ЛАТВИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им.П.СТУЧКИ
ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР

На правах рукописи

БИЧЕВСКИЙ ЯНИС ЯЗЕПОВИЧ

АВТОМАТИЧЕСКОЕ ПОСТРОЕНИЕ ПОЛНЫХ СИСТЕМ ПРИМЕРОВ

Специальность 01.01.10 - математическое обеспечение вычислительных машин и систем

Диссертация на соискание ученой степени кандидата физико-математических наук

Научный руководитель
доктор физико-математических наук
БАРЗДИНЬ Я.М.

РИГА - 1977

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Диссертация посвящена исследованию одного подхода к автоматизации отладки программ, выдвинутого совместно Барздинем Я.М., Калниньшем А.А. и автором диссертации в /I9/. Суть этого подхода заключается в следующем: в настоящее время отладка программ обычно состоит в том, что программист строит определенную "достаточно богатую" систему примеров, на которой проверяется отлаживаемая программа. Если программа на этой системе работает правильно, то считается, что она составлена верно. Для автоматизации описанного процесса отладки предлагается систему примеров составлять не программисту, а автоматически по тексту заданной программы и в качестве "достаточно богатой" считать систему примеров, на которой программа проходит все свои ветви, которые в принципе можно пройти. Такую систему в дальнейшем будем называть полной системой примеров (ПСП).

В общем случае из того, что программа работает правильно на некоторой ПСП еще не следует, что она работает правильно на остальных примерах. Однако на практике этим эвристическим принципом пользуются весьма успешно и поэтому в данной работе он принят за основу. Цель диссертации - разработать и экспериментально проверить алгоритм построения ПСП для реальных программ обработки данных.

В диссертации получены следующие основные результаты:

- формализованы основные средства реальных языков программирования обработки данных, которые существенны с точки зрения построения ПСП;

- исследованы границы разрешимости проблемы построения ПСП в зависимости от видов команд, используемых в программе;
- разработан и практически реализован на ЭВМ "Минск-32" алгоритм построения ПСП для реальных программ обработки данных, записанных на языке СМОД /22/.

Экспериментальное построение ПСП для реальных программ подтверждает, что для весьма широкого класса программ обработки данных в приемлемое время на серийных отечественных ЭВМ можно построить ПСП. Разработанный алгоритм с некоторыми техническими изменениями применим также для построения ПСП для КОБОЛ-программ.

Первые попытки автоматического генерирования примеров для отладки программ были осуществлены уже в 1962 году Sauder R.L. /18/, а затем развиты Hicks H.T. /12/, Hanford K.V. /10/ и др. Однако в этих случаях примеры только внешне (по своему формату) напоминали те данные, для обработки которых была составлена программа. Конкретные значения примеров обычно генерировались случайно и поэтому не было гарантии в том, что сгенерированные примеры проходят все ветви программы. Этот подход к автоматизации отладки программ не нашел более широкого применения /11/.

Системы автоматического построения примеров, в которых анализируется текст программы с целью реализации того или иного пути программы появились только в последние два года в работах Miller E.F. /16/, Howden W.E. /13,14,15/, Clarke L.A. /8/, Gabow H. /9/, Miller W. /17/ и др. Эти работы по своей тематике и методам решения проблемы наиболее близки к данной диссертации. Однако авторы этих экспериментальных систем пока еще ограничиваются анализом путей, заданных человеком. Система в лучшем

случае строит примеры, которые проходят заданные пути и вопрос об автоматическом выборе анализируемых путей не ставится. Кроме того, все эти системы ориентированы на программы вычислительного характера, записанные как правило на ФОРТРАНе.

В некоторых упомянутых выше работах уже указывается на необходимость построения и проверки программ на ПСП (см./II,I6/ и др.). Однако сведений о теоретическом или практическом решении поставленной задачи другими авторами пока еще не имеется. С другой стороны появление в течение I976 года ряда работ /8, 9,I5,I7/ свидетельствует о быстром развитии исследований в этом направлении.

Перейдем к более подробному изложению результатов диссертации. Диссертацию условно можно разделить на две части. Первая часть (главы I,2,3) носит более теоретический характер и в ней исследуется возможность автоматического построения ПСП в зависимости от видов команд, используемых в программе. Во второй части диссертации (глава 4) теоретические исследования доводятся до практических результатов.

В главе I излагаются основные идеи построения ПСП. Для этого вводится так называемый базисный язык и для него рассматривается алгоритм построения ПСП.

В § I.I определяется базисный язык, который оперирует с переменными и с файлами. Различаются входные и выходные файлы, которые представляют внешнюю информацию, обрабатываемую или выдаваемую программой. Значениями файлов служат конечные последовательности целых чисел. Элементы последовательностей называются записями. Доступ к записям - последовательный. Переменные представляют внутреннюю память программы; их значениями служат

любые целые числа. В базисном языке имеется следующие команды: команда последовательного считывания записей с входных файлов в переменные (причем каждая запись может быть считана ровно один раз), команда перемещения значений переменных в записи выходных файлов, команда пересылки значений из одной переменной в другую и команда сравнения значений переменных. Программа - это граф, вершинами которого служат указанные выше команды, а другими - возможные передачи управления. Пример - это набор значений входных файлов, используемых в программе. Пример называется СТОП - примером, если программа на этом примере останавливается. Путь программы называется реализуемым, если существует пример, проходящий этот путь. Система примеров называется полной для некоторой программы, если она состоит из СТОП-примеров и программа на этой системе примеров проходит все свои дуги, которые в принципе можно пройти на каком-нибудь СТОП-примере.

В первой главе основной является следующая теорема:

Существует алгоритм, строящий конечную ПСП для любой программы, записанной в базисном языке.

Доказательство этой теоремы приводится в следующих трех параграфах.

В § I.2 вводится система неравенств $N(\alpha)$ описывающая условия прохождения пути α. Согласно лемме I.I исследование реализуемости некоторого пути α сводится к решению системы неравенств $N(\alpha)$ в целых числах.

В § I.3 вводится центральное для построения ПСП понятие - состояние. Содержательно состояние после пути α - это система неравенств, содержащая информацию о возможных значениях переменных после прохождения пути α. Излагается метод построения состояний, при котором число состояний для любой программы в базисном языке конечно.

В § I.4 излагается алгоритм построения ПСП для базисного языка. Его работа заключается в построении набора путей программы, на которых достигаются всевозможные состояния программы. Доказывается, что этот набор содержит все реализуемые дуги программы.

§ I.5 посвящается разрешимости проблемы останова программ, записанных в базисном языке. Приводится алгоритм строящий для любой программы, записанной в базисном языке, циклически полную систему примеров - т.е. систему примеров, на которой программа зацикливается "всевозможными способами".

В главе 2 продолжается исследование границ разрешимости проблемы автоматического построения ПСП в зависимости от видов команд, используемых в программе. Часть этих результатов получены в соавторстве с Барздинем Я.М. и Калниньшем А.А. в /24/.

В § 2.I базисный язык расширяется специальными переменными счетчиками, в которые можно засылать константу, прибавить положительную константу и сравнивать с константой. Оказывается, что проблема построения ПСП для вновь полученного языка также является разрешимой. Однако если разрешить к счетчикам не только прибавить, но и вычитать положительную константу, то в случае одного счетчика в программе разрешимость проблемы построения ПСП сохраняется, а в случае двух счетчиков - теряется. По-

строение ПСП неразрешимо и в том случае, если разрешить сравнение значения хотя бы одного одностороннего счетчика с другими переменными.

В § 2.2 базисный язык расширяется командой возврата входного файла на начало. В случае применения этой команды к одному входному файлу, проблема построения ПСП разрешима, а в случае применения уже к двум входным файлам (даже один раз) проблема построения ПСП неразрешима.

В § 2.3 вводится понятие текстового сравнения данных. Доказывается, что если в командах условного перехода переменные сравниваются только в текстовом отношении, то проблема построения ПСП разрешима.

В главе 3 рассматривается построение ПСП для промежуточного языка, который является более совершенной моделью обычных языков программирования обработки данных нежели базисный язык.

В § 3.1 определяется промежуточный язык. По сравнению с базисным языком он дополнительно содержит команды сложения и вычитания, переменные счетчики и массивы переменных.

В § 3.2 определяются понятия системы неравенств и состояния для промежуточного языка. Основные отличия в построении состояния от базисного языка заключаются в присутствии арифметических выражений; из-за этого число состояний для произвольной программы может оказаться бесконечным. Главное внимание в § 3.2 уделяется тому, чтобы по возможности больше "простых" программ имели бы конечное число состояний.

В § 3.3 приводится алгоритм построения ПСП для промежуточного языка. В тех случаях, когда число состояний для программы конечно, этот алгоритм работает аналогично алгоритму построения

ПСП в базисном языке. Однако если число состояний для программы бесконечно, то алгоритм на этой программе не останавливается.

Глава 4 посвящается практической реализации алгоритма построения ПСП, разработанного в главе 3. В этой практической реализации кроме автора диссертации участвовали также Борзов Ю.В., Василевский М.П., Страукмс У.М. и Зариньш А.К.

В § 4.I рассматривается первая экспериментальная система автоматического построения ПСП - система СМОТЛ. Входным языком этой системы является язык Системы Макрокоманд Обработки Данных (СМОД) /22/, по своим возможностям напоминающий КОБОЛ /3/. Основная трудность реализации алгоритма заключается в уменьшении количества запоминаемых состояний и в ускорении работы системы. Кроме этого необходимо считаться с ситуацией, когда работа алгоритма может быть прекращена из-за ограничения времени или памяти выделяемой для работы системы. В этих случаях система все-таки должна выдавать практически полезные результаты.

В § 4.2 анализируются результаты работы системы СМОТЛ на реальных программах двух действующих АСУ. Оказывается, что система успешно обрабатывает программы, количество операторов в которых не превышает 300; для программ, количество операторов в которых превышает 300, технические ограничения работы системы оказываются слишком жесткими.

Системой СМОТЛ доказано, что на ЭВМ "Минск-32" в приемлемое время для реальных программ обработки данных можно построить ПСП.

В приложении приведен пример работы системы СМОТЛ.

Основные результаты диссертации опубликованы в /I9/, /20/, /24/, /25/, а также докладывались на III Всесоюзной конференции

по проблемам теоретической кибернетики (г.Новосибирск, 1974 г.), на Всесоюзном симпозиуме "Языки системного программирования и методы их реализации" (г.Алушта, сентябрь 1974 г.) и на семинарах по математическому обеспечению при университетах городов Рига, Прага и Брно.

Автор выражает свою благодарность научному руководителю Барздиню Я.М. за постановку проблемы и ценные советы, а также Калниньшу А.А., Зариньшу А.К. и Борзову Ю.В. за ценные обсуждения результатов.

Глава I
АВТОМАТИЧЕСКОЕ ПОСТРОЕНИЕ ПСП ДЛЯ ПРОГРАММ В БАЗИСНОМ ЯЗЫКЕ

I.I. Базисный язык программирования и основные понятия.

В данной главе будут изложены основные идеи автоматического построения ПСП. Для этого мы выберем максимально простой язык программирования, так называемый базисный язык, содержащий с точки зрения построения ПСП наиболее существенные команды, и для этого языка рассмотрим алгоритм построения ПСП. С другой стороны, как будет показано во второй главе, базисный язык окажется в некотором смысле максимальным, для которого проблема автоматического построения ПСП является разрешимой в общем случае. Точнее, большинство естественных расширений базисного языка новыми командами приводят проблему автоматического построения ПСП к неразрешимости.

В базисном языке мы будем описывать действия над объектами двух видов: над переменными программы и над файлами.

Значениями переменных программы служат целые числа, притом абсолютная величина этих чисел - неограничена. Содержательно это соответствует той ситуации в программировании задач обработки данных, когда в одну ячейку можно записать практически неограниченное число, а обрабатываемой информацией являются целые числа.

В базисном языке на переменные программы мы будем ссылаться по их именам. Поэтому зафиксируем некоторый бесконечный ал-

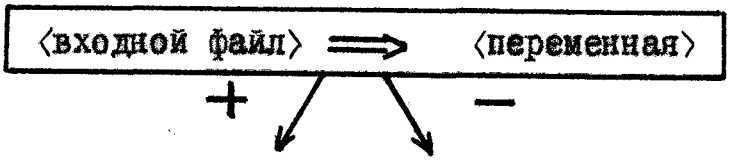
фавит $\mathcal{A}=\{a,b,c,\ldots\}$, символы которого в базисном языке будут служить именами переменных программы. В дальнейшем переменные программы будем называть просто переменными.

Внешняя информация, над которой будут описываться действия в базисном языке, будет представляться в виде файлов; значениями файлов служат конечные последовательности целых чисел, например, (5,-6, 0, I7), (3, 2, III). Элементы последовательностей, следуя традиции, будем называть записями. Доступ к записям - последовательный. Будем различать входные и выходные файлы. Из входных файлов информацию разрешается только считывать, а в выходные только записывать. На файлы, также как на переменные, будем ссылаться по их именам. Поэтому зафиксируем еще два бесконечных алфавита: $\mathcal{B}=\{A,B,C,\ldots\}$ и $\mathcal{L}=\{T,U,V,\ldots\}$, символы которых в базисном языке будут служить именами, соответственно, входных и выходных файлов.

Кроме этого в базисном языке будем использовать константы, которые будем задавать их значениями, например, 5, -I0, и т.д.

В базисном языке имеются следующие типы команд (в описаниях команд вместо "имя переменной", "имя входного файла", "имя выходного файла" будем писать просто "переменная","входной файл" и "выходной файл"):

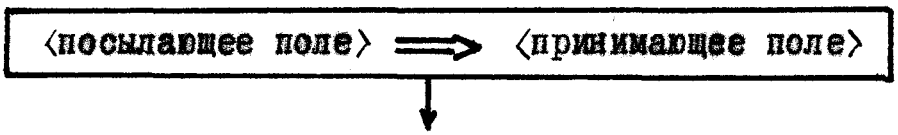
I) Команда чтения записи.



По этой команде очередная запись указанного файла переписывается в указанную переменную (при первом применении данной

команды в указанную переменную переписывается первая запись файла, при втором применении – вторая запись и т.д.). Команда имеет два выхода: если при предыдущем применении данной команды не считана последняя запись файла и файл не является пустым, то управление передается по первому выходу. Этот выход для определенности обозначим через "+". Если же при предыдущем применении данной команды была считана последняя запись файла или же файл является пустым, то управление передается по второму выходу, который обозначим через "–" (в этом случае значение указанной переменной не меняется).

2) Команда пересылки.

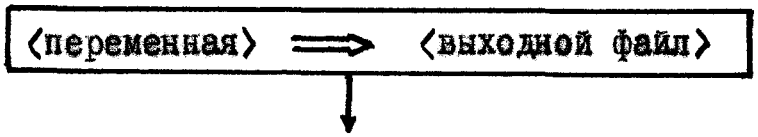


где ⟨посылающее поле⟩ ::= {⟨переменная⟩ | ⟨константа⟩},

⟨принимающее поле⟩ ::= {⟨переменная⟩}.

Команда имеет один выход.

3) Команда вывода записи.



Значение указанной переменной становится очередной записью указанного выходного файла (при первом применении данной команды образуется первая запись файла, при втором применении – вторая запись файла и т.д.). Команда имеет один выход.

4) Команда условного перехода.

где ⟨условие⟩ ::= {⟨переменная$_I$⟩ < ⟨переменная$_2$⟩ |
|⟨переменная⟩ < ⟨константа⟩ |
|⟨константа⟩ < ⟨переменная⟩}

Команда имеет два выхода: если условие выполнено, то управление передается по выходу "+", иначе по выходу "-".

5) Команда - конец работы.

Команда не имеет выходов.

Под программой в базисном языке будем понимать графсхему, составленную из описанных выше команд. Будем считать, что все команды программы пронумерованы числами 1,2,3,... Программа всегда начинает выполняться с первой команды при нулевых начальных значениях переменных и останавливается, когда попадает на команду СТОП.

На рис.1 приведена программа ПРИМЕР-1, которая одинаковые записи из двух входных файлов A и B, упорядоченных в порядке возрастания значений записей, выводит в выходной файл U.

В дальнейшем команды программы будем отождествлять с их номерами. Пару номеров команд (n_1, n_2) будем называть дугой программы, если один из выходов команды n_1 непосредственно ведет в команду n_2. Например, (1,2), (2,7), (3,4) являются дугами программы ПРИМЕР-1, а (3,7), (7,8) - не являются. В дальнейшем будем считать, что оба выхода любой команды программы, имеющей два выхода, не ведут в одну и ту же команду программы. Таким образом, задавая дугу парой (n_1, n_2) номеров команд, не возникнет недоразумений относительно того, который

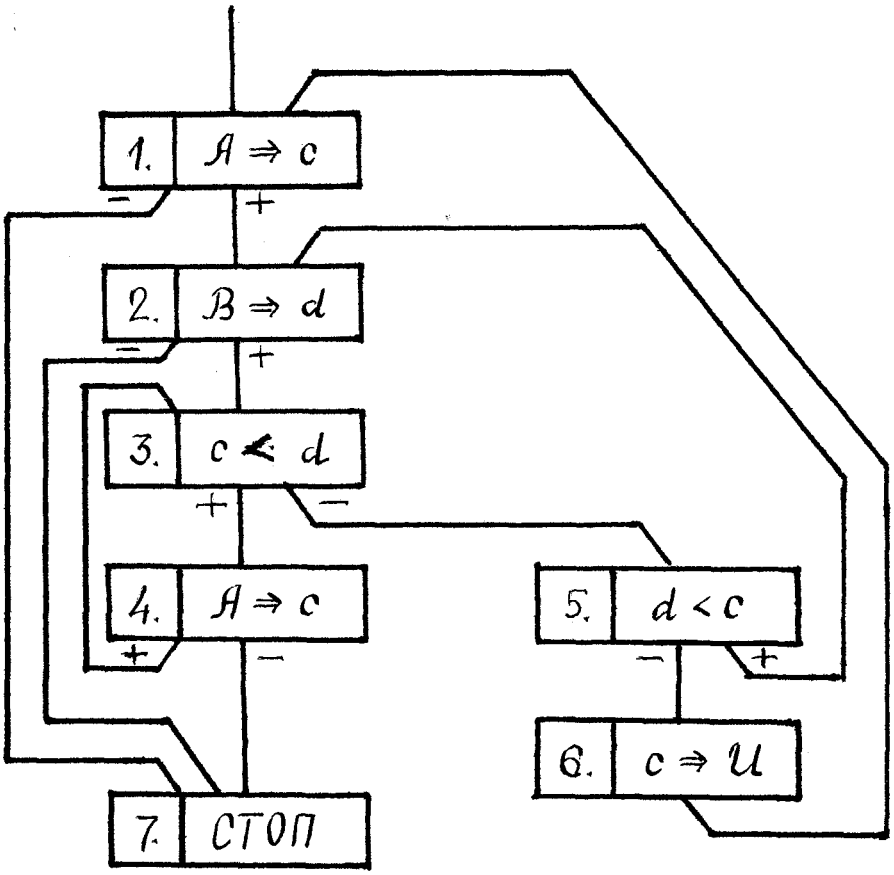


Рис.I.

из выходов "+" или "-" команды n_1 ведет в команду n_2 .

Путь программы - это любая конечная последовательность дуг вида $((n_1,n_2),(n_2,n_3),\dots(n_{k-1},n_k))$; этот путь обозначим через $\alpha=(n_1,n_2,\dots,n_k)$. Пусть путь α заканчивается командой n_k . Тогда любой путь, начинающийся командой n_k , будем называть продолжением пути α . В дальнейшем, если не оговорено противоположное, под путем будем понимать путь, начинающийся первой командой программы.

Под примером для программы P будем понимать набор значений всех входных файлов, используемых в программе P . При-

мер будем задавать, сопоставляя каждому имени входного файла, используемого в программе P, конкретную последовательность целых чисел. Будем говорить, что пример T реализует путь $\alpha = (n_1, n_2, \ldots, n_k)$, вообще говоря, начинающийся не с первой команды программы, если запуская программу на этом примере, она с некоторого момента выполнит последовательность команд $(n_1, n_2, \ldots, n_k)$. Будем говорить, что путь α реализуем, если существует пример T, который реализует α. Пример будем называть СТОП-примером, если программа на этом примере останавливается.

Систему примеров $\mathcal{M}$ будем называть полной для программы P, если выполнены условия:

1) $\mathcal{M}$ состоит из СТОП-примеров,
2) каждая дуга программы P, которая реализуема на некотором СТОП-примере, реализуема на некотором примере из $\mathcal{M}$.

ТЕОРЕМА I.I. Существует алгоритм, который для любой программы в базисном языке строит конечную полную систему примеров.

В следующих параграфах будет дано подробное доказательство теоремы I.I, которое понадобится также в других главах.

I.2. Системы неравенств и реализуемость путей.

Пусть P - произвольная программа в базисном языке и α - произвольный путь этой программы. В этом параграфе мы сопоставим пути α систему неравенств $N(\alpha)$ такую, что путь α является реализуемым тогда и только тогда, когда $N(\alpha)$ имеет решение в целых числах.

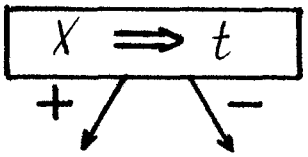
Каждое неравенство из $N(\alpha)$ будет выражать отношение между значениями переменных, сравниваемых в командах условного перехода пути α . Значения переменных после прохождения пути α выразим в виде некоторой системы равенств, обозначаемой через $E(\alpha)$. $E(\alpha)$ будет состоять из столько отдельных равенств, сколько переменных и входных файлов используются в программе. Каждое равенство будет описывать значение одной переменной или задавать некоторую информацию об одном входном файле.

$E(\alpha)$ и $N(\alpha)$ строим индуктивно. Пусть t, u, v - произвольные переменные, X - произвольный входной файл, а Y - произвольный выходной файл. Записи файла X обозначим через $x_1, x_2, \ldots, x_j, \ldots$ Для пустого пути (базис индукции), который обозначим через α_0 , система $N(\alpha_0)$ пустая, а $E(\alpha_0)$ состоит из всевозможных равенств вида $g = 0$, где g - имя переменной или входного файла, используемого в программе. Содержательно это означает, что начальные значения переменных равны нулю.

Шаг индукции. Через α_i обозначим путь, содержащий первые i дуги пути α , т.е. $\alpha_i = (n_1, n_2, \ldots, n_i, n_{i+1})$.

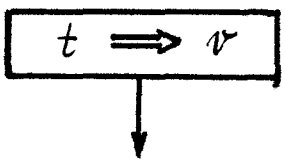
Пусть для пути α_i уже построены системы $N(\alpha_i)$ и $E(\alpha_i)$, и пусть $E(\alpha_i)$ состоит из равенств вида $g = r$, где g - имя переменной или входного файла, используемого в программе, а r - константа или обозначение записи входного файла. $E(\alpha_{i+1})$ строится из $E(\alpha_i)$ заменяя соответствующее равенство новым равенством (но общее число равенств не меняется), а $N(\alpha_{i+1})$ строится из $N(\alpha_i)$ добавлением новых неравенств. Конкретно это зависит от $(i+1)$-ой дуги (n_{i+1}, n_{i+2}), точнее, от вида команды n_{i+1} и выхода, ведущего в команду n_{i+2}.

I) Пусть командой n_{i+1} является команда чтения записи



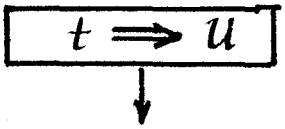
Согласно конструкции $E(\alpha_i)$ содержит равенства вида $t = r_1$ и $X = r_2$. Если команда n_{i+1} передает управление команде n_{i+2} по выходу "+", то заменяем равенство $t = r_1$ на новое равенство $t = x_j$, где x_j - обозначение очередной считываемой записи файла X (после первого применения данной команды имеем $t = x_1$, после второго $t = x_2$ и т.д.). Если же команда n_{i+1} передает управление команде n_{i+2} по выходу "-", то заменяем равенство $X = r_2$ на $X = 1$. Содержательно это означает - файл X на пути α_{i+1} прочитан до конца. В случае, когда файл X уже прочитан до конца на пути α_i, а команда n_{i+1} передает управление команде n_{i+2} по выходу "+", мы добавляем к $N(\alpha_i)$ противоречивое неравенство, например, $0 > 1$.

2) Пусть командой n_{i+1} является команда пересылки



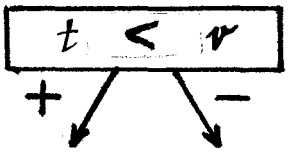
и пусть $E(\alpha_i)$ содержит равенства $t = r_1$ и $v = r_2$. Тогда в $E(\alpha_i)$ равенство $v = r_2$ заменяем на новое равенство $v = r_1$. Если t - константа c, то равенство $v = r_2$ заменяем на новое равенство $v = c$. $N(\alpha_i)$ не меняется.

3) Пусть командой n_{i+1} является команда вывода записи



Тогда $E(\alpha_i)$ и $N(\alpha_i)$ не меняются.

4) Пусть командой n_{i+1} является команда условного перехода



и пусть $E(\alpha_i)$ содержит равенства $t = r_1$ и $v = r_2$. Тогда, если команда n_{i+1} передает управление команде n_{i+2} по выходу "+" (соответственно по выходу "-"), то к $N(\alpha_i)$ добавляется неравенство $r_1 < r_2$ (соответственно $r_1 \geq r_2$). Если v - константа c, то к $N(\alpha_i)$ добавляется $r_1 < c$ (соответственно $r_1 \geq c$). $E(\alpha_i)$ не меняется.

Для программы ПРИМЕР-I имеем:

$E(\alpha_0)$ = $\{c=0,\ d=0,\ A=0,\ B=0\}$

$N(\alpha_0)$ = $\{$ ‹пусто› $\}$

E (I,2) = $\{c=a_1,\ d=0,\ A=0,\ B=0\}$

N (I,2) = $\{$ ‹пусто› $\}$

E (I,2,3) = $\{c=a_1,\ d=b_1,\ A=0,\ B=0\}$

N (I,2,3) = $\{$ ‹пусто› $\}$

E (I,2,3,4) = $\{c=a_1,\ d=b_1,\ A=0,\ B=0\}$

N (I,2,3,4) = $\{a_1 < b_1\}$

$E(1,2,3,4,3) = \{c = a_2, d = b_2, A = 0, B = 0\}$

$N(1,2,3,4,3) = \{a_1 < b_1\}$

$E(1,2,3,4,3,5) = \{c = a_2, d = b_1, A = 0, B = 0\}$

$N(1,2,3,4,3,5) = \{a_1 < b_1, a_2 \geq b_1\}$

$E(1,2,3,4,7) = \{c = a_1, d = b_1, A = 1, B = 0\}$

. . .

$E(1,2,3,5,2,3,4,3,4) = \{c = a_2, d = b_2, A = 0, B = 0\}$

$N(1,2,3,5,2,3,4,3,4) = \{a_1 \geq b_1, b_1 < a_1, a_1 < b_2, a_2 < b_2\}$

и т.д.

Таким образом левыми частями равенств из $E(\alpha)$ служат имена переменных и входных файлов, а правыми частями - константы и обозначения записей входных файлов. В неравенствах из $N(\alpha)$ правыми и левыми частями служат константы и обозначения записей входных файлов, а неизвестными системы неравенств $N(\alpha)$ являются записи входных файлов. Кроме того, $N(\alpha)$ содержит столько отдельных неравенств, сколько команд условного перехода содержится в α (за исключением случая, указанного в I)).

Из построения $N(\alpha)$ вытекает:

ЛЕММА I.I. Путь α реализуем тогда и только тогда, когда система неравенств $N(\alpha)$ имеет решение в целых числах; решение системы $N(\alpha)$ задает пример, который реализует путь α .

Например, для программы ПРИМЕР-I имеем $N(1,2,3,4,3,5,2) = \{a_1 < b_1, a_2 \geq b_1, b_1 < a_2\}$. Решение этой системы неравенств, например, $a_1 = 1$, $b_1 = 2$, $a_2 = 3$, задает пример $T = \{A = (1,3), B = (2)\}$ который реализует путь (1,2,3,4,3,5,2).

I.3. Состояние программы.

