



ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ,
ПАТЕНТАМ И ТОВАРНЫМ ЗНАКАМ

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(21), (22) Заявка: 2006130519/09, 24.08.2006

(24) Дата начала отсчета срока действия патента:
24.08.2006

(43) Дата публикации заявки: 27.02.2008

(45) Опубликовано: 20.08.2009 Бюл. № 23

(56) Список документов, цитированных в отчете о
поиске: RU 2137185 C1, 10.09.1999. EP 1420562 A1,
19.05.2004. JP 2004302584, 28.10.2004. GB
2283341 A1, 03.05.1995. DE 10247914 A1,
22.04.2004. GB 2366640 A1, 13.03.2002. FR
2559599, 16.08.1985.

Адрес для переписки:
119192, Москва, Мичуринский пр-кт, 1,
Институт механики МГУ, к. 119, Ф.М.
Пучкову

(72) Автор(ы):

Пучков Федор Михайлович (RU),
Шапченко Кирилл Александрович (RU)

(73) Патентообладатель(и):

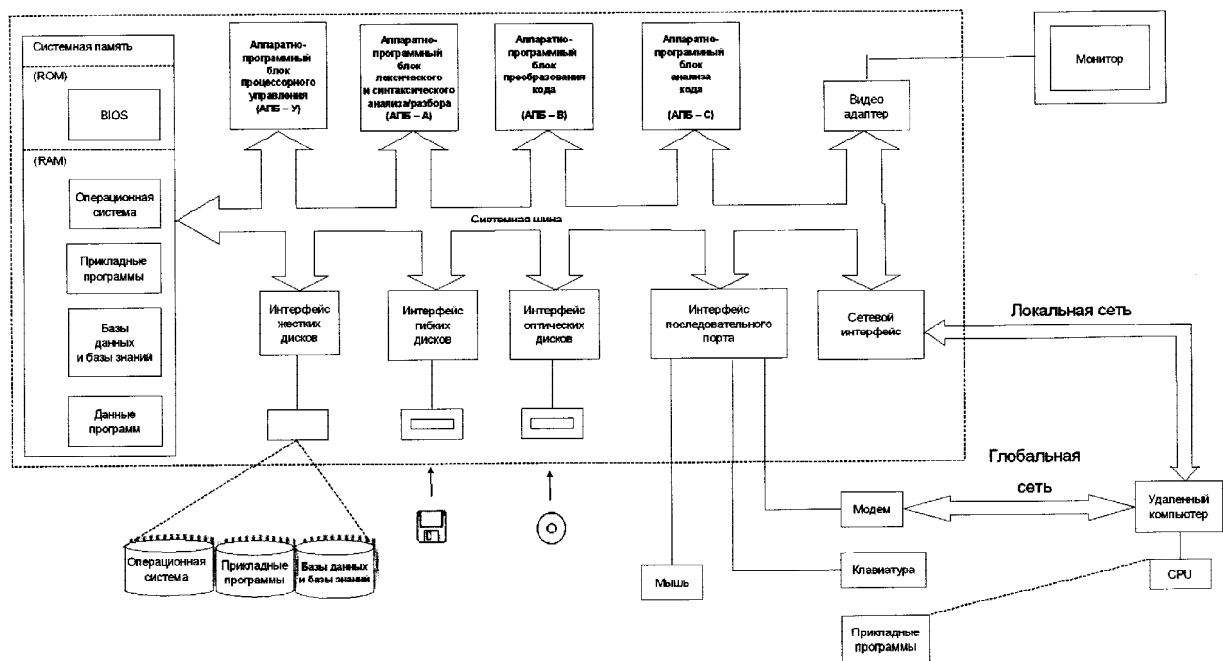
Пучков Федор Михайлович (RU),
Шапченко Кирилл Александрович (RU)

**(54) СПОСОБ ГЕНЕРАЦИИ БАЗ ЗНАНИЙ ДЛЯ СИСТЕМ ВЕРИФИКАЦИИ ПРОГРАММНОГО
ОБЕСПЕЧЕНИЯ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ КОМПЛЕКСОВ И
УСТРОЙСТВО ДЛЯ ЕГО РЕАЛИЗАЦИИ**

(57) Реферат:

Изобретение относится к средствам автоматизации обучения и научных исследований и может быть использовано в интерактивных системах в процессе верификации программного обеспечения (ПО) распределенных вычислительных комплексов. Технический результат заключается в расширении функциональных возможностей процессов верификации ПО. В способе и устройстве обеспечивают полную управляемость и наблюдаемость основных процессов проверки исходного кода ПО, совмещают процессы ввода и обработки исходного кода ПО по зависимым или

независимым интерфейсным каналам. Участки или точки уязвимости исходного кода ПО определяют на основе преобразования исходного кода ПО во внутреннее представление, которое хранят в виде баз знаний, а точки или участки уязвимости исходного кода ПО определяют на основе автоматического составления и решения соответствующих систем уравнений. При этом в процессе верификации осуществляют синтаксическую подсветку на экране участков уязвимости исходного кода ПО, используя граф операторов исходного кода ПО во внутреннем представлении инструкций языка программирования. 2 н. и 7 з.п. ф-лы, 39 ил.



Фиг. 1



FEDERAL SERVICE
FOR INTELLECTUAL PROPERTY,
PATENTS AND TRADEMARKS

(12) **ABSTRACT OF INVENTION**(21), (22) Application: **2006130519/09, 24.08.2006**(24) Effective date for property rights:
24.08.2006(43) Application published: **27.02.2008**(45) Date of publication: **20.08.2009 Bull. 23**

Mail address:

**119192, Moskva, Michurinskij pr-kt, 1, Institut
mekhaniki MGU, k. 119, F.M. Puchkovu**

(72) Inventor(s):

**Puchkov Fedor Mikhajlovich (RU),
Shapchenko Kirill Aleksandrovich (RU)**

(73) Proprietor(s):

**Puchkov Fedor Mikhajlovich (RU),
Shapchenko Kirill Aleksandrovich (RU)**(54) **GENERATION METHOD OF KNOWLEDGBASES FOR SYSTEMS OF VERIFICATION OF
DISTRIBUTED COMPUTER COMPLEXES SOFTWARE AND DEVICE FOR ITS IMPLEMENTATION**

(57) Abstract:

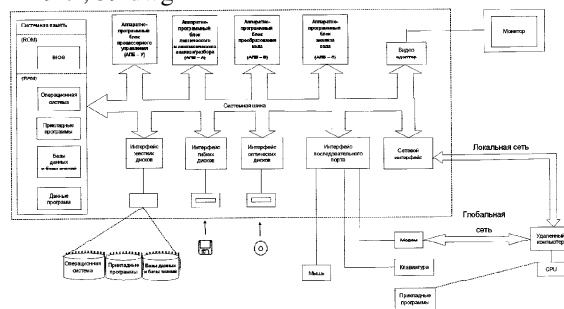
FIELD: physics; computer facilities.

SUBSTANCE: invention concerns resources of automation of training and scientific researches and can be used in interactive systems in the course of software verification (S/W) of the distributed computer complexes. In the given way and the device complete controllability and observability of the main processes of check of a S/W source code is provided, processes of input and processing of a S/W source code on dependent or independent interface channels are combined. Sites or points of vulnerability of a S/W source code are determined on the basis of conversion of a S/W source code in an internal form which is stored in sort of knowledgebases, and points or vulnerability sites of a S/W source code are determined on the basis of automatic compilation and solution of appropriate set

of equationses. Thus in the course of verification syntactic highlighting of sites of vulnerability of a S/W source code on the screen is carried out, using the graph of operators of S/W source code in an internal form of instructions of the programming language.

EFFECT: expansion of functionality of S/W verification processes.

9 cl, 39 dwg



Фиг. 1

Изобретение относится к средствам автоматизации обучения и научных исследований и предназначено для использования в интерактивных системах автоматизации научно-исследовательских работ в процессе верификации (проверки, тестирования и т.п.) программного обеспечения (ПО) распределенных
5 вычислительных комплексов (РВК).

Известны интерактивные системы автоматизации основных процессов научных исследований в процессе верификации (проверки, тестирования и т.п.) ПО РВК (см. например, DE 10247914 A1 (BOSCH GMBH ROBERT (DE)) 22.04.2004, G 09 B 9/08; DE
10 10141737 A1 (DAIMLER CHRYSLER AG (DE)) 03.04.2003, G 09 B 9/08; H 04 L 9/30; H 04 L 12/40; B 60 R 16/02; FR 2559599 A1 (GAUER BERNARD (FR)) 16.08.1985, G 09 B 9/08; GB 2366640 A (IBM) 13.03.2002 G 06 F 1/00; GB 2283341 (SOPHOS PLC (GB)) 03.05.1995 G 06 F 1/00; JP 2004302584 A (NIPPON ELECTRIC CO) 28.10.2004, G 09 B 9/08; JP 2004164617 A (MICROSOFT CORP) 10.06.2004, G 09 B 9/08). В основе работы вышеперечисленных
15 систем использованы способы динамического или статического тестирования ПО. Вышеперечисленные технические решения позволяют для некоторых классов ПО избежать их некорректного исполнения, однако существенным недостатком таких решений является сложность выявления скрытых уязвимостей в исходном коде (ИК) ПО РВК, а также невозможность гарантировать правильность полученных
20 результатов на ранних стадиях проектирования ПО РВК. При этом технические реализации статического способа анализа ПО, как правило, представляют различные модификации лексических сканеров ИК ПО и не позволяют осуществить глубокий анализ ни хода выполнения ПО РВК, ни содержания используемой памяти (например, массивов, переменных и т.п.).

В этой связи главный недостаток известных технических решений предопределен тем, что процессы верификации ПО РВК, как правило, представляют ненаблюдаемые/неуправляемые во времени процессы. При этом сам процесс поиска
30 уязвимости в ИК ПО представляет в большей степени искусство, нежели какую-либо формализованную процедуру. При этом низкая производительность данного класса технических решений обусловлена потерей времени, затрачиваемого на формирование тестовых наборов данных, выбора количественных и качественных характеристик, позволяющих оценить уязвимость ИК ПО, определения входных данных
35 определенных типов, в особенности строкового или процедурного типа, являющиеся наиболее распространенными в практике разработки ПО РВК.

Достижению требуемого технического результата во всех приведенных аналогах, включая соответствующий способ верификации ПО, препятствует то, что ни одно из
40 них не может обеспечить полную наблюдаемость/управляемость за динамикой процесса генерации баз знаний на основных этапах верификации ИК ПО РВК систем автоматизации научно-исследовательских работ.

Задачей изобретения является расширение функциональных возможностей интерактивных процессов выполнения научно-исследовательских работ на основе
45 автоматизации основных процессов генерации баз знаний для верификации ПО РВК, а также повышение точности и достоверности выявления уязвимостей в ИК ПО на основных этапах проектирования и сопровождения (аудита) программного кода сложных вычислительных систем и сетей.

Для решения поставленной задачи предложен способ генерации баз знаний для систем верификации программного обеспечения распределенных вычислительных комплексов и устройство для его реализации.

Способ генерации баз знаний для систем верификации программного обеспечения

(ПО) распределенных вычислительных комплексов (РВК), включает овмещение процессов ввода и обработки исходного кода ПО по зависимым или независимым каналам на основе использования сенсорных или механических манипуляторов (клавиатуры, мыши, джойстика и т.п.) рабочего места оператора ЭВМ, сетевых интерфейсов локальной или глобальной сети, при этом участки или точки уязвимости исходного кода ПО определяют на основе преобразования исходного кода ПО во внутреннее представление, которое хранят в виде баз знаний, а точки или участки уязвимости исходного кода ПО определяют на основе автоматического составления и решения соответствующих систем уравнений.

При этом внутреннее представление исходного кода ПО на языке программирования Си представляется в виде динамических массивов и соответствующих баз знаний, которые используются в реальном масштабе времени для верификации ПО распределенных вычислительных комплексов.

Динамические массивы и соответствующие базы знаний трансформации или преобразования дерева исходного кода программного обеспечения в процессе верификации/разбора исходного листинга программы состоят в том, что если дерево с указанной вершиной соответствует определению и содержит внутри себя спецификатор "extern", то такое дерево удаляется, кроме этого, если дерево соответствует блочной инструкции, то каждое поддереву блочной инструкции с вершиной в "ребенке" обрабатываемой вершины заменяется на список поддеревьев, соответствующих инструкциям в этой блочной инструкции.

Точки или участки уязвимости исходного кода ПО определяют на основе автоматического составления и решения линейных систем уравнений.

В качестве зависимых каналов используют интерфейсы жестких, гибких или оптических дисков.

В качестве независимых каналов используют интерфейс последовательного порта или сетевой интерфейс.

В качестве независимых каналов используют интерфейс последовательного порта или сетевой интерфейс, а в качестве зависимых каналов используют интерфейсы жестких, гибких или оптических дисков.

При этом генерацию баз знаний осуществляют на основе использования аннотаций внешних функций ИК ПО РВК.

База знаний представляет набор правил или моделей протоколов верификации ИК ПО, обеспечивающих поддержку интерактивных процессов поиска уязвимости ПО.

Кроме этого, в процессе верификации ИК ПО осуществляют синтаксическую подсветку на экране блока видеоконтроля (БВ), предназначенного для визуализации (БВ) участков уязвимости ИК ПО, при этом используют граф (дерево разбора и т.п.) операторов ИК ПО во внутреннем представлении инструкций языка программирования.

Дигностику процесса верификации ИК ПО осуществляют на основе измерения длительности выполнения основных этапов (циклов и т.п.) поиска уязвимости и сопоставления полученных значений с ранее предписанными или предсказанными оценками показателя (степени, энтропии и т.п.) критичности уязвимости ИК ПО.

Визуальный язык в виде раскрасок графа максимально приближен к профессиональному языку специалистов в прикладной области.

Для реализации предлагаемого способа генерации баз знаний предлагается интерактивное программируемое устройство (ИПУ) для систем верификации программного обеспечения распределенных вычислительных комплексов (СВПО)

РВК), содержащее аппаратно-программный блок (АПБ) лексического и семантического анализа/разбора, АПБ преобразования кода, АПБ анализа кода; АПБ процессорного управления, видео адаптер, интерфейсы жестких, гибких и оптических дисков, интерфейс последовательного порта, сетевой интерфейс и системную память, которые объединены системной шиной, при этом системная память содержит постоянное запоминающее устройство (ROM) и оперативное запоминающее устройство (RAM/ОЗУ), в ячейках оперативной памяти и жестких дисков размещают/записывают операционные системы, прикладные программы, базы данных и базы знаний, которые содержат листинги исходных программ, грамматику языка программирования (например, грамматику языка программирования Си), правила преобразования дерева разбора листинга программы, дерево разбора листинга программы, таблицу типов языка программирования, аннотации внешних функций, включающие их грамматику и семантику, код программ на языке внутреннего представления, условия корректности языка внутреннего представления исходного кода программы, условия проверки корректности подозрительных точек исходного кода программы, информационную базу, содержащую системы ограничений в виде алгебраических уравнений и неравенств, отчеты об обнаруженных уязвимостях программного кода, включающие:

указание на местоположение возможной уязвимости в исходном коде программы, которое содержит имя файла листинга программы, номер строки и номер позиции в строке программы в котором возможно переполнение буфера запоминающего устройства, контекст исходной программы, содержащий возможное переполнение (некоторую "окрестность" потенциально опасной точки переполнения буфера запоминающего устройства);

указание причины переполнения буфера запоминающего устройства - значения исходных переменных, приводящих к возникновению уязвимости исходного кода программного обеспечения;

показатель или степень критичности обнаруженной уязвимости исходного кода программного обеспечения;

указание на перечень правил или алгоритмов для устранения уязвимости исходного кода программного обеспечения.

При этом аппаратно-программный блок процессорного управления, обеспечивающий синхронизацию основных режимов интерактивной верификации программного обеспечения распределенных вычислительных комплексов, содержит последовательно соединенные блок нормирующих преобразователей (БНП), модуль коммутатора, модуль аналого-цифрового преобразователя (АЦП), модуль формирования статических координат графоаналитического раstra (МФСКГАР), блок видеоконтроля (БВ), импульсный регулятор (ИР) и модуль формирования динамической развертки графоаналитического раstra (МФДРГАР), первый и второй информационные входы/выходы которого соединены соответственно с управляющими входами модуля коммутации и аналого-цифрового преобразователя, а третий, четвертый, пятый, шестой, седьмой и восьмой - с вторым, третьим, четвертым, пятым, шестым и седьмым информационными входами/выходами МФСКГАР, при этом второй вход блока видеоконтроля объединен с седьмым информационным входом/выходом МФСКГАР и восьмым информационным входом/выходом МФДРГАР.

Блок видеоконтроля (БВ) содержит модуль регистратора (МР), на выходе которого установлены маска, оптическая система и преобразователь оптического сигнала,

выходом подключенный к управляющему входу блока динамической развертки.

При этом оптический коэффициент пропускания маски, через которую проходит оптический сигнал, имеет одно и то же значение для точек одной гипотезы/энтропии о состоянии уязвимости ИК ПО и различное для других значений гипотез.

Маска изготовлена так, что в пространстве ее координат (X, Y, Z) выделены области уязвимости ИК ПО и соответствующие значения могут быть определены на основе оценки степени уязвимости ИК ПО ВРК.

В качестве оценки степени уязвимости ИК ПО ВРК могут быть использованы значения энтропии состояния верифицируемого ИК ПО.

Отличительные признаки предлагаемого способа и устройства для его реализации позволяют расширить функциональные возможности основных процессов верификации ИК ПО распределенных вычислительных комплексов непосредственно при выполнении научно-исследовательских работ на основе использования реальных динамических характеристик распределенных вычислительных систем и сетей, при этом предоставляется возможность повысить достоверность и объективность принимаемых решений на основных этапах исследования, разработки и сопровождения ПО распределенных вычислительных комплексов.

Предлагаемое изобретение поясняется чертежами.

На фиг.1 приведена структура интерактивного программируемого устройства (ИПУ) для систем верификации программного обеспечения распределенных вычислительных комплексов (СВПО РВК), содержащая аппаратно-программный блок (АПБ) лексического и семантического анализа/разбора (АПБ-А), АПБ преобразования кода (АПБ-В), АПБ анализа кода (АПБ-С), АПБ процессорного управления (АПБ-У), видеоадаптер, интерфейсы жестких, гибких и оптических дисков, интерфейс последовательного порта, сетевой интерфейс и системную память, которые объединены системной шиной, при этом системная память содержит постоянное запоминающее устройство (ROM) и оперативное запоминающее устройство (RAM/ОЗУ), в ячейках оперативной памяти и жестких дисков размещают/записывают операционные системы, прикладные программы, базы данных и базы знаний, которые содержат листинги исходных программ (D1), грамматику языка программирования (D2) (например, грамматику языка программирования Си), правила преобразования дерева разбора листинга программы (D3), дерево разбора листинга программы (D4), таблицу типов языка программирования (D5), аннотации внешних функций, включающие их грамматику и семантику (D6), код программ на языке внутреннего представления (D7), условия корректности языка внутреннего представления исходного кода программы (D8), условия проверки корректности подозрительных точек исходного кода программы (D9), информационную базу (D10), содержащую системы ограничений в виде алгебраических уравнений и неравенств, отчеты об обнаруженных уязвимостях программного кода (D11), включающие:

указание на местоположение возможной уязвимости в исходном коде программы, которое содержит имя файла листинга программы, номер строки и номер позиции в строке программы, в котором возможно переполнение буфера запоминающего устройства, контекст исходной программы, содержащий возможное переполнение (некоторую "окрестность" потенциально опасной точки переполнения буфера запоминающего устройства);

указание причины переполнения буфера запоминающего устройства - значения исходных переменных, приводящих к возникновению уязвимости исходного кода программного обеспечения;

показатель или степень критичности обнаруженной уязвимости исходного кода программного обеспечения;

указание на перечень правил или алгоритмов для устранения уязвимости исходного кода программного обеспечения.

5

На фиг.2 приведен пример реализации аппаратно-программного блока (АПБ) лексического и семантического анализа/разбора (АПБ - А), в котором приняты следующие обозначения:

10	Обозначение	Описание
	D1	База данных, содержащая листинги исходных программ (D1): набор файлов с исходными текстами (листингами) анализируемой программы Вход 1: чтение файлов листингов программ
15	D2	База данных, содержащая грамматику языка программирования (D2): грамматика языка (правила лексического и синтаксического разбора) Вход 1: чтение правил грамматики языка
	D3	База знаний, содержащая правила преобразования дерева разбора листинга программы (D3): правила преобразования дерева Вход 1: чтение правил преобразования дерева разбора
	D4	База данных дерева разбора листинга программы (D4): дерево разбора после выполнения части А Вход 1: сохранение дерева разбора
20	D5	База данных, содержащая таблицу типов языка программирования (D5): таблица типов после работы АПБ-А Вход 1: сохранение таблицы типов
25	A.1	Блок проверки условия: остались ли листинги программ для разбора? Выход 1: переход в случае истинности условия Выход 2: переход в случае ложности условия
30	A.2	Вызов процедуры PARSE FILE аппаратно-программного модуля А1 "Лексический и синтаксический разбор" для разбора очередного листинга программы. Работа данного модуля раскрыта на схеме, изображенной на фиг.3. Вход 1: прием управляющего сигнала Выход 1: обращение на чтение к данным - файлам листингов программ Выход 2: обращение на чтение к данным - грамматике языка программирования Выход 3: обращение на запись к данным - дереву разбора листингов программ после выполнения модуля А1 Выход 4: обращение на запись к данным - таблице типов А.5 после выполнения модуля А1 Выход 5: передача управления и полученных результатов (дерево разбора и таблица типов) в следующий этап обработки А.3.
35	A.3	Вызов процедуры SIMPLIFY SYNTAX TREE аппаратно-программного модуля А2 "Преобразование дерева разбора листингов программ к виду, пригодному для перевода во внутреннее представление" для обработки результатов выполнения результатов работы этапа А.2. Пример реализации процедуры SIMPLIFY SYNTAX TREE аппаратно-программного модуля А2 раскрыт на фиг.4. Вход 1: передача управления и получение результатов выполнения процедуры PARSE FILE аппаратно-программного модуля А1 Выход 1: передача управления на вход А.1 Выход 2: обращение на чтение к данным - дереву разбора после выполнения аппаратно-программного модуля А1 Выход 3: обращение на чтение к данным - таблице типов после выполнения аппаратно-программного модуля А1 Выход 4: обращение на запись к данным - дереву разбора после выполнения - модуля А2 Выход 5: обращение на запись к данным - таблице типов после выполнения - модуля А2 Выход 6: обращение на чтение к базе данных D3
40		
45	A.4	Данные - дерево разбора листингов программ после выполнения модуля А1. Вход 1: обращение из модуля А1 Вход 2: обращение из модуля А2
	A.5	Данные - таблица типов после выполнения модуля А1. Вход 1: обращение из модуля А1 Вход 2: обращение из модуля А2
50	A.6	Данные - дерево разбора листингов программ после выполнения модуля А2. Вход 1: обращение из модуля А2 Вход 2: обращение из модуля А3
	A.7	Данные - таблица типов после выполнения аппаратно-программного

	модуля А2. Вход 1: обращение из модуля А2 Вход 2: обращение из модуля А3
5 10	Вызов процедуры MERGE SYNTAX TREES аппаратно-программного модуля А3 "Поддержка многофайловых проектов". Пример реализации модуля А3 раскрыт на фиг.7. Выход 1: передача управления Выход 2: обращение на чтение к данным - дереву разбора после выполнения модуля А2 Выход 3: обращение на чтение к данным - таблице типов после выполнения модуля А2 Выход 4: сохранение итоговой таблицы типов Выход 5: сохранение итогового дерева разбора

На вход АПБ-А поступает набор файлов с исходными текстами анализируемой программы. На выходе АПБ-А получают преобразованное дерево разбора исходного кода программы и таблицу типов, используемых в анализируемой программе, готовые к проведению преобразования во внутреннее представление "Анализатора" - АПБ-А.

На фиг.3-7 приведены примеры реализации основных модулей АПБ-А.

На фиг.3 приведен пример реализации модуля А1 АПБ-А. Аппаратно программный модуль А1 обеспечивает режим "лексического и синтаксического разбора исходных текстов программ", перечень принятых обозначений имеет следующий вид:

Обозначение	Описание
25 30	А1.1 Разбиение исходного файла программы на поток лексем Вход 1: прием управляющего сигнала Выход 1: обращение на чтение к базе данных D1, содержащей листинги исходных программ Выход 2: обращение на чтение к базе данных D2, содержащей грамматику языка программирования Выход 3: передача управления
А1.2	Инициализация стека ОЗУ, после этой операции стек пуст
30 35	А1.3 Проверка условия: остались непрочитанные лексемы в потоке лексем? Выход 1: передача управления в случае, когда остались лексемы Выход 2: передача управления в случае, когда весь поток лексем уже обработан
А1.4	Извлечение лексемы из потока и помещение ее на вершину стека ОЗУ
А1.5	Инициация чтения синтаксических правил грамматики языка программирования, начиная с первого правила
35 40	А1.6 Проверка условия: остались непрочитанные синтаксические правила? Выход 1: передача управления в случае истинности условия Выход 2: передача управления в случае ложности условия
40 45	А1.7 Получение следующего синтаксического правила Выход 1: передача сигнала управления Выход 2: обращение на чтение к данным - грамматике языка
45 50	А1.8 Проверка условия: возможно ли применение правила к последовательности элементов на вершине стека? Выход 1: передача управления в случае истинности условия Выход 2: передача управления в случае ложности условия
А1.9	Замена последовательности элементов на вершине стека, подходящей под правило, на поддерево разбора, соответствующее правилу
45 50	А1.10 Проверка условия: синтаксическое правило соответствует введению нового типа? Выход 1: передача управления в случае истинности условия Выход 2: передача управления в случае ложности условия
50 	А1.11 Добавление записи в таблицу типов Выход 1: передача управления Выход 2: обращение на запись к данным - таблице типов после выполнения модуля А1.
50 	А1.12 Проверка условия: в стеке ровно один элемент, соответствующий дереву разбора исходного текста? Выход 1: передача управления в случае истинности условия Выход 2: передача управления в случае ложности условия

A1.13	Извлечение дерева разбора из стека Выход 1: обращение на запись к данным - дереву разбора после выполнения модуля A1 Выход 2: передача управляющего сигнала
-------	---

В основе работы данного модуля использован алгоритм LR-разбора файла с исходным текстом программы по набору лексических и синтаксических правил разбора. В процессе синтаксического разбора строится дерево разбора и таблица типов.

На фиг.4 приведен пример реализации модуля A2 АПБ-А. Аппаратно программный модуль A2 обеспечивает режим "преобразования дерева разбора кода программы к виду, пригодному для перевода его во внутреннее представление", перечень принятых обозначений имеет следующий вид:

Обозначение	Описание
A2.1	Вызов процедуры MARK SCOPES модуля A2 для корневой вершины дерева разбора. Пример реализации данной процедуры приведен на фиг.5. Вход 1: прием управляющего сигнала Выход 1: передача управления и результатов выполнения Выход 2: обращение на чтение к данным - дереву разбора после выполнения модуля A1 Выход 3: обращение на чтение к данным - таблице типов после выполнения модуля A1
A2.2	Инициация обхода детей корневой вершины дерева разбора начиная с первой вершины-ребенка
A2.3	Условие: остались дети корневой вершины с типом "определение функции"? Выход 1: передача управления в случае истинности условия Выход 2: передача управления в случае ложности условия
A2.4	Найти все объявления в поддереве, соответствующем телу функции, и переместить их таким образом, что они станут детьми текущей вершины с типом "определение функции"
A2.5	Переход к обработке следующей вершины-ребенка корневой вершины
A2.6	Вызов процедуры TRANSFORM SYNTAX TREE модуля A2, раскрываемой на фиг.6. Вход 1: передача управления Выход 1: передача управления Выход 2: обращение на чтение к данным - правилам преобразования дерева Выход 3: обращение на запись к данным - дереву разбора после выполнения модуля A2 Выход 4: обращение на запись к данным - таблице типов после выполнения модуля A2

Модуль A2 представляет процедуру "SIMPLIFY SYNTAX TREE", производящую надлежащее упрощение дерева разбора, результатом которого является дерево разбора и таблица типов, преобразование которых ко внутреннему представлению значительно проще по сравнению с преобразованием "сырого" дерева разбора.

Схема процедуры "SIMPLIFY SYNTAX TREE" представлена на фиг.4. В работе процедуры используются две внутренние рекурсивные процедуры "MARK SCOPES" и "TRANSFORM SYNTAX TREE", осуществляющие, соответственно, пометку областей видимости объявлений и определений в дереве разбора и преобразование дерева разбора по заранее заданному набору правил.

На фиг.5 приведен пример реализации рекурсивной процедуры A2.1. "MARK SCOPES", при этом приняты следующие обозначения:

Обозначение	Описание
A2.1.1	Синтез дерева разбора и таблицы типов Вход: прием управляющего сигнала Выход 1: передача управления Выход 2: обращение на чтение к данным - дереву разбора после выполнения модуля A1 Выход 3: обращение на чтение к данным - таблице типов после выполнения модуля A1

5	A2.1.2	Выбор: тип вершины? Выход 1: передача управления в случае, если вершина имеет тип "блочная инструкция" или "определение функции" Выход 2: передача управления в других случаях
	A2.1.3	Добавление к вершине атрибута "область видимости (scope)" с уникальным значением
	A2.1.4	Все идентификаторы объявляемых переменных, находящиеся в поддереве обрабатываемой вершины, пометить атрибутом "область видимости" с полученным значением на предшествующем шаге
10	A2.1.5	Инициализация обработки "детей вершины" начиная с первой вершины - "ребенка" Проверка условия: остались необработанные "дети вершины"?
	A2.1.6	Выход 1: передача управления в случае истинности условия Выход 2: передача управления в случае ложности условия
	A2.1.7	Рекурсивный вызов процедуры MARK SCOPES модуля A3 Вход 1: прием сигнала управления
		Выход 1: передача управления Выход 2: обращение на чтение к данным - дереву разбора после выполнения модуля A1 Выход 3: обращение на чтение к данным - таблице типов после выполнения модуля A1

На фиг.6 приведен пример реализации рекурсивной процедуры A2.6. "TRANSFORM SYNTAX TREE", при этом приняты следующие обозначения:

Обозначение	Описание
A2.6.1	Инициализация обработки "детей" корневой вершины дерева начиная с первой вершины - "ребенка"
A2.6.2	Проверка условия: остались необработанные дети вершины? Выход 1: передача управления в случае истинности условия Выход 2: передача управления в случае ложности условия
A2.6.3	Рекурсивный вызов процедуры TRANSFORM SYNTAX TREE модуля A2 для поддерева очередного ребенка обрабатываемой вершины Вход 1: прием сигнала управления Выход 1: передача сигнала управления Выход 2: обращение на чтение к данным - правилам преобразования дерева Выход 3: обращение на запись к данным - дереву разбора после выполнения модуля A2 Выход 4: обращение на запись к данным - таблице типов после выполнения модуля A2
A2.6.4	Сохранение полученного дерева разбора для вершины Выход 1: передача управления Выход 2: обращение на запись к данным - массиву результатов преобразования для детей вершины
A2.6.5	Инициализация процесса чтения правил преобразования дерева разбора, начиная с первого правила
A2.6.6	Проверка условия: остались непрочитанные правила преобразования? Выход 1: передача управления в случае истинности условия Выход 2: передача управления в случае ложности условия
A2.6.7	Прочитать очередное правило преобразования в D3 Выход 1: передача управления Выход 2: обращение на чтение к данным - правилам преобразования дерева в D3
A2.6.8	Проверка условия: правило преобразования применимо? Выход 1: передача управления в случае истинности условия Выход 2: передача управления в случае ложности условия
A2.6.9	Формирования в ОЗУ временных данных: временный массив результатов - деревьев разбора для "детей" обрабатываемой вершины
A2.6.10	Применение правила преобразования дерева к текущей вершине и набору деревьев разбора, хранимых во временном массиве результатов Вход 1: прием сигнала управления Выход 1: передача сигнала управления Выход 2: обращение на чтение к данным - массиву результатов преобразования для "детей" вершины Выход 3: обращение на запись к данным - дереву разбора после выполнения модуля A2 Выход 4: обращение на запись к данным - таблице типов после выполнения модуля A2

На вход процедуры TRANSFORM SYNTAX TREE поступает дерево (или поддерево) разбора, на выходе получается результирующее преобразованное дерево разбора. Внутри процедуры происходит рекурсивный вызов этой процедуры и сохранение

результатов (дерева разбора) выполнения этого вызова во временном массиве результатов, который размещается в ОЗУ.

При этом оригинальность предлагаемого способа трансформации или преобразования дерева в процессе верификации/разбора исходного листинга программы состоит в том, что если дерево с указанной вершиной соответствует определению и содержит внутри себя спецификатор "extern", то такое дерево удаляется, кроме этого, если дерево соответствует блочной инструкции, то каждое поддереву блочной инструкции с вершиной в "ребенке" обрабатываемой вершины заменяется на список поддеревьев, соответствующих инструкциям в этой блочной инструкции.

