ТАРТУСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

На правах рукописи

Пёйал Яанус Раймондович

УДК 681.3.06

СПЕЦИФИКАЦИИ ФОРТ-ПРОГРАММ И ИХ ПРИМЕНЕНИЕ В СИСТЕМАХ ПОСТРОЕНИЯ ТРАНСЛЯТОРОВ

05.13.11 - математическое и программное обеспечение вычислительных машин и систем

02 сентября 1986 г. Pöial

Диссертация
на соискание ученой степени
кандидата технических наук

Научный руководитель :
кандидат физико-математических наук,
профессор Каазик Ю.Я.

Тарту - 1986

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность работы. Развитие вычислительной техники и расширение области ее применения требуют повышения производительности труда в программировании. Одним средством решения этой задачи является разработка и внедрение инструментальных систем и систем построения трансляторов (СПТ).

На данном этапе актуальна проблема создания подобных систем для персональных ЭВМ, т.к. именно персональные ЭВМ применяются в самых разных областях человеческой деятельности. В то же время ресурсы широко распространенных микро-ЭВМ (объем оперативной памяти, быстродействие и т.п.) обычно весьма ограничены. Поэтому непосредственное использование тех систем и методов, которые разработаны для больших и средних ЭВМ, затруднено, но тем не менее нужно учитывать накопленный в этой области опыт.

В условиях быстрого развития микропроцессорной техники нецелесообразно разрабатывать программное обеспечение заново для каждого нового типа ЭВМ. Таким образом, переносимость программного обеспечения является актуальным требованием.

В практике последних лет средством создания мобильного и компактного программного обеспечения микро-ЭВМ (в том числе для разработки СПТ) служили расширяемые языки типа ФОРТ ([54, 47, 51, 52, 35, 56]).

Актуальной задачей является изучение и совершенствование этого инструмента в целях повышения надежности программного обеспечения. Необходимо обратить внимание на улучшение методов описания и реализации языков программирования в ФОРТ-

ориентированных СПТ ([2, 3, I0, II, I2, I8, I9]) на основе имеющегося опыта.

Цель работы. Основными целями данной диссертационной работы являются:

I. Исследование и разработка методов описания языков программирования в ФОРТ-ориентированных СПТ.
2. Разработка методов, позволяющих повысить надежность системы программирования ФОРТ и созданных на ее основе СПТ.

Методы исследования. В работе используются методы теории формальных языков и методы теории полугрупп, а также некоторые положения технологии программирования.

Научная новизна.

I. Определено понятие спецификации ФОРТ-программы и разработан новый формальный аппарат - алгебра спецификаций языка ФОРТ. Изучены свойства этой алгебры.
2. На основе алгебры спецификаций определено понятие корректной ФОРТ-программы и предложено решение проблемы статического контроля типов для языка ФОРТ.
3. Проанализированы существующие методы описания языков в ФОРТ-ориентированных СПТ и введено определение корректности таких описаний.
4. Разработаны методы, позволяющие проверить корректность описания исходного языка относительно вышеуказанного определения и выявить некоторые ошибки в этом описании на самых ранних этапах реализации языка.

Практическая ценность и реализация результатов. Предложенные в работе методы позволяют повысить надежность системы программирования ФОРТ и созданных на ее основе СПТ. Пользователям ФОРТ-ориентированной СПТ предлагаются средства для создания и отладки описаний реализуемых языков.

Все полученные результаты применяются в СПТ ТАРТУ, но статический контроль типов для языка ФОРТ можно использовать и отдельно.

СПТ ТАРТУ реализована на языке fig-FORTH и внедрена на ЭВМ СМ-4, Apple-II и Искра-226.

Структура работы.

В первой главе рассматриваются инструментальные системы программирования на базе языков типа ФОРТ. Дается характеристика таких языков как инструмента реализации компактного и мобильного программного обеспечения. Анализируются недостатки языка ФОРТ и недостатки существующих ФОРТ-ориентированных СПТ. Рассматривается СПТ ТАРТУ, в контексте которой применяются основные результаты данной диссертации. В результате анализа уточняются конкретные задачи исследования.

В разделе I.I. обсуждаются некоторые вопросы, связанные с областью автоматизированной реализации языков программирования. Рассматриваются мобильность программного обеспечения, выбор промежуточного языка в трансляторах, создание набора абстракций в виде хорошо структурированной совокупности модулей и т.п. ([37]). Приводятся примеры систем, ориентированных на конкретный язык высокого уровня ([32]).

В разделе I.2. описываются системы программирования на

основе "сшитого кода" ([28]) и языки типа ФОРТ. Анализируются недостатки подобных систем.

Раздел I.3. посвящен изучению отечественных ФОРТ-ориентированных СПТ (рассматриваются 4 системы).

В разделе I.4. подробнее описывается конкретный представитель ФОРТ-ориентированных систем - СПТ ТАРТУ ([18, 19]).

В разделе I.5. приводятся основные выводы первой главы .

Во второй главе определяется понятие спецификации ФОРТ-программы и разрабатывается новый формальный аппарат - алгебра спецификаций языка ФОРТ. На базе алгебры спецификаций определяется понятие корректной ФОРТ-программы и предлагается частичное решение одной острой проблемы в языке ФОРТ - проблемы контроля типов передаваемых через магазин (стек) параметров. В целях получения практических алгоритмов изучаются алгебраические свойства полугруппы спецификаций и свойства частичного порядка на этой полугруппе.

В третьей главе рассматривается применение алгебры спецификаций в СПТ. Определяется корректность схемы синтаксически управляемого перевода (СУ-схемы, см. [44, 25]),выходным языком которой является ФОРТ. Доказывается основная теорема о корректности перевода. На базе этой теоремы создаются алгоритмы проверки корректности. Рассматривается практическое применение предложенного метода спецификации.

В заключении перечисляются основные результаты диссертационной работы и направления дальнейших исследований.

Список использованной литературы содержит 64 наименования. Бо́льшая часть этих работ посвящена инструментальным системам программирования на базе языков типа ФОРТ.

В приложении I приводится реализация алгебраических операций над ФОРТ-спецификациями.

В приложении 2 излагается реализация статического контроля типов для ФОРТ-программ.

Программы в приложениях написаны на языке fig-FORTH.

Глава I. ИНСТРУМЕНТАЛЬНЫЕ СИСТЕМЫ ПРОГРАММИРОВАНИЯ НА БАЗЕ ЯЗЫКА ФОРТ

§ I.I. Введение

Наличие инструментальных систем позволяет существенно повысить производительность труда в программировании. Уже достигнут немалый успех в автоматизации процесса реализации языков программирования. В обзоре [37] обсуждаются некоторые технологические аспекты создания систем построения трансляторов и обобщается накопленный в этой области опыт.

Отмечается, что актуальным требованием к программному обеспечению является его переносимость - т.е. способность программного обеспечения работать на разных архитектурах ЭВМ. В СПТ такое требование привело к попытке разрабатывать универсальный промежуточный язык для трансляторов - UNCOL . При переходе на новую архитектуру следовало-бы переписать только UNCOL-транслятор. К сожалению разработка такого языка не удавалась.

Позже та же идея использовалась для достижения переносимости конкретных языков - т.н. "p-code" является промежуточным языком в PASCAL -трансляторах, DIANA - в ADA-трансляторах.

Во многих СПТ переносимость достигается посредством ориентации на какой-то широко распространенный язык высокого уровня. Приведем несколько примеров таких систем (данные взяты из обзора [32]) :

СУПЕР, CWS1, CWS2, LILA, LINGUIST-86 - системы, ориентиро-

ванные на язык PASCAL ;

YACC, VATS, APARSE - на язык C ;

ELMA - на язык FORTRAN ;

LINGUA - на язык PL/I .

В дальнейшем используем термин "ориентированный на язык L ", если L является одновременно языком реализации СПТ и объектным языком трансляторов.

В языках программирования встречаются разного рода абстракции и понятия, которые достаточно универсальны (цикл , подпрограмма, массив, тип данных и т.п.). Поэтому наряду с СПТ целесообразно создавать набор таких абстракций, позволяющий разработчику транслятора использовать необходимые понятия "в готовом виде". Такой набор может, например, представлять собой хорошо структурированную совокупность готовых модулей, которая расширяется пользователем СПТ. Способ организации набора абстракций зависит, конечно, от конкретной системы.

Перспективным направлением является создание машин, ориентированных на языки высокого уровня. В этой связи в обзоре [37] отмечается, что наличие подобных машин способствует решению проблемы эффективной генерации кода в трансляторах.

Рассмотренные направления хорошо согласуются с концепциями ФОРТ-ориентированных систем, которым посвящены последующие разделы данной главы.

§ I.2. Характеристика языков типа ФОРТ

I.2.I. Обзор литературы

Проблемы производительности труда в программировании были актуальны уже в начале I960-х годов ([55]). Работа над инструментальными средствами и технологией программирования мини-ЭВМ приводила в I969 г. к созданию целостной и развитой системы программирования с названием ФОРТ (FORTH) Автором языка ФОРТ является Ч. МУР ([54, 55, 57]).

Более широкое распространение язык ФОРТ нашел в связи с появлением микро-ЭВМ, т.к. ресурсы первых микро-ЭВМ были весьма ограничены, а характерным свойством системы ФОРТ является именно ее компактность. В настоящее время язык ФОРТ реализован почти что для всех мини- и микро-ЭВМ. Перечислим некоторые отечественные реализации :

- Электроника-60 [2I],
- СМ-4 [8],
- ЕС (комплекс ЕС-7970) [5],
- БЭСМ-6 [I],
- Искра-226 [I3].

Следует подчеркнуть, что язык ФОРТ не является отдельностоящим феноменом. Те же принципы, которые нашли отражение в системе ФОРТ, используются в других разработках, развиваются и обобщаются в практике программирования. В этой связи отметим концепцию "рабочей смеси" [20] и систему ДССП [6].

На основе реализации языка ФОРТ лежит т.н. "сшитый код" (threaded code) [28]. Техника "сшитого кода" используется

в разных приложениях ([33, 42]), в том числе при разработке языков программирования [39, 47, 58]. Такими языками являются, например, STOIC [59], IPS [53], Fifth [49], ДССП [6], SL/5 , MAGIC/L и др.

Языки типа ФОРТ используются в самых разных областях программирования - при разработке операционных систем ([57 , 34]), при создании баз данных ([4I]), в области искусственного интеллекта ([29]), а также в астрономии, в промышленности, в образовании, при разработке разных систем реального времени и т.п. ([48]). В данной работе рассматривается применение языка ФОРТ при автоматизированной реализации языков программирования.

Появление новых, более мощных микро-ЭВМ не закрывает пути развития языка ФОРТ ([27, 60, 6I]). Уже созданы т. н. ФОРТ-машины ([64, 4, 34, 48]), производительность которых достигает 20 миллионов примитивных ФОРТ-операций в секунду ([48]).

Полное описание языка ФОРТ содержится в работах [35, 47, 5I, 52, 56]. Разработан стандарт I983 года ([35]).

I.2.2. Основные концепции языка ФОРТ

Синтаксической единицей ФОРТ-программы является "слово", причем разделителем слов служит пробел. Программирование в системе ФОРТ представляет собой постепенное расширение языка новыми словами. При определении нового слова можно использовать все имеющиеся в системе слова. Новые слова включаются в язык и уже ничем не отличаются от "стандартных" слов. Ядро системы содержит т.н. примитивы, которые реализованы на языке ассемблера конкретной машины. Объем машинно-зависимой части системы обычно не превышает 2К байт.

Расширение языка может происходить на разных уровнях :

а) создание примитивов для реализации понятий некоторой области, связанной с "внешним миром", или для использования всех возможностей данной ЭВМ,

б) определение новых понятий в терминах уже имеющихся понятий (основной вид ФОРТ-программирования),

в) расширение языка новыми типами данных,

г) введение новых конструкций управления и других понятий , влияющих на фазу компиляции ФОРТ-программы,

д) расширение языка новыми средствами его расширения.

О расширяемости языка ФОРТ подробнее написано в работе [40].

ФОРТ-система является открытой - все ее части реализованы в едином стиле и могут использоваться в прикладных программах. Даже язык ассемблера конкретной машины реализуется как совокупность ФОРТ-слов (т.н. ФОРТ-ассемблер).

Характерной чертой языка ФОРТ является наличие единого механизма передачи данных - магазина. Все понятия получают исходные данные из магазина и возвращают результаты в магазин. Поэтому в ФОРТ-программах используется т.н. обратная польская запись ([25]).

Развитость средств расширения и открытость придают системе необходимую гибкость - несущественные различия в разных версиях языка легко устраняются посредством переопределения некоторых понятий.

Небольшой объем машинно-зависимой части, несложная реализация и гибкость создают благоприятные условия для достижения мобильности системы ФОРТ и написанного на языке ФОРТ программного обеспечения ([29]).

С точки зрения программиста важным качеством является интерактивность языка. Способ передачи параметров через магазин позволяет тестировать и отладить ФОРТ-программы без лишних затрат на создание среды. Если понятие определено, то оно готово к выполнению без всяких дополнительных действий.

Конструкции управления, которые включены в ядро системы, являются структурными. Механизм вызова подпрограмм поддерживает программирование в понятиях нужного уровня. В работе [34] отмечается, что использование языка ФОРТ повышает дисциплинированность программистов.

ФОРТ-программы очень лаконичны ([43, 62]). С одной стороны, такое качество не позволяет читать ФОРТ-программы поверхностно. С другой стороны, ФОРТ-программы очень компактны во внутреннем представлении.

Внутреннее представление ФОРТ-программы типично представляет собой "сшитый код", который примерно I.5 раза компактнее программы, написанной на языке ассемблера. Особенно существенный эффект достигается при реализации больших систем. Интерпретатор "сшитого кода" примерно I0 раз превышает в скорости интерпретатор языка BASIC, работая лишь 4-5 раз медленнее машинного кода. Данные об эффективности взяты из работы [45].

Расширяемость языка ФОРТ позволяет настраивать язык на нужную проблемную область и создавать понятия достаточно высокого уровня. Поэтому актуальной задачей будущего является выявление базовых понятий для конкретных проблемных областей и их реализация, вплоть до создания проблемно-ориентированных виртуальных ФОРТ-машин ([63]).

I.2.3. О недостатках языка ФОРТ

Наряду с преимуществами, которыми обладает язык ФОРТ при разработке программного обеспечения микро-ЭВМ, в литературе уделяют внимание на недостатки этого языка. Обсуждаются следующие проблемы :

- ядро языка не содержит арифметики чисел с плавающей точкой ([45, 24, 48]);
- стандартная файловая система (экраны) является весьма примитивным механизмом, не позволяющим решать более сложные задачи в области системного программирования ([45, 24]);
- не хватает динамического распределения памяти ([50]) ;
- обратная польская запись непривычна программистам ([50 , 24]) ;
- ФОРТ-программы слишком лаконичны и нелегко читаются ([45, 48, 50, 63]) ;
- в языке нет средств контроля типов передаваемых через магазин параметров (ответственность за свои ошибки полностью лежит на пользователе) ([24, 45, 50, 63]).

В работе [63] отмечается, что большинство ошибок в ФОРТ программах связана с неправильным использованием магазина . Подчеркивается, что важнейшей задачей при документировании ФОРТ-программы является спецификация использования магазина программой, т.е. указание числа и типов входных и выходных параметров. Такие спецификации существенно повышают читаемость ФОРТ-программ.

§ I.3. Применение языка ФОРТ в СПТ

I.3.I. Общая характеристика ФОРТ-ориентированных СПТ

Опираясь на перечисленные в разделе I.I направления в технологии разработки СПТ, рассмотрим те качества языка ФОРТ, которые обосновывают создание ФОРТ-ориентированных СПТ.

I. ФОРТ является распространенным языком и легко реализуется на микро- и мини-ЭВМ.

2. Программное обеспечение, написанное на языке ФОРТ, относительно мобильно. Такое качество облегчает перенос СПТ и построенных в этой системе трансляторов на новые модели ЭВМ.

3. ФОРТ достаточно эффективен по сравнению с языками высокого уровня (особенно следует отметить компактность ФОРТ - программ). Появление высокопроизводительных ФОРТ-машин в некоторой степени решает проблемы эффективности интерпретации ФОРТ-программ.

4. Гибкость и расширяемость позволяют настраивать язык на нужную проблемную область и работать в понятиях этой проблемной области.

5. ФОРТ является открытой системой программирования,все компоненты которой (ФОРТ-ассемблер, редактор текстов, монитор внешних устройств, разные пакеты и т.п.) выполнены в едином стиле - этим создана целостная среда программирования, в которой используется только один язык.