Используя лемму I,1 может показаться естественным следующий прием построения ПСП: последовательно перебираем пути программы, оканчивающиеся командой СТОП (например, в порядке увеличения их длин), и для каждого пути α составляем систему неравенств $N(\alpha)$. По этой системе неравенств проверяем: реализуем ли этот путь или нет. Если удается построить набор реализуемых путей, содержащий все дуги программы, то задача построения ПСП решена. Однако эта процедура может быть весьма неэффективной, и кроме этого она не будет останавливаться в том случае, если хотя бы одна дуга программы на самом деле нереализуема. Для преодоления этих трудностей, мы введем понятие состояния. Содержательно, состояние - это вся та информация, которая определяет дальнейшее поведение программы. Более формально: пусть F - алгоритм, который каждому пути α сопоставляет некоторый объект $F(\alpha)$, например, систему неравенств, граф, слово и т.п. Если для любой программы P и любых двух реализуемых путей α_1 и α_2 программы P, заканчивающихся одной и той же командой, из того, что $F(\alpha_1) = F(\alpha_2)$, следует, что множества реализуемых продолжений путей α_1 и α_2 совпадают, то алгоритм F будем называть <u>методом построения состояний</u>, а $F(\alpha)$ - <u>состоянием</u> программы после пути α.

Теперь для того, чтобы выяснить, реализуема ли некоторая дуга программы или нет, мы кроме систем неравенств будем строить также состояния. Если после нескольких путей, ведущих в рассматриваемую дугу, состояния совпадут, то из всех этих пу-

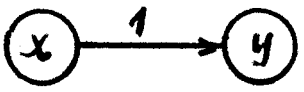
тей мы будем рассматривать только один путь. Если рассматриваемая дуга реализуема как продолжение одного из этих путей, то она реализуема также как продолжение выбранного пути. Если общее число состояний для программы окажется конечным, то мы сможем ограничиться анализом конечного числа путей, ведущих в рассматриваемую дугу. Это означает, что мы сможем для любой дуги выяснить, реализуема ли она или нет, и, следовательно, построить ПСП. Таким образом, в алгоритме построения ПСП центральной проблемой является нахождение "хорошего" метода построения состояний.

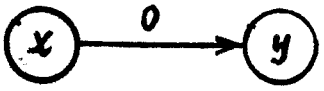
Нетрудно убедиться, что состоянием может служить система, получаемая объединением $E(\alpha)$ и $N(\alpha)$ в одну систему, которую обозначим через $EN(\alpha)$. Однако, в этом случае уже для простых программ число состояний может оказаться бесконечным. Поэтому мы постараемся исключить из $EN(\alpha)$ излишнюю информацию, и в качестве состояния будем употреблять сокращенную систему, обозначаемую через $S(\alpha)$. Точнее, пусть α произвольный реализуемый путь, β - произвольное продолжение пути α , $\alpha+\beta$ - путь, составленный из α и β , и $N_\alpha(\beta)$ - система неравенств, добавляемых на пути β к $N(\alpha)$ при построении $N(\alpha+\beta)$ (т.е. $N(\alpha+\beta)=N(\alpha)+N_\alpha(\beta)$. $EN(\alpha)$ будем преобразовывать таким способом, чтобы система, получаемая объединением сокращенной системы $S(\alpha)$ и системы неравенств $N_\alpha(\beta)$, имела бы решение тогда и только тогда, когда решение имеет $N(\alpha+\beta)$. Ясно, что система $S(\alpha)$, удовлетворяющая этому свойству, является состоянием.

Перед изложением алгоритма построения состояния, мы введем еще несколько новых понятий.

Во первых, изобразим $EN(\alpha)$ (и ей подобные системы) в виде графа, которого обозначим через $G(EN(\alpha))$. Напомним, что $EN(\alpha)$ может содержать обозначения записей, константы и имена переменных и входных файлов, а также знаки отношений ">", "$\geq$" и "=". Записи и константы мы изобразим вершинами графа $G(EN(\alpha))$, отношения ">" и "$\geq$" дугами этого графа, а имена переменных и входных файлов мы припишем к вершинам графа $G(EN(\alpha))$ в виде дополнительных меток (знак "=" в $G(EN(\alpha))$ не изображается).

Неравенство $x>y$ (соответственно $x\geq y$), где x, y - константа или запись, изобразим вершинами x и y и дугой весом "1" (соответственно весом "0"), ведущей из вершины x в вершину y :



(соответственно,


).

Равенство вида $g=x$, где g - имя переменной или входного файла, а x - константа или запись, изобразим вершиной x , к которой припишем метку g , называемую <u>дополнительной</u>:

Если некоторая константа или запись в $EN(\alpha)$ встречается несколько раз, то для него строится только одна вершина. Аналогично, если две вершины должны соединяться несколько дугами, то строится лишь одна дуга с максимальным весом. Вершины, изображающие записи, будем называть <u>основными</u>, а вершины, изображающие константы, - <u>числовыми</u>.

Основные вершины, имеющие дополнительные метки, и все

числовые вершины назовем активными, а остальные - пассивными. Содержательно, активные вершины изображают константы программы и последние значения переменных после пути α, а пассивные вершины - записи, которые уже не являются значениями ни одной из переменных программы. Ясно, что на константы и записи, изображаемые активными вершинами, на некотором продолжении пути α возможно наложение новых логических отношений, а на записи, изображаемые пассивными вершинами, это невозможно.

Весом ориентированного пути графа $G(EN(\alpha))$ назовем сумму весов дуг, составляющих этот путь. Для программы ПРИМЕР-I система $EN(I,2,3,4,3,5,2,3,5) = \{c = a_2,\ d = b_2,\ A = 0, B = 0,$ $a_1 < b_1,\ a_2 \geq b_1,\ b_1 < a_2,\ a_2 \geq b_2\}$ изображается графом, представленным на рис.2.

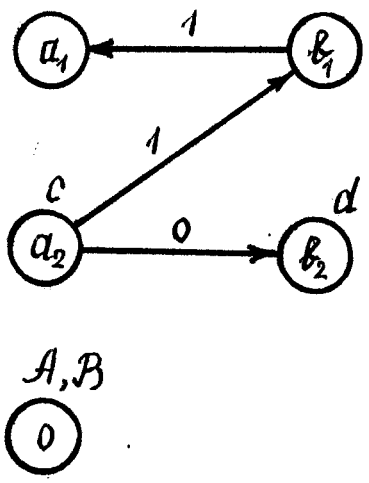


Рис.2.

В этом графе активными являются основные вершины a_2, b_2 и числовая вершина 0, а вес пути $((a_2, b_1), (b_1, a_1))$ равен двум.

При построении состояния нам понадобится необходимое и достаточное условия существования решения для $N(\alpha)$. Эти ус-

ловия мы сформулируем в виде следующей леммы.

Произвольную систему неравенств N назовем <u>базисной</u>, если она содержит только неравенства вида: $x \geq y+m$, $x \leq c$, $x \geq c$, $x > c$, где x, y - неизвестные, m - целое неотрицательное число, а c - целое число. Очевидно, $x > y$ равносильно $x \geq y+1$; поэтому система неравенств $N(\alpha)$ любого пути α любой программы в базисном языке является базисной. Ясно, что каждую базисную систему неравенств N можно изобразить в виде графа $G(N)$ аналогично как уже описано выше, только для неравенств вида $x \geq y+m$ весом дуги, ведущей из вершины x в вершину y, является число m.

<u>ЛЕММА I.2.</u> Произвольная базисная система неравенств N имеет решение тогда и только тогда, когда граф $G(N)$, изображающий эту систему неравенств, обладает свойствами:

1) $G(N)$ не имеет циклических путей весом больше нуля;

2) вес любого пути, ведущего из любой числовой вершины c_1 в любую другую числовую вершину c_2, не больше разности $c_1 - c_2$.

Необходимость очевидна. Докажем достаточность. Решение системы будем строить исходя из $G(N)$ постепенно приписывая основным вершинам некоторые значения. Эти значения будут служить решением системы N. Будем считать, что $G(N)$ является связным графом; в противном случае описанную ниже процедуру будем проводить для каждой связной части отдельно. Будем говорить, что вершина x графа $G(N)$ больше вершины y на p $(p>0)$ единиц (или y меньше x на p единиц), если мак-

симальный из весов путей, ведущих из x в y, равен p. Будем считать, что числовым вершинам значения уже присвоены - это соответствующие числа. Если $G(N)$ числовых вершин не имеет, то выбираем какую-нибудь вершину и присваиваем ей произвольное значение. Остальным вершинам значения будут присваиваться индуктивно. Опишем шаг индукции. Выбираем произвольную вершину x, которая больше (или меньше) какой-нибудь вершины y, которой уже присвоено значение a. Из связности $G(N)$ следует, что такая вершина x существует. Пусть вершина x больше (или меньше) вершины y на p единиц. Тогда вершине x присваиваем значение $a+p$ (соответственно $a-p$). Этот процесс продолжаем до момента, когда всем вершинам будут присвоены значения; эти значения являются решением системы N. Из условий леммы следует, что этот процесс всегда выполним. Лемма доказана.

Опишем теперь некоторый алгоритм S преобразования системы $EN(\alpha)$ для произвольного пути α. В дальнейшем будет доказано, что алгоритм S является методом построения состояний.

Сначала изобразим систему $EN(\alpha)$ в виде графа $G(EN(\alpha))$ и отметим в этом графе активные и пассивные вершины. Далее выполним следующие преобразования.

Преобразование I. Поочередно находим в $G(EN(\alpha))$ пассивные вершины, в которые или только входят или из которых только выходят дуги, и стираем эти вершины вместе со всеми дугами, входящими или выходящими из них. Если в результате этих преобразований образуются пассивные вершины, в которые вообще не входят и из которых не выходят дуги, то стираем также и эти вершины. Содержательно этими преобразованиями из $EN(\alpha)$ ис-

ключаются те пассивные записи, значения которых неравенствами из $N(\alpha)$ или вообще не ограничиваются или же с учетом транзитивности отношений ">" и "$\geqslant$" ограничиваются только с одной стороны (сверху или снизу).

Преобразование 2. Исключаем из $G(EN(\alpha))$ пассивные вершины, учитывая транзитивность отношений ">" и "$\geqslant$": если в пассивную вершину x входят дуги d_i с весами p_i и выходят дуги e_j с весами q_j, то любую пару дуг (d_i, e_j) заменяем на новую дугу f_k с весом $r_k = p_i + q_j$, а вершину x стираем. Если после этого две вершины соединены несколько дугами, то из этих дуг оставляем лишь одну с максимальным весом. Очевидно, эти преобразования не меняют максимальные веса путей, соединяющих активные вершины, однако эти веса могут оказаться сколь-угодно большими.

Преобразование 3. Заменяем веса дуг, превышающих число $d = \bar{c} - \underline{c} + 1$, где $\bar{c}$ - максимальная и $\underline{c}$ - минимальная константа программы, встречающаяся в командах пересылки и условного перехода, на новый вес равный числу d (если программа не имеет констант, то $d = 1$).

Граф, полученный после указанных выше преобразований, обозначим через $\tilde{G}(EN(\alpha))$. Таким образом $\tilde{G}(EN(\alpha))$ содержит только активные вершины, общее количество которых не превышает сумму числа переменных и констант, используемых в программе. Эти вершины могут быть соединены дугами, веса которых не превышают число d.

Далее сопоставим графу $\tilde{G}(EN(\alpha))$ систему $S(\alpha)$, которую назовем сокращенной. Если в $\tilde{G}(EN(\alpha))$ содержится дуга с весом m, которая ведет из вершины x в вершину y,

то в $S(\alpha)$ включим неравенство $x \geq y + m$. Если в $\tilde{G}(EN(\alpha))$ содержится активная вершина x с дополнительной меткой g , то в $S(\alpha)$ включим равенство $g = x$. Сокращенную систему $S(\alpha)$ будем считать результатом работы алгоритма S .

На рис.3 изображен граф, полученный из графа, представленного на рис.2, указанными выше преобразованиями; соответствующая сокращенная система имеет вид $\{c = a_2, d = b_2, A = 0, B = 0, a_2 \geq b_2\}$.

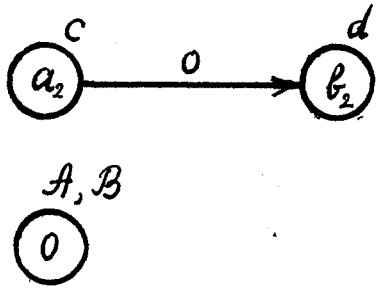


Рис.3.

Две сокращенные системы $S(\alpha_1)$ и $S(\alpha_2)$ будем считать равными (обозначение: $S(\alpha_1) = S(\alpha_2)$), если между обозначениями записей в $S(\alpha_1)$ и $S(\alpha_2)$ можно установить взаимно однозначное соответствие такое, что замена обозначений записей в $S(\alpha_1)$ соответствующими обозначениями из $S(\alpha_2)$ переводит $S(\alpha_1)$ в $S(\alpha_2)$. Например, система $S(1,2,3,4) = \{c = a_1, d = b_1, A = 0, B = 0, a_1 < b_1\}$ равна системе $S(1,2,3,4,3,4) = \{c = a_2, d = b_1, A = 0, B = 0, a_2 < b_1\}$

Ясно, что если $S(\alpha_1) = S(\alpha_2)$ и $S(\alpha_1)$ имеет решение, то также $S(\alpha_2)$ имеет решение и наоборот.

Теперь докажем, что сокращенная система является состоянием для программы в базисном языке. Сначала докажем два свойст-

ва сокращенных систем. Пусть α - произвольный путь программы и β - произвольное продолжение этого пути. Через $S(\alpha)+N_\alpha(\beta)$ обозначим систему, получаемую объединением сокращенной системы $S(\alpha)$ и системы неравенств $N_\alpha(\beta)$. (Напомним, что $N_\alpha(\beta)$ состоит из неравенств, добавляемых к $N(\alpha)$ при построении $N(\alpha+\beta)$, т.е. $N(\alpha+\beta)=N(\alpha)+N_\alpha(\beta)$).

СВОЙСТВО I.I. Пусть α_1 и α_2 - пути, оканчивающиеся одной и той же командой программы, и пусть $S(\alpha_1)=S(\alpha_2)$. Тогда для любого продолжения β путей α_1 и α_2 системы $S(\alpha_1)+N_{\alpha_1}(\beta)$ и $S(\alpha_2)+N_{\alpha_2}(\beta)$ равны.

Сразу ясно, что число отдельных неравенств в $S(\alpha_1)+N_{\alpha_1}(\beta)$ и в $S(\alpha_2)+N_{\alpha_2}(\beta)$ совпадает, и эти неравенства могут отличаться лишь обозначениями записей. Мы должны установить взаимно однозначное соответствие между обозначениями записей этих систем таким образом, чтобы замена обозначений записей в одной системе на соответствующие им обозначения из второй системы перевели бы одну систему в другую и наоборот. Ввиду того, что $S(\alpha_1)=S(\alpha_2)$, будем считать, что между обозначениями записей этих систем уже установлено необходимое соответствие. Из этого следует, что, если значением некоторой переменной g после пути α_1 является запись r_1, то значение этой же переменной после пути α_2 также является запись, например r_2, притом записи r_1 и r_2 соответствуют друг другу. Если же значением переменной g после пути α_1 является число, то значением этой же переменной после пути α_2 является то же самое число.

Допустим, что путь β содержит команду чтения записей и эта команда в переменную q на пути $\alpha_1+\beta$ засылает запись κ_3. Тогда эта же команда на пути $\alpha_2+\beta$ засылает в переменную q также запись, например, κ_4; будем считать, что записи κ_3 и κ_4 соответствуют друг другу. Ясно, что вновь установленное соответствие не противоречит ранее установленным. Если путь β содержит команду пересылки, и эта команда присваивает значение переменной q, то эта команда на обоих путях $\alpha_1+\beta$ и $\alpha_2+\beta$ присваивает переменной q значения, которые соответствуют друг другу. Таким образом установлено соответствие между очередными значениями переменных такое, что после любой команды пути β значения одной и той же переменной на путях $\alpha_1+\beta$ и $\alpha_2+\beta$ соответствуют друг другу. Теперь справедливость свойства I.I следует из того, что системы $N_{\alpha_1}(\beta)$ и $N_{\alpha_2}(\beta)$ строятся исходя только из очередных значений переменных, между которыми необходимое соответствие уже установлено. Свойство доказано.

СВОЙСТВО I.2. Для любого реализуемого пути α и его продолжения β система неравенств $N(\alpha+\beta)$ имеет решение тогда и только тогда, когда решение имеет система $S(\alpha)+N_\alpha(\beta)$.

Необходимость. Пусть $N(\alpha+\beta)$ имеет решение. Тогда граф $G(N(\alpha+\beta))$ удовлетворяет условиям леммы I.2. Мы убедимся, что также граф $G(S(\alpha)+N_\alpha(\beta))$ удовлетворяет условиям леммы I.2, следовательно, $S(\alpha)+N_\alpha(\beta)$ имеет решение.

Пусть x и y - произвольные вершины графа $G(S(\alpha)+N_\alpha(\beta))$ и пусть эти вершины соединяет некоторый путь. Нетрудно убедиться, что вершины x и y содержатся также в графе $G(N(\alpha+\beta))$. Теперь напомним, что при построении $S(\alpha)$ граф $G(EN(\alpha))$ преобразовался таким образом, что максимального веса путей, соединяющих активные вершины, возможно было только уменьшать. Таким образом максимальный вес пути, соединяющего вершины x и y в $G(S(\alpha)+N_\alpha(\beta))$ не больше максимального веса пути, соединяющего эти вершины в $G(N(\alpha+\beta))$ Поэтому, если в $G(N(\alpha+\beta))$ не имеется циклических путей весом больше нуля и вес любого пути, ведущего из одной числовой вершины c_1 в любую другую числовую вершину c_2 , не больше разницы c_1-c_2 , то тем более в $G(S(\alpha)+N_\alpha(\beta))$ не имеется циклических путей весом больше нуля и веса путей ведущих из числовой вершины c_1 в вершину c_2 не больше разницы c_1-c_2 Необходимость доказана.

Достаточность. Пусть $S(\alpha)+N_\alpha(\beta)$ имеет решение. Это означает, что $G(S(\alpha)+N_\alpha(\beta))$ удовлетворяет условиям леммы 1.2. Убедимся, что граф $G(N(\alpha+\beta))$ также удовлетворяет условиям леммы I.2; из этого будет следовать, что $N(\alpha+\beta)$ имеет решение.

В начале рассмотрим условие 2 из леммыI.2. Пусть c_1 и c_2 - произвольные числовые вершины графа $G(N(\alpha+\beta))$ (допустим, что $c_1>c_2$) и γ^* - произвольный путь, ведущий из c_1 в c_2 . Мы должны убедиться, что вес пути γ^* не превышает число c_1-c_2 .

Сначала напомним, что вершины c_1 и c_2 содержатся также в $G(S(\alpha)+N_\alpha(\beta))$ и из c_1 в c_2 ведет некоторый путь.

Допустим, что максимальный вес путей, ведущих из c_1 в c_2, равен числу m. Ввиду того, что $G(S(\alpha)+N_\alpha(\beta))$ удовлетворяет условию 2 леммы I.2 имеет место $c_1 - c_2 \geq m$. Далее из того, что $\underline{c} \leq c_1 \leq \bar{c}, \underline{c} \leq c_2 \leq \bar{c}$ и $d = \bar{c} - \underline{c} + 1$ ясно, что $m < d$. Таким образом ни для одной дуги, принадлежащей пути, ведущему из c_1 в c_2, при построении $S(\alpha)$ не применены преобразования 3 (т.е. не уменьшены веса дуг). Поэтому максимальный вес путей, ведущих из c_1 в c_2 в графе $G(N(\alpha+\beta))$, также равен числу m. Ввиду того, что $m \leq c_1 - c_2$, ясно, что вес пути γ^ℓ также не больше числа $c_1 - c_2$. Аналогично доказывается выполнение условия I леммы I.2. Свойство доказано.

Сформулируем теперь основную лемму:

ЛЕММА I.3. Алгоритм S является методом построения состояний.

Справедливость леммы легко вытекает из предыдущих утверждений. Пусть β - произвольное продолжение реализуемых путей α_1 и α_2, оканчивающихся одной и той же командой, для которых $S(\alpha_1) = S(\alpha_2)$. Мы должны убедиться, что $N(\alpha_1+\beta)$ имеет решение тогда и только тогда, когда решение имеет $N(\alpha_2+\beta)$. Согласно свойству I.2 ясно, что $N(\alpha_1+\beta)$ имеет решение тогда и только тогда, когда решение имеет $S(\alpha_1)+N_{\alpha_1}(\beta)$. Согласно свойству I.I, система $S(\alpha_1)+N_{\alpha_1}(\beta)$ и $S(\alpha_2)+N_{\alpha_2}(\beta)$ равны и поэтому $S(\alpha_2)+N_{\alpha_2}(\beta)$ имеет решение тогда и только тогда, когда решение имеет $S(\alpha_1)+N_{\alpha_1}(\beta)$. Теперь, повторно применяя свойство I.2, получаем, что $N(\alpha_2+\beta)$ имеет решение тогда и только тогда, когда решение имеет $S(\alpha_2)+N_{\alpha_2}(\beta)$, откуда следует необходимое утверждение.

ЛЕММА I.4. При методе построения состояний S общее число состояний любой программы в базисном языке (при переборе всевозможных путей программы) конечно.

Справедливость этой леммы вытекает из следующих простых соображений. Во первых, в графе $\tilde{G}(EN(\alpha))$, получаемого после преобразований графа $G(EN(\alpha))$, содержатся только активные вершины, общее количество которых не превышает сумму числа переменных и констант, используемых в программе. Во вторых, эти вершины могут быть соединены дугами, веса которых не превышают число d. Лемма доказана.

I.4. Алгоритм построения ПСП для программ в базисном языке.

В этом параграфе мы изложим алгоритм построения ПСП и, следовательно, завершим доказательство теоремы I.I. Для этого мы построим дерево, состоящее из путей программы, на которых "достигаются" всевозможные состояния программы . Отсюда будет следовать, что это же дерево содержит все дуги программы, которые в принципе реализуемы. Переходим к точным определениям.

Строим развертку программы P в виде дерева: к корню дерева (0-ой ярус) приписываем номер первой команды программы, т.е. единицу. Если q_k - произвольная вершина k-го яруса и к ней приписан номер команды $n(q_k)$, то из вершины q_k проводим столько дуг, сколько выходов имеет команда $n(q_k)$. В концах этих дуг строим вершины $(k+1)$-го яруса, к которым приписываем номера соответствующих команд. Между вершинами k-го яруса и путями программы устанавливаем взаимно-однозначное соответствие: вершине q_k, лежащей на ветви $h=(q_1, q_2, \dots q_k)$, сопоставим путь $(n(q_1), n(q_2), \dots n(q_k))$, который обозначим через $\alpha(q_k)$. Далее, просматриваем дуги, выходящие из вершин k-го яруса и обрезаем (стираем) дугу $(n(q_k), n(q_{k+1}))$ вместе с вершиной $(k+1)$-го яруса в следующих случаях:

1) Если путь $\alpha(q_k)$ реализуем, но путь, полученный из $\alpha(q_k)$ добавлением дуги $(n(q_k), n(q_{k+1}))$, - нереализуем; в этом случае будем говорить, что ветвь обрезана по нереализуемости.

2) Если $n(q_k)$ является номером команды СТОП (тривиальное обрезание). В этом случае ветвь дерева, на которой находится

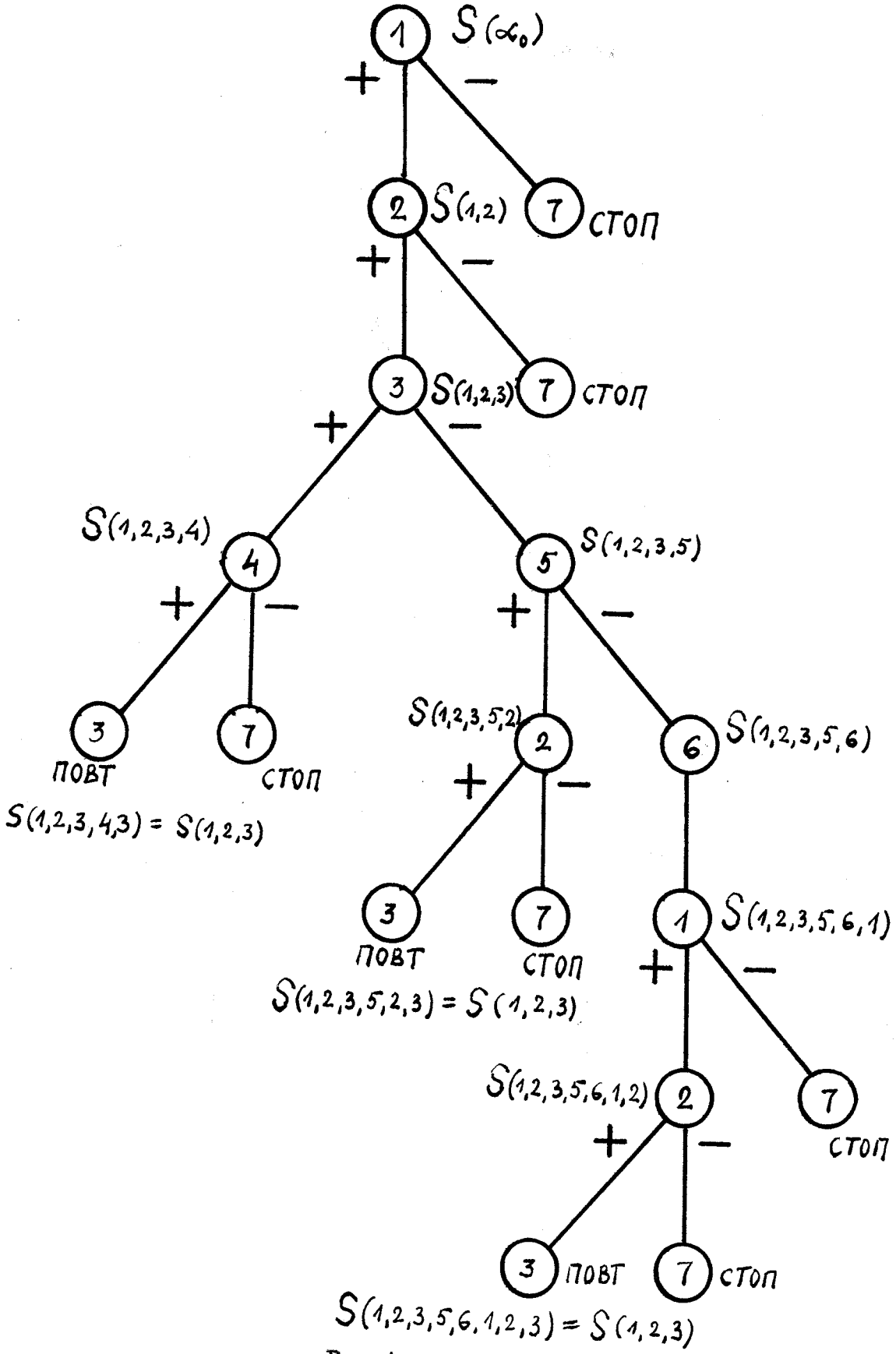

1
S(x0)
+
−
2
S(1,2)
7
СТОП
+
−
3
S(1,2,3)
7
СТОП
+
−
S(1,2,3,4)
4
S(1,2,3,5)
5
+
−
+
−
3
ПОВТ
7
СТОП
S(1,2,3,4,3) = S(1,2,3)
S(1,2,3,5,2)
2
6
S(1,2,3,5,6)
+
−
3
ПОВТ
7
СТОП
S(1,2,3,5,2,3) = S(1,2,3)
1
S(1,2,3,5,6,1)
+
−
S(1,2,3,5,6,1,2)
2
7
СТОП
+
−
3
ПОВТ
7
СТОП
S(1,2,3,5,6,1,2,3) = S(1,2,3)

Рис.4.а.

вершина q_k , назовем СТОП-ветвью.

3) Если I) или 2) не имеет место, то к вершинам ветви $h=(q_1,q_2,\dots q_k)$ приписываем состояния: к вершине q_i приписываем состояние $S(\alpha(q_i))$. Далее, просматриваем вершины ветви h и обрезаем дугу $(n(q_k),n(q_{k+1}))$, если среди вершин этой ветви имеется вершина q_i $(0\le i<k)$ такая, что к этой вершине приписан тот же номер команды и состояние, которые приписаны к вершине q_k (т.е. $n(q_i)=n(q_k)$ и $S(\alpha(q_i))=S(\alpha(q_k))$). В этом случае ветвь h будем называть обрезанной по повторению, а вершину q_i будем называть обрезающей для ветви h .

Из леммы I.I и эффективности построения состояний алгоритмом S видно, что процесс обрезания дуг - эффективный. После обрезания дуг, выходящих из вершин k-того яруса, переходим к построению вершин $(k+2)$-го яруса и обрезанию дуг, выходящих из вершин $(k+1)$-го яруса.

Ввиду того, что число состояний для любой программы конечно, процедура построения дерева всегда завершится через конечное число шагов. Построенное дерево назовем деревом реализуемости и обозначим его через $D(P)$.

