М. И. Кахро,
А. П. Калья,
Э. Х. Тыугу

Инструментальная система программирования ЕС ЭВМ (ПРИЗ)

МОСКВА «ФИНАНСЫ И СТАТИСТИКА» 1981

ББК 32.973
К30

К $\frac{30502-107}{010(01)-81}$ 85—81(С) 2405000000

ВВЕДЕНИЕ

Понятию «система программирования» трудно дать точное определение. В то же время роль системы программирования как совокупности средств разработки программ ЭВМ постоянно возрастает. С увеличением быстродействия вычислительных машин появляется возможность все в большей степени автоматизировать программирование. В настоящее время получили распространение системы программирования высокого уровня, в которых ЭВМ выступает как интеллектуальный партнер программиста. К таким системам принадлежит и ПРИЗ ЕС, отличающаяся от большинства других систем программирования высокого уровня тем, что является промышленной, а не экспериментальной системой. Данная система, имеющая прототипы: системы СМП и ПРИЗ-32, апробированные на ЭВМ типа «Минск», прошла достаточно длительную эксплуатацию и в настоящее время распространяется научно-производственным объединением «Алгоритм» как компонент математического обеспечения ЕС ЭВМ.

В книге описываются возможности и концепции системы ПРИЗ, ее входной язык и технология программирования в данной системе. Книга рассчитана в первую очередь на программистов, занимающихся разработкой специализированных языков программирования и трансляторов, пакетов программ и крупных программных комплексов. Первая глава, в которой рассматриваются основные понятия, назначение и возможности системы, а также последняя глава, в которой приводится пример пакета программ, представляют интерес также для специалистов прикладных областей, интересующихся построением пакетов программ.

Книга содержит также материал, посвященный реализации новых концепций. Поскольку в данной системе впервые широко применен автоматический синтез программ, описание некоторых принципов реализации может оказаться интересным для системных программистов и специалистов по трансляторам.

Авторы пытались изложить материал книги так, чтобы для чтения не требовалось специальных знаний по системному программированию.

1-я глава написана Э. Тыугу, 3-я — М. Кахро, 5-я — А. Калья, остальной материал книги является совместной работой авторов.

ГЛАВА 1
НАЗНАЧЕНИЕ СИСТЕМЫ. ОСНОВНЫЕ ПОНЯТИЯ

Место системы ПРИЗ в СМО ЕС ЭВМ. Система математического обеспечения (СМО) ЕС ЭВМ является чрезвычайно крупным комплексом программ, предназначенным для эксплуатации ЭВМ и разработки новых программ. Основной принцип при разработке программ — *принцип модульности* — позволяет составлять и обрабатывать независимо друг от друга куски программ необходимого размера — модули.

Можно сказать, что СМО ЕС ЭВМ поддерживает технологию модульного программирования. Работа над построением программы, по существу, является работой над ее модулями. Каждый из них проходит определенные этапы обработки, ввод, генерацию текста, текстовое редактирование, трансляцию, редактирование связей, загрузку, пуск. При этом меняется форма модуля, который превращается из исходного модуля, записанного на некотором входном языке, в объектный модуль (результат трансляции) и, наконец, в загрузочный модуль, готовый к выполнению.

ЭВМ Единой серии позволяют накапливать практически неограниченное количество модулей в библиотеках на магнитных дисках. Но при этом становится все труднее разобраться в большом числе модулей, выбрать нужный из них и задать информацию, необходимую для объединения их в единую программу. Кроме того, при построении новой программы из готовых модулей практически всегда приходится создавать некоторое число новых, часто весьма маленьких кусочков программ, т. е. модулей для стыковки уже существующих. Такие модули связи могут потребоваться для передачи данных, преобразования форматов, преобразования единиц измерений и т. д. Система программирования ПРИЗ содержит средства автоматизации выборки модулей и автоматического построения модулей для связывания других.

Важным свойством данной системы является полная совместимость с обрабатывающими программами ОС ЕС, с трансляторами с Фортрана, Кобола и с макроассемблером. Заметим, что совместимость понимается в данном случае так, что система ПРИЗ не накладывает дополнительных ограничений на модули. Ее можно считать расширением ОС ЕС, дающим новые средства для обработки модулей.