Известно, что языки "сшитого кода" использовались при непосредственной реализации языков Pascal [42], Extended Basic, UCSD Pascal , Lisp [34].В дальнейшем идея использова-

ния языка ФОРТ в качестве промежуточного языка транслятора развивалась и приводила к созданию ФОРТ-ориентированных СПТ.

Автору данной работы известны четыре системы :

I) система "ШАГ" (Агамирзян И.Р. - ИТА АН СССР, Ленинград), [2, 3] ;

2) система, разработанная в ЛГУ (авторы Кириллин В.А., Клубович А.А., Ноздрунов Н.Р. - Ленинград), [II, I2] ;

3) система, разработанная в РГУ (авторы Литвиненко А.Н., Демин С.П. - ВЦ РГУ, Ростов-на-Дону), [I0] ;

4) СПТ ТАРТУ (Томбак М.О., Соо В.К., Пёйал Я.Р. - ТГУ, Тарту), [I5, I6, I7, I8, I9].

Все эти системы основываются на языке ФОРТ (хотя на разных версиях ФОРТ-а), используя ФОРТ как целостную технологическую среду. Средства описания синтаксиса в этих системах разные и не рассматриваются здесь подробнее. Общей чертой является использование расширенного (ФОРТ-ориентированного) метода семантических процедур, при котором "процедурами" являются любые средства языка ФОРТ - подпрограммы, типы данных, макросредства и т.п. Эти т.н. семантические понятия образуют своего рода лексикон языка, элементы которого связываются с (абстрактной) синтаксической структурой языка. Несмотря на различия в методах установления такой связи можно сказать, что рассматриваемые системы создают средства для перевода с исходного языка на язык семантических понятий. В пункте I.4.2 излагается один из методов описания такого перевода.

I.3.2. О недостатках существующих систем

Особую роль в СПТ играют средства, позволяющие проверить корректность некоторых аспектов описания исходного языка (напр. алгоритм Кнута для проверки корректности системы правил вычисления значений атрибутов [46]). Наличие таких средств повышает надежность создаваемых трансляторов и дает возможность обнаружить некоторые ошибки в описании языка на самых ранних этапах разработки.

В первых трех системах, которые перечислялись в предыдущем разделе, упомянутые средства обнаружения ошибок в описании семантики языка практически отсутствуют (но имеются средства отладки, которые позволяют выявить подобные ошибки позже).

Наиболее трудным вопросом в рассматриваемых системах является согласование набора семантических понятий с синтаксической структурой реализуемого языка. Первым шагом при решении этой проблемы является создание средств, позволяющих по описанию семантических понятий и описанию синтаксической структуры проверить те аспекты их согласованности, которые являются самыми важными. От выбора таких аспектов зависит аппарат описания семантических понятий.

Таким образом, актуальной задачей является оснащение существующих систем более совершенными средствами отладки описаний реализуемых языков.

§ I.4. СПТ ТАРТУ

I.4.I. Общие сведения о системе

В настоящем разделе рассматривается такая система построения трансляторов (СПТ ТАРТУ), в которой объектным языком является язык ФОРТ. Сама система также написана на этом языке.

В общем случае результатом работы СПТ ТАРТУ являются интерпретаторы, но если ФОРТ-система оснащена средствами целевой компиляции или реализована в виде ФОРТ-машины, то можно получить и компиляторы. СПТ ТАРТУ позволяет построить также трансляторы интерактивных языков программирования и способна работать в условиях ограниченной оперативной памяти - в этом выражается ориентированность системы на микро-ЭВМ.

В целях достижения компактности реализация системы основывается на концепции разреженного дерева вывода ([38 , 7]).

Рассмотрим теперь методы описания языков в данной СПТ. Сначала исходим из предположения, что семантику реализуемого языка L можно представить как описание перевода с языка L на некоторый другой язык M, семантика которого считается известной (см. [23]). Сложность такого описания зависит от выбора языка M. Правила перевода усложняются в случае, если язык L содержит конструкцию, которая не имеет аналога в M. В таком случае придется моделировать эту конструкцию прави-

лами перевода, формализм которых для такой задачи не очень удобен. Поэтому нецелесообразно строго зафиксировать язык M. Если же выбрать в качестве M расширяемый язык, то нужно разбить описание семантики на две части - перевод с языка L на (расширенный) язык M и расширения языка M, которые реализуют необходимые понятия языка L. В данном случае в качестве языка M выбран язык ФОРТ. С одной стороны, этим достигается мобильность системы, с другой стороны, упрощается аппаратура перевода (перевод в постфиксную запись, см. [26], стр.210).

Описание исходного языка в СПТ ТАРТУ состоит из нескольких частей. Рис. I на стр. 23 иллюстрирует связи этих частей с этапами работы транслятора.

Описание перевода основывается на формализме синтаксически управляемого перевода ([44, 25]) и представляет собой КС-грамматику исходного языка, к правилам которой приписаны т.н. компоненты перевода, Исходная грамматика должна принадлежать классу LR(K) - преобразование в грамматику предшествования производится автоматически ([22]). В случае, если полученная грамматика предшествования не позволяет построить MSP-анализатор, пользователю, к сожалению, придется преобразовывать грамматику вручную.

Семантика получаемого в результате перевода ФОРТ—текста описывается как совокупность ФОРТ-понятий. Некоторые из них (т.н. IMMEDIATE-понятия) управляют ФОРТ-компиляцией, а остальные запускаются во время выполнения программы. В терминах описания языка можно сказать, что во время ФОРТ-компиляции работают понятия, связанные с контекстными условиями и стати-

ческой семантикой исходного языка. В конечном итоге на этом этапе компилируется программа, состоящая из понятий динамической семантики. Внутреннее представление готовой к выполнению программы зависит от конкретной ФОРТ-системы.

Так как СПТ ТАРТУ ориентирована на микро-ЭВМ, она может в дальнейшем служить для ускоренной реализации программного обеспечения на "голом оборудовании".

Пути развития системы :

- улучшение методов описания реализуемых языков,
- создание высокоинтерактивных средств для облегчения работы пользователя,
- накопление "лексикона пользователя СПТ" в виде хорошо структурированной совокупности ФОРТ-понятий.

Система реализована на языке fig-FORTH на ЭВМ :

- СМ-4 (ОС-РВ),
- СМ-4 (UNIX),
- AppIe II,
- "Искра-226" (разработка ведется).

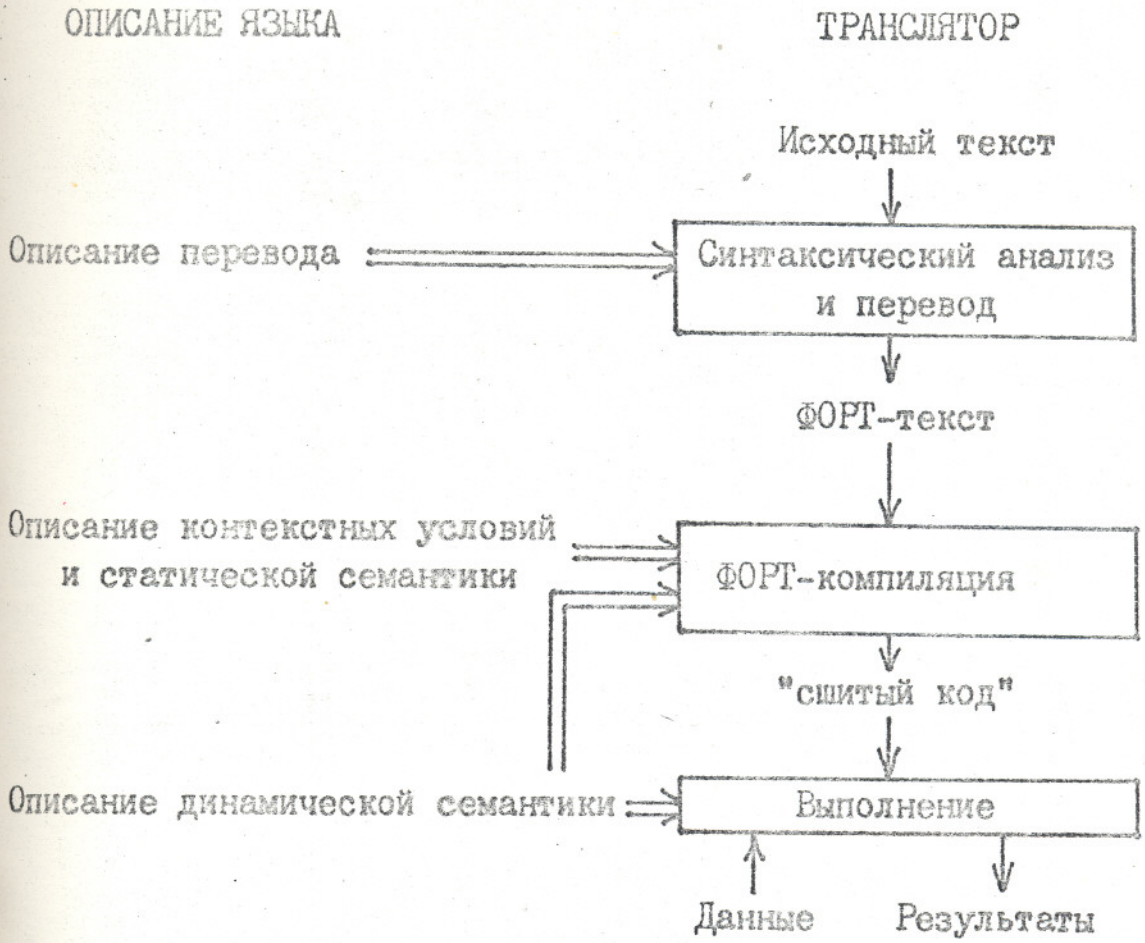
ОПИСАНИЕ ЯЗЫКА
ТРАНСЛЯТОР
Исходный текст
Описание перевода
Синтаксический анализ и перевод
ФОРТ-текст
Описание контекстных условий и статической семантики
ФОРТ-компиляция
"сшитый код"
Описание динамической семантики
Выполнение
Данные
Результаты

Рис. I

I.4.2. Синтаксически управляемый перевод

Схема СУ-перевода - это кортеж $T = (N, \Sigma, \Delta, R, S)$, где N - нетерминальный алфавит, $S \in N$ - фиксированный начальный символ (аксиома), Σ - входной алфавит, Δ - выходной алфавит, а R - множество правил перевода вида

$$A_0 \to x_0 A_1 x_1 \dots x_{n-1} A_n x_n \ , \ z_0 B_1 z_1 \dots z_{n-1} B_n z_n \qquad (1)$$

($x_i \in \Sigma^*$, $z_i \in \Delta^*$, A_i, $B_i \in N$). Вектор $(B_1, \dots, B_n)$ является некоторой перестановкой вектора $(A_1, \dots A_n)$.

Если $(B_1, \dots, B_n) = (A_1, \dots, A_n)$ во всех правилах перевода из R , то получаем схему т.н. простого СУ-перевода.

Схема СУ-перевода естественным образом определяет множество пар цепочек, выводимых из пары (S, S). Первые компоненты всех выводимых пар составляют входной язык L_1 , а вторые компоненты - выходной язык L_2 . Если (x, y) является выводимой парой, то цепочка y называется переводом цепочки x.

В нашем случае рассматривается перевод на язык ФОРТ, в котором большинство конструкций имеет постфиксный вид. Известно, что для перевода из инфиксной записи в постфиксную достаточно пользоваться такой простой СУ-схемой, в которой $z_0 = \dots = z_{n-1} = \Lambda$ (Λ - обозначение пустой строки). Нам придется рассматривать более общий случай, так как некоторые ФОРТ-конструкции отличаются от постфиксного вида, а также не все исходные языки строго инфиксны.

В дальнейшем будем использовать такую схему перевода, в которой все правила имеют вид

$$A_0 \to x_0 A_1 x_1 A_2 \dots x_{n-1} A_n x_n \ , \ u B_1 B_2 \dots B_n v \qquad (2)$$

($u, v \in \Delta^*$).

Оказывается, что при помощи правил такого вида можно задавать любой СУ-перевод. В общем случае для этого придется разбивать правила вида (I) на куски вида (2), т.е. вводить новые нетерминалы и дополнительные правила.

В каждой СПТ имеются средства для лексического анализа. Иногда в этой области применяется понятие класса лексем, которое позволяет сократить описание исходного языка (одним метасимволом выражается целое множество конкретных лексем). Обычно выделяются классы "идентификатор", "константа" и т.п. Для нас существенно, чтобы представители таких классов сохранялись при переводе. Поэтому будем считать, что Σ и Δ содержат одинаковые классы лексем, а соответствующие метасимволы трактуем как нетерминальные. Такое соглашение позволяет также переставить представители классов лексем, если это нужно.

Пример I. Рассмотрим следующую СУ-схему :

N = { Seq, Stm, Dcl, Id, Dim, Const, Var, Idlp, Exp, Mon },

Σ = { ';' , ':=', 'VAR', 'DIM', '(', ')' ,[I], [C], '+' } ,

Δ = { '!' , 'VAR' , 'ARRAY' , [I], [C], '+', '@' } ,

S = Seq и R состоит из правил :

I)	Seq	→	Stm	, Stm
2)	Seq	→	Seq ';' Stm	, Seq Stm
3)	Stm	→	Dcl	, Dcl
4)	Stm	→	Var ':=' Exp	, Exp Var '!'
5)	Dcl	→	'VAR' Id	, 'VAR' Id
6)	Dcl	→	'DIM' Id '(' Dim ')'	, Dim Id
7)	Id	→	[I]	,[I]

8) Dim	→	Const	, Const 'ARRAY'
9) Const	→	[C]	, [C]
10) Var	→	Id	, Id
11) Var	→	Idlp Exp ')'	, Exp Idlp
12) Idlp	→	Id '('	, Id
13) Exp	→	Mon	, Mon
14) Exp	→	Exp '+' Mon	, Exp Mon '+'
15) Mon	→	Const	, Const
16) Mon	→	Var	, Var '@'

Здесь [I] обозначает класс идентификаторов, [C] - класс целых констант без знака, а апострофы трактуют как метасимволы, выделяющие ключевые слова и разделители.

При такой интерпретации программе

Var A ; dim B(6) ; A := 4 ; B(3) := A+2

соответствует перевод

```
VAR A
6 ARRAY B
4 A !
A @ 2 + 3 B !
```

Так как правила перевода имеют вид (2), то способ их задания может быть несколько упрощен. Для этого отбрасываем вторые компоненты правил и введем новые обозначения :

PRE = u ,

POST = v ,

PERM = $(k_1 , \dots , k_n)$ задает перестановку вектора $(1,\dots, n)$, причем i -тая компонента вектора $(B_1,\dots,B_n)$ определяется как k_i -тая компонента вектора $(A_1,\dots, A_n)$.

По этой информации вторые компоненты правил вида (2) могут быть восстановлены. Для сокращения записи при этом считаем, что по умолчанию PRE= Λ , POST = Λ и PERM = (1,..., n).

Пример 2. Для задания СУ-схемы примера I достаточно дополнить первые компоненты правил перевода (которые образуют КС-грамматику исходного языка) следующей информацией :

4) PERM = (2, 1) , POST = '!'

5) PRE = 'VAR'

6) PERM = (2, 1)

8) POST = 'ARRAY'

II) PERM = (2, 1)

I4) POST = '+'

I6) POST = '@'

Указанный вид схемы СУ-перевода позволяет использовать дерево вывода исходной программы (см. [25] , стр.I63) для выполнения перевода. Чтобы обеспечить перевод представителей классов лексем, определяем еще PRE и POST для концевых вершин дерева вывода следующим образом. В каждой такой вершине PRE = Λ ; если вершине соответствует представитель класса лексем (идентификатор, константа и т.п.), то принимается POST = текст лексемы, в противном случае POST = Λ .

Для внутренних вершин PRE , POST и PERM уже определены, так как каждой внутренней вершине соответствует некоторое правило перевода.

Приведем теперь алгоритм СУ-перевода на дереве вывода программы (W является любой вершиной дерева вывода).

АЛГОРИТМ GENTEXT(W) :

G1. Выдать текст PRE(W).

G2. Если W имеет прямые нетерминальные потомки $V_1,\ldots, V_n$, то переставить их согласно PERM(W) . Вместе с вершинами переставляются и поддеревья, корнями которых они служат .