На рис.4 представлено дерево реализуемости и состояния программы ПРИМЕР-I. В концах ветвей дерева реализуемости приписано: ПОВТ - для ветвей, обрезанных по повторению, СТОП - для СТОП-ветвей.

$S(\alpha_0) = \{c=0, d=0, A=0, B=0\}$

$S(I,2) = \{c=a_1, d=0, A=0, B=0\}$

$S(I,2,3) = \{c=a_1, d=b_1, A=0, B=0\}$

$S(I,2,3,4) = \{c=a_1, d=b_1, A=0, B=0, a_1<b_1\}$

$S(I,2,3,4,3) = \{c=a_2, d=b_1, A=0, B=0\}$

$S(1,2,3,5) = \{ c = a_1,\ d = b_1,\ A = 0,\ B = 0,\ a_1 \geq b_1 \}$

$S(1,2,3,5,2) = \{ c = a_1,\ d = b_1,\ A = 0,\ B = 0,\ a_1 > b_1 \}$

$S(1,2,3,5,2,3) = \{ c = a_1,\ d = b_2,\ A = 0,\ B = 0 \}$

$S(1,2,3,5,6) = \{ c = a_1,\ d = b_1,\ A = 0,\ B = 0,\ a_1 \geq b_1,\ a_1 \leq b_1 \}$

$S(1,2,3,5,6,1) = \{ c = a_1,\ d = b_1,\ A = 0,\ B = 0,\ a_1 \geq b_1,\ a_1 \leq b_1 \}$

$S(1,2,3,5,6,1,2) = \{ c = a_2,\ d = b_1,\ A = 0,\ B = 0 \}$

$S(1,2,3,5,6,1,2,3) = \{ c = a_2,\ d = b_2,\ A = 0,\ B = 0 \}$.

Рис.4.6.

Будем говорить, что произвольный путь $\alpha = (n_0, n_1, \ldots, n_k)$ <u>принадлежит ветви</u> $h = (q_1, q_2, \ldots, q_i, \ldots, q_{i+k}, \ldots, q_s)$ начиная с вершины q_i по вершине q_{i+k}, если $n_0 = n(q_i)$, $n_1 = n(q_{i+1})$, ..., $n_k = n(q_{i+k})$. Если q_{i+k} - последняя вершина ветви h, то будем говорить, что путь принадлежит концу ветви h.

Будем говорить, что путь α <u>согласуется</u> с $D(P)$, если α можно разделить на части $\alpha_1, \alpha_2, \ldots, \alpha_n$ ($\alpha = \alpha_1 + \alpha_2 + \ldots + \alpha_n$) такие, что:

1. Любой из путей α_i $i = 1,2,\ldots,n-1$ принадлежит концу некоторой обрезанной по повторению ветви h_i.

2. Любой из путей α_i $i = 2,3,\ldots n$ принадлежит h_i, начиная с обрезающей вершины для ветви h_{i-1}. (Разумеется ветви h_i и h_{i-1} имеют общую часть по крайней мере до вершины, которая является обрезающей для ветви h_{i-1}.)

Например, для программы ПРИМЕР-1 пути α = = (1,2,3,4,3,4,3,4) = (1,2,3,4,3) + (3,4,3) + (3,4) и β = = (1,2,3,5,2,3,4,3,5,6,1,2,3,4,3,5,2,7) = (1,2,3,5,2,3) + + (3,4,3) + (3,5,6,1,2,3) + (3,4,3) + (3,5,2,7) согласуются с деревом реализуемости.

ЛЕММА I.5. Произвольный путь α программы P (вообще говоря начинающийся не с первой команды программы P) реализуем тогда и только тогда, когда α согласуется с деревом реализуемости $D(P)$.

Достаточность. Пусть $\alpha = \alpha_1 + \alpha_2 + \ldots + \alpha_n$ согласуется с $D(P)$ и α_i - i-тая часть пути α, которая принадлежит концу некоторой ветви $h_i = (q_1, q_2, \ldots, q_s)$. Пусть обрезающей для h_i является вершина q_j, т.е. $n(q_s) = n(q_j)$ и $S(\alpha(q_s)) = S(\alpha(q_j))$. Ясно, что вершина q_j принадлежит также ветви h_{i+1}, которой, начиная с вершины q_j, принадлежит $(i+1)$-ая часть α_{i+1} пути α. Предположим, что путь $\alpha_1 + \alpha_2 + \ldots + \alpha_i$ - реализуем и $S(\alpha_1 + \alpha_2 + \ldots + \alpha_i) = S(\alpha(q_s))$. Мы должны убедиться, что путь $\alpha_1 + \alpha_2 + \ldots + \alpha_i + \alpha_{i+1}$ реализуем и $S(\alpha_1 + \alpha_2 + \ldots + \alpha_i + \alpha_{i+1}) = S(\alpha(q_j) + \alpha_{i+1})$.

Согласно построению дерева реализуемости путь $\alpha(q_j) + \alpha_{i+1}$ реализуем. Ввиду того, что $S(\alpha(q_j)) = S(\alpha(q_s)) = S(\alpha_1 + \alpha_2 + \ldots + \alpha_i)$, и по определению состояния заключаем, что реализуем также путь $\alpha_1 + \alpha_2 + \ldots + \alpha_i + \alpha_{i+1}$. Равенство состояний $S(\alpha_1 + \alpha_2 + \ldots + \alpha_i + \alpha_{i+1})$ и $S(\alpha(q_j) + \alpha_{i+1})$ вытекает непосредственно из свойства I.I. Достаточность доказана.

Необходимость. Пусть $\alpha = (n_1, n_2, \ldots, n_k)$ - произвольный реализуемый путь программы. Мы должны убедиться, что α согласуется с $D(P)$. Не ограничивая общности будем считать, что α начинается с первой команды программы. Рассмотрим ветвь $h_1 = (q_1, q_2, \ldots q_s)$ дерева реализуемости, задаваемую путем α, т.е. ветвь для которой $n(q_1) = n_1$, $n(q_2) = n_2, \ldots$. Ясно, что эта ветвь может продолжаться или до команды СТОП или же

может быть обрезана. Если α целиком принадлежит h_1, то лемма доказана. В противоположном случае ясно, что ветвь h_1 может быть обрезана только по повторению. Пусть обрезающей для h_1 является вершина q_j $(1 \le j < s)$. Выберем $\alpha_1 = \alpha(q_s)$, а оставшуюся часть пути α после α_1 обозначим через β, т.е. $\alpha = \alpha_1 + \beta$. Ясно, что путь β содержит меньше дуг чем α. Согласно обрезанию ветвей по повторению $S(\alpha(q_s)) = S(\alpha(q_j))$ и поэтому путь $\alpha(q_j) + \beta$ реализуем. Теперь рассмотрим ветвь h_2 дерева $Д(P)$, задаваемую путем $\alpha(q_j) + \beta$. (До вершины q_j она совпадает с ветвью h_1, а далее задается путем β.) Аналогичными рассуждениями получаем, что β можно разделить на две реализуемые части α_2 и γ такие, что $\alpha = \alpha_1 + \alpha_2 + \gamma$ и α_2 принадлежит концу ветви h_2 начиная с обрезающей вершины q_j ветви h_1. Ввиду того, что длина оставшейся части уменьшается, эту процедуру можно продолжать до момента, когда вся оставшаяся часть пути α целиком принадлежит некоторой ветви дерева реализуемости. Лемма доказана.

СЛЕДСТВИЕ I.I. Дуга программы P реализуема тогда и только тогда, когда она принадлежит некоторой ветви $Д(P)$.

Множество $Q(P)$ путей программы P будем называть покрывающим множеством для этой программы, если выполнены условия:

I. любой путь из $Q(P)$ согласуется с деревом реализуемости и заканчивается командой СТОП;

2. любая дуга программы, принадлежащая некоторому пути программы, который согласуется с $Д(P)$ и заканчивается коман-

дой СТОП, принадлежит также некоторому пути из $Q(P)$.

Рассмотрим процедуру построения покрывающего множества $Q(P)$ для произвольной программы P . Пусть уже построено дерево реализуемости $D(P)$. В начале включаем в $Q(P)$ все пути, задаваемые СТОП-ветвьями дерева реализуемости (базис индукции). Если дерево реализуемости СТОП-ветвей не содержит, то для этой программы $Q(P)$ пустая. Далее, отмечаем вершины дерева реализуемости, которые являются обрезающими хотя бы для одной ветви этого дерева, и назовем эти вершины активными. Процедуру построения $Q(P)$ выполняем по шагам. На каждом шаге включим в $Q(P)$ один путь, отличающийся от всех предыдущих, а количество активных вершин уменьшим на единицу. Шаг индукции: Ищем в уже созданном множестве путь, который содержит команду, соответствующую некоторой активной вершине дерева реализуемости: пусть этот путь имеет вид $\alpha = (n_1, n_2, \dots, \bar{n}, \dots, n_K)$, где $\bar{n}$ - номер команды, которая соответствует активной обрезающей вершине $\bar{q}$ дерева реализуемости $(\bar{n} = n(\bar{q}))$. Пусть $\bar{q}$ является обрезающей для ветви $h = (q_1, q_2, \dots, \bar{q}, q_{j+1}, \dots, q_{s-1}, q_s)$, т.е. $n(\bar{q}) = n(q_s)$ и $S(\alpha(\bar{q})) = S(\alpha(q_s))$. Тогда в $Q(P)$ включаем путь $\bar{\alpha} = (n_1, n_2, \dots, \bar{n}, n(q_{j+1}), \dots, n(q_{s-1}), \bar{n}, \dots, n_K)$ а вершину $\bar{q}$ исключаем из множества активных вершин. Если $\bar{q}$ является обрезающей для несколько ветвей дерева реализуемости, то в $Q(P)$ включаем не один, а несколько путей, задаваемых этими ветвями. Описанную выше процедуру продолжаем до момента, когда ни один путь из построенного множества не содержит команд, соответствующих активным вершинам дерева реализуемости.

Для программы ПРИМЕР-I покрывающее множество имеет вид

Q (ПРИМЕР-I) = { α_1 = (I,2,3,4,7), α_2 = (I,2,3,5,2,7),

$\alpha_3 = (1,2,3,5,6,1,2,7)$, $\alpha_4 = (1,2,3,5,6,1,7)$, $\alpha_5 = (1,2,7)$, $\alpha_6 = (1,7)$, $\alpha_7 = (1,2,3,4,3,4,7)$, $\alpha_8 = (1,2,3,5,2,3,4,7)$, $\alpha_9 = (1,2,3,5,6,1,2,3,4,7)$.

Убедимся, что описанная выше процедура действительно строит покрывающее множество. Ясно, что любой путь из построенного множества заканчивается командой СТОП и согласуется с деревом реализуемости. Пусть теперь δ - произвольная дуга программы, которая принадлежит некоторому пути α программы, который согласуется с деревом реализуемости и заканчивается командой СТОП. Мы должны убедиться, что δ принадлежит некоторому пути из $Q(P)$.

Вместо полного индуктивного доказательства, мы рассмотрим несколько упрощенный случай. Пусть α согласуется с $D(P)$ и его можно разделить на три части $\alpha_1, \alpha_2, \alpha_3$ $(\alpha = \alpha_1 + \alpha_2 + \alpha_3)$, где $\alpha_1, \alpha_2, \alpha_3$ принадлежат соответственно ветвям h_1, h_2 и h_3 дерева реализуемости (см. рис.5). Будем считать, что h_1, h_2, h_3 заканчиваются соответственно вершинами q_1, q_2, q_3 , ветви h_1 и h_2 обрезаны по повторению и обрезающими для них являются соответственно вершины $\bar{q}_1$ и $\bar{q}_2$, а h_3 является СТОП-ветвью. Вершину, являющуюся корнем $D(P)$, обозначим через q_0 . Путь, задаваемый участком ветви с вершины p_1 до вершины p_2 , обозначим через $\alpha(p_1, p_2)$.

Ввиду того, что α согласуется с деревом реализуемости, имеет место: $\alpha_1 = \alpha(q_0, q_1)$, $\alpha_2 = \alpha(\bar{q}_1, q_2)$, $\alpha_3 = \alpha(\bar{q}_2, q_3)$. Описанной выше процедурой в $Q(P)$ будет включены три пути:

$\beta_1 = \alpha(q_0, q_3)$, $\beta_2 = \alpha(q_0, \bar{q}_1) + \alpha(\bar{q}_1, q_1) + \alpha(\bar{q}_1, q_3)$,

$\beta_3 = \alpha(q_0, \bar{q}_2) + \alpha(\bar{q}_2, q_2) + \alpha(\bar{q}_2, q_3)$.

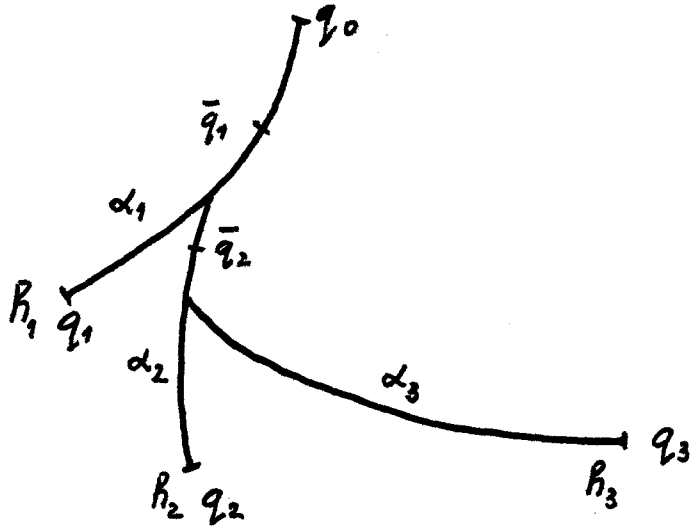


Рис.5.

Теперь ясно, что если дуга γ принадлежит пути α_1, то она принадлежит пути $\beta \in Q(P)$. Если γ принадлежит α_2, то она непременно принадлежит пути β_3, а если γ принадлежит α_3, то она принадлежит β_1. Этим доказана

ЛЕММА I.6. Дуга программы реализуема на допустимом СТОП-примере тогда и только тогда, когда она принадлежит некоторому пути из покрывающего множества $Q(P)$.

Доказательство теоремы I.I закончим кратким изложением алгоритма построения ПСП для произвольной программы P:

(I) Для данной программы P строим дерево реализуемости $D(P)$.

(2) Далее, строим покрывающее множество $Q(P) = \{\alpha_1, \alpha_2, \ldots \alpha_s\}$.

(3) Составляем для путей α_i $(i = 1,2,\ldots,s)$ системы неравенств $N(\alpha_i)$.

(4) Решаем системы неравенств $N(\alpha_i)(i=1,2,\ldots,s)$. Согласно леммы I.I решение системы неравенств $N(\alpha_i)$ задает пример T_i, который реализует путь α_i. Согласно леммы I.6 система примеров $\mathcal{M} = \{T_1, T_2, \ldots, T_s\}$ является полной для программы P. Теорема доказана.

I.5. Циклически полные системы примеров.

В данном параграфе мы рассмотрим вопрос о выявлении "ненормальных" ситуаций, когда программа на некотором конечном примере вообще не останавливается. В этом случае будем говорить, что программа зацикливается. Точнее, будем говорить, что пример T зацикливает программу P в множестве команд $E=\{n_1, n_2, \dots, n_s\}$, если запуская программу P на примере T, она начиная с некоторого момента выполняет только команды из E, притом каждую из них неограниченно много раз.

Систему примеров $\mathcal{N}$ будем называть циклически полной для программы P, если для любого множества команд E, в котором программа зацикливается на каком-нибудь примере, она зацикливается на некотором примере из $\mathcal{N}$.

ТЕОРЕМА I.2. Существует алгоритм, который для любой программы в базисном языке строит конечную циклически полную систему примеров.

Сначала докажем несколько лемм.

ЛЕММА I.7. Пусть пример T реализует путь $\alpha = (n_1, n_2, \dots, n_r, \dots, n_{r+m}, \dots)$. Пример T зацикливает программу в множестве команд E тогда и только тогда, когда существуют положительные числа r и m такие, что

1) $n_r = n_{r+m}$,

2) $S(n_1, n_2, \dots, n_r) = S(n_1, n_2, \dots, n_r, \dots, n_{r+m})$,

3) путь $\gamma^k = (n_r, n_{r+1}, \ldots, n_{r+m})$ содержит те и только те команды, которые содержатся в множестве E,

4) путь γ^k или вообще не содержит команд чтения записей, или же, если содержит, то эти команды передают управление следующей команде пути γ^k по выходу "-".

Достаточность. Пусть существуют положительные r и m такие, что выполнены условия 1), 2) и 4). Покажем, что пример T зацикливает программу в множестве команд, содержащее те и только те команды, которые содержатся в пути γ^k.

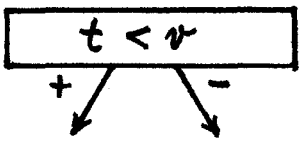
Справедливость этого утверждения вытекает из следующего вспомогательного утверждения:

Пусть команда n_r на рассматриваемом примере T передает управление следующей команде по выходу ε ($\varepsilon \in \{+|-\}$) и пусть выполняются условия 1), 2) и 4). Тогда имеет место:

(а) команда n_{r+m} на примере T также передает управление по выходу ε (следовательно, $n_{r+1} = n_{r+m+1}$);

(б) $S(n_1, n_2, \ldots, n_r, n_{r+1}) = S(n_1, n_2, \ldots, n_r, \ldots, n_{r+m}, n_{r+m+1})$.

Так как следующее состояние однозначно определяется предыдущим состоянием, командой n_r и ее выходом, то справедливость (б) вытекает из справедливости (а). Остается убедиться в справедливости (а). Утверждение (а) нетривиально, если команда n_r является командой условного перехода



Для определенности предположим, что на примере T условие $t < v$ выполняется, т.е. переход из n_r в n_{r+1} осу-

ществляется по выходу "+".

В системе $E(n_1, n_2, \dots, n_r)$ должны содержаться равенства вида $t = r_1$ и $v = r_2$. Поэтому в $EN(n_1, n_2, \dots, n_r, n_{r+1})$ содержится неравенство $r_1 < r_2$. Ввиду того, что $S(n_1, n_2, \dots, n_r) = S(n_1, n_2, \dots, n_r, \dots, n_{r+m})$ и на пути $\gamma = (n_r, n_{r+1}, \dots, n_{r+m})$ не проводится чтение записей, то неравенство $r_1 < r_2$ будет содержаться также в $S(n_1, n_2, \dots, n_r)$. Допустим, что в $S(n_1, n_2, \dots, n_r, \dots, n_{r+m})$ содержатся равенства $t = r_1'$ и $v = r_2'$. Ввиду того, что состояния $S(n_1, n_2, \dots, n_r)$ и $S(n_1, n_2, \dots, n_r, \dots, n_{r+m})$ равны, в $S(n_1, n_2, \dots, n_r, \dots, n_{r+m})$ содержится неравенство $r_1' < r_2'$. Теперь очевидно, что переход из команды n_{r+m} в команду n_{r+m+1} может осуществляться только по выходу "+".

Случай, когда переход из команды n_r в команду n_{r+1} осуществляется по выходу "-", анализируется аналогично.

Необходимость. Пусть пример T реализует путь $\alpha = (n_1, n_2, \dots, n_r, \dots, n_{r+m}, \dots)$ и зацикливает программу в некотором множестве E. Ввиду того, что количество команд и состояний для любой программы конечно, можно найти r и m такие, что $n_r \in E$, $n_r = n_{r+m}$ и $S(n_1, n_2, \dots, n_r) = S(n_1, n_2, \dots, n_r, \dots, n_{r+m})$. Кроме этого r и m можно выбрать таким образом, чтобы на пути $\gamma = (n_r, n_{r+1}, \dots, n_{r+m})$ не проводилось чтение записей (точнее, выход по "+"). Это обусловлено тем, что пример T конечный, а каждая команда из E выполняется неограниченно много раз. Теперь из вспомогательного утверждения, доказанного выше, следует, что пример T зацикливает программу в множестве команд, содержащем те и только те команды, которые содержатся в пути $\gamma = (n_r, n_{r+1}, \dots, n_{r+m})$. Ясно, что один пример не

может программу зацикливать в двух различных множествах команд; поэтому множество E совпадает с множеством команд, содержащихся в пути γ . Лемма доказана.

Ветвь дерева реализуемости $h = (q_1, q_2, \dots, q_r, \dots, q_{r+m})$ будем называть циклической, если выполнены условия:

1) h обрезана по повторению; пусть обрезающей вершиной для ветви h является вершина q_r ;

2) на пути $\gamma = (\pi(q_r), \pi(q_{r+1}), \dots, \pi(q_{r+m}))$ нет команд чтения записей, которые передают управление следующей команде пути γ по выходу "+".

Множество команд, приписанных от обрезающей по концевую вершину циклической ветви, назовем циклическим множеством этой ветви.

ЛЕММА I.8. Пусть $h = (q_1, q_2, \dots, q_r, \dots, q_{r+m})$ - циклическая ветвь дерева реализуемости и пусть обрезающей для ветви h является вершина q_r . Тогда программа зацикливается в циклическом множестве ветви h на любом примере, который реализует путь $\alpha(q_{r+m}) = (\pi(q_1), \pi(q_2), \dots, \pi(q_r), \dots, \pi(q_{r+m}))$.

Пусть T - произвольный пример, который реализует путь $\alpha(q_{r+m})$. Очевидно, для этого примера выполнены условия леммы I.7. Отсюда вытекает справедливость леммы I.8.

ЛЕММА I.9. Пусть E - множество команд, в котором пример T зацикливает программу P . Тогда в дереве реализуемости $D(P)$ существует циклическая ветвь с циклическим множеством E .

Пусть пример T зацикливает программу P в множестве команд E. Этому примеру соответствует бесконечный путь $\alpha = (n_1, n_2, \ldots, n_i, \ldots)$. Введем обозначения $S_i = S(n_1, n_2, \ldots, n_i)$. Теперь рассмотрим последовательность пар $((S_1, n_1), (S_2, n_2), \ldots$ $\ldots, (S_i, n_i), \ldots)$.

Из леммы I.7 вытекает, что начиная с некоторого места, эта последовательность станет периодической, т.е. существуют положительные числа r и m такие, что $n_{r+i} = n_{r+m+i} = n_{r+2m+i} = \ldots$ и $S_{r+i} = S_{r+m+i} = S_{r+2m+i} = \ldots$ где $i = 0, 1, 2, \ldots, m-1$. Притом путь $\gamma = (n_r, n_{r+1}, \ldots, n_{r+m})$ не содержит команд чтения записей, которые передают управление следующей команде пути γ по выходу "+". Рассмотрим наименьшие числа r и m такие, что в последовательности $((S_r, n_r), (S_{r+1}, n_{r+1}), \ldots, (S_{r+m}, n_{r+m}))$ все пары различны.

Согласно леммы I.5 из того, что путь α реализуем, заключаем, что путь α согласуется с деревом реализуемости. С другой стороны путь α начиная с команды n_r становится периодическим. Теперь нетрудно убедиться, что в дереве реализуемости существует ветвь $h = (q_1, q_2, \ldots, q_s, \ldots, q_{s+m})$, обрезающей для которой является вершина q_s, для которой $n(q_s) = n_r$, $n(q_{s+1}) = n_{r+1}, \ldots, n(q_{s+m}) = n_{r+m}$. Ясно, что ветвь является циклической и циклическим множеством этой ветви является множество E. Лемма доказана.

Для завершения доказательства теоремы I.2 вкратце изложим алгоритм построения циклически полной системы примеров для произвольной программы P:

(I) строим дерево реализуемости для программы P;

(2) находим циклические ветви в дереве реализуемости;

(3) строим примеры для путей, задаваемых циклическими ветвями дерева реализуемости.

Из доказанных лемм непосредственно вытекает, что построенная система примеров является циклически полной.

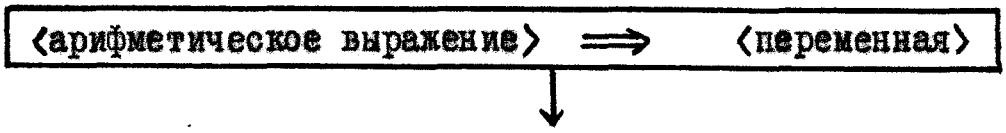
Глава 2.

РАСШИРЕНИЯ БАЗИСНОГО ЯЗЫКА

2.I. Расширение базисного языка командами сложения и вычитания.

В данной главе продолжается исследование границ разрешимости проблемы автоматического построения ПСП в зависимости от видов команд, используемых в программе. Большинство из этих результатов получены в совместной статье /24/, и они приводятся для полноты картины.

Первым наиболее естественным расширением базисного языка является введение команд сложения и вычитания. Формально это означает, что в программе, кроме команд базисного языка, можно использовать команды вида:



где ⟨арифметическое выражение⟩ ::= ⟨операнд⟩ ⟨знак операции⟩ ⟨операнд⟩ ,

⟨операнд⟩ ::= {⟨переменная⟩ | ⟨константа⟩} ,

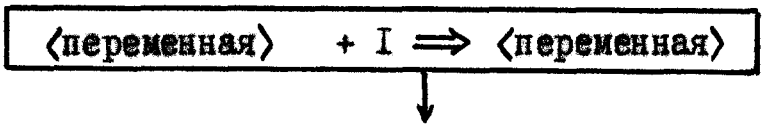
⟨знак операции⟩ ::= {+ | -} .

Вновь полученный язык назовем базисным языком с адитивной арифметикой. Понятия пути программы, примера программы, полной системы примеров программы и т.п. полностью аналогичны соответствующим понятиям для базисного языка.

Нетрудно убедиться, что не существует алгоритм, строящий

ПСП для любой программы в базисном языке с адитивной арифметикой. Этот факт также вытекает из следующего более сильного утверждения:

УТВЕРЖДЕНИЕ 2.I. Если в программах, кроме команд базисного языка, разрешается использовать команды вида



,

причем только для одной переменной, то проблема автоматического построения ПСП неразрешима.

Идея доказательства состоит в том, что проблему останова машин Минского /7/, которая является неразрешимой /6/, можно свести к проблеме построения ПСП. Каждой машине Минского M и значению аргумента x сопоставим программу $P(M,x)$ такую, что $P(M,x)$ имеет непустое ПСП тогда и только тогда, когда машина M на x останавливается. $P(M,x)$ имеет один входной файл и она проверяет, записано ли в этом файле последовательность значений регистров (конфигураций) машины M, начинающей работу с x. В этой проверке по существу используем команды сложения, а именно, мы должны убедиться, что очередное значение регистра отличается на единицу от предыдущего значения того же регистра. Такую проверку можно осуществлять при помощи одной переменной, к которой применяются команды сложения с единицей. Если в файле действительно записана последовательность конфигураций машины M, начинающей работу с x, и машина M на x останавливается, то $P(M,x)$ также останав-

ливается. Таким образом любая полная система примеров для $P(M,x)$ должна содержать файл, содержащий последовательность конфигураций работы машины M на x, на котором M останавливается.

В предыдущем доказательстве по существу использовался тот факт, что некоторая переменная, к которой применялись команды сложения, сравнивалась с другими переменными. Исключим теперь эту возможность и введем базисный язык со счетчиками.

Под счетчиками понимаются переменные, которым можно присваивать значения только двумя операциями: засылкой константы и прибавлением или вычитанием константы и которые можно сравнивать только с константами.

УТВЕРЖДЕНИЕ 2.2. Не существует алгоритм, строящий ПСП для любой программы в базисном языке с двумя счетчиками.

Проблему останова машин Минского опять сводим к проблеме построения ПСП. Любой двухрегистровой машине Минского M и аргументу x сопоставим программу $P(M,x)$ в базисном языке с двумя счетчиками, которая работает следующим образом. В начале засылаем в первый счетчик константу, равную x, а во второй счетчик нуль. Дальше $P(M,x)$ работает как машина M. Это возможно потому, что в базисном языке со счетчиками можно смоделировать все команды машин Минского. Ясно, что если M останавливается на x, то любой файл образует ПСП для $P(M,x)$ В противоположном случае ПСП для $P(M,x)$ является пустой.

С другой стороны имеет место:

УТВЕРЖДЕНИЕ 2.3. Существует алгоритм, который строит ПСП для любой программы в базисном языке с одним счетчиком.

Основная идея доказательства заложена в утверждении: Если в программе P какая-то дуга реализуема, то она реализуема на таком примере, на котором значение счетчика по абсолютной величине не превосходит значения некоторой эффективной функции $S(P)$ от размера программы P (числа команд, числа переменных, величин и числа констант). Теперь состоянием программы может служить аналогичная система как в базисном языке, только для счетчика в состоянии включается его значение. Притом, значения счетчика, превышающие число $S(P)$, в состоянии изображаются одинаково. Ясно, что общее число состояний для любой программы в базисном языке с одним счетчиком конечно и для построения ПСП можно пользоваться алгоритмом, описанным в предыдущей главе.