На фиг.7 приведен пример реализации аппаратно-программного модуля А3, обеспечивающего режим верификации многофайловых программных проектов на основе реализации процедуры "MERGE SYNTAX TREES" для компоновки деревьев разбора и таблиц типов, при этом приняты следующие обозначения:

Обозначения	Описание
А3.1	Инициализация результирующего дерева разбора и таблицы типов
А3.2	Проверка условия: остались деревья разбора для компоновки? Выход 1: передача сигнала управления в случае истинности условия Выход 2: передача сигнала управления в случае ложности условия
А3.3	Получение очередного дерева разбора Выход 1: передача сигнала управления Выход 2: обращение на чтение к данным - набору деревьев разбора после выполнения модуля А2
А3.4	Проверка условия: остались необработанные дети корневой вершины? Выход 1: передача сигнала управления в случае истинности условия Выход 2: передача сигнала управления в случае ложности условия
А3.5	Добавить поддерево, в котором корневой вершиной является очередной "ребенок", в результирующее дерево разбора таким образом, что корневая вершина добавляемого дерева окажется "ребенком" корневой вершины результирующего дерева Выход 1: передача сигнала управления Выход 2: обращение на запись к данным D4 - дереву разбора после выполнения части А
А3.6	Проверка условия: остались таблицы типов для компоновки? Выход 1: передача сигнала управления в случае истинности условия Выход 2: передача сигнала управления в случае ложности условия
А3.7	Получение очередной таблицы типов Выход 1: передача сигнала управления Выход 2: обращение на чтение к данным - набору таблиц типов после выполнения модуля А2
А3.8	Добавление всех элементов из таблицы типов в результирующую таблицу типов Выход 1: передача сигнала управления Выход 2: обращение на запись к данным - таблице типов после выполнения части А

На фиг.8 приведен пример реализации аппаратно-программного блока (АПБ) преобразования кода (АПБ-В), обеспечивающего преобразование дерева разбора исходной программы к внутреннему представлению, при этом приняты следующие обозначения:

Обозначение	Описание
В1	Обращение к аппаратно-программному модулю В1 "Встраивание функций". Работа данного модуля раскрыта на схеме, изображенной на фиг.9. Вход 1: прием сигнала управления Выход 1: передача сигнала управления Выход 2: обращение к базе данных D4 дерева разбора листинга программы

5	B2	Обращение к аппаратно-программному модулю B2 "Разворачивания структур". Работа данного модуля раскрыта на схеме, изображенной на фиг.11. Вход 1: прием сигнала управления Выход 1: передача сигнала управления Выход 2: обращение к базе данных D5 таблиц типов языка программирования
	B3	Обращение к аппаратно-программному модулю B3 "Разворачивания выражений". Работа данного модуля раскрыта на схеме, изображенной на фиг.12. Вход 1: прием сигнала управления Выход 1: передача сигнала управления
10	B4	Обращение к аппаратно-программному модулю B4 "Преобразования к внутреннему представлению". Работа данного модуля раскрыта на схеме, изображенной на фиг.13. Вход 1: прием сигнала управления Выход 1: передача сигнала управления Выход 2: сохранение в базе данных D7 программы на языке внутреннего представления Выход 3: сохранение базы условий корректности в базе данных D8 Выход 4: сохранение базы соответствия условий корректности подозрительным точкам исходного кода в базе данных D9 Выход 5: обращение к базе знаний D6 аннотаций внешних функций
15	D4	База данных D4 дерева разбора листинга программы Вход 2: обращение к дереву разбора листинга программы
20	D5	База данных D5 таблиц типов языка программирования Вход 2: обращение к дереву разбора листинга программы
	D6	База знаний D6 аннотаций внешних функций Вход 1: обращение к базе аннотаций
	D7	База данных D7 программ на языке внутреннего представления Вход 1: сохранение программы на языке внутреннего представления
25	D8	Базы данных D8 условий корректности языка внутреннего представления исходного кода программы Вход 1: сохранение базы условий корректности
	D9	База знаний D9 условий проверки корректности подозрительных точек исходного кода программы Вход 1: сохранение базы соответствия условий корректности подозрительным точкам исходного кода

На фиг.9 приведен пример реализации модуля B1 АПБ-В. Аппаратно программный модуль B1 (встраивание функций) обеспечивает режим модифицирования дерева разбора, которое находится в базе данных D4. Перечень принятых обозначений имеет следующий вид:

	Обозначение	Описание
35		Проверка условия: есть ли необработанные локальные переменные.
	B1.1	Выход 1: да Выход 2: нет Выход 3: обращение к дереву разбора программы
40	B1.2	Убрать вершину, помеченную как объявление переменной x, из детей вершины, помеченной как объявление функции f, добавить к дереву вершину, помеченную как объявление переменной x@f.
	B1.3	Для всех потомков вершины-ребенка функции f, которые помечены
	Обозначение	Описание
	B1.4	как идентификаторы x заменить их на идентификаторы x@f.Вызов процедуры UF (разворачивание функций) для дерева разбора с выделенной вершиной, соответствующей функции main. Вход 1: прием сигнала управления
45	D4	Выход 1: передача сигнала управления База данных дерева разбора листинга программы программы Вход 2: обращение к дереву разбора программы

На фиг.10 приведен пример реализации процедуры разворачивания функций (Unwinding Functions - UF). На вход процедуры подается дерево разбора с выделенной вершиной, соответствующей некоторой функции, на выходе получается дерево разбора. На фиг.10 приняты следующие обозначения:

	Обозначение	Описание
5	UF.1	Добавить в стек (ОЗУ) вызовов имя вызываемой функции. Проверить условие: есть ли необработанные вызовы функций.
	UF.2	Выход 1: да Выход 2: нет
	UF.3	Взять следующий необработанный вызов функции f. Проверить условие: лежит f в стеке вызовов.
	UF.4	Выход 1: да Выход 2: нет
	UF.5	Добавить переменную того же типа, что и возвращаемое значение функции f. Проверить условие: является ли вызов функции ребенком цикла, условной конструкции или тела функции?
10	UF.6	Выход 1: да Выход 2: нет
	UF.7	Заменить вершину, соответствующую вызову идентификатором временной переменной.
	UF.8	Вызов процедуры UF с помеченной вершиной, соответствующей функции f.
15		Описание
	Обозначение	Вход 1: прием сигнала управления Выход 1: передача сигнала управления
	UF.9	Вставить вместо вызова f ее текст.
	UF.10	Добавить в конец текста f метку и заменить во вставленном тексте вершины return на переход на эту метку.
20	UF.11	Добавить перед вставленным текстом f присваивания аргументов из вызова параметрам функции.

На фиг.11 приведен пример реализации модуля В2 АПБ-В (аппаратно-программный модуль разворачивания структур). На вход модуля В2 подается дерево разбора и таблица типов, на выходе получают модифицированное дерево разбора. Перечень принятых обозначений имеет следующий вид:

	Обозначение	Описание
30		Проверка условия: есть ли необработанные структурные типы. Вход 1: прием сигнала управления
	B2.1	Выход 1: да Выход 2: нет Выход 3: обращение к таблице типов
	B2.2	Взять следующий необработанный тип t с полями a ₀ ,..., a _n с типами t ₀ ,..., t _n .
	B2.3	Заменить каждое объявление переменной x типа t на объявления переменных x.a ₀ ,...,x.a _n типа t ₀ ,...,t _n .
	B2.4	Для каждого структурного типа s с полем x типа t заменить на поля s.x.a ₀ ,..., s.x.a _n типа t ₀ ,..., t _n .
35	B2.5	Каждое присваивание переменных x=y типа t заменить на семейство присваиваний x.a ₀ =y.a ₀ ,..., x.a _n =y.a _n
	D5	База данных D5 таблиц типов языка программирования Вход 2: обращение к таблице типов

На фиг.12 приведен пример реализации модуля В3 АПБ-В. На вход аппаратно-программного модуля В3 (модуль разворачивания выражений) подается преобразованное дерево разбора, а на его выходе получается новое преобразованное дерево разбора. Перечень принятых обозначений имеет следующий вид:

	Обозначение	Описание
45	V3.1	Проверить условие: есть ли необработанные циклы. Выход 1: да Выход 2: нет
		Вместо вершины? отвечающей циклу? дописывается список ее детей, в начале и в конце добавляются метки. Потомки вершины-цикла, помеченные continue заменяются на goto к метке в начале цикла, потомки break заменяются на goto к метке в конце цикла. Перейти к следующему циклу.
50	V3.3	Проверить условие: есть ли необработанные условные конструкции. Выход 1: да Выход 2: нет

- V3.4 Добавить три метки. Заменить условную конструкцию на cond-goto, с тем же условием, что и условная конструкция, которая переходит на первую метку в случае истинности условия и на вторую в случае его ложности. После cond-goto добавить первую метку, после нее добавить детей потмока, помеченного true-branch, после чего добавить вторую метку и добавить детей потомка, помеченного false-branch.
- V3.5 Проверить условие: присутствуют ли обращения к массивам внутри арифметических выражений.
- 5 Выход 1: да
Выход 2: нет
- V3.6 Добавить переменную, соответствующую типу элемента массива.
- V3.7 Добавить присваивание временной переменной элемента массива и заменить обращение к массиву на временную переменную.

10 На фиг.13 приведен пример реализации модуля В4 АПБ-В (аппаратно-программный модуль преобразования к внутреннему представлению). На вход модуля В4 подается дерево в виде преобразованного дерева разбора исходного кода и база знаний D6 аннотаций внешних функций, на выходе модуля В4 получают
15 базу данных D7 программ на языке внутреннего представления, базу данных D8 условий корректности языка внутреннего представления исходного кода программы и базу знаний D9 условий проверки корректности подозрительных точек исходного кода программы. Перечень принятых обозначений имеет следующий вид:

20	Обозначение	Описание
	B4.1	Установить счетчик меток в ноль. Сделать таблицу меток пустой. Таблица меток представляет собой таблицу соответствия символьных меток, встречающихся в дереве, поступающему на вход алгоритму, числовым меткам, которыми будут помечены все инструкции внутреннего представления (метки имеют вид Lxx, где xx - номер инструкции). Счетчик меток содержит последний использованный числовой идентификатор.
25	B4.2	Проверка условия: есть ли необработанные дети тела функции main. Выход 1: да Выход 2: нет
	B4.3	Взять следующую необработанную вершину. Определить тип вершины. Выход 1: присваивание
30	B4.4	Выход 2: переход Выход 3: метка Выход 4: вызов функции Вызвать процедуру АТ (преобразования присваиваний) для поддеревя с корнем в избранной вершине. Вход 1: прием сигнала управления Выход 1: передача сигнала управления
35	B4.5	Выход 2: модификация программы на языке внутреннего представления Выход 3: пополнение базы данных D8 условий корректности языка внутреннего представления исходного кода программы Выход 4: пополнение базы знаний D9 условий проверки корректности подозрительных точек исходного кода программы подозрительными точкам исходного кода Вызвать процедуру GT (преобразования переходов) для поддеревя с корнем в избранной вершине.
40	B4.6	Вход 1: прием сигнала управления Выход 1: передача сигнала управления Выход 2: модификация программы на языке внутреннего представления
	B4.7	Добавить в таблицу меток связь символьной метки, которой помечена вершина с текущим счетчиком меток. Проверить условие: присутствует ли функция в базе аннотаций.
45	B4.8	Выход 1: да Выход 2: нет
	Обозначение	Описание

50

Из базы аннотаций D6 взять код функции, произвести подстановку меток, к номерам меток, присутствующим в базе добавить текущий номер метки, после этого увеличить его на число инструкций. Произвести подстановку параметров, которыми заменить метки вершин-детей обрабатываемой вершины, начиная со второй, меткой последней вершины-ребенка заменить выходное значение. Для параметров-массивов произвести подстановку ассоциированных переменных.

5	B4.9	Вход 1: прием сигнала управления
		Выход 1: передача сигнала управления
		Выход 2: модификация программы на языке внутреннего представления
		Выход 3: пополнение базы условий корректности
10	B4.10	Выход 4: пополнение базы соответствия условий корректности подозрительным точкам исходного кода
		Перейти к следующей вершине.
		Проверить условие: есть ли условные или безусловные переходы на символьные метки, а не на метки вида Lxx.
		Выход 1: да
15	B4.11	Выход 2: нет
		Выход 3: обращение к базе знаний аннотаций D6
		Взять инструкцию перехода и заменить символьную метку в соответствии с таблицей меток.
		База знаний D6 аннотаций внешних функций
20	D6	Вход 1: обращение к базе аннотаций
		База данных D7 программ на языке внутреннего представления
		Вход 1: сохранение программы на языке внутреннего представления
		Базы данных D8 условий корректности языка внутреннего представления исходного кода программы
25	D7	Вход 1: сохранение базы условий корректности
		База знаний D9 условий проверки корректности подозрительных точек исходного кода программы
		Вход 1: сохранение базы соответствия условий корректности подозрительным точкам исходного кода

На фиг.14 приведен пример реализации процедуры преобразования присваиваний (Assign Transform - AT). На вход процедуры AT подается вершина дерева, соответствующая присваиванию, на выходе процедуры получают инструкции внутреннего представления. На фиг.14 приняты следующие обозначения:

Обозначение	Описание
30	Проверить условие: является ли левый операнд а обращением к массиву.
	Выход 1: да
	Выход 2: нет
	Преобразовать индекс массива а к выражению во внутреннем представлении х процедурой SAE.
35	Вход 1: прием сигнала управления
	Выход 1: передача сигнала управления
	Проверить условие: является ли правый операнд b обращением к массиву.
	Выход 1: да
40	Выход 2: нет
	Преобразовать правый операнд к выражению во внутреннем представлении у процедурой SAE.
	Вход 1: прием сигнала управления
	Выход 1: передача сигнала управления
45	Передать следующему шагу инструкцию внутреннего представления assign a[a[x]]=y, условие корректности x+a@offset<a@size и соответствие условий корректности точке в исходном коде и увеличить счетчик меток.
	Добавить временную переменную t.
	Передать следующему шагу инструкцию внутреннего представления assign a t=a[x], условие корректности x+a@offset<a@size и соответствие условий корректности точке в исходном коде и увеличить счетчик меток.
	Преобразовать индекс массива b к выражению во внутреннем представлении у процедурой SAE.
50	Вход 1: прием сигнала управления
	Выход 1: передача сигнала управления
	Передать следующему шагу инструкцию внутреннего представления assign b[y]=t, условие корректности y+b@offset<b@size и соответствие условий корректности точке в исходном коде и
	увеличить счетчик меток.
50	Проверить условие: является ли правый операнд b обращением к массиву.
	Выход 1: да
	Выход 2: нет

		Преобразовать индекс массива b к выражению во внутреннем представлении x с помощью процедуры SAE.
AT.11		Вход 1: прием сигнала управления Выход 1: передача сигнала управления
AT.12		Передать следующему шагу инструкцию внутреннего представления a assign a=b[x], условие корректности x+b@offset<b@size и соответствие условий корректности точке в исходном коде и увеличить счетчик меток.
5		Преобразовать правый операнд к выражению во внутреннем представлении x с помощью процедуры SAE.
AT.13		Вход 1: прием сигнала управления Выход 1: передача сигнала управления
AT.14		Передать следующему шагу инструкцию внутреннего представления assign a=x и увеличить счетчик меток.
10		Модификация программы на языке внутреннего представления, запись в базу условий корректности и базу соответствия условий корректности подозрительным точкам исходного кода в соответствии с переданными данными.
AT.15		Вход 1: прием сигнала управления Выход 1: передача управления Выход 2: модификация программы на языке внутреннего представления Выход 3: пополнение базы условий корректности Выход 4: пополнение базы соответствия условий корректности подозрительным точкам исходного кода
15		Модификация программы на языке внутреннего представления, запись в базу условий корректности и базу соответствия условий корректности подозрительным точкам исходного кода в соответствии с переданными данными.
AT.16		Выход 1: передача сигнала управления Выход 2: модификация программы на языке внутреннего представления Выход 3: пополнение базы условий корректности Выход 4: пополнение базы соответствия условий корректности
20		
	Обозначение	Описание
		подозрительным точкам исходного кода
		База данных D7 программ на языке внутреннего представления
D7		
25		Вход 1: модификация программы на языке внутреннего представления
D8		Базы данных D8 условий корректности языка внутреннего представления исходного кода программы
		Вход 1: пополнение базы условий корректности
D9		База знаний D9 условий проверки корректности подозрительных точек исходного кода программы
		Вход 1: пополнение базы соответствия условий корректности подозрительным точкам исходного кода

30 На фиг.15 приведен пример реализации процедуры преобразования переходов (Goto Transform - GT). На вход процедуры подается вершина, соответствующая условному/безусловному переходу, на выходе процедуры получается набор инструкций внутреннего представления. На фиг.15 приняты следующие обозначения:

35	Обозначение	Описание
		Проверить условие: является ли переход условным.
	GT.1	Выход 1: да Выход 2: нет
40		Проверить условие: имеет ли условие вид сравнения ("равно", "больше" и т.п.)
	GT.2	Выход 1: да Выход 2: нет
		Преобразовать левый операнд в выражение во внутреннем представлении x процедурой SAE.
	GT.3	Вход 1: прием сигнала управления Выход 1: передача сигнала управления
45	GT.4	Преобразовать левый операнд в выражение во внутреннем представлении y процедурой SAE.
	Обозначение	Описание
		Вход 1: прием сигнала управления Выход 1: передача сигнала управления
		Проверить условие: равны ли x или y unknown .
50	GT.5	Выход 1: да Выход 2: нет
	GT.6	Задать условие condition unknown .
	GT.7	Задать условие x предикат y.

		Добавить инструкцию condition с заданным условием и увеличить на единицу счетчик меток.
5	GT.8	Выход 1: передача сигнала управления Выход 2: модификация программы на языке внутреннего представления Добавить во внутреннее представление инструкцию goto и увеличить на единицу счетчик меток.
	GT.9	Выход 1: передача сигнала управления Выход 2: модификация программы на языке внутреннего представления
	D7	База данных D7 программ на языке внутреннего представления Вход 1: модификация программы на языке внутреннего представления

10 На фиг.16 приведен пример реализации процедуры преобразования выражений (Simple Arithmetic Expression transform - SAE). На вход процедуры подается вершина, соответствующая арифметическому выражению, на выходе - получают выражение на языке внутреннего представления. На фиг.16 приняты следующие обозначения:

15	Обозначение	Описание
	SAE.1	Проверка условия: является ли вершина является операцией. Выход 1: да Выход 2: нет
20	SAE.2	Преобразовать левый операнд в выражение во внутреннем представлении x процедурой SAE. Вход 1: прием сигнала управления Выход 1: передача сигнала управления
	Обозначение	Описание
	SAE.3	Преобразовать правый операнд в выражение во внутреннем представлении y процедурой SAE. Вход 1: прием сигнала управления
25	SAE.4	Выход 1: передача сигнала управления Проверка условия: равны ли x или y unknown . Выход 1: да Выход 2: нет
	SAE.5	Вернуть unknown .
30	SAE.6	Вернуть x операнд y.
	SAE.7	Тип переменной x, которой помечена вершина численный? Выход 1: да Выход 2: нет
	SAE.8	Вернуть unknown .
35	SAE.9	Вернуть идентификатор переменной, которой помечена вершина.

На фиг.17 приведен пример реализации аппаратно-программного блока анализа кода (АПБ-С) программы на языке внутреннего представления и формирование отчета об обнаруженных "уязвимостях", при этом приняты следующие обозначения:

40	Обозначение	Описание
	C1	Обращение к аппаратно-программному модулю C1 (получение инвариантов в точках внутреннего представления) Вход 1: прием сигнала управления Выход 1: передача сигнала управления Выход 2: считывание инструкций программы на языке внутреннего представления
45	C2	Обращение к аппаратно-программному модулю C2 (генерация систем информационной базы) Вход 1: прием сигнала управления Выход 1: передача сигнала управления Выход 2: считывание базы условий корректности Выход 3: сохранение информационной базы
50	C3	Обращение к аппаратно-программному модулю C3 (исследование систем ограничений информационной базы)
	Обозначение	Описание
		Вход 1: прием сигнала управления Выход 1: передача сигнала управления Выход 2: считывание систем ограничений информационной базы

		Обращение к аппаратно-программному модулю C4 (формирование отчета о возможных уязвимостях)
		Вход 1: прием сигнала управления
	C4	Выход 1: передача сигнала управления
		Выход 2: считывание записей базы соответствия условий корректности подозрительным точкам исходного кода
5		Выход 3: вывод отчета об обнаруженных уязвимостях
	D7	База данных D7 программ на языке внутреннего представления
		Вход 2: обращение к программе на языке внутреннего представления
	D8	Базы данных D8 условий корректности языка внутреннего представления исходного кода программы
		Вход 2: обращение к базе условий корректности
10	D9	База знаний D9 условий проверки корректности подозрительных точек исходного кода программы
		Вход 2: обращение к базе соответствия условий корректности подозрительным точкам исходного кода
		Информационная база знаний (D10), содержащая системы ограничений в виде алгебраических уравнений и неравенств
	D10	Вход 1: сохранение информационной базы
		Вход 2: извлечение систем ограничений информационной базы
15	D11	База данных (D11) отчетов об обнаруженных уязвимостях программного кода
		Вход 1: вывод отчета об обнаруженных уязвимостях

На фиг.18-27 приведены примеры реализации основных модулей АПБ-С. На фиг.18 приведен пример реализации модуля C1 АПБ-С. Аппаратно программный модуль C1

20 обеспечивает поддержку режима "получения инвариантов в точках внутреннего представления", перечень принятых обозначений имеет следующий вид:

Обозначение	Описание
25	C1.1 Считать инструкции программы и создать граф управления программы процедурой GG (см. пояснение 1 в тексте описания).
	Вход 1: прием управляющего сигнала
Обозначение	Описание
	Выход 1: передача сигнала управления
	Выход 2: программа на языке внутреннего представления
30	C1.2 Найти максимальные простые пути, ведущие из начальной вершины графа процедурой SP (см. пояснение 2 в тексте описания).
	C1.3 Инициализация таблицы инвариантов (см. пояснение 3 в тексте описания).
	Проверка условия: есть ли необработанные простые пути.
	C1.4 Выход 1: да
	Выход 2: нет
35	C1.5 Вычислить инварианты вдоль очередного простого пути L процедурой PI (см. пояснение 4 в тексте описания).
	C1.6 Уточнить таблицу инвариантов и закончить обработку пути L (см. пояснение 5 в тексте описания).
	D7 База данных D7 программ на языке внутреннего представления
	Вход 2: обращение к программе на языке внутреннего представления

На фиг.19 приведен пример реализации процедуры определения инвариантов вдоль

40 одного простого пути (Path Invariants - PI). На ее вход поступает некоторый простой путь $L=(L[0],...,L[m-1])$ управляющего графа программы. На фиг.19 приняты следующие обозначения:

Обозначение	Описание
45	PI.1 Присваивание $p=0$
	Проверка условия: $p < m-1$
	PI.2 Выход 1: да
	Выход 2: нет
	Вычисление: $Y=YS(L, p+1)$ (см. пояснение в тексте описания).
50	PI.3 Вход 1: прием сигнала управления
	Выход 1: передача сигнала управления
	PI.4 Присваивание: $U=TP[p]$
	PI.5 Определение вида инструкции $L[p]$ (см. пояснение 6 в тексте описания).
	Обозначение Описание

		Выход 1: assign(a) x=expr
		Выход 2: condition x R const
		Выход 3: condition expr1 R expr2
		Выход 4: другой
5	PI.6	Пересчет спектра переменной x (см. пояснение 7 в тексте описания).
	PI.7	Пересчет множества предикатов U (см. пояснение 8 в тексте описания).
	PI.8	Пересчет спектра переменной x (см. пояснение 9 в тексте описания).
	PI.9	Пополнение множества предикатов U новым предикатом (см. пояснение 10 в тексте описания).
10	PI.10	Запись спектров всех переменных во временную таблицу S (см. пояснение 11 в тексте описания).
		Фильтрация спектров всех переменных и всех предикатов с помощью процедуры YF (см. пояснение 12 в тексте описания).
	PI.11	Вход 1: прием сигнала управления
15		Выход 1: передача сигнала управления
	PI.12	Заполнение (p+1)-го столбца таблицы спектров и (p+1)-го элемента таблицы предикатов (см. пояснение 13 в тексте описания).
	PI.13	Присваивание p=p+1

20 На фиг.20 приведен пример реализации процедуры фильтрации спектров и предикатов (Y-Filter-YF). На фиг.20 приняты следующие обозначения:

	Обозначение	Описание
		Проверка условия: есть ли в множестве Y необработанные инструкции?
25	YF.1	Выход 1: да Выход 2: нет
		Определение вида очередной инструкции y из множества Y (см. пояснение 14 в тексте описания).
	YF.2	Выход 1: assign x=x+const Выход 2: assign x=const Выход 3: assign(a) x= ...
30		Выход 4: другой
		Сравнение const, 0
	YF.3	Выход 1: const<0 Выход 2: const>0
	Обозначение	Описание
35		Проверка условия: ограниченности S[x] сверху (или sup S[x])?
	YF.4	Выход 1: да Выход 2: нет
		Проверка условия: ограниченности S[x] снизу (или inf[x])?
	YF.5	Выход 1: да Выход 2: нет
40	YF.6	Присваивание: S[x]=(-inf, +inf)
	YF.7	Присваивание: S[x]=(-inf, max(S[x]))
	YF.8	Присваивание: S[x]=(min(S[x]), +inf)
	YF.9	Объединение (join) спектр переменной x со спектром {const} (см. пояснение 15 в тексте описания).
	YF.10	Удалить из множества U все предикаты, куда входит переменная x (см. пояснение 16 в тексте описания).
45	YF.11	Окончание обработки инструкции y.

На фиг.21 приведен пример реализации процедуры вычисления множества Y (Y-Set-YS). На фиг.21 приняты следующие обозначения:

50	Обозначение	Описание
	YS.1	Очистка множеств A, B, C, D.
	YS.2	Присваивание A=A∪B.
	YS.3	Инициализация множества B.

		Проверка условия: есть необработанные элементы в A.
	УЗД	Выход 1: да Выход 2: нет
5	YS.5	Взять из A очередную необработанную вершину и пополнить множество B, закончить обработку вершины (см. пояснение 17 в тексте описания). Проверка $B \subseteq A$.
	YS.6	Выход 1: да Выход 2: нет
	YS.7	Присваивание $C = C \cup D$.
10	YS.8	Инициализация множества D. Проверка, есть необработанные элементы в C?
	YS.9	Выход 1: да Выход 2: нет
	YS.10	Взять из C очередную необработанную вершину v, пополнить множество D, закончить обработку вершины v (см. пояснение 18 в
15	Обозначение	Описание
		тексте описания). Проверка $D \subseteq C$.
	YS.11	Выход 1: да Выход 2: нет
20	YS.12	Присваивание $Y = A \cap C$.

На фиг.22 приведен пример реализации процедуры определения максимально простых путей в графе (Simple Paths-SP). На фиг.22 приняты следующие обозначения:

	Идентификатор	Описание
25	SP.1	Инициализировать очереди Q, R (см. пояснение 19 в тексте описания). Проверка условия: пуста ли очередь Q.
	SP.2	Выход 1: да Выход 2: нет
	SP.3	Взять путь L из начала очереди Q.
30	SP.4	Присваивание $max = false$
	SP.5	Взять v - конец пути L и A - множество соседних с v вершин. Проверка условия: есть в A необработанные вершины.
	SP.6	Выход 1: да Выход 2: нет Определить значение max.
35	SP.7	Выход 1: true Выход 2: false
	SP.8	Добавить путь L в очередь R.
	SP.9	Взять очередную вершину и из A. Проверка условия: проходит ли путь L через и.
40	SP.10	Выход 1: да Выход 2: нет
	SP.11	Присваивание $L' = (L, u)$ (см. пояснение 20 в тексте описания)
	SP.12	Добавить путь L' в конец очереди Q.
45	SP.13	Присвоить $max = true$

На фиг.23 приведен пример реализации процедуры генерации управляющего графа программы (Generate Graph - GG). На фиг.23 приняты следующие обозначения:

	Обозначение	Описание
50		Считать инструкции внутреннего представления в массив P; N - их число.
	GG.1	Выход 1: передача сигнала управления Выход 2: считывание инструкций программы на языке внутреннего представления
	GG.2	Присвоить $i = 0$.

		Проверка условия $i < N$.
	GG.3	Выход 1: да Выход 2: нет
	GG.4	Создать в графе G новую вершину с пометкой P[i].
5	GG.5	Взять id - идентификатор инструкции P[i] и добавить пару (id, i) в словарь D.
	GG.6	$i = i + 1$
	GG.7	$i = 0$
		Проверка условия $i < N$.
	GG.8	Выход 1: да Выход 2: нет
10		Определить вид инструкции P[i] (см. пояснение 21 в тексте описания). выход 1: инструкция P[i] имеет тип assign, a assign, assign a
	GG.9	Выход 2: инструкция P[i] имеет вид goto id; Выход 3: инструкция P[i] имеет вид condition...? Id1: id2; Выход 4: инструкция P[i] имеет тип stop
15	GG.10	Определить значения j1, j2, соответствующие ключам id1, id2 в словаре D.
	GG.11	Создать в графе G пару ребер (i, j1) и (i, j2).
	GG.12	Определить значение j, соответствующее ключу id в словаре D.
	GG.13	Создать в графе G ребро (i, j).
	GG.14	Создать в графе G ребро (i, i+1).
20	GG.15	Присваивание $i = i + 1$
	D7	База данных D7 программ на языке внутреннего представления Вход 2: считывание инструкций программы на языке внутреннего представления

На фиг.24 приведен пример реализации модуля C2 АПБ-С. Аппаратно программный модуль C2 обеспечивает режим генерации систем информационной базы знаний D10 и позволяет свести задачу проверки условий корректности в программе на языке внутреннего представления к стандартной задаче линейного программирования. Перечень принятых обозначений имеет следующий вид:

Обозначение	Описание
C2.1	Считать условия корректности в массив CC. Вход 1: передача сигнала управления Вход 2: считывание базы данных условий корректности условия корректности языка внутреннего представления исходного кода программы (D8) Блок проверки условия: если ли в массиве CC необработанные условия корректности.
C2.2	Выход 1: да Выход 2: нет
C2.3	Взять из массива CC очередное условие корректности (P, id) (см. пояснение 22).
C2.4	Взять из таблицы инвариантов TI множество условий W, соответствующих вершине управляющего графа, помеченной инструкцией с идентификатором id.
40	Сформировать очередную систему информационной базы: $W_A \rightarrow P$.
C2.5	Выход 1: передача сигнала управления Выход 2: пополнение информационной базы новой системой ограничений
C2.6	Закончить обработку условия корректности (P, id).
D8	Базы данных условий корректности условия корректности языка внутреннего представления исходного кода программы (D8)
45	Вход 2: считывание базы условий корректности
D10	Информационная база знаний (D10), содержащая системы ограничений в виде алгебраических уравнений и неравенств Вход 1: сохранение пополненной информационной базы знаний

На фиг.25 приведен пример реализации модуля C3 АПБ-С. Аппаратно-программный модуль C3 обеспечивает режим исследования систем ограничений информационной базы знаний D10. Перечень принятых обозначений на фиг.25 имеет следующий вид:

	Обозначение	Описание
		Считывание систем информационной базы знаний D10 в массив Sys.
	C3.1	Выход 1: передача сигнала управления
		Выход 2: считывание систем ограничений информационной базы
5	C3.2	Формирование набора из t параллельно выполняемых задач (см. пояснение 23).
	Обозначение	Описание
		Выход 1: передача сигнала управления процессу #1
		Выход 2: передача сигнала управления процессу #2
		Выход t : передача сигнала управления процессу # t
		Проверка условия: есть необработанные системы для процесса # i ?
10	C3.3	Выход 1: да
		Выход 2: нет
	C3.4	Взять очередную систему S
		Проверка условия: в S есть тождественно ложные предикаты или пары противоречащих друг другу предикатов(см. пояснение 24 в тексте описания)?
15	C3.5	Выход 1: да
		Выход 2: нет
	C3.6	Приведение системы S к виду задачи LP - классического линейного программирования (см. пояснение 25 в тексте описания).
		Симплекс-метод для решения задачи LP с помощью процедуры SIMPLEX-METHOD (см. пояснение 26 в тексте описания)
20	C3.7	Вход 1: прием сигнала управления
		Выход 1: передача сигнала управления
		Сравнить L^* , 0.
	C3.8	Выход 1: $L^* < 0$
		Выход 2: иначе
25	C3.9	Добавить номера системы S в список DNG.
	C3.10	Закончить обработку системы S.
		Синхронизовать процессы #1,..., # t .
	C3.11	Вход 1: передача сигнала управления от процесса #1
		Вход 2: передача сигнала управления от процесса #2
		Вход t : передача сигнала управления от процесса # t
30		Информационная база знаний (D10), содержащая системы ограничений в виде алгебраических уравнений и неравенств
	D10	Выход 2: считывание систем ограничений информационной базы

На фиг.26 приведен пример реализации процедуры решения задачи линейного программирования симплекс-методом (Simplex Method - SM). На фиг.26 приняты следующие обозначения:

	Обозначение	Описание
	SM.1	Инициализация массивов E, B, D.
40		Проверка условия: есть в массиве D отрицательные элементы?
	SM.2	Выход 1: да
		Выход 2: нет
	SM.3	Определение $\min \{D[i]\}$ и выбор вектора A[k], включаемого в базис.
	SM.4	Определение вектора A[B[r]] для исключения из базиса.
45	SM.5	Пересчет массивов E, B для нового базиса (см. пояснение 27 в тексте описания).
	SM.6	Пересчет массива D для текущей итерации (см. пояснение 28 в тексте описания).
	SM.7	Вычислить L^* - значение линейной формы на плане X.

На фиг.27 приведен пример реализации модуля C4 АПБ-С. Аппаратно программный модуль C4 обеспечивает режим формирования отчета об обнаруженных уязвимостях. Перечень принятых обозначений на фиг.27 имеет следующий вид:

Обозначение	Описание
-------------	----------

	Проверка условия: есть необработанные номера в списке DNG?
C4.1	Выход 1: да Выход 2: нет
C4.2	Взять очередной номер d из DNG.
5	Извлечь из базы соответствия условий корректности подозрительным точкам исходного кода запись #d и выдать ее на экран.
C4.3	Выход 1: передача сигнала управления Выход 2: считывание записи #d из базы D9 соответствия условий корректности подозрительным точкам исходного кода Выход 3: выдача на экран отчета об уязвимостях
C4.4	Закончить обработку номера d
10	База знаний D9 условий проверки корректности подозрительных точек исходного кода программы
D9	Вход 2: считывание записи #d из базы соответствия условий корректности подозрительным точкам исходного кода
D11	База данных отчетов об обнаруженных уязвимостях программного кода (D11)
15	Вход 1: выдача на экран отчета об уязвимостях

Далее приведены пояснения к обозначениям, использованным для фиг.18-27.

1. На данном шаге происходит считывание инструкций внутреннего представления программы в массив Р. Число инструкций (длина массива) предполагается равным N. Затем осуществляется вызов GG - вызов функции построения управляющего графа программы, которой передаются на вход массив Р и его размер N.

2. Вызов функции поиска максимальных простых путей в управляющем графе, начинающихся из начальной его вершины.

3. Инициализация таблицы инвариантов. Данный шаг заключается в выделении массива размера N, значениями которого являются множества инвариантов, причем i-й элемент массива будет содержать множество инвариантов, полученное в виде вершины графа с индексом i. На данном шаге множества инвариантов во всех элементах массива пусты.

4. Вызов функции PI - поиска инвариантов вдоль простого пути. На вход функции передается управляющий граф и простой путь в виде последовательности принадлежащих ему вершин. На выходе функция возвращает таблицу спектров и таблицу предикатов вдоль данного простого пути.

5. Процедура уточнения таблицы инвариантов.

- Сначала для каждой вершины v простого пути L берется объединенный список X предикатов таблиц TS(L), TP(L), соответствующих вершине v - затем, если вершина v уже обрабатывалась ранее на другом простом пути L', то из элемента таблицы TI, соответствующего вершине v, выбрасываются все предикаты, которые не встретились в списке X;

- если вершина v обрабатывается впервые, то в элемент таблицы TI, соответствующий вершине v, добавляются все предикаты из списка X.

6. Вычислить $Y=Y(L, p+1)$. В данной функции происходит определение множества вершин v графа, таких что существует пара путей L', L'', не проходящих через вершины L[0], ..., L[p], причем путь L' соединяет вершину L[p+1] с вершиной v, путь L'' соединяет вершину v с вершиной L[p+1].