G3. Если W имеет прямые потомки $W_1,\ldots, W_k$, то для каждого $i = 1,\ldots,k$ выполнить GENTEXT(W_i).

G4. Выдать текст POST(W).

G5. Выход.

Выполнив этот алгоритм на корне дерева вывода получим в выходном потоке перевод входной цепочки.

I.4.3. Модификация метода перевода

Главную трудность перевода составляют структурные различия входного и выходного языков. СУ-схема является удобным средством для перевода отдельных конструкций языков программирования. Но выбранный нами объектный язык - ФОРТ - налагает некоторые более общие требования, которым СУ-схема не всегда удовлетворяет. Например, все объекты языка ФОРТ должны быть описаны до их использования. Если в реализуемом языке такого ограничения нет, то необходимо сначала перевести описания, а лишь потом остальную часть. СУ-схема в этом случае может оказаться громоздкой и плохо поддающейся методам синтаксического анализа (или такой схемы не существует вообще). В целях преодоления подобных трудностей нам пришлось несколько обобщить алгоритм перевода.

В результате выполнения алгоритма GENTEXT над некоторой внутренней вершиной v дерева вывода мы получим в выходном потоке определенную часть перевода. Обозначим такую часть через $\tau(v)$. Соответствующий отрезок входной цепочки образует крону поддерева с корнем v .

Вышеупомянутую проблему о перестановке описаний можно поставить в следующем виде : перевести цепочку

$$u_0 v_1 u_1 v_2 u_2 \dots u_{n-1} v_n u_n$$

в цепочку

$$w_1 w_2 \dots w_n u' \quad ,$$

где $u_i \in \Sigma^*$, $v_j \in \Sigma^*$ является кроной поддерева, корень которого обозначаем через v_j , $w_j = \tau(v_j)$, а $u' \in \Delta^*$ является оставшейся частью перевода.

Чтобы получить более общий алгоритм перевода, будем предполагать, что корни поддеревьев, подлежащих переводу раньше других (вершины V_j), помечены некоторым специальным образом. Такие метки при этом можно приписать к правилам грамматики исходного языка, так как любой внутренней вершине дерева вывода соответствует некоторое правило грамматики. Указанные метки будем в дальнейшем называть режимами и обозначать через REG.

Допускаем в вершинах, которые снабжены режимом, наряду с применением алгоритма GENTEXT возможность некоторых дополнительных действий. Таким образом, REG может иметь разные значения для разных правил грамматики. Каждое такое значение определяет набор действий, подлежащих выполнению при обработке соответствующей вершины дерева вывода.

Приведем теперь алгоритм, управляющий выполнением действий, определяемых режимами (W является любой вершиной дерева вывода).

АЛГОРИТМ SEARCH(W) :

S1. Если W имеет прямые потоки $W_1,\ldots, W_k$, то для каждого $i = 1,\ldots,k$ выполнить SEARCH(W_i).

S2. Если в вершине W определен реким, то :

S2.1. выполнить действия, указанные в REG(W) ,

S2.2. удалить поддеревья с корнями $W_1,\ldots, W_k$, но не считать вершину W в дальнейшем терминальной вершиной (при выполнении шага G2 в алгоритме GENTEXT).

S3. Выход.

Исходной вершиной для алгоритма SEARCH является корень дерева вывода.

Опираясь на модифицированный таким образом алгоритм перевода, переходим к рассмотрению возможных режимов. В данной работе опишим лишь два режима, которые уже применялись при разработке конкретных трансляторов, но в принципе количество режимов не ограничено (пользователю предоставляются средства для расширения набора режимов).

Режим TRANS в вершине W означает, что в выходной поток поступает перевод поддерева с корнем W . Именно этот режим используется для перестановки описаний.

Режим TRANS(W) :

T1. Выполнить GENTEXT(W).

T2. PRE(W) := Λ ; POST(W) := Λ .

ФОРТ-программы обычно представляются как т.н. ":" -определения (colon definitions). Поэтому введен режим COLON , который обеспечит перевод операторов в такие определения . Если в вершине W указан режим COLON , то $\mathcal{T}$(W) оформляется как отдельное ":"-определение, для которого генерируется специальное системное имя. Чтобы в дальнейшем пользоваться этим определением, его имя хранится как POST(W).

Режим COLON(W) :

C1. Генерировать новое системное имя.

C2. Выдать строку, состоящую из ':' и системного имени.

C3. Выполнить GENTEXT(W).

C4. PRE(W) := Λ ; POST(W) : = системное имя.

Предполагается, что строка ';' , закрывающая ":"-определение, выдается алгоритмом GENTEXT .

I.4.4. Пример перевода

В качестве примера рассмотрим небольшой исходный язык, в котором имеются целые константы, переменные, массивы, операторы присваивания и операторы цикла. На стр.34 (таблица I) приведена грамматика этого языка (левый столбец), а также описание перевода (столбцы REG, PRE, POST и PERM) .

Иллюстрируем перевод при помощи следующей исходной программы :

```
var K ;
while K<6
  do
    var S ;
    S := S + A(K) ;
    dim  A(5) ;
    K := K + 1
  od
```

В результате выполнения алгоритма SEARCH над корнем дерева вывода этой программы, получим перевод :

```
VAR K   VAR S   5 ARRAY A
: SYS1
  BEGIN
    V? K  @  6  <
  WHILE
    V? S  @  V? K  @  A?  A  @  +  V? S  !
    V? K  @  1 +  V? K  !
  REPEAT
;
SYS1
```

В этой ФОРТ-программе все объекты определены до того, как они будут использованы. Также выполняется требование, по которому конструкция

BEGIN ... WHILE ... REPEAT

должна находиться внутри ":"-определения. В данном случае ":"-определению дается сгенерированное системное имя SYS1 . Появление имени SYS1 в конце перевода (использование объекта SYS1), гарантирует запуск программы. Именно поэтому в корне дерева должен быть установлен реким TRANS .

Понятия V? и A? являются расширением ФОРТ-а и управляют компиляцией ФОРТ-программы следующим образом :

а) если идентификатор, который следует за словом V? , не является именем переменной, то происходит прерывание компиляции и выдается сообщение об ошибке,

б) слово A? выполняет такуюже проверку для имени массива .

Например программе var P ; P(5) := 3 соответствует перевод

VAR P 3 5 A? P !

и сообщение " P не является именем массива" выдается во время обработки этого ФОРТ-текста.

Таким образом, контексные условия исходного языка переводятся в контекстные условия (расширенного) ФОРТ-а и обрабатываются во время компиляции ФОРТ-текста, т.е. до образования выполняемой ФОРТ-программы. Мы не включили проверку контекстных условий в состав алгоритма перевода по двум причинам :

I) нецелесообразно слишком усложнять формализм перевода,

2) необходимые средства для подобной работы уже существуют в виде возможности управлять ФОРТ-компиляцией.

		REG	PRE	POST	PERM
1	Prog ::= Unit	TRANS			
2	Unit ::= Seq	COLON		;	
3	Seq ::= Stm				
4	::= Seq ';' Stm				
5	Stm ::= Dcl	TRANS			
6	::= Var ':=' Exp			!	2 1
7	::= Header Body		BEGIN	REPEAT	
8	Dcl ::= 'VAR' Id		VAR		
9	::= 'DIM' Id '(' Dim ')'				2 1
10	Id ::= [I]				
11	Dim ::= Const			ARRAY	
12	Const ::= [C]				
13	Var ::= Id		V?		
14	::= Idlp Exp ')'				2 1
15	Idlp ::= Id '('		A?		
16	Exp ::= Mon				
17	::= Exp '+' Mon			+	
18	Mon ::= Const				
19	::= Var			@	
20	Header ::= 'WHILE' Cond			WHILE	
21	Body ::= 'DO' Seq 'OD'				
22	Cond ::= Exp '<' Exp			<	

Таблица 1

§ I.5. Выводы

I. Анализ применяемых методов описания языков в ФОРТ-ориентированных СПТ указывает на необходимость повышения технологичности этих методов.

2. Одним из путей для достижения этой цели является создание средств автоматизированной проверки наиболее важных аспектов согласованности лексикона семантических понятий и синтаксической структуры языка.

3. Ввиду того, что семантические понятия реализуются на языке ФОРТ, существенным аспектом при описании семантических понятий является спецификация использования магазина ими.

4. Для исследования такого аспекта целесообразно создать формальный аппарат.

5. На основе этого аппарата необходимо уточнить понятие корректности описания языка и разработать методы проверки такой корректности.

Глава 2. АЛГЕБРА СПЕЦИФИКАЦИЙ ЯЗЫКА ФОРТ

В данной главе определяется понятие спецификации ФОРТ-программы и вводится новый формальный аппарат - алгебра спецификаций языка ФОРТ. Основное внимание уделяется изучению свойств этого аппарата.

На базе алгебры спецификаций определяется понятие корректной ФОРТ-программы и предлагается частичное решение проблемы контроля типов передаваемых через магазин параметров - статический контроль типов по тексту ФОРТ-программы.

ФОРТ-понятия обмениваются информацией через магазин данных. Ориентированное на человека описание языка всегда содержит спецификации использования магазина для всех понятий в виде :

типы входных параметров --- типы выходных параметров.

Списки типов упорядочены - верхний элемент магазина является последним в списке. Количество типов и их имена выбираются в зависимости от проблемной области. Для стандарта ФОРТ-83, например, перечень имеющихся типов содержится в работе [35] , стр. 23-24.

Если мы рассматриваем типы "адрес" (ad) и " значение " (val), то спецификация понятия "store" (запись значения по адресу) пишется в виде [val ad ---] , а спецификация понятия "fetch" (разыменование) - в виде [ad --- val] . Основой при создании формального аппарата является именно такое, привычное для ФОРТ-программиста понятие спецификации.

§ 2.I. Основные свойства полугруппы спецификаций

Введем сначала некоторые обозначения и определения :

A - алфавит (конечное множество символов),

A^* - множество слов в алфавите A .

Λ - пустое слово ($\Lambda \in A^*$ при любом A),

$|a|$ - длина слова a ,

ab - конкатенация слов a и b .

Примечание. Если $A = \varnothing$ (в данной работе такой случай не исключается), то $A^* = \{\Lambda\}$.

Обозначим т.н. нулевую спецификацию через 0 и определим множество спецификаций над алфавитом A следующим образом :

$$\Phi(A) = (A^* \times A^*) \cup \{0\}.$$

Пару $(s_1, s_2) \in A^* \times A^*$ обозначим через $[s_1 --- s_2]$.

Определим т.н. пустую спецификацию как пару

$$(\Lambda, \Lambda) = [---] \in A^* \times A^*$$

и обозначим ее через $\mathbb{1}$.

В дальнейшем будем писать Φ вместо $\Phi(A)$, если из контекста ясно, о каком алфавите A идет речь.

Определим теперь произведение (композицию) спецификаций следующим образом :

I) $\forall s \in \Phi$: $s0 = 0s = 0$,

2) $\forall s, t \in \Phi \setminus \{0\}$: $st = [s_1 --- s_2][t_1 --- t_2] =$

$$= \begin{cases} [as_1 --- t_2], \text{ если } t_1 = as_2 , \\ [s_1 --- bt_2], \text{ если } s_2 = bt_1 , \\ 0, \text{ в остальных случаях.} \end{cases}$$

Нулевая спецификация характеризует ошибочную ситуацию — несогласованность типов.

Следствие I. Из определения произведения непосредственно следует, что

$$[s_1 \text{ --- } s_2][t_1 \text{ --- } t_2] = [r_1 \text{ --- } r_2] \neq 0$$

тогда и только тогда, когда существуют $a, b \in A^*$ такие, что

I) $a = \Lambda$ или $b = \Lambda$,

2) $as_1 = r_1$,

3) $as_2 = bt_1$,

4) $bt_2 = r_2$.

Пример 3. Рассмотрим алфавит $A = \{ad, val\}$ и вычислим произведения некоторых спецификаций над этим алфавитом :

$$[val \text{ ---}][\text{ --- } ad] = [val \text{ --- } ad]$$
$$[\text{ --- } ad][val \text{ --- }] = 0$$
$$[\text{ --- } ad][val\ ad \text{ ---}] = [val \text{ ---}]$$
$$[ad \text{ --- } val][\text{ --- } ad] = [ad \text{ --- } val\ ad]$$

Теорема I. Множество Φ образует полугруппу с единицей и нулем, т.е.

I) $\forall s, t \in \Phi : st \in \Phi$,

2) $\forall r, s, t \in \Phi : (rs)t = r(st)$,

3) $\forall s \in \Phi : s1 = 1s = s$,

4) $\forall s \in \Phi : s0 = 0s = 0$.

Доказательство.

Справедливость утверждений I, 3 и 4 следует из определения произведения.

Для доказательства ассоциативности умножения рассмотрим таблицу 2 на стр. 40, в которой приведены все варианты выбора r , s и t , такие, что либо $(rs)t \neq 0$, либо $r(st) \neq 0$. Важно заметить, что в остальных случаях $(rs)t = r(st) = 0$.

Пусть $s \in \Phi$. Определим для него элемент $s^{-1} \in \Phi$ следующим образом :

I) если $s = 0$, то $s^{-1} = 0$,

2) если $s = [s_1 \text{ --- } s_2]$, то $s^{-1} = [s_2 \text{ --- } s_1]$.

Непосредственно можно проверить, что $ss^{-1}s = s$ и $s^{-1}ss^{-1} = s^{-1}$. В таком случае элемент s^{-1} называется <u>инверсным</u> к элементу s . Для любых элементов s и t имеют место равенства $(st)^{-1} = t^{-1}s^{-1}$ и $(s^{-1})^{-1} = s$.

Выделим из Φ некоторые подмножества, которые играют важную роль при изучении свойств произведения :

$$L = \{ [a \text{ --- }] \mid a \in A^* \},$$
$$R = \{ [\text{ --- } b] \mid b \in A^* \},$$
$$E = \{ [c \text{ --- } c] \mid c \in A^* \} \cup \{0\}.$$

Соответствующая диаграмма Венна изображена на рис. 2.

Элементы s , для которых выполняется равенство $ss = s$, называются <u>идемпотентами</u> полугруппы.

<u>Теорема 2</u>. Справедливы следующие утверждения :

I) E является коммутативной подполугруппой с единицей и нулем, причем элементы множества E и только они являются идемпотентами полугруппы Φ,

2) L и R являются подполугруппами с единицей, причем $s \in L$ тогда и только тогда, когда $s^{-1} \in R$,

3) $st \in L$ влечет $t \in L$ и $st \in R$ влечет $s \in R$,

4) $LR = \Phi \setminus \{0\}$ и $RL \subset R \cup L \cup \{0\}$,

5) если $s \in L$ и $t \neq 0$ или $s \neq 0$ и $t \in R$, то справедливо $st \neq 0$.

	r	s	rs
1	$[\ r_1 \dashrightarrow r_2\ ]$	$[\ ar_2 \dashrightarrow s_2\ ]$	$[\ ar_1 \dashrightarrow s_2\ ]$
2	$[\ r_1 \dashrightarrow r_2\ ]$	$[\ ar_2 \dashrightarrow dt_1\ ]$	$[\ ar_1 \dashrightarrow dt_1\ ]$
3	$[\ r_1 \dashrightarrow bs_1\ ]$	$[\ s_1 \dashrightarrow s_2\ ]$	$[\ r_1 \dashrightarrow bs_2\ ]$
4	$[\ r_1 \dashrightarrow bs_1\ ]$	$[\ s_1 \dashrightarrow dt_1\ ]$	$[\ r_1 \dashrightarrow bdt_1\ ]$
5	$[\ r_1 \dashrightarrow b'cs_1\ ]$	$[\ s_1 \dashrightarrow s_2\ ]$	$[\ r_1 \dashrightarrow b'cs_2\ ]$

	t	st	(rs)t = r(st)
1	$[\ cs_2 \dashrightarrow t_2\ ]$	$[\ car_2 \dashrightarrow t_2\ ]$	$[\ car_1 \dashrightarrow t_2\ ]$
2	$[\ t_1 \dashrightarrow t_2\ ]$	$[\ ar_2 \dashrightarrow dt_2\ ]$	$[\ ar_1 \dashrightarrow dt_2\ ]$
3	$[\ c'bs_2 \dashrightarrow t_2\ ]$	$[\ c'bs_1 \dashrightarrow t_2\ ]$	$[\ c'r_1 \dashrightarrow t_2\ ]$
4	$[\ t_1 \dashrightarrow t_2\ ]$	$[\ s_1 \dashrightarrow dt_2\ ]$	$[\ r_1 \dashrightarrow bdt_2\ ]$
5	$[\ cs_2 \dashrightarrow t_2\ ]$	$[\ cs_1 \dashrightarrow t_2\ ]$	$[\ r_1 \dashrightarrow b't_2\ ]$

Таблица 2

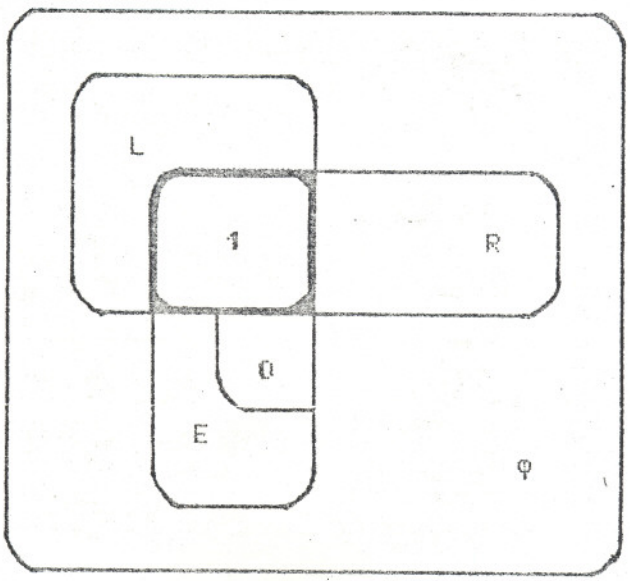

Рис. 2

Доказательство.