Последним рассмотрим случай, когда все счетчики, используемые в программе, являются односторонними. Односторонними будем называть счетчики, к которым в данной программе только прибавляются (соответственно, вычитаются) положительные константы и которые можно сравнивать только с константами. Имеет место:

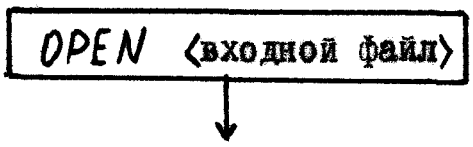
УТВЕРЖДЕНИЕ 2.4. Существует алгоритм, который строит ПСП для любой программы в базисном языке с односторонними счетчиками.

Справедливость этого утверждения вытекает из результатов третьей главы.

Что касается автоматического построения циклически полных систем примеров (ЦПСП) для рассмотренных расширений базисного языка, то во всех случаях, когда возможно построение ПСП, возможно также построение ЦПСП и, наоборот, во всех случаях, когда проблема построения ПСП неразрешима, неразрешима также проблема построения ЦПСП.

2.2. Расширение базисного языка средствами повторного чтения данных.

Рассмотрим ситуацию, когда одной программой входные файлы разрешается просматривать несколько раз. Для этого будем использовать команду открытия файла:



После выполнения этой команды очередной считываемой записью становится первая запись указанного входного файла. Вновь полученный язык будем называть базисным языком с повторным чтением данных. Имеет место:

УТВЕРЖДЕНИЕ 2.5. Не существует алгоритм, строящий ПСП для любой программы в базисном языке с повторным чтением двух файлов (даже один раз).

Доказательство этого утверждения состоит в том, что проблему соответствия Поста, которая является неразрешимой /6/, можно свести к построению ПСП.

Проблема соответствия формулируется следующим образом (см./7/): Пусть A - алфавит, содержащий не менее двух символов, и E - конечное множество пар слов (g_i, h_i) в алфавите A. Возможно ли для произвольного множества пар слов E найти последовательность индексов $i_1 i_2 \dots i_N$ такую, что слова $g_{i_1} g_{i_2} \cdots g_{i_N}$ и $h_{i_1} h_{i_2} \dots h_{i_N}$ одинаковы?

Теперь покажем, что для любого множества пар слов E можно построить программу $P(E)$ в базисном языке с повторным чтением данных такую, что программа $P(E)$ имеет непустое ПСП тогда и только тогда, когда для E существует последовательность индексов $i_1 i_2 \dots i_N$ такая, что слова $g_{i_1} g_{i_2} \dots g_{i_N}$ и $h_{i_1} h_{i_2} \dots h_{i_N}$ одинаковы. Программа $P(E)$ имеет два входных файла, и она работает следующим образом: сначала $P(E)$ считывает оба файла и проверяет: записано ли в первом файле слово $g_{i_1} g_{i_2} \dots g_{i_N}$, последовательность индексов $i_1 i_2 \dots i_N$ которого записан во втором файле. Если это не имеет место, то $P(E)$ зацикливается; в противоположном случае для обоих файлов применяется команда $OPEN$. Затем оба файла считываются заново, но этот раз проверяется записано ли в первом файле слово $h_{i_1} h_{i_2} \dots h_{i_N}$, последовательность индексов которого записан во втором файле. Если это не имеет место, то $P(E)$ зацикливается; в противоположном случае $P(E)$ останавливается. Ясно, что полная система примеров программы $P(E)$ содержит пару слов $(g_{i_1} g_{i_2} \dots g_{i_N}, i_1 i_2 \dots i_N)$, для которой слова $g_{i_1} g_{i_2} \dots g_{i_N}$ и $h_{i_1} h_{i_2} \dots h_{i_N}$ одинаковы.

Теперь сформулируем один частный случай, для которого проблема автоматического построения ПСП является разрешимой.

УТВЕРЖДЕНИЕ 2.6. Существует алгоритм, строящий ПСП для любой программы в базисном языке с повторным чтением данных, в которой команда $OPEN$ используется только для одного входного файла.

Доказательство этого утверждения приведено в /24/.

2.3. Расширение базисного языка средствами текстового сравнения данных.

На практике данные сравниваются не только в обычном числовом отношении, но также в так называемом текстовом отношении. Пусть x, y - целые неотрицательные числа и пусть $l(x)$ и $l(y)$ - длины десятичных представлений этих чисел. Пусть $l(x) \geq l(y)$. Тогда дополняем число y нулями справа таким образом, чтобы вновь полученно число y' имело бы ту же длину, что x, т.е. $l(x) = l(y')$. Если теперь число x больше числа y', то будем говорить, что число x в текстовом отношении больше числа y (обозначение: $x\ (T>)\ y$).

Например, I (T>) 0, 55 (T>) 54, 5 (T>) 49, 60 (T>) 5 и т.д. Аналогично вводится понятие: число x в текстовом отношении не меньше числа y. Подобное сравнение данных можно найти в алгоритмических языках КОБОЛ /3/ и РЛ/I /4/, только дополнение кратчайшего слова проводится не нулями, а некоторым другим символом.

Рассмотрим теперь язык программирования, который отличается от базисного языка только командой условного перехода

В данном случае условие имеет вид:

⟨условие⟩ ::= { ⟨переменная$_I$⟩ (T>) ⟨переменная$_2$⟩ |
| ⟨переменная$_I$⟩ (T>) ⟨константа⟩ |
| ⟨константа⟩ (T>) ⟨переменная$_I$⟩ }.

Этот язык назовем базисным языком с текстовым сравнением.

Имеет место:

УТВЕРЖДЕНИЕ 2.7. Существует алгоритм, строящий ПСП для любой программы в базисном языке с текстовым сравнением.

Это утверждение доказывается аналогично как теорема I.I, только вместо леммы I.2 используется:

ЛЕММА I.IO. Произвольная система неравенств N, содержащая только текстовые неравенства, имеет решение тогда и только тогда, когда граф $G(N)$, изображающий эту систему неравенств, не имеет циклических путей весом больше нуля.

В том случае, когда в одной программе разрешаются переменные сравнивать как в числовом, так и в текстовом отношении, вопрос о том, возможно ли автоматическое построение ПСП, пока нерешен. В основном это обусловлено тем, что не удается получить аналог леммы I.2.

В частном случае, когда в одной программе одни и те же переменные сравниваются или только в числовом, или только в текстовом отношении (с учетом пересылки данных!), проблема автоматического построения ПСП разрешима.

Глава 3

ПРИНЦИПЫ ПОСТРОЕНИЯ ПСП ДЛЯ РЕАЛЬНЫХ ПРОГРАММ ОБРАБОТКИ ДАННЫХ

3.I. Промежуточный язык программирования.

Ясно, что базисный язык является слишком бедной моделью реальных языков программирования. Поэтому в данной главе мы рассмотрим более совершенный, так называемый промежуточный язык. Этот язык кроме команд базисного языка будет содержать также команды сложения и вычитания. Из результатов второй главы сразу видно, что не существует алгоритм, строящий ПСП для любой программы в промежуточном языке. Таким образом алгоритм, излагаемый в данной главе, не применим во всех случаях; он ориентирован на построение ПСП для "простых" программ. Кроме этого мы будем учитывать много рационализаций, полезных при его практической реализации.

В промежуточном языке мы будем оперировать над объектами четырех видов: над переменными программы, счетчиками, файлами и массивами. Переменные программы будем понимать в таком же смысле как в базисном языке; их значениями служат любые целые числа. Под счетчиками будем понимать переменные, которым значения можно присваивать только двумя операциями: засылкой константы и прибавлением или вычитанием положительной константы, и которые можно сравнивать только с константами. Значениями файлов также как в базисном языке будут служить конечные

последовательности целых чисел; элементы этих последовательностей также будем называть записями. Будем различать входные и выходные файлы; доступ к записям - последовательный.

Под массивом будем понимать кортеж переменных, которые, следуя традиции, будем называть элементами массива. Элементы массивов также могут принимать только целые значения. На элементы массивов будем ссылаться двумя способами:

- ⟨имя массива⟩ (⟨константа⟩), например, $\mathcal{M}(1), \mathcal{M}(2)$;
- ⟨имя массива⟩ (⟨имя счетчика⟩), например, $\mathcal{M}(s)$,

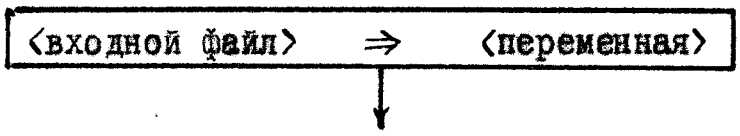
где s - счетчик; при s равном 3 $\mathcal{M}(s)$ означает $\mathcal{M}(3)$.

Именами массивов будут служить символы из бесконечного алфавита $\mathcal{R} = \{\mathcal{M}, \mathcal{P}, \ldots\}$. Кроме этого будем считать, что каждому символу из $\mathcal{R}$ сопоставлено число, указывающее количество элементов в массиве с данным именем.

В дальнейшем переменные программы, счетчики и элементы массивов будем называть внутренними переменными, а переменные программы, если это не вызовет недоразумений, - просто переменными.

Далее следуют команды промежуточного языка.

1) Команда чтения записи.

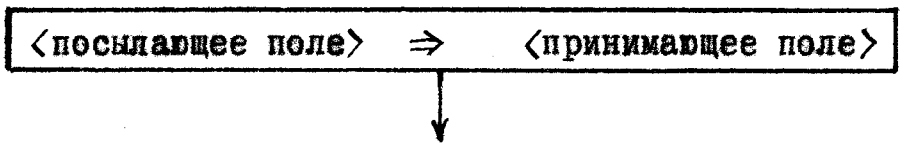


Смысл команды такой же как в базисном языке.

2) Команда пересылки.

Будем различать две модификации этой команды:

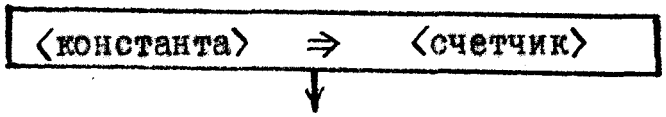
2.1)



где <посылающее поле> ::= {<переменная> | <константа> |
| <элемент массива>} ,
<принимающее поле> ::= {<переменная> | <элемент массива>},
<элемент массива> ::= {<массив> (<счетчик>) |
| <массив> (<константа>)} .

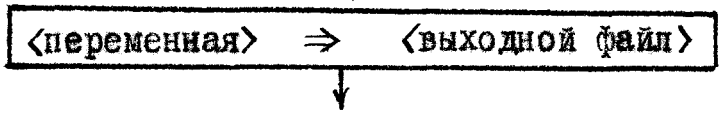
Если при выполнении команды значение счетчика, указывающего элемент массива, превышает количество элементов в массиве, то команда игнорируется (не выполняется).

2.2)



Команды обоих модификаций имеют один выход.

3) Команда вывода записей.



Смысл команды такой же как в базисном языке.

4) Команда условного перехода.

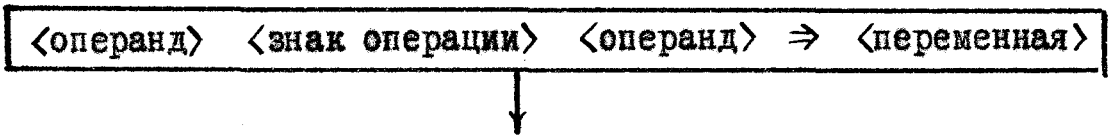
где <условие> ::= { <переменная$_1$> < <переменная$_2$> |
| <переменная> < <константа> |
| <константа> < <переменная> |
| <счетчик> < <константа> |
| <константа> < <счетчик>}.

Смысл команды очевиден.

5) Команда сложения-вычитания.

Команда имеет две модификации:

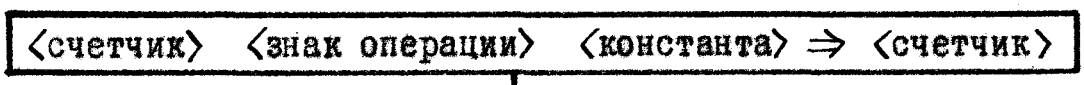
5.1)



где ⟨операнд⟩ ::= {⟨переменная⟩ | ⟨константа⟩},

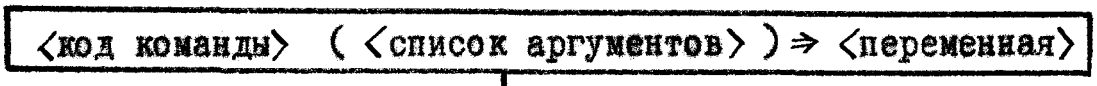
⟨знак операции⟩ ::= { + | - }.

5.2)



Слева и справа от знака "⇒" должен быть указан один и тот же счетчик, а константа всегда должна быть положительной.

6) Команды сложных вычислений. Обычные языки программирования содержат ряд других команд, которые с точки зрения описываемого алгоритма построения ПСП проводят слишком сложные преобразования значений переменных, например, умножение, деление, вычисление стандартных функций и т.п. Поэтому введем в промежуточный язык набор команд под общим видом:



где ⟨список аргументов⟩ ::= {⟨переменная⟩ |

| ⟨список аргументов⟩ ,

⟨переменная⟩ },

⟨код команды⟩ ::= $\{R_1 \mid R_2 \mid R_3 \mid \ldots \mid R_K\}$.

Для всех этих команд в начале выполняются действия, указанные кодом команды над значениями переменных из списка аргументов. Затем полученный результат присваивается указанной переменной.

В примере нам понадобится только одна команда сложных вычислений - команда умножения. Поэтому будем считать, что она имеет код - R_1 ; остальные команды сложных вычислений более подробно детализировать не будем.

7) Команда - конец работы.

Команда не имеет выходов.

Под программой в промежуточном языке будем понимать граф-схему, составленную из описанных выше команд. Программа всегда начинает выполняться с первой команды при нулевых значениях внутренних переменных и останавливается, когда попадает на команду СТОП. Такие понятия как путь программы, реализуемый путь, полная система примеров и т.п. аналогичны соответствующим понятиям для базисного языка. В дальнейшем будем рассматривать только такие пути, которые начинаются первой командой программы.

В отличие от базисного языка мы будем разрешать выходы некоторых команд оставлять свободными, т.е. выходы этих команд могут не вести ни в одну команду программы. Эти свободные выходы будем считать нереализуемыми.

В промежуточном языке значениями внутренних переменных служат только числа; в реальных языках программирования рассматриваются также текстовые данные. Разрешимые случаи построения ПСП для текстового сравнения данных уже рассмотрены в § 2.3. В данной главе основное внимание обращается на построение ПСП для программ, в которых используются арифметические команды. Поэтому средства обработки текстовых данных, хотя

и это не представляло бы принципиальных трудностей, не включены в промежуточный язык.

Отметим еще ту особенность промежуточного языка, что элементы массивов могут участвовать только в командах пересылки. Однако это ограничение с точки зрения принципиальных возможностей промежуточного языка несущественно. Например, если необходимо сравнить два элемента массива, то это можно смоделировать командами пересылки необходимых элементов массивов в некоторые переменные с последующим сравнением этих переменных.

Напомним, что в реальных языках программирования каждая запись обычно содержит целую группу реквизитов. Однако обработка записей проводится по отдельным реквизитам. Поэтому для удобства теоретических исследований в промежуточном языке принято, что каждая запись содержит только один реквизит; чтение группы реквизитов можно смоделировать несколькими командами чтения записи.

В качестве примера рассмотрим программу ПРИМЕР-2, представленную на рис.6. Для этой программы входным является файл A, который по своему содержанию разделяется на группы записей:

1-ая запись группы - шифр товара,
2-ая " - " - количество товара,
3-ая " - " - цена товара,
4-ая " - " - контрольная сумма первых трех записей.

Файл A упорядочен в порядке возрастания значений шифра товара. Для каждого товара в файле A может содержаться

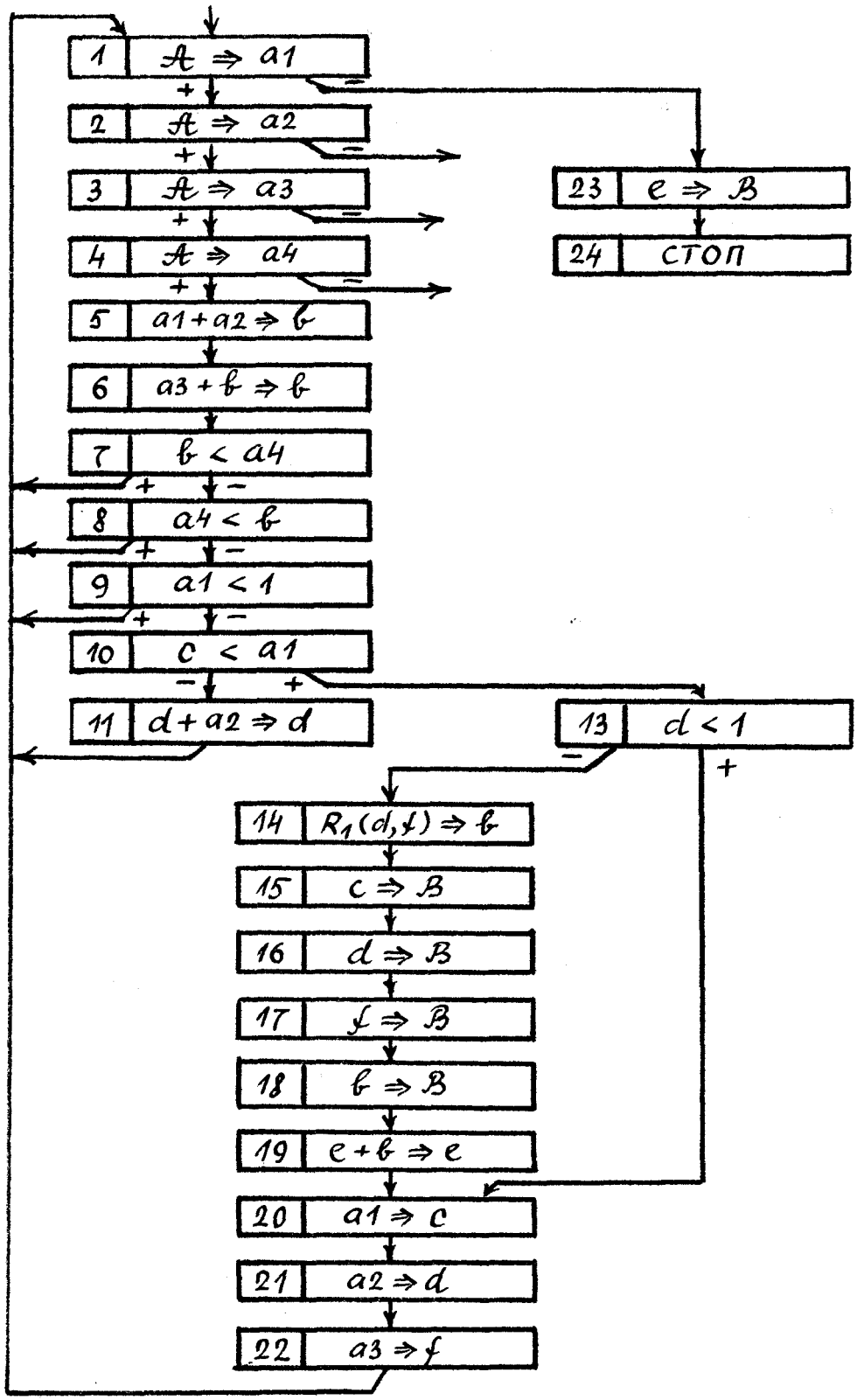

1 A ⇒ a1
2 A ⇒ a2
3 A ⇒ a3
4 A ⇒ a4
5 a1+a2 ⇒ b
6 a3+b ⇒ b
7 b < a4
8 a4 < b
9 a1 < 1
10 c < a1
11 d+a2 ⇒ d
13 d < 1
14 R1(d,f) ⇒ b
15 c ⇒ B
16 d ⇒ B
17 f ⇒ B
18 b ⇒ B
19 e+b ⇒ e
20 a1 ⇒ c
21 a2 ⇒ d
22 a3 ⇒ f
23 e ⇒ B
24 СТОП

Рис. 6.

несколько групп записей. Программа ПРИМЕР-2 из входного файла $\mathcal{A}$ образует новый файл $\mathcal{B}$, в котором для каждого товара имеется ровно одна группа записей со следующей информацией:

1-ая запись группы - шифр товара,

2-ая " - " - общее количество товара,

3-ая " - " - цена товара,

4-ая " - " - общая сумма товара.

Последней записью файла $\mathcal{B}$ является общая сумма всех товаров. Свободные выходы в командах 2,3,4 указывают, что число записей в любом входном файле, на котором будет выполнена эта программа, кратно четырем. Если в файле $\mathcal{A}$ для некоторой группы записей не верна контрольная сумма или шифр товара не является положительным числом, то данная группа записей просто пропускается.

Программа написана ошибочно: в файл $\mathcal{B}$ не выводятся данные о последнем товаре. Кроме этого программа содержит команду сложных вычислений с номером 14: $R_1(d, f) \Rightarrow b$, которая умножает значения переменных d и f, а результат записывает в переменную b.

3.2. Реализуемость путей и метод построения состояний для промежуточного языка.

В алгоритме построения ПСП для программ в промежуточном языке центральное место, также как для программ в базисном языке, занимает выбор "хорошего" метода построения состояний. В данной главе, аналогично как и раньше, состоянием после пути α будет служить некоторая сокращенная система $S(\alpha)$, которая также будет строится из системы $EN(\alpha)$. Основные отличия от базисного языка будут заключаться в следующем.

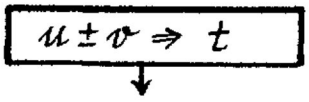
Во первых, в преобразованиях системы $EN(\alpha)$ мы будем учитывать присутствие в $EN(\alpha)$ арифметических выражений. Во вторых, мы будем использовать новое понятие активности записей. В первой главе активными считались записи, выражавшие последние значения переменных. В данной главе активными будем считать лишь те записи, которые на каком нибудь продолжении рассматриваемого пути вновь встречаются в неравенствах. Этим удается существенно сократить количество состояний, что важно при практической реализации.

В третьих, нельзя гарантировать, что общее число состояний для произвольной программы в промежуточном языке будет конечным. (Результаты второй главы фактически показывают, что даже в принципе невозможен метод построения состояний для промежуточного языка, при котором число состояний любой программы было бы конечным.) В том случае, когда число состояний для программы окажется конечным, ПСП будет строится аналогично как

в базисном языке. В противоположном случае наш алгоритм будет строить дерево реализуемости неограниченно долго, следовательно, он неприменим к этой программе. Заметим, что на практике эта ситуация не столь опасна. Дело в том, что память и время, выделяемое для работы любой программы, конечно, и поэтому построение дерева реализуемости всегда будет прекращено. Возможно, что в этих случаях будут построены примеры, пригодные для отладки программы, однако гарантии в том, что построенная система примеров является полной, не будет.

Переходим к построению систем $E(\alpha)$ и $N(\alpha)$. Пусть α - произвольный путь программы. $E(\alpha)$ и $N(\alpha)$ строятся аналогично как для программ в базисном языке, только описания значений строятся для всех внутренних переменных. Мы рассмотрим построение $E(\alpha)$ и $N(\alpha)$ только для команд сложения-вычитания и сложных вычислений; остальные случаи приведены уже в § I.2. Рассмотрим сразу шаг индукции. Пусть для пути $\alpha_i = (n_1, n_2, \dots, n_{i+1})$ состоящего из первых i дуг пути α построены уже системы $E(\alpha_i)$ и $N(\alpha_i)$. Строим $E(\alpha_{i+1})$ и $N(\alpha_{i+1})$.

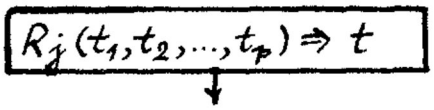
I) Пусть n_{i+1} -ой командой является команда



Пусть $E(\alpha_i)$ содержит равенства $t = r_1$, $u = r_2$, $v = r_3$. Тогда в $E(\alpha_i)$ равенство $t = r_1$ заменяем на новое равенство $t = r_2 \pm r_3$. Если u (соответственно, v) - константа c, то равенство $t = r_1$ заменяем на равенство $r_2 \pm c$ (соответственно, $c \pm r_3$). Например, если $E(\alpha_i)$ содержит равенст-

ва $u = x_1 + 3$ и $v = x_3 - 1$ и команда n_{i+1} имеет вид $u + v \Rightarrow t$ то в $E(\alpha_{i+1})$ появится равенство $t = x_1 + 3 + x_3 - 1$, которое эквивалентно $t = x_1 + x_3 + 2$. $N(\alpha_i)$ не меняется, т.е. $N(\alpha_{i+1})$ совпадает с $N(\alpha_i)$.

2) Пусть n_{i+1} -ой командой является команда сложных вычислений



где R_j - код команды. Пусть в $E(\alpha_i)$ содержится равенство $t = r$. Тогда это равенство заменяем на новое равенство $t = *$, где "*" - специальный символ, содержательно выражающий тот факт, что значение переменной t "деформировано" сложной командой и поэтому в дальнейшем не будет анализироваться. $N(\alpha_i)$ не меняется.

Для программы ПРИМЕР-2 имеем:

E (I,2,3,4,5,6) $= \{a1 = a_1, a2 = a_2, a3 = a_3, a4 = a_4, b = a_1 + a_2, c = 0, d = 0, e = 0, f = 0, h = 0\}$

E (I,2,3,4,5,6,7) $= \{a1 = a_1, a2 = a_2, a3 = a_3, a4 = a_4, b = a_1 + a_2 + a_3, c = 0, d = 0, e = 0, f = 0, h = 0\}$

E (I,2,3,4,5,6,7,8,9,I0,I3,I4,I5) $= \{a1 = a_1, a2 = a_2, a3 = a_3, a4 = a_4, b = *, c = 0, d = 0, e = 0, f = 0, h = 0\}$

E (I,2,3,4,5,6,7,8) $= \{a_1 + a_2 + a_3 \geq a_4\}$

N (I,2,3,4,5,6,7,8,9) $= \{a_1 + a_2 + a_3 \geq a_4, a_1 + a_2 + a_3 \leq a_4\}$

N (I,2,3,4,5,6,7,8,9,I0,I3,20) $= \{a_1 + a_2 + a_3 \geq a_4, a_1 + a_2 + a_3 \leq a_4, a_1 \geq 1, 0 < a_1, 0 < 1\}$.

Таким образом, левыми частями равенств из $E(\alpha)$ служат имена внутренних переменных и входных файлов, а правыми частя-

ми - числа, обозначения записей входных файлов, символ "*" и выражения, составленные из чисел, обозначений записей и символа "*" операциями сложения и вычитания. $N(\alpha)$ состоит из неравенств вида $x > y$ и $x \geq y$ где x, y - число, обозначение записи, символ "*" или выражение, составленное опять-таки из чисел, обозначений записей и символа "*" операциями сложения и вычитания. Неизвестными системы $N(\alpha)$ являются записи (точнее, обозначения записей) входных файлов.

Неравенство, не содержащее символ "*", назовем простым. Путь будем называть простым, если все неравенства из - простые. Содержательно, простыми являются пути, на которых командами сложных вычислений не деформируются значения тех переменных, которые сравниваются в командах условного перехода. Из построения $N(\alpha)$ вытекает:

ЛЕММА 3.I. Простой путь α программы в промежуточном языке реализуем тогда и только тогда, когда система неравенств $N(\alpha)$ имеет решение в целых числах; решение системы $N(\alpha)$ задает пример, который реализует путь α .

Ясно, что система неравенств любого простого пути линейна. Поэтому для ее решения в целых числах можно воспользоваться методом Гомори /2/.

В отличие от первой главы, мы будем пользоваться несколько другим понятием состояния. Вместо реализуемых путей мы будем рассматривать лишь простые реализуемые пути. Формально, пусть F - алгоритм, который каждому пути α сопоставляет

некоторый объект $F(\alpha)$ из некоторого множества объектов $\mathcal{F}$, например, систему неравенств, граф, слово и т.п. и пусть в множестве $\mathcal{F}$ определено отношение равенства объектов. Если для любой программы в промежуточном языке и любых двух простых реализуемых путей α_1 и α_2 программы, оканчивающихся одной и той же командой, из того, что $F(\alpha_1) = F(\alpha_2)$ следует, что множества реализуемых простых продолжений путей α_1 и α_2 совпадают, то алгоритм F будем называть методом построения состояний для промежуточного языка, а $F(\alpha)$ - состоянием программы после пути α.