Связь системы ПРИЗ с компонентами ОС ЕС и способы ее применения показаны на рис. 1. Программирование на Фортране или на языке Ассемблера (линии 7 и 6 на рис. 1) требует использования редактора связей для получения загрузочного модуля и операционной системы для пуска программ. Программирование с при-

менением системы ПРИЗ требует всегда еще дополнительно использование языка Ассемблера для получения объектных модулей. При этом возможны варианты совместного использования системы программирования:

программирование на Фортране с использованием расширений языка в виде операторов задач (8);

программирование полностью на входном языке системы ПРИЗ (1);

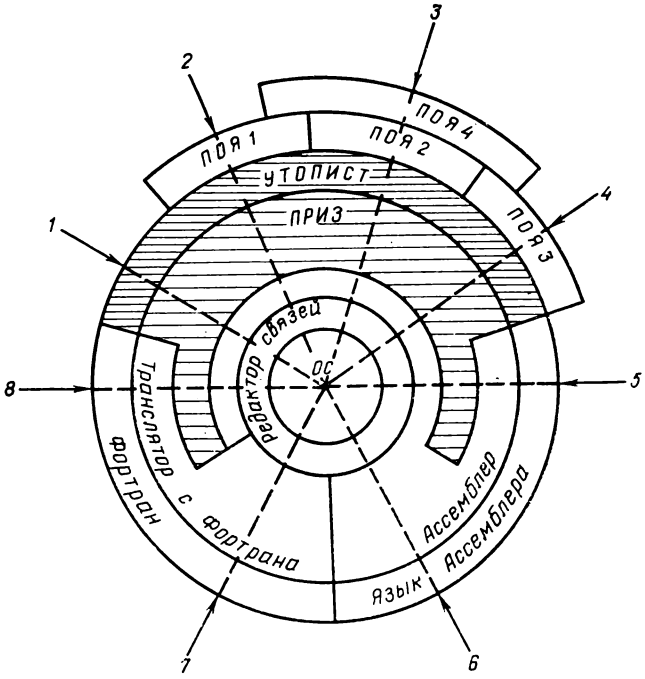


Рис. 1. Связь системы ПРИЗ с СМО ЕС ЭВМ

решение задач, записанных на входных языках пакетов (ПОЯ) прикладных программ, которые построены с помощью системы ПРИЗ (2—4);

программирование на языке Ассемблера с использованием расширений языка в виде операторов задачи (5).

Допускается также такое применение системы, где часть модулей записана на Фортране, часть — на языке Ассемблера, а часть — на Коболе.

Совместимость с языком ПЛ/1 в системе ПРИЗ специально не поддерживается, так как транслятор с этого языка выдает объектные модули, трудно объединяемые с модулями, полученными из других языков. Но пользуясь приемами, описанными в инструкциях по программированию на ПЛ/1, можно использовать модули, полученные с этого языка.

Если для объединения модулей в единую программу в систе-

мах программирования ОС ЕС необходимо их явно указывать в вызове подпрограммы или макрооператоре, то система ПРИЗ позволяет во многих случаях отказаться от явного указания модулей, которые должны быть собраны в окончательную рабочую программу. Поэтому пользователь системы в значительной степени освобождается от программирования, но его обязанностью является описание условий задачи, по которым далее автоматически синтезируется программа. Очевидно, что правильность программы определяется теперь только правильностью задания условий задачи, так как автоматически работающий синтезатор программ свободен от ошибок. Это значительно повышает производительность и качество труда программистов, т. е. повышает технологический уровень разработки программ для ЕС ЭВМ.

ПРИЗ как система построения трансляторов. В 1968 г. Д. Кнут предложил представлять семантику языков программирования атрибутами, приписываемыми символам языка, и отношениями, задаваемыми между атрибутами [1]. Этот метод, называемый атрибутным, является в настоящее время наиболее распространенным методом автоматизации построения семантической части трансляторов. Однако на практике он не обеспечивает полной автоматизации построения транслятора по описанию семантики. Во-первых, еще недостаточно широко распространен опыт определения подходящих атрибутов и отношений для всех конструкций языков программирования. Во-вторых, сама атрибутная техника имеет некоторые трудности при организации вычисления значений атрибутов [2].

Система программирования ПРИЗ позволяет также определять семантику понятий (т. е. нетерминальных символов) новых языков путем описания соответствующих атрибутов, называемых их компонентами, и отношений между компонентами, позволяющих организовать вычисления последних.