I) Замкнутость множества E относительно умножения, коммутативность умножения и идемпотентность элементов множества E проверяются непосредственно по определениям. Остается показать, что больше идемпотентов в полугруппе Φ нет. Для этого предположим, что $s = [c \text{ --- } d]$, $c \neq d$ и $ss = s$. Согласно следствию I должны существовать $a, b \in A^*$ такие, что

(а) $a = \Lambda$ или $b = \Lambda$,

(б) $ac = c$,

(в) $ad = bc$,

(г) $bd = d$.

Из (б) и (г) следуют равенства $a = \Lambda$ и $b = \Lambda$. Тогда в силу (в) справедливо $c = d$, что противоречит предположению $c \neq d$.

2) Справедливость утверждения проверяется непосредственно.

3) Если $st \in L$, то существует $r_1 \in A^*$ так, что выполняется равенство $st = [r_1 \text{ ---}]$. Из следствия I вытекает, что для некоторой $b \in A^*$ имеет место $bt_2 = \Lambda$. Поэтому $t_2 = \Lambda$ и $t \in L$. Вторая часть доказывается двойственным образом.

4) Любой ненулевой элемент $[a \text{ --- } b]$ полугруппы Φ можно представить в виде произведения $[a \text{ ---}][\text{ --- } b]$. С другой стороны, для всех $[a \text{ ---}] \in L$ и $[\text{ --- } b] \in R$ справедливо $[a \text{ ---}][\text{ --- } b] = [a \text{ --- } b] \neq 0$.

Значением произведения $[\text{ --- } b][a \text{ --- }]$ является:

а) $[c \text{ ---}]$, если $a = cb$,

б) $[\text{ --- } d]$, если $b = da$,

в) 0, в остальных случаях.

Следовательно, $[\text{ --- } b][a \text{ --- }] \in R \cup L \cup \{0\}$.

Если алфавит A содержит в точности один символ, то выполняется равенство $RL = R \cup L$. Поэтому в общем случае нельзя утверждать, что $RL = R \cup L \cup \{0\}$.

5) Справедливость утверждения вытекает из определений.

Теорема 2 доказана.

Если любой элемент s полугруппы Φ имеет единственный инверсный к нему элемент s^{-1}, то полугруппа Φ называется инверсной.

Следствие 2. Φ является инверсной полугруппой.

Справедливость этого утверждения вытекает из пункта I теоремы 2 и теоремы I.I7 из книги [30], стр. 50.

Следует отметить, что подполугруппа R и свободная полугруппа A^* изоморфны :

$$[--- c\,][--- d\,] = [--- cd\,].$$

Можно показать, что если $s \in L \setminus \{1\}$, то множества вида $s\Phi$ являются собственными главными правыми идеалами полугруппы Φ, а множества вида Φt, где $t \in R \setminus \{1\}$ - собственными главными левыми идеалами. Более того, Φ является 0-бипростой полугруппой (см. [30], стр.72-74, II0), в которой отсутствуют 0-минимальные идеалы (см. [30], стр. 98). Отметим, что бипростым инверсным полугруппам посвящен § 8.4 в книге [3I].

Идея определения произведения спецификаций таким образом, как это сделано в настоящем разделе, взята из книги [I4], гл. III, § 6. Теорема 2 уточняет свойства произведения (ср. [I4], стр. 302).

§ 2.2. Частичный порядок на полугруппе спецификаций

2.2.I. Свойства частичного порядка

Определим отношение $\leqslant$ на полугруппе Φ, полагая $s \leqslant t$ тогда и только тогда, когда $st^{-1} = ss^{-1}$. Это отношение является частичным порядком (доказательство см. [3I], стр. 55). Так как $0t^{-1} = 00 = 0$, то $0 \leqslant t$ для любого $t \in \Phi$.

Доказано также, что определенное таким образом отношение частичного порядка является стабильным, т.е. $s \leqslant t$ влечет $rs \leqslant rt$, $sr \leqslant tr$ и $s^{-1} \leqslant t^{-1}$.

Легко показать, что неравенство $s \leqslant t$ эквивалентно любому из следующих равенств :

(а) $st^{-1}s = s$,

(б) $ts^{-1}s = s$.

Теорема 3. Следующие утверждения равносильны :

I) $[s_1 -\!- s_2] \leqslant [t_1 -\!- t_2]$,

2) $\exists\, a \in A^*$: $[s_1 -\!- s_2] = [at_1 -\!- at_2]$,

3) $[-\!- s_1][t_1 -\!- t_2][s_2 -\!-] = [-\!-]$,

4) $[-\!- s_1][t_1 -\!- t_2] = [-\!- s_2]$,

5) $[t_1 -\!- t_2][s_2 -\!-] = [s_1 -\!-]$.

Доказательство.

Пусть $[s_1 -\!- s_2] \leqslant [t_1 -\!- t_2]$. Из равенства (б) следует, что

$$[t_1 -\!- t_2][s_2 -\!- s_1][s_1 -\!- s_2] = [s_1 -\!- s_2].$$

Умножив обе части этого равенства справа на элемент $[s_2 -\!-]$, получим равенство $[t_1 -\!- t_2][s_2 -\!-] = [s_1 -\!-]$.

Из следствия I вытекает, что существуют $a, b \in A^*$ такие, что I) $a = \Lambda$ или $b = \Lambda$,

2) $at_1 = s_1$,

3) $at_2 = bs_2$,

4) $b = \Lambda$.

Следовательно, существует $a \in A^*$, при котором

$$[s_1 \text{ --- } s_2] = [at_1 \text{ --- } at_2].$$

Если $s_1 = at_1$ и $s_2 = at_2$, то $[\text{--- } s_1][t_1 \text{ --- } t_2] = [\text{--- } at_1][t_1 \text{ --- } t_2] = [\text{--- } at_2] = [\text{--- } s_2]$.

Теперь умножим оба конца этой цепи равенств справа на элемент $[s_2 \text{ --- }]$ и получим равенство

$$[\text{--- } s_1][t_1 \text{ --- } t_2][s_2 \text{ --- }] = [\text{---}].$$

Далее умножим обе части полученного равенства справа на элемент $[\text{--- } s_1]$, а потом слева на элемент $[s_2 \text{ --- }]$. Таким образом, мы получим равенство $s^{-1}ts^{-1} = s^{-1}$, которое влечет равенство (а). Теорема 3 доказана.

Так как отношение $\leqslant$ является частичным порядком, нас интересует, какие элементы сравнимы между собой. Известно, что нуль сравним с любым элементом.

Теорема 4. Следующие утверждения эквивалентны :

I) $s \neq 0$, $t \neq 0$ и s сравним с t ,

2) существует $r \neq 0$ такой, что $r \leqslant s$ и $r \leqslant t$,

3) существует $u \in \Phi$ такой, что $s \leqslant u$, $t \leqslant u$ и либо $st^{-1} \neq 0$, либо $s^{-1}t \neq 0$.

Доказательство.

Если ненулевые элементы s и t сравнимы между собой, то в качестве элемента r можно использовать минимальный из них.

Если $r \neq 0$, $r \leqslant s$ и $r \leqslant t$, то $s \neq 0$ и $t \neq 0$. Из того, что $r \leqslant s$, следует, что существует $a \in A^*$ такое, что $r_1 = as_1$ и $r_2 = as_2$. С другой стороны , $r \leqslant t$ влечет $r_1 = bt_1$ и $r_2 = bt_2$ для некоторого $b \in A^*$. Рассмотрим два случая.

а) Если $|a| \leqslant |b|$, то определим $d \in A^*$ так, что $ad = b$. Тогда $s_1 = dt_1$ и $s_2 = dt_2$, т.е. $s \leqslant t$.

б) Если $|a| > |b|$, то определим $d \in A^*$ так, что $bd = a$, и получим равенства $t_1 = ds_1$ и $t_2 = ds_2$, т.е. $t \leqslant s$.

Для того, чтобы вывести из утверждения I утверждение 3, рассмотрим опять-таки два случая.

а) Если $s \leqslant t$, то $s_1 = at_1$ и $s_2 = at_2$ при некотором a. Теперь вычислим произведения $st^{-1} = [at_1 — at_1] \neq 0$ и $s^{-1}t = [at_2 — at_2] \neq 0$. В качестве u используем t.

б) Если $t \leqslant s$, то проведем аналогичное рассуждение и определим $u = s$.

Осталось показать, что утверждение 3 влечет утверждение I. Если $st^{-1} \neq 0$ или $s^{-1}t \neq 0$, то $s \neq 0$ и $t \neq 0$. $s \leqslant u$ означает, что $s_1 = au_1$ и $s_2 = au_2$ при некотором a. Из $t \leqslant u$ следует, что существует b, при котором $t_1 = bu_1$ и $t_2 = bu_2$. Теперь, как и раньше, придется рассматривать два случая.

а) Если $st^{-1} \neq 0$, то либо существует $c \in A^*$, при котором $t_2 = cs_2$, либо выполняется равенство $s_2 = dt_2$ для некоторой $d \in A^*$. В первом случае $t_2 = cs_2 = cau_2$, с одной стороны, и $t_2 = bu_2$, с другой стороны. Тогда $b = ca$, из чего получим, что $t_1 = bu_1 = cau_1 = cs_1$ и $t_2 = cs_2$, т.е. $t \leqslant s$. Во втором случае выходит, что $a = db$, $s_1 = dt_1$ и $s_2 = dt_2$, т.е. $s \leqslant t$.

б) Если $s^{-1}t \neq 0$, то проводим аналогичные рассуждения и получим, что либо $s \leqslant t$, либо $t \leqslant s$.

Теорема 4 доказана.

Приведем еще несколько полезных свойств неравенств.

Предложение I. Следующие утверждения равносильны в полугруппе Φ :

I) $s \leqslant rs$,

2) $s = rs$,

3) $ss^{-1} \leqslant r$.

Доказательство.

Из того, что $s \leqslant rs$ следует, что $s^{-1}rs = s^{-1}s$. Тогда имеет место $ss^{-1}rs = ss^{-1}s = s$. Умножив оба конца этой цепи справа на элемент s^{-1}, получим равенство $ss^{-1}rss^{-1} = ss^{-1}$. То же равенство справедливо для инверсных элементов, т.е.

$$(ss^{-1}rss^{-1})^{-1} = (ss^{-1})^{-1} ,$$

откуда получим

$$ss^{-1}r^{-1}ss^{-1} = ss^{-1} .$$

Это равенство влечет $ss^{-1} \leqslant r$.

Если $s = 0$, то все утверждения предложения I справедливы. Пусть $s = [s_1 \text{---} s_2] \neq 0$. Тогда $r \neq 0$ и утверждение 3 принимает вид

$$[s_1 \text{---} s_2]\,[s_2 \text{---} s_1] \leqslant [r_1 \text{---} r_2] \quad , \text{ т.е.}$$

$$[s_1 \text{---} s_1] \leqslant [r_1 \text{---} r_2].$$

Согласно теореме 3

$$\exists\, a \in A^* : \quad s_1 = ar_1 = ar_2.$$

Вычислим теперь произведение rs:

$$rs = [r_1 \text{---} r_2][ar_2 \text{---} s_2] = [ar_1 \text{---} s_2] = s .$$

Если $s = rs$, то, конечно, $s \leqslant rs$ и предложение I доказано.

Из предложения I вытекает, что утверждения $\mathbb{1} \leqslant r$ и $r = \mathbb{1}$ эквивалентны.

Предложение 2. Следующие утверждения равносильны в полугруппе Φ:

I) $s \leqslant st$,

2) $s = st$,

3) $s^{-1}s \leqslant t$.

Предложение 2 доказывается двойственным образом.

Предложение 3. При $s \neq 0$ неравенство $s \leqslant rst$ имеет место тогда и только тогда, когда существуют $a, b, c \in A^*$ такие, что

I) $ar_1 = s_1$,

2) $ar_2 = bs_1$,

3) $ct_1 = bs_2$,

4) $ct_2 = s_2$.

Доказательство опускается, т.к. этот результат в данной работе не используется.

Лемма. Следующие утверждения эквивалентны в любой инверсной полугруппе:

1) $s \leqslant rst$

2) $(ss^{-1} \leqslant rr^{-1}) \& (sts^{-1} = r^{-1}ss^{-1})$

3) $(s^{-1}s \leqslant t^{-1}t) \& (s^{-1}r^{-1}s = ts^{-1}s)$

2.2.2. Понятия верхней и нижней грани

Перейдем теперь к изучению понятий верхней и нижней грани подмножеств полугруппы Φ.

Всякое двухэлементное подмножество $\{s, t\} \subset \Phi$ имеет нижнюю грань, которую можно представить следующим образом :

$$\inf\{s, t\} = \begin{cases} s \text{ , если } s \leqslant t \text{ ,} \\ t \text{ , если } t \leqslant s \text{ ,} \\ 0 \text{ , если } s \text{ несравним с } t. \end{cases}$$

Для нас более существенным является понятие верхней грани. Предположим, что для элементов $s, t \in \Phi \setminus \{0\}$ существуют $a, b, c, d, e \in A^*$ такие, что $s = [abd \text{ —— } abe]$ и $t = [cbd \text{ —— } cbe]$, причем если $a \neq \Lambda$ и $c \neq \Lambda$, то последний символ слова a не совпадает с последним символом слова c. Тогда существует $\sup\{s, t\} = [bd \text{ —— } be]$. Дополнительно определим $\sup\{r, 0\} = r$ для любого $r \in \Phi$. Если s и t не могут быть представлены в вышеуказанном виде, то множество $\{s, t\}$ верхней грани не имеет.

Предложение 4. Имеют место следующие свойства граней :

I) Если подмножество $M \subset \Phi$ имеет верхнюю границу, то для него существует единственная верхняя грань $\sup M$.

2) Если подмножество $M \subset \Phi$ имеет ненулевую нижнюю границу, то $\inf M \in M$.

Первое свойство следует из того, что между верхней границей и любым конкретным ненулевым элементом подмножества M имеется конечное число элементов.

Для доказательства второго свойства отметим, что число элементов, бóльших заданной ненулевой нижней границы, конечно, т.е. M является конечным множеством.

Предложение 5. Подполугруппа идемпотентов $E \subset \Phi$ является решеткой.

Доказательство следует из определений (см. также [30], стр. 45 и [9], стр. 58-66). Если $s \in E$, то выполняется соотношение $0 \leqslant s \leqslant 1$.

§ 2.3. Понятия корректности и замкнутости ФОРТ-программы

Полугруппа Φ была определена как множество спецификаций ФОРТ-понятий, ненулевые элементы которого имеют вид $[\, a \text{ --- } b \,]$, где слово a трактуется как список типов, получаемых из магазина входных данных, а b как список типов, возвращаемых в магазин выходных данных (верхний элемент магазина является в списках последним). Нулевая спецификация характеризует ошибочную ситуацию - несогласованность типов.

Обозначим множество рассматриваемых ФОРТ-понятий через Δ и определим отображение $s : \Delta^* \to \Phi$:

1) $\forall \Pi \in \Delta :$ $s(\Pi) \in \Phi \setminus \{0\}$ является спецификацией понятия Π,
2) $s(\Lambda) = 1$,
3) $\forall w \in \Delta^*, \forall \Pi \in \Delta :$ $s(w\Pi) = s(w)s(\Pi)$.