Также как в первой главе изобразим $EN(\alpha)$ в виде графа $G(EN(\alpha))$. Содержательно, вершины этого графа будут изображать записи, числа, символ "ж" и выражения из $EN(\alpha)$, а дуги - отношения между этими записями, числами, символом "ж" и выражениями, налагаемые системой неравенств $N(\alpha)$.

Неравенство $x > y$ (соответственно $x \geq y$) из $N(\alpha)$, где x, y - число, запись, символ "ж" или выражение, изобразим вершинами x и y и дугой с весом "1" (соответственно, весом "0"), ведущей из вершины x в вершину y. Эти дуги назовем логическими. Равенство $g = x$ из $E(\alpha)$, где g - имя внутренней переменной или входного файла, а x - число, запись, символ "ж" или выражение, изобразим вершиной x, к которой припишем метку g, называемую дополнительной. Если некоторое число, запись, символ "ж" или выражение в $EN(\alpha)$ встречается несколько раз, то для него строится только одна вершина, а также из нескольких дуг, соединяющих одни и те же вершины, строится только одна дуга с максимальным весом. Вершины, изобража-

ющие записи, назовем основными; вершины, изображающие числа - числовыми, а вершины, изображающие выражения - арифметическими. Далее, из арифметических вершин проводим дуги, называемые арифметическими, в те вершины, которые изображают записи, числа и символ "*", входящие в эти выражения. Кроме этого припишем к этим дугам вес "+" (соответственно вес "-"), если соответствующее число, запись или символ "*" входит в выражение со знаком "+" (соответственно, со знаком "-"). Например, если $EN(\alpha)$ имеет вид $\{g=x_1, x_1+x_2+x_3>x_4\}$, то $G(EN(\alpha))$ имеет вид, представленный на рис.7. В этом рисунке арифметические дуги изображены прерывистыми стрелочками.

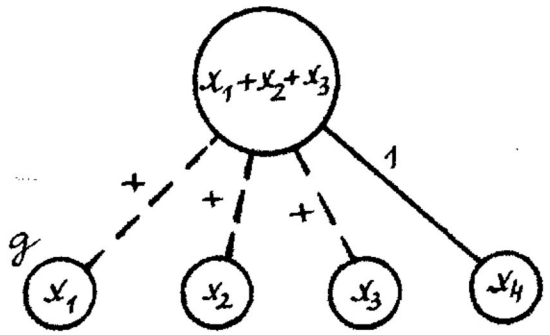


Рис. 7.

В дальнейшем под термином "вершина не имеет логических (соответственно, арифметических) дуг" будем подразумевать, что в эту вершину не входят и из этой вершины не выходят логические (соответственно, арифметические) дуги. Вершину назовем логически изолированной (соответственно, арифметически изолированной) если она не имеет логических (соответственно, арифметических) дуг. Вершину назовем изолированной, если она является логически и арифметически изолированной.

Под <u>путем в графе $G(EN(\alpha))$</u> будем понимать путь, составленный только из логических дуг. Вершины x и y назовем <u>связанными</u>, если их соединяет некоторый путь в $G(EN(\alpha))$. Под <u>весом пути в $G(EN(\alpha))$</u> будем понимать сумму весов дуг составляющих этот путь.

Для программы ПРИМЕР-2 система EN (I,2,3,4,5,6,7,8,9, I0,I3,20,2I,22,I,2,3,4,5,6,7,8,9,I0,II,I,2,3,4,5,6,7,8,9,I0, I3,I4,I5,I6,I7,I8,I9,20) = $\{ a1=a_9,\ a2=a_{10},\ a3=a_{11},\ a4=a_{12},\ b=*,\ c=a_1,\ d=a_2+a_6,\ e=*,\ f=a_3,\ a_1+a_2+a_3 \geq a_4,\ a_1+a_2+a_3 \leq a_4,\ a_1 \geq 1,\ 0 < a_1,\ 0 < 1,\ a_5+a_6+a_7 \geq a_8,\ a_5+a_6+a_7 \leq a_8,\ a_5 \geq 1,\ a_1 \geq a_5,\ a_9+a_{10}+a_{11} \geq a_{12},\ a_9+a_{10}+a_{11} \leq a_{12},\ a_9 \geq 1,\ a_1 < a_9,\ a_2+a_6 \geq 1 \}$. изображается графом, представленным на рис.8.

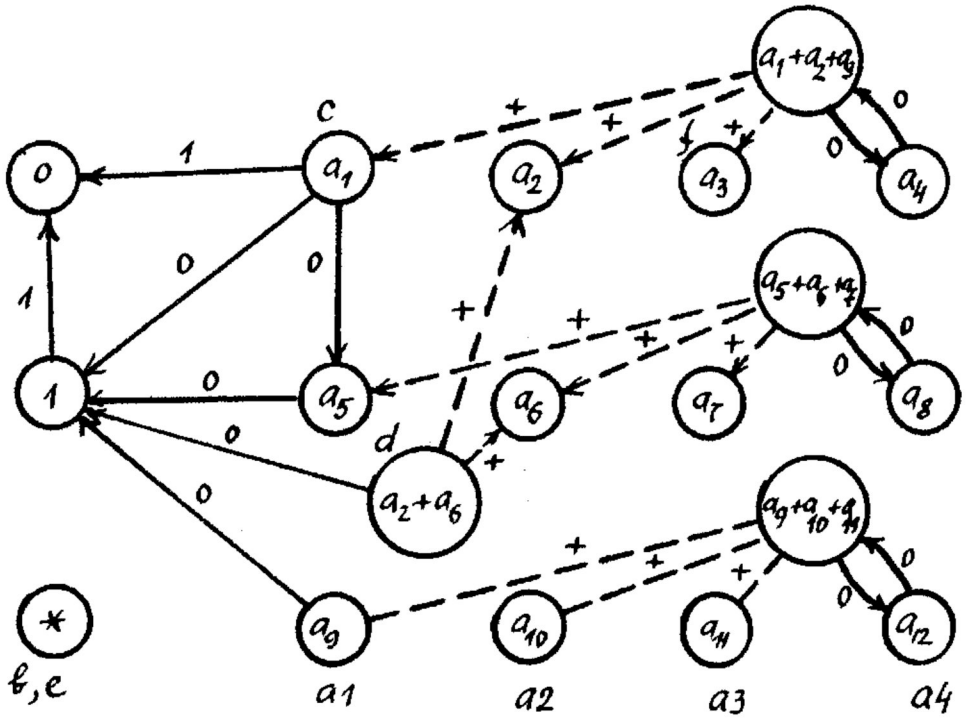
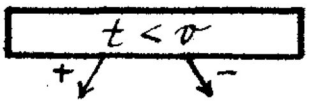


Рис.8.

Переходим к определению активных вершин графа $G(EN(\alpha))$. Содержательно, активными будем считать те вершины, которые изображают последние значения внутренних переменных и на которые в дальнейшем могут быть наложены новые арифметические или логические связи.

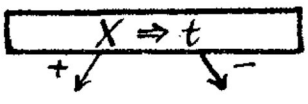
Пусть n_0 - произвольная команда программы и пусть $\beta = (n_0, n_1, \dots, n_j)$ - произвольный путь этой программы, начинающийся командой n_0. Введем понятие множества существенных переменных перед командой n_0 относительно пути β. Это множество обозначим через $H(n_0, \beta)$. Определение индуктивное. Множество существенных переменных в конце пути β, т.е. перед командой n_j - пустое ($H(n_j, \beta) = \varnothing$). Далее, просматриваем команды пути β в обратном направлении начиная с конца пути β т.е. сперва рассматриваем команду n_{j-1}, затем n_{j-2}, затем n_{j-3} и т.д., до команды n_0 включительно. В зависимости от вида рассматриваемой команды n_i $(j-1 \geq i \geq 0)$ и множества существенных переменных $H(n_{i+1}, \beta)$ перед предыдущей командой n_{i+1} строим множество существенных переменных $H(n_i, \beta)$ перед командой n_i следующим образом.

I) Пусть командой n_i является команда



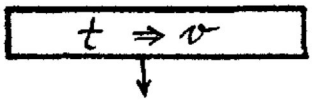
Тогда $H(n_i, \beta) = H(n_{i+1}, \beta) \cup \{t, v\}$. Если t и/или v - константа, то она в $H(n_i, \beta)$ не включается.

2) Пусть командой n_i является команда



Тогда $H(n_i, \beta) = H(n_{i+1}, \beta) \setminus \{t\}$.

3) Пусть командой n_i является команда



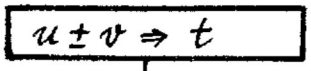
Если внутренняя переменная v не содержится в $H(n_{i+1},\beta)$, то $H(n_i,\beta)=H(n_{i+1},\beta)$. Если $v \in H(n_{i+1},\beta)$, тогда $H(n_i,\beta)$ строим следующим образом:

- если v - счетчик или переменная, то она исключается из $H(n_{i+1},\beta)$; если v - элемент массива, то он из $H(n_{i+1},\beta)$ не исключается;
- если t - переменная, то она включается в $H(n_i,\beta)$, если t - константа, то она в это множество не включается, а если t - элемент массива, то в $H(n_i,\beta)$ включаются все элементы этого массива и кроме этого, если элемент массива задается в виде

  <имя массива> (<имя счетчика>),

  то в это множество включается также указанный счетчик.

4) Пусть командой n_i является команда



Если $t \in H(n_{i+1},\beta)$, то $H(n_i,\beta) = H(n_{i+1},\beta) \cup \{u,v\} \setminus \{t\}$. Если u и/или v - константа, то она в $H(n_i,\beta)$ не включается. Если $t \notin H(n_{i+1},\beta)$, то $H(n_i,\beta) = H(n_{i+1},\beta)$.

При остальных командах множество существенных переменных не меняется.

Внутреннюю переменную будем называть существенной у команды n_0, если она содержится в множестве существенных переменных относительно какого-либо пути, начинающегося командой n_0. Вершину x графа $G(EN(\alpha))$ назовем активной, если к ней в виде дополнительной метки приписана внутренняя перемен-

ная, которая является существенной y последней команды пути α. Вершины графа $G(EN(\alpha))$, которые не являются активными, назовем пассивными.

Нетрудно построить алгоритм нахождения активных вершин графа $G(EN(\alpha))$. Работа этого алгоритма заключается в некотором конечном переборе продолжений пути α и в построении для каждого продолжения соответствующего множества существенных переменных.

У команды I программы ПРИМЕР-2 множество существенных переменных имеет вид $\{c,d\}$ у команды 5 - $\{a1,a2,a3,a4,c,d\}$, а у команды 20 $\{a1,a2\}$ следовательно, в графе, изображенном на рис.8, активными являются вершины a_9 и a_{10}.

Из приведенных конструкций следует:

СВОЙСТВО 3.I. Пусть α - произвольный путь программы и пусть x - произвольная пассивная вершина графа $G(EN(\alpha))$. Тогда для любого продолжения β пути α, если в графе $G(EN(\alpha+\beta))$ содержится вершина x то она имеет в точности те же самые арифметические и логические дуги (входящие и выходящие), которые имеет вершина x в графе $G(EN(\alpha))$.

Перед изложением алгоритма построения состояния докажем еще одну лемму. Пусть граф $G(EN(\alpha))$ содержит арифметическую вершину κ Граф, полученный после стирания арифметических дуг, выходящих из вершины κ, назовем κ-упрощенным.

В дальнейшем мы часто будем строить решение системы $N(\alpha)$ (и ей подобных систем) исходя из $G(N(\alpha))$. Для этого мы припи-

шем к вершинам этого графа значения, которые будут служить решением системы $N(\alpha)$. В этом случае будем говорить, что решение имеет также граф $G(N(\alpha))$.

Теперь рассмотрим один частный случай, когда при проверке существования решения для $G(N(\alpha))$ можно отбрасывать (не принимать во внимание) арифметические дуги выходящие из некоторой вершины κ

Вершину x графа $G(N(\alpha))$ назовем <u>свободной</u>, если она не связана с числовыми вершинами и любая другая вершина y, которая связана с x, не имеет арифметических дуг (сама вершина x может иметь арифметические дуги). Будем говорить, что арифметическая вершина κ имеет разрешающую вершину x, если вершина x является свободной и в вершину x входит только одна арифметическая дуга, притом из вершины κ. Например, в графе, изображенном на рис.8, свободными являются вершины $a_1+a_2+a_3$, a_3, $a_5+a_6+a_7$ и др., а вершины a_2+a_6, a_1, a_2 не являются свободными; вершина $a_1+a_2+a_3$ имеет разрешающую вершину a_3.

Арифметическую вершину κ графа $G(N(\alpha))$ назовем вершиной с <u>разрешимыми связями</u>, если выполнено хотя бы одно из условий:

- вершина κ является свободной,
- вершина κ имеет разрешающую вершину.

Смысл вершин с разрешимыми связями выражает следующая лемма:

ЛЕММА 3.2. Пусть граф $G(N(\alpha))$ содержит вершину κ с разрешимыми связями. Тогда граф $G(N(\alpha))$ имеет решение тогда и только тогда, когда решение имеет κ -упрощенный граф.

Необходимость очевидна. Достаточность. Пусть $G(N(\alpha))$ содержит арифметическую вершину κ, которая является свободной, и пусть вершинам, в которые ведут арифметические дуги из κ уже присвоены значения. Тогда вершине κ присвоим значение выражения κ, если в κ вместо неизвестных поставить им присвоенные значения. Вершинам связанным с κ значения присваиваются процедурой, описанной в доказательстве леммы I.2. Случай, когда κ имеет разрешающую вершину, анализируется аналогично.

Переходим к изложению алгоритма S преобразования системы $EN(\alpha)$ для построения состояния. В начале изобразим систему $EN(\alpha)$ в виде графа $G(EN(\alpha))$ и отметим в этом графе активные и пассивные вершины (дополнительные метки оставляем лишь у активных вершин). Далее проводим следующие преобразования в таком порядке, в каком они ниже изложены.

Преобразование I. Исключение пассивных частей. Стираем в $G(EN(\alpha))$ подграфы, все вершины которых пассивны и не связаны ни логическими, ни арифметическими дугами с другими вершинами графа $G(EN(\alpha))$ не содержащимся в рассматриваемых подграфах. В частности, если у команды, в которую ведет путь α не имеется существенных переменных, то все вершины графа $G(EN(\alpha))$ пассивны, и, следовательно, мы стираем весь граф $G(EN(\alpha))$. В этом случае результатом работы алгоритма S является пустая система неравенств (пустое состояние).

Преобразование 2. Отбрасывание арифметических дуг. Поочередно просматриваем арифметические вершины и стираем арифметические дуги, выходящие из очередной вершины $\varkappa$ если выполнено хотя бы одно из условий:

- вершина $\varkappa$ является свободной, пассивной и все вершины, с которыми логическими дугами связана вершина $\varkappa$ являются пассивными;
- вершина $\varkappa$ имеет пассивную разрешающую вершину x, притом все вершины, которые связаны логическими дугами с вершиной x являются пассивными.

Содержательно, этими преобразованиями из $EN(\alpha)$ исключаются те выражения, значения которых неравенствами из $N(\alpha)$ вообще не ограничиваются или которым можно присвоить произвольное значение благодаря тому, что одна из записей этого выражения опять таки неравенствами из $N(\alpha)$ вообще не ограничивается.

Преобразование 3. Замена значений односторонних счетчиков. (Напомним, что односторонними называются счетчики, к которым в данной программе только прибавляются (соответственно, вычитаются) положительные константы и которые можно сравнивать только с константами.) Пусть значение одностороннего счетчика S в программе может только увеличиваться (соответственно, уменьшаться) и пусть это значение после пути α больше числа $c_1 = max(\tilde{c}_1, \tilde{c}_2)$, где $\tilde{c}_1$ - максимальная константа, с которой в программе сравнивается счетчик S и $\tilde{c}_2$ - максимальное число элементов в массиве, для указания элементов которого используется счетчик S (соответственно, меньше числа $c_2 = min(\underset{\sim}{c}_1, 0)$, где $\underset{\sim}{c}_1$ - минимальная константа, с ко-

торой в программе сравнивается счетчик S). Тогда значение счетчика S заменяем на число c_1+1 (соответственно, c_2-1). Формально это означает: стираем в графе дополнительную метку S и строим числовую вершину c_1+1 (если такая еще не имеется) и приписываем к ней дополнительную метку S .

Преобразование 4. Исключение числовых вершин. Стираем дуги, соединяющие числовые вершины, а также изолированные пассивные числовые вершины.

Преобразование 5. Исключение записей, которые ограничены только сверху или только снизу. Поочередно находим в $G(EN(\alpha))$ пассивные арифметически изолированные вершины, которые изображают записи и в которые или только входят или из которых только выходят логические дуги, и стираем эти вершины вместе со всеми дугами, входящими или выходящими из них. Если в результате этого образуются пассивные изолированные вершины, то стираем также и эти вершины. Содержательно, этими преобразованиями из $G(EN(\alpha))$ исключаются те пассивные записи, которые не содержатся в арифметических выражениях из системы $N(\alpha)$ и значения которых с учетом транзитивности отношений "$>$" и "$\geqslant$" неравенствами из $N(\alpha)$ ограничиваются только с одной стороны (сверху или снизу).

Преобразование 6. Исключение записей с учетом транзитивности отношений "$>$" и "$\geqslant$". Поочередно исключаем пассивные арифметически изолированные вершины, которые изображают записи; если в исключаемую вершину x входят логические дуги d_i с весами p_i и выходят дуги e_j с весами q_j то любую пару дуг (d_i, e_j) заменяем на новую дугу f_k с весом $r_k=p_i+q_j$, а

вершину x стираем. Если после этого две вершины соединяют параллельные дуги, то дуги с меньшим весом стираем.

Преобразование 7. Уменьшение весов дуг. Преобразования 7 проводим только в том случае, когда в данной программе не имеется команд сложных вычислений, сложения и вычитания. Тогда веса дуг, превышающих число $d = \bar{c} - \underline{c} + 1$, где $\bar{c}$ - максимальная, а $\underline{c}$ - минимальная константа программы, заменяем на новый вес равный числу d (если программа не содержит констант, то $d = 1$).

Графу $\tilde{G}(EN(\alpha))$, полученному после указанных выше преобразований, сопоставим систему $S(\alpha)$, называемую сокращенной, аналогично как в главе I: Если в $G(EN(\alpha))$ содержится дуга с весом m ведущая из вершины x в вершину y, то в $S(\alpha)$ включаем неравенство $x \geq y + m$. Если в $\tilde{G}(EN(\alpha))$ содержится активная вершина x с дополнительной меткой g, то в $S(\alpha)$ включим равенство $g = x$. Система $S(\alpha)$ является результатом работы алгоритма S.

На рис.9 представлен граф, полученный указанными выше преобразованиями графа из рис.8. Соответствующая сокращенная система имеет вид $\{ a1 = a_9, a2 = a_{10}, a_9 \geq 2 \}$.

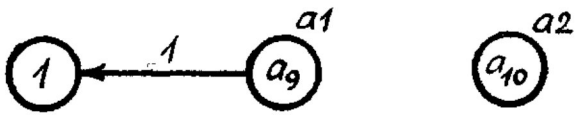


Рис. 9.

Теперь переходим к доказательству того, что алгоритм S является методом построения состояний для промежуточного языка.

Пусть β - произвольное продолжение пути α. Напомним, что система $N_\alpha(\beta)$, содержащая неравенства, добавляемые на пути β к $N(\alpha)$ при построении $N(\alpha+\beta)$ строится исходя только из $E(\alpha)$ и команд пути β. Кроме этого при построении $N_\alpha(\beta)$ используются значения тех переменных, которые являются существенными после пути α, Теперь рассмотрим систему неравенств, получаемую той же процедурой, что $N_\alpha(\beta)$ только в качестве значений внутренних переменных выбираем не значения из $E(\alpha)$, а из $S(\alpha)$ Эту систему обозначим через $N_{S(\alpha)}(\beta)$. (В $S(\alpha)$ значения задаются только для тех переменных, которые являются существенными после пути α, и эти значения могут отличаться от значений в $E(\alpha)$ только у односторонних счетчиков.) Имеет место:

СВОЙСТВО 3.2. Пусть α_1 и α_2 - пути, оканчивающиеся одной и той же командой программы, и пусть $S(\alpha_1) = S(\alpha_2)$. Тогда для любого продолжения β путей α_1 и α_2 системы $S(\alpha_1) + N_{S(\alpha_1)}(\beta)$ и $S(\alpha_2) + N_{S(\alpha_2)}(\beta)$ равны.

Свойство 3.2 доказывается аналогично как в § I.3 свойство I.I.

ЛЕММА 3.3. Для любого простого реализуемого пути α и его простого продолжения β система неравенств $N(\alpha+\beta)$ имеет решение тогда и только тогда, когда решение имеет система $S(\alpha) + N_{S(\alpha)}(\beta)$.

Доказательство. Мы должны убедиться, что граф $G(N(\alpha+\beta))$ имеет решение тогда и только тогда, когда решение имеет граф $G(S(\alpha)+N_{S(\alpha)}(\beta))$.

Во первых убедимся, что граф $G(N(\alpha+\beta))$ имеет решение тогда и только тогда, когда решение имеет граф G_1, получаемый из $G(N(\alpha+\beta))$ проведением преобразований I в подграфе $G(N(\alpha))$. Действительно, вершины подграфов из $G(N(\alpha))$ стираемые преобразованиями I, никак не связаны с другими вершинами графа $G(N(\alpha))$. Поэтому решение $G(N(\alpha))$ сводится к решению отдельных его частей. Вниду того, что в стираемых подграфах все вершины являются пассивными, они не связаны с другими вершинами также в графе $G(N(\alpha+\beta))$. Теперь учитывая, что α - реализуемый путь, получаем, что стираемые преобразованиями I подграфы всегда имеют решение. Следовательно, G_1 имеет решение тогда и только тогда, когда решение имеет $G(N(\alpha+\beta))$.

Во вторых, нетрудно убедиться, что если преобразованиями 2 в $G(N(\alpha))$ стираются арифметические дуги, выходящие из некоторой вершины κ, то в графе $G(N(\alpha+\beta))$ она является вершиной с разрешимыми связями. Применяя лемму 3.I получаем, что граф G_1 имеет решение тогда и только тогда, когда решение имеет его κ - упрощенный граф. Аналогичными рассуждениями для всех вершин, к которым применены преобразования 2, мы получаем, что граф G_1 имеет решение тогда и только тогда, когда решение имеет граф G_2, получаемый из $G(N(\alpha+\beta))$ проведением преобразований I и 2 в подграфе $G(N(\alpha))$.

В третьих, рассмотрим преобразования 3. Пусть путь β содержит команду условного перехода, которая сравнивает значение

$\tilde{c}$ одностороннего счетчика S с константой c . Тогда в $N(\alpha+\beta)$ содержится одно из неравенств $\tilde{c}<c$, $\tilde{c}\leq c$, $\tilde{c}>c$, $\tilde{c}\geq c$. Будем считать, что значение $\tilde{c}$ счетчика S преобразованиями 3 заменено на c_m+1 , где $c_m<\tilde{c}$ и $c_m\geq c$. Тогда в $N_{S(\alpha)}(\beta)$ будет содержаться неравенство $c_m+1<c$ (соответственно, $c_m+1\leq c$, $c_m+1>c$, $c_m+1\geq c$), которое с точки зрения существования решения равносильно $\tilde{c}<c$ (соответственно, $\tilde{c}\leq c$, $\tilde{c}>c$, $\tilde{c}\geq c$).

Далее рассмотрим преобразования 4. Пусть $G(N(\alpha))$ содержит две числовые вершины c_1 и c_2 Допустим, что $c_1\geq c_2$ и из вершины c_1 в вершину c_2 ведет некоторая дуга. Из того, что α - реализуемый путь и из условия 2 леммы I.2 получаем, что вес этой дуги не превышает разницу c_1-c_2. Поэтому отсутствие этой дуги в $G(N(\alpha))$ не влияет на существование решения.

Таким образом мы убедились, что граф $G(N(\alpha+\beta))$ имеет решение тогда и только тогда, когда решение имеет граф G_4 получаемый из $G(N(\alpha+\beta))$ проведением преобразований I,2,3,4 в подграфе $G(N(\alpha))$. Мы должны убедиться, что граф G_4 имеет решение тогда и только тогда, когда решение имеет граф $G(S(\alpha)+N_{S(\alpha)}(\beta))$.

Необходимость. Пусть граф G_4 имеет решение. Мы должны построить решение для $G(S(\alpha)+N_{S(\alpha)}(\beta))$. Изолированным вершинам значения присвоим произвольно. Пусть x - произвольная вершина из $G(S(\alpha)+N_{S(\alpha)}(\beta))$, которая не является изолированной. Тогда в G_4 также существует вершина x и пусть при решении G_4 этой вершине присвоено значение a. Присвоим вершине x в $G(S(\alpha)+N_{S(\alpha)}(\beta))$ такое же значение a как

вершине x в G_4. Аналогично как в доказательстве леммы I.3 можно убедиться, что преобразованиями 5,6,7 максимальный вес путей, соединяющих вершину x с любой другой вершиной y графа $G(S(\alpha)+N_{S(\alpha)}(\beta))$ не больше максимального веса путей, соединяющего вершину x с вершиной y в G_4. С другой стороны преобразованиями 5 и 6 не стираются вершины, имеющие арифметические дуги, а также не меняются максимальные веса путей, соединяющих арифметические вершины. Кроме этого преобразования 7 не проводятся, если в G_4 имеется арифметические дуги. Поэтому значение a присвоенное вершине x удовлетворяет $S(\alpha)+N_{S(\alpha)}(\beta)$.

Достаточность. Пусть граф $G(S(\alpha)+N_{S(\alpha)}(\beta))$ имеет решение. Мы должны построить решение для G_4.

Во первых, если $G(S(\alpha)+N_{S(\alpha)}(\beta))$ не содержит арифметических дуг, то арифметические дуги не содержатся также в G_4 и, следозательно, G_4 изображает базисную систему неравенств. В этом случае существование решения для G_4 доказывается аналогично как в лемме I.3.

В случае, когда $G(S(\alpha)+N_{S(\alpha)}(\beta))$ содержит арифметические дуги, преобразованиями 7 веса дуг не уменьшаются, и, следовательно, максимальный вес путей в $G(S(\alpha)+N_{S(\alpha)}(\beta))$, соединяющих произвольные вершины x и y совпадает с максимальным весом путей, соединяющих эти вершины в G_4. Кроме этого все арифметические дуги, содержащиеся в $G(S(\alpha)+N_{S(\alpha)}(\beta))$, содержатся также в G_4 и наоборот. Ясно, что если для $G(S(\alpha)+N_{S(\alpha)}(\beta))$ существует решение, то эти же значения будут служить решением для G_4 а вершинам исключаемым преобра-

зованиями 5 и 6, значения можно построить процедурой описанной в доказательстве леммы I.2. Лемма 3.3 доказана.

Теперь из лемм 3.2 и 3.3 непосредственно вытекает:

ЛЕММА 3.4. Алгоритм S является методом построения состояний для промежуточного языка.

3.3. Алгоритм построения ПСП для программ в промежуточном языке.

Пусть P - произвольная программа в промежуточном языке. Аналогично как в первой главе строим дерево реализуемости $D(P)$ для программы P. Опишем сразу шаг индукции. Пусть уже построено k ярусов дерева и пусть q_k - произвольная вершина k-того яруса, которой в программе P соответствует команда с номером $n(q_k)$. Строим вершины (k+1)-го яруса: из вершины q_k проводим столько дуг, сколько выходов имеет команда $n(q_k)$. В концах этих дуг строим вершины (k+1)-го яруса, к которым приписываем номера соответствующих команд.

Далее, просматриваем ветвь $h=(q_1,q_2,\dots,q_k,q_{k+1})$ построенного дерева, на которой лежит вершина q_k и обрежем дугу $(n(q_k),n(q_{k+1}))$ в следующих случаях:

1) Если путь $\alpha(q_k)=(n(q_1),n(q_2),\dots,n(q_k))$ реализуем, но путь, полученный из $\alpha(q_k)$ добавлением дуги $(n(q_k),n(q_{k+1}))$, нереализуем. Эти ветви будем называть обрезанными по нереализуемости.

2) Если $n(q_k)$ - номер команды СТОП (тривиальное обрезание). Эти ветви назовем СТОП-ветвями.

3) Если система неравенств $N(\alpha(q_{k+1}))$ пути $\alpha(q_{k+1})$ не является простой, т.е. если $N(\alpha(q_{k+1}))$ содержит символ "*". Эти ветви назовем обрезанными по деформированию.