7. Рассматривается инструкция L[p], выделяются 4 основных случая.

- Инструкция имеет вид assign x=expr, либо assign_a x=...[...], где x - имя переменной, expr - некоторое арифметическое выражение.

- Инструкция имеет вид condition x R const, где x - имя переменной, R - один из предикатных символов {<, <=, >, >=, ==, !=}, const - некоторое целое число.

- Инструкция имеет вид condition expr1 R expr2, где expr1, expr2 - некоторые

арифметические выражения, R - один из предикатных символов $\{<, <=, >, >=, ==, !=\}$.

- Другой вид инструкции ($a_assign, goto, stop$).

8. Пересчитать спектр переменной x , подставив в арифметическое выражение $expr$ вместо переменных, входящих туда, их спектры из таблицы TS , столбца p . В

результате получим новый спектр переменной x . Если инструкция имеет тип $assign_a$, то считаем, что спектр правой части равен $(-inf, +inf)$.

9. Пересчитать множество предикатов U . Поскольку переменная x изменилась, то условия, содержащие x , которые были выполнены до инструкции $L[p]$, перестанут быть выполнены после инструкции $L[p]$. В некоторых простейших случаях возможно скорректировать условие таким образом, что после инструкции $L[p]$ оно вновь станет выполнено. Укажем эти условия.

- Пусть $expr$ имеет вид $x+const$. Здесь $const$ - некоторое целое число.

- Пусть предикат u имеет вид $a * x + w R c1$, где $a, c1$ - некоторые целые числа, w - предикат, не зависящий от x , R - один из предикатных символов $\{<, <=, >, >=, ==, !=\}$.

В этом случае можно скорректировать предикат следующим образом:

$u' = a * x + w R c2$. Здесь $c2 = c1 + const * a$. Если инструкция или предикат не удовлетворяют указанным условиям, то предикат необходимо из множества удалить.

10. Пересчитать спектр переменной x . Поскольку предикат, стоящий в инструкции $L[p]$ имеет вид $x R const$, то ему всегда соответствует некоторый спектр s , выражающий то же самое условие. Например, предикату $x \leq 10$ будет соответствовать спектр $(-inf, 11)$, а предикату $x \neq 0$ будет соответствовать спектр $(-inf, 0) \cup [1, +inf)$. Тогда текущий спектр переменной x нужно пересечь (с помощью операции $intersect$) со спектром s . При этом крайне важно учесть, какой ветке условного перехода отвечает ребро $(L[p], L[p+1])$. Если эта ветка соответствует ложности предиката, предикатный символ R нужно заменить на ему противоположный. Пары противоположных предикатных символов: $(<, >=)$; $(<=, >)$; $(==, !=)$.

11. Пополнить множество предикатов U новым предикатом, который записан в теле инструкции условного перехода $L[p]$. Вне зависимости от вида данного условия, оно всегда будет выполнено. При выполнении данного шага необходимо учесть истинность или ложность ветки условного перехода, отвечающей ребру $(L[p], L[p+1])$.

12. Записать спектры всех переменных из p -го столбца таблицы спектров с учетом возможных поправок спектра для отдельной переменной во временную таблицу S .

13. Произвести "фильтрацию" спектров таблицы S и предикатов множества U . В результате фильтрации будут по-возможности скорректированы, а в худшем случае - отброшены все спектры и предикаты, которые могут оказаться неверны, если учесть инструкции из множества Y , которые могут появиться на трассе выполнения программы раньше точки $L[p+1]$. Такими инструкциями будут инструкции множества Y (см. шаг 1). Блок-схема работы функции YF будет далее приведена отдельно.

14. Заполнить $(p+1)$ -й столбец таблицы спектров значениями, записанными во временной таблице S и $(p+1)$ -й элемент таблицы предикатов положить равным U .

15. Рассматривается очередная инструкция u , выделяются 4 случая.

А. Инструкция u имеет вид $assign\ x = x + const$, где $const$ - некоторое целое число.

В. Инструкция u имеет вид $assign\ x = const$, где $const$ - некоторое целое число.

С. Инструкция u имеет вид $assign\ x = \dots$

Д. Инструкция u имеет другой вид (тип отличен от $assign$).

16. Объединить при помощи операции $join$ текущий спектр переменной x (из массива S) со спектром $\{const\}$.

17. Удалить из множества U все предикаты, куда входит переменная x .

18. Взять из A очередную необработанную вершину u , пополнить множество B вершинами, смежными с u , но не принадлежащими множеству $\{L[0], \dots, L[p-1]\}$, закончить обработку вершины u .

19. Взять из C очередную необработанную вершину u , пополнить множество D вершинами, не принадлежащими множеству $\{L[0], \dots, L[p-1]\}$, для которых вершина v - смежная, закончить обработку вершины v .

20. Инициализировать очереди Q, R . Очередь R изначально пуста. В очередь Q положить последовательность $l = \{v_0\}$, где v_0 - начальная вершина графа (то есть вершина, входящая степень которой равна 0).

21. $L' = (L, u)$. Данное выражение следует понимать следующим образом.

Путь L' получается из пути L добавлением вершины u .

22. Взять из массива CC очередное условие корректности, которое представляет собой пару вида (P, id) , где P - предикат от переменных внутреннего представления, а id - идентификатор инструкции внутреннего представления, перед которой должен быть истинен предикат P .

23. Формирование набора t параллельно выполняемых задач происходит путем разделения массива Sys на t примерно равных частей (частей, отличающихся не более чем на 1 систему). Затем каждый процесс будет заниматься исследованием своего подмножества информационной базы.

24. Проверка того, есть ли в системе S тождественно ложные предикаты или пары противоречащих друг другу предикатов.

25. Приведение системы S к виду LP - задачи классического линейного программирования. Ограничения информационной базы являются алгебраическими уравнениями и неравенствами. Для упрощения дальнейших рассуждений выбросим из системы все предикаты, в которых степень многочлена превышает 1. После этого систему S тривиальными арифметическими преобразованиями можно будет привести к следующему виду:

$$a(1,1) \cdot x(1) + a(1,2) \cdot x(2) + \dots + a(1,n) \cdot x(n) \leq b(1)$$

...

$$a(q,1) \cdot x(1) + a(q,2) \cdot x(2) + \dots + a(q,n) \cdot x(n) \leq b(q)$$

$$a(q+1,1) \cdot x(1) + a(q+1,2) \cdot x(2) + \dots + a(q+1,n) \cdot x(n) = b(q+1)$$

...

$$a(m,1) \cdot x(1) + a(m,2) \cdot x(2) + \dots + a(m,n) \cdot x(n) = b(m).$$

Вводятся дополнительные неотрицательные переменные:

$$x(n+1), \dots, x(n+q), x'(1), x''(1), \dots, x'(n), \dots, x''(n) \geq 0$$

так, чтобы выполнялись условия $x(i) = x'(i) - x''(i)$.

Тогда систему S можно переписать в таком виде:

$$a(1,1) \cdot x'(1) - a(1,1) \cdot x''(1) + \dots + a(1,n) \cdot x'(n) - a(1,n) \cdot x''(n) + x(n+1) = b(1)$$

...

$$a(q,1) \cdot x'(1) - a(q,1) \cdot x''(1) + \dots + a(q,n) \cdot x'(n) - a(q,n) \cdot x''(n) + x(n+q) = b(q)$$

$$a(q+1,1) \cdot x'(1) - a(q+1,1) \cdot x''(1) + \dots + a(q+1,n) \cdot x'(n) - a(q+1,n) \cdot x''(n) = b(q+1)$$

...

$$a(m,1) \cdot x'(1) - a(m,1) \cdot x''(1) + \dots + a(m,n) \cdot x'(n) - a(m,n) \cdot x''(n) = b(m).$$

Для простоты записи переименуем все переменные и коэффициенты таким образом, чтобы система S приняла вид:

$$t(1,1) \cdot y(1) + \dots + t(1,n') \cdot y(n') = d(1)$$

...

$$t(m',1) * y(1) + \dots + t(m',n') * y(n') = d(m')$$

$$y(i) \geq 0, i = 1, \dots, n',$$

где $m' = m$, $n' = 2 * m + q$, причем $d(i) \geq 0$.

Задача классического линейного программирования:

Рассмотрим задачу максимизации линейной формы $L(\vec{y}) = - \sum_{i=n'+1}^{n'+m'} y(i)$ при условиях

$$t(1,1) * y(1) + \dots + t(1,n') * y(n') + y(n'+1) = d(1),$$

...

$$t(m',1) * y(1) + \dots + t(m',n') * y(n') + y(n'+m') = d(m') \quad (1)$$

$$y(i) \geq 0, i = 1, \dots, n' + m'.$$

Пусть L^* - решение указанной задачи. Тогда если

- $L^* < 0$, то множество решений системы ограничений S пусто;

- $L^* = 0$, то система S имеет решения.

26. Вызов функции SIMPLEX-METHOD.

27. Рекуррентные формулы пересчета массивов E , B для нового базиса.

$$B[r] = k;$$

$$E[r][j] = E[r][j] / A[r], \text{ где } j = 1, \dots, m.$$

$$E[i][j] = E[i][j] - E[r][j] * A[i], \text{ где } j = 1, \dots, m, i = 1, \dots, m+1, i \neq r.$$

28. Формулы пересчета массива D для нового базиса.

$$D[j] = \sum_{i=1}^m E[m+1][j+1] * A[i][j] - c[j], j = 1, \dots, n.$$

На фиг.28 приведен пример реализации аппаратно-программного блока процессорного управления (АПБ-У), обеспечивающего синхронизацию основных режимов интерактивной верификации программного обеспечения распределенных вычислительных комплексов. На фиг.28 приняты следующие обозначения: У1 - блок нормирующих преобразователей (БНП); У2 - модуль коммутатора; У3 - модуль аналого-цифрового преобразователя (АЦП); У4 - модуль формирования статических координат графоаналитического раstra (МФСКГАР); У5 - модуль формирования динамической развертки графоаналитического раstra (МФДРГАР); У6 - блок видеоконтроля (БВ); У7 - импульсный регулятор (ИР).

На фиг.29 приведен пример реализации модуля формирования статических координат графоаналитического раstra (МФСКГАР) У4 АПБ-У. На фиг.29 приняты следующие обозначения: У4.1 - первое запоминающее устройство (ЗУ); У4.2 - блок интерполирования; У4.3 - блок выделения экстремума; У4.4 - блок масштабирования; У4.5 - схема формирования дискретных приращений статических координат графоаналитического раstra; У4.6 - блок преобразования статических координат графоаналитического раstra; У4.7 - второе ЗУ.

На фиг.30 приведен пример реализации модуля формирования динамических координат графоаналитического раstra (МФДРГАР) У5 АПБ-У. На фиг.30 приняты следующие обозначения: У5.1 - схема формирования дискретных приращений динамических координат графоаналитического раstra; У5.2 - блок задания постоянных коэффициентов; У5.3 - блок синхронизации; У5.4 - схема формирования адреса записи.

На фиг.31 приведен пример реализации блока видеоконтроля (БВ) У6 АПБ-У. На фиг.31 приняты следующие обозначения: У6.1 - модуль регистратора (МР) АПБ-У, на выходе которого установлены маска У6.2, оптическая система У6.3 и преобразователь

оптического сигнала У6.4, выходом подключенный к входу блока динамической развертки У6.5.

На фиг.32 приведен пример реализации импульсного регулятора У 7 АПБ-У, который содержит блок определения абсолютных значений У 7.1, первый релейный элемент У7.2, первый логический элемент "И" У7.3, второй релейный элемент У7.4, блок возведения в квадрат У7.5, триггер У7.6, первый, второй и третий блоки выборки-хранения У7.7, У7.8 и У7.9, первый формирователь импульсов У 7.10, источник эталонных напряжений У7.11, первый, второй и третий сумматоры У7.12, У7.13 и У7.14, электронный ключ У7.15, второй формирователь импульсов У7.16, третий релейные элементы У7.17, второй логический элемент "И" У7.18.

На фиг.33-39 приведены фрагменты результатов работы предлагаемого способа и устройства в процессе верификации ИК ПО распределенных вычислительных комплексов.

В качестве первого примера (пример 1) рассмотрим процесс верификации исходного текста ПО на языке программирования Си. Для ясности изложения будем полагать, что листинг исходного текста программы поступил по независимому интерфейсному каналу последовательного порта (например, введен оператором ЭВМ в ручном режиме на основе использования клавиатуры см. фиг.1, детали сопровождения (ввода, редактирования, распределенного хранения и т.п.) исходного текста ИК ПО опускаются, поскольку они в данном контексте не имеют существенного значения, т.е. в данном примере не будем рассматривать особенности процессов сопровождения ИК ПО распределенных вычислительных комплексов. Основное внимание далее уделим процессу автоматического поиска участков уязвимости в ИК ПО. При этом исходная Си программа представляет собой набор из двух файлов следующего вида:

Файл main.c:

```
extern int f(int *arr, int ind);
```

```
int main()
```

```
{
```

```
int a[100];
```

```
int i;
```

```
int j;
```

```
for(i=0; i<10; ++i)
```

```
for(j=9; j>=i; --j)
```

```
a[10*i+j]=0;
```

```
return f(a, 55);
```

```
}
```

Файл f.c:

```
int f(int *arr, int ind)
```

```
{
```

```
return arr[ind];
```

```
}
```

ИК ПО заносится в базу данных D1. После окончания ввода ИК ПО АПБ-У формирует на системной шине (см. фиг.1) управляющий сигнал в момент времени t_1 , который подается на вход 1 АПБ-А (см. фиг.2). Таким образом, АПБ-У фиксирует все моменты формирования управляющих сигналов и реакций, соответствующих АПБ и их модулей, на эти воздействия. При этом для краткости изложения далее эти моменты в описании опускаются. После поступления управляющего сигнала запускается АПБ-А, а именно (см. фиг.2) тело цикла, состоящее из аппаратно-программных

модулей А.1, А.2, А.3 АПБ-А (далее для краткости изложения будем обозначать первым символом - А. *, принадлежность каждого из используемых аппаратно-программных модулей, т.е. будем идентифицировать соответствующей буквой АПБ, а именно, АПБ-А, просто А.1, А.2 и т.д., АПБ-В - В.1 ... и т.п.). При этом этот цикл в рассматриваемом примере будет выполнен ровно 2 раза, так как в наборе анализируемых файлов Си ИК ПО в наличии 2 файла. Соответственно АПБ-У обеспечивает контроль и измерение длительности выполнения не только каждого цикла, но и длительность работы каждого модуля АПБ-А. После выполнения этого цикла будут выполнены действия по соединению результатов разбора листингов ИК ПО в модуле А.8.

Операции, производимые в модуле А.2, раскрыты на фиг.3. Опишем процессы модуля А1 на примере разбора фрагментов листинга из файла main.c.

В процессе выполнения модуля А 1.1 происходит разбиение листинга ИК ПО, получаемого из базы данных D1, на поток лексем согласно правилам из грамматики языка, находящихся в базе данных D2.

Фрагмент начала последовательности лексем с обозначением типа лексем представлен ниже:

```
'extern' - ключевое слово
'int' - ключевое слово
'f' - идентификатор
'(' - символ '('
'int' - ключевое слово
'*' - символ '*'
'arg' - идентификатор
',' - символ ','
'int' - ключевое слово
'ind' - идентификатор
')' - символ ')'
';' - символ ';'
...

```

Приведем пример фрагмента грамматики языка программирования Си (см. стандарт ISO:IEC 9899:1999, также называемой "C99").

```
...
statement:
labeled_statement
lcompound_statement
lexpression_statement
lselection_statement
literation_statement
ljump_statement
labeled_statement:
identifier ':' statement
'case' constant_expression ':' statement
'default' ':' statement
compound_statement: compound_statement_startblock_item '*'
compound_statement_start: '{'
block item: declarationstatement
expression_statement: expression '?'

```

```

selection_statement:
  'if'('expression') statement
  'if'('expression') statement
  'else' statement
5  'switch'('expression') statement
iteration_statement:
  'while'('expression ')statement
  'do' statement 'while'('expression');'
10  'for'('expression?'; 'expression?'; 'expression?') statement
  'for'('declaration expression?'; 'expression?')statement
jump_statement:
  'goto' identifier';'
  'continue';'
15  'break';'
  'return' expression?';'
translation_unit: lextemal_declaration
translation_unit: translation_unit external_declaration
20  external_declaration: declaration|function_definition
  function_definition: declaration specifiers? declarator declaration *
  compound_statement

```

Для синтаксического разбора рассматриваемого здесь примера ИК ПО, построения
 25 дерева разбора и таблицы типов потребуются следующие синтаксические правила,
 являющиеся подмножеством грамматики языка Си:

```

1: declaration: declaration_specifiers init_declarator';'
2: declaration_specifiers: (storage_class_specifier|type_specifier|type_qualifier|
function_specifier)+;
3: init_declarator: declarator;
30 4: declarator: pointer? direct_declarator;
5: direct_declarator: direct_declarator '(' parameter_list)';
6: parameter_list: parameter_declaration *;
7: direct_declarator: identifier;
35 8: parameter_declaration: declaration_specifiers declarator;
9: storage_class_specifier: 'typedef'|'extern'|'static'|'auto'|'register';
10: type_specifier: 'void'|'char'|'short'|'int'|'long'|'float'|'double'|'signed'|'unsigned'|
'Bool'|'_Complex'|'_Imaginary'|'struct_or_union_specifier'|'enum_specifier'|'typedef_name'|'typeof_ex
40 prltypeof_type;
11: pointer: '*' '+';

```

При выполнении модуля A1.2 происходит очистка временного внутреннего стека, используемого при разборе.

45 В цикле, состоящем из модулей A1.3, A1.4, A1.5, A1.6, A1.7, A1.8, A1.9, A1.10 и A1.11, производится применение синтаксических правил и построение дерева разбора и таблицы типов. В процессе выполнения последовательности A1.3, A1.4, A1.5, A1.6, A1.7 и A1.8 лексемы помещаются на вершину стека до тех пор, пока не станет возможным применение одного из правил синтаксического разбора. Соответственно
 50 длительность выполнения данных циклов и им подобных контролируется АПБ-У на основе измерения соответствующих интервалов времени и сопоставления их с некоторыми наперед заданными значениями, которые соответствуют нормальному режиму работы устройства верификации в целом. Подобные временные параметры

устанавливаются на основе опытных данных и при первом обращении могут быть установлены эмпирически, а затем на тестовых примерах уточнены, например, на ИК ПО с известными точками уязвимости. Таким образом, обеспечивается режим обучения (самообучения) устройства верификации в каждом цикле верификации ИК ПО.

В рассматриваемом здесь примере на этапе работы модуля A 1.4 на вершину стека помещается лексема 'extern'. При выполнении модулей A1.5, A1.6, A1.7 и A1.8 последовательность из 1 элемента 'extern' приводит к срабатыванию правила 9 и к замене последовательности на поддерево с корневой вершиной sc-spec на этапе выполнения модуля A 1.9. На этапе выполнения модуля A. 10 определяется то, что в ИК ПО, приведенного выше примера, новые типы при обработке не вводятся, поэтому модуль A.11 не выполняется.

На этапе выполнения модуля A 1.4 на вершину стека, а именно в его первую ячейку, помещается лексема 'int'. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 последовательность из 1 элемента 'int' приводит к срабатыванию правила 10 и замене последовательности на поддерево с корневой вершиной type-spec. На этапе выполнения модуля A. 10 определяется, что в ИК ПО приведенного выше примера новые типы при обработке не вводятся, поэтому модуль A.11 не выполняется.

На следующем этапе выполнения модуля A1.4 на вершину стека, а именно в его первую ячейку, помещается лексема 'f'. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 никакое синтаксическое правило не срабатывает.

На следующем этапе выполнения модуля A1.4 на вершину стека помещается лексема '('. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 никакое синтаксическое правило не срабатывает.

На следующем этапе выполнения модуля A1.4 на вершину стека помещается лексема 'int'. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 последовательность из 1 элемента 'int' приводит к срабатыванию правила 10 и замене последовательности на поддерево с корневой вершиной type-spec. На этапе выполнения модуля A.10 определяется, что новые типы при обработке не вводятся, модуль A.11 не выполняется.

На следующем этапе выполнения модуля A 1.4 на вершину стека помещается лексема '*'. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 последовательность из 1 элемента '*', приводит к срабатыванию правила 11 и замене последовательности на поддерево с корневой вершиной pointer. На этапе выполнения модуля A.10 определяется, что новые типы при обработке не вводятся, модуль A.11 не выполняется.

На следующем этапе выполнения модуля A 1.4 на вершину стека помещается лексема 'arr'. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 срабатывает правило 8 и на шаге A1.9 производится замена последовательности на поддерево с вершиной param-decl. На этапе выполнения модуля A.10 определяется, что новые типы при обработке не вводятся, модуль A.11 не выполняется.

На следующем этапе выполнения модуля A1.4 на вершину стека помещается лексема ','. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 никакое синтаксическое правило не срабатывает.

На следующем этапе выполнения модуля A 1.4 на вершину стека помещается лексема 'inf'. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 последовательность из 1 элемента 'inf' приводит к срабатыванию правила 10 и замене последовательности на поддерево с корневой вершиной type-spec. На этапе

выполнения модуля A.10 определяется, что новые типы при обработке не вводятся, модуль A.11 не выполняется.

На следующем этапе выполнения модуля A1.4 на вершину стека помещается лексема 'ind'. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 срабатывает
5 правило 8 и на этапе выполнения модуля A1.9 производится замена последовательности на поддерево с вершиной param-decl. На этапе выполнения модуля A.10 определяется, что новые типы при обработке не вводятся, модуль A.11 не выполняется.

На следующем этапе выполнения модуля A1.4 на вершину стека помещается лексема ')'. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 срабатывает
10 правило 5 и на этапе выполнения модуля A1.9 производится замена последовательности на поддерево с вершиной d-d-param. На этапе выполнения модуля A.10 определяется, что новые типы при обработке не вводятся, модуль A.11 не
15 выполняется.

На следующем этапе выполнения модуля A1.4 на вершину стека помещается лексема';'. В процессе выполнения модулей A1.5, A1.6, A1.7 и A1.8 срабатывает
20 правило 1 и на этапе выполнения модуля A1.9 производится замена последовательности на поддерево с вершиной decl. На этапе выполнения модуля A.10 определяется, что новые типы при обработке не вводятся, модуль A.11 не выполняется.

Дальнейший поток лексем обрабатывается аналогичным образом, пока на этапе
25 выполнения модуля A1.3 не обнаружится, что поток лексем закончен. На этапе выполнения модуля A1.12 определяется, что стек содержит только один элемент - дерево разбора. При этом выполнение всех вышеперечисленных этапов контролируется АПБ-У, т.е. если, например, на этапе выполнения модуля A1.12 произойдет аварийный останов цикла разбора ИК ПО, то АПБ-У обеспечит
30 самодиагностику данного состояния устройства разбора. На этапе выполнения модуля A1.13 производится запись дерева разбора в динамический массив A.4.

Фрагмент дерева разбора, полученного на этапе работы модуля A1, может быть представлен в БВ АПБ-У (см. фиг.28) в виде (см. фиг.33-39), приемлемом для
35 зрительного восприятия и анализа оператором ЭВМ.

После работы модуля A2 дерево разбора для файла main.c примет вид, изображенный на фиг.33.

Фрагмент дерева разбора, полученного на этапе применения модуля лексического и синтаксического разбора к файлу f.c, приведен на фиг.35.

После работы модуля A2 дерево разбора для файла f.c примет вид, изображенный
40 на фиг.36.

После выполнения модуля A3 будет получено преобразованное дерево разбора, вид которого представлен на фиг.37. На фиг.37 темным цветом выделено поддерево, соответствующее описанию функции f., светлым цветом выделен вызов этой функции в
45 исходной программе.

Поскольку пример ИК программ не содержит обращений к внешним функциям, ее внутреннее представление и база данных условий корректности могут быть получены на основе преобразованного дерева разбора программы на этапе применения
50 модулей трансляции дерева разбора во внутреннее представление и генерации условий корректности (см. фиг.8).

В процессе работы модуля B1 осуществляется обращение к процедуре UF для обработки функции main. При обработке вызова функции f будет вызвана

процедура UF для функции f. Эта процедура оставит функцию f без изменений, так как она не содержит вызовов функций. После этого, код функции f будет добавлен в код main.

Рассмотрим для простоты преобразованное дерево в виде программы на языке Си.

```

5   int a[100];
    int i;
    int j;
    int *arr;
10  int ind;
    int f_ret;
    for(i=0; i<10; ++i)
    for(j=9; j>=i; --j)
15  a[10*i+j]=0;
    arr=a;
    ind=55;
    f_ret=arr[ind];
    return f_ret;

```

В приведенном примере присутствуют лишь простые типы, поэтому модуль В2 не выполняет преобразований.

В приведенном примере присутствуют циклы, которые будут на этапе выполнения модуля В3 развернуты. Так например, при преобразовании первого цикла будет получен следующий код на "псевдо-Си":

```

25  int a[100];
    int i;
    int j;
    int *arr;
30  int ind;
    int f_ret;
    i=0;
    goto (i<10) ? label1: label2;
35  label1:
    for(j=9; j>=i; --j)
    a[10*i+j]=0;
    ++i;
    label2:
40  arr=a;
    ind=55;
    f_ret=arr[ind];
    return f_ret;

```

При преобразовании второго цикла будет получен следующий код на "псевдо-Си":

```

45  int a[100];
    int i;
    int j;
    int *arr;
50  int ind;
    int f_ret;
    i=0;
    label0:

```

```

goto (i<10) ? label1: label2;
label1:
J=9;
label3:
5 goto (j>=i) ? label4: label5;
label3:
a[10*i+j]=0;
j++;
10 goto label3;
label4:
++i;
goto label0;
label2:
15 arr=a;
ind=55;
f_ret=arr[ind];
return f_ret;
20

```

На следующем этапе выполнения модуля В4 получается внутреннее представление программы следующего вида:

```

L1 assign    a@size=100;
L2 assign    a@offset=0;
25 L3 assign    i=0;
L4 condition i<10 ? L5:L12;
L5 assign    j=9;
L6 condition j>=i ? L7: L10;
L7 a_assign  a[10*i+j]=0;
L8 assign    j=j-1;
30 L9 goto      L6;
L10 assign   i=i+1;
L11 goto     L4;
L12         stop;

```

База данных условий корректности D8 будет представлена в следующем виде:

```

L7 a@offset+ 10*i+j<a@size;
L7 a@offset+ 10*i+j>=0;
L12 a@offset+ 55<a@size;
40 L12 a@offset+ 55>=0;

```

База знаний D9 соответствия условий корректности подозрительным точкам исходного кода:

```

1 main.c:3:9:16 1.0 Unable to deduce dereferencing upper bound;
45 2 main.c:3:9:16 1.0 Unable to deduce dereferencing lower bound;
3 main.c:14:7:15 1.0 Unable to deduce dereferencing upper bound;
4 main.c:14:7:15 1.0 Unable to deduce dereferencing lower bound;

```

Далее АПБ-У передаст управление АПБ-С. Рассмотрим в рамках данного примера работу каждого из модулей АПБ-С (см. фиг.17-27).

При этом в качестве дополнительного пояснения следует указать на то, что модуль С1 АПБ-С обеспечивает детализированный статический анализ ИК ПО на упрощенном языке внутреннего представления (далее - просто программы). Он

позволяет собрать по данному описанию программы сведения, необходимые для доказательства условий корректности, что эквивалентно условию отсутствия в исходной программе уязвимостей, связанных с переполнением буферов ОЗУ. Далее целесообразно изложить описание принципов работы данного модуля.

Функционирование данного модуля тесно связано со следующими объектами:

N - множество целых неотрицательных чисел;

I - множество инструкций внутреннего представления. Размеченный граф представляет собой пару $G=(V, E)$, где

V - множество вершин, являющееся подмножеством $N \times M$, где M - множество меток вершин графа. Для вершины $v=(n, m)$ число n будем называть индексом вершины, а m - ее меткой,

E - множество ребер, $E \subseteq V \times V$.

Путь в графе - последовательность вершин $v_1, \dots, v_n$, где $(v_i, v_{i+1}) \in E$ для каждого $i=1, \dots, n-1$. Путь называется простым, если в нем нет повторяющихся вершин.

Управляющим графом программы называют размеченный граф, для которого множество меток совпадает с множеством I - инструкций внутреннего представления программы.

Интервал значений переменной x - это одной из следующих множеств:

- $-\infty < x < +\infty$ - `undefined_value`

- $-\infty < x < b$ - `left_infinity`

- $a < x < +\infty$ - `right_infinity`

- $a \leq x < b$ - `interval`

- $x=a$ - `exact_value`

- $\emptyset$ - `impossible_value`

Здесь также указаны обозначения типов возможных интервалов. Спектром переменной x называют условие на множестве ее возможных значений, заданное в виде объединения непересекающихся интервалов. При этом количество интервалов не должно превышать некоторого наперед заданного значения M .

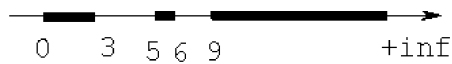
Над множеством спектров введем бинарные операции `intersect(P, Q)`, `join(P, Q)`.

При этом:

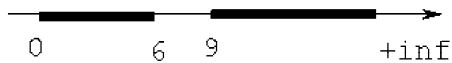
`intersect(P, Q)` - пересечение спектра P со спектром Q . Можно показать, что пересечением двух семейств из не более, чем M непересекающихся интервалов является спектр, состоящим из не более, чем 2^*M непересекающихся интервалов;

`join(P, Q)` - объединение спектра P со спектром Q . Объединение двух семейств из не более, чем M непересекающихся интервалов является спектром, состоящим из не более, чем 2^*M непересекающихся интервалов.

Для того чтобы при выполнении операций над спектрами количество интервалов в их описании не возрастало до бесконечности, после выполнения указанных операций в случае, когда число интервалов результирующего спектра превосходит M , необходимо несколько раз провести операцию "редукции": случайно выбрать пару соседних интервалов и заменить ее на минимальный интервал, содержащий оба указанных. Пример:



5



В данном примере изображен спектр, состоящий из трех интервалов. Если $M=2$, то
 10 будут случайным образом выбраны два соседних интервала (например, первый и второй) и объединены в один интервал.

Учитывая возможность выполнения подобной операции "редукции" спектра будем считать, что введенные ранее операции join и intersect преобразуют пару спектров, состоящих из не более, чем M интервалов в спектр, также состоящий из не более чем
 15 M интервалов.

Помимо указанных операций, для спектров также определены обычные арифметические операции - сложение, вычитание, умножение и деление. При этом:

$$P+Q=\{y\in Z:y=p+q, p\in P, q\in Q\};$$

$$P-Q=\{y\in Z:y=p-q, p\in P, q\in Q\};$$

$$P*Q=\{y\in Z:y=p*q, p\in P, q\in Q\};$$

$$P/Q=\{y\in Z:y=p/q, p\in P, q\in Q\}.$$

Здесь также после выполнения арифметической операции со спектрами следует
 25 производить "редукцию" результирующего спектра с тем, чтобы количество интервалов в них не превосходило M .

Спектр переменной x можно понимать, как некий предикат от x , например, спектру $[0,6)\cup\{10\}$ соответствует предикат $(0\leq x<6)\vee(x==10)$.

Пусть N - число инструкций во внутреннем представлении программы, K - число
 30 переменных.

Таблица инвариантов - это массив TI длины N , элементами которого являются множества предикатов от переменных внутреннего представления программы.

Получение таблицы инвариантов является главной задачей данного модуля.

Таблица спектров вдоль простого пути $L=(L[1],\dots, L[n])$ - это двумерный массив TS
 35 размера $K*n$, элементами которого являются спектры переменных. $TS[k][i]$ - спектр k -й переменной в точке $L[i]$ программы.

Отметим, что спектр можно тоже понимать как предикат, зависящий от одной переменной. При этом указанный предикат будет обозначать условие, которое
 40 выполняется в точке $L[i]$ при условии, что трасса выполнения программы прошла через вершины $L[1],\dots,L[i]$.

Использование спектров предоставляет более гибкий механизм оперирования выведенными предикатами, поскольку над спектрами, как над обычными
 45 переменными, можно производить арифметические операции.

Таблица предикатов вдоль простого пути L представляет собой массив TP длины n , элементами которого являются множества предикатов, причем $TP[i]$ - множество предикатов, которые будут выполнены в точке $L[i]$ программы, но тоже при условии, что трасса выполнения программы прошла через вершины $L[1],\dots,L[i]$. Использовать
 50 таблицу предикатов необходимо вместе с таблицей спектров, поскольку спектр не позволяет задавать предикаты, зависящие от нескольких переменных, а условия, хранимые в таблице предикатов, невозможно корректировать в случае, когда на простом пути встречается инструкция, изменяющая значение переменной, входящей в

предикат.

Таким образом, на этапе выполнения модуля С1 (см. фиг.18) будет осуществлен вызов процедура GG, которая считает 12 инструкцией внутреннего представления и соединит их в управляющий граф. Результат ее работы представлен на фиг.38.

Далее процедура SP модуля С 1 обнаружит следующие максимальные пути в данном графе:

path1=(L1, L2, L3, L4, L5, L6, L7, L8, L9);

path2=(L1, L2, L3, L4, L5, L6, L10, L11);

path3=(L1, L2, L3, L4, L12).

Для каждого из путей path1, path2 и path3 будет вызвана процедура определения инвариантов и с ее помощью определены множество Y, таблицы спектров и таблицы предикатов вдоль каждого из них:

1) Вдоль пути path1.

Таблица спектров и множество Y:

	Y	a@size	a@offset	i	j
L1:	{}	[-inf, +inf)	[-inf, +inf)	[-inf, +inf)	[-inf, +inf)
L2:	{}	{100}	[-inf, +inf)	[-inf, +inf)	[-inf, +inf)
L3:	{}	{100}	{0}	[-inf, +inf)	[-inf, +inf)
L4:	{L4, L5, L6, L7, L8, L9, L10, L11}	{100}	{0}	[0, +inf)	[-inf, +inf)
L5:	{}	{100}	{0}	[0, 10)	[-inf, +inf)
L6:	{L6, L7, L8, L9}	{100}	{0}	[0, 10)	[-inf, 10)
L7:	{}	{100}	{0}	[0, 10)	[-inf, 10)
L8:	{}	{100}	{0}	[0, 10)	[-inf, 10)
L9:	{}	{100}	{0}	[0, 10)	[-inf, 9)

Множество предикатов будет непусто только в точках L7, L8, L9:

L7: {j-i>=0};

L8: {j-i>=0};

L9: {j-i>=-1}.