Назовем последовательность ФОРТ-понятий (ФОРТ-программу) $w \in \Delta^*$ <u>корректной</u>, если $s(w) \neq 0$, и <u>замкнутой</u>, если $s(w) = 1$.

Понятия, из которых состоит корректная ФОРТ-программа, согласуются по типам входных и выходных параметров.

Корректная программа является замкнутой, если у нее нет ни входных ни выходных параметров.

<u>Пример 4</u>. Рассмотрим алфавит

$$A = \{ad, val\}$$

и множество понятий

$$\Delta = \{const, var, plus, fetch, dot, store\}.$$

Определим спецификации

s(const) = [--- val] ,

s(var) = [--- ad] ,

s(plus) = [val val --- val],

s(fetch) = [ad --- val],

s(dot) = [val ---],

s(store) = [val ad ---].

Программа "const var store" является замкнутой, программа "fetch dot" корректна, но не замкнута, и программа "const store" ошибочна.

Процесс проверки корректности ФОРТ-программы и вычисление ее спецификации по спецификациям базовых понятий можно автоматизировать. В приложении 2 данной работы приведена реализация подобного механизма для линейных ФОРТ-программ.

Глава 3. ПРИМЕНЕНИЕ АЛГЕБРЫ СПЕЦИФИКАЦИЙ В ФОРТ-ОРИЕНТИРОВАННЫХ СПТ

Основной целью данной работы является оснащение ФОРТ-ориентированных СПТ средствами отладки описаний исходных языков. В настоящей главе изучаются проблемы корректности описания перевода с исходного языка на язык семантических понятий. Определяется корректность СУ-схемы, выходным языком которой является ФОРТ. Доказывается основная теорема о корректности перевода. На базе этой теоремы создаются алгоритмы проверки корректности. Рассматривается практическое применение предложенного метода спецификации.

§ 3.I. Теорема о замкнутости объектных программ

В области построения транслирующих систем, переводящих с исходного языка на язык ФОРТ, возникает проблема корректности ФОРТ-программ, полученных в результате перевода. Данный раздел посвящен изучению корректности СУ-схемы.

Обозначим СУ-схему через $T = (N, \Sigma, \Delta, R, S)$ В дальнейшем предположим, что входная грамматика G_1 схемы T является приведенной (см. [25], стр. I75), и рассмотрим только выходную грамматику

$G_2 = (N, \Delta, P, S)$.

Выходной язык определяется как множество

$$L_2 = \{ t \mid t \in \Delta^* \ \& \ S \Rightarrow^+ t \}.$$

Предположим, что выходные символы $\Pi \in \Delta$ имеют спецификации $s(\Pi)$, т.е.

$$s(\Delta) \subset \Phi \setminus \{0\}.$$

По комплекту $s(\Delta)$ определяется гомоморфизм

$s : \Delta^* \to \Phi$ (см. стр. 50).

СУ-схема T называется корректной, если любая выходная последовательность $t \in L_2$ является замкнутой, т.е. $s(t) = 1$. Замкнутость (не просто корректность) объектных программ требуется потому, что магазины ФОРТ-системы скрыты от пользователя конкретного транслятора и не могут служить средством передачи параметров объектной программе.

Построим систему неравенств I(T, s) следующим образом :

I) каждому нетерминалу $A \in N$ ставим в соответствие неизвестную $Z(A) \in \Phi$,

2) заменим правила выходной грамматики на неравенства - из правила вида $A \to X_1 \dots X_k$ построим неравенство

$Z(A) \leqslant Y_1 \ldots Y_k$, где $Y_i = s(X_i)$, если $X_i \in \Delta$ и
$Y_i = Z(X_i)$, если $X_i \in N$.

Если правая часть правила пуста, то положим $Z(A) \leqslant 1$.

3) добавим неравенство $1 \leqslant Z(S)$, где S - аксиома схемы T.

Определим для нетерминальных символов выходной грамматики следующие вспомогательные множества :

$$C(A) = \{ (u, v) \in \Delta^* \times \Delta^* \mid S \Rightarrow^* uAv \},$$
$$L(A) = \{ w \in \Delta^* \mid A \Rightarrow^+ w \}.$$

Теорема 5 (основная теорема). Следующие утверждения эквивалентны :

I) все выходные последовательности СУ-схемы T замкнуты,

2) система неравенств $I(T, s)$ разрешима,

3) для каждого нетерминала $A \in N$ существует велечина $m(A) = \sup \{ (s(vu))^{-1} \mid (u, v) \in C(A) \}$ и выполняется неравенство $m(A) \leqslant \inf \{ s(w) \mid w \in L(A) \}$.

Доказательство.

Покажем сначала, что из утверждения 2 следует утверждение I. Обозначим полученное из решения системы $I(T, s)$ значение неизвестной $Z(A)$ через $s(A)$ и расширим таким образом отображение s на множество $(N \cup \Delta)^*$. Рассмотрим произвольный вывод $S \Rightarrow^* uAv \Rightarrow u\alpha v \Rightarrow^* t$, где $t \in L_2$, $(u, v) \in C(A)$ и $A \to \alpha \in P$ ($\alpha \in (N \cup \Delta)^*$, причем не исключено, что $\alpha = \Lambda$). Согласно утверждению 2 справедливо $s(A) \leqslant s(\alpha)$. В силу стабильности отношения $\leqslant$ на каждом шаге вывода имеет место неравенство $s(uAv) \leqslant s(u\alpha v)$. Отсюда заключаем, что $s(S) \leqslant s(t)$, если только $S \Rightarrow^+ t$ и $t \in \Delta^*$, т.е. для всех выходных последовательностей схемы T. Теперь ис-

пользуем неравенство $\mathbb{1} \leqslant s(S)$ и получим, что $\mathbb{1} \leqslant s(t)$ для всех $t \in L_2$. Согласно предложению I справедливо $s(t) = \mathbb{1}$. Первая часть доказательства завершена.

Далее предполагаем, что $S \Rightarrow^* uAv \Rightarrow^+ uwv$ является произвольным выводом, причем $s(uwv) = \mathbb{1}$. Тогда, согласно пунктам I и 3 теоремы 3 и пункту 3 теоремы 2, справедливо неравенство $(s(vu))^{-1} \leqslant s(w)$. Зафиксируем какую-либо пару $(u', v') \in C(A)$. Для всех $w \in L(A)$ выполняется соотношение $(s(v'u'))^{-1} \leqslant s(w)$, причем $(s(v'u'))^{-1} \neq 0$, так как $(s(u'))^{-1} \in L$ и $(s(v'))^{-1} \neq 0$ (пункт 5 теоремы 2). Согласно пункту 2 теоремы 4 и пункту 2 предложения 4 существует хотя бы один элемент $w' \in L(A)$ такой, что выполняется $s(w') = \inf \{ s(w) \mid w \in L(A) \}$. Величина $s(w')$ является верхней границей множества $\{ (s(vu))^{-1} \mid (u,v) \in C(A) \}$. Из пункта I предложения 4 следует, что существует верхняя грань $m(A)$. Так как $(s(v'u'))^{-1} \leqslant m(A)$, с одной стороны, и $(s(v'u'))^{-1} \leqslant s(w')$, с другой стороны, то $m(A)$ и $s(w')$ сравнимы согласно теореме 4. Если при этом не выполняется неравенство $m(A) \leqslant s(w')$, то по определению верхней грани и по теореме 3 существует пара $(u'', v'') \in C(A)$ такая, что $s(u''w'v'') \neq \mathbb{1}$. Вторая часть доказательства (из утверждения I следует утверждение 3) завершена.

Определим $s(A) = \inf \{ s(w) \mid w \in L(A) \}$ для всех нетерминалов $A \in N$. Покажем, что комплект величин $\{ s(A) \mid A \in N \}$ является решением системы $I(T, s)$. Если правило $A \to \alpha \in P$ имеет правую часть $\alpha \in L(A)$, то выполняется соотношение $s(A) \leqslant s(\alpha)$. Предположим теперь, что α имеет вид $z_0 B_1 z_1 \ldots z_{p-1} B_p z_p$, где $z_i \in \Delta^*$ и $B_i \in N$.

Рассмотрим вывод

$S \Rightarrow^* uAv \Rightarrow uz_0B_1z_1 \dots z_{p-1}B_pz_pv \Rightarrow^+ uz_0w_1z_1 \dots z_{p-1}w_pz_pv$,

в котором $s(B_i) = s(w_i)$, т.е. слова w_i являются "минимальными" в множествах $L(B_i)$. Пусть $w = z_0w_1z_1 \dots z_{p-1}w_pz_p$. Известно, что $w \in L(A)$. Следовательно, $s(A) \leqslant s(w)$. С другой стороны, $s(\alpha) = s(w)$ в силу равенств $s(w_i) = s(B_i)$. Таким образом $s(A) \leqslant s(\alpha)$ для всех правил выходной грамматики. Из того, что $S \Rightarrow^* S$, следует, что $(\Lambda,\Lambda) \in C(S)$. Поэтому имеет место $(11)^{-1} \leqslant s(w') = s(S)$ (слово w' является "минимальным" в множестве $L(S)$). Теорема 5 доказана.

Пример 5. Используем множество понятий примера 4 и рассмотрим выходную грамматику СУ-схемы T

⟨S⟩ → const ⟨A⟩ dot

⟨A⟩ → ⟨B⟩ const

⟨A⟩ → var fetch dot

⟨B⟩ → dot ⟨A⟩

Соответствующая система неравенств I(T,s) имеет вид

$1 \leqslant S$

$S \leqslant [\ \text{---}\ \text{val}]\ A\ [\text{val}\ \text{---}]$

$A \leqslant B\ [\ \text{---}\ \text{val}\ ]$

$A \leqslant [\ \text{---}\ ]$

$B \leqslant [\text{val}\ \text{---}\ ]\ A$

Оказывается, что эта система неразрешима. Например программа

" const dot dot var fetch dot const const dot ",

которая выводима в данной грамматике, не является замкнутой, а имеет спецификацию [val --- val].

Множество L(<A>) состоит из таких слов w , для которых $s(w) \neq 0$, причем все s(w) попарно сравнимы. В то же время L(<A>)является счетным множеством и не содержит минимального элемента, т.е.

$$\inf \{ s(w) \mid w \in L(\langle A \rangle) \} = 0.$$

Именно поэтому в теореме 5 (утверждение 3) нельзя использовать минимум вместо нижней грани.

Проблемы, связанные с описанием магазинно-ориентированных языков посредством КС-грамматик, изучены в работе [36]. Основная теорема этой работы (теорема 28) может в терминах алгебры спецификаций сформулироваться следующим образом :

Если $|A| = 1$ (т.е. имеется в точности один тип), то по (выходной) грамматике и по спецификациям терминальных символов можно установить, порождает ли эта грамматика безконечное множество таких последовательностей, спецификации которых принадлежат подполугруппе R (т.е. множество программ, не имеющих входных параметров).

Идея создания аппарата для проверки корректности описаний исходных языков в ФОРТ-ориентированных СПТ взята с работы Кнута ([46]), в которой подобный аппарат разработан для атрибутных описаний.

§ 3.2. Алгоритмы, проверяющие корректность перевода

В данном разделе приводятся алгоритмы, позволяющие по СУ-схеме T и комплекту спецификаций семантических понятий $s(\Delta)$ установить их согласованность. Если СУ-схема некорректна относительно комплекта спецификаций, то дается дополнительная информация о причинах несогласованности. Обработка исходных данных делится на три этапа :

I) вычисление приближенного решения системы $I(T,s)$,

2) вычисление нижних границ компонент решения,

3) уточнение решения.

В алгоритмах будем писать $s < t$, если $s \leqslant t$ и $s \neq t$.

Алгоритм I. Вычисление первого приближения решения системы $I(T,s)$.

Вход :

$G_2 = (N, \Delta, P, S)$ - выходная грамматика СУ-схемы T (предполагается, что входная грамматика G_1 является приведенной),

$s(\Delta) \subset \Phi \setminus \{0\}$ - спецификации выходных символов.

Выход :

$\{1(A) \mid A \in N\}$ - приближенное решение системы $I(T,s)$ или ответ "Система неразрешима".

Локальные объекты:

W - множество символов, для которых вычислены спецификации,

J - множество порядковых номеров неиспользованных данным алгоритмом правил грамматики,

q - переменная типа спецификации.

Метод :

I. Положить $W := \Delta$ и $J := \{1, \ldots, |P|\}$.

2. Выбрать такое правило $P_j = A_j \Rightarrow \alpha_j \in P$, при котором $j \in J$ и $\alpha_j \in W^*$. Положить $J := J \setminus \{j\}$.

3. Если $\alpha_j = \Lambda$, то положить $q := 1$ и перейти к шагу 6.

4. Если $\alpha_j = X_1 \ldots X_k$, то вычислить $q := Y_1 \ldots Y_k$, где
$Y_i = s(X_i)$, если $X_i \in \Delta$ и
$Y_i = 1(X_i)$, если $X_i \in N$.

5. Если $q = 0$, то работа алгоритма прерывается и выдается сообщение "Система неразрешима" (дополнительно выдается номер правила j).

6. Если $A_j \in W$, то

6.1. если q и $1(A)$ несравнимы, то работа алгоритма прерывается и выдается сообщение "Система неразрешима" (дополнительно выдается номер правила j, спецификации q и $1(A_j)$),

6.2. если $q < 1(A_j)$, то положить $1(A_j) := q$ и перейти к шагу 8.

7. Если $A_j \in N \setminus W$, то положить $1(A_j) := q$ и $W := W \cup \{A_j\}$.

8. Если $J \neq \varnothing$ (т.е. имеются неиспользованные правила), то возвратиться к шагу 2.

9. Если $1(S) \neq 1$, то работа алгоритма прерывается и выдается сообщение "Система неразрешима" (дополнительно выдается спецификация $1(S)$).

I0. Выход.

Если система $I(T, s)$ имеет решение $\{s(A) \mid A \in N\}$,

то после работы алгоритма I справедливо условие $s(A) \leqslant 1(A)$ для всех $A \in N$. Именно так следует трактовать выражение "приближенное решение".

Алгоритм 2. Вычисление нижних границ компонент решения системы $I(T, s)$.

Вход :

$G_2 = (N, \Delta, P, S)$,

$s(\Delta) \subset \Phi \setminus \{0\}$,

$\{1(A) \mid A \in N\}$ - приближенное решение.

Выход :

$\{c(A) \mid A \in N\}$ - комплект нижних границ

или ответ "Система неразрешима".

Локальные объекты :

W - множество нетерминалов, для которых вычислены нижние границы соответствующих компонент решения,

J - множество порядковых номеров неиспользованных данным алгоритмом правил грамматики,

r, t, q - переменные типа спецификации.

Метод :

I. Положить $c(S) := 1$, $W := \{S\}$ и $J := \{1, \ldots, |P|\}$.

2. Выбрать такое правило $p_j = A_j \to \alpha_j \in P$, при котором $j \in J$ и $A_j \in W$. Положить $J := J \setminus \{j\}$.

3. Если $\alpha_j = \Lambda$ то перейти к шагу 6.

4. Если $\alpha_j = X_1 \ldots X_k$, то повторить шаг 5 для всех $i = 1, \ldots, k$ и перейти к шагу 6.

В дальнейшем используем следующие обозначения :

$[a \text{ --- } b] = c(A_j)$,

$Y_n = s(X_n)$, если $X_n \in \Delta$ и

$Y_n = l(X_n)$, если $X_n \in N$.

5. Если $X_i \in N$, то

5.1. если $i = 1$, то положить $r := [\text{ --- } a]$,

в противном случае положить $r := [\text{ --- } a] Y_1 \ldots Y_{i-1}$,

5.2. если $i = k$, то положить $t := [b \text{ --- }]$,

в противном случае положить $t := Y_{i+1} \ldots Y_k [b \text{ --- }]$,

5.3. если $r \in R$ и $t \in L$ (см. теорема 2), то положить $q := (tr)^{-1}$ и перейти к шагу 5.5,

5.4. если $r \in \Phi \setminus R$ или $t \in \Phi \setminus L$, то работа алгоритма прерывается и выдается сообщение "Система неразрешима" (дополнительно выдаются номер правила j, номер символа i, спецификации r и t),

5.5. если $X_i \in W$, то

5.5.1. если существует $\sup\{c(X_i), q\}$, то положить $c(X_i) := \sup\{c(X_i), q\}$ и перейти к шагу 5.7,

5.5.2. если такой верхней грани не существует, то работа алгоритма прерывается и выдается сообщение "Система неразрешима" (дополнительно выдаются номер правила j , номер символа i , спецификации q и $c(X_i)$),

5.6. если $X_i \in N \setminus W$, то положить $c(X_i) := q$ и $W := W \cup \{X_i\}$,

5.7. если $l(X_i) < c(X_i)$, то работа алгоритма прерывается и выдается сообщение "Система неразрешима" (дополнительно выдаются номер правила j , номер сим-

вола i , спецификации $c(X_i)$ и $l(X_i)$),

5.8. Конец шага 5.