В этом смысле существует полная аналогия между атрибутной техникой, предложенной Д. Кнутом, и техникой семантических моделей, применяемой в ПРИЗе. Основное отличие заключается в том, что система ПРИЗ применима для создания новых проблемно-ориентированных языков как расширения одного базового языка — УТОПИСТ. В отличие от большинства систем построения трансляторов в ней вместо синтаксически управляемого анализа для синтаксической обработки текста применяется техника макрогенерации. Основными достоинствами системы являются мощные и удобные средства семантической обработки и простые в использовании синтаксические средства. Это дает возможность чрезвычайно быстро создавать трансляторы для многих простых языков.

ПРИЗ как система построения пакетов программ. Разработка пакета программ настолько сильно отличается от разработки отдельной программы, что опыт программистов, накопленный при решении прикладных задач, явно недостаточен для построения хорошо функционирующего пакета.

В построении пакета должны принимать участие как специа-

листы по предметной области, так и системные программисты. Специалисты по предметной области пакета ответственны за правильный выбор понятий входного языка пакета и за выбор вычислительных методов, которые принимаются за основу при программировании модулей пакета. Деятельность специалистов по предметной области при построении пакета принято называть *модульным анализом* — это первый этап построения программ. Результатом модульного анализа может быть проект пакета программ.

Качественную реализацию проекта пакета осуществляют системные программисты, которые должны разрабатываемый пакет в конечном счете выдать в виде готового программного продукта, к качеству которого предъявляются весьма высокие требования.

Нередки случаи, когда за разработку пакета программ принимаются прикладные специалисты без участия системных программистов. В этом случае следует опасаться того, что пакет программ примет форму частично работающего макета задуманной системы. Разработка готового пакета программ отодвинется на неопределенное время из-за непредвиденных трудностей, возникающих на завершающем этапе работ. Эти же моменты наблюдаются, когда пакет программ разрабатывается системными программистами без участия достаточно квалифицированных специалистов прикладной предметной области. В этом случае может возникнуть опасность, что класс решаемых задач, выбранный создателями пакета, не будет удовлетворять пользователей пакета, а методы решения задач окажутся недостаточно эффективными и ограниченными по применимости.

Разработку качественного пакета может обеспечить только четкая технология работ, поддерживаемая соответствующим программным обеспечением. Система ПРИЗ является системой построения пакетов программ, поддерживающей все основные этапы разработки пакетов:

модульный анализ;

программирование, отладку и испытание программых модулей, входящих в пакет;

создание и реализацию входного языка;

разработку управляющей части, т. е. организующей программы пакета;

обеспечение услуг, необходимых для работы с пакетом (связь с базами данных, ввод-вывод, диалог).

Для модульного анализа в системе ПРИЗ имеются средства формализации описания моделей систем и процессов из самых разных областей. Обеспечена возможность автоматической обработки и хранения формализованных описаний.

Для второго этапа — программирования модулей — применимы Кобол, Фортран и язык макроассемблера, расширенные возможностями включения автоматически синтезируемых частей программ.

Все же эти два этапа меньше всего поддаются автоматизации, так как здесь требуются знания конкретной предметной области,

и основная трудность заключается не в машинной реализации, а в принятии правильных содержательных решений.

Остальные этапы, начиная с реализации входного языка, в системе ПРИЗ автоматизированы в очень высокой степени. Технология работ по построению пакетов будет изложена в гл. 4.

ПРИЗ как система искусственного интеллекта. Во второй половине 70-х годов наметилась явная тенденция использования методов искусственного интеллекта в практических применениях.

В трансляторах и системах программирования перспективным является применение этих методов для повышения уровня языка, на котором общаются с ЭВМ. Это относится как к языкам программирования, так и к специализированным языкам описания задач.

В системе ПРИЗ использованы следующие приемы из области искусственного интеллекта.

1. Семантика входного языка описывается семантическими моделями, содержащими объекты и отношения. Обработка входного текста — перевод его в семантическое представление и работа с семантическими моделями.

2. Создана очень удобная возможность расширения входного языка путем описания новых понятий через существующие таким же способом, как это делается в других системах искусственного интеллекта. Для этого используется семантическая память и средства работы с семантической памятью.

Фактически организована работа с фреймами понятий, т. е. с такими семантическими моделями, в которых автоматически осуществляются существенные изменения для настройки на конкретную ситуацию (использование объектов типа НЕОПР') и которые позволяют автоматически (без явного вызова) выполнять необходимые процедуры.

3. Реализован автоматический синтез программ и созданы возможности решения задач по их заданным условиям.