Процедура Y-FILTER в точке L4 для спектра переменной i: {0} и множества Y={L4, L5, L6, L7, L8, L9, L10, L11} обнаружит в точке L10 инструкцию "i=i+1" и пересчитает новый спектр для i по правилу: $\text{спес}(i)=[\min(\{0\}), +\text{inf})=[0, +\text{inf})$.

Аналогично, в точке L6 для спектра переменной j: {9} и множества Y={L6, L7, L8, L9} процедура обнаружит в точке L8 инструкцию "j=j-1" и пересчитает новый спектр для j по правилу $\text{спес}(j)=[-\text{inf}, \max(\{9\})+1)=[-\text{inf}, 10)$.

2) Вдоль пути path2.

Таблица спектров и множество Y:

	Y	a@size	a@offset	i	j
L1:	{}	[-inf, +inf)	[-inf, +inf)	[-inf, +inf)	[-inf, +inf)
L2:	{}	{100}	[-inf, +inf)	[-inf, +inf)	[-inf, +inf)
L3:	{}	{100}	{0}	[-inf, +inf)	[-inf, +inf)
L4:	{L4, L5, L6, L7, L8, L9, L10, L11}	{100}	{0}	[0, +inf)	[-inf, +inf)
L5:	{}	{100}	{0}	[0, 10)	[-inf, +inf)
L6:	{L6, L7, L8, L9}	{100}	{0}	[0, 10)	[-inf, 10)
L10:	{}	{100}	{0}	[0, 10)	[-inf, 10)
L11:	{}	{100}	{0}	[1, 11)	[-inf, 9)

Множество предикатов будет непусто только в точках L10, L11:

L10: {j-i<0};

L11: {j-i<-1}.

Процедура Y-FILTER на данном пути отработает точно так же как и на пути path1.

3) Вдоль пути path3

Таблица спектров и множество Y:

		Y	a@size	a@offset	i	j
5	L1	{}	$[-inf, +inf)$	$[-inf, +inf)$	$[-inf, +inf)$	$[-inf, +inf)$
	:					
	L2	{}	{100}	$[-inf, +inf)$	$[-inf, +inf)$	$[-inf, +inf)$
	:					
	L3	{}	{100}	{0}	$[-inf, +inf)$	$[-inf, +inf)$
	:					
10	L4	{L4, L5, L6, L7, L8, L9, L10, L11}	{100}	{0}	$[0, +inf)$	$[-inf, +inf)$
	:					
	L1	{}	{100}	{0}	$[10, +inf)$	$[-inf, +inf)$
	2:					

Множество предикатов будет пусто во всех точках.

- 15 Процедура Y-FILTER на данном пути в точке L4 отработает точно также как и на пути path1 в точке L4. Уточнение таблицы инвариантов для каждого из путей выдаст следующий результат:

	в точке L1:	
20	в точке L2:	
	a@size==100	# spectrum a@size: {100}, path1, path2, path3
	в точке L3:	
	a@size==100	# spectrum a@size: {100}, path1, path2, path3
	a@offset==0	# spectrum a@offset: {0}, path1, path2, path3
25	в точке L4:	
	a@size==100	# spectrum a@size: {100}, path1, path2, path3
	a@offset==0	# spectrum a@offset: {0}, path1, path2, path3
	i>=0	# spectrum i: [0, +inf), path1, path2, path3
	в точке L5:	
30	a@size==100	# spectrum a@size: {100}, path1, path2
	a@offset==0	# spectrum a@offset: {0}, path1, path2
	i>=0	# spectrum i: [0, 10), lower bound, path1, path2
	i<10	# spectrum i: [0, 10), upper bound, path1, path2
	в точке L6:	
35	a@size==100	# spectrum a@size: {100}, path1, path2
	a@offset==0	# spectrum a@offset: {0}, path1, path2
	i>=0	# spectrum i: [0, 10), lower bound, path1, path2
	i<10	# spectrum i: [0, 10), upper bound, path1, path2
	j<10	# spectrum j: [-inf, 10), path1, path2
	в точках L7, L8:	
40	a@size==100	# spectrum a@size: {100}, path1
	a@offset==0	# spectrum a@offset: {0}, path1
	i>=0	# spectrum i: [0, 10), lower bound, path1
	i<10	# spectrum i: [0, 10), upper bound, path1
	j<10	ft spectrum j: [-inf, 10), path1
45	j-i>=0	ft predicate, path1
	в точке L9:	
	a@size==100	# spectrum a@size: {100}, path1
	a@offset==0	# spectrum a@offset: {0}, path1
	i>=0	# spectrum i: [0, 10), lower bound, path1
	i<10	# spectrum i: [0, 10), upper bound, path1
50	J<9	# spectrum j: [-inf, 9), path1
	j-i>=-1	# predicate, path1
	в точке L10:	
	a@size==100	# spectrum a@size: {100}, path2

```

a@offset==0      # spectrum a@offset: {0}, path2
i>=0             # spectrum i: [0, 10), lower bound, path2
i<10             # spectrum i: [0, 10), upper bound, path2
j<10             # spectrum j: [-inf, 9), path2
j-i<0            # predicate, path1
5
в точке L11:
a@size==100      # spectrum a@size: {100}, path2
a@offset==0      # spectrum a@offset: {0}, path2
i>=1             # spectrum i: [1, 11), lower bound, path2
i<11             # spectrum i: [1, 10), upper bound, path2
10
j<9              # spectrum j: [-inf, 9), path2
j-i<-1           # predicate, path1
в точке L12:
a@size==100      # spectrum a@size: {100}, path3
a@offset==0      # spectrum a@offset: {0}, path3
15
i>=10            # spectrum i: [10, +inf), lower bound, path3

```

После работы модуля С.1 управление передается модулю С.2 ("Генерация систем информационной базы"), который для каждого условия корректности синтезирует свою система ограничений. Предикаты записываются в форме "expr R const", где expr - некоторое арифметическое выражение, R - предикатный символ, а const - некоторое целое число.

```

System 1: # correctness condition "L7 a@offset+10*i+j<a@size;"
a@size==100
25
a@offset==0
i>=0
i<10
j<10
j-i>=0
30
a@offset+10*i+j-a@size>=0;
System 2: # correctness condition "L7 a@offset+10*i+j>=0;"
a@size==100
a@offset==0
35
i>=0
i<10
j<10
j-i>=0
40
a@offset+10*i+j<0
System 3: # correctness condition "L12 a@offset+55<a@size;"
a@size==100
a@offset==0
i>=10
45
a@offset-a@size>=-55
System 4: # correctness condition "L12 a@offset+55>=0;"
a@size==100
a@offset==0
i>=10
50
a@offset<=-55

```

Далее управление передается модулю С.3 ("Исследование систем ограничений информационной базы"). В качестве примера будем полагать, что число

параллельных или независимых процессов равно четырем ($t=4$). На этапе формирования набора параллельно выполняемых задач будет получено задание для каждого из процессов:

для процесса № i - исследовать систему № i на предмет существования решений.

5 Рассмотрим далее работу процесса № 1.

В системе № 1 нет тождественно ложных предикатов или пар противоречащих друг другу предикатов. Поэтому система может быть сведена к задаче линейного программирования, которая может быть представлена в канонической форме (здесь строгие неравенства приведены к нестрогим, а также введены новые неотрицательные переменные, позволяющие перейти от неравенств к равенствам):

$-i_1+i_2+x_1+y_1==0;$

$i_1-i_2+x_2+y_2==9;$

$j_1-j_2+x_3+y_3==9;$

15 $i_1-i_2-j_1+j_2+x_4+y_4==0;$

$-10*i_1+10*i_2-j_1+j_2+a@size1-a@size2-a@offset1+a@offset2+x_5+y_5==0;$

$a@size1-a@size2+y_6==100;$

$a@offset1-a@offset2+y_7==0.$

20 Здесь для простоты были введены новые неотрицательные переменные:

$i_1, i_2, j_1, j_2, a@size1, a@size2,$

$a@offset1, a@offset2, x_1, x_2, x_3, x_4, x_5, y_1, y_2, y_3, y_4, y_5, y_6, y_7$

с условиями $i=i_1-i_2, j=j_1-j_2$ и т.д.

Необходимо максимизировать следующий функционал:

25 $L(Y)=-y_1-y_2-y_3-y_4-y_5-y_6-y_7.$

Решением данной задачи является значение $L^*=-1$, следовательно? исходная система не имеет решений. Для остальных систем информационной базы также можно показать, что они не имеют решений. Таким образом $DNG=\{\}$.

30 Следовательно, модуль С.4 получит на вход пустой список номеров, не выведенных без условий корректности - DNG, и выдаст в базу данных D11 отчет, подтверждающий корректность программы:

No dangerous points found

No buffer overflows are possible

35 Рассмотрим второй пример ИК ПО следующего вида:

Пример 2.

Исходная С программа

#include<string.h>

40 #include<stdio.h>

int main(int argc, char** argv)

{

if (argc<2)

45 return 0;

int len=strlen(argv[1]);

char* p=malloc(len+1);

strcpy (p, argv[1]);

printf("%s\n", p);

50 return 0;

}

На первом этапе работы АПБ-А фрагмент дерева разбора ИК ПО, полученного на этом этапе применения объединенного модуля лексического и синтаксического

разбора, представлен на фиг.34.

Темным цветом на иллюстрации выделено поддерево разбора, соответствующее синтаксической конструкции

```
int len=strlen(argv[1]);
```

АПБ-А содержит обращения только к внешним библиотечным функциям, поэтому встраивание функций в данном случае не требуется.

Таблица типов пуста, поскольку в данном примере используются только простые типы данных или указатели на них.

Внутреннее представление может быть получено на основе дерева разбора, приведенного выше, а также на основе базы аннотаций для внешних функций.

Рассмотрим два случая:

случай (а): база знаний D6 содержит аннотации для функций strcpy, malloc:

```
function-annotation (strcpy) {
```

```
  params { *src }
```

```
  output { *dest }
```

```
  pre-conditions {
```

```
    m_0 src@is_string==1;
```

```
    m_0 src@size>src@strlen;
```

```
    m_0 dest@size>dest@offset+src@strlen;
```

```
  }
```

```
  instruction {
```

```
    m_0 dest@is_string=1;
```

```
    m_1 dest@strlen=src@strlen;
```

```
  }
```

```
}
```

```
function-annotation (malloc) {
```

```
  params { n }
```

```
  output { *ptr }
```

```
  pre-conditions {
```

```
  }
```

```
  instructions {
```

```
    m_0 condition _unknown__ ? m_1 : m_5;
```

```
    m_1 assign ptr@is_null=0;
```

```
    m_2 assign ptr@size=n;
```

```
    m_3 assign ptr@offset=0;
```

```
    m_4 goto m_7;
```

```
    m_5 assign ptr@is_null=0;
```

```
    m_6 assign ptr@size=0;
```

```
  }
```

```
}
```

случай (б): база знаний D6 пополнена аннотацией для функции strlen:

```
function-annotation (strlen) {
```

```
  params { buf }
```

```
  output { length }
```

```
  pre-condition {
```

```
    m_0 buf@is_string==1;
```

```
    m_0 buf@size>buf@strlen;
```

```
  }
```

```

instructions {
  m_0 length=buf@strlen;
}
}

```

5 При этом в случаях (а) и (б) будут получены различные
внутренние представления исходной программы - база данных D7;
базы данных D8 условий корректности;
информационные базы знаний D10;

10 отчеты об обнаруженных уязвимостях - база данных D111.

В приведенном втором примере присутствуют лишь вызовы библиотечных функций, поэтому модули В.1 и В.2 не выполняют преобразований.

В данном примере присутствует условная конструкция if. Поэтому на этапе работы модуля В.3 она будет преобразована в конструкцию cond-goto (эта конструкция
15 отсутствует в языке Си, однако для простоты мы будем представлять
модифицированное дерево разбора в виде программы на языке "псевдо-Си").

После выполнения преобразования модулем В.3 из исходных данных получается следующий код:

```

20 int len;
   char *p;
   goto (argc<2) ? label1 : label2;
   label1:
25 return 0;
   label2:
   len=strlen(argv[1]);
   p=malloc(len+1);
   strcpy (p, argv[1]);
30 printf("%s\n", p);
   return 0;

```

Как можно видеть, добавлены 2 метки и инструкция условного перехода.

На этапе работы модуля В.4 получают внутреннее представление ИК ПО в виде:
(а)

```

35
                                     L1 assign      argv@size=argc;
                                     L2 condition    argc<2 ? L5 : L3;
                                     L3 assign      argv@1@is_string=1;
                                     L4 assign      argv@1@size=argv@1@strlen+1;
40                                     L5 assign      argv@1@offset=0;
                                     L5 assign      len= unknown ;
                                     L6 assign      p@size=len+1;
                                     L7 assign      p@offset=0;
                                     L8 assign      p@is_string=1;
                                     L9 assign      p@strlen=argv@1@strlen;
45                                     L10           stop;

```

50 Инструкции L1, L3 и L4 соответствуют специальным аннотациям, используемым для входных параметров функции main - argc, argv в данном случае. Эти аннотации показывают, что параметр argc равен длине массива argv, а каждый используемый в исходном коде элемент argv[i] является корректной Си-строкой, то есть последовательностью char-символов, оканчивающейся символом '\0'.

Инструкция L2 соответствует проверке условия

```
if (argc<2)
```

```
return 0;
```

Инструкция L5 соответствует выражению

```
int len=strlen(argv[1]);
```

5 (при этом unknown указывает на неопределенность возвращаемого значения неизвестной функции strlen).

Инструкция L6 получается из выражения

```
p=malloc(len+1);
```

10 и правила вывода в базе аннотаций для функции malloc (секция instructions):

```
buf@size=size
```

```
buf@offset=offset
```

применяемого к выражениям p (вместо buf) и len+1 (вместо size).

Инструкции L8, L9 получаются из выражения

15 strcpy (p, argv[1]);

и правил вывода в базе аннотаций (функция strcpy, секция instructions).

```
dest@is_string=1
```

```
dest@strlen=src@strlen
```

20 применяемым к выражениям p (вместо dest) и argv@1 (вместо src).

(б)

	L1 assign	argv@size=argc;
	L2 condition	argc<2 ? L10 : L3;
25	L3 assign	argv@1@is_string=1;
	L4 assign	argv@1@size=argv@1@strlen+1;
	L5 assign	len=argv@1@strlen;
	L6 assign	p@size=len+1;
	L7 assign	p@offset=0;
	L8 assign	p@is_string=1;
30	L9 assign	p@strlen=argv@1@strlen;
	L10	stop;

Жирным шрифтом выделены инструкции, в которых наблюдается различие внутренних представлений программы в случаях (а) и (б). В случае (а) выделенная

35 жирным инструкция показывает неопределенность возвращаемого значения внешней функции strlen. Добавление аннотации функции strlen в базу знаний D6 в случае (б) позволит на этапе преобразования во внутреннее представление воспользоваться содержащимся в аннотации (в секции instructions) правилом вывода

40 length=buf@strlen,

примененным к выражениям len (вместо length) и argv@1 (вместо buf), что породит инструкцию L5.

База знаний D9 условий корректности в случаях (а) и (б) будет иметь следующий вид:

45 L5 1<argv@size;

L5 argv@1@is_string==1;

L5 argv<?1@size>argv@1@strlen:

L8 1<argv@size;

50 L8 argv@1@is_string==1;

L8 argv@1@size>argv@1@strlen;

L8 p@size>p@offset+argv@1@strlen;

Жирным выделены условия, появляющиеся в D9 при пополнении базы знаний D6

аннотаций. Рассмотрим каждое из условий более подробно.

Условие:

L5 1<argv@size;

позволяет гарантировать отсутствие переполнения при разыменовании argv[1] в выражении

int len=strlen(argv[1]).

Пара условий в точке L5, выделенных жирным шрифтом:

L5 argv@1@is_string==1;

L5 argv@1@size>argv@1@strlen;

получается в случае (б) по правилам вывода в базе аннотаций (функция strlen, секция conditions):

buf@is_string==1

buf@size>buf@strlen

примененным к выражению argv@1 (вместо buf).

Условие

L8 1<argv@size;

позволяет гарантировать отсутствие переполнения при разыменовании argv [1] в выражении

strcpy(p, argv[1]);

Тройка условий в точке L8

L8 argv@1@is_string=1;

L8 argv@1@size>argv@1@strlen;

L8 p@size>p@offset+argv@1@strlen;

получается по правилам вывода в базе аннотаций (функция strcpy, секция conditions):

src@is_string==1

src@size>src@strlen

dest@size>src@strlen

примененным к выражениям argv@1 (вместо src) и p (вместо dest).

База знаний D9 соответствия условий корректности подозрительным точкам исходного кода в случае а):

1 main.c:9:20:26 0.67 Unable to deduce dereferencing upper bound;

2 main.c:11:13:19 0.67 Unable to deduce dereferencing upper bound;

3 main.c:11:3:20 1.0 Buffer is not a valid string;

4 main.c:11:3:20 1.0 Source buffer size is less than string length;

5 main.c:11:3:20 1.0 Target buffer size is less than string length;

База знаний D9 соответствия условий корректности подозрительным точкам исходного кода в случае б):

1 main.c:9:20:26 0.67 Unable to deduce dereferencing upper bound;

2 main.c:11:13:20 1.0 Buffer is not a valid string;

3 main.c:11:13:20 1.0 Target buffer size is less than string length;

4 main.c:11:13:19 0.67 Unable to deduce dereferencing upper bound;

5 main.c:11:3:20 1.0 Buffer is not a valid string;

6 main.c:11:3:20 1.0 Source buffer size is less than string length;

7 main.c:11:3:20 1.0 Target buffer size is less than string length;

На следующем этапе разбора управление передается АПБ-С. При этом работа модуля С1 достаточно полно продемонстрирована в примере 1, поэтому укажем только результат его работы:

в точке L1:

в точке L2:

```
argv@size - argc==0 # predicate - L1
```

5 в точке L3:

```
argv@size - argc==0 # predicate - L1
argc>=2 # spectrum argc: [2, +inf) - L2
```

10 в точке L4:

```
argv@size - argc==0 # predicate - L1
argc>=2 # spectrum argc: [2, +inf) - L2
argv@1@is_string==1 # spectrum argv@1@is_string: {1} - L3
```

15 в точке L5:

```
argv@size - argc==0 # predicate - L1
argc>=2 # spectrum argc: [2, +inf) - L2
argv@1@is_string==1 # spectrum argv@1@is_string: {1} - L3
argv@1@size - argv@1@strlen==1 # predicate - L4
```

20

в точке L6:

```
argv@size - argc==0 # predicate - L1
argc>=2 # spectrum argc: [2, +inf) - L2
argv@1@is_string==1 # spectrum argv@1@is_string: {1} - L3
argv@1@size - argv@1@strlen==1 # predicate - L4
len - argv@1@strlen==0 # predicate - L5
```

25

в точке L7:

```
argv@size - argc==0 # predicate - L1
argc>=2 # spectrum argc: [2, +inf) - L2
argv@1@is_string==1 # spectrum argv@1@is_string: {1} - L3
argv@1@size - argv@1@strlen==1 # predicate - L4
len - argv@1@strlen==0 # predicate - L5
p@size - len=1; # predicate - L6
```

30

35

в точке L8:

```
argv@size - argc==0 # predicate - L1
argc >=2 # spectrum argc: [2, +inf) - L2
argv@1@is_string==1 # spectrum argv@1@is_string: {1} - L3
argv@1@size - argv@1@strlen==1 # predicate - L4
len - argv@1@strlen==0 # predicate - L5
p@size - len=1; # predicate - L6
p@offset=0 # spectrum p@offset: {0} - L7
```

40

45

в точке L9:

```
argv@size - argc==0 # predicate - L1
argc>=2 # spectrum argc: [2, +inf) - L2
argv@1@is_string==1 # spectrum argv@1@is_string: {1} - L3
argv@1@size - argv@1@strlen==1 # predicate - L4
len - argv@1@strlen==0 # predicate - L5
p@size - len==1; # predicate - L6
```

50

```

p@offset==0                                # spectrum p@offset: {0} - L7
p@is_string==1                             # spectrum p@is_string: {1} - L8

```

в точке L10:

```

5      argv@size - argc==0                    # predicate - L1
      argc>=2                                # spectrum argc: [2, +inf) - L2
      argv@1@is_string==1                    # spectrum argv@1@is_string: {1} - L3
      argv@1@size - argv@1@strlen=1          # predicate - L4
10     len - argv@1@strlen==0                #predicate - L5
      p@size - len==1;                       # predicate - L6
      p@offset==0                            # spectrum p@offset: {0} - L7
      p@is_string==1                        # spectrum p@is_string: {1} - L8
      p@strlen - argv@1@1@strlen==0          # predicate - L9

```

15 **Жирным выделены те условия, которые будут выведены при пополнении базы знаний аннотаций D6, то есть в случае (б).**

На следующем этапе работы модуля С.2 осуществляется генерация информационной базы знаний D10. При этом для рассматриваемого случая - случай (а), она будет содержать следующие сведения:

```

20   System 1: # correctness condition "L5 1<argv@size;"
      argv@size - argc==0
      argc>=2
      argv@1@is_string==1
25   argv@1@size - argv@1@strlen==1
      argv@size<=1
      System 2: # correctness condition "L8 1<argv@size;"
      argv@size - argc==0
      argc>=2
30   argv@1@is_string==1
      argv@1@size - argv@1@strlen=1
      p@size - len==1
      p@offset=0
35   argv@size<=1
      System 3; # correctness condition "L8 argv@1@is_string==1;"
      argv@size - argc==0
      argc>=2
      argv@1@is_string=1
40   argv@1@size - argv@1@strlen==1
      p@size - len=1
      p@offset==0
      argv@1@is_string !=1
45   System 4: # correctness condition "L8 argv@1@size>argv@1@strlen;"
      argv@size - argc==0
      argc>=2
      argv@1@is_string==1
      argv@1@size - argv@1@strlen==1
50   p@size - len==1
      p@offset=0
      argv@1@size - argv@1@strlen<=0
      System 5: # correctness condition "L8 p@size>p@offset+argv@1@strlen;"

```

```

argv@size - argc==0
argc>=2
argv@1@is_string==1
argv@1@size - argv@1@strlen==1
5 p@size - len==1
p@offset=0
p@size - p@offset - argv@1@strlen<=0
для случая (б):
10 System 1: # correctness condition "L5 1<argv@size;"
argv@size - argc==0
argc>=2
argv@1@is_string==1
argv@1@size - argv@1@strlen=1
15 argv@size<=1
System 2: # correctness condition "L5 argv@1@is_string==1;"
argv@size - argc=0
argc>=2
20 argv@1@is_string==1
argv@1@size - argv@1@strlen==1
argv@1@is_string !=1
System 3: # correctness condition "L5 argv@1@size>argv@1@strlen;"
argv@size - argc=0
25 argc>=2
argv@1@is_string==1
argv@1@size - argv@1@strlen=1
argv@1@size - argv@1@strlen<=0
30 System 4: # correctness condition "L8 1<argv@size;"
argv@size - argc==0
argc>=2
argv@1@is_string==1
argv@1@size - argv@1@strlen==1
35 len - argv@1@strlen==0
p@size - len==1
p@offset==0
argv@size<=1
40 System 5: # correctness condition "L8 argv@1@is_string==1;"
argv@size - argc==0
argc>=2
argv@1@is_string==1
argv@1@size - argv@1@strlen==1
45 len - argv@1@strlen==0
p@size - len==1
p@offset==0
argv@1@is_string !=1
50 System 6: # correctness condition "L8 argv@1@size>argv@1@strlen;"
argv@size - argc==0
argc>=2
argv@1@is_string==1

```

```

argv@1@size - argv@1@strlen==1
len - argv@1@strlen==0
p@size - len==1
p@offset==0
5 argv@1@size - argv@1@strlen<=0
System 7: # correctness condition "L8 p@size>p@offset+argv@1@strlen;"
argv@size - argc==0 argc>=2
argv@1@is_string==1
10 argv@1@size - argv@1@strlen==1
len - argv@1@strlen==0
p@size - len==1
p@offset==0
p@size - p@offset - argv@1@strlen<=0
15

```

На следующем этапе управление передается модулю С.3. Работа данного модуля достаточно полно раскрыта в примере 1, поэтому укажем только на окончательный результаты.

В случае (а): системы 1-4 не имеют решений, система 5 имеет целочисленное решение:

```

20 argc=2
argv@size=2
argv@1@size=1
argv@1@strlen=1
25 argv@1@is_string=1
len=0
p@size=0
p@is_string=1
30 p@offset=0

```

В случае (б): системы 1-7 не имеют решений.

Таким образом, в случае

(а) DNG={5}

(б) DNG={ }.

35 На следующем этапе управление передается модулю С.4 "Формирование отчета об обнаруженных уязвимостях".

В случае (а) список DNG={5}. Из базы соответствия условий корректности подозрительным точкам исходного кода будет извлечена запись

```

40 "5 main.c:11:3:20 1.0 Target buffer size is less than string length;"

```

Извлекая из 11-й строки файла main.c выражение, записанное с 3-й по 20-ю позиции, получим фрагмент исходного кода, в котором возможно переполнение буфера: strcpy(p, argv[1]).

Поэтому, будет сгенерирован отчет:

```

45 List of dangerous points:

```

1) Possible buffer overflow vulnerability found at line 11,
positions 3...20, danger level 1.0

Fragment of the source code:

```

50 ...
strcpy(p, argv [1]);

```

...
Possible reasons: Target buffer size is possibly less then string length

В случае (б) список DNG пуст, поэтому будет получен отчет, подтверждающий корректность программы:

No dangerous points found

No buffer overflows are possible

5 Пример 3.

Поскольку в предыдущих примерах таблица типов была пуста, приведем дополнительный пример ИК ПО, демонстрирующий работу модуля С.3 ("разворачивание структур"). Данный пример для простоты будет содержать

10 отдельные участки программного кода.

Исходный код ПО:

typedef int type0;

typedef struct

{

15 type0 *a;

char *b;

} type1;

typedef struct

20 {

type1 *e1;

type1 e2;

type0 *e3;

25 } type2;

...

type1 x;

type2 y;

...

30 *y.e1=x;

x.a[20]= *y.e3;

Таблица типов для данной части кода:

type0: int

35 type1: struct {type0 *a; char *b;}

type2: struct {type1 *e1; type1 e2; type0 *e3;}

На основании таблицы типов объявление переменной x типа type1 заменяется на объявление двух переменных: x_a и x_b, соответствующих полям структуры type1.

40 Аналогичная замена происходит с объявлением переменной y, в которой сначала разворачивается объявление типа type2, а потом разворачиваются объявления получающихся переменных типа type1, в результате чего образуются 5 переменных: y_e1_a, y_e1_b, y_e2_a, y_e2_b, y_e3.

Присваивание

45 *y.e1=x;

преобразуется в пару присваиваний, соответствующих полям типа type1:

*Y_e1_a=x_a;

*y_e2_b=x_b;

50 Остальные операции преобразуются аналогичным образом.

Получающийся при этом код эквивалентен коду исходной программы (в смысле операций ячейками памяти), однако содержит только переменные простых типов и указатели:

```

int * x_a;
char * x_b;
int ** y_e1_a;
5 char ** y_e1_b;
int * y_e2_a;
char * y_e2_b;
int * y_e3;
10 * y_e1_a=x_a;
    * y_e2_b=x_b;
    x_a[20]= * y_e3.

```

В общем случае процесс верификации ИК ПО для распределенных вычислительных комплексов представляет динамическую процедуру приема и обработки ИК ПО по 15 зависимым и независимым каналам. При этом АПБ-У осуществляет обработку сигналов, возникающих на системной шине устройства верификации (см. фиг.1) в процессе интерактивной верификации ИК ПО и работы АПБ-А, АПБ-В и АПБ-С. Эти сигналы поступают на входы У1 (блока нормирующих преобразователей (БНП)) (см. 20 фиг.28), при этом согласно предлагаемому способу эти сигналы разбивают на две группы. Для определенности будем полагать, что первая группа сигналов характеризует независимые по времени интерактивные процессы верификации ИК ПО, например, первая группа сигналов представляет набор наблюдаемых 25 интерактивных процессов обработки ИК ПО, поступающих по независимым интерфейсным каналам:

$$\tilde{X}^{\wedge} = \{X(1), X(2), Y(1), Y(2), Z(1), Z(2)\}, n=3.$$

Вторая группа сигналов поступает по зависимым интерфейсным каналам. Для 30 определенности дальнейшего изложения будем полагать, что эти сигналы представляют работу АПБ-А, АПБ-В, АПБ-С или оператора ЭВМ (в общем случае лица, принимающего решение - ЛПР, агента и т.п.), т.е. это некоторая реакция вида:

$$\tilde{U}^{\wedge} = \{U(i); i = \overline{1, 2^n}\}$$

Следовательно, в рассматриваемом примере реализации АПБ-У в первой группе 35 будет три независимых канала ($n=3$) измерений состояния процесса верификации ИК ПО, а во второй - 2^n (в данном случае восемь) зависимых каналов измерений состояния процесса верификации ИК ПО распределенных вычислительных комплексов. При этом согласно способу верификации ИК ПО в распределенных 40 вычислительных сетях необходимо хранить в базе данных соответствующие оценки верхних и нижних значений результатов верификации по каждому каналу обработки ИК ПО, т.е. $\forall t > 0$:

```

X(1)=inf X1(t) или X(1)→min X1(t),
45 X(2)=sup X1(t) или X(2)→max X1(t),
    аналогично
    Y(1)=inf Y1(t) или Y(1)→min Y1(t),
    Y(2)=sup Y1(t) или Y(2)→max Y1(t),
50 Z(1)=inf Z1(t) или Z(1)→min Z1(t),
    Z(2)=sup Z1(t) или Z(2)→max Z1(t).

```

Для этого перед началом каждого цикла верификации ИК ПО, например, в момент

t_0 формируется сигнал общего сброса на все регистры памяти и обработки сигналов измерений о состоянии основных циклов верификации ИК ПО (этот процесс на фиг. не показан). На первом временном интервале с помощью модуля коммутатора У2 осуществляется опрос следующих каналов БНП У1: первый канал БНП У1 из первой группы независимых интерфейсных каналов устройства верификации ИК ПО, на который поступает в рассматриваемом случае - $X(1)$; третий канал - $Y(1)$ и пятый - $Z(1)$. Одновременно из второй группы каналов опрашивается первый канал, на который подается для измерения сигнал $U(1)$.

Для рассматриваемого примера этот процесс комбинаторного опроса каналов системной шины устройства верификации ИК ПО из разных групп и запоминание значений сигналов завершается за восемь тактов. Описанная последовательность операций интерактивной верификации ИК ПО реализована на основе фиксации верхних $\{X(2), Y(2), Z(2)\}$ и нижних $\{X(1), Y(1), Z(1)\}$ границ изменения амплитуд сигналов первой группы каналов измерения состояния основных процессов интерактивной верификации ИК ПО.

Описанную последовательность опроса в предложенной реализации обеспечивает модуль формирования динамической развертки графоаналитического раstra (МФДРГАР) У5. Эту процедуру в рассматриваемом случае обеспечивает блок У5.3 синхронизации (см. фиг.30) за восемь тактов. Здесь для простоты пояснений полагаем, что значения амплитуд возмущающих воздействий, в качестве которых в данном случае выступает поток ИК ПО, поступающий по независимым интерфейсным каналам, имеет следующий вид:

$$\tilde{X} = \{X(1), X(2), Y(1), Y(2), Z(1), Z(2)\}, n=3.$$

Соответственно, реакция на эти воздействия:

$$\tilde{U} = \{U(i); i = \overline{1, 2^n}\}$$

исследуемого процесса верификации ИК ПО, для всех $t = \{t_1, \dots, t_8\}$.

При этом комбинаторика процедуры (или стратегия) опроса каждого последующего цикла определения точек уязвимости в процессе верификации ИК ПО будет зависеть не только от того, какие претерпели изменения соответствующие измеряемые переменные по каждому из каналов, но и непосредственно зависеть от результатов верификации ИК ПО, которые сопровождаются визуализацией результатов - отображением на БВ У6 (см. фиг.28, 37 и 39).

Так, вышеописанный пример последовательности операций позволяет по каждому из входов второй группы каналов измерения получить значения комбинаторных оценок следующего вида:

$$U'(1) = f_{j_1}(X'(1), Y'(1), Z'(1)),$$

$$U'(2) = f_{j_2}(X'(2), Y'(2), Z'(2)),$$

...

$$U'(8) = f_{j_8}(X'(2), Y'(2), Z'(2)).$$

Одновременно (в данном примере, начиная с такта t_g) в блоке выделения экстремума У4.3 МФСКГАР У4 осуществляется процедура определения $\max U(.)$ и $\min U(.)$.

Схема У 4.5 формирования дискретных приращений (см. фиг.29) статических координат графоаналитического раstra обеспечивает циклическое изменение следующих переменных:

$$X(t_1) \rightarrow X_1, \min U(.) \rightarrow U_1,$$

$$YX(I) = (X_1 - BX) * MX, I=1, N3, N3 = \text{int}_2[N+1],$$

$$YU(I) = (U_1 - BU) * MU, I=1, N3, N3 = \text{int}_2[N+1], N = \{1, 2, \dots, n\}.$$

При этом на каждом i -м такте (в рассматриваемом примере начиная с t_{18})

5 осуществляется запоминание и накопление приращений, соответственно в $XI = XI + NX$ и $UI = UI + NU$, где NX и NU - постоянные приращения, зависят от физических размеров окна экрана для отображения на БВ У6 результатов верификации ИК ПО. На каждом из i циклов, синхронно с изменением значения i , осуществляется запись
10 полученных $YX(.)$ и $YU(.)$ во второе ЗУ У4.7 МФСКГАР У4 (см. фиг.29) по адресу $(i+k)$, где k - некоторое фиксированное (заданное) смещение адреса в ОЗУ. Начиная с такта $t_{k1} = 19 + 10N3$, начинает работу вторая схема У5.1 формирования дискретных приращений динамических координат графоаналитического растра МФДРГАР (см. фиг.30), которая обеспечивает приращения как со знаком "+" так и со знаком "-",
15 соответствующие амплитуды сигналов: YI^+ , ZI^+ и YI^- , ZI^- ; с постоянным приращением NY и NZ . Причем формирование каждого приращения

$YI^+(1)$ на такте $t_{k1} = 19 + 10N3$,

20 $YI^+(2)$ на такте $t_{k2} = 24 + 10N3$,

...

$YI^+(N3)$ на такте $t_{k3} = 14 + 15N3$,

совмещено с формированием на выходах блока преобразования статических координат графоаналитического растра У4.6:

25 $FX^+(1)$ на такте t_{s1} , $s_1 = 22 + 10N3$,

$FX^+(2)$ на такте t_{s2} , $s_2 = 27 + 10N3$,

...

30 $FX^+(N3)$ на такте t_{s3} , $s_3 = 17 + 15N3$.