6. Если $J \neq \varnothing$ (т.е. имеются неиспользованные правила), то возвратиться к шагу 2.

7. Выход.

Если система $I(T, s)$ имеет решение $\{s(A) \mid A \in N\}$, то после работы алгоритма 2 справедливо условие $c(A) \leqslant s(A)$ для всех $A \in N$. Именно так следует трактовать выражение "нижняя граница компоненты решения".

Алгоритм 3. Уточнение решения системы $I(T, s)$.

Вход :

$G_2 = (N, \Delta, P, S)$,

$s(\Delta) \subset \Phi \setminus \{0\}$,

$\{l(A) \mid A \in N\}$- приближенное решение,

$\{c(A) \mid A \in N\}$- комплект нижних границ.

Выход :

$\{l(A) \mid A \in N\}$ - решение системы $I(T, s)$ или ответ "Система неразрешима".

Локальные объекты :

q - переменная типа спецификации.

Метод :

I. Если комплект $\{l(A) \mid A \in N\}$ является решением, то выход, в противном случае повторить шаг 2 для всех $j = 1, \ldots, |P|$ и возвратиться к шагу I.

2. Пусть очередное правило $P_j \in P$ имеет вид $A_j \to \alpha_j$.

2.1. Если $\alpha_j \in \Delta^*$, то конец шага 2.

2.2. Если $\alpha_j = X_1 \dots X_k \in (N \cup \Delta)^+$, то вычислить

$q := Y_1 \dots Y_k$, где

$Y_i = s(X_i)$, если $X_i \in \Delta$ и

$Y_i = 1(X_i)$, если $X_i \in N$.

2.3. Если $q = 0$ или q и $1(A_j)$ несравнимы, то работа алгоритма прерывается и выдается сообщение "Система неразрешима" (дополнительно выдаются номер правила j , спецификации q и $1(A_j)$).

2.4. Если $q < c(A_j)$, то работа алгоритма прерывается и выдается сообщение "Система неразрешима" (дополнительно выдаются номер правила j , спецификации $c(A_j)$ и q).

2.5. Если $q < 1(A_j)$, то положить $1(A_j) := q$.

2.6. Конец шага 2.

Справедливость этих алгоритмов следует из второй части доказательства теоремы 5, причем необходимы некоторые дополнительные замечания.

В алгоритме I приведенность входной грамматики позволяет выбрать очередное правило на шаге 2, причем при завершении работы алгоритма имеет место $W = \Delta \cup N$, т.е. компоненты $1(A)$ вычислены для всех $A \in N$. Вычисление минимума на шаге 6.2 алгоритма I ускоряет процесс решения (величины $1(A)$ будут более точными приближениями). Если непосредственная проверка показывает, что $\{1(A) \mid A \in N\}$ является решением системы, то, естественно, можно опускать последующие этапы.

В алгоритме 2 выбор очередного правила на шаге 2 также обеспечивается приведенностью входной грамматики. При завершении работы имеет место $W = N$, т.е. границы $c(A)$ вычислены для всех $A \in N$. Шаг 5.5.I опять-таки ускоряет процесс решения - величины $c(A)$ будут более точными нижними границами.

В алгоритме 3 вычисление минимума (шаг 2.5) обязательно (в этом и состоится уточнение решения). Если же процесс уточнения "зацикливается" то шаг 2.4 гарантирует завершение работы.

§ 3.3. Практическое применение метода спецификации

В предыдущих разделах предполагалось, что семантические понятия описаны на языке формальных спецификаций. Рассмотрим теперь некоторые вопросы, связанные с разработкой таких спецификаций.

Оказывается, что описание одних и тех же понятий может в разных ситуациях быть разным. В качестве примера приведем конструкции управления языка ФОРТ (реализация этих конструкций излагается в работе [56]).

Прикладной программист использует в своей работе спецификации, описывающие интерфейс управляющих слов во время выполнения программы :

```
IF      [Bool ---]
ENDIF   [---]
ELSE    [---]
BEGIN   [---]
AGAIN   [---]
WHILE   [Bool ---]
REPEAT  [---]
```

По этим спецификациям можно сказать, что во время выполнения программы понятия IF и WHILE используют верхний элемент магазина как булевское значение. Остальные понятия на состояние магазина не влияют.

Системный программист знает, что понятия, связанные с управлением, работают во время компиляции ФОРТ-программы. Вышеуказанные спецификации отражают лишь то, что понятия IF и WHILE компилируют в программу понятие OBRANCH , которое

во время выполнения организует переход по условию. "Истинные" спецификации конструкций управления, т.е. спецификации, характеризующие использование магазина во время компиляции, пишутся в виде (см. [56]) :

```
IF    [ ---  ad 2 ]
ENDIF [ ad 2 --- ]
ELSE  [ ad 2 ---  ad 2 ]
BEGIN [ ---  ad 1 ]
AGAIN [ ad 1 --- ]
WHILE [ ---  ad 4 ]
REPEAT [ ad 1  ad 4  --- ]
```

В данном случае в качестве "типа" используются и конкретные числа.

Выбор имен типов зависит от проблемной области. Если же передаваемые параметры плохо классифицируются по типам, то можно, например, иметь только один тип - 16Bit. Несмотря на малую информативность спецификаций в таком случае, предложенные средства все-таки позволяют выявить грубые ошибки.

Чем точнее задаются спецификации, тем больше ошибок можно найти автоматически. Полезно иметь несколько спецификаций одного и того же понятия - для времени выполнения, для времени компиляции, для магазина возвратов и т.п. Таким путем можно выявить ошибки использования этого понятия в разных ситуациях.

Трудным вопросом является описание динамических свойств ФОРТ-понятий. Неформальная спецификация может, например, иметь вид :

```
---  pfa len true  /  false
```

т.е. в одном случае (в зависимости от конкретных данных) понятие оставляет в магазине параметры pfa, len и значение "истина", в другом случае только значение "ложь". Предложенный в данной работе формализм пока еще не охватывает такие аспекты.

Многие проблемы, связанные с описанием свойств ФОРТ-программы, являются открытыми. Автор надеется, что методы, предложенные в данной работе, оказываются полезными в дальнейших исследованиях.

ЗАКЛЮЧЕНИЕ

В диссертации получены следующие основные результаты :

I. Разработан метод спецификации ФОРТ-понятий, который согласуется с общепринятым способом описания входных и выходных параметров ФОРТ-программы. Создан соответствующий формальный аппарат - алгебра спецификаций языка ФОРТ.

2. Определено понятие корректности ФОРТ-программы относительно согласованности типов передаваемых через магазин параметров и реализованы средства для статической проверки такой корректности на основе алгебры спецификаций.

3. Проанализированы существующие методы описания и реализации языков программирования в ФОРТ-ориентированных СПТ. Изучены проблемы корректности описания перевода с исходного языка на язык семантических понятий.

4. Теоретическим исследованием свойств алгебры спецификаций проблема проверки корректности описания перевода приведена к проблеме разрешимости некоторой системы неравенств.

5. В виде конкретных алгоритмов реализован метод решения этой системы, позволяющий установить корректность описаний реализуемых языков в ФОРТ-ориентированных СПТ.

Полученные результаты позволяют повысить надежность системы программирования ФОРТ и созданных на ее основе инструментальных систем, особенно СПГ. Методы предложенные в диссертации, оказались целесообразными и практичными при автоматизированной реализации языков программирования в СПГ ТАРТУ, а также при разработке программного обеспечения на языке ФОРТ. Предложенная алгебра спецификаций охватывает пока только один из существенных аспектов ФОРТ-программы - обмен информацией через магазин. Поэтому актуальной задачей будущего является усовершенствование метода спецификации. Эта задача, естественно, требует дальнейших теоретических исследований и может быть успешно решена лишь при наличии богатого практического опыта.

Л И Т Е Р А Т У Р А

I. Агамирзян И.Р. Реализация ФОРТ-системы на БЭСМ-6. - В сб.: Тезисы докладов IV Всесоюзной конференции "ДИАЛОГ Человек - ЭВМ", ч.I. - Киев: ИК АН УССР, I985, с. 26-27.

2. Агамирзян И.Р. Система технологической поддержки разработки трансляторов "ШАГ". Подсистема построения анализаторов. - Алгоритмы небесной механики вып. 46. - Ленинград: ИТА АН СССР, I986, с. I-53.

3. Агамирзян И.Р. Средства автоматизации и технология разработки трансляторов и диалоговых систем. - Дисс. канд. физ.-мат. наук. - Ленинград, I986. - I42 с.

4. Астановский А., Ломунов В. Процессор, ориентированный на язык ФОРТ. - В сб.: Программирование микропроцессорной техники. - Таллин: ИК АН ЭССР, I984, с. 50-58.

5. Баранов С.Н., Кириллин В.А., Ноздрунов Н.Р. Реализация языка ФОРТ на дисплейном комплексе ЕС-7970. - В сб.: Программирование микропроцессорной техники. - Таллин : ИК АН ЭССР, I984, с. 4I-49.

6. Брусенцов Н.П., Златкус Г.В., Руднев И.А. ДССП - диалоговая система структурного программирования. - В кн.: Программное оснащение микрокомпьютеров. - Москва : МГУ, I982, с. II-40.

7. Вооглайд А.О., Томбак М.О. Об одной системе построения трансляторов с LR(K) семантикой. - Программирование, №5, I976, с. 28-38.

8. Вяйнасте Р.В., Юрик А.Э. Реализация интерпретирующей стековой системы FORTH на ЭВМ СМ-4. - Информационный сборник / ЦНИИТЭИ приборостроения, ТС-I2, вып. 4 : Программное обеспечение мини- и микроЭВМ семейства СМ ЭВМ. - Москва, I984, с. 57-58.

9. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. - Киев : Наукова думка, I978. - 320 с.

I0. Демин С.П., Литвиненко А.Н. Средства разработки синтаксически управляемых программ в системе программирования ФОРТ. - Тезисы докл. Всесоюзного семинара по языку ФОРТ, Тарту, 26-29 мая I986. - Ростов-на-Дону : ВЦ РГУ, I986. - 2 с.

II. Кириллин В.А. Инструментальная система разработки языковых средств микропроцессорной техники. - Дисс. канд. физ.-мат. наук.- Ленинград, I985. - 90 с.

I2. Кириллин В.А., Клубович А.А., Ноздрунов Н.Р. Система разработки программного обеспечения ФОРТ-7970. - В кн.: Всесоюзный семинар "Промышленная технология создания и применения программных средств в организационном управлении и НИОКР" : Тезисы докл. - Свердловск, I984, с. I67-I68.

I3. Лезин Г.В., Болдырев А.Ю. Язык FORTH для ПЭКВМ "Искра-226". - В сб. : Индивидуальные диалоговые системы на базе микро-ЭВМ (персональные компьютеры). Диалог-84--микро : Тезисы докл. - Ленинград : Наука, I984,с.68-7I.

I4. Ляпин Е.С. Полугруппы. - Москва : Физматгиз, I960. - 592 с.

I5. Пёйал Я.Р. Алгебра спецификаций языка ФОРТ и ее применение в СПТ. - В сб.: III Всесоюзная конференция "Автоматизация производства систем программирования":Тезисы докл. - Таллин, I986, с. 40-42.

I6. Пёйал Я.Р. Опыт применения FORTH -ориентированной СПТ для СМ ЭВМ. - Информационный сборник / ЦНИИТЭИ приборостроения, ТС-I2, вып. 4 : Программное обеспечение мини- и микроЭВМ семейства СМ ЭВМ. - Москва, I984, с.57.

I7. Пёйал Я.Р. О свойствах спецификаций языка ФОРТ. - Труды ВЦ Тартуского госунив. вып. 54. - Тарту, I986, с.68-84.

I8. Пёйал Я.Р., Соо В.К., Томбак М.О. Использование расширяемого языка в СПТ. - Труды ВЦ Тартуского госунив. вып.52. - Тарту, I985, с. 39-54.

I9. Пёйал Я.Р., Соо В.К., Томбак М.О. СПТ для микро-ЭВМ. - В сб.: Индивидуальные диалоговые системы на базе микро-ЭВМ (персональные компьютеры). Диалог-84-микро: Тезисы докл. - Ленинград: Наука, I984, с. I02-I04.

20. Руднев С.Б., Четвернин В.А. Переносимая рабочая смесь - ядро программного обеспечения персональной ЭВМ. - В сб.: Персональные ЭВМ в задачах информатики, ред. Ершов А.П. - Новосибирск : ВЦ СО АН СССР, I984, с. 58-72.

2I. Семенов Ю.А. Язык четвертого поколения FORTH. - Ин-т теор. и эксп. физ. Препр. № 30. - Москва, I984. - 50 с.

22. Томбак М. Об устранении конфликтов предшествования. - Труды ВЦ Тартуского госунив. вып. 37. - Тарту, I976, с. 60-9I.

23. Томбак М. Описание семантики языков программирования при помощи расширяемого языка. - В сб.: Теоретические и прикладные вопросы математики, ч. II : Тезисы докл. - Тарту : ТГУ, 1985, с. 149-150.

24. Хротко Г.М. Языки программирования высокого уровня для микро-ЭВМ. - М., 1985. - 91 с.

25. Aho A.V., Ullman J.D. The theory of parsing, translation and compiling. vol.1 : Parsing. - Englewood Cliffs, 1972. - Русский перевод : Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Т.I : Синтаксический анализ. - М.: Мир, 1978. - 612 с.

26. Aho A.V., Ullman J.D. The theory of parsing, translation and compiling. vol.2 : Compiling. - Englewood Cliffs, 1973. - Русский перевод : Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. т.2 : Компиляция. - М.: Мир, 1978. - 487 с.

27. Barnhart J. Forth and the Motorola 68000. - Dr. Dobb's J. v. 8, n. 9, 1983, p. 18-26.

28. Bell J.R. Threaded code. - CACM v. 16, n. 6, 1973, p. 370-372.

29. Bryce H. Advanced computer languages. - Electron. Des. v. 32, n. 9, 1984, p. 229-248.

30. Clifford A.H., Preston G.B. The algebraic theory of semigroups. vol. 1. - Rhode Island, 1964. - Русский перевод : Клиффорд А., Престон Г. Алгебраическая теория полугрупп. т.I. - М.: Мир, 1972. - 285 с.

31. Clifford A.H., Preston G.B. The algebraic theory of semigroups. vol. 2. - Rhode Island, 1967. - Русский

перевод : Клиффорд А., Престон Г. Алгебраическая теория полугрупп. т.2. - М.: Мир, 1972. - 422 с.

32. Deransart P., Jourdan M., Lorho B. A survey on attribute grammars. Part 2 : Review of existing systems. - Report n. 510, INRIA, Rocquencourt, 1986. - 140 pp.
33. Dewar R.B.K., McCann A.P. Macro Spitbol - a Snobol 4 compiler. - Software : Pract. and Exper. v. 7, n. 1, 1977, p. 95-113.
34. Feierbach G. Forth - the language of machine independence. - Comput. Des. v. 20, n. 6, 1981, p. 117-121.
35. FORTH-83 Standard. - A publication of the FORTH Standards Team, August 1983. - 82 pp.
36. Goodwin D.T. The computability of stack non-underflow. - Comput. J. v. 28, n. 3, 1985, p. 257-263.
37. Graham S.L. Advances in compiler technology. - COMPSAC--83 : Proc. IEEE 7th Conf. - Silver Spring, 1983, p. 66-75.
38. Gray J.N., Harrison M.A. On the covering and reduction problems for context-free grammars. - JACM v. 19, n. 4, 1972, p. 675-698.
39. Greene R.L. Faster FORTH. Reducing overhead in threaded interpretive languages. - BYTE, June, 1984, p. 127-129, 418-424.
40. Harris K. FORTH extensibility or how to write a compiler in 25 words or less. - BYTE, August, 1980, p. 164-184.
41. Haydon G.B. Elements of a FORTH data base design. - Dr. Dobb's J. v. 6, n. 9, 1981, p. 42-56.