С другой стороны, практика показала, что система ПРИЗ хорошо применима для программирования задач искусственного интеллекта. В частности, при ее использовании можно за короткий срок ставить такие машинные эксперименты по формированию понятий, обучению, использованию нечетких понятий и т. д., которые ранее требовали значительно больше времени.

Технологические услуги системы. Кроме основных функций, выполняемых системой ПРИЗ, пользователь может получить от нее следующие услуги.

1. Программы виртуальной памяти, которые можно использовать в модулях, написанных на языке Фортран или на языке Ассемблера, обеспечивают работу с математической памятью большого объема, выполняя автоматически замену страниц с дисковой памяти.

2. Диалоговый режим трансляции и генерации программ, обеспечивающий непосредственный контакт пользователя с системой.

3. Директивы работы с семантической памятью для более удоб-

ной работы с библиотекой моделей, чем средства работы с библиотечными файлами.

4. Информирующую программу HELP, предназначенную для получения оперативной информации о расширениях языка УТОПИСТ.

5. Отладочные средства высокого уровня, обеспечивающие автоматизацию комплексной отладки синтезированных программ, с учетом возможных ошибок в модулях и описаниях условий задачи.

6. Набор каталогизированных процедур, обеспечивающих работу в разных режимах: пакетном и диалоговом; в режиме расширения и изменения модели предметной области; в режиме макрогенерации; в режиме решения задач, написанных на входных языках пакетов; в режиме решения задач, описание которых включено в язык Фортран или язык Ассемблера, и др.

7. Специальные утилиты, обеспечивающие удобное обслуживание библиотек семантических моделей и макропроцессора системы ПРИЗ.

8. Пакет прикладных программ для систем управления базами данных, совместимый с другими пакетами, созданными в системе ПРИЗ, который дает возможность построения индивидуальных баз данных пользователям системы ПРИЗ.

9. Генератор диалоговых программ, позволяющий генерировать диалоговые модули, включаемые в пакеты программ.

Оператор задачи и автоматический вызов модулей. Система ПРИЗ является нетрадиционной системой программирования. В ней используется в основном автоматический синтез программ, который дает программисту возможность необыкновенно быстро составлять крупные и совершенно правильные программы. Система программирования ПРИЗ позволяет в тексте программы на входном языке описывать задачи, не задавая явно алгоритма решения задачи. Описание каждой задачи задается в виде *оператора задачи.* Последний является семантически правильным только в том случае, если задача разрешима.

Для решения семантически правильной задачи, описанной оператором задачи, система автоматически синтезирует алгоритм и оформляет его на языке Ассемблера в виде модуля решения задачи.

Оператор задачи имеет следующую форму:

НА' Z ВЫЧИСЛИТЬ' $Y_1, \ldots, Y_м$ ПО' $X_1, \ldots, X_к$,

где $X_1, \ldots, X_к, Y_1, \ldots, Y_м$ — имена понятий, описанных семантической моделью с именем Z. Семантическая модель с именем Z называется моделью задачи;

$X_1, X_2, \ldots, X_к$ — входы (входные переменные) задачи;

$Y_1, Y_2, \ldots, Y_м$ — выходы (выходные переменные) задачи.

Если список входных переменных задачи пустой, то допускается применять сокращенную форму оператора постановки задачи

НА' Z ВЫЧИСЛИТЬ' $Y_1, \ldots, Y_м$

При выполнении программы, исходный текст которой содержит оператор задачи, происходит обращение к автоматически составленному модулю решения задачи. Модуль решения задачи в свою очередь содержит обращения к другим модулям, которые нужны в ходе решения задачи, такие модули называются автоматически вызываемыми модулями.

Автоматически вызываемыми модулями могут быть:

подпрограммы, доступные редактору связей, собирающему рабочую программу; обращением к данному модулю является вызов подпрограммы ВЫЗОВ' $P(T_1, \ldots, T_N)$;

макроопределения, которые должны быть доступны Ассемблеру, переводящему текст модуля решения задачи; обращением к данному модулю является макровызов.