Данные запоминаются во втором У4.7 по соответствующему i -му адресу. Начиная с $i = N31 = \text{int}_2[N3+1]$ такта осуществляется аналогичный циклический процесс формирования приращений $ZI^+(1), \dots, ZI^+(N3)$ и $FY^+(1), \dots, FY^+(N3)$.

35 Начиная с такта t_{k9} ($i = N4 = \text{int}_2[2 * N + 1]$), осуществляется формирование на основе циклических приращений следующих переменных:

$YI^-(1), YI^-(2), \dots, YI^-(N3)$ и $FX^-(1), FX^-(2), \dots, FX^-(N3)$.

Особенность формирования каждого из данных сигналов (например, амплитуда $FX^-(.)$ формируется за девять тактов), которые представляют координаты искомого отображения графоаналитического бинарного поля состояния процесса верификации ИК ПО, состоит в том, что этот процесс осуществляется без промежуточных операций запоминания соответствующих графических объектов в базах данных D4, D5 и D11.

Это позволяет достигнуть предельного быстродействия процесса генерации
45 искомым взаимосвязанных параметрических зависимостей в базах данных и базах знаний в виде обобщенных визуальных отображений динамики процесса верификации ИК ПО распределенных вычислительных комплексов. При этом в блоке У5.3 синхронизации (см. фиг.30) достаточно наличия простого цикла счета (например, последовательного счета импульсов) для формирования адреса записи/считывания.
50 Таким образом, описанное совмещение нескольких функций, связанных с формированием адресов (координат) записи их содержимого по каждому из данных адресов, обеспечивает наиболее высокое быстродействие получения визуальных

графо-аналитических образов многомерных данных на плоскости БВ У6.

Режим вывода сформированных графических объектов о состоянии процесса верификации ИК ПО в МР У6.1 (см. фиг.31) может быть осуществлен на основе использования стандартных средств, например, используя ЭЛТ с интерфейсным блоком в виде буферного ЗУ. При этом, если количество строк (разрешение) ЭЛТ равно v , а количество точек в строке n , то объем буферного ЗУ должен составлять $n \cdot v$. Режим вывода информации о состоянии процесса верификации ИК ПО на экран БВ У6 в этом случае будет осуществляться по аналогии режима ввода, при одном дополнительном условии - согласования его начала с началом кадрового синхроимпульса. При этом строб готовности (начало кадра) устанавливает на адресных входах буферного ЗУ нулевой адрес и по каждому синхросигналу увеличивается текущее значение адреса на единицу. В этом случае максимальный адрес будет равен $(n \cdot v - 1)$. Следовательно, информация в каждой ячейке буферного ЗУ МР У6.1 должна храниться в течение времени развертки одного кадра.

Проекция электронного луча на экране электронно-лучевой трубки МР У6.1 передается на рабочую поверхность преобразователя оптического сигнала У6.4 (в простейшем случае это может быть фотодатчик или ПЗС-матрица) через оптическую систему У6.3 и маску У6.2, выполненную так, что в трехмерном пространстве выделены области, соответствующие гипотезам о различных состояниях уязвимости анализируемого ИК ПО.

При этом оптический коэффициент пропускания маски У6.2, через которую проходит оптический сигнал, имеет одно и то же значение для точек одной гипотезы о состоянии уязвимости ИК ПО и различное для других значений гипотез. Маска У6.2 изготовлена так, что в пространстве ее координат (X, Y, Z) выделены области уязвимости ИК ПО и соответствующие значения могут быть определены из следующего выражения энтропии:

$$H(X_i, Y_j, Z_k) \approx \int_0^1 \int_0^1 \int_0^1 f(x_i, y_j, z_k) dx dy dz, \quad i, j, k = 1, 2, \dots,$$

где $f(.)$ - любая интегрируемая по Риману функция или параметрическая зависимость оценки уязвимости ИК ПО (см. пример №2),

X_i, Y_j и Z_k - текущие значения переменных (или параметрических зависимостей) - входных переменных - (параметров) независимых (или зависимых) координат, о различных состояниях верифицируемого ИК ПО.

Оптический коэффициент пропускания маски У6.2 имеет одно и то же значение для точек, принадлежащих одному и тому же значению $H_{ijk}(.)$, и различное - для других значений гипотез уязвимости $H_{ijk}(.)$ ИК ПО. Данное свойство может быть реализовано, например, с помощью интенсивности почернения, густотой штриховки или частотой точек соответствующих участков маски при учете апертуры электронного луча и разрешающей способности преобразователя оптического сигнала (в простейшем случае - фотоприемника, ПЗС-матрицы и т.п.). Это позволяет при входных величинах X_i, Y_j и Z_k получать на выходе 2 блока динамической развертки У6.5 сигнал с амплитудой, равной $H_{ijk}(.)$, и с частотой, определяемой параметрами соответствующих генераторов развертки.

В простейшем случае реализации БВ У6 электронный луч перемещается по экрану модуля регистрации У6.1 (см. фиг.31) по заданной траектории на основе управляющих сигналов, поступающих на его вход 2 из блока динамической развертки У6.5.

Рассмотрим процесс формирования периода и длительности следования

управляющих воздействий на выходе 1 импульсного регулятора У7, когда на его входе 1 абсолютное значение гипотез уязвимости $|H_{ijk}(\cdot)|$ ИК ПО, получаемое на выходе блока определения абсолютных значений У7.1, меньше $\Delta_n/2$, где $\Delta_n/2$ - зона срабатывания первого релейного элемента У7.2 (см. фиг.32, пример реализации импульсного регулятора) и соответствующее ей нормированное значение $|e(t)| < 1$. Предполагаем, что нормирование текущих значений $H_{ijk}(\cdot)$ (соответственно значения ошибки определения точек уязвимости ИК ПО - $e(t)$) осуществляется с коэффициентом $\Delta = \Delta_n/2$.

Пусть в исходном положении триггер У7.6 находится в единичном состоянии, тогда на выходе первого блока выборки-хранения У7.7 будет напряжение

$$U_i = \int_{t_{i-1}}^{t_{i-1}+T_{i-1}} e(t) | dt,$$

которое подается на первый вход первого сумматора У7.12, на второй вход которого поступает постоянное напряжение (Е) от источника эталонных напряжений У7.11.

В результате на вход первого формирователя импульсов У7.10 подается постоянное напряжение вида:

$$U_{\phi i} = U_i + E.$$

Постоянная составляющая Е может быть выбрана, например, из интервала значений с нижней границей, определяемой длительность выполнения самой продолжительной последовательности операций преобразований (обработки) в АПБ устройства верификации ИК ПО, и верхней границей, определяемой, например, как $1/(K_f \cdot F_{\max})$, где $F_{\max}$ - максимальная частота изменения значений $H_{ijk}(\cdot)$, K_f - постоянный коэффициент, который согласно теореме отсчетов во временном представлении непрерывных сигналов в дискретной форме можно установить ≥ 2 .

В этой связи верхняя граница значений данного интервала может быть априорно выбрана из необходимого условия адекватности представления процесса верификации ИК ПО своими дискретными значениями. Это позволяет обеспечить измерение и обработку значений сигнала $H_{ijk}(\cdot)$ с наименьшей потерей информации о состоянии процесса верификации ИК ПО.

Далее в соответствии с $U_{\phi i}$ на выходе формирователя импульсов У7.10 будет сформирован импульс длительностью:

$$T_i = K_{\phi i} \cdot U_{\phi i},$$

где $K_{\phi i}$ - коэффициент передачи формирователя импульсов У 7.10.

Во время действия данного периода T_i в первом блоке выборки-хранения У7.7 формируется напряжение вида:

$$U_i = \int_{t_{i-1}}^{t_{i-1}+T_{i-1}} e(t) | dt.$$

В конце периода T_i формирователь импульсов У7.10 устанавливает триггер У7.6 в следующее (например, нулевое) устойчивое состояние.

Описанные выше последовательности операций непрерывно повторяются в функциональной зависимости от изменения ошибки определения точек уязвимости ИК ПО - $e(t)$. При этом на каждом из периодов T_i , $i=1, 2, \dots, n$, на первый вход первого логического элемента И У7.3 от первого формирователя импульсов У7.10 подается разрешающий сигнал на включение ключа У7.15, коммутирующего входную цепь

второго формирователя импульсов У7.16. В момент окончания каждого периода T_i и начала следующего T_{i+1} , составляющие незначительный по продолжительности промежуток времени, формируется сигнал, достаточный для отключения электронного ключа У7.15. Этим обеспечивается требуемая степень синхронизации между началом момента формирования периода управляющих воздействий в первом формирователе импульсов У7.10 и длительности импульса во втором формирователе импульсов У7.16 для всех T_i , $i=1,2,\dots,n$.

При этом формирование напряжений на входе второго формирователя импульсов У7.16 реализовано по схеме, аналогичной вышеописанной на входе первого формирователя импульсов У7.10, с тем лишь отличием, что формирование длительности (t_i) выходных импульсов осуществляется в соответствии со значением оценки интегрального изменения квадрата нормированного сигнала $H_{ijk}(\cdot)$ (ошибки определения точки уязвимости ИК ПО) и во все периоды T_i , для которых

$$\Delta_{mp} \leq |e(t)| \leq \Delta_n / 2,$$

где Δ_{tr} - зона нечувствительности второго релейного элемента У7.4, устанавливаемая на основе критерия точности воспроизведения (распознавания, прогнозирования и т.п. точек уязвимости ИК ПО распределенных вычислительных комплексов) текущих значений сигнала $H_{ijk}(\cdot)$. Контроль за выполнением этих условий реализован с помощью первого логического элемента И У7.3, управляющего состоянием электронного ключа У7.15 в соответствии с сигналами, поступающими с первого формирователя импульсов У7.10 и релейных элементов У7.2 и У7.4, контролирующих значения $\Delta_n/2$ и Δ_{tr} соответственно. Следовательно, формирование каждого периода T_i осуществляется на основе последовательности операций $(K_{fi}^* E)$, априорных по отношению к динамике изменения значений сигнала $H_{ijk}(\cdot)$, и корректируется текущими значениями интегральных оценок нормированной ее оценки, полученными на предшествующем периоде дискретизации процесса верификации ИК ПО распределенных вычислительных комплексов.

При этом для всех $e(t) \in [-\Delta_n/2, \Delta_n/2]$ и $t > 0$ всегда выполняется неравенство вида $T_i > t_i$, за исключением только тех моментов времени, для которых вероятность наличия уязвимости в ИК ПО равна нулю (т.е., $e(t)=0$) или она не выходит за пределы допустимых значений (а именно, не больше $|e(t)| = \Delta_n/2$).

Таким образом, отличительные признаки предлагаемого способа позволяют расширить функциональные возможности основных процессов верификации ИК ПО распределенных вычислительных комплексов непосредственно при выполнении научно-исследовательских работ на основе использования реальных динамических характеристик распределенных вычислительных систем и сетей, при этом предоставляется возможность повысить достоверность и объективность принимаемых решений на основных этапах исследования, разработки и сопровождения ПО распределенных вычислительных комплексов.

Формула изобретения

1. Способ генерации баз знаний для систем верификации программного обеспечения (ПО) распределенных вычислительных комплексов (РВК), включает совмещение процессов ввода и обработки исходного кода ПО по зависимым или независимым каналам на основе использования сенсорных или механических

манипуляторов рабочего места оператора ЭВМ, сетевых интерфейсов локальной или глобальной сети, при этом участки или точки уязвимости исходного кода ПО определяют на основе преобразования исходного кода ПО во внутреннее представление, которое хранят в виде баз знаний, а точки или участки уязвимости исходного кода ПО определяют на основе автоматического составления и решения соответствующих систем уравнений, в процессе верификации исходного кода ПО на экране блока видеоконтроля (БВ) аппаратно-программного блока процессорного управления осуществляют синтаксическую подсветку для визуализации участков уязвимости исходного кода ПО, при этом используют граф операторов исходного кода ПО во внутреннем представлении инструкций языка программирования.

2. Способ по п.1, отличающийся тем, что диагностику процесса верификации исходного кода ПО осуществляют на основе измерения длительности выполнения основных этапов (циклов) поиска уязвимости и сопоставления полученных значений с ранее предписанными или предсказанными оценками показателя критичности уязвимости исходного кода ПО.

3. Способ по п.1, отличающийся тем, что внутреннее представление исходного кода ПО на языке программирования Си представляется в виде динамических массивов и соответствующих баз знаний, которые используются в реальном масштабе времени для верификации ПО распределенных вычислительных комплексов.

4. Способ по п.1, отличающийся тем, что точки или участки уязвимости исходного кода ПО определяют на основе автоматического составления и решения линейных систем уравнений.

5. Способ по п.1, отличающийся тем, что в качестве зависимых каналов используют интерфейсы жестких, гибких или оптических дисков.

6. Способ по п.1, отличающийся тем, что в качестве независимых каналов используют интерфейс последовательного порта или сетевой интерфейс.

7. Способ по п.1, отличающийся тем, что в качестве независимых каналов используют интерфейс последовательного порта или сетевой интерфейс, а в качестве зависимых каналов используют интерфейсы жестких, гибких или оптических дисков.

8. Способ по любому из пп.1-7, отличающийся тем, что осуществляют генерацию баз знаний на основе использования аннотаций внешних функций ИК ПО РВК, при этом база знаний представляет набор правил или моделей протоколов верификации ИК ПО, обеспечивающих поддержку интерактивных процессов поиска уязвимости ПО.

9. Устройство для реализации способа генерации баз знаний для систем верификации программного обеспечения распределенных вычислительных комплексов (СВПО РВК), содержит аппаратно-программный блок (АПБ) лексического и семантического анализа/разбора, АПБ преобразования кода, АПБ анализа кода, АПБ процессорного управления, видеоадаптер, интерфейсы жестких, гибких и оптических дисков, интерфейс последовательного порта, сетевой интерфейс и системную память, которые объединены системной шиной, при этом системная память содержит постоянное запоминающее устройство (ROM) и оперативное запоминающее устройство (RAM/ОЗУ), в ячейках оперативной памяти и жестких дисков размещают/записывают операционные системы, прикладные программы, базы данных и базы знаний, которые содержат листинги исходных программ, грамматику языка программирования (например грамматику языка программирования Си), правила преобразования дерева разбора листинга программы, дерево разбора листинга программы, таблицу типов языка программирования, аннотации внешних

функций, включающие их грамматику и семантику, код программ на языке внутреннего представления, условия корректности языка внутреннего представления исходного кода программы, условия проверки корректности подозрительных точек исходного кода программы, информационная база, содержащая системы ограничений в виде алгебраических уравнений и неравенств, отчеты об обнаруженных уязвимостях программного кода, включает:

указание на местоположение возможной уязвимости в исходном коде программы, которое содержит имя файла листинга программы, номер строки и номер позиции в строке программы в котором возможно переполнение буфера запоминающего устройства, контекст исходной программы, содержащий возможное переполнение или некоторую "окрестность" потенциально опасной точки переполнения буфера запоминающего устройства;

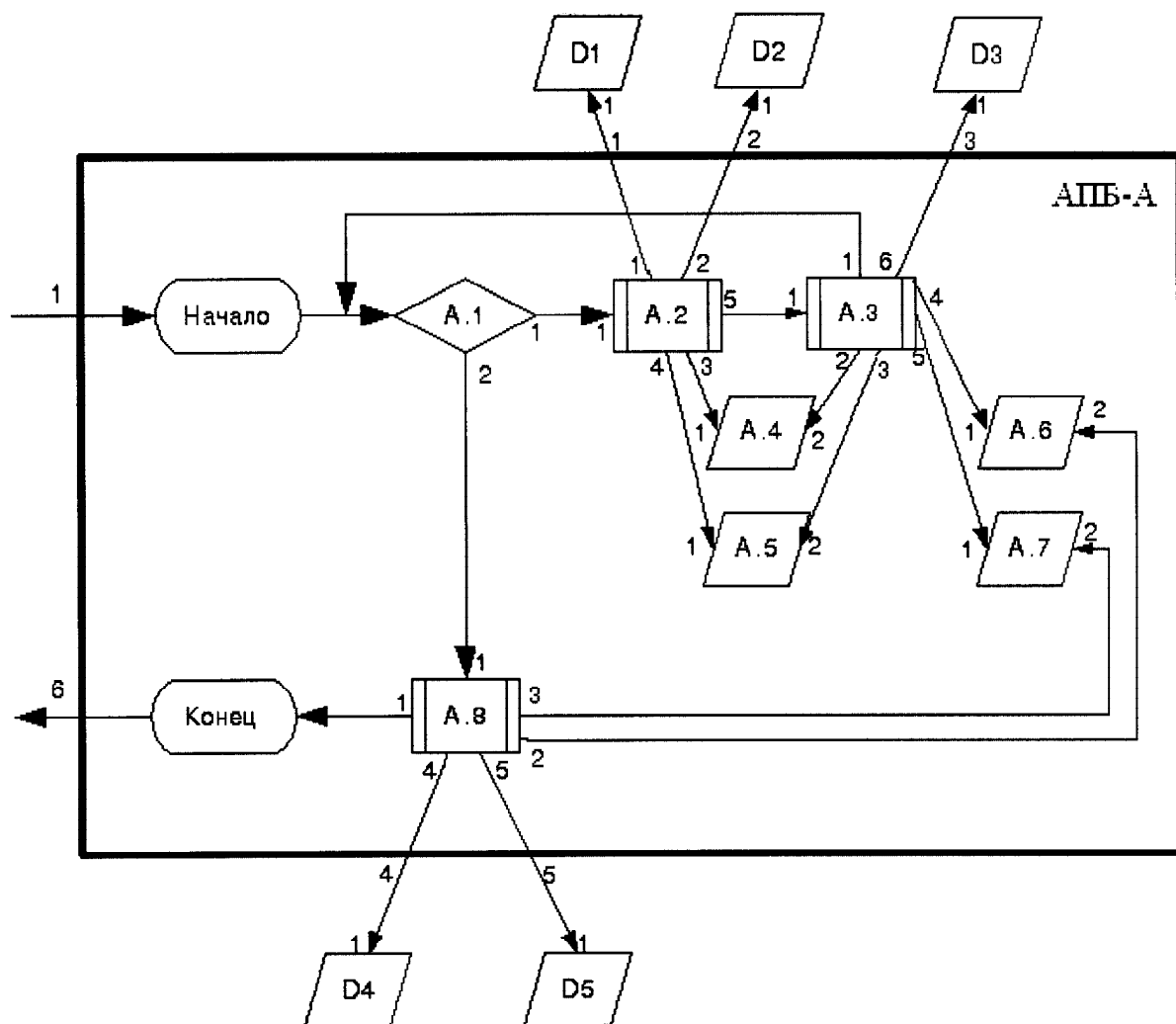
указание причины переполнения буфера запоминающего устройства - значения исходных переменных, приводящих к возникновению уязвимости исходного кода программного обеспечения;

показатель или степень критичности обнаруженной уязвимости исходного кода программного обеспечения;

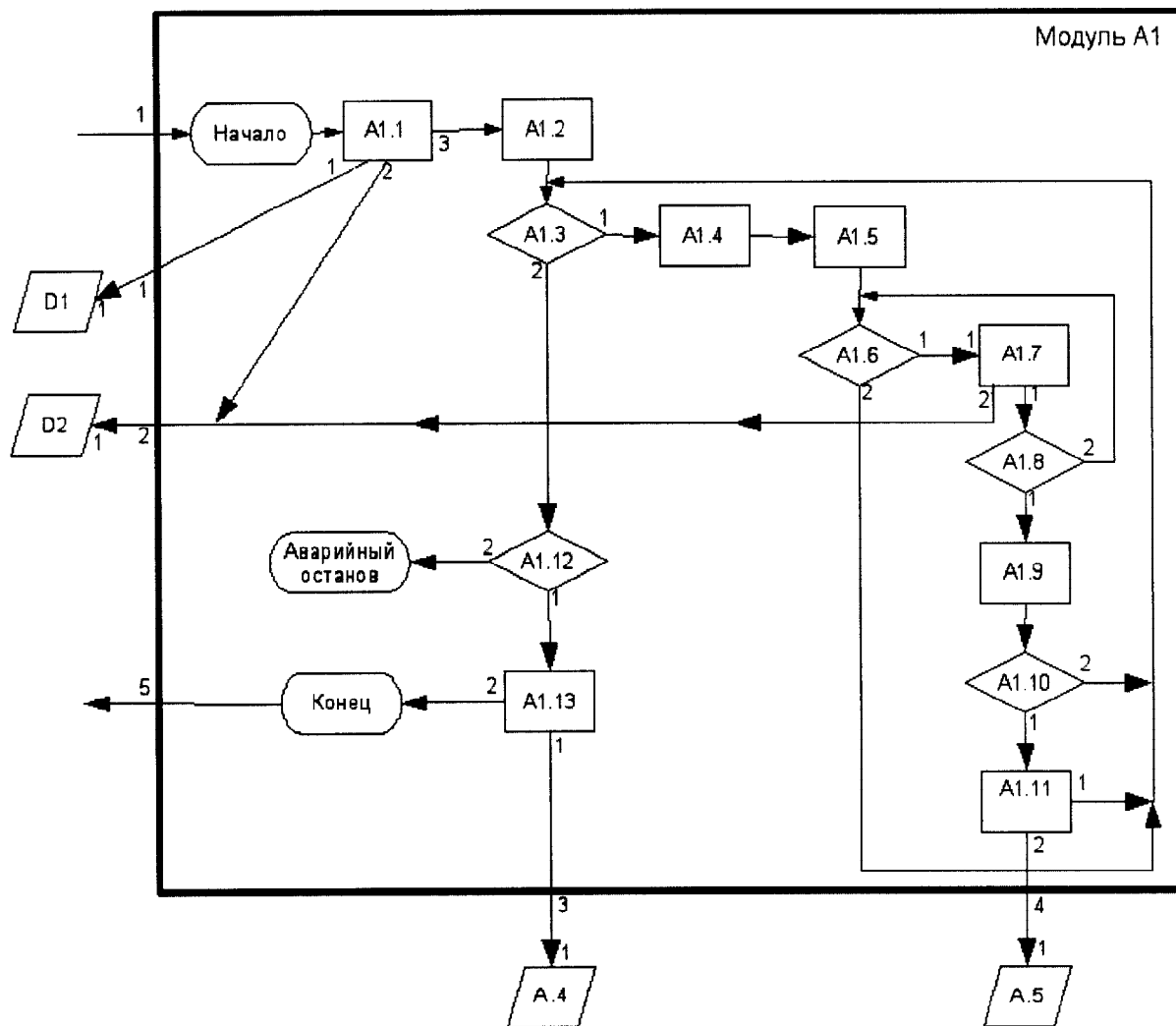
указание на перечень правил или алгоритмов для устранения уязвимости исходного кода программного обеспечения,

при этом АПБ процессорного управления, обеспечивающий синхронизацию основных режимов интерактивной верификации программного обеспечения распределенных вычислительных комплексов, содержит последовательно соединенные блок нормирующих преобразователей (БНП), модуль коммутатора, модуль аналого-цифрового преобразователя (АЦП), модуль формирования статических координат графоаналитического раstra (МФСКГАР), блок видеоконтроля (БВ), импульсный регулятор (ИР) и модуль формирования динамической развертки графоаналитического раstra (МФДРГАР), первый и второй информационные входы/выходы которого соединены соответственно с управляющими входами модуля коммутации и аналого-цифрового преобразователя, а третий, четвертый, пятый, шестой, седьмой и восьмой - с вторым, третьим, четвертым, пятым, шестым и седьмым информационными входами/выходами МФСКГАР, при этом второй вход блока видеоконтроля объединен с седьмым информационным входом/выходом МФСКГАР и восьмым информационным входом/выходом МФДРГАР,

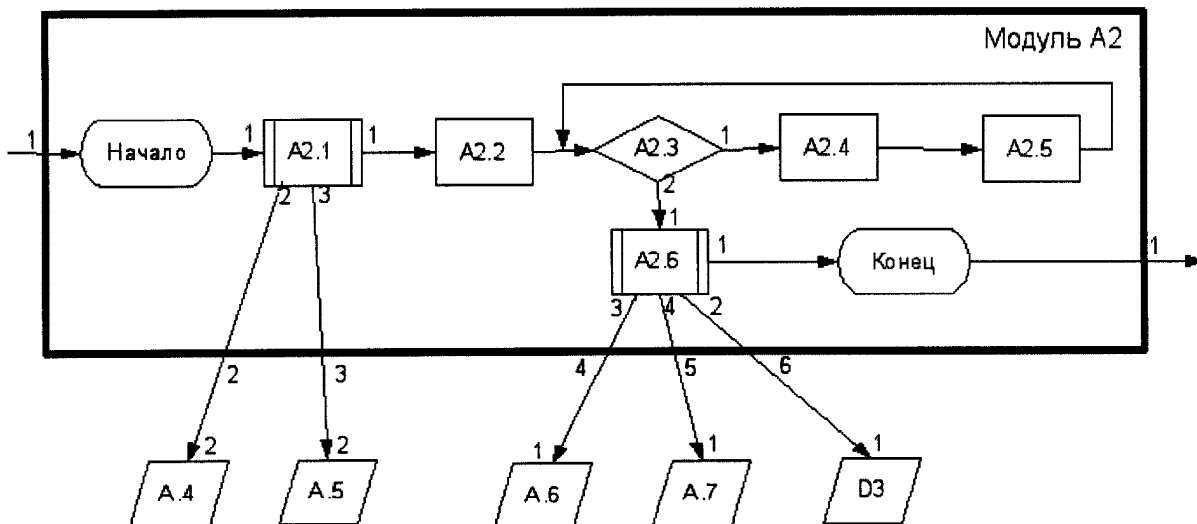
блок видеоконтроля (БВ) содержит модуль регистратора (МР), на выходе которого установлены маска, оптическая система и преобразователь оптического сигнала, выходом подключенный к управляющему входу блока динамической развертки, при этом оптический коэффициент пропускания маски имеет одно и то же значение для точек одной гипотезы/энтропии о состоянии уязвимости ИК ПО и различное для других значений гипотез.