42. Helliwell A.M. Implementation of a highly portable Pascal interpreter using indirect threaded code techniques. - IUCC Bulletin v.5, n. 3, 1983, p. 124-127.

43. Hicks S.M. FORTH's forte is tighter programming. - Electronics, March 15, 1979. - Русский перевод : Хикс С.М. Лаконичность программ - основное достоинство языка ФОРТ. - Электроника, №6, 1979, с. 44-50.

44. Irons E.T. A syntax directed compiler for ALGOL 60. - CACM v.4, n. 1, 1961, p. 51-55.

45. James J.S. What is FORTH? A tutorial introduction. - BYTE, August, 1980, p. 100-126.

46. Knuth D.E. Semantics of context-free languages. - Math. Syst. Theory v.2, n.2, 1968, p.127-146. - Русский перевод : Кнут Д.Е. Семантика контекстно-свободных языков. - В сб. : Семантика языков программирования, ред. Курочкин В.М. - М.: Мир, 1980, с. 137-161.

47. Loeliger R.G. Threaded interpretive languages, their design and implementation. Second printing. - Peterborough : BYTE Books, 1981. - 322 pp.

48. MacIntyre F. FORTH : the optimum language for microcomputers. - Int. Lab., March, 1985, p. 12-18.

49. Mayer-Lindenberg F. Some applications of the Fifth programming environment. - Microprocessing and Microprogramming v. 14, n. 3-4, 1984, p. 169-172.

50. Mayer-Lindenberg F. Three interactive programming languages for microcomputers. - Microprocessing and Microprogramming v. 13, n. 1, 1984, p. 31-40.

51. McCabe C.K. FORTH Fundamentals, vol.1 : Language usage.

- Beaverton : dilithium Press, 1983. - 129 pp.

52. McCabe C.K. FORTH Fundamentals. vol.2 : Language glossary. - Beaverton : dilithium Press, 1983. - 134 pp.

53. Meinzer K. IPS, an unorthodox high level language. - BYTE, January, 1979, p. 146-159.

54. Moore C.H. FORTH : a new way to program a minicomputer. - Astron. Astrophys. Suppl., 15, 1974, p. 497-511.

55. Moore C.H. The evolution of FORTH, an unusual language. - BYTE, August, 1980, p. 76-90.

56. Ragsdale W.F. Fig-FORTH installation manual, glossary, model. - San Carlos : Forth Interest Group, Aug. 1980. - 54 pp.

57. Rather E.D., Moore C.H. The FORTH approach to operating systems. - Proc. ACM, October, 1976, p. 233-239.

58. Ritter T., Walker G. Varieties of threaded code for language implementation. - BYTE, Sept., 1980, p.206-227.

59. Sachs J.M., Burns S.K. STOIC, an interactive programming system for dedicated computing. - Software : Pract. and Exper. v. 13, n. 1, 1983, p. 1-16.

60. Sweet B. Adapting FORTH to a multi-user world. - Comput. Des. v. 22, n. 4, 1983, p. 111-115.

61. Tello E. PolyFORTH and PC/FORTH. Two FORTH development systems for the IBM PC. - BYTE, Nov., 1984, p. 303-313.

62. Tesler L.G. Programming languages. - Sci. Amer. v.251, n. 3, 1984, p. 58-66.

63. Ting C.H. Formal definition of FORTH. - Dr. Dobb's J. v. 7, n. 2, 1982, p. 19-20, 22, 24, 27-30.

64. Vickery C., Berkley G.D. FORTH as the machine : a computer for software engineering. - 4th Jerusalem Conf. Inf. Technol. (JCIT). - Silver Spring, 1984, p. 370-375.

Приложение I. Реализация алгебраических операций над ФОРТ-спецификациями

Программы обработки спецификаций описываются в стиле словаря. Такой метод является общепринятым в языке ФОРТ.

SSYMB — в форме: SSYMB ‹name›
Определение имени типа (символа алфавита A).

S[— в форме: S[‹left-side› --- ‹right-side›]
--- ‹s-val›
Ненулевая спецификация.
‹left-side› , ‹right-side› $\in$ A*

SZERO — --- ‹s-val›
Нулевая спецификация.

SCONSTANT — в форме: ‹s-val› SCONSTANT ‹name›
константа типа спецификации.
‹name› : --- ‹s-val›

SVARIABLE — в форме: ‹s-val› SVARIABLE ‹name›
Переменная типа спецификации.
‹name› : --- ‹s-ad›

SARRAY — в форме: ‹#elements› SARRAY ‹name›
Одномерный массив спецификаций.
‹name› : ‹index› --- ‹s-ad›

S@ ⟨s-ad⟩ --- ⟨s-val⟩

"S-fetch"

S! ⟨s-val⟩ ⟨s-ad⟩ ---

"S-store"

S. ⟨s-val⟩ ---

"S-dot" - вывод на экран.

S? ⟨s-ad⟩ ---

S@ S.

S* ⟨s-val⟩ ⟨s-val⟩ --- ⟨s-val⟩

Произведение спецификаций.

SZERO= ⟨s-val⟩ --- flag

true, если ⟨s-val⟩ = 0

S< ⟨s-val⟩ ⟨s-val⟩ --- flag

S> ⟨s-val⟩ ⟨s-val⟩ --- flag

S= ⟨s-val⟩ ⟨s-val⟩ --- flag

S>= ⟨s-val⟩ ⟨s-val⟩ --- flag

S<= ⟨s-val⟩ ⟨s-val⟩ --- flag

S+REL ⟨s-val⟩ ⟨s-val⟩ --- flag

true, если элементы сравнимы

S-REL ⟨s-val⟩ ⟨s-val⟩ --- flag

true, если элементы несравнимы

SINV ⟨s-val⟩ --- ⟨s-val⟩

Инверсный элемент.

SLPONLY ⟨s-val⟩ --- ⟨s-val⟩

Левая часть: [a --- b] → [a ---]

SRPONLY ⟨s-val⟩ --- ⟨s-val⟩

Правая часть: [a --- b] → [--- b]

SLPLEN ⟨s-val⟩ --- int

Длина левой части / -I, если ⟨s-val⟩=0

SRPLEN ⟨s-val⟩ --- int

Длина правой части / -I

SINF ⟨s-val⟩ ⟨s-val⟩ --- ⟨s-val⟩

Нижняя грань двухэлементного подмножества.

-SSUP ⟨s-val⟩ ⟨s-val⟩ --- ⟨s-val⟩ true / false

Верхняя грань, если она существует (flag = true).

```
SCR # 90

( SPECIFICATION HANDLING -- TARTU 1986
                          JAANUS  PQIAL )
BASE @ DECIMAL

( WORDS TO OPERATE WITH SYMBOLS )

0 VARIABLE #SYMB ( # SYMBOLS IN USE )
: SSCONS
( DEF: CODE SSCONS NN ; USE: --- CODE )
DUP 128 U< IF   <BUILDS ,   ELSE CR
." IMPOSSIBLE CODE " . QUIT ENDIF
DOES> @ ;
0 SSCONS SSPATT ( PATTERN )
: SSYMB? ( PFA --- FLAG )
@ ' SSPATT @ = ;
128 ARRAY SSNFAS ( NAME FIELD AD.-S )
42 CONSTANT WLD

: SSINIT ( ---  ; USERS WORD TO START )
128 0 DO 0 I SSNFAS ! LOOP 0 #SYMB ! ;

SSINIT
 -->

SCR # 91

( SPECIFICATION HANDLING
USERS LEVEL TO OPERATE WITH SYMBOLS   )
: SSYMB ( MAIN WORD TO CREATE SYMBOLS.
    DEF: SSYMB NN ; USE:  --- CODE    )
#SYMB @ 1+ DUP SSCONS #SYMB !
LATEST #SYMB @ SSNFAS ! ;
: AUXSYMB ( WORD TO DEFINE SYNONYMS.
 DEF: CODE AUXSYMB NN ; USE: --- CODE )
DUP 128 U< OVER SSNFAS @ 0= 0= AND IF
SSCONS  ELSE CR ." ILLEGAL AUX " . QUIT
ENDIF ;
: EXALTSYMB ( IN FORM:  EXALTSYMB NN
AUX-SYMBOL NN WILL BE TREATED AS MAIN )
-FIND IF DROP DUP SSYMB? IF DUP NFA SWAP
 CFA EXECUTE SSNFAS ! ELSE CR NFA ID.
." IS NOT A SYMBOL" QUIT ENDIF  ELSE CR
HERE COUNT TYPE ."  ??" QUIT ENDIF ;
: SS. ( CODE ---  ; SYMBOL OUTPUT )
DUP 0 > OVER 128 < AND IF SSNFAS @ DUP
IF ID. ELSE DROP CR ." SS OUTPUT ERROR"
QUIT ENDIF ELSE DUP 128 > OVER 160 < AND
 IF 31 AND 0 DO WLD EMIT LOOP SPACE ELSE
 ." ILLEGAL CODE " . QUIT ENDIF ENDIF ;
 -->
```

```
SCR # 92

( SPECIFICATION HANDLING WORDS )
16 CONSTANT SSIZE ( MAX SPECIF. SIZE )
: SCOUNTS ( AD --- AD+2 L1 L2 )
DUP 2+ SWAP C@ OVER 1 - C@ ;

: SZERO= ( AD --- FLAG )
SCOUNTS + 2+ SWAP DROP SSIZE > ;

: S. ( AD --- ; SPECIF. OUTPUT )
DUP SZERO= IF DROP ." SZERO " ELSE
SCOUNTS ." SI " ROT ROT DUP IF 0 DO DUP
C@ SS. 1+ LOOP ELSE DROP ENDIF ." --- "
SWAP DUP IF 0 DO DUP C@ SS. 1+ LOOP ELSE
 DROP ENDIF DROP ." J " ENDIF ;
0 VARIABLE SRSTFLAG
: SRSTEST ( AD --- )
DUP SCOUNTS >R + R> DUP IF 0 DO DUP C@
128 > IF 1 SRSTFLAG ! OVER SCOUNTS DROP
DUP IF 0 DO OVER C@ OVER C@ = IF 0
SRSTFLAG ! LEAVE ENDIF 1+ LOOP ELSE DROP
 ENDIF DROP SRSTFLAG @ IF DROP CR S.
." IS NOT CORRECT" QUIT ENDIF ENDIF 1+
LOOP ELSE DROP ENDIF DROP DROP ;
 -->
SCR # 93

( SPECIFICATION HANDLING WORDS )
HERE 255 C, 255 C, CONSTANT SZERO
: SWINCR? ( AD --- FLAG )
0 SWAP SCOUNTS + DUP IF 0 DO DUP C@ 128
> IF SWAP DROP 1 SWAP DUP C@ 32 + OVER
C! ENDIF 1+ LOOP ELSE DROP ENDIF DROP ;
0 VARIABLE SWRENOLD 0 VARIABLE SWRENNEW
: SWRENAME ( AD --- )
129 SWRENNEW ! DUP SCOUNTS DROP DUP >R
+ R> 0 DO 1 - DUP C@ 160 > IF DUP C@
SWRENOLD ! OVER SCOUNTS + 0 DO DUP C@
SWRENOLD @ = IF SWRENNEW @ OVER C!
ENDIF 1+ LOOP DROP 1 SWRENNEW +! ENDIF
LOOP DROP DROP ;
: SWNORM ( AD --- )
DUP SZERO= IF SZERO SWAP 2 CMOVE ELSE
DUP SRSTEST DUP SWINCR? IF SWRENAME ELSE
 DROP ENDIF ENDIF ;
: LONG= ( AD1 AD2 L --- FLAG )
>R >R >R 1 R> R> R> DUP IF 0 DO OVER C@
OVER C@ = 0= IF >R >R DROP 0 R> R> LEAVE
 ENDIF 1+ SWAP 1+ SWAP LOOP ELSE DROP
ENDIF DROP DROP ;
 -->
```

```
SCR # 94

( SPECIFICATION HANDLING WORDS )
0 VARIABLE SLDIFF
: SLESS? ( AD1 AD2 --- FLAG )
OVER C@ OVER C@ - SLDIFF ! OVER DUP C@
SWAP 2+ DUP ROT + SLDIFF @ LONG= IF
OVER 2+ SLDIFF @ + OVER DUP C@ SWAP 2+
SWAP LONG= IF OVER DUP C@ + 2+ SLDIFF @
+ OVER DUP DUP C@ + 2+ SWAP 1+ C@ LONG=
IF DROP DROP 1 ELSE DROP DROP 0 ENDIF
ELSE DROP DROP 0 ENDIF ELSE DROP DROP 0
ENDIF ;
: NZSREL ( AD1 AD2 --- REL )
OVER C@ OVER C@ = IF DUP C@ OVER 1+ C@
+ 2+ LONG= IF 4 ELSE 0 ENDIF ELSE OVER
C@ OVER C@ > IF SLESS? IF 1 ELSE 0
ENDIF ELSE SWAP SLESS? IF 2 ELSE 0
ENDIF ENDIF ENDIF ;
: NOWILD? ( AD --- FLAG )
1 SWAP SCOUNTS + DUP IF 0 DO DUP C@ 128
> IF >R DROP 0 R> LEAVE ENDIF 1+ LOOP
ELSE DROP ENDIF DROP ;
 -->


SCR # 95

( SPECIFICATION HANDLING WORDS )
: SREL ( AD1 AD2 --- REL )
OVER SZERO= IF DUP SZERO= IF DROP DROP
4 ELSE DROP DROP 1 ENDIF ELSE DUP SZERO=
 IF DROP DROP 2 ELSE OVER C@ OVER C@ -
>R OVER 1+ C@ OVER 1+ C@ - R> = IF OVER
C@ OVER C@ < IF OVER NOWILD? ELSE DUP
NOWILD? ENDIF IF
NZSREL ELSE DROP DROP 0 ENDIF ELSE DROP
DROP 0 ENDIF ENDIF ENDIF ;
: ALIGN ( N --- N ) 1+ 2 / 2 * ;
: SPTOHERE ( AD --- )
DUP SZERO= IF DROP SZERO HERE 2 ALLOT
2 CMOVE ELSE HERE OVER DUP C@ SWAP 1+
C@ + 2+ ALIGN DUP ALLOT CMOVE ENDIF ;
: SCONSTANT
<BUILDS SPTOHERE DOES> ;
: SVARIABLE
<BUILDS HERE 2+ , HERE SSIZE ALIGN DUP
ALLOT CMOVE DOES> ;
 -->
```

```
SCR # 96

( SPECIFICATION HANDLING WORDS )
32 CONSTANT #AUXSP
: SPARRAY
<BUILDS SSIZE * ALIGN ALLOT
DOES> SWAP SSIZE * + ;
#AUXSP SPARRAY SPAREA
0 VARIABLE SPUSED
: SPMEM ( --- AD )
SPUSED @ #AUXSP MOD SPAREA
1 SPUSED +! ;
0 VARIABLE SPCUR
0 VARIABLE SCANAREA 32 ALLOT
0 VARIABLE SCPTR
0 VARIABLE SCCEIL
0 VARIABLE SPMOFFSET
0 VARIABLE SPLRFLAG
: SSFIND ( NFA --- PFA/0 )
BEGIN SCANAREA SWAP (FIND) IF DROP DUP
SSYMB? IF 1 ELSE LFA @ 0 ENDIF ELSE 0 1
ENDIF UNTIL ;
 -->


SCR # 97

( SPECIFICATION HANDLING WORDS )
: MAKESMB ( CODE --- )
SPCUR @ SPMOFFSET @ DUP SSIZE < IF + C!
1 SPMOFFSET +! SPCUR @ SPLRFLAG @ + DUP
C@ 1+ SWAP C! ELSE DROP DROP DROP
CR ." TOO MANY SYMBOLS" QUIT ENDIF ;
: MAKEWLD ( --- )
SCANAREA C@ 128 + MAKESMB ;
: MAKESS ( PFA --- )
CFA EXECUTE MAKESMB ;
: WLD? ( --- )
SCANAREA DUP C@ 0 DO 1+ DUP C@ WLD = 0=
IF DROP CR ." ILLEGAL SYMBOL " SCANAREA
COUNT TYPE QUIT ENDIF LOOP DROP ;
: WLDORSEP ( --- )
SPLRFLAG @ IF WLD? MAKEWLD ELSE SCANAREA
 1+ C@ 45 = IF 1 SPLRFLAG ! ELSE WLD?
MAKEWLD ENDIF ENDIF ;
 -->
```

```
SCR # 98

( SPECIFICATION HANDLING WORDS )
: SPMAKE ( --- )
93 WORD HERE 1+ SCPTR !
HERE DUP C@ + 1+ SCCEIL !   SCCEIL @
BL OVER C! 1+ 93 OVER C! 1+ BL SWAP C!
SPMEM SPCUR ! 2 SPMOFFSET ! 0 SPCUR @ C!
 0 SPCUR @ 1+ C! 0 SPLRFLAG !
BEGIN SCPTR @ BL ENCLOSE SCPTR +! OVER
- 31 AND >R R SCANAREA C! + DUP SCCEIL
@ < SWAP SCANAREA 1+ R> 1+ CMOVE WHILE
CONTEXT @ @ SSFIND DUP IF MAKESS ELSE
DROP LATEST SSFIND DUP IF MAKESS ELSE
DROP WLDORSEP ENDIF ENDIF REPEAT
SPLRFLAG @ IF SPCUR @ SWNORM ELSE CR
." RIGHT PART NOT FOUND" QUIT ENDIF ;
: (SI)  R 2+ R> @ >R ;

: SI
SPMAKE STATE @ IF COMPILE (SI) HERE 0 ,
SPCUR @ SPTOHERE HERE SWAP ! ELSE SPCUR
@ ENDIF ; IMMEDIATE
 -->


SCR # 99

( SPECIFICATION HANDLING WORDS )

: SUBSTI ( AD --- ; SWRENOLD,SWRENNEW )
SCOUNTS + DUP IF 0 DO DUP C@ SWRENOLD @
= IF SWRENNEW @ OVER C! ENDIF 1+ LOOP
ELSE DROP ENDIF DROP ;
: MPREN ( SOURCEAD DESTAD --- )
129 SWRENNEW ! OVER OVER SSIZE CMOVE
OVER SCOUNTS DROP DUP >R + R> DUP IF 0
DO 1 - DUP C@ 128 > IF DUP C@ SWRENOLD
! OVER SUBSTI SWRENNEW @ 0 SWRENNEW !
>R >R >R DUP SUBSTI R> R> R> 1+
SWRENNEW  ! ENDIF LOOP ELSE DROP ENDIF
DROP DROP DROP ;
 -->
```

```
SCR # 100

( SPECIFICATION HANDLING WORDS )

: MPSS ( OP1AD OP2AD DESTAD --- )
>R OVER 1+ C@ OVER C@ > IF OVER C@ R C!
OVER SCOUNTS DROP R 2+ SWAP CMOVE OVER
SCOUNTS >R + OVER C@ R> SWAP - DUP R 1+
C! R SCOUNTS DROP + SWAP CMOVE SWAP
DROP SCOUNTS >R + R> R SCOUNTS + + OVER
R 1+ DUP C@ ROT + DUP R C@ + SSIZE > IF
DROP DROP DROP DROP DROP R> DROP CR
." TOO LONG PRODUCT" QUIT ENDIF SWAP C!
SWAP CMOVE ELSE OVER OVER C@ SWAP 1+ C@
- R C! DUP 1+ C@ R 1+ C! DUP 2+ R 2+ R
C@ CMOVE OVER SCOUNTS DROP R SCOUNTS
DROP + SWAP DUP R C@ + DUP R 1+ C@ +
SSIZE > IF DROP DROP DROP DROP DROP DROP
 R> DROP CR ." TOO LONG PRODUCT" QUIT
ENDIF R C! CMOVE SWAP DROP SCOUNTS DROP
+ R SCOUNTS >R + R> CMOVE ENDIF R> DROP
;
 --->


SCR # 101

( SPECIFICATION HANDLING WORDS )

: S* ( AD1 AD2 --- AD3 )
OVER SZERO= IF DROP DROP SZERO ELSE DUP
SZERO= IF DROP DROP SZERO ELSE
>R SPMEM DUP >R SSIZE CMOVE R> R> SPMEM
DUP >R SSIZE CMOVE R> DUP SWINCR? DROP
0 SPLRFLAG ! OVER 1+ C@ OVER C@ MIN DUP
IF >R OVER SCOUNTS + + OVER SCOUNTS
DROP + R> 0 DO 1 - SWAP 1 - SWAP
DUP C@ 128 < IF OVER C@ 128 < IF OVER
C@ OVER C@ = 0= IF 1 SPLRFLAG ! LEAVE
ENDIF ELSE OVER C@ SWRENOLD ! DUP C@
SWRENNEW ! >R >R >R DUP SUBSTI R> R> R>
ENDIF ELSE OVER C@ SWRENNEW ! DUP C@
SWRENOLD ! >R >R DUP SUBSTI R> R> ENDIF
LOOP DROP DROP ELSE DROP ENDIF
SPLRFLAG @ IF DROP DROP SZERO ELSE OVER
OVER SPMEM DUP SPCUR ! MPSS DROP DUP
SPCUR @ SWAP MPREN ENDIF ENDIF ENDIF ;

 -->
```

```
SCR # 102

( SPECIFICATION HANDLING WORDS )

: SLPONLY ( AD --- AD )
DUP SZERO= IF DROP SZERO ELSE SPMEM
OVER OVER SSIZE CMOVE SWAP DROP 0 OVER
1+ C! ENDIF ;

: SRPONLY ( AD --- AD )
DUP SZERO= IF DROP SZERO ELSE SPMEM
OVER 1+ C@ OVER 1+ C! 0 OVER C! OVER
SCOUNTS >R + OVER SCOUNTS DROP + R>
CMOVE SWAP DROP DUP SWNORM ENDIF ;

: SINV ( AD --- AD )
DUP SZERO= IF DROP SZERO ELSE SPMEM
OVER C@ OVER 1+ C! OVER 1+ C@ OVER C!
OVER SCOUNTS >R + OVER 2+ R> CMOVE OVER
SCOUNTS DROP >R OVER SCOUNTS DROP + R>
CMOVE SWAP DROP DUP SWNORM ENDIF ;
 -->



SCR # 103

( SPECIFICATION HANDLING WORDS )

: SPROJMAX ( AD --- AD )
DUP SZERO= IF DROP S[ --- ] DUP CR
." WARNING: PROJECTION OF SZERO IS " S.
ELSE DUP SCOUNTS MIN SWAP DROP DUP IF
0 SLDIFF ! OVER DUP 2+ SWAP SCOUNTS DROP
 + ROT 0 DO OVER SLDIFF @ + C@ OVER
SLDIFF @ + C@ = IF 1 SLDIFF +! ELSE
LEAVE ENDIF LOOP SLDIFF @ IF SLDIFF @ +
>R SLDIFF @ + >R SPMEM OVER SCOUNTS
SLDIFF @ - >R SLDIFF @ - >R DROP R>
OVER C! R> OVER 1+ C! DUP SCOUNTS DROP
R> ROT ROT CMOVE DUP SCOUNTS >R + R> R>
ROT ROT CMOVE SWAP DROP ELSE DROP DROP
ENDIF ELSE DROP ENDIF ENDIF ;

: SINF ( AD1 AD2 --- AD3 )
OVER OVER SREL DUP 0= IF DROP DROP DROP
SZERO ELSE 2 = IF SWAP DROP ELSE DROP
ENDIF ENDIF ;
 -->
```

```
SCR # 104

( SPECIFICATION HANDLING WORDS )

: S< ( AD1 AD2 --- FLAG ) SREL 1 = ;
: S> ( AD1 AD2 --- FLAG ) SREL 2 = ;
: S= ( AD1 AD2 --- FLAG ) SREL 4 = ;
: S-REL ( AD1 AD2 --- FLAG ) SREL 0= ;
: S+REL ( AD1 AD2 --- FLAG ) S-REL 0= ;
: S>= SREL 6 AND 0= 0= ;
: S<= SREL 5 AND 0= 0= ;

: SRPLEN ( AD --- L2 ) DUP SZERO= IF
DROP -1 ELSE 1+ C@ ENDIF ;

: SLPLEN ( AD --- L1 ) DUP SZERO= IF
DROP -1 ELSE C@ ENDIF ;

: S@ ( VARAD --- AD ) @ ;
: S! ( AD VARAD --- ) @ SSIZE CMOVE ;
: S? ( VARAD --- ) S@ S. ;

: SARRAY
<BUILDS 0 DO HERE 2+ , SSIZE ALIGN ALLOT
 LOOP DOES> SWAP SSIZE ALIGN 2+ * + ;
 -->
SCR # 105

( SPECIFICATION HANDLING WORDS )
: MAKESUP ( AD1 AD2 ADPR --- ADSUP )
C@ ROT DUP SPCUR ! C@ OVER - SLDIFF !
OVER C@ SWAP - SLDIFF @ 0= IF DROP DROP
SPCUR @ ELSE DUP 0= IF DROP ELSE DUP
SLDIFF @ MIN 0 DO OVER 1+ OVER + C@
SPCUR @ 1+ SLDIFF @ + C@ DUP 128 < IF
OVER 128 < IF = IF 1 - SLDIFF @ 1 -
SLDIFF ! ELSE LEAVE ENDIF ELSE DROP DROP
 LEAVE ENDIF ELSE DROP DROP LEAVE ENDIF
LOOP DROP DROP SPMEM SPCUR @ C@ SLDIFF
@ - OVER C! SPCUR @ 1+ C@ SLDIFF @ -
OVER 1+ C! SPCUR @ 2+ SLDIFF @ + OVER
SCOUNTS DROP CMOVE SPCUR @ SCOUNTS DROP
+ SLDIFF @ + OVER SCOUNTS >R + R> CMOVE
ENDIF ENDIF ;
: -SSUP ( AD1 AD2 --- AD3 1  /  0 )
OVER SZERO= IF DUP SZERO= IF DROP DROP
SZERO 1 ELSE SWAP DROP 1 ENDIF ELSE
DUP SZERO= IF DROP 1 ELSE OVER SPROJMAX
OVER SPROJMAX OVER S= IF MAKESUP 1 ELSE
DROP DROP DROP 0 ENDIF ENDIF ENDIF ;

 BASE !
```