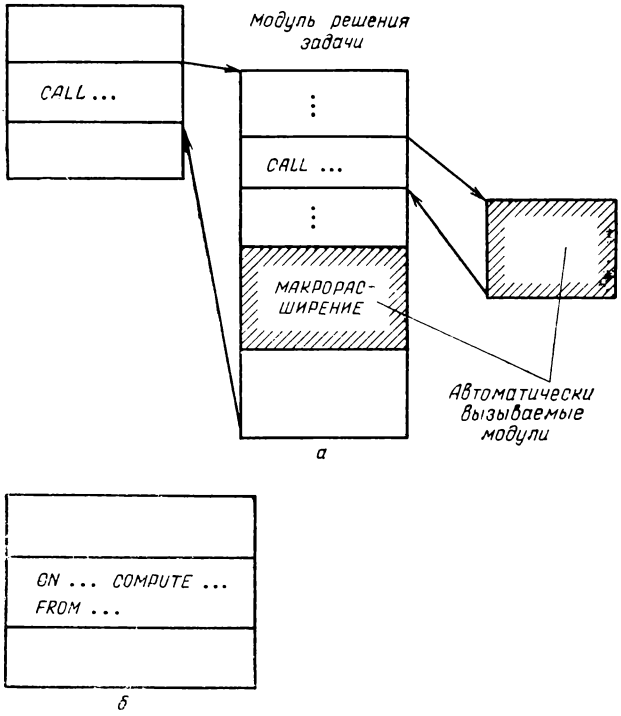


Рис. 2. Структура автоматически построенной программы

На рис. 2, *а* показана структура рабочей программы, построенной по исходной программе, содержащей один оператор постановки задачи (рис. 2, *б*). В готовой рабочей программе макровызовы заменены макрорасширениями, вставленными в модуль решения задачи.

Семантическая память. Модели, на которых операторами задач описываются задачи, хранятся в семантической памяти, являющейся некоторым файлом на диске. Семантическая память имеет структуру в виде дерева, в вершинах которого расположены понятия. В терминальных вершинах находятся элементарные понятия, например на рис. 3 такими являются ПЛОЩАДЬ, ШИРОТА, ДОЛГОТА.

Вершины, которые непосредственно подчинены некоторой вершине, содержат понятия, называемые непосредственными компонентами последней. Для вершины ГОРОД (см. рис. 3) такими являются, например, СВЯЗЬ, ТРАНСПОРТ, КООРДИНАТЫ.

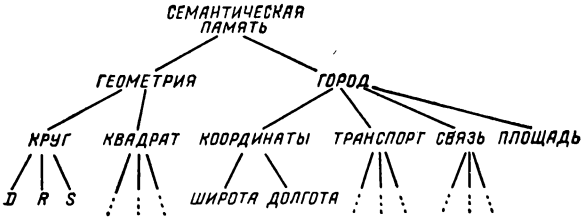


Рис. 3. Структура семантической памяти

Подчиненными понятиями или просто компонентами некоторого понятия служат понятия всех вершин поддерева, корнем которого оно является. Каждое понятие в семантической памяти имеет описание — семантическую модель, куда входят все его компоненты со своими семантическими моделями, а также отношения между понятиями.

Для работы с семантической памятью имеются директивы ВВЕСТИ, УДАЛИТЬ, ПО, которые позволяют вводить новые понятия в семантическую память, удалять оттуда понятия или выделять для работы некоторую часть семантической памяти. Понятно, что введение или исключение понятий меняет семантические модели всех тех понятий, которым вводимые (исключаемые) понятия подчиняются.

Для указания конкретного понятия используется его имя. Именами понятий самого высокого уровня являются идентификаторы, например ГОРОД, ГЕОМЕТРИЯ (см. рис. 3). В общем случае имя понятия состоит из имени понятия, которому оно непосредственно подчиняется, и некоторого идентификатора, приписанного к этому имени через точку. Так образуются составные имена: ГОРОД.ТРАНСПОРТ, ГОРОД.КООРДИНАТЫ.ДОЛГОТА и др.

Имена всех понятий в семантической памяти должны быть разные. Поэтому идентификаторы непосредственных компонентов понятия должны быть между собой различны. Мы будем также считать, что семантическая модель понятия имеет то же самое имя, что и само понятие.

Семантические модели. Для синтеза модуля решения задачи необходима информация о содержании задачи. Такая информация

содержится в семантической модели, имя которой задается в операторе задачи. Семантическая модель содержит переменные и отношения.

Переменная обладает именем и видом. Переменные семантической модели соответствуют содержательным понятиям той предметной области, из которой задача решается. Переменными семантической модели являются понятия, подчиненные понятию, описываемому данной моделью в семантической памяти.