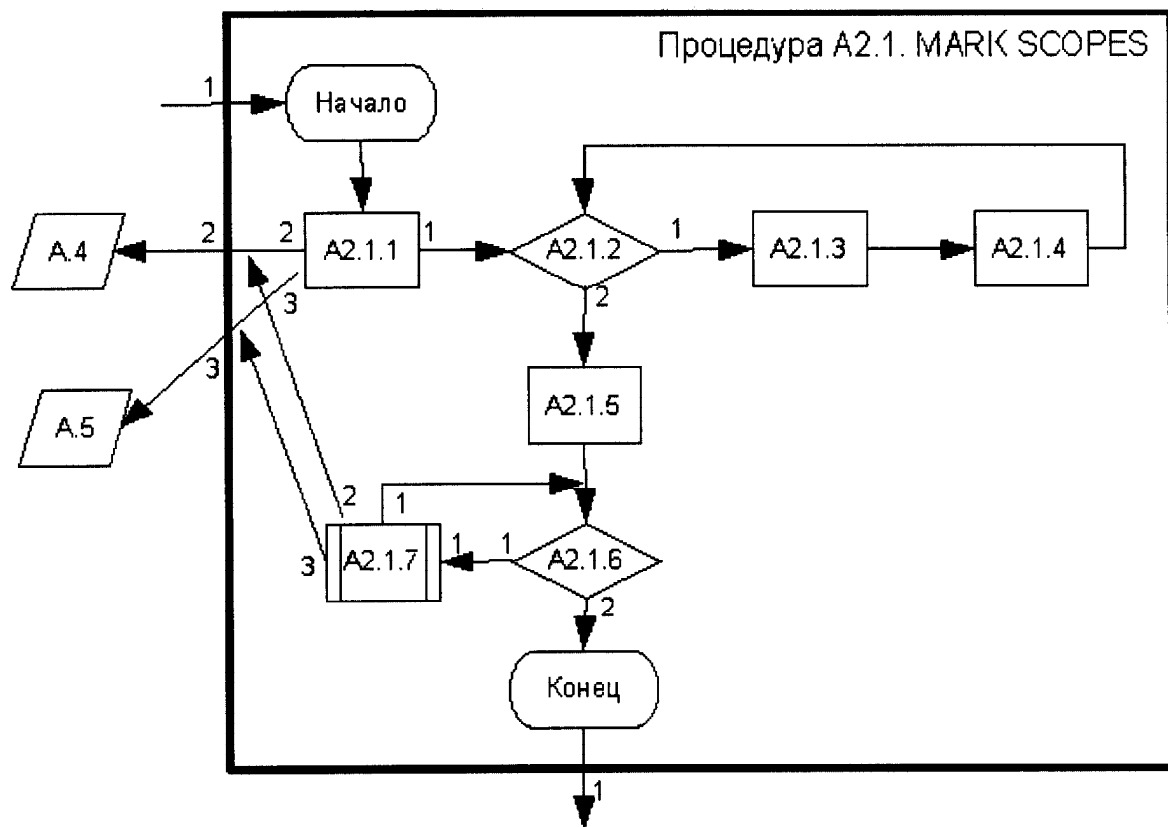
Фиг. 2



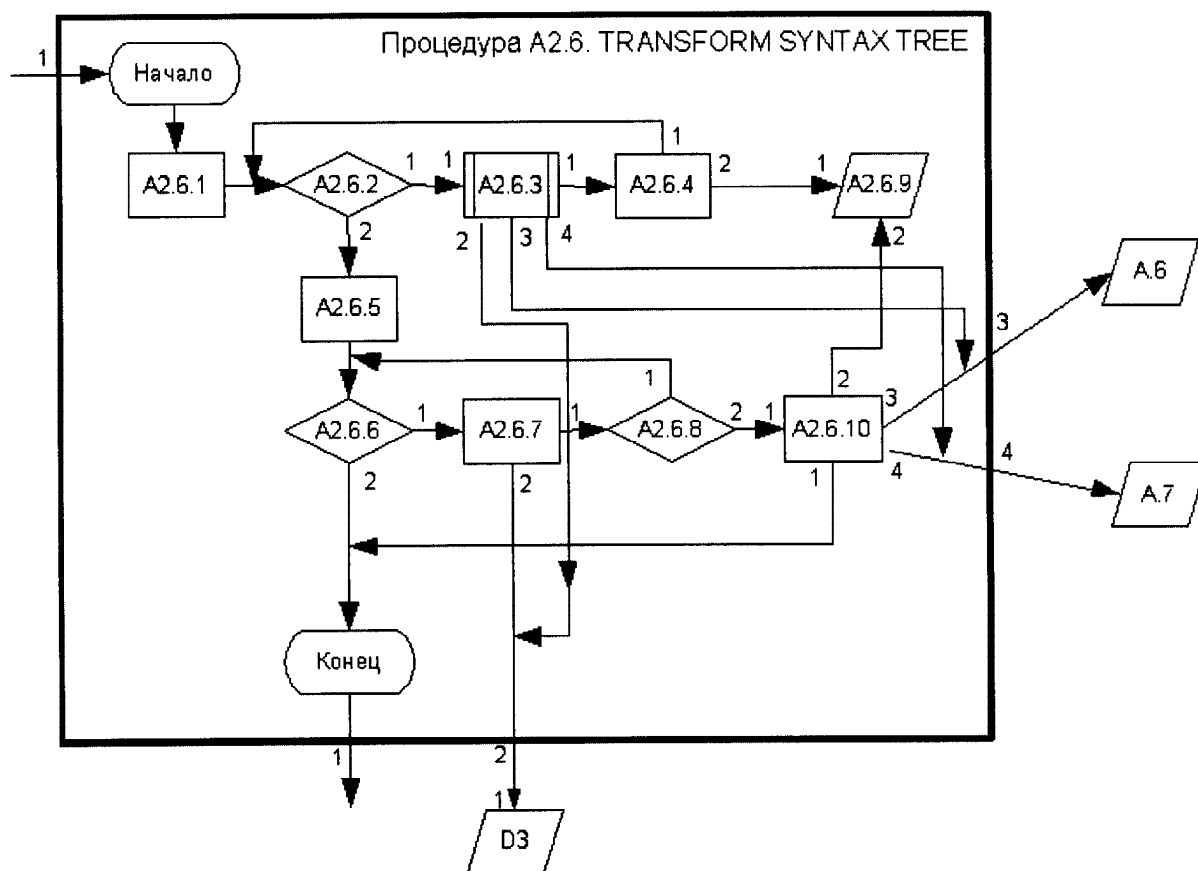
Фиг. 3



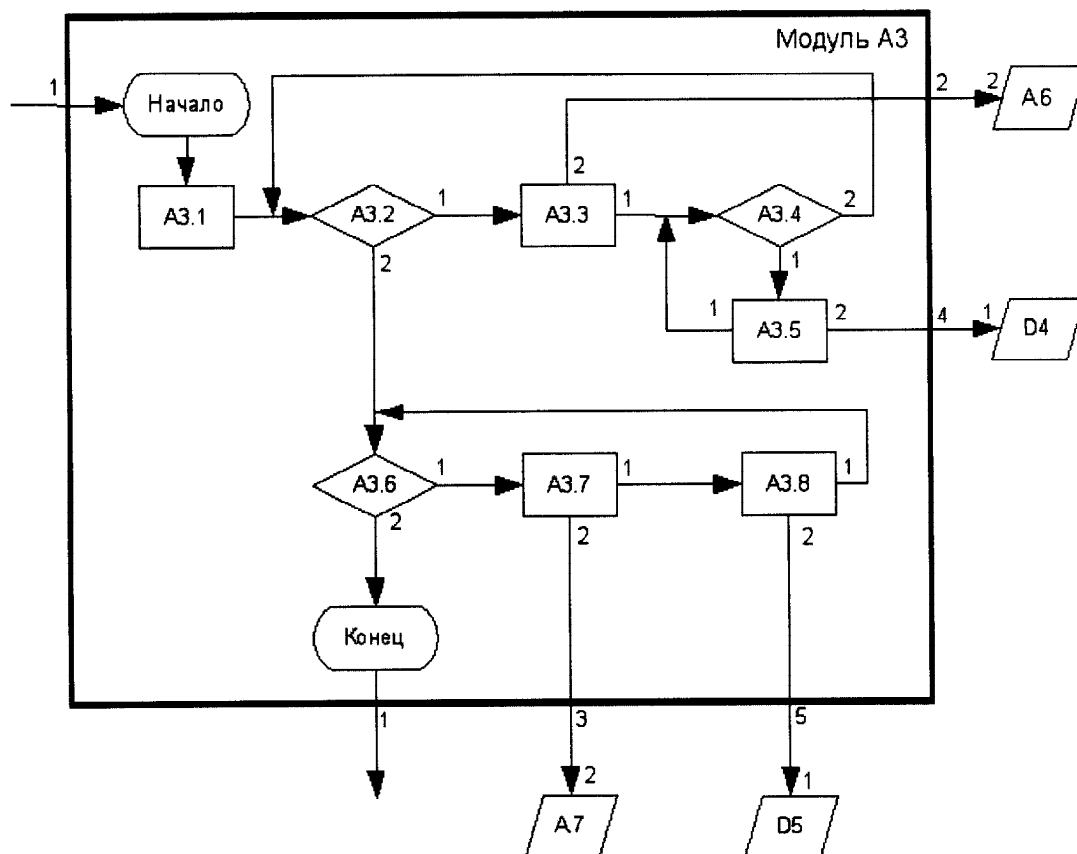
Фиг. 4



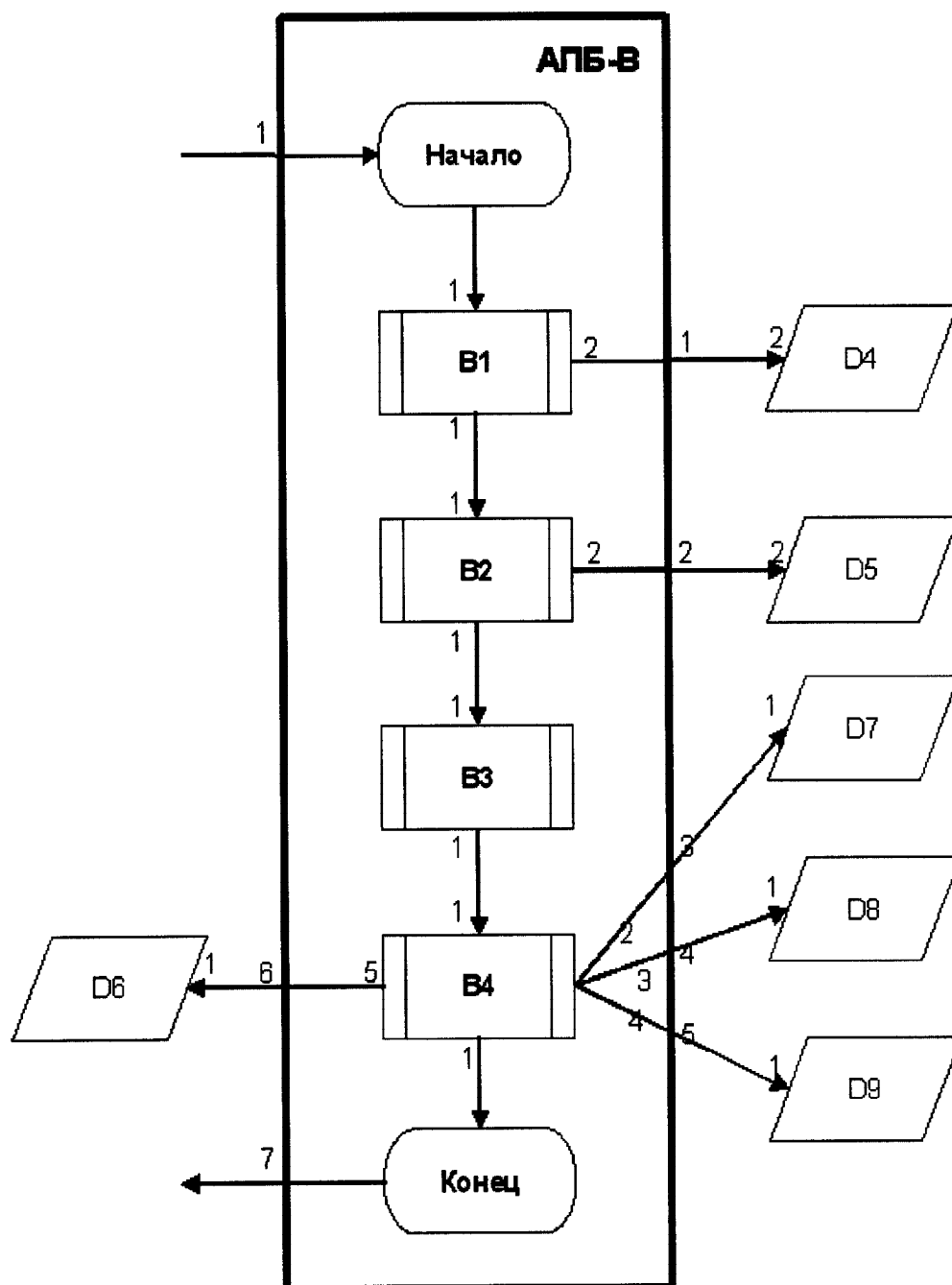
Фиг. 5



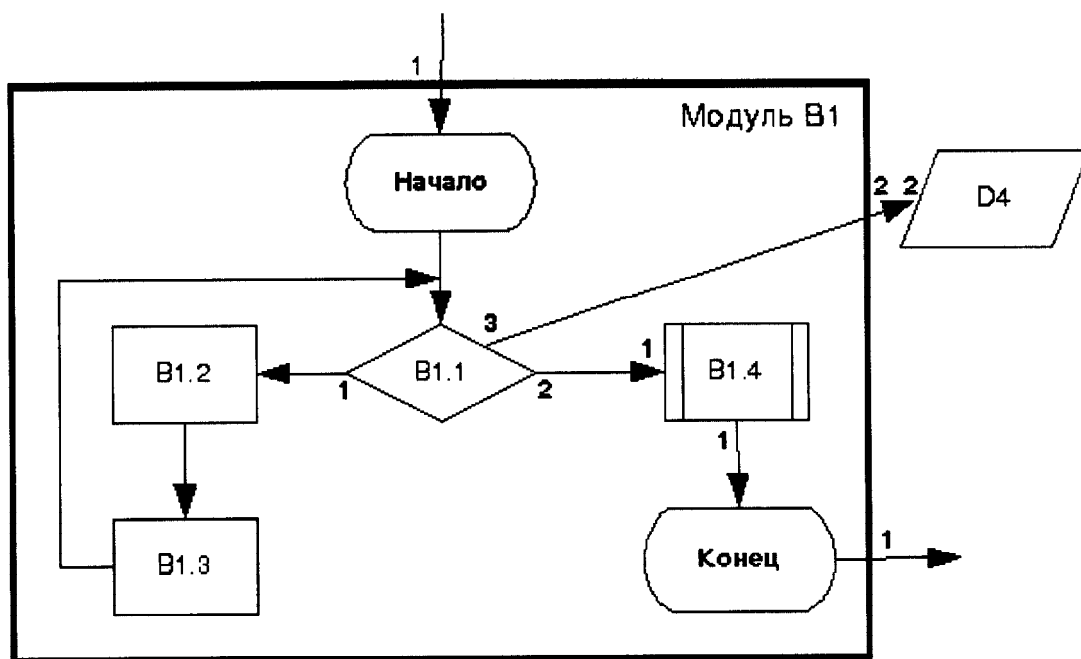
Фиг. 6



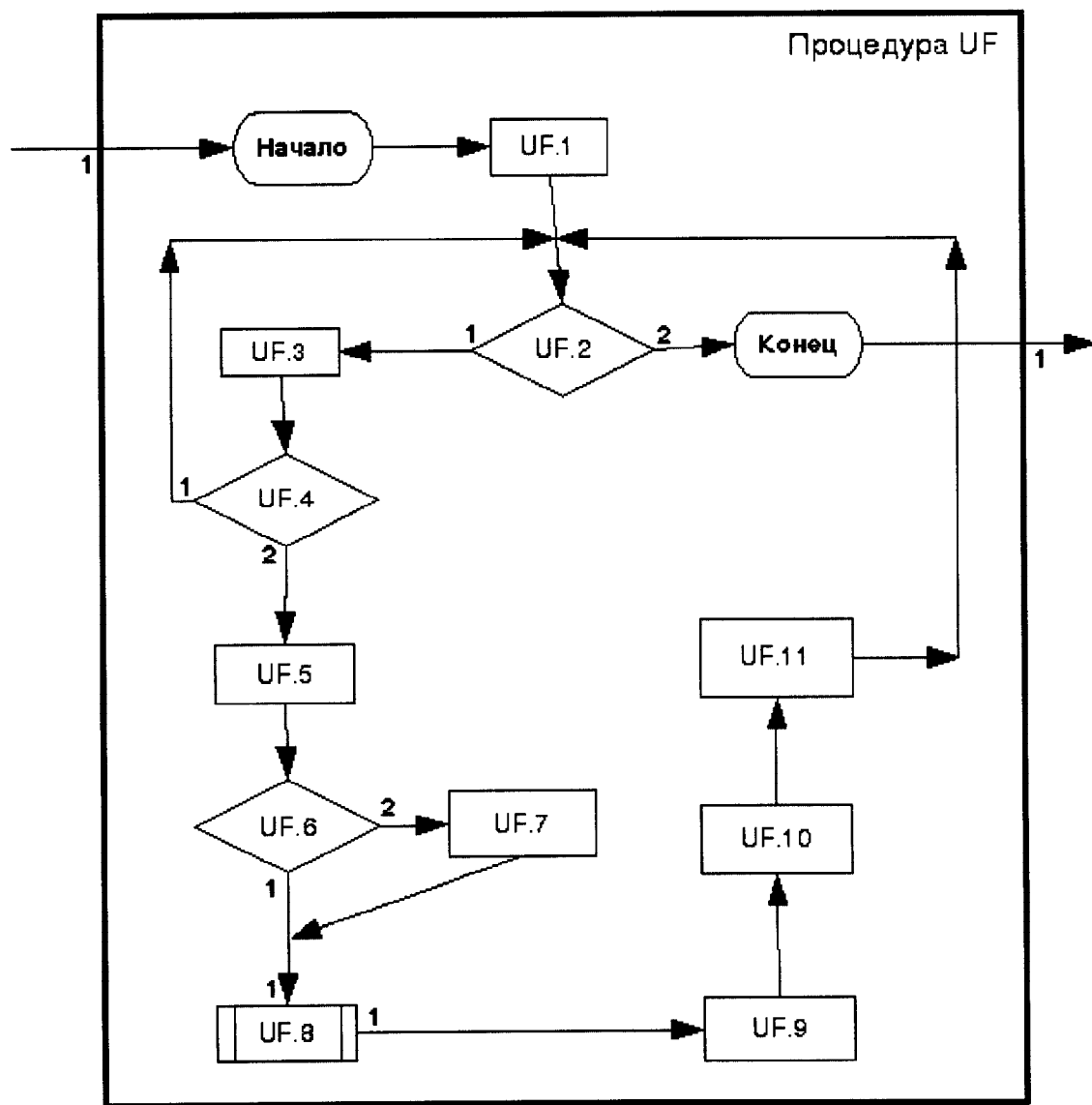
Фиг. 7



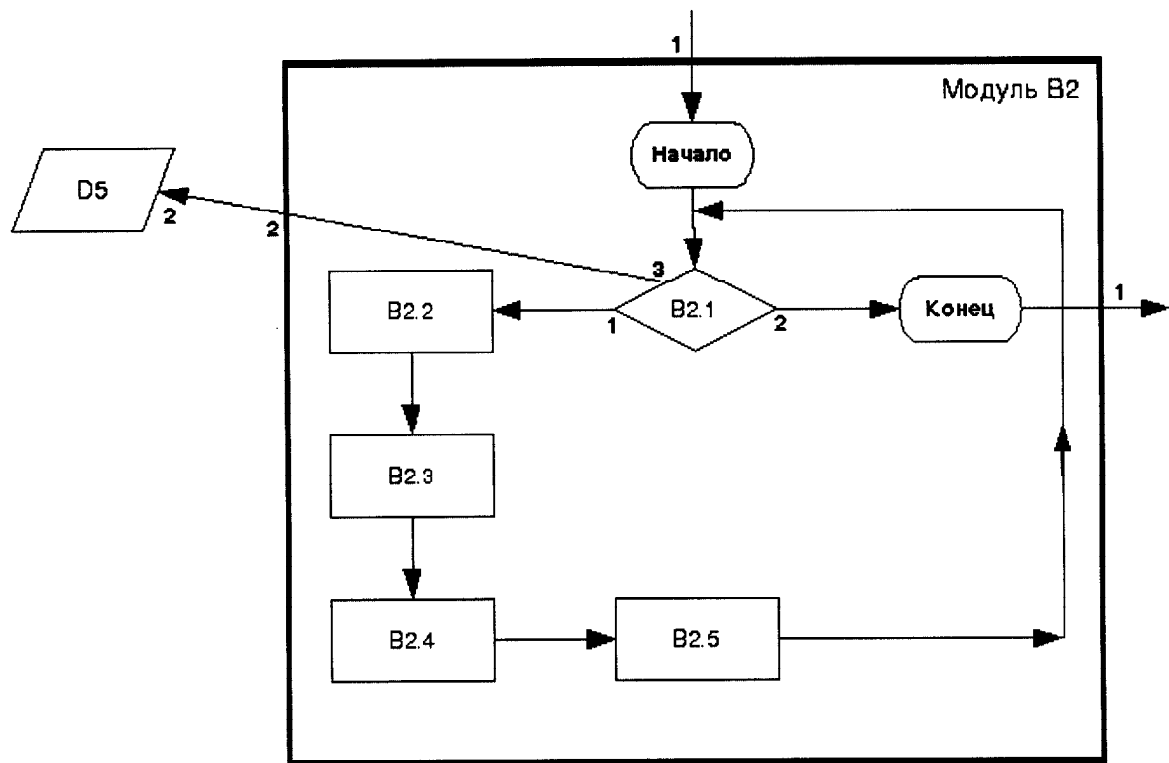
Фиг. 8



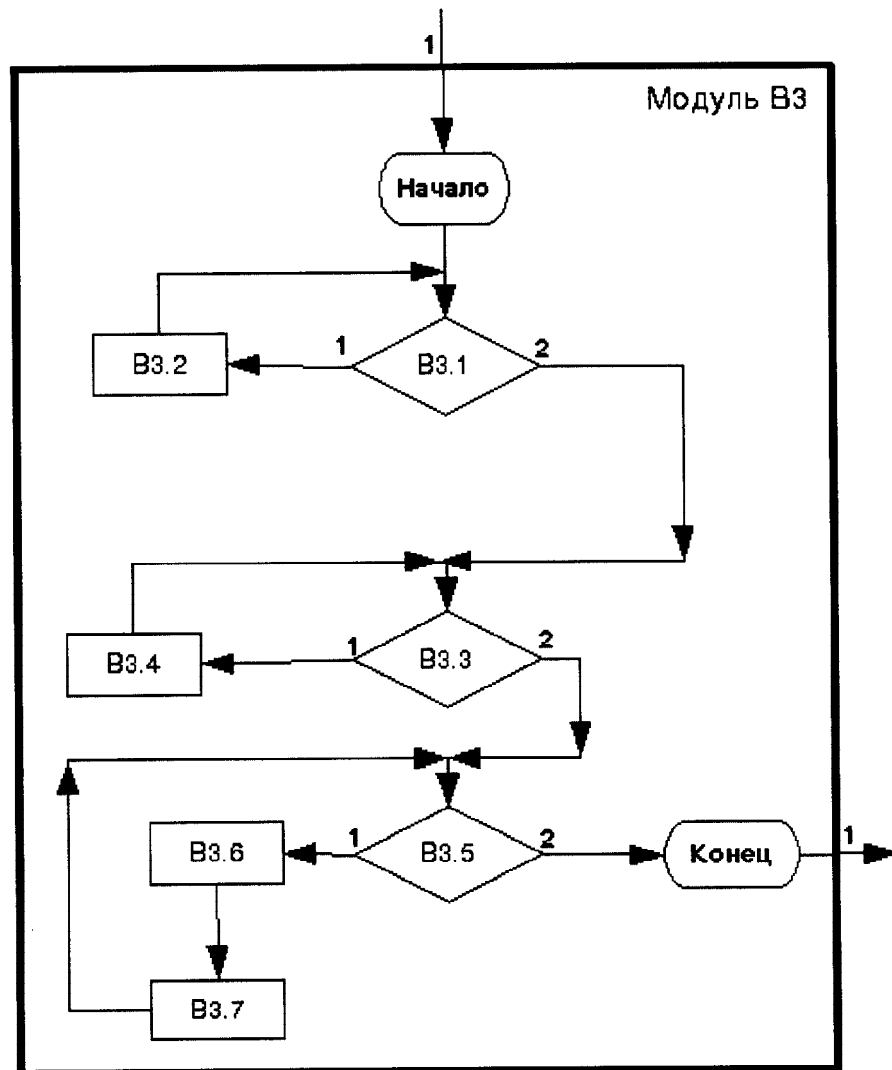
Фиг. 9



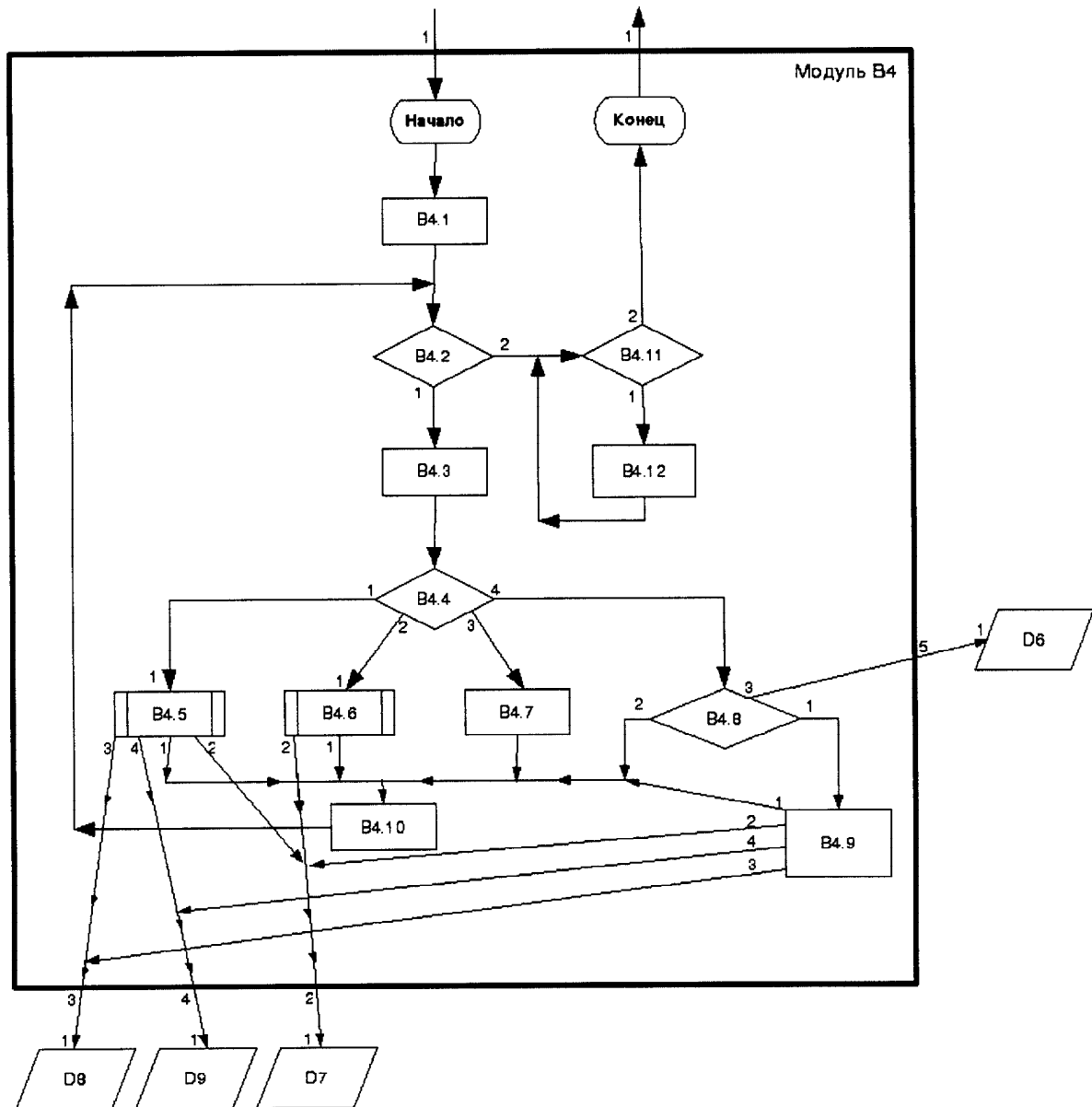
Фиг. 10



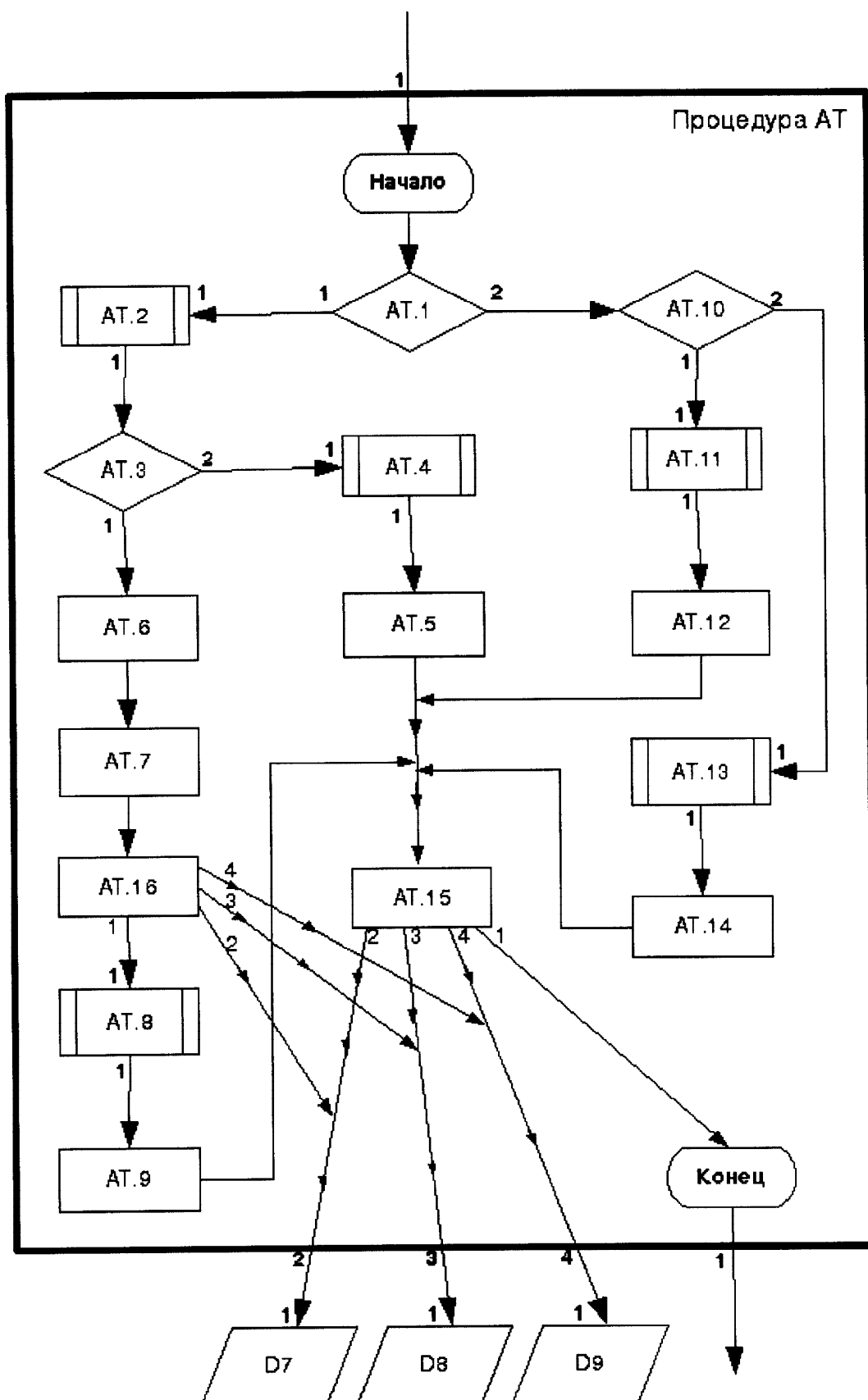
Фиг. 11



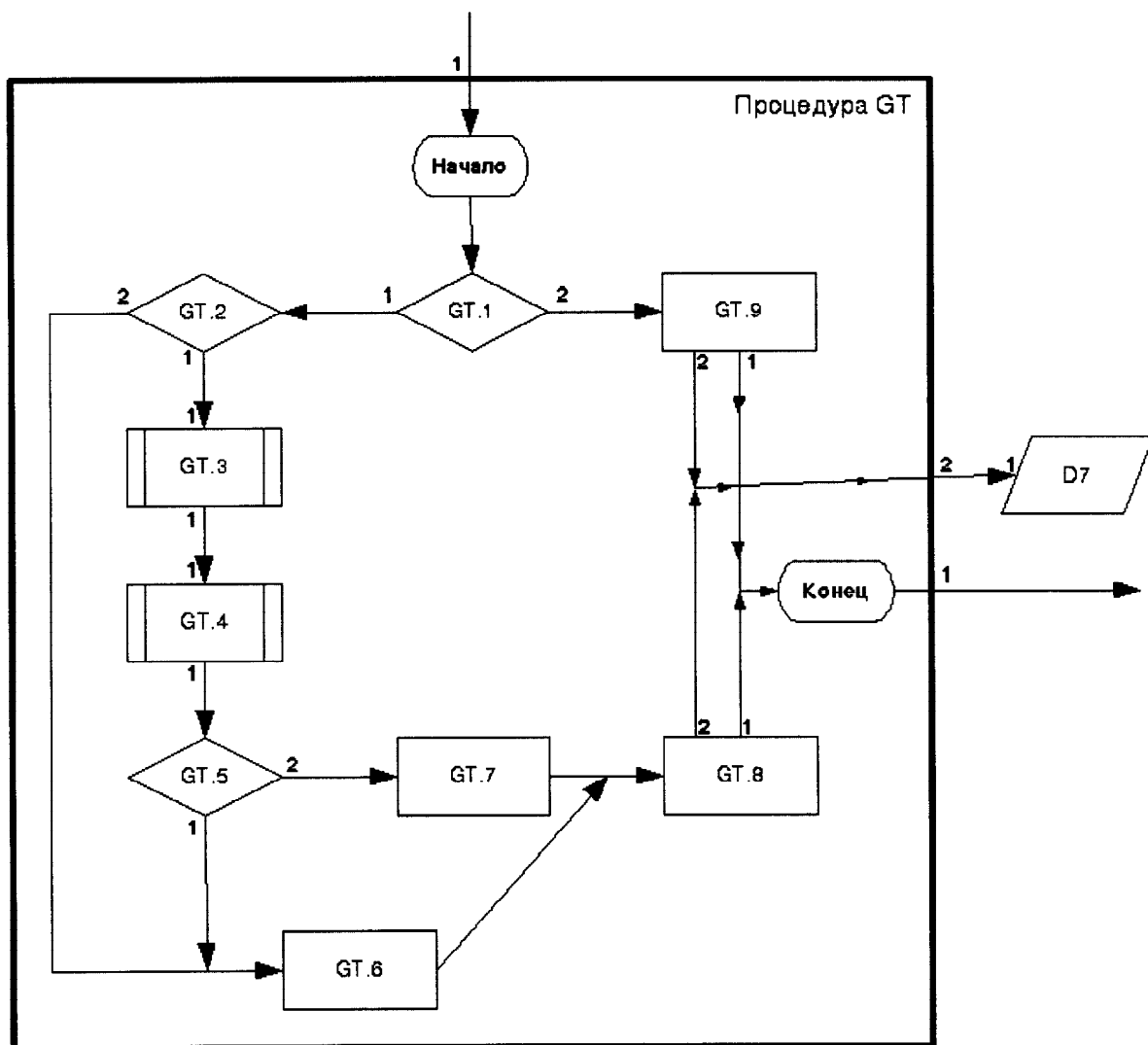
Фиг. 12



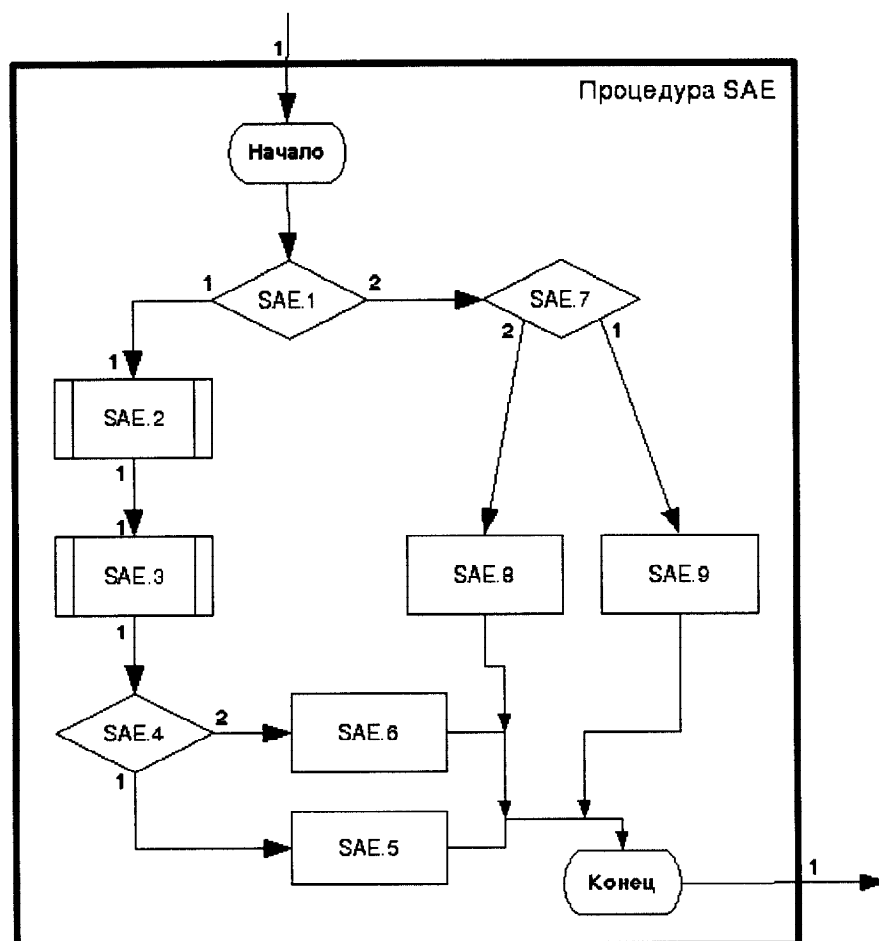