4) Если 1), 2) и 3) не имеет место, то к вершине q_k приписываем состояние $S(\alpha(q_k))$. Далее, просматриваем вершины, лежащие на той же ветви h дерева, на которой находится вер-

шина q_k и обрезаем дугу $(n(q_k), n(q_{k+1}))$, если среди этих вершин имеется вершина q_i $(0 \le i < k)$ такая, что $n(q_i) = n(q_k)$ и $S(\alpha(q_i)) = S(\alpha(q_k))$. В этом случае ветвь h будем называть обрезанной по повторению, а вершину q_i будем называть обрезающей для ветви h.

Из леммы 3.I и построения состояний видно, что процесс обрезания дуг - эффективный. После обрезания дуг, выходящих из вершин k-того яруса, переходим к построению вершин (к+2)-го яруса и обрезания дуг, выходящих из вершин (к+I)-го яруса. В общем случае эта процедура может длиться неограниченно долго. В этом случае описанный нами алгоритм не применим к программе P. Если же эта процедура завершается (что неизбежно, если программа P имеет конечное число состояний), то построенное конечное дерево будем называть <u>деревом реализуемости</u>.

Аналогично леммы I.5 доказывается

<u>ЛЕММА 3.5.</u> Пусть P - произвольная программа в промежуточном языке и пусть описанный выше алгоритм строит для этой программы конечное дерево реализуемости $D(P)$. Тогда произвольный простой путь программы P реализуем тогда и только тогда, когда этот путь согласуется с деревом реализуемости $D(P)$.

Теперь рассмотрим алгоритм Ω, который произвольной программе P будет сопоставлять некоторую систему примеров, обозначаемую через $\mathcal{M}$.

I) Строим дерево реализуемости для программы P. Если

эта процедура завершается, то переходим к 2). Иначе алгоритм Ω для программы P будет работать неограниченно долго, и результат его работы будем считать неопределенным.

2) Строим покрывающее множество $Q(P)=\{\alpha_1,\alpha_2,\ldots,\alpha_m\}$ согласно алгоритму описанному в § I.4.

3) Составляем для путей α_i $(i=1,2,\ldots,m)$ системы неравенств $N(\alpha_i)$.

4) Решаем системы неравенств $N(\alpha_i)$ $(i=1,2,\ldots,m)$. Согласно леммы 3.I решение системы неравенств $N(\alpha_i)$ задает пример T_i который реализует путь α_i. Полученная система примеров $\mathcal{M}(P)=\{T_1,T_2,\ldots,T_m\}$ является результатом работы алгоритма Ω.

Из доказанных лемм вытекает:

ТЕОРЕМА 3.I. Пусть P - произвольная программа в промежуточном языке и пусть описанный выше алгоритм Ω для этой программы строит конечное дерево реализуемости $D(P)$. Если $D(P)$ не содержит обрезанных по деформированию ветвей, то система примеров $\mathcal{M}(P)$ является полной для программы P.

Теперь дадим некоторые характеристики класса программ, для которых алгорити Ω строит ПСП. Из теоремы 3.I непосредственно вытекает

СЛЕДСТВИЕ 3.I. Пусть P - программа в промежуточном языке, которая не содержит команд сложных вычислений, и пусть все команды сложения-вычитания этой программы применяются

только к односторонним счетчикам. Тогда алгоритм Ω для программы P строит ПСП.

Будем говорить, что в команде условного перехода сравниваются простые переменные, если на любом пути, ведущем в эту команду, переменным, сравниваемым в условии этой команды, значения присваиваются только командой чтения записей. Имеет место

СЛЕДСТВИЕ 3.2. Алгоритм Ω строит ПСП для любой программы, в которой используются только односторонние счетчики и во всех командах условного перехода сравниваются только простые переменные.

Справедливость этого утверждения следует из того, что система неравенств любого пути программы с выше указанными свойствами является базисной. Следовательно, количество состояний программы - конечное.

На самом деле имеет место более сильное утверждение, а именно, переменным из команд условного перехода можно присваивать значения также командой сложения-вычитания, если сумма или хотя бы один из слагаемых не сравнивается с константами (с учетом транзитивности отношений "$>$" и "$\geqslant$"). Отметим, что в реальных языках программирования области значений переменных, которые с теоретической точки зрения всегда ограничены, практически все-таки бесконечны. Поэтому ограничения на области допустимых значений переменных при построении ПСП для реальных программ мы не будем рассматривать как сравнение с кон-

стантами. Это обусловлено тем, что если решение имеет система неравенств некоторого пути программы, записанной в реальном языке программирования, то обычно эта система неравенств имеет такое решение, которое содержится в области значений переменных рассматриваемой программы.

Обычные программы вывода табуляграмм, слияния файлов, ввода и контроля данных удовлетворяют указанным выше требованиям, и, следовательно, алгорити Ω с некоторыми модификациями, изложенными в следующей главе, применим для построения ПСП для этих программ.

Далее рассмотрим две рационализации алгоритма Ω, используемые при его практической реализации. Первая из них состоит в рациональном построении дерева реализуемости. Нетрудно заметить, что состояния достаточно строить лишь у одной команды из каждого цикла программы. Точнее, множество команд программы назовем множеством существенно расположенных команд, если любой путь программы, начинающийся и оканчивающийся одной и той же командой, содержит по крайней мере одну команду из множества существенно расположенных команд. Команды из этого множества назовем существенно расположенными (СРК). Нетрудно убедиться, что при построении дерева реализуемости состояния достаточно приписывать лишь к вершинам, соответствующим СРК. Притом нет необходимости обрезать ветви дерева реализуемости по повторению после построения каждого этажа, а только при достижении вершин, соответствующих СРК.

При практической реализации используется алгорити, который

строит множество существенно расположенных команд по принципу: каждый циклический путь программы содержит одну СРК. Этот алгоритм не гарантирует построение множества существенно расположенных команд с минимальным числом команд, однако он оказывается достаточно быстродействующим.

Вторая рационализация алгоритма Ω состоит в уменьшении количества путей в покрывающем множестве. Это необходимо потому, что на практике неудобно работать с большим количеством примеров.

Систему примеров $\mathcal{M}_m$ назовем <u>минимальной полной системой примеров</u> (по количеству примеров) для программы P, если она является полной для программы P и любая другая полная система примеров содержит не меньше примеров чем $\mathcal{M}_m$. Фактически существует алгоритм построения минимальной полной системы примеров (по количеству примеров) для любой программы, которая удовлетворяет условиям теоремы 3.I. Однако этот алгоритм является переборным. Поэтому в практической реализации мы будем пользоваться другим методом, который не всегда дает минимальную полную систему примеров. Из описанного выше покрывающего множества $Q(P)$ строится новое покрывающее множество путей $\bar{Q}(P)$ следующим образом: первым в множество $\bar{Q}(P)$ включается тот путь из $Q(P)$, который содержит максимальное количество различных дуг программы, вторым в $\bar{Q}(P)$ включается путь, содержащий максимальное количество оставшихся различных дуг программы, т.е. дуг, не содержащихся в первом пути, и т.д. Эту процедуру продолжаем до момента, когда в $\bar{Q}(P)$ включены все дуги программы, содержащиеся в $Q(P)$.

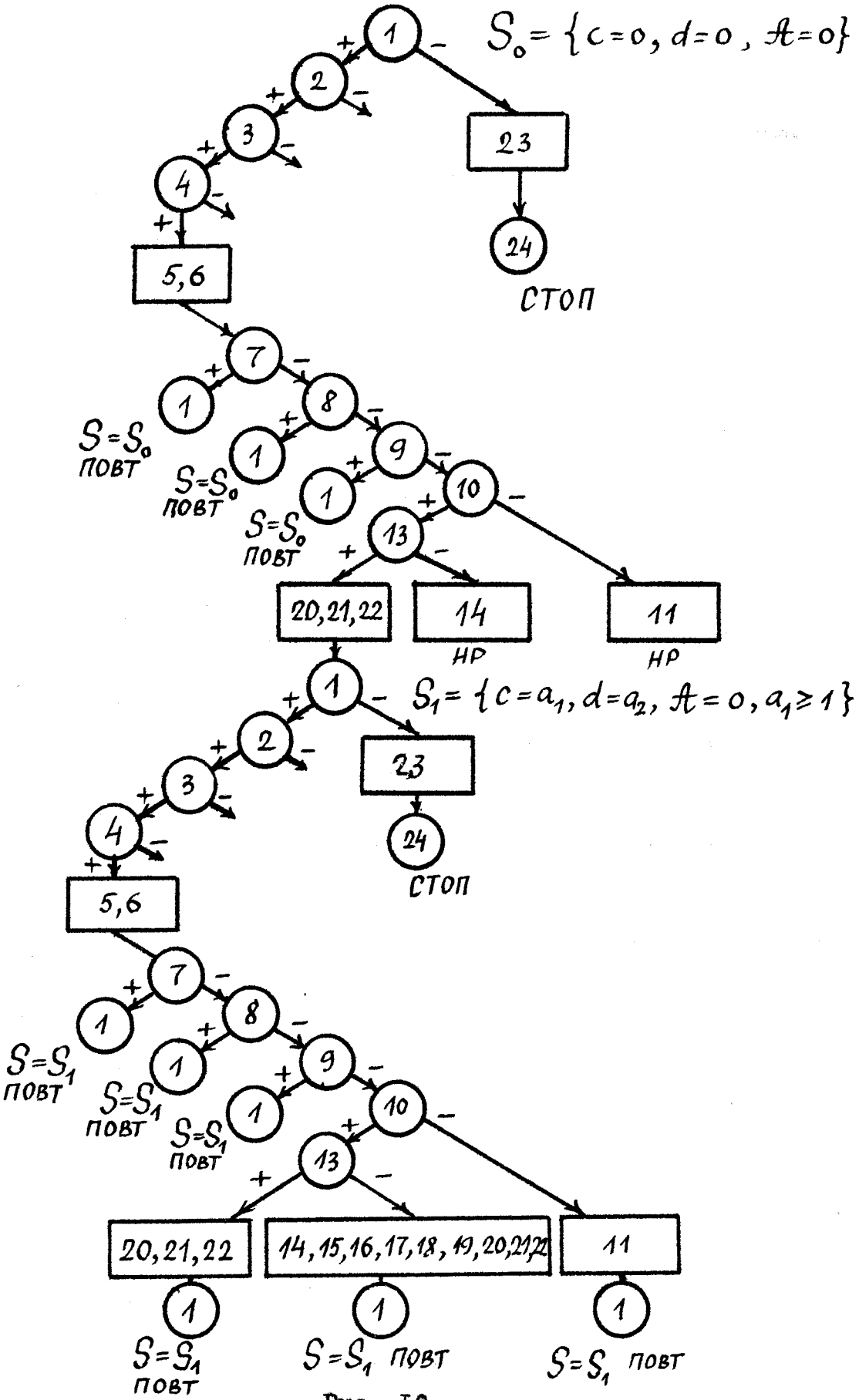

$S_0 = \{c=0, d=0, \mathcal{A}=0\}$
1
2
3
4
5,6
7
8
9
10
13
23
24
СТОП
20,21,22
14
НР
11
НР
$S=S_0$ ПОВТ
$S=S_1$ ПОВТ
$S_1 = \{c=a_1, d=a_2, \mathcal{A}=0, a_1 \geq 1\}$
14,15,16,17,18,19,20,21,22

Рис. 10.

$\beta_1 = (1,2,3,4,5,6,7,8,9,10,13,20,21,22,1)$.

$EN(\beta_1) = \{a1 = a_1,\ a2 = a_2,\ a3 = a_3,\ a4 = a_4,\ b = a_1+a_2+a_3,\ c = a_1,\ d = a_2,$
$e = 0,\ f = a_3,\ h = 0,\ a_1+a_2+a_3 \geq a_4,\ a_1+a_2+a_3 \leq a_4,\ a_1 \geq 1,\ 0 < a_1,\ 0 < 1\}$.

а) $G(EN(\beta_1))$ б) $\widetilde{G}(EN(\beta_1))$

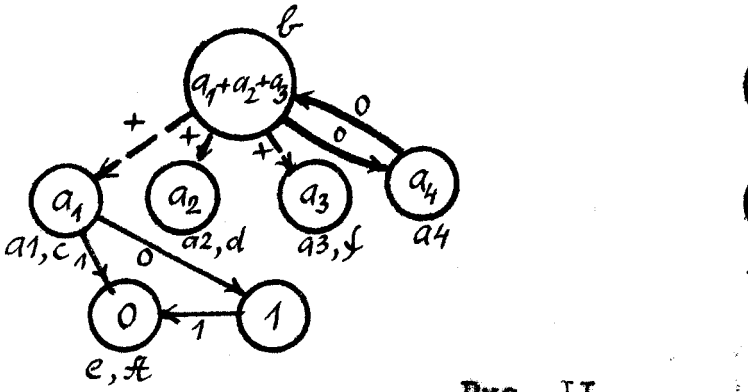


Рис. II.

$\beta_2 = (1,2,3,4,5,6,7,8,9,10,13,20,21,22,1,2,3,4,5,6,7,8,9,10,$
$13,14,15,16,17,18,19,20,21,22,1)$

$EN(\beta_2) = \{a1 = a_5,\ a2 = a_6,\ a3 = a_7,\ a4 = a_8,\ b = *,\ c = a_5,\ d = a_6,\ e = * + 0,$
$f = a_7,\ h = 0,\ a_1+a_2+a_3 \geq a_4,\ a_1+a_2+a_3 \leq a_4,\ a_1 \geq 1,\ 0 < a_1,\ 0 < 1,$
$a_5+a_6+a_7 \geq a_8,\ a_5+a_6+a_7 \leq a_8,\ a_5 \geq 1,\ a_1 < a_5,\ a_2 \geq 1\}$.

а) $G(EN(\beta_2))$ б) $\widetilde{G}(EN(\beta_2))$

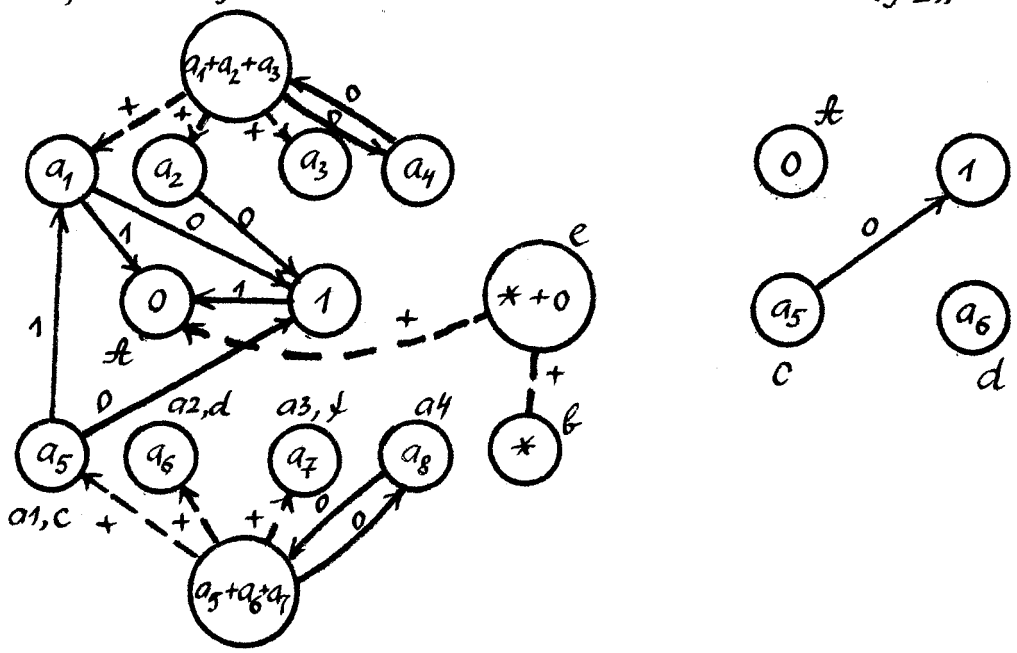


Рис. I2.

В заключении рассмотрим программу ПРИМЕР-2 и согласно алгоритму Ω с учетом указанных выше рационализаций построим для нее ПСП. Во первых, множество существенно расположенных команд состоит из одной команды с номером I. Существенными переменными у этой команды являются переменные c и d. Дерево реализуемости программы ПРИМЕР-2 представлено на рис.I0. В концах ветвей, обрезанных по нереализуемости, приписано "НР", в концах СТОП-ветвей - "СТОП", а в концах ветвей, обрезанных по повторению - "ПОВТ". Номера команд, имеющих один выход, записаны в одну вершину и обведены прямыми линиями. Системы неравенств, состояния и соответствующие им графы приведены только для двух путей и представлены на рис.II и рис.I2.

Покрывающее множество имеет вид Q (ПРИМЕР-2) =

$\{\alpha_1 = (1,2,3,4,5,6,7,8,9,10,13,20,21,22,1,23,24),$

$\alpha_2 = (1,23,24),$

$\alpha_3 = (1,2,3,4,5,6,7,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,9,1,2,3,4,$
$5,6,7,8,9,10,13,20,21,22,1,23,24),$

$\alpha_4 = (1,2,3,4,5,6,7,8,9,10,13,20,21,22,1,2,3,4,5,6,7,1,2,3,4,$
$5,6,7,8,1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,10,13,20,21,$
$22,1,2,3,4,5,6,7,8,9,10,13,14,15,16,17,18,19,20,21,22,$
$1,2,3,4,5,6,7,8,9,10,11,1,23,24)\}.$

При минимизации количества примеров из Q (ПРИМЕР-2) будет выбран только один путь α_4, который содержит все дуги программы ПРИМЕР-2. Система неравенств для пути α_4 имеет вид:

$N(\alpha_4) = \{a_1+a_2+a_3 \geq a_4,\ a_1+a_2+a_3 \leq a_4,\ a_1 \geq 1,\ 0<a_1,\ 0<1,$
$a_5+a_6+a_7 < a_8,\ a_9+a_{10}+a_{11} \geq a_{12},\ a_9+a_{10}+a_{11} > a_{12},\ a_{13}+a_{14}+a_{15} \geq a_{16},$
$a_{13}+a_{14}+a_{15} \leq a_{16},\ a_{13}<1,\ a_{17}+a_{18}+a_{19} \geq a_{20},\ a_{17}+a_{18}+a_{19} \leq a_{20},\ a_{17} \geq 1,$
$a_1 < a_{17},\ a_2<1,\ a_{21}+a_{22}+a_{23} \geq a_{24},\ a_{21}+a_{22}+a_{23} \leq a_{24},\ a_{21} \geq 1,\ a_{17} < a_{21},$
$a_{18} \geq 1,\ a_{25}+a_{26}+a_{27} \geq a_{28},\ a_{25}+a_{26}+a_{27} \leq a_{28},\ a_{25} \geq 1,\ a_{21} \geq a_{25}.$

Решением системы $N(\alpha_4)$ является, например, $a_1=1$, $a_2=0$, $a_3=3$, $a_4=4$, $a_5=1$, $a_6=2$, $a_7=3$, $a_8=7$, $a_9=1$, $a_{10}=2$, $a_{11}=3$, $a_{12}=5$, $a_{13}=0$, $a_{14}=1$, $a_{15}=2$, $a_{16}=3$, $a_{17}=2$, $a_{18}=3$, $a_{19}=4$, $a_{20}=9$, $a_{21}=3$, $a_{22}=1$, $a_{23}=2$, $a_{24}=6$, $a_{25}=3$, $a_{26}=2$, $a_{27}=3$, $a_{28}=8$. Это решение задает пример $T = \{\mathcal{A} = (1,0,3,4;\ 1,2,3,7;\ 1,2,3,5;\ 0,1,2,3;\ 2,3,4,9;\ 3,1,2,6;\ 3,2,3,8)\}$, который реализует путь α_4 (группы записей отделяются точкой с запятой). Результатом работы ПРИМЕР-2 на этом примере является файл $\mathcal{B} = (2,3,4,12;12)$. Система примеров, состоящая всего лишь из одного примера T является полной для программы ПРИМЕР-2. Она показывает, что в программе ПРИМЕР-2 действительно имеется ошибка, которая состоит в том, что по окончании файла $\mathcal{A}$ в файл $\mathcal{B}$ не выводится результат обработки последних групп записей с одинаковым шифром товара. Правильным результатом работы программы должен был быть файл $\mathcal{B} = (2,3,4,12;3,3,2,6;18)$.

Глава 4

ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ПОСТРОЕНИЯ ПСП

4.I. Система автоматического построения ПСП - СМОТЛ.

Первая экспериментальная система автоматического построения ПСП для реальных программ обработки данных (система СМОТЛ) была разработана в I974-I976 годах сотрудниками Вычислительного центра Латвийского госуниверситета им.П.Стучки. Основные алгоритмы были разработаны диссертантом на основе результатов главы 3. При практической реализации кроме автора диссертации участвовали также Борзов Ю.В., Василевский М.П., Страупис У.М. и Зариньш А.К. Общий объем системы - около 30000 команд языка символического кодирования (ЯСК) ЭВМ "Минск-32" (см. /5/).

В качестве входного языка для системы СМОТЛ выбран язык системы макрокоманд обработки данных (СМОД) /22/, по своим возможностям напоминающий КОБОЛ. Выбор этого языка обусловлен чисто технической причиной, а именно тем, что транслятор для ЭВМ "Минск-32" с языка СМОД /23/ разработан также в ВЦ ЛГУ им.П.Стучки (автор диссертации является одним из его разработчиков), и некоторые части этого транслятора можно было использовать в создаваемой системе. Что касается самого языка СМОД, то он более гибкий и более близкий к языку символического кодирования чем КОБОЛ, что, вообще говоря, только усложняло постро-

ение ПСП для СМОД - программ. В целом все-таки можно считать, что алгоритм построения ПСП для языков КОБОЛ и СМОД отличается только техническими деталями. Поэтому вполне возможна разработка аналогичной системы автоматического построения ПСП для КОБОЛ - программ.

В основных чертах система СМОТЛ реализует описанный в главе 3 алгорити. Однако необходимо отметить, что ни один реальный язык программирования полностью не сводим к промежуточному языку. Это обусловлено во первых тем, что в реальных языках обрабатываются не только целые числа, а также дробные числа, слова и т.д. Во вторых команды реальных языков программирования наделены спецификой их реализации. Конечно, можно было рассматривать модель более близкую к некоторому реальному языку программирования нежели промежуточный язык. Однако технические детали, связанные с спецификой конкретного языка, загромождало бы суть дела. Поэтому в диссертации переход от алгоритма построения ПСП для промежуточного языка к алгоритму построения ПСП для СМОД - програми, который связан со спецификой операторов СМОД, более детально рассматривать мы не будем так как с идейной точки зрения это содержит мало нового. Отметим только несколько особенностей.

Во первых, при построении ПСП для СМОД-программ допускается текстовое сравнение данных, которое не предусмотрено в промежуточном языке, но проанализирована в главе 2.

Во вторых, при построении описаний значений переменных и, следовательно, систем неравенств путей и состояний для СМОД-программ возможно возникновение более сложных ситуаций чем в

промежуточном языке: усечение части значения переменной из-за недостаточной длины принимающего поля (значение переменной не помещается в принимающее поле), сравнение одних и тех же данных в числовом и в текстовом отношении, сравнение значений переменных, которые состоят из независимых частей, которые в свою очередь сравниваются между собой и т.п. В этих ситуациях считается, что значения переменных деформированы сложными командами. Однако анализ путей СМОД-програми не прекращается и в зависимости от конкретной ситуации переменным присваиваются константные значения. Путем такой фиксации сложные системы неравенств преобразуются в простые. Ясно, что после этих преобразований в общем случае не гарантируется построение ПСП.

В третьих, описания значений элементов массивов строятся только для первых четырех элементов каждого массива. Это делается для уменьшения памяти при кодировки состояний. В основу этого своеобразного ограничения положен тот эвристический факт, что в реальных программах массивы обычно используются "регулярно" и реализуемость той или другой дуги программы не зависит от количества элементов массивов, а только от их значений. Поэтому в большинстве случаев достаточно следить только за несколько элементами массивов.

Кроме указанных выше трудностей алгоритмического характера, при разработке системы СМОТЛ появились ряд новых проблем связанных с ограниченностью используемых ресурсов времени и памяти.

I) Уменьшение количества запоминаемых состояний. Для этого при построении дерева реализуемости состояния не строятся у

каждой команды, а только у одной команды из каждого цикла программы (см. предыдущую главу). Кроме того, из многих состояний, отличающихся только числовыми значениями переменных или счетчиков, в памяти запоминается лишь одно состояние.

2) Ускорение работы системы. Для этого системы неравенств путей решаются не методом Гомори, который весьма малоэффективен, а "методом интервалов", который вообще не всегда гарантирует нахождение решения. Суть этого метода состоит в том, что для неизвестных решаемой системы строятся интервалы допустимых значений, а решение системы ищется подбором значений из этих интервалов.

3) Частичное построение дерева реализуемости. Необходимо считаться с ситуацией, когда построение дерева реализуемости может быть прекращено до его полного завершения из-за недостатка памяти или времени. При этом желательно, чтобы система все-таки выдавала практически ценные результаты. Для этого дерево реализуемости строится не по этажам, а целенаправленно по ветвям.

Особое место в системе СИОТЛ занимают средства обнаружения так называемых динамических ошибок. Дело в том, что при построении дерева реализуемости проводится проверка реализуемости путей программы. В реальных языках программирования нереализуемость путей программы зависит не только от противоречивости системы неравенств этого пути, но также от ряда других факторов:

I) неверного применения команд ввода-вывода, например, обращения к неоткрытому файлу;

2) обращения к данным без предварительного присвоения им значений;

3) обращения к несуществующему элементу массива;

4) превышения длины поля, отведенного для размещения значения переменной и т.п.

Для выявления этих динамических ошибок необходимо "выполнить" и проверить реализуемость различных путей программы. Однако обычными средствами отладки програм, включая синтаксический анализ программ в трансляторах, это невозможно.

Теперь рассмотрим систему СМОТЛ подробнее. Входной информацией для системы является текст программы, а результатом работы - список ошибок и предупреждений, примеры и протоколы работы программы на автоматически построенных примерах. СМОТЛ состоит из управляющего и из IO независимых блоков: ПСП0I, ПСП02, ПСП03, ПСП04, ПСП05, ПСП06, ПСП07, ПСП08, ПСП09, ПСПI0.

Управляющий блок является резидентом системы. Он организует загрузку в оперативную память остальных блоков. Блоки ПСП0I, ПСП02 и ПСП03 являются вспомогательными; они предназначены для анализа управляющих карт, формирования системных таблиц, синтаксического анализа текста програми и в случае необходимости - трансляции программы.

Блок ПСП04 преобразовывает текст программы во внутреннее представление в виде графа и размещает его в оперативной памяти ЭВМ. Дуги этого графа изображают возможные передачи управления в программе, а вершины - команды программы. Кроме этого во внутреннем представлении идентификаторы объектов программы заменены адресами размещения их описаний. Это позволяет быстро

обращаться к соответствующим объектам и компактно хранить программу (программа, состоящая из 400 операторов, занимает обычно не более 1500 ячеек).

Основная задача блока ПСП05 - нахождение множества существенно расположенных команд программы (см.§ 3.3) и существенных переменных у этих команд. В качестве побочного результата ПСП05 выявляет недостижимые команды программы и переменные, к которым производится обращение без предварительного присвоения им какого-нибудь значения.

Наиболее сложным является блок ПСП06, который предназначен для построения дерева реализуемости. Функционально ПСП06 состоит из двух частей: СТРАТЕГИИ и АНАЛИЗАТОРА.

СТРАТЕГИЯ управляет построением дерева реализуемости по принципу "сначала вглубь", т.е. сначала строится одна ветвь дерева реализуемости до ее полного завершения, затем выбирается начало следующей ветви и она опять строится до полного ее завершения и т.д. Началом каждой следующей ветви служит вершина, лежащая на некоторой ранее построенной ветви дерева реализуемости. Притом выбирается всегда та вершина, из которой выходит еще неисследованная ветвь. СТРАТЕГИЯ прекращает построение дерева реализуемости в следующих 4-х случаях:

- дерево реализуемости построено полностью;
- покрывающее множество частично построенного дерева реализуемости содержит все дуги программы;
- использована вся оперативная память, предоставленная для работы системы;
- прошло время, предусмотренное для работы системы (обычно

для построения дерева реализуемости отводилось 5 минут).

Построение каждой ветви дерева реализуемости проводится по шагам. На каждом шаге СТРАТЕГИЯ достраивает к рассматриваемой ветви так называемый элементарный путь, соединяющий две соседние, существенно расположенные команды (СРК), т.е. не содержащий других СРК. Началом следующего элементарного пути служит команда, на которой оканчивается элементарный путь, построенный на предыдущем шаге. Кроме этого в конце каждого элементарного пути строится состояние, которое служит начальным для следующего шага. СТРАТЕГИЯ продолжает построение очередной ветви, пока не достигнута команда СТОП. Это построение может быть прекращено также в случаях, указанных ниже.