Отношение выражает связь между значениями переменных и может использоваться для вычисления значений некоторых из них по заданным значениям других переменных. Отношение может быть задано:

программным модулем;

уравнением, заданным на входном языке системы;

семантической моделью;

описанием структуры данных, заданным на входном языке системы;

макроопределением.

Пример. Пусть имеются переменные I1, U1, R1, I2, U2, R2, I3, U3, R3, которые принимают вещественные значения, и имеются два модуля A и B.

Модуль A имеет параметры I, U, R и описывает следующие три алгоритма:

1) I : = U/R;
2) U : = I * R;
3) R : = U/I;

Модуль B имеет параметры I1, I2, I3 и описывает следующие алгоритмы:

1) I1 : = —I2—I3;
2) I2 : = —I1—I3;
3) I3 : = —I1—I2.

Предполагается, что для того и другого модуля можно при выполнении вычислений определить, какой из алгоритмов следует применять. Такие средства предоставлены пользователю системы ПРИЗ в виде стандартных функций, позволяющих опросить состояние системы.

Пользуясь модулем A, опишем теперь три отношения Q1, Q2, Q3, первое из которых связывает переменные I1, U1, R1, второе — I2, U2, R2 и третье — I3, U3, R3.

Установим также между параметрами модуля A и связанными переменными отношений следующие соответствия:

Q1:	U ... U1	Q2:	U ... U2	Q3:	U ... U3
	I ... I1		I I2		I ... I3
	R ... R1		R ... R2		R ... R3

Модулем B опишем отношение Q4 между переменными I1, I2, I3, установив между параметрами модуля B и этими переменными следующее соответствие:

Q4: I1 ... I1
I2 ... I2
I3 ... I3

Отношения Q1, Q2, Q3, Q4 задают потенциальную возможность вычисления значений переменных.

На семантической модели, содержащей N переменных, может быть описано 2^{2N} разных задач. Это все те задачи, у которых в качестве входных и выходных переменных выступают переменные из данной модели. Некоторые из задач могут оказаться разрешимыми. Такой будет, например, задача

НА' Z ВЫЧИСЛИТЬ' I3 ПО' R1, R2, R3, U1, U2.

Для решения задачи, заданной оператором постановки задачи, система ПРИЗ синтезирует алгоритм ее решения и оформляет этот алгоритм в виде модуля решения задачи.

Ниже описаны две формы представления семантических моделей: формульная и графическая. Формульное представление достаточно компактное и вполне строгое. Но графическое представление настолько наглядное, что оно оказывается более удобным во многих случаях при исследовании решения задачи.

Пусть x, y, ... — конечные множества переменных или отдельные переменные. Обозначим через $x \mapsto y$ то, что, зная переменные x, можно вычислить переменные y, используя для этого программу по имени f, например, так:

$$y := f(x).$$

Или учитывая, что y — множество и оно состоит из конечного числа элементов y_i, то $y_i = f_i(x)$ для всех y_i, где все f_i опять определены в программе f.

Обозначим через $P(w) => x \underset{f}{\mapsto} y$ то, что

задана программа, по которой можно для любых значений переменных w вычислить истинно или ложно условие $P(w)$;

из x можно путем применения f вычислить y, если $P(w)$ истинно.

Обозначим через $solv\ (x, y, M)$ то, что, используя семантическую модель с именем M, имеют способ построения программы, вычисляющей из x значения y по заданным значениям x. Теперь формула $solv\ (u, v, M) => x \underset{f}{\mapsto} y$ будет обозначать, что из x можно вычислить y, если можно составить программу, которая из u вычисляет v.

Формулы вида

$$x \underset{f}{\mapsto} y$$

$$P(w) => x \underset{f}{\mapsto} y$$

$$solv\ (u, v, M) => x \underset{f}{\mapsto} y$$

называются предложениями вычислимости.

Будем иногда опускать обозначение программы f в $x \underset{f}{\mapsto} y$, если несущественно знание того, какая программа используется.

Формульным представлением семантической модели является совокупность предложений вычислимости, выражающая возможности вычисления по отношениям семантической модели.

В графическом представлении семантической модели изображается отдельно каждая переменная и каждое отношение. Переменная соединяется с теми отношениями, которые ее вычисляют, используют для вычислений или на применимость которых переменная влияет (через условие P).

Хорошо известными примерами графического представления семантических моделей являются схемы информационных потоков, например приведенная на рис. 4, *а* схема обновления файла. Переменными здесь являются ТЕКСТ, F1, F2, СООБЩЕНИЯ и ДОБАВКА. Отношения — чтение и слияние.