Фиг. 13



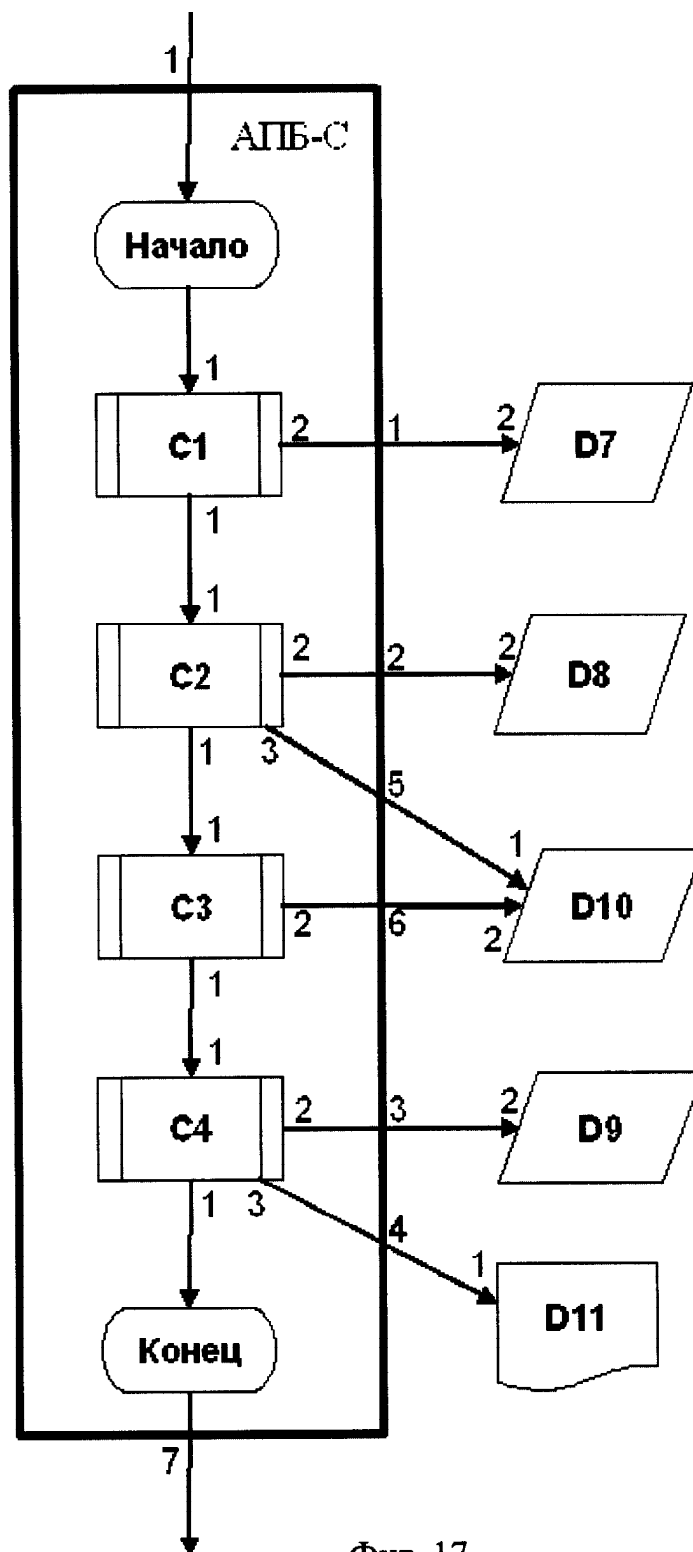
Фиг. 14



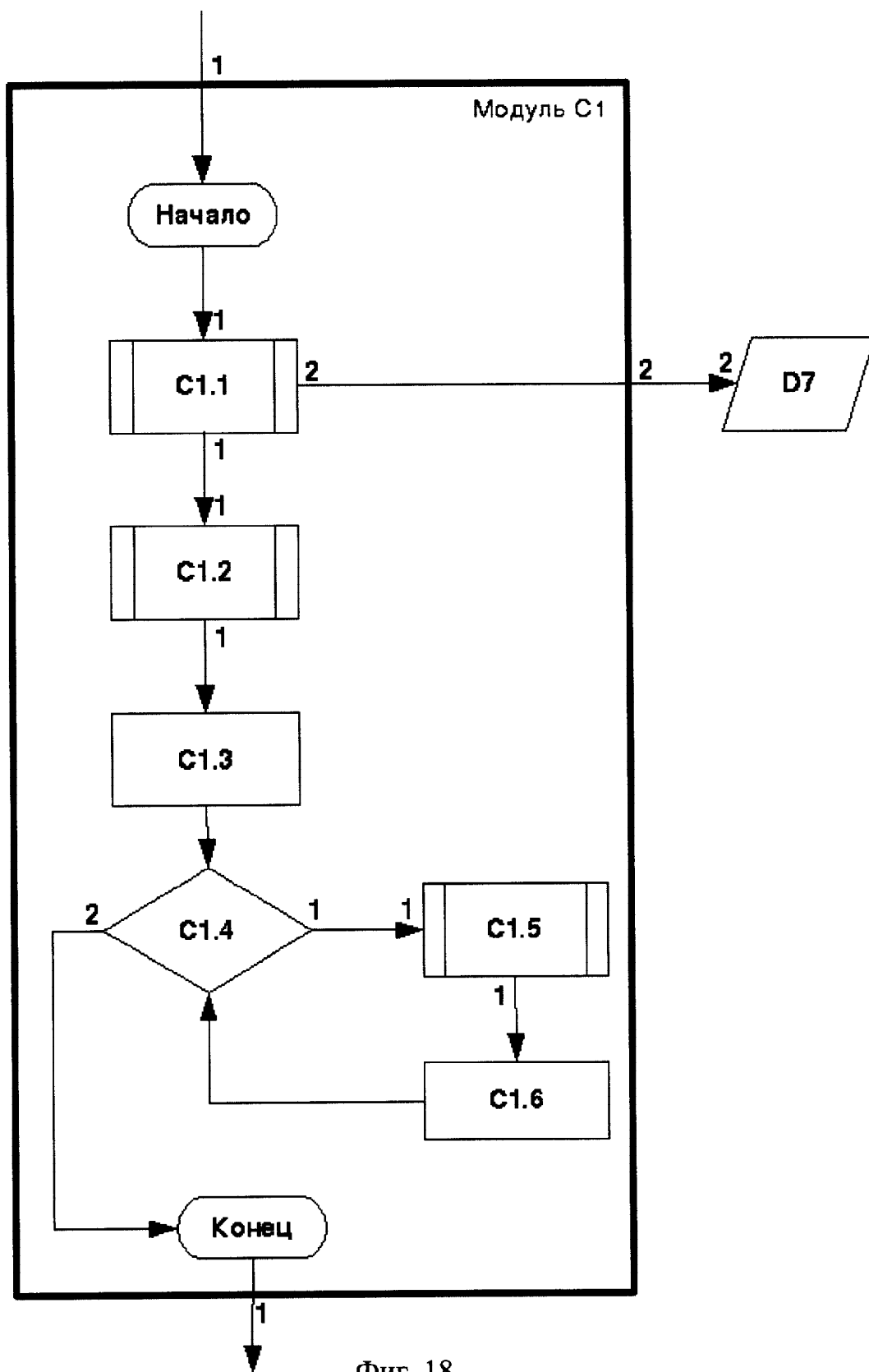
Фиг. 15



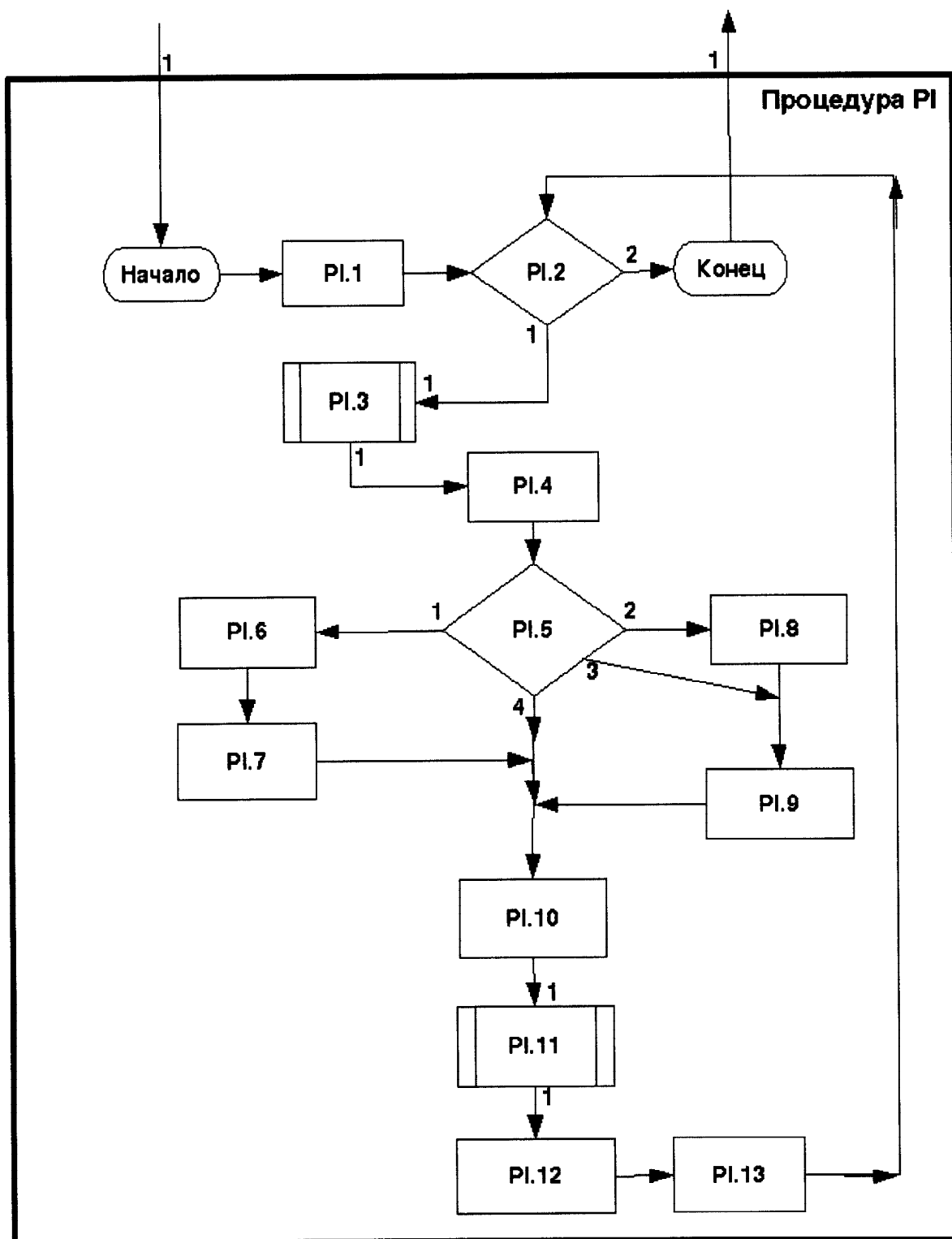
Фиг. 16



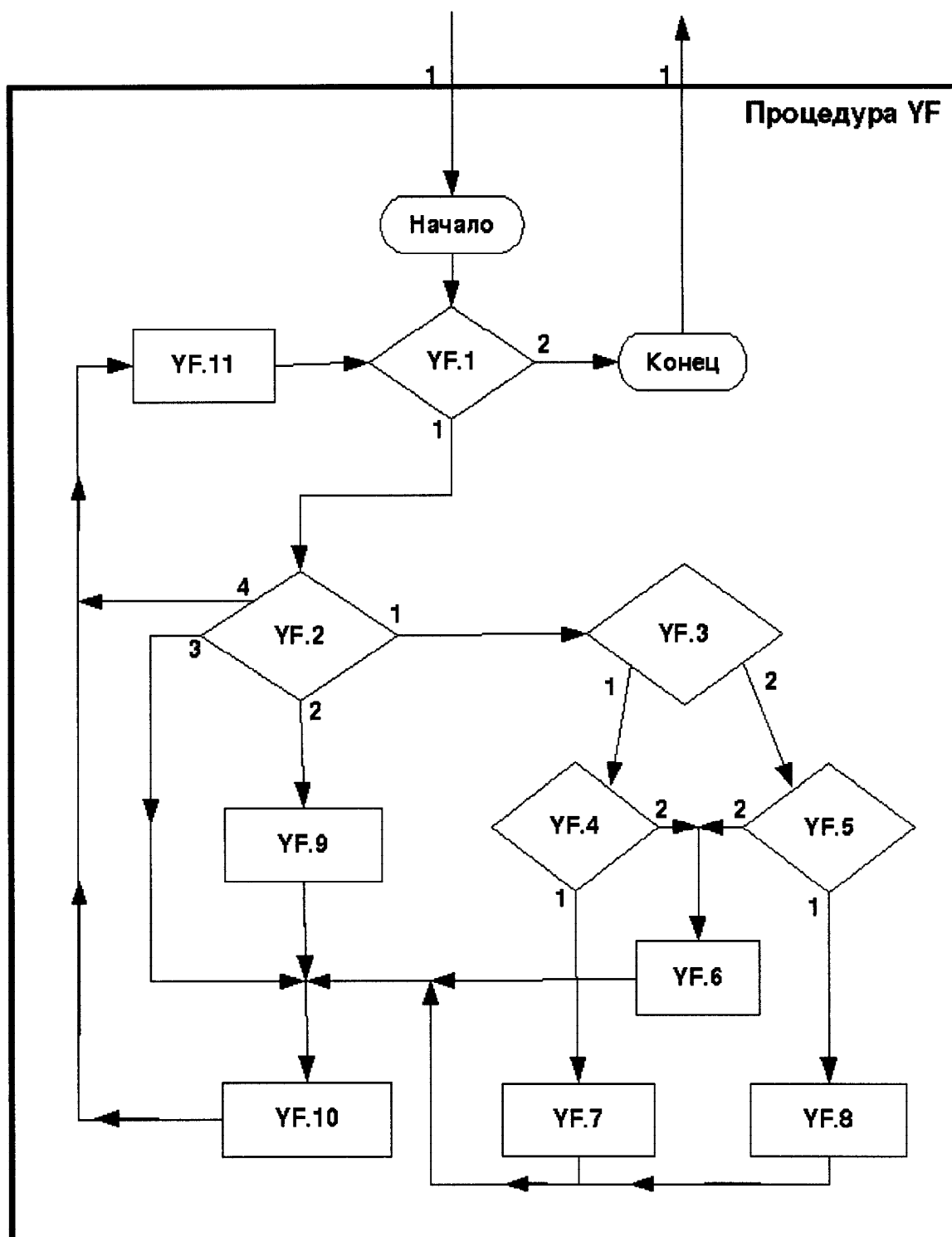
Фиг. 17



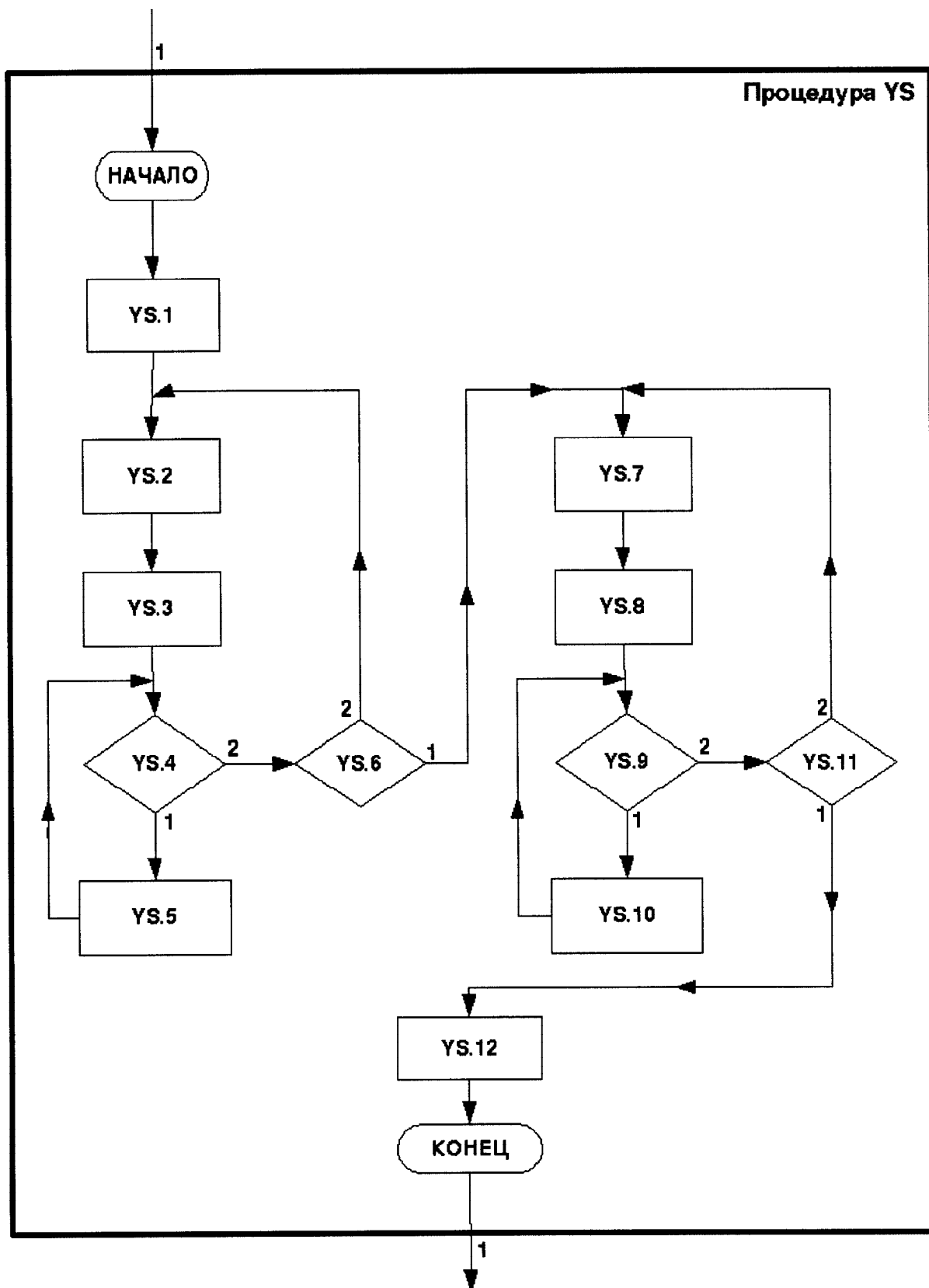
Фиг. 18



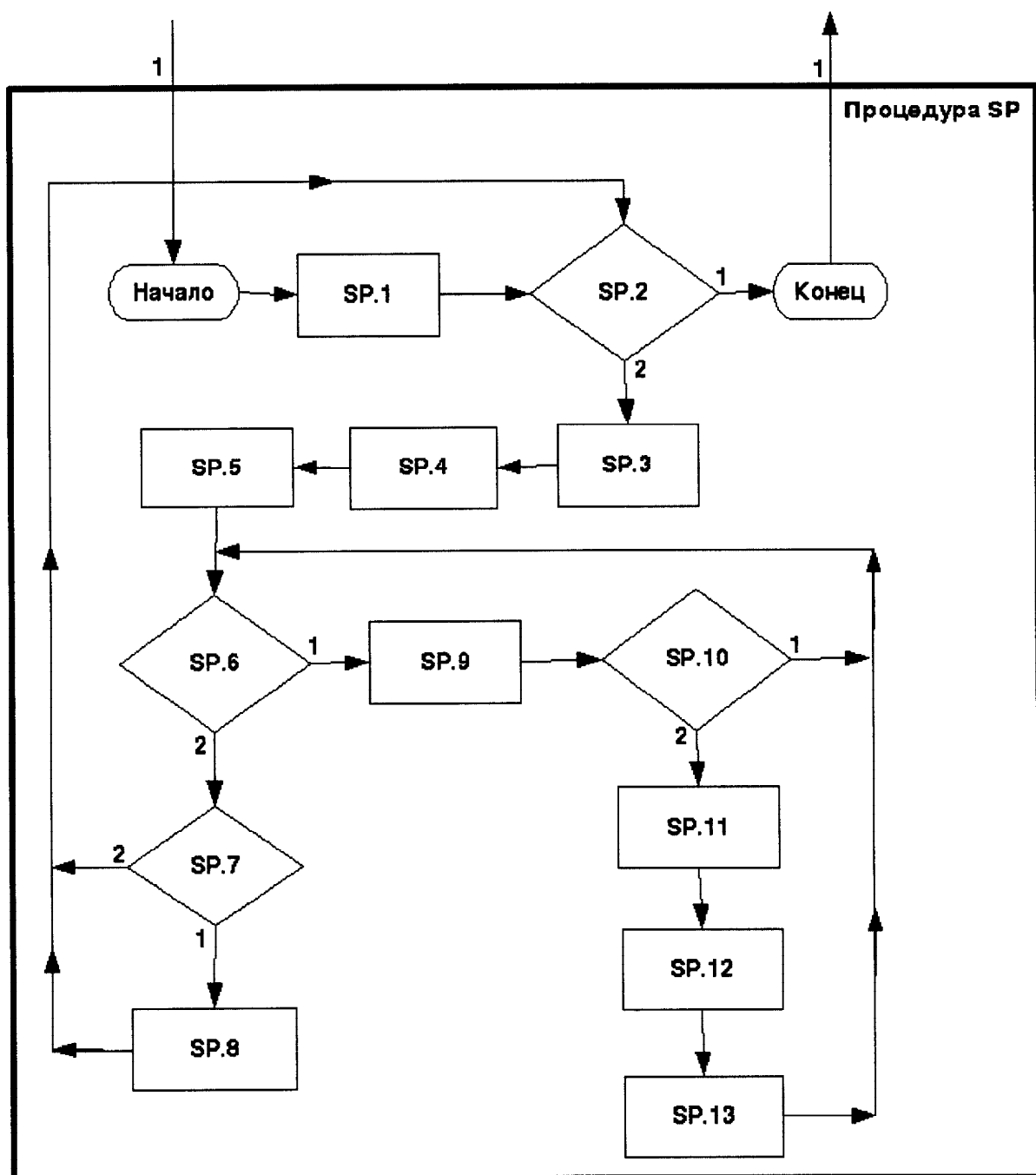
Фиг. 19



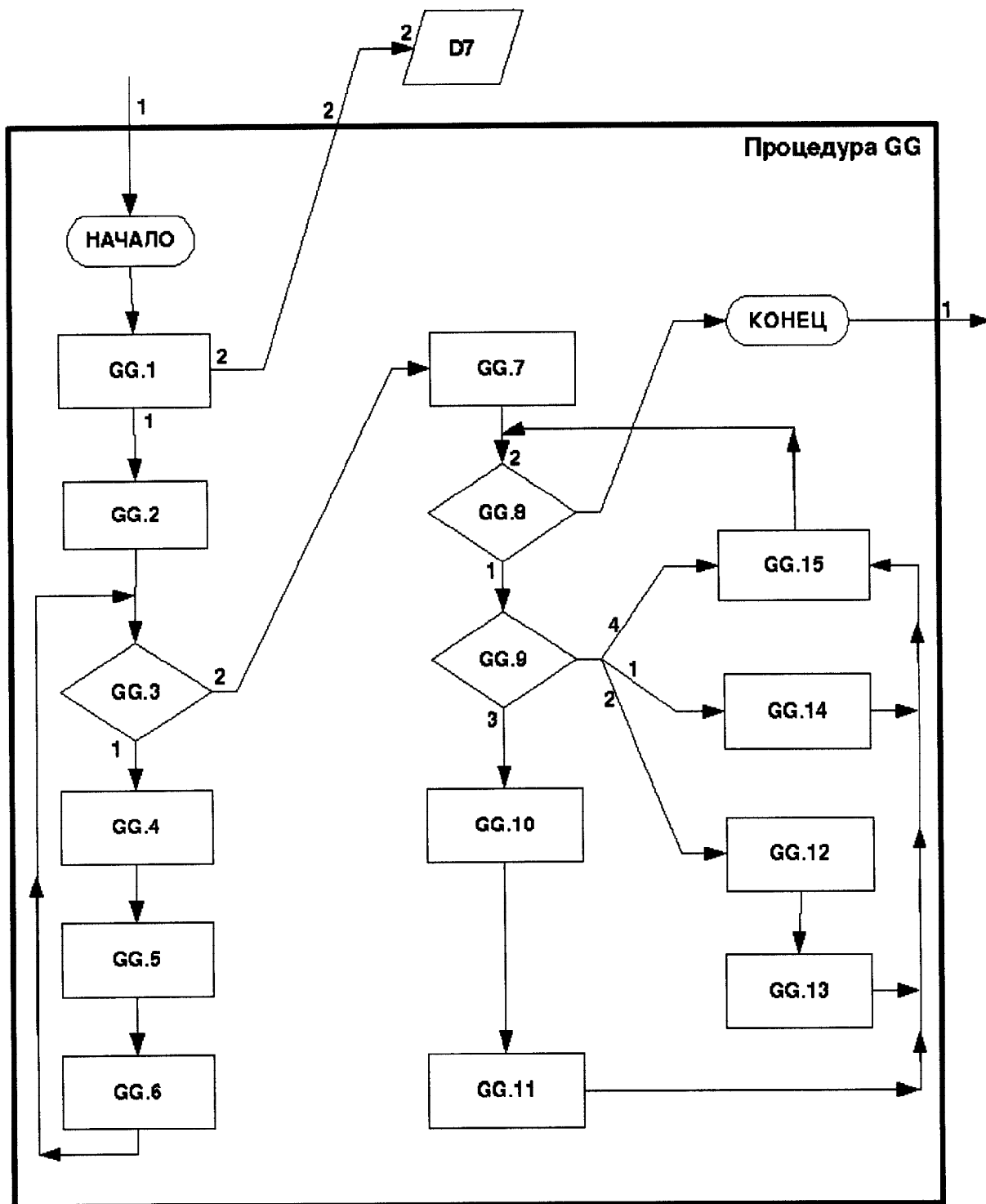
Фиг. 20



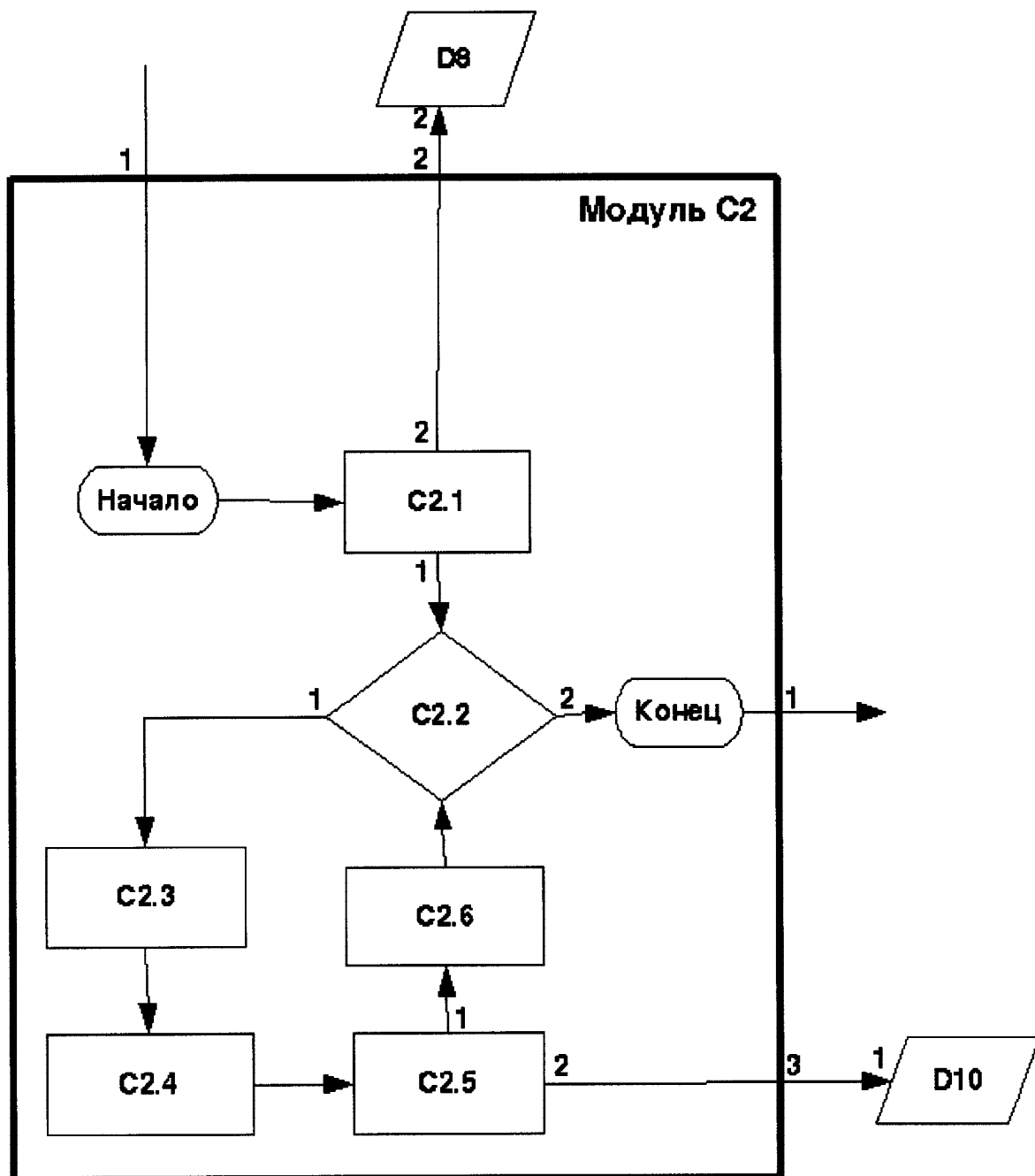
Фиг. 21



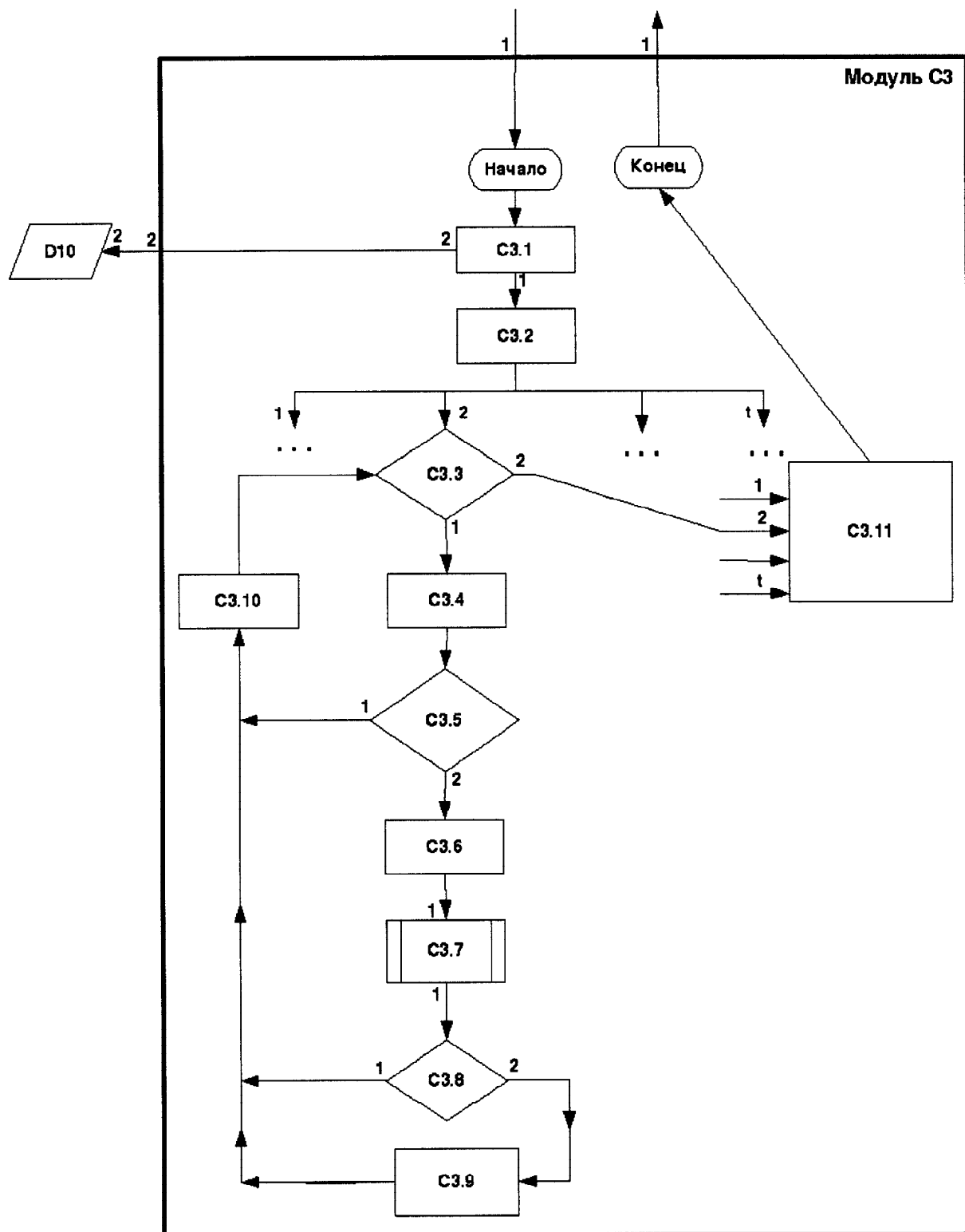
Фиг. 22



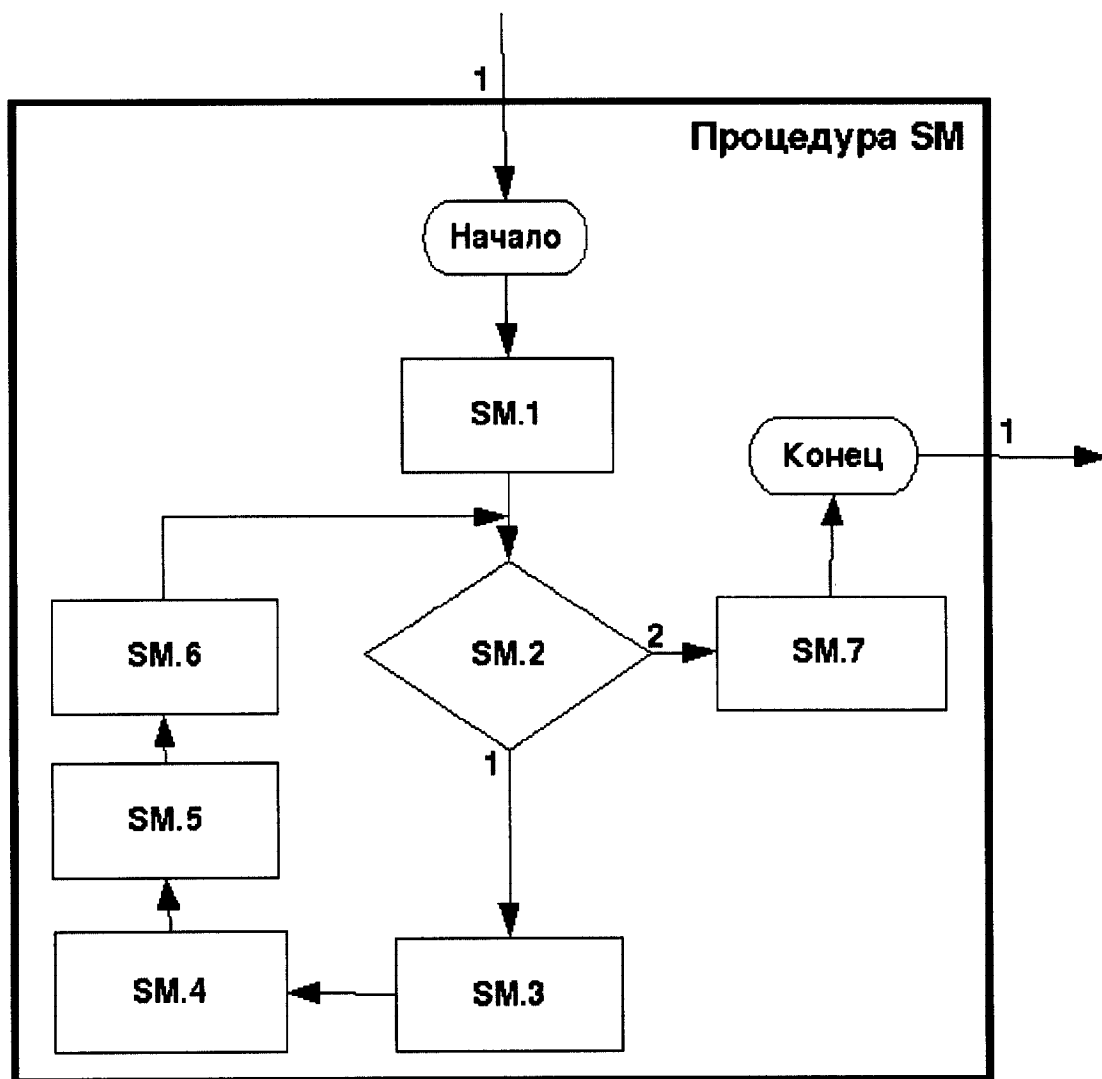
Фиг. 23



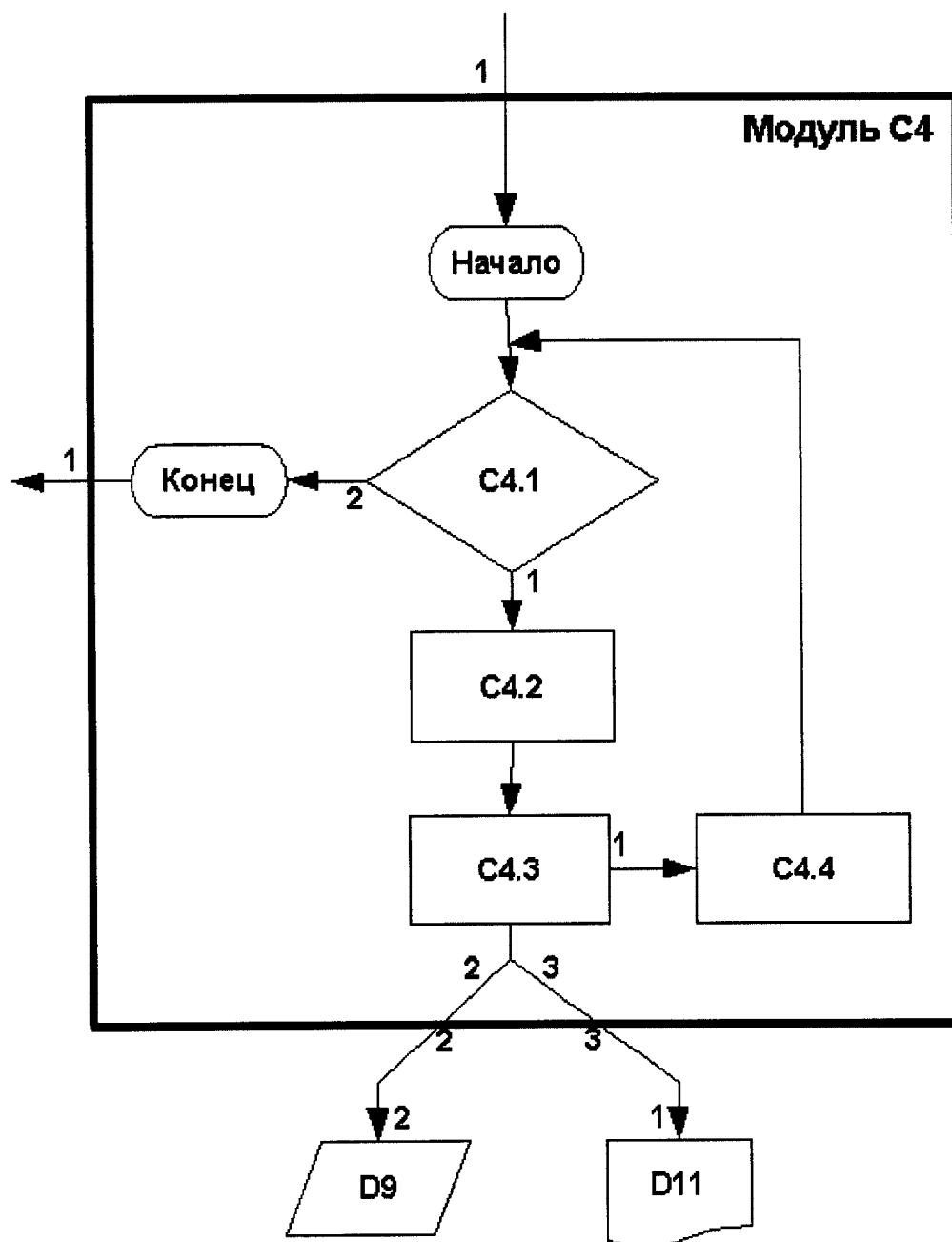
Фиг. 24



Фиг. 25

