятором корректным), для других описанный в обзоре языков такая проверка была уместна, так как тексты программ пишутся человеком, которому свойственно ошибаться. Таким образом 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 [11].

Рисунок 4 — Логотип Wasm

Байткод Wasm в основном поставляется в виде бинарного представления, которое при необходимости может быть транслировано в текстовое человеко-читаемое представление. Текстовый вариант основан на записи S-выражений [12], что отличает этот язык от рассмотренных ранее. Листинг 4 демонстрирует определение функции сложения числа с самим собой в человеко-читаемом виде.

Листинг 4 — Определение функции сложения числа с самим собой

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

Продолжение листинга 4

```
  i32.add
)
```

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.

Как стековый язык исполнения Wasm опирается на неявный операндный стек и структурированное управление потоком [11]. Инструкции потребляют значения с вершины стека и помещают результаты обратно, при этом реализация не обязана хранить «настоящий» стек: спецификация описывает интерпретацию стека как набора анонимных регистров, а статическая типовая проверка делает высоту стека известной на этапах валидации и компиляции. Типизация в Wasm является статической и рассчитана на проверку до исполнения: функции имеют явные типы параметров и результатов, инструкции типизируются через эффекты над стеком, а модуль проходит валидацию, гарантирующую корректность применения инструкций к операндам нужных типов и отсутствие underflow-ошибок. Верификация модуля выполняется за один проход, что соответствует целям языка по скорости исполнения [11]. Управление потоком задаётся структурными конструкциями (block, loop, if) и переходами к меткам (br, br_if, br_table). Единица трансляции Wasm называется модулем [11].

В качестве оптимизационных целей Wasm фиксируются компактность двоичного представления и быстрая однопроходная валидация и компиляция [11]. Низкоуровневый стековый байткод занимает меньше пространства и может исполняться с меньшими накладными расходами по сравнению с традиционным для браузеров кодом на ECMAScript, поскольку исполняется в специализированной виртуальной машине и компилируется в машинный код средствами рантайма.

Управление памятью осуществляется в ручном режиме: так как Wasm является целью компиляции, ответственность за гарантии при работе с

памятью ложится на средства исходного языка [11] (например, статические проверки Rust, сборщики мусора, либо, в случае C/C++, на разработчика программного обеспечения).

Язык и виртуальная машина используются в качестве безопасной «песочницы» с производительностью низкоуровневого кода, в основном исполняемой в веб-браузерах. По сравнению с ECMAScript, байткод Wasm обычно требует меньших накладных расходов при выполнении (в частности, по сравнению с исполнением через движки ECMAScript, например V8), что делает его удобной целью компиляции для переносимых модулей клиентского кода.

2 Результаты сравнительного анализа

Для обеспечения полноты работы необходимо сделать сравнительный анализ свойств описанных языков. Далее предложены сранительные анализы по различным свойствам.

2.1 Вывод

В целом же в пути развития стековых языков видна явная тенденция: изначально работающие на высоких скоростях и низких уровнях абстракций благодаря простоте, высокой переносимости и оптимизациям (например, нитевой код Forth), доступным фактическо бесплатно за счет структуры программ и модели исполнения, стековые языки обзаводятся мощной формальной базой, которая позволяет точно описывать работу программ и проводить мощный статический анализ. Статический анализ открывает для стековых языков новые возможности: верификация кода и агрессивные оптимизации являются несомненно большими преимуществами для низкоуровневых языков.

Одной из вершин достижений стековых языков можно считать Wasm. Исходные цели проектирования [11] диктуют идеальные для использования стекового языка условия, на основе которых выросла система, сочетающая в себе сразу несколько преимуществ, изначально считающихся несочетаемыми: отличная портируемость под разные платформы, но при этом высокая производительность (например, в сравнении с традиционными движками ECMAScript) за счет низкоуровневости и мощных оптимизаций, а также верифицируемость (и оттого частично безопаность). Wasm вбирает в себя теоретические и практические наработки в области стековых языков, решая с помощью них сложную задачу исполнения прикладного кода с учетом заданных требований.

В результате анализа можно сделать вывод, что современная ниша стековых языков – это быстрые виртуальные машины, запускаемые на широком диапазоне устройств. Это утверждение также подтверждается фактом, что спецификации самых распространенных виртальных машин (JVM, .NET CLR, Wasm) используют стековую модель исполнения кода.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

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

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

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

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

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

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

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

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

11. WebAssembly Specification 1.0 [Электронный ресурс]. URL: https://webassembly.github.io/spec/versions/core/WebAssembly-1.0.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. Cat language repository [Электронный ресурс]. URL: https://github.com/cdiggins/cat-language (дата обращения: 10.10.2025).

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