Один шаг построения ветви дерева реализуемости выполняется следующим образом. Предыдущим шагом или выбором начала ветви задана СРК и состояние, приписанное к этой команде. СТРАТЕГИЯ по этой начальной команде и состоянию определяет элементарный путь, который или вообще не встречается в уже построенной части дерева реализуемости, или встречается в ней минимальное число раз. (СТРАТЕГИЯ старается всегда пройти по ранее неисследованным частям программы.) После этого СТРАТЕГИЯ обращается к АНАЛИЗАТОРУ с вопросом: реализуем ли выбранный элементарный путь или нет. АНАЛИЗАТОР, используя начальное состояние и заданный элементарный путь строит систему неравенств, анализирует ее и выдает СТРАТЕГИИ один из 4-х возможных ответов:

- заданный элементарный путь реализуем; в качестве дополнительной информации в этом случае АНАЛИЗАТОР выдает состояние после исследованного элементарного пути;

- заданный элементарный путь нереализуем по той причине, что значение счетчика еще не достигло значения, при котором происходит выход из цикла программы;
- заданный элементарный путь нереализуем по другим причинам или реализуемость этого пути нельзя установить; в качестве дополнительной информации в этом случае сообщается причина нереализуемости (ошибка программы, противоречивые логические условия, сложные команды программы и т.п.);
- реализуемость заданного элементарного пути невозможно проверить из-за недостатка оперативной памяти.

При получении первого ответа СТРАТЕГИЯ достраивает к рассматриваемой ветви элементарный путь. Затем она сравнивает состояние, построенное АНАЛИЗАТОРОМ, со состояниями, которые построены ранее и приписаны к СРК конца элементарного пути. Если такое состояние уже приписано ранее, то построение ветви дерева реализуемости прекращается. Этим предотвращается возможность зацикливания блока ПСПОб в том случае, когда зацикливается анализируемая программа. В противоположном случае СТРАТЕГИЯ приписывает состояние, построенное АНАЛИЗАТОРОМ, к СРК в конце элементарного пути и переходит к следующему шагу.

Если СТРАТЕГИЯ получает третий ответ, то причина нереализуемости отмечается в дереве реализуемости, а СТРАТЕГИЯ повторяет этот же шаг при другом элементарном пути.

Если СТРАТЕГИЯ получает второй ответ, то этот же шаг повторяется при другом элементарном пути только в том случае, если значение счетчика приближается к значению, при котором происходит выход из цикла программы. Если это не имеет место, то

построение ветви прекращается (этим эвристическим принципом мы несколько отклоняемся от описанного в предыдущей главе алгоритма построения дерева раелизуемости).

Необходимо отметить, что все состояния, приписываемые к вершинам дерева реализуемости, помещаются в оперативной памяти. Размещение этих состояний на внешних носителях информации невозможно по той причине, что частое обращение к уже построенным состояниям повлекло бы за собой недопустимые потери машинного времени. Поэтому особое внимание при разработке ПСП06 было уделено рациональному кодированию состояний, но более подробно на этом мы здесь останавливаться не будем.

Блок ПСП07 строит покрывающее множество путей программы согласно алгоритму, который изложен уже в § I.4 и § 3.3. Кроме этого ПСП07 печатает ошибки программы и предупреждения, выявленные АНАЛИЗАТОРОМ.

Примеры, реализующие пути покрывающего множества, строятся блоком ПСП08 и записываются на внешние носители информации блоком ПСП09. Многократный запуск отлаживаемой программы на построенных примерах и выдача протоколов работы программы на этих примерах осуществляется блоком ПСП10.

Важно подчеркнуть, что если из-за недостатка памяти, времени или из-за сложных средств, используемых в программе, блок ПСП06 строит дерево реализуемости только частично, то все-таки система выдает примеры, которые возможно построить исходя из частично построенного дерева реализуемости. Поэтому имеет смысл систему применять не только к тем программам, для которых алгоритм, описанный в третей главе, гарантирует построение ПСП, а также к более сложным программам.

4.2. Анализ результатов работы системы СМОТЛ.

Разработанная экспериментальная система автоматического построения ПСП СМОТЛ была проверена на реальных программах АСУ - Министерства социального обеспечения и АСУ министерства связи Латвийской ССР. Первая из них уже эксплуатировалась на реальных данных, а вторая находилась в стадии отладки. Важно подчеркнуть, что проверке подвергались все программы названных АСУ за исключением, разумеется, таких как программы сортировки.

Система СМОТЛ эксплуатировалась при следующих технических ограничениях. Во первых, для работы системы была выделена память 26II2 машинных слов. Это является максимальной памятью, которая доступна в стандартном комплекте ЭВМ "Минск-32". Во вторых, как показывает практика для полного построения дерева реализуемости отдельных программ может понадобиться более 20 минут работы ЭВМ "Минск-32". Но это с практической точки зрения недопустимо. Поэтому время построения дерева реализуемости было ограничено 5 минутами. Благодаря этому для обработки одной программы системой СМОТЛ в среднем было израсходовано I0-I2 минут машинного времени, а вместе с трансляцией и сборкой программы - 20 минут машинного времени ЭВМ "Минск-32".

При проверке программ получены весьма обнадеживающие результаты: для программ, количество команд которых не больше 300 (можно считать что одна команда языка СМОД соответствует одному оператору раздела процедур или одному предложению описания данных КОБОЛа), из 25 проверенных программ ПСП было постро-

но для I6 программ; для 3 программ были построены примеры, которые реализовали более 90% из всех команд программ; для 4 программ примеры реализовали от 49% до 70% из всех команд программы. Только для 2 программ не были построены примеры из-за ограниченности реализованного алгоритма.

В каждой проверяемой программе на основе ПСП было обнаружено в среднем по три ошибки. Как правило эти ошибки не являлись существенными, например, неверная обработка пустых файлов; некоторые из найденных ошибок возможны были только на весьма неестественных примерах. Однако обнаружение в отлаженных программах ошибок, о которых программист, как показывает практика, даже не догадывается, весьма ценно.

В целом можно утверждать, что для программ, количество команд в которых не превышает 300, изложенный в главе 3 алгоритм для большинства реально разрабатываемых программ за приемлемое время строит ПСП и таким образом вполне удовлетворяет требованиям практики.

При проверке программ с числом команд больше 300 из I4 проверенных программ примеры были построены только для 5 программ (из I4 программ II программ имели количество команд свыше 500 команд).

Это обусловлено, во первых, тем, что проверяемые программы имели очень сложную логическую структуру, например, программы содержади циклы обработки массивов, внешний цикл из которых организован по счетчику, а внутренний содержит много различных контролей. Поэтому для обработки этих программ необходимо накапливать в оперативной памяти несколько сотен состояний и в ре-

зультате этого системе обычно было мало памяти.

Во вторых, используемая концепция состояния в некоторых случаях оказалась недостаточно "хорошей", особенно, если в программе происходит нерегулярное обращение к элементам массивов, а также при использовании комаид умножения и деления.

В качестве положительного результата работы системы на программах, количество операторов которых превышает 300, необходимо отметить то, что в проверяемых программах было найдено много ошибок (в среднем 7 ошибок в программе), которые являлись более существенными чем в первом случае. Это показывает только то, что программисты в больших программах не в состоянии проверить всевозможные ситуации.

Л И Т Е Р А Т У Р А

1. Барздинь Я.М., Калниньш А.А. Построение полных систем примеров для программ, работающих с прямым методом доступа. - В сб. Ученые записки Латв.гос.ун-та, Рига,1975,т.233,123-155.

2. Корбут А.А., Финкельштейн Ю.Ю. Дискретное программирование. - Наука, Москва, 1969.

3. Кулаковская В.П., Романовская Л.М., Савченко Т.А., Фельдман Л.С. КОБОЛ ЭВМ "МИНСК-32". - Статистика, Москва,1973.

4. Курочкин В.М. Универсальный язык программирования РЛ/I. - Мир, Москва, 1968.

5. Кушнерев Н.Т., Неменман М.Е., Цегельский В.И. Программирование для ЭВМ "Минск-32". - Статистика, Москва, 1973.

6. Мальцев А.И. Алгоритмы и рекурсивные функции. - Наука, Москва, 1965.

7. Минский М. Вычисления и автоматы. - Мир, 1971.

8. Clarke L.A. A System to Generate Test Data and Symbolically Execute Programs. - IEEE Transactions on Software Engineering, Vol.Se-2, September 1976, 215-222.

9. Gabow H., Shachindra M.N., Osterweil L.J. On Two Problems in the Generation of Program Test Paths. - IEEE Transactions on Software Engineering, Vol.Se-2, September 1976, 227-231.

10. Hanford K.V. Automatic Generation of Test Cases. - IBM Systems Journal, Vol.9.4, 1970, 242-257.

11. Hetzel W.C. Principles of Computer Program Testing. - Program Test Methods /based on the proceedings of the Computer Program Test Methods Symposium held at the University of North Carolina, Chapel Hill, June 21-23, 1972/, Prentice-Hall, INC., Englewood Cliffs, New Jersey, 17-28.

12. Hicks H.T. The Air Force Cobol Compiler Validation System.- Datamation, August 1969, 73-81.

13. Howden W.E. Methodology for the Generation of Program Test Data. - IEEE Transactions on Computers, Vol.SE-24, May 1975, 554-560.

14. Howden W.E. Symbolic Testing and the DISSECT Symbolic Evaluation System. - Computer Science Technical Report Number 11, University of California, May 1976, 1-35.

15. Howden W.E. Reliability of the Path Analysis Testing Strategy. - IEEE Transactions on Software Engineering, Vol.Se-2, September 1976, 208-215.

16. Miller E.F., Paige M.R. Automatic Generation of Software Testcases. - Eurocomp. Conference Proceedings, 1974, 1-12.

17. Miller W., Spooner D.L. Automatic Generation of Floating--Point Test Data. - IEEE Transactions on Software Engineering, Vol.SE-2, September 1976, 223-226.

18. Sauder R.L. General Test Data Generator For Cobol. - AFIPS Conference Proceedings SICC, 1962, 317-323.

19. Барздинь Я.М., Бичевский Я.Я., Калниньш А.А. Построение полной системы примеров для проверки корректности программ.- В сб. Ученые записки Латв.гос.ун-та, Рига, 1974, т.210, 152-188.

20. Бичевский Я.Я. Автоматическое построение систем примеров.- ж.Программирование, 1977 № 3.

21. Бичевский Я.Я., Калниньш А.А., Барздинь Я.М. Построение полной системы примеров для проверки корректности программ.- Тезисы докладов III Всесоюзной конференции по проблемам теоретической кибернетики, Новосибирск, ИМ СОАН, 1974.

22. Бичевский Я.Я., Зариньш А.К., Калниньш А.А. Инструкция по работе с системой макрокоманд обработки данных (СМОД). Часть I. Основные операторы. - Ассоциация пользователей ЭВМ типа "Минск", Москва, 1973, выпуск 9-10, 1-109.

23. Бичевский Я.Я., Балодис Р.П. Система макрокоманд обработки данных (СМОД). Часть III. Инструкция по использованию транслятора на ЭВМ "Минск-32". - Ассоциация пользователей ЭВМ типа "Минск", Москва, 1973, выпуск 9-10, 143-189.

24. Калниньш А.А., Бичевский Я.Я., Барздинь Я.М. Разрешимые и неразрешимые случаи проблемы построения полной системы примеров. - В сб. Ученые записки Латв.гос.ун-та, Рига, 1974, т.210, 188-206.

25. Barzdin J.M., Bičevskis J.J., Kalninsh A.A. Construction of complete sample system for correctness testing. - Mathematical Foundations of Computer Science, Berlin, Springer Verlag, 1975, 1-12.

26. Barzdin J.M., Bicevskis J.J., Kalninsh A.A. Automatic Construction of Complete Sample System for Program Testing.- IFIP'77 Congress Proceedings.

ПРИЛОЖЕНИЕ

В качестве примера рассмотрим работу системы СМОТЛ на программе ВОМ I9 из математического обеспечения АСУ Министерства социального обеспечения Латвийской ССР. Эта программа состоит из 226 операторов СМОДа /22/, и она предназначена для выдачи двух табулограмм "Перечень информации, учтенной при расчете на ЭВМ сумм пенсий (пособий) по ведомостям" (форма Ф.77П05) и "Перечень обработанных документов по учету обслуживаемых лиц" (форма Ф.77П06).

Входной информацией для ВОМI9 является массив А, описание которого приведено в тексте программы в строке 01010. Массив А вводится с магнитной ленты (МЛ); записи массива А фиксированной длины - 110 символов. (Напомним, что для ЭВМ "Минск-32" вместо термина "файл" принято пользоваться термином "массив"). Табулограммы, выдаваемые ВОМI9, оформляются в виде массивов П и П1 (см. строки 01020 и 01030). Согласно постановке задачи первый символ записей массива А служит признаком типа записи: если первый символ записи имеет значение "2" или "3", то по информации, содержащейся в этой записи, должна образоваться строка формы Ф.77П05, а если первый символ записи имеет значение "1", то должна образоваться строка формы Ф.77П06 (образцы этих форм. см. на стр.126 и 127). Формы должны печататься по страницам; переход к новой странице формы Ф.77П05 должен осуществляться, если меняется или зона обслуживания, или почтовый

индекс, или день выплаты пенсии, а переход к новой странице формы Ф.77П06 должен осуществляться, если меняется номер комплекта. Значения указанных выше реквизитов содержатся в каждой записи массива А.

Теперь для того, чтобы возможно было бы ознакомиться с программой ВОМ19 более подробно, разъясним некоторые операторы СМОДа. Программа ВОМ19 начинается операторами ЯСК (строки 01040-01090), которые предназначены для организации выхода из программы в случае аварии ЭВМ. Эти операторы системой СМОТЛ не анализируются. Оператор ОВ - Открыть Входной в строке 01110 открывает массив А, операторы ПЛ - Переслать Литерал в строках 01150-01250 определяют реквизиты К, К1, К2 и т.д. и присваивают им начальные значения. Например, реквизит К определяется и ему присваивается начальное значение '0'. Оператор ПД - Переслать Дату в строке 01260 засылает в реквизит ДАТА дату в форме ГГГГММДД. Затем оператор ЧРЦЦ - Читать из Реквизита из ДАТА начиная с 5ой цифры считывает 2 цифры и засылает в реквизит М. Оператор условного перехода И - Идти анализирует достоверность установленной даты. Если номер месяца равен нулю, то выдается Указание Оператору (см.строку 01290), в противном случае управление передается оператору с меткой А01. Оператором ВЗ - Ввести Запись делается доступной следующая запись массива А и управление передается следующему оператору в строке 01310. Если массив А исчерпан, то управление передается оператору с меткой ВЫХ, а если произошла авария внешнего устройства, то оператору с меткой ВВ2 (этот выход оператора системой СМОТЛ не анализируется). Оператор ЧВСС - Читать с Ввода в строке 01320 счи-

С М О Т Л

13.23 1977.04.25

ЗАКАЗ:

ИДЕНТВОМ19БОЛ19ВОР19
ЛЕНТЫПРОГРПРОГРПВОГР111
ПСПСМ
ПЧПРИ
ВЫПОЛ
ПЧПРО
ТЕКСТ

СМОД МИНСК_32 ВОМ19

ОПИСАНИЯ МАССИВОВ И ТАБЛИЦ

```
                :   :   1:1   1:1:12:22:2:2   2:2:3:3 3:3:3  3:3 4:4   4:4                          8: ОШИБ
      1   5:6 8:9  2:3   7:8:90:12:3:4   8:9:0:1 3:4:5  8:9 1:2   6:7                          0:  КА
-------------------------------------------------------------------------------------------------------
      01010:   :ОМ  :А    :В:МЛ:06:М:     :Ф: :110: :     :   :     :                          :
      01020:   :ОМ  :П    : :ПЧ:01: :  36: : :   : :     :   :     :                          :
      01030:   :ОМ  :П1   : :ПЧ: 1: :  36: : :   : :     :   :     :                          :
```

```
                    *      В Х О Д      *П А М Я Т Б*   В Ы Х О Д    * А Р И Ф М Е Т И К А *                    *
 N    :МЕТ:КОД *И :НАЗВА: Ч :КОЛ: *НАЗВА:ДЛИ:Д*НАЗВА: N :*:Г:З*ОПЕР.:О:ОПЕР.:МЕТ:МЕТ* Л И Т Е Р А Л    * ОШИБ
СТР.:КА :     *  : НИЕ :СИМ:СИМ: *НИЕ   :НА : *НИЕ  :СИМ: : : *  1  :П:  2  :КА1:КА2*                    * КА
                1*11:1   1:2 2:2 2:2*2   3:3 3:3*3   4:4 4:4:4:4*4   5:5:5   5:5 6:6 6*6                 8*
1   5:6 8:9  2*34:5   9:0 2:3 5:6*7   1:2 4:5*6   0:1 3:4:5:6*7   1:2:3   7:8 0:1 3*4                 0*
01040:    :БАЗ :  :0
01050:AAA:РЗВ :  :3
01060:    :РИП :  :16
01070:    :СУ  :  :4;+16
01080:    :ПОВТ:  :
01090:    :КА  :  :ВВВ;AAA
01100:    :И   :  :      :   :   :   :      :   :      :   :   :   :   :      :   :      :CCC:   :
01110:CCC:ОВ   :  :A     :   :   :   :      :   :      :   :   :   :   :      :   :      :   :   :
01150:    :ПЛ  :  :'0'   :   :   :   :К     : 8:      :   :   :   :   :      :   :      :   :   :
01160:    :ПЛ  :  :'0'   :   :   :   :К1    : 1:      :   :   :   :   :      :   :      :   :   :
01170:    :ПЛ  :  :'0'   :   :   :   :К2    : 6:      :   :   :   :   :      :   :      :   :   :
01180:    :ПЛ  :  :'0'   :   :   :   :К3    : 2:      :   :   :   :   :      :   :      :   :   :
01190:    :ПЛ  :  :'0'   :   :   :   :К4    : 1:      :   :   :   :   :      :   :      :   :   :
01200:    :ПЛ  :  :+0    :   :   :   :ДП    : 2:      :   :   :   :   :      :   :      :   :   :
01210:    :ПЛ  :  :+0    :   :   :   :ДП1   : 2:      :   :   :   :   :      :   :      :   :   :
01220:    :ПЛ  :  :+1    :   :   :   :СТР   : 3:      :   :   :   :   :      :   :      :   :   :
01230:    :ПЛ  :  :+1    :   :   :   :СТР1  : 3:      :   :   :   :   :      :   :      :   :   :
01240:    :ПЛ  :  :+1    :   :   :   :НПП   : 4:      :   :   :   :   :      :   :      :   :   :
01250:    :ПЛ  :  :+1    :   :   :   :НПП1  : 4:      :   :   :   :   :      :   :      :   :   :
01260:AB1:ПД   :  :      :   :   :   :ДАТА  :  :      :   :   :   :   :      :   :      :   :   :
01270:    :ЧРЧЦ:  :ДАТА  :  5:  2:   :М     :  :      :   :   :   :   :М     :=:+0      :   :A01:
01280:    :И   :  :      :   :   :   :      :  :      :   :   :   :   :' УСТ:А:НОВИТ:Ь Д:АТУ:  НА ПУЛЬТЕ
01290:    :ЧО  :  :      :   :   :   :      :  :      :   :   :   :   :      :   :      :A51:
01300:    :И   :  :      :   :   :   :      :  :      :   :   :   :   :      :   :      :ВЫХ:ВВ2:
01310:A01:ВЗ   :  :A     :   :   :   :      :  :      :   :   :   :   :      :   :      :   :   :
01320:    :ЧВСС:  :A     :  1:  1:   :МАК   :  :      :   :   :   :   :      :   :      :   :   :
01330:    :И   :  :      :   :   :   :      :  :      :   :   :   :   :МАК   :=:'1'     :   :A03:
01340:    :ЧВСС:  :A     : 23:  8:   :КОМП  :  :      :   :   :   :   :      :   :      :   :   :
01350:    :И   :  :      :   :   :   :      :  :      :   :   :   :   :КОМП  :=:К      :A02:   :
01360:    :ПЛ  :  :+1    :   :   :   :НПП   :  :      :   :   :   :   :      :   :      :   :   :
01370:    :ПИСТ:  :      :   :   :   :      :  :П     :   :   :   :   :      :   :      :   :   :
01380:    :ПЛ  :  :+1    :   :   :   :СТР   :  :      :   :   :   :   :      :   :      :   :   :
01390:    :ПЛ  :  :+0    :   :   :   :ДП    : 2:      :   :   :   :   :      :   :      :   :   :
01400:B01:ПЧИ  :  :      :   :  1:   :      :  :П     :   :   :   :   :      :   :      :   :   :
01410:    :ПТ  :  :      :   :   :   :      :  :П     : 11:   :   :   :'/   /':   :      :   :   :
01420:    :ПТ  :  :      :   :   :   :      :  :П     : 38:   :   :   :'ПОДС:И:СТЕМА:  *:В Ы: П Р А Т А*!
01430:    :ПТ  :  :      :   :   :   :      :  :П     : 79:   :   :   :'Ф.77:П:06'     :   :   :
01440:    :ПТ  :  :      :   :   :   :      :  :П     :110:   :   :   :'СТР.':        :   :   :
01450:    :ЧРЧЦ:  :ДАТА  :  3:  2:   :Г     :  :      :   :   :   :   :      :   :      :   :   :
01460:    :ЗЫЧС:  :      :   :   :   :Г     :  :П     :  9:   :   :   :      :   :      :   :   :
```

```
                    *      В Ы Х О Д      *П А М Я Т Ь*   В Ы Х О Д    * А Р И Ф М Е Т И К А *                    *
 N     :МЕТ:КОД *И :НАЗВА: Ч :КОЛ: *НАЗВА:ДЛИ:Д*НАЗВА: N  :*:Г:З*ОПЕР.:О:ОПЕР.:МЕТ:МЕТ* Л И Т Е Р А Л    * ОШИБ
СТР.:КА :       *  : НИЕ :СИМ:СИМ: *НИЕ  :НА : *НИЕ  :СИМ: : : *  1  :П:  2  :КА1:КА2*                    * КА
                1*11:1    1:2 2:2 2:2*2    3:3 3:3*3    4:4 4:4:4:4*4    5:5:5    5:5 6:6 6*6                8*
 1   5:6 8:9    2*34:5    9:0 2:3 5:6*7    1:2 4:5*6    0:1 3:4:5:6*7    1:2:3    7:8 0:1 3*4                0*
01470:    :ЗЦЧС:  :       :   :   : :М     :   : :П   : 12:  :  :  :       :  :       :   :   :
01480:    :ЧРЧЦ:  :ДАТА   : 7:  2: :Д     :   : :    :   :  :  :  :       :  :       :   :   :
01490:    :ЗЦЧС:  :       :   :   : :Д     :   : :П   : 15:  :  :  :       :  :       :   :   :
01500:    :ЗЦЧС:  :       :   :   : :СТР   :   : :П   :114:  :  :  :       :  :       :   :   :
01510:    :ПЧ   : :       :   :   : :       :   : :П   :   :  :  :  :       :  :       :   :   :
01520:    :ПЧИ  : :       :   : 1: :       :   : :П   :   :  :  :  :       :  :       :   :   :
01530:Ь01:А     : :       :   :   : :ДП    :   : :    :   :  :  :ДП    :*:+3    :  -:   :   :
01540:    :ПТ   : :       :   :   : :       :   : :П   : 19:  :  :'ПЕРЕ:Ч:ЕНЬ О:БРА:БОТ:АННЫХ ДОКУМЕНТОВ
01550:    :     : :       :   :   : :       :   : :    :   :  :  :ПО УЧ:Е:ТУ ОБ:СЛУ:ЖИВ:АЕМЫХ ЛИЧ.'
01560:    :ПЧ   : :       :   :   : :       :   : :П   :   :  :  :  :       :  :       :   :   :
01570:    :ПЧИ  : :       :   : 1: :       :   : :П   :   :  :  :  :       :  :       :   :   :
01580:    :ПТ   : :       :   :   : :       :   : :П   : 19:  :  :'КОНП:Л:ЕКТ:'
01590:    :ЗЦСС:  :       :   :   : :КОМП   :   : :П   : 29:  :  :  :       :  :       :   :   :
01600:    :ПЧ   : :       :   :   : :       :   : :П   :   :  :  :  :       :  :       :   :   :
01610:    :ПЧИ  : :       :   : 1: :       :   : :П   :   :  :  :  :       :  :       :   :   :
01620:    :ПМ   : :'-'    :   : 59: :       :   : :П   : 15:  :  :  :       :  :       :   :   :
01630:    :ПЧ   : :       :   :   : :       :   : :П   :   :  :  :  :       :  :       :   :   :
01640:    :ПТ   : :       :   :   : :       :   : :П   : 15:  :  :': N:  : : N:  ::   :НОМЕР    : ПОЧТ:
01650:    :     : :       :   :   : :       :   : :    :   :  :  :ОБЫП ::  ВИ:Д  :  ::   N   :'
01660:    :ПЧ   : :       :   :   : :       :   : :П   :   :  :  :  :       :  :       :   :   :
01670:    :ПТ   : :       :   :   : :       :   : :П   : 15:  :  :': П/:П: :ФОР:МЫ:: ПЕ:НОИСНЕРА  : ИНД:
01680:    :     : :       :   :   : :       :   : :    :   :  :  :ЕКС  ::  ПЕН:СИИ:  :: ОШИБКИ :'
01690:    :ПЧ   : :       :   :   : :       :   : :П   :   :  :  :  :       :  :       :   :   :
01700:    :А    : :       :   :   : :ДП    :   : :    :   :  :  :ДП    :*:+7    :   :   :
01710:Г01:ПМ   : :'-'    :   : 59: :       :   : :П   : 15:  :  :  :       :  :       :   :   :
01720:    :ПЧ   : :       :   :   : :       :   : :П   :   :  :  :  :       :  :       :   :   :
01730:    :ПТ   : :       :   :   : :       :   : :П   : 15:  :  :': 1.  : : 2. ::'
01740:    :ПТ   : :       :   :   : :       :   : :П   : 35:  :  :'3.'
01750:    :ПТ   : :       :   :   : :       :   : :П   : 42:  :  :': :4.  : : :5.  : 6.  :':
01760:    :ПЧ   : :       :   :   : :       :   : :П   :   :  :  :  :       :  :       :   :   :
01770:    :ПМ   : :'-'    :   : 59: :       :   : :П   : 15:  :  :  :       :  :       :   :   :
01780:    :ПЧ   : :       :   :   : :       :   : :П   :   :  :  :  :       :  :       :   :   :
01790:Е01:А     : :       :   :   : :ДП    :   : :    :   :  :  :ДП    :*:+3    :   :   :
01800:    :ЧРСС:  :КОМП   : 1:  8: :К      :   : :    :   :  :  :  :       :  :       :   :   :
01810:А02:ЗЦЧС:  :       :   :   : :НПП   :   : :П   : 16:  :П:  :       :  :       :   :   :
01820:    :ЧВСС:  :А      : 31:  2: :НОФОР: 2:  :    :   :  :  :  :       :  :       :   :   :
01830:    :ЧВСС:  :А      : 33: 12: :НОМЕР: 12: :    :   :  :  :  :       :  :       :   :   :
01840:    :ЧВСС:  :А      : 45:  6: :ПОЧИН: 6:  :    :   :  :  :  :       :  :       :   :   :
01850:    :ЧВСС:  :А      : 51:  8: :ВИДП : 8:  :    :   :  :  :  :       :  :       :   :   :
01860:    :ЧВСС:  :А      : 59:  2: :НОШИБ: 2:  :    :   :  :  :  :       :  :       :   :   :
```