Приложение 2. Реализация статического контроля типов для линейных ФОРТ-программ

Предлагаемые средства позволяют по спецификациям базовых понятий и по ФОРТ-тексту получить сведения о корректности этого текста. Тестируемые тексты могут содержать ":"-определения и определения переменных (слова ":" и " VARIABLE " переопределены таким образом, чтобы обеспечить вычисление спецификаций новых понятий). Все базовые понятия необходимо описывать (неописанные понятия работают согласно "старому" определению, если такое имеется в системе). В результате обработки ФОРТ-текста вычисляются спецификации понятий, которые определяются в этом тексте, и пользователь может заказать их вывод на экран.

COLDEF — в форме: ⟨s-val⟩ COLDEF ⟨name⟩
Определение спецификации базового понятия.

PRSPOF — в форме: PRSPOF ⟨name⟩
Вывод спецификации понятия на экран.

```
SCR # 130

( FORTH - STATIC TYPE CHECKING )
S[ --- ] SVARIABLE SACCU
: COLSP ( AD --- )
SACCU S@ SWAP S* SACCU S! ;
: COLDEF (  <SVALUE> COLDEF <NAME> )
<BUILDS SPTOHERE DOES> COLSP ;
: PRSPOF ( PRSPOF <NAME> )
-FIND IF DROP DUP @ @ ' COLSP CFA = IF
2+ S. ELSE NFA ID. ." ?!" QUIT ENDIF
ELSE HERE COUNT TYPE ." ?" QUIT ENDIF ;
: <: [COMPILE] : ; IMMEDIATE
: ;> [COMPILE] ; ; IMMEDIATE

SSYMB AD
: VARIABLE
DROP S[ --- AD ] COLDEF ;

<: :
<BUILDS S[ --- ] DUP SACCU S! HERE SSIZE
 ALIGN DUP ALLOT CMOVE DOES> COLSP ;>

<: ;
SACCU S@ LATEST PFA 2+ SSIZE CMOVE ;>
```

```
        ( TYPE CHECKING EXAMPLE )

        ( BASIC SPECIFICATIONS )
SSYMB AD
SSYMB VAL
S[ AD --- VAL ]          COLDEF  @
S[ VAL AD --- ]          COLDEF  !
S[ VAL --- ]             COLDEF  .
S[ VAL VAL --- VAL ]     COLDEF  +
S[ --- VAL ]             COLDEF  0
S[ --- AD ]              COLDEF  V

        ( FORTH-PROGRAMS )
: T1 . V ;
: T2 V . ;
: T3 V ! ;
: T4 @ V ;
: T5 0 V ! ;
: T6 @ . ;
: T7 0 ! ;
: T8 0 . . V @ . 0 0 . ;
: T9 @ V @ + V ;

        ( RESULTS )

PRSPOF T1 S[ VAL --- AD ]  OK
PRSPOF T2 SZERO  OK
PRSPOF T3 S[ VAL --- ]  OK
PRSPOF T4 S[ AD --- VAL AD ]  OK
PRSPOF T5 S[ --- ]  OK
PRSPOF T6 S[ AD --- ]  OK
PRSPOF T7 SZERO  OK
PRSPOF T8 S[ VAL --- VAL ]  OK
PRSPOF T9 S[ AD --- VAL AD ]  OK
```

```
( SPECIFICATIONS )
SSYMB   AD
SSYMB   0
SSYMB   1
SSYMB   2
SSYMB   4
 S[ --- ]              COLDEF  COMPILE
 S[ --- ]              COLDEF  [COMPILE]
 S[ --- ]              COLDEF  ?COMP
 S[ --- ]              COLDEF  OBRANCH
 S[ --- ]              COLDEF  BRANCH
 S[ --- ]              COLDEF  IMMEDIATE
 S[ ** * --- * ** ]    COLDEF  SWAP
 S[ * --- ]            COLDEF  DROP
 S[ **** *** ** * --- *** ** * **** ]
                       COLDEF  4ROLL
 S[ --- AD ]           COLDEF  HERE
 S[ --- 0 ]            COLDEF  0
 S[ --- 1 ]            COLDEF  1
 S[ --- 2 ]            COLDEF  2
 S[ --- 4 ]            COLDEF  4
 S[ * --- ]            COLDEF  ,
 S[ * * --- ]          COLDEF  ?PAIRS
 S[ * AD --- ]         COLDEF  !


( IF, ENDIF, ELSE, BEGIN, AGAIN, WHILE,
                REPEAT  -  FIG-FORTH )

: IF     COMPILE OBRANCH HERE 0 , 2 ;
                                  IMMEDIATE
: ENDIF    ?COMP 2 ?PAIRS HERE SWAP ! ;
                                  IMMEDIATE
: ELSE 2 ?PAIRS COMPILE BRANCH HERE 0 ,
   SWAP 2 [COMPILE] ENDIF 2 ; IMMEDIATE
: BEGIN    ?COMP HERE 1 ;       IMMEDIATE
: AGAIN    1 ?PAIRS COMPILE BRANCH , ;
                                  IMMEDIATE
: WHILE    [COMPILE] IF DROP 4 ;
                                  IMMEDIATE
: REPEAT   4ROLL 4ROLL  [COMPILE] AGAIN
     DROP 2 [COMPILE] ENDIF ; IMMEDIATE



( RESULTS )

PRSPOF IF     S[ --- AD 2 ]  OK
PRSPOF ENDIF  S[ AD 2 --- ]  OK
PRSPOF ELSE   S[ AD 2 --- AD 2 ]  OK
PRSPOF BEGIN  S[ --- AD 1 ]  OK
PRSPOF AGAIN  S[ * 1 --- ]  OK
PRSPOF WHILE  S[ --- AD 4 ]  OK
PRSPOF REPEAT S[ ** 1 AD * --- ]  OK
```

УТВЕРЖДАЮ

Зам. директора СКБ ПС

ПО "СЧЕТМАШ" по научной части

... Афанасов В.М.

"... 12 ..... 1986 г.

СПРАВКА

о внедрении научных результатов диссертационной работы ПЁЙАЛА Я.Р. "Спецификации ФОРТ-программ и их применение в системах построения трансляторов"

Настоящим подтверждается, что описанный в диссертационной работе метод спецификации ФОРТ-программ внедрен в составе системы построения трансляторов ТАРТУ в ПО "СЧЕТМАШ". СПТ ТАРТУ используется для реализации языка программирования ФОРТРАН на ПЭВМ "Искра-226".

Средства, разработанные Пёйалом Я.Р., позволили отладить описание языка ФОРТРАН.

Зав. отд. № 8 Самотес Л.А.

Болотский Е.Н.