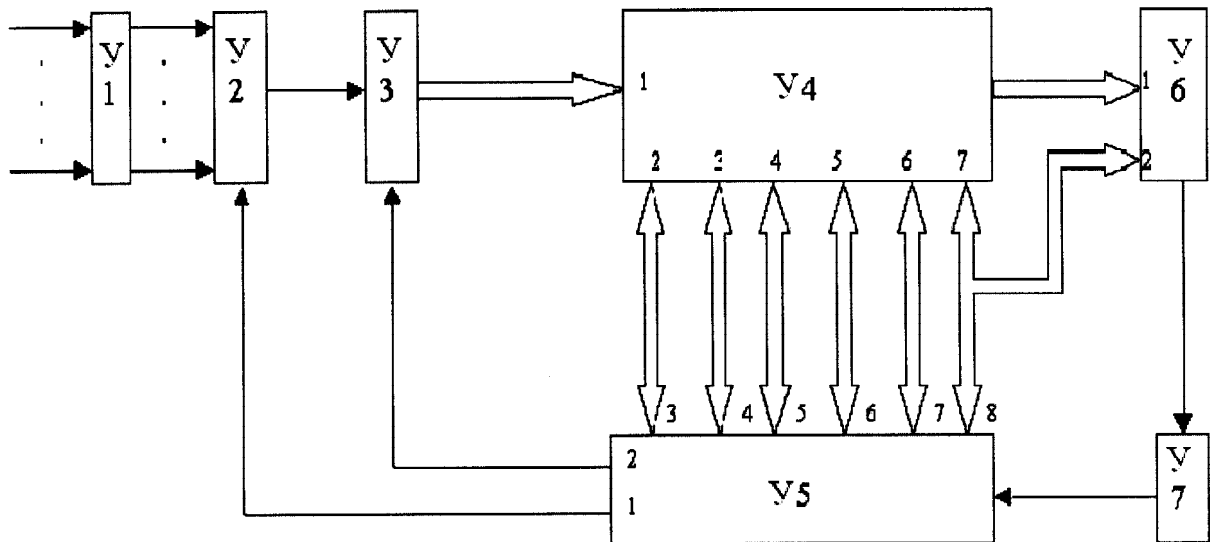


Фиг. 26

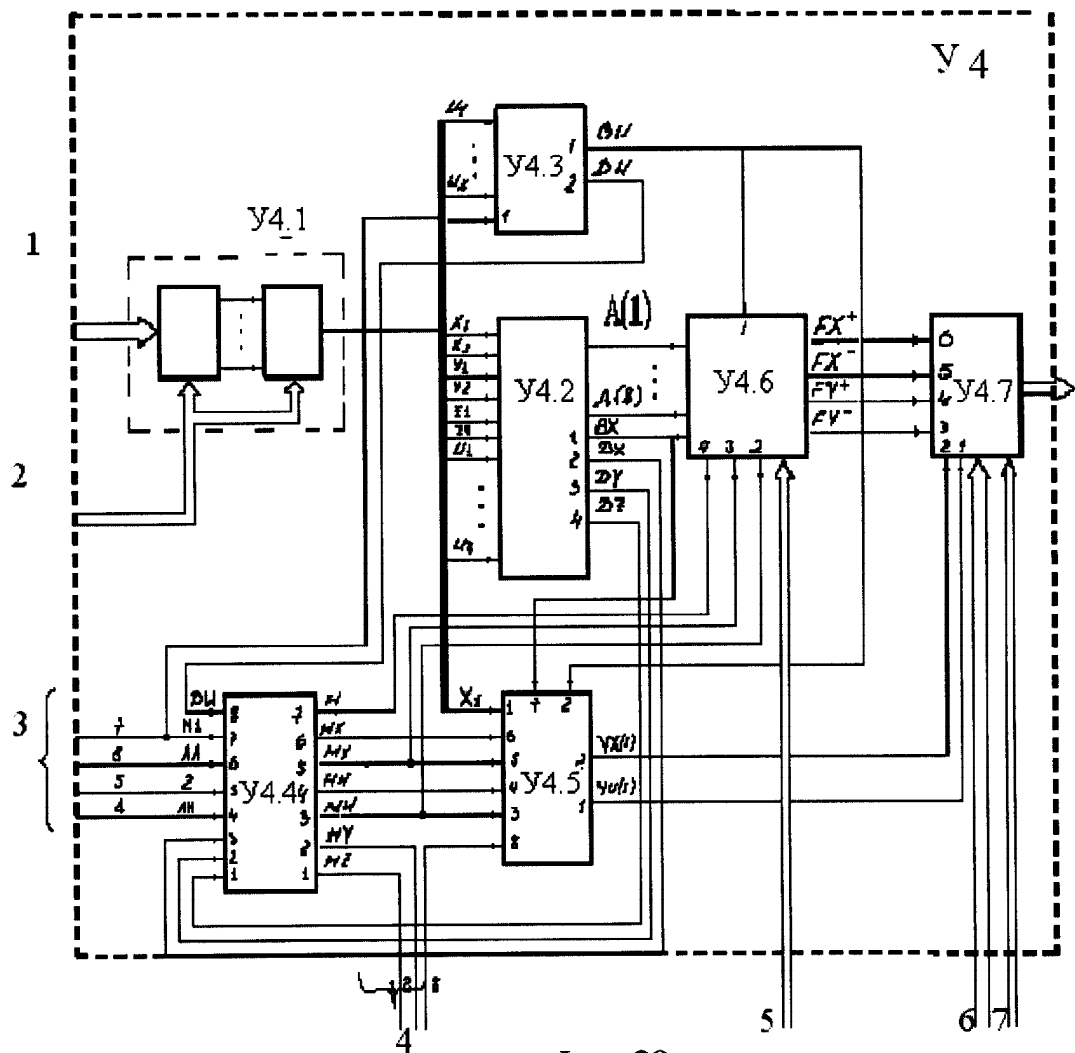


Фиг. 27

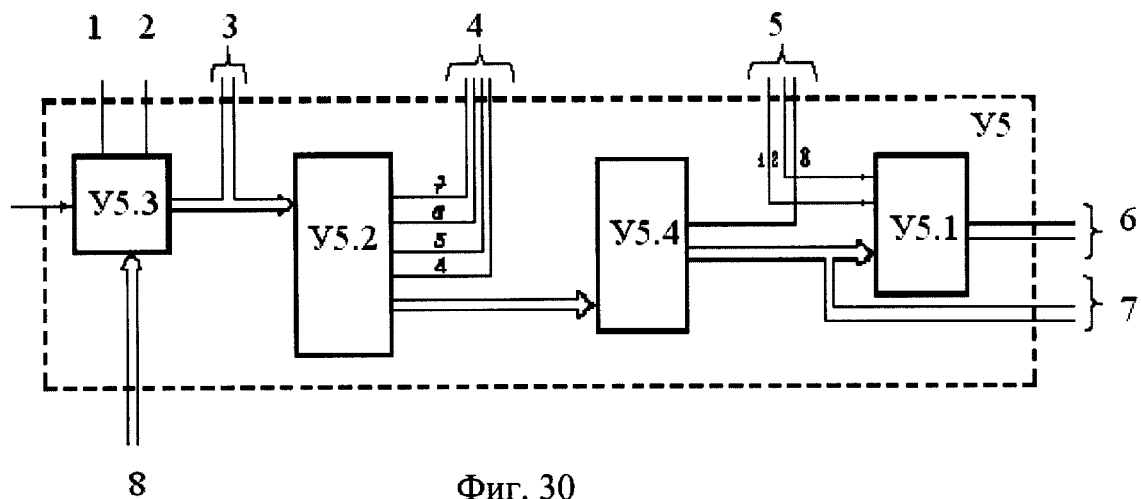
АПБ-У



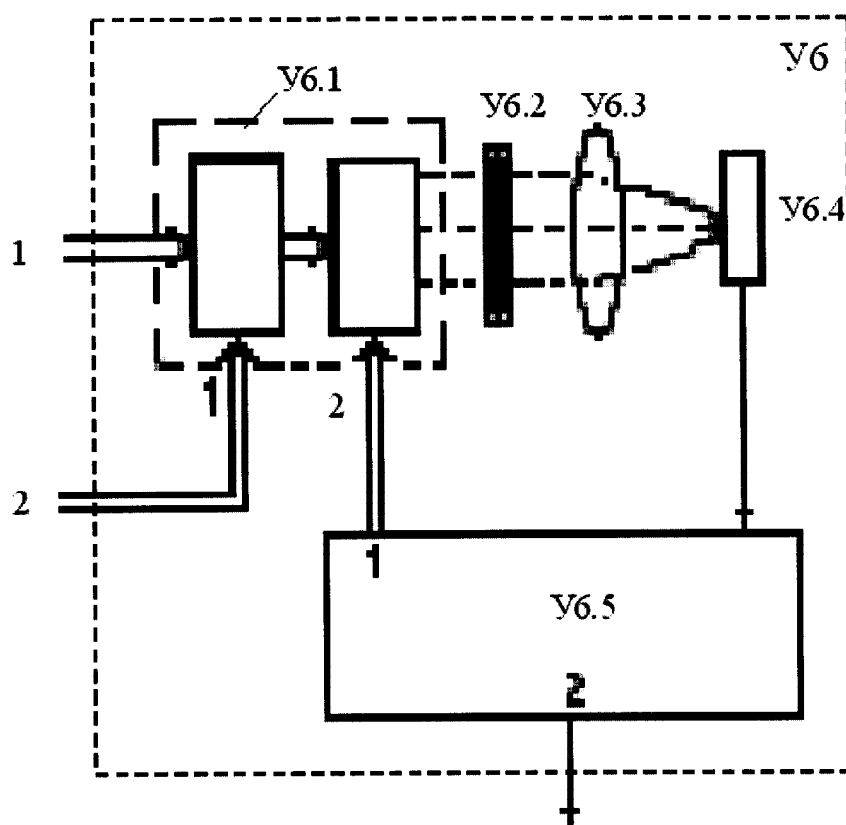
Фиг. 28



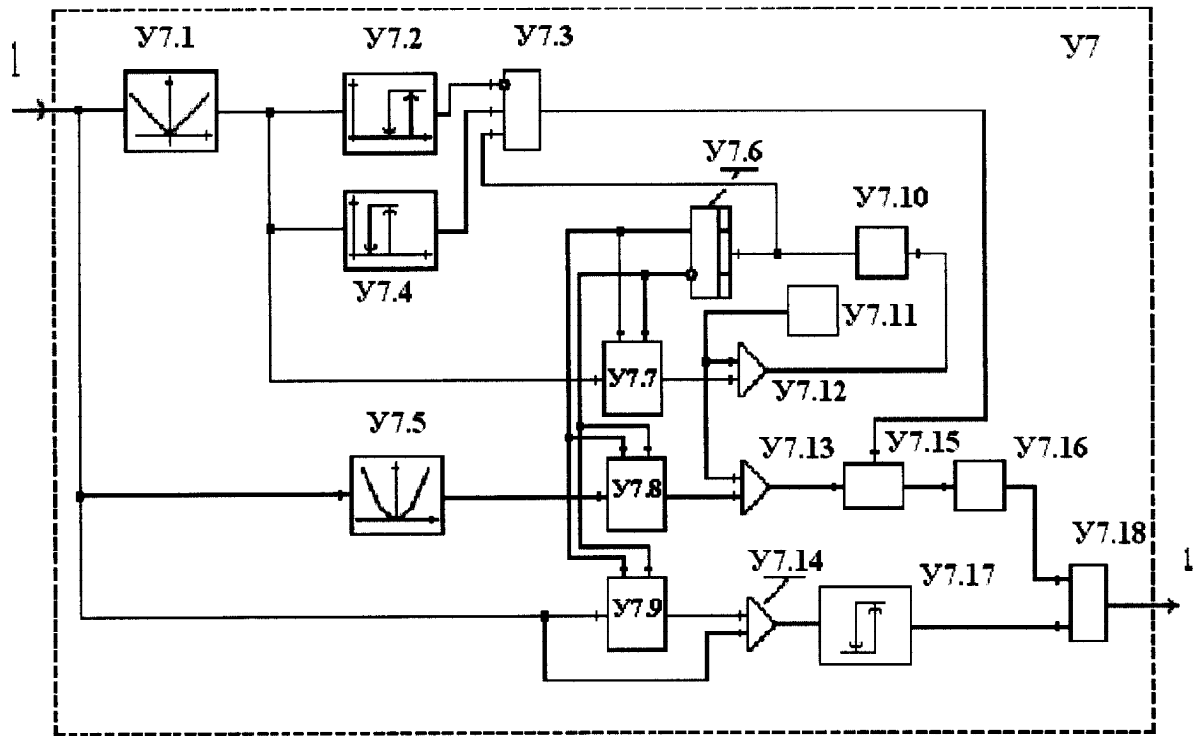
Фиг. 29



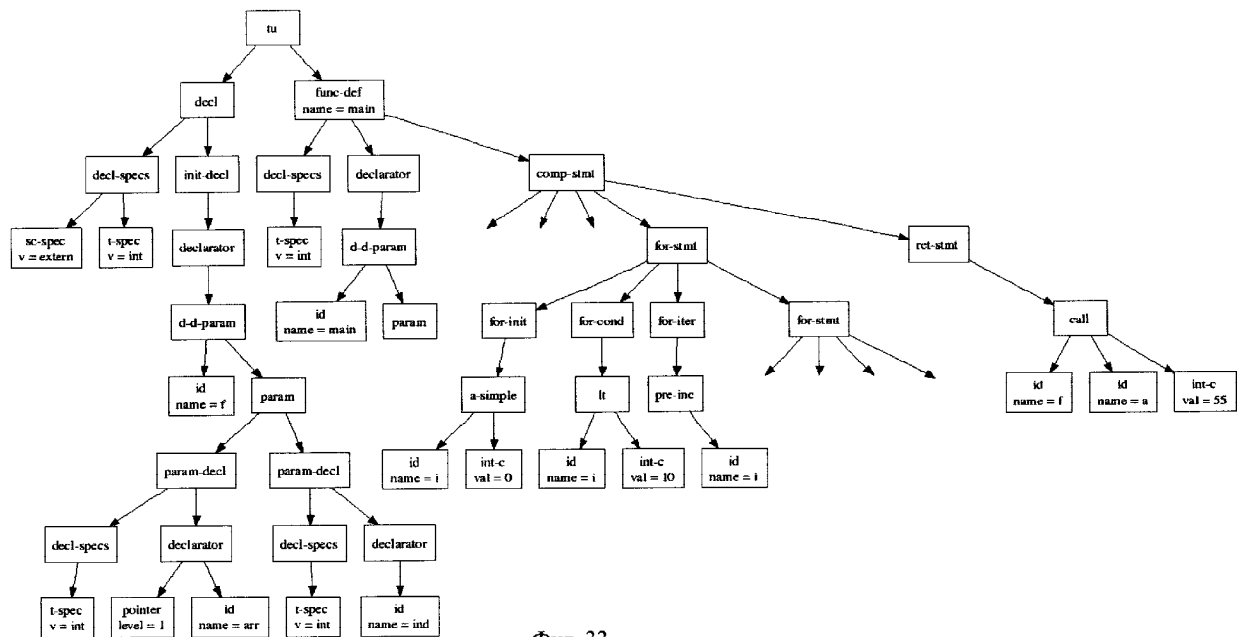
Фиг. 30



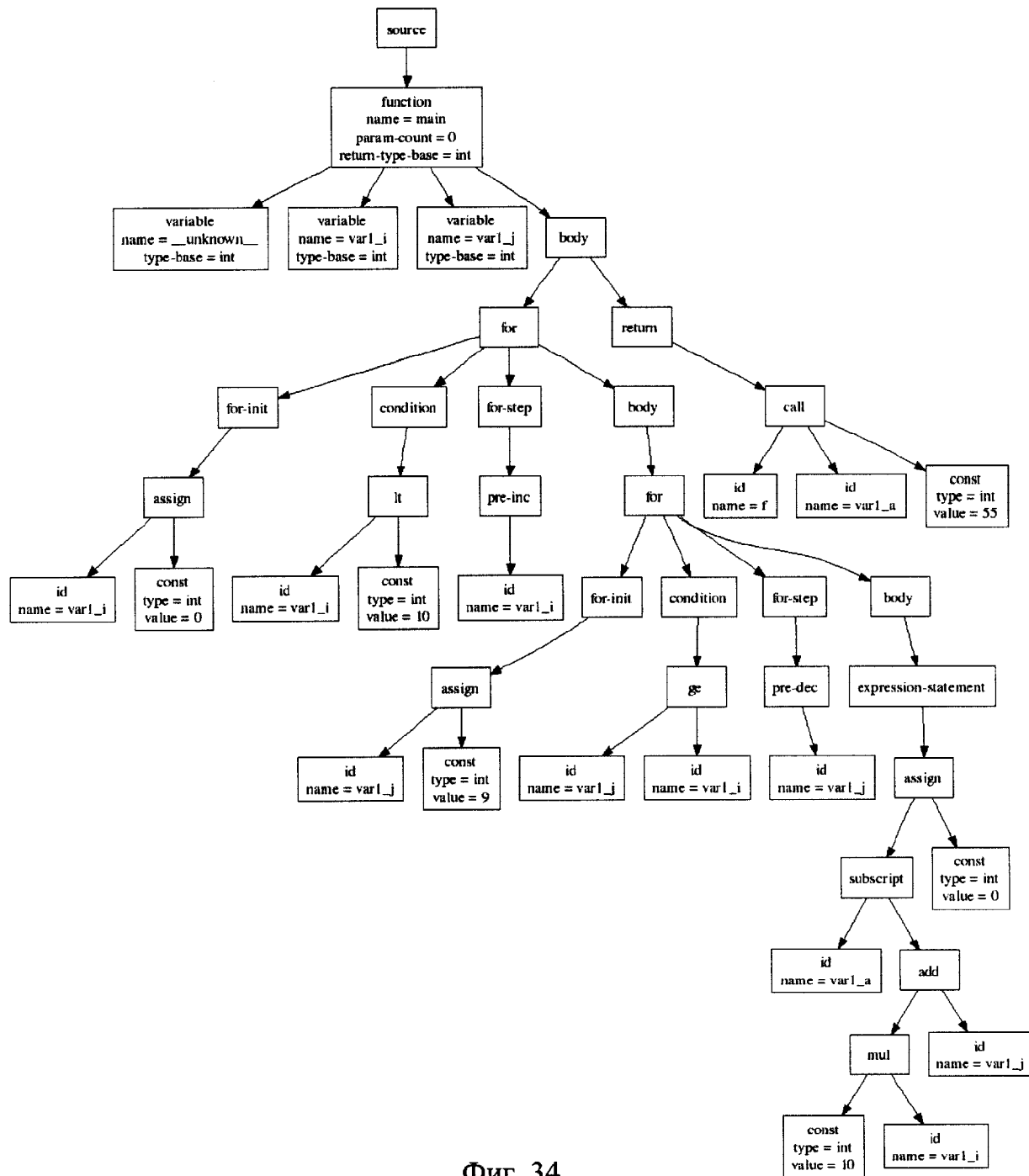
Фиг. 31



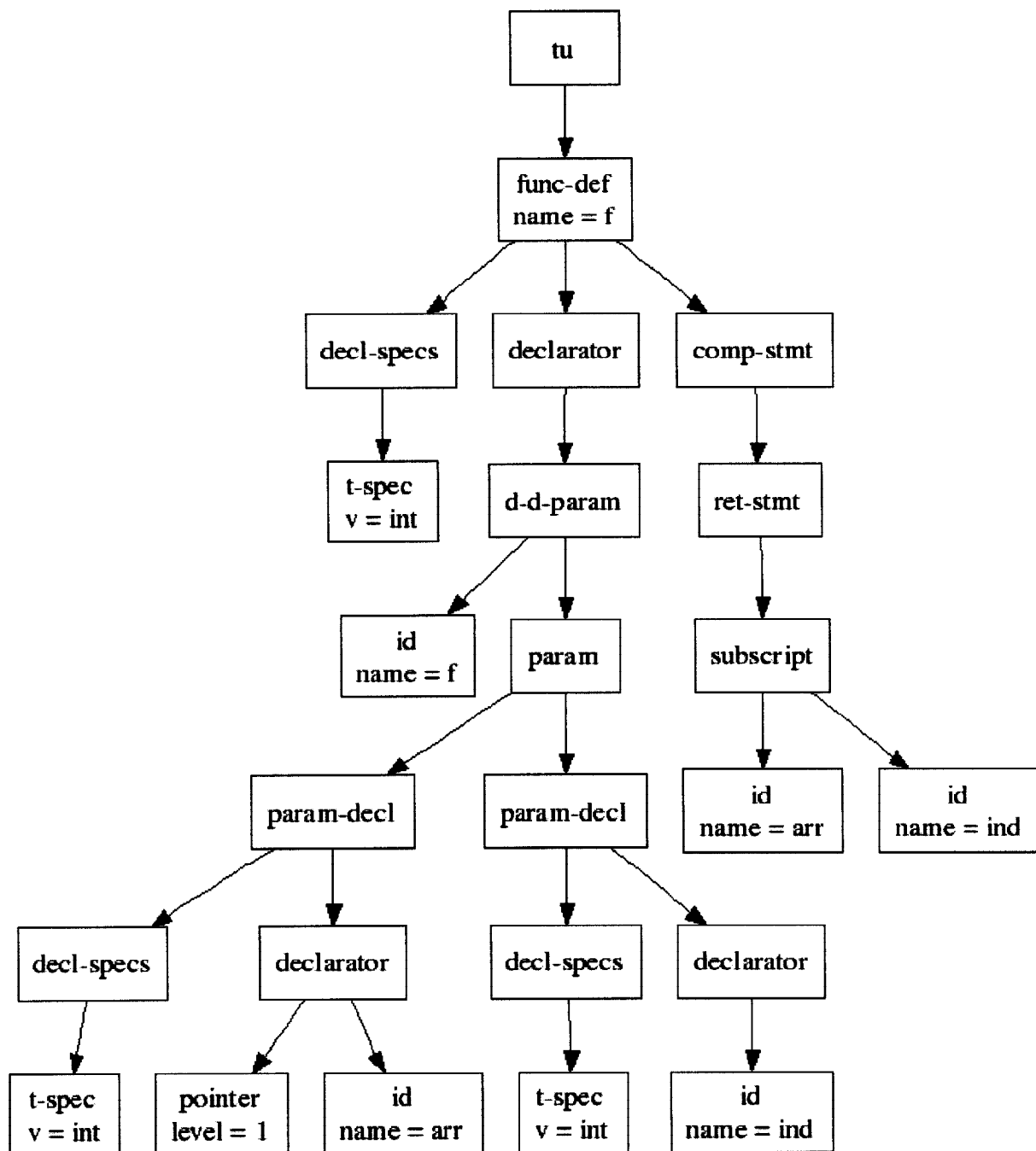
Фиг. 32



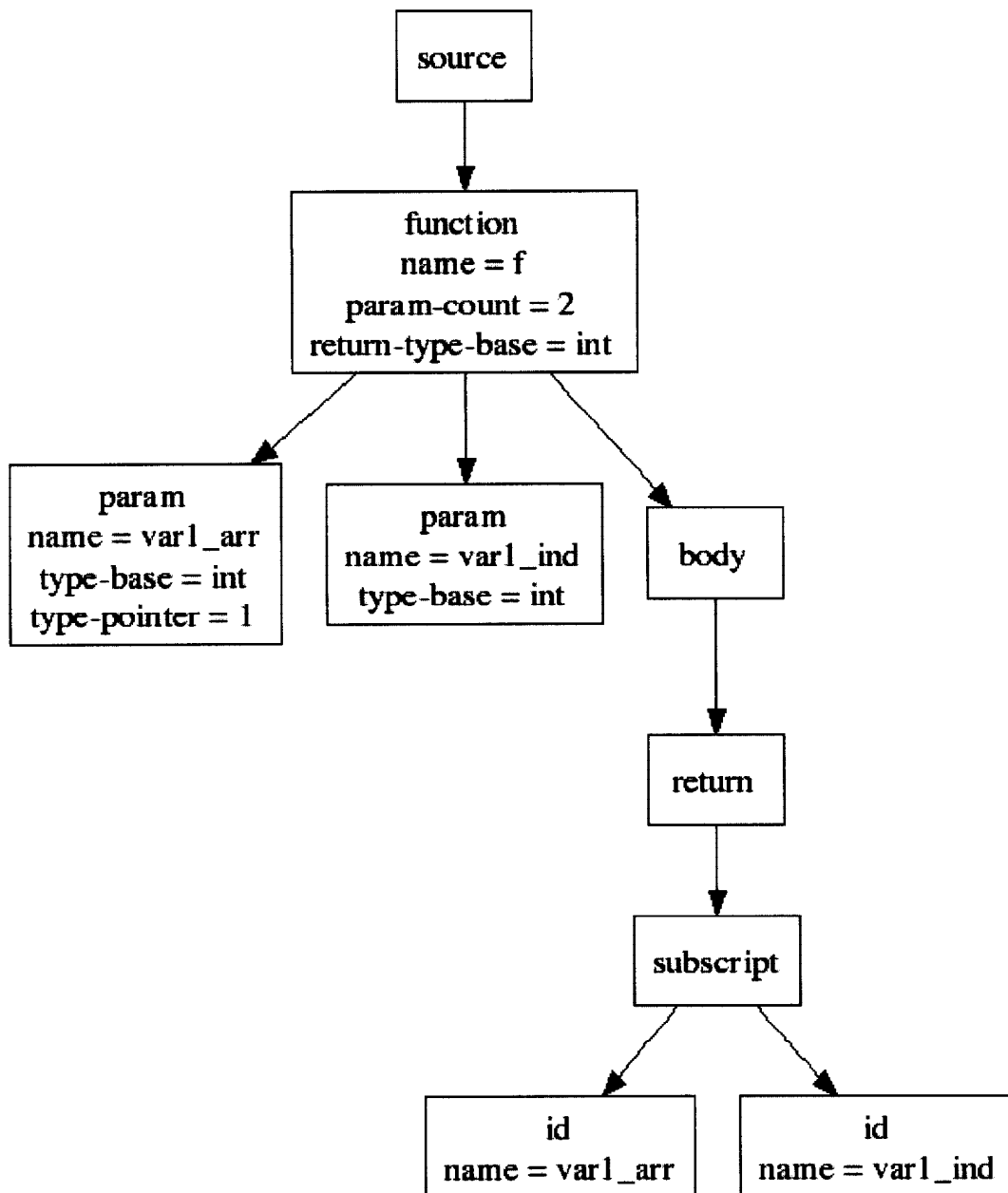
Фиг. 33



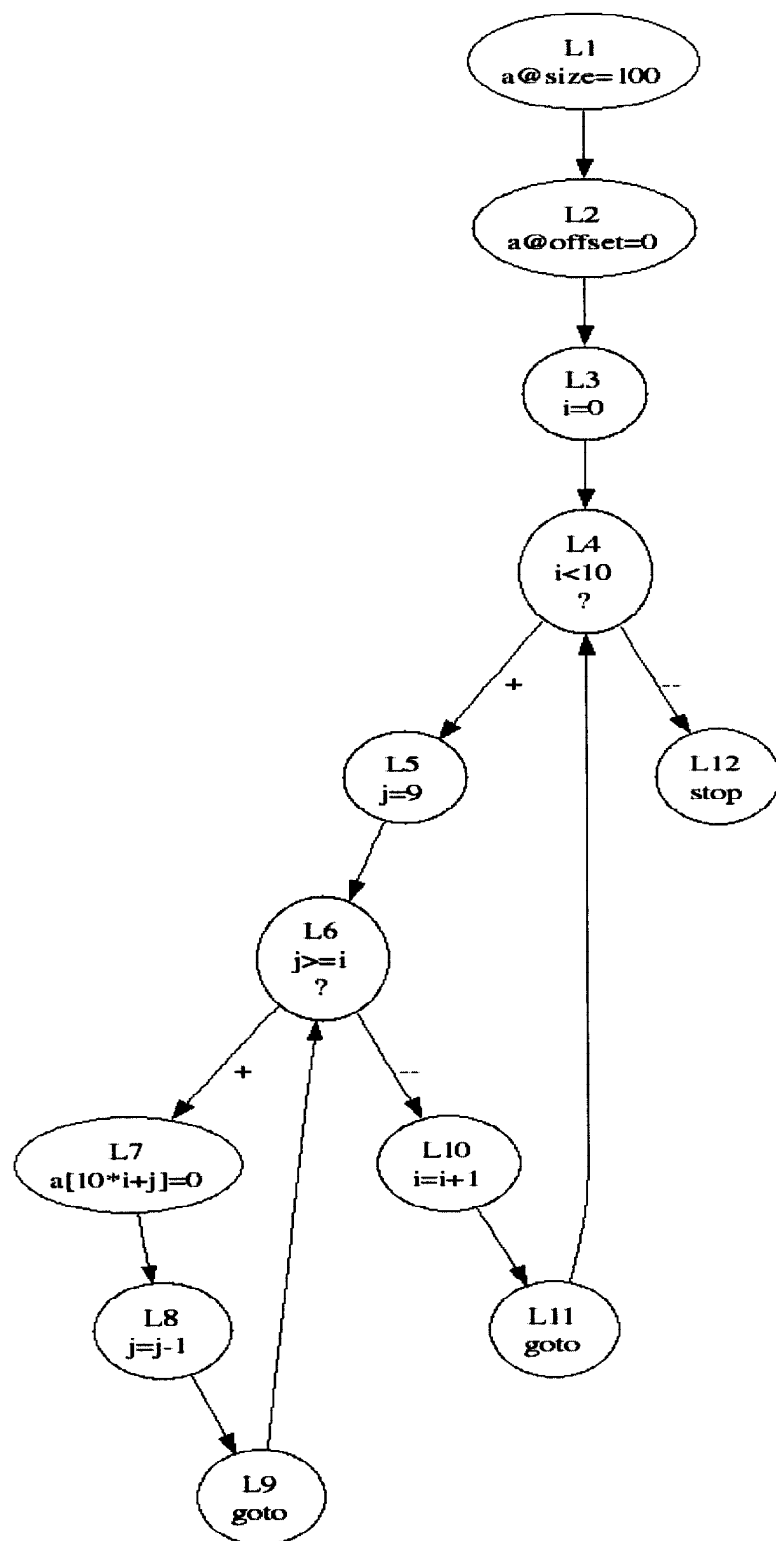
Фиг. 34



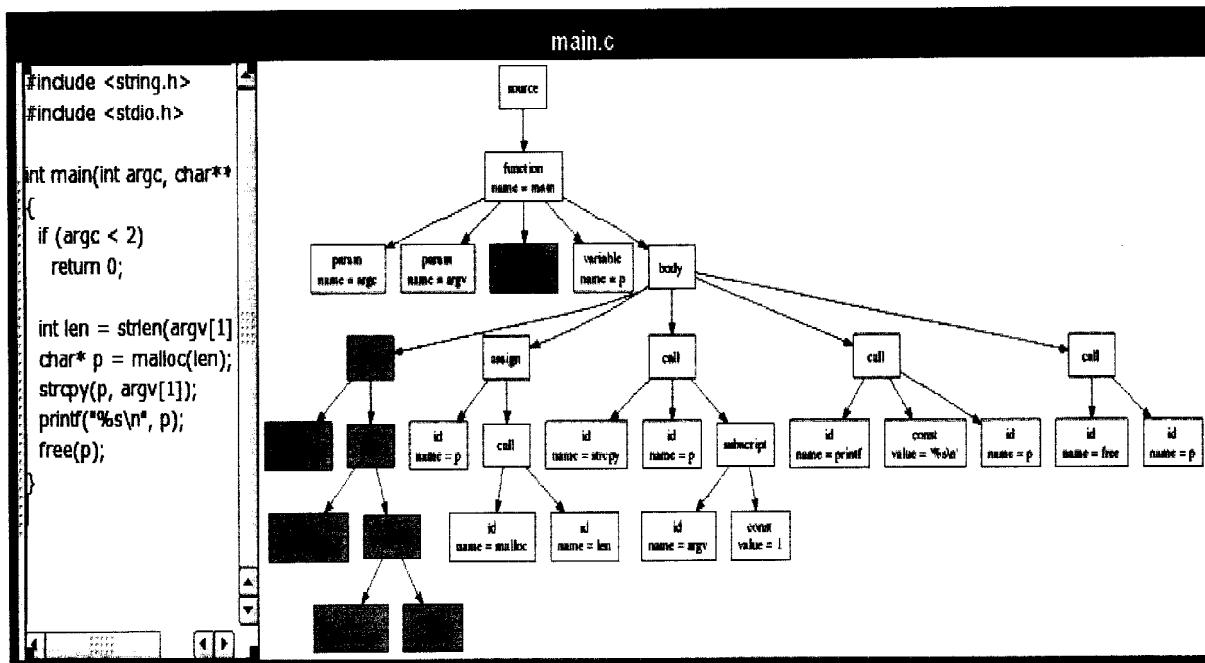
Фиг. 35



Фиг. 36



Фиг. 38



Фиг. 39