N СТР.	МЕТКА	КОД	И	НАЗВАНИЕ	N СИМ	КОЛ СИМ	НАЗВАНИЕ	ДПИНА	Д НАЗВАНИЕ	N СИМ		Г	З ОПЕР. 1	О П	ОПЕР. 2	МЕТКА1	МЕТКА2	ЛИТЕРАЛ	ОШИБКА
1	5:6	8:9	1*34:5		9:0	2:3	5:6*7	1:2	4:5*6	0:1	3:4	5:6*7	1:2	3:4	7:8	0:1	3*4	0*	
01870:		ЗЦСС					НОФОР		П	24									
01880:		ЗЦСС					НОМЕР		П	29									
01890:		ЗЦСС					ПОЧИН		П	45									
01900:		ЗЦСС					ВИДП		П	56									
01910:		ЗЦСС					НОШИБ		П	68		П							
01920:		ПЧ							П										
01930:		А					ДЛ						ДЛ	+	+1				
01940:		А					НПП						НПП	+	+1				
01950:		И											ДЛ	<	+33	A01			
01960:		ЛИСТ							П										
01970:		А					СТР						СТР	+	+1				
01980:		ПЛ	+0				ДЛ												
01990:		ИС														B01	E01		
02000:		И														A01			
02010:																			
02020:																			
02030:	A03	И											K4	=	'0'		E03		
02040:		ЛИСТ							П										
02050:		ПЛ	'1'				K4												
02060:	E03	ЧВСС	A	2	1		ЗОНА												
02070:		ЧВСС	A	3	6		ПОЧ												
02080:		ЧВСС	A	9	2		ДЕНЬ												
02090:		И											ЗОНА	=	K1		A05		
02100:		И											ПОЧ	=	K2		A05		
02110:		И											ДЕНЬ	=	K3	A04			
02120:	A05	ПЛ	+1				СТР1												
02130:		ПЛ	+1				НПП1												
02140:		ПЛ	+0				ДП1												
02150:		ЛИСТ							П1										
02160:	B05	ПЧИ			1				П1										
02170:		ПТ							П1	11			//	/					
02180:		ПТ							П1	36			'ПОДС	И	СТЕМА			В Ы: П Л А Т А*'	
02190:		ПТ							П1	79			'Ф.77	П	05'				
02200:		ПТ							П1	110			'СТР.	:					
02210:		ЗЦЧС					Г		П1	9									
02220:		ЗЦЧС					М		П1	12									
02230:		ЗЦЧС					Д		П1	15									
02240:		ЗЦЧС					СТР1		П1	114									
02250:		ПЧ							П1										
02260:		ПЧИ			1				П1										

```
                *       В Х О Д      *П А М Я Т Б*   В Ы Х О Д    * А Р И Ф М Е Т И К А *                  *
----------------------------------------------------------------------------------------------------------------
   N   :МЕТ:КОД *И :НАЗВА: Ч :КОЛ: *НАЗВА:ДЛИ:Д*НАЗВА: N  :*:Г:З*ОПЕР.:О:ОПЕР.:МЕТ:МЕТ* Л И Т Е Р А Л   * ОШИБ
 СТР.:КА :     *Ы :  НИЕ :СИМ:СИМ: *НИЕ  :НА : *НИЕ :СИМ: : : *  1  :П:  2  :КА1:КА2*                  *  КА
----------------------------------------------------------------------------------------------------------------
                1*11:1   1:2 2:2 2:2*2   3:3 3:3*3   4:4 4:4:4:4*4   5:5:5   5:5 6:6 6*6                 8*
    1   5:6 8:9 2*34:5   9:0 2:3 5:6*7   1:2 4:5*6   0:1 3:4:5:6*7   1:2:3   7:8 0:1 3*4                 0*
----------------------------------------------------------------------------------------------------------------
 02270:Б05:А   :  :     :   :   : :ДЛ1  :   : :     :   : : : :ДЛ1  :*:+3  :   :   :
 02280:    :ПТ  :  :     :   :   : :     :   : :П1   : 22: : : :'ПЕРЕ:Ч:ЕНЬ И:НФО:РМА:ЦИИ,УЧТЕННОЙ ПРИ :
 02290:    :    :  :     :   :   : :     :   : :     :   : : : :РАСЧЕ:Т:Е НА :ЗВМ: СУ:ММ ПЕНСИЙ (ПОСОБИ:
 02300:    :    :  :     :   :   : :     :   : :     :   : : : :А) ПО: :ВЕДОМ:ОСТ:ЯМ.:'
 02310:    :ПЧ  :  :     :   :   : :     :   : :П1   :   : : : :     :   :   :   :
 02320:    :ПТ  :  :     :   :   : :     :   : :П1   : 11: : : :'ЗОНА:  :'  :   :   :
 02330:    :ЗЫСС:  :     :   :   : :ЗОНА  :   : :П1   : 17: : : :     :   :   :   :
 02340:    :ПЧ  :  :     :   :   : :     :   : :П1   :   : : : :     :   :   :   :
 02350:    :ПТ  :  :     :   :   : :     :   : :П1   : 11: : : :'ПОЧТ:О:ВЫЙ И:НДЕ:КС:
 02360:    :ЗЫСС:  :     :   :   : :ПОЧ   :   : :П1   : 28: : : :     :   :   :   :
 02370:    :ПЧ  :  :     :   :   : :     :   : :П1   :   : : : :     :   :   :   :
 02380:    :ПТ  :  :     :   :   : :     :   : :П1   : 11: : : :'ДЕНЬ:  :ВЫПЛА:ТЫ:'
 02390:    :ЗЫСС:  :     :   :   : :ДЕНЬ  :   : :П1   : 25: : : :     :   :   :   :
 02400:    :ПЧ  :  :     :   :   : :     :   : :П1   :   : : : :     :   :   :   :
 02410:    :ПЧИ :  :     :   : 1 : :     :   : :П1   :   : : : :     :   :   :   :
 02420:    :ПЦ  :  :'-'  :   :122: :     :   : :П1   :  2: : : :     :   :   :   :
 02430:    :ПЧ  :  :     :   :   : :     :   : :П1   :   : : : :     :   :   :   :
 02440:    :ПТ  :  :     :   :   : :     :   : :П1   :  2: : : :';    :  :,'  :   :   :
 02450:    :ПТ  :  :     :   :   : :     :   : :П1   : 29: : : :'СВЕД:Е:НИЯ  :ИЗ : ОС:НОЗНОГО  ИНФОРМАЦ:
 02460:    :    :  :     :   :   : :     :   : :     :   : : : :ИОНОГ:О:  МАС:СИВ:А'
 02470:    :ПТ  :  :     :   :   : :     :   : :П1   :110: : : :'; ОС:Н:ОВАНИ:Е  ::'
 02480:    :ПЧ  :  :     :   :   : :     :   : :П1   :   : : : :     :   :   :   :
 02490:    :ПТ  :  :     :   :   : :     :   : :П1   :  2: : : :';    :  :,'  :   :   :
 02500:    :ПЦ  :  :'-'  :   :122: :     :   : :П1   :  2: : : :     :   :   :   :
 02510:    :ПЧ  :  :     :   :   : :     :   : :П1   :   : : : :     :   :   :   :
 02520:    :ПТ  :  :     :   :   : :     :   : :П1   :  2: : : :'; N  : ::'  :   :   :
 02530:    :ПТ  :  :     :   :   : :     :   : :П1   : 22: : : :';'  :   :   :   :
 02540:    :ПТ  :  :     :   :   : :     :   : :П1   : 55: : : :';'  :   :   :   :
 02550:    :ПТ  :  :     :   :   : :     :   : :П1   : 66: : : :'А Д :Р: Е С':   :   :
 02560:    :ПТ  :  :     :   :   : :     :   : :П1   : 88: : : :';К В:Ы:ПЛ. П:О В:ЕДО:МОСТИ :  ШИФР  : :
 02570:    :    :  :     :   :   : :     :   : :     :   : : : :N :' :   :   :   :   :
 02580:    :ПЧ  :  :     :   :   : :     :   : :П1   :   : : : :     :   :   :   :
 02590:    :ПТ  :  :     :   :   : :     :   : :П1   :  2: : : :';П/П: ::    :НОМ:ЕР  :     :  ФАМИЛИЯ :
 02600:    :    :  :     :   :   : :     :   : :     :   : : : :, ИМЯ: :, ОТЧ:ЕСТ:ВО  :   :'
 02610:    :ПЦ  :  :'-'  :   : 55: :     :   : :П1   : 55: : : :     :   :   :   :
 02620:    :ПТ  :  :     :   :   : :     :   : :П1   :120: : : :';КОМ:П:ЛЕК-::*ОР::'
 02630:    :ПЧ  :  :     :   :   : :     :   : :П1   :   : : : :     :   :   :   :
 02640:    :ПТ  :  :     :   :   : :     :   : :П1   :  2: : : :';   : :: ПЕ:НСИ:ОНЕ:РА  :'
 02650:    :ПТ  :  :     :   :   : :     :   : :П1   : 33: : : :'ПОЛУ:Ч:АТЕЛЯ:'
 02660:    :ПТ  :  :     :   :   : :     :   : :П1   : 55: : : :';   : : УЛИЦ:А  :   :  : N : N : N :
```

N СТР.	МЕТ КА	КОД	И	ВХОД НАЗВАНИЕ	ВХОД ЧИСЛО СИМ	ВХОД КОЛ СИМ	ПАМЯТЬ НАЗВАНИЕ	ПАМЯТЬ ДЛИНА	ДЛ НАЗВАНИЕ	ВЫХОД N СИМ	ВЫХОД Г	АРИФМЕТИКА З	АРИФМЕТИКА ОПЕР. 1	АРИФМЕТИКА О/П	АРИФМЕТИКА ОПЕР. 2	МЕТКА1	МЕТКА2	ЛИТЕРАЛ	ОШИБКА
1	5:6	8:9	1*11:1 2*34:5	1:2 9:0	2:2 2:3	2:2*2 5:6*7	3:3 1:2	3:3*3 4:5*6	4:4 0:1	4:4:4*4 3:4:5:6*7	5:5:5 1:2:3	5:5 7:8	6:6 0:1	6*6 3*4					8* 0*
02670																			
02680																			
02690		ПЧ							П1										
02700		ПТ							П1	2									
02710		ПТ							П1	22									
02720		ПТ							П1	55									
02730		ПТ							П1	73									
02740		ПТ							П1	74			ДОНА		КРП	К	ВАР	СУММА	КАКОГО КАК
02750													ОЕ						
02760		ПТ							П1	119									
02770		ПЧ							П1										
02780		А					ДП1						ДЛ1		+1				
02790	ГО5	ПМ		-		122			П1	2									
02800		ПЧ							П1										
02810		ПТ							П1	2			1.			2.			
02820		ПТ							П1	37			3.						
02830		ПТ							П1	55					4.			5. 6. 7.	
02840													8.		9.			10. 11. 12	
02850																			
02860		ПЧ							П1										
02870		ПМ		-		122			П1	2									
02880		ПЧ							П1										
02890	Е05	А					ДП1						ДЛ1		+3				
02900		ЧРСС		ЗОНА	1:	1:	К1												
02910		ЧРСС		ПОЧ	1:	6:	К2												
02920		ЧРСС		ДЕНЬ	1:	2:	К3												
02930	А04	ЗЦИС					НП1		П:	3:	П:								
02940		П		А	11:	12:			П1	9:									
02950		П		А	33:	30:			П1	24:									
02960		П		А	63:	15:			П1	57:									
02970		П		А	78:	4:			П1	74:	П:								
02980		П		А	82:	2:			П1	80:	П:								
02990		П		А	84:	4:			П1	84:	П:								
03000		П		А	88:	5:			П1	89:	П:2								
03010		П		А	93:	6:			П1	97:									
03020		П		А	99:	6:			П1	104									
03030		П		А	23:	8:			П1	111									
03040		П		А	31:	2:			П1	120									
03050		ПЧ							П1										
03060		А					ДП1						ДЛ1		+1				

```
                    *      В Х О Д      *П А М Я Т 6*   В Ы Х О Д   * А Р И Ф М Е Т И К А *                    *
---------------------------------------------------------------------------------------------------------------------
    N   :МЕТ:КОД *И :НАЗВА: N :КОЛ: *НАЗВА:ДЛИ:Д*НАЗВА: N :*:Г:З*ОПЕР.:О:ОПЕР.:МЕТ:МЕТ* Л И Т Е Р А Л  * ОШИБ
  СТР.:КА :     *  : НИЕ :СИМ:СИМ: *НИЕ  :НА : *НИЕ  :СИМ: : : *  1  :П:  2  :КА1:КА2*                  * КА
---------------------------------------------------------------------------------------------------------------------
                1*11:1   1:2 2:2 2:2*2   3:3 3:3*3   4:4 4:4:4:4*4   5:5:5   5:5 6:6 6*6                 8*
    1   5:6 8:9 2*34:5   9:0 2:3 5:6*7   1:2 4:5*6  -0:1 3:4:5:6*7   1:2:3   7:8 0:1 3*4                 0*
---------------------------------------------------------------------------------------------------------------------
03070:    :А   :  :      :   :  :  :НПП1 :   :  :     :   :  :  :НПП1 :+:+1 :   :  :                          :
03080:    :И   :  :      :   :  :  :     :   :  :     :   :  :  :ДЛ1  :<:+34 :A01:  :                          :
03090:    :ЛИСТ:  :      :   :  :  :     :   :  :П1   :   :  :  :     :   :  :   :                          :
03100:    :А   :  :      :   :  :  :СТР1 :   :  :     :   :  :  :СТР1 :+:+1 :   :  :                          :
03110:    :ПД  :  :+0    :   :  :  :ДП1  :   :  :     :   :  :  :     :   :  :   :                          :
03120:    :ИС  :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :B05:E05:                       :
03130:    :И   :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :A01:  :                       :
03140:ВЫХ:КВ   :  :А     :   :  :  :     :   :  :     :   :  :  :     :   :  :   :  :                       :
03150:    :ЛИСТ:  :      :   :  :  :     :   :  :П1   :   :  :  :     :   :  :   :                          :
03160:    :ВЫХ :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
03170:ВВВ:КЧ   :  :+0    :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
03180:    :СО  :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
03190:    :    :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
03200:ВВ1:КВ   :  :А     :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
03210:    :VO  :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
03220:    :    :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
03230:    :ВЫХ :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
03240:ВВ2:СО   :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
03250:    :    :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
03260:    :И   :  :      :   :  :  :     :   :  :     :   :  :  :     :   :  :   :                          :
```

Пример 001

Входной массив A НА МЛ

00/04/00 ПОДСИСТЕМА «В Ы П Л А Т А» ф.77005 СТР.001

ПЕРЕЧЕНЬ ИНФОРМАЦИИ,УЧТЕННОЙ ПРИ РАСЧЕТЕ НА ЭВМ СУММ ПЕНСИЙ (ПОСОБИЙ) ПО ВЕДОМОСТЯМ.

ЗОНА: 0
ПОЧТОВЫЙ ИНДЕКС: 000000
ДЕНЬ ВЫПЛАТЫ: 00

N П/П	НОМЕР ПЕНСИОНЕРА	ФАМИЛИЯ , ИМЯ , ОТЧЕСТВО ПОЛУЧАТЕЛЯ	АДРЕС: УЛИЦА	АДРЕС: N ДОМА	АДРЕС: N КРП	АДРЕС: N КВАР.	К ВЫПЛ. ПО ВЕДОМОСТИ: СУММА	К ВЫПЛ. ПО ВЕДОМОСТИ: С КАКОГО	К ВЫПЛ. ПО ВЕДОМОСТИ: ПО КАКОЕ	ШИФР КОМПЛЕКТА	ОСНОВАНИЕ: N ФОРМЫ
1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
1	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SR	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US
2	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SR	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US

00/04/00 ПОДСИСТЕМА «В Ы П Л А Т А» ф.77005 СТР.001

ПЕРЕЧЕНЬ ИНФОРМАЦИИ,УЧТЕННОЙ ПРИ РАСЧЕТЕ НА ЭВМ СУММ ПЕНСИЙ (ПОСОБИЙ) ПО ВЕДОМОСТЯМ.

ЗОНА: 1
ПОЧТОВЫЙ ИНДЕКС: 000000
ДЕНЬ ВЫПЛАТЫ: 00

N П/П	НОМЕР ПЕНСИОНЕРА	ФАМИЛИЯ , ИМЯ , ОТЧЕСТВО ПОЛУЧАТЕЛЯ	АДРЕС: УЛИЦА	АДРЕС: N ДОМА	АДРЕС: N КРП	АДРЕС: N КВАР.	К ВЫПЛ. ПО ВЕДОМОСТИ: СУММА	К ВЫПЛ. ПО ВЕДОМОСТИ: С КАКОГО	К ВЫПЛ. ПО ВЕДОМОСТИ: ПО КАКОЕ	ШИФР КОМПЛЕКТА	ОСНОВАНИЕ: N ФОРМЫ
1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
1	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SR	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US
2	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SR	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US

00/04/00 ПОДСИСТЕМА «В Ы П Л А Т А» ф.77705 СТР.001

ПЕРЕЧЕНЬ ИНФОРМАЦИИ,УЧТЕННОЙ ПРИ РАСЧЕТЕ НА ЭВМ СУММ ПЕНСИЙ (ПОСОБИЙ) ПО ВЕДОМОСТЯМ.

ЗОНА: 0
ПОЧТОВЫЙ ИНДЕКС: 000000
ДЕНЬ ВЫПЛАТЫ: 01

N П/П	НОМЕР ПЕНСИОНЕРА	ФАМИЛИЯ , ИМЯ , ОТЧЕСТВО ПОЛУЧАТЕЛЯ	СВЕДЕНИЯ ИЗ ОСНОВНОГО ИНФОРМАЦИОННОГО МАССИВА: АДРЕС: УЛИЦА	N ДОМА	N КРП	N КВАР.	К ВЫПЛ. ПО ВЕДОМОСТИ: СУММА	С КАКОГО	ПО КАКОЕ	ШИФР КОМПЛЕКТА	ОСНОВАНИЕ: N ФОРМЫ
1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
1	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SR	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US
2	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SP	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US

00/04/00 ПОДСИСТЕМА «В Ы П Л А Т А» ф.77705 СТР.001

ПЕРЕЧЕНЬ ИНФОРМАЦИИ,УЧТЕННОЙ ПРИ РАСЧЕТЕ НА ЭВМ СУММ ПЕНСИЙ (ПОСОБИЙ) ПО ВЕДОМОСТЯМ.

ЗОНА: 0
ПОЧТОВЫЙ ИНДЕКС: 000000
ДЕНЬ ВЫПЛАТЫ: 00

N П/П	НОМЕР ПЕНСИОНЕРА	ФАМИЛИЯ , ИМЯ , ОТЧЕСТВО ПОЛУЧАТЕЛЯ	СВЕДЕНИЯ ИЗ ОСНОВНОГО ИНФОРМАЦИОННОГО МАССИВА: АДРЕС: УЛИЦА	N ДОМА	N КРП	N КВАР.	К ВЫПЛ. ПО ВЕДОМОСТИ: СУММА	С КАКОГО	ПО КАКОЕ	ШИФР КОМПЛЕКТА	ОСНОВАНИЕ: N ФОРМЫ
1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
1	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SP	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US
2	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SP	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US
3	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SP	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US
4	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SP	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US
5	USRSUUSRSUUS	RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS	RSUUSRSUUSRSUUS	RSUU	SR	SUUS	RSU,US	RSUUSR	SUUSRS	RSUUSRSU	US

00/04/00 ПОДСИСТЕМА «В Ы П Л А Т А» Ф.77П05 СТР.001

ПЕРЕЧЕНЬ ИНФОРМАЦИИ,УЧТЕННОЙ ПРИ РАСЧЕТЕ НА ЭВМ СУММ ПЕНСИЙ (ПОСОБИЙ) ПО ВЕДОМОСТЯМ.

ЗОНА: 1
ПОЧТОВЫЙ ИНДЕКС: 000000
ДЕНЬ ВЫПЛАТЫ: 01

N П/П	НОМЕР ПЕНСИОНЕРА	ФАМИЛИЯ , ИМЯ , ОТЧЕСТВО ПОЛУЧАТЕЛЯ	СВЕДЕНИЯ ИЗ ОСНОВНОГО ИНФОРМАЦИОННОГО МАССИВА: АДРЕС: УЛИЦА	№ ДОМА	№ КРП	№ КВАР.	К ВЫПЛ. ПО ВЕДОМОСТИ: СУММА	С КАКОГО	ПО КАКОЕ	ОСНОВАНИЕ: ШИФР КОМПЛЕКТА	№ ФОРМЫ
1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.

, USRSUUSRSUUS RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS RSUUSRSUUSRSUUS RSUU SR SUUS RSU,US RSUUSR SUUSRS RSUUSRSU US

00/04/00 ПОДСИСТЕМА «В Ы П Л А Т А» Ф.77П05 СТР.001

ПЕРЕЧЕНЬ ИНФОРМАЦИИ,УЧТЕННОЙ ПРИ РАСЧЕТЕ НА ЭВМ СУММ ПЕНСИЙ (ПОСОБИЙ) ПО ВЕДОМОСТЯМ.

ЗОНА: 0
ПОЧТОВЫЙ ИНДЕКС: 000000
ДЕНЬ ВЫПЛАТЫ: 01

N П/П	НОМЕР ПЕНСИОНЕРА	ФАМИЛИЯ , ИМЯ , ОТЧЕСТВО ПОЛУЧАТЕЛЯ	СВЕДЕНИЯ ИЗ ОСНОВНОГО ИНФОРМАЦИОННОГО МАССИВА: АДРЕС: УЛИЦА	№ ДОМА	№ КРП	№ КВАР.	К ВЫПЛ. ПО ВЕДОМОСТИ: СУММА	С КАКОГО	ПО КАКОЕ	ОСНОВАНИЕ: ШИФР КОМПЛЕКТА	№ ФОРМЫ
1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.

, USRSUUSRSUUS RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS RSUUSRSUUSRSUUS RSUU SR SUUS RSU,US RSUUSR SUUSRS RSUUSRSU US

00/04/00 ПОДСИСТЕМА «В Ы П Л А Т А» Ф.77П05 СТР.001

ПЕРЕЧЕНЬ ИНФОРМАЦИИ,УЧТЕННОЙ ПРИ РАСЧЕТЕ НА ЭВМ СУММ ПЕНСИЙ (ПОСОБИЙ) ПО ВЕДОМОСТЯМ.

ЗОНА: 0
ПОЧТОВЫЙ ИНДЕКС: 000000
ДЕНЬ ВЫПЛАТЫ: 00

	СВЕДЕНИЯ ИЗ ОСНОВНОГО ИНФОРМАЦИОННОГО МАССИВА										ОСНОВАНИЕ	
N П/П	НОМЕР ПЕНСИОНЕРА	ФАМИЛИЯ , ИМЯ , ОТЧЕСТВО ПОЛУЧАТЕЛЯ	АДРЕС				К ВЫПЛ. ПО ВЕДОМОСТИ				ШИФР КОМПЛЕКТА	N ФОРМЫ
			УЛИЦА	N ДОМА	N КРП	N КВАР.	СУММА	С КАКОГО	ПО КАКОЕ			
1.	2.	3.	4.	5.	6.	7.	8.	9.	10.		11.	12.
, USRSUUSRSUUS		RSUUSRSUUSRSUUSRSUUSRSUUSRSUUS		RSUUSRSUUSRSUUS		RSUU	SR	SUUS RSU,US		RSUUSR SUUSRS RSUUSRSU US		

77/04/25 ПОДСИСТЕМА «В Ы П Л А Т А» Ф.77706 СТР.001

ПЕРЕЧЕНЬ ОБРАБОТАННЫХ ДОКУМЕНТОВ ПО УЧЁТУ ОБСЛУЖИВАЕМЫХ ЛИЦ.

КОМПЛЕКТ: 00000001

N П/П	N ФОРМЫ	НОМЕР ПЕНСИОНЕРА	ПОЧТОВЫЙ ИНДЕКС	ВИД ПЕНСИИ	N ОШИБКИ
1.	2.	3.	4.	5.	6.
1	US	RSUUSRSUUSRS	UUSRSU	USRSUUSR	SU

77/04/25 ПОДСИСТЕМА «В Ы П Л А Т А» Ф.77706 СТР.001

ПЕРЕЧЕНЬ ОБРАБОТАННЫХ ДОКУМЕНТОВ ПО УЧЁТУ ОБСЛУЖИВАЕМЫХ ЛИЦ.

КОМПЛЕКТ: 00000000

N П/П	N ФОРМЫ	НОМЕР ПЕНСИОНЕРА	ПОЧТОВЫЙ ИНДЕКС	ВИД ПЕНСИИ	N ОШИБКИ
1.	2.	3.	4.	5.	6.
1	US	RSUUSRSUUSRS	UUSRSU	USRSUUSR	SU
2	US	RSUUSRSUUSRS	UUSRSU	USRSUUSR	SU

```
*AAAAAКОРЕК КОНЕЦ Т =00.02
*Т=I3.28
*2.AAAAACMOTЛ МЛ 004-004
ЗГР:УСТАНОВИТЬ ВОП I9
*2
*2.AAAAACMOTЛ МЛ  006-006
 УСТ МЛ ДЛЯ ВХ
*2
*2.AAAAACMOTЛ МЛ I45-002
УСТ МЛ МП
*2
*Т=I3.38
*ЛAAAAACMOTЛ
ПРОГРАММА ВОП I9
*2.AAAAACMOTЛ МЛ 006-006
 УСТ МЛ С ВХОДН А
*2
*AAAAACMOTЛ МЛ 006-006
А СНЯТЬ МЛ
*AAAAACMOTЛ
РЕЗУЛЬТАТЫ
*2.AAAAACMOTЛ
УСТ РЕАЛЬНЫЙ ПРИМЕР
*2-*****
*AAAAACMOTЛ КОНЕЦ Т=00.08
```

тывает первый символ очередной записи массива А и засылает в реквизит МАК. Затем в строке 01330 значение этого реквизита сравнивается с единицей и т.д. Разъясним еще несколько часто употребляемых операторов: ПТ - Переслать Текст в строке 01440 в очередную запись массива П, начиная с 110 символа засылает текст "СТР", оператор ЗЦС в строке 01460 засылает значение реквизита Г в очередную запись массива П, начиная с 9 символа, оператор ПЧ выдает сформированную запись на печать, а оператор А - Арифметический в строке 01530 увеличивает значение реквизита ДЛ на три.

Важно подчеркнуть, что описанные выше действия можно описать также на языке КОБОЛ, притом примерно таким же количеством операторов. Например, оператору СМОДа ВЗ соответствует оператор КОБОЛа ЧИТАТЬ, операторам ПЛ и ПТ - ПОМЕСТИТЬ или же описание данного с фразой ЗНАЧЕНИЕ, оператору Е - ЕСЛИ, ПЧ - ПИСАТЬ или ВЫДАТЬ, ЧРЦЦ - ПОМЕСТИТЬ и т.д.

Переходим к рассмотрению работы системы СМОТЛ на программе ВОМ19. СМОТЛ управляется заказом, который приведен перед текстом программы на странице 115. Оператор заказа ИДЕНТ указывает имя исходного текста (ВОМ19), имя рабочей программы, получаемой после трансляции (ВОП19), и имя массива примеров, создаваемого системой СМОТЛ (ВОР19).Оператор ЛЕНТЫ указывает, что исходный текст, оттранслированная программа и массив примеров размещаются на одной и той же магнитной ленте (МЛ) - ПРОГР. Оператор ПСПСМ указывает, что должна строиться ПСП, ПЧПРИ - должны распечатываться построенные примеры, ВЫПОЛ - оттранслированная программа должна запускаться на построенных

примерах, ПЧПРО - должно распечатываться текст программы, ТЕКСТ - исходный текст уже записан на МД ПРОГР.

Далее, на страницах II5-I2I распечатан текст программы, а на страницах I22 и I23 пример, построенный системой СМОТЛ (в данном случае построен всего лишь один пример). Общее число построенных записей - I7, и они отделены звездочками. В тех местах записей, которые анализируются в логических условиях программы, система СМОТЛ помещала числа, а в тех местах, содержимое которых используется только для печати, помещены буквы " USR ".

После примера, созданного системой СМОТЛ, на страницах I24-I28 следуют протоколы работы программы ВОМI9 на созданном примере. По выданным формам пользователь, знающий решаемую задачу, легко может убедиться в правильности работы программы. В данном случае обнаруживается одна ошибка: дата печати формы Ф.77П05 сформирована неверно (см.верхний левый угол формы!). Для формы Ф.77П06 эта дата сформирована верно.

На странице I29 приведен протокол работы системы СМОТЛ на программе ВОМI9, выдаваемой на пультовой пишущей машинке оператора ЭВМ. В начале этого протокола отпечатано сообщение об окончании работы программы, выполняемой непосредственно перед СМОТЛ. Затем выдано время I3 часов и 28 минут. В данном случае система СМОТЛ свою работу начала в I3 часов 23 минуты (см.страницу II5 верхний правый угол). Затем следуют указания оператору об установке оттранслированной программы и необходимых при ее работе МД. Запуск оттранслированной программы на

примере, построенном системой СМОТЛ, осуществляется в I3 часов 38 минут. Затем следуют сообщения и указания, выдаваемые при работе программы ВОМI9, а в заключении выдано сообщение об окончании работы СМОТЛ. Время работы процессора на программе ВОМI9, включая трансляцию, построение примеров и работу программы на построенном примере, составляет 8 минут, а общее время в однопрограммном режиме ЭВМ "Минск-32" для выполнения указанных выше действий составляет примерно I6 минут.