Если нет надобности указывать физический носитель значений переменных, переменные будем обозначать точками, отношения — кружочками. Условимся также, что связь переменной с отношением будем обозначать по-разному, в зависимости от того, какие возможности вычислений по отношению существуют. Различаются:

входные переменные, значения которых должны быть заданы;

выходные переменные, значения которых всегда вычисляются;

слабо связанные переменные, значения которых иногда задаются, иногда могут вычисляться;

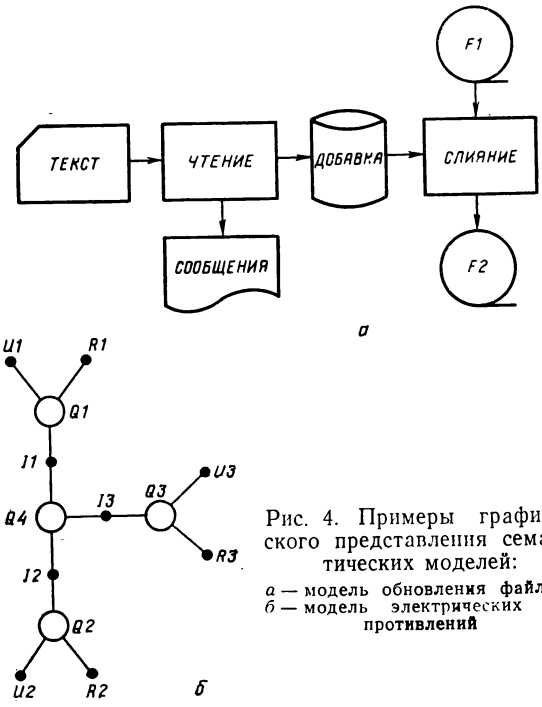


Рис. 4. Примеры графического представления семантических моделей:

а — модель обновления файлов; *б* — модель электрических сопротивлений

сильно связанные переменные, значения которых меняются отношением;

определяющие переменные, значения которых определяют применимость отношения.

На рис. 5 показаны обозначения этих типов связи переменных с отношениями.

На рис. 4, *б* приводится графическое представление семантической модели, описанной выше.

Элементы синтеза программ. Путь от задачи к программе, составляемой автоматически, содержит следующие этапы:

составление формального описания задачи;

доказательство существования решения задачи;

извлечение программы из полученного доказательства.

Если два последних этапа выполнены автоматически, то можно быть уверенным в том, что программа точно соответствует описанию задачи, т. е. она в этом смысле правильная. Но само описание задачи задается человеком, и программист вместо отладки программ должен заниматься отладкой описаний задач. Эта деятельность в каком-то смысле проще. По крайней мере программист не зависит от машинно-зависимых процессов вычислений.

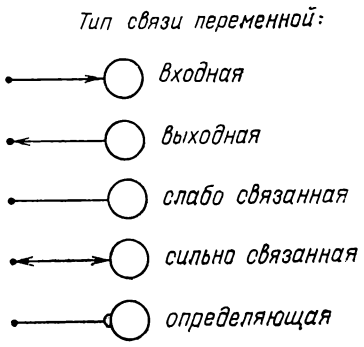


Рис. 5. Типы связи переменных

Будем исходить из того, что описание задачи задано в виде

НА' M ВЫЧИСЛИТЬ' v ПО' u,

и это описание правильное.

В описании задачи u, v — конечные множества переменных. Рассмотрим, как по формальному представлению семантической модели M найти доказательство того, что $solv(u, v, M)$.

Предположим, что все предложения вычислимости имеют вид $x \mapsto y$. Рассмотрим следующие правила вывода новых предложений вычислимости из существующих:

$$\frac{y \subseteq x}{x \xrightarrow[s_{x,y}]{} y} \qquad (1)$$

$$\frac{x \xrightarrow[f_1]{} y,\ x \xrightarrow[f_2]{} z,\ w = y \cup z}{x \xrightarrow[(f_1; f_2)]{} w} \qquad (2)$$

$$\frac{x \xrightarrow[f_1]{} y,\ y \xrightarrow[f_2]{} z}{x \xrightarrow[(f_1; f_2)]{} z} \qquad (3)$$

В этих правилах над чертой записаны уже известные формулы, под чертой — новое предложение, которое получается как следствие из формул над чертой.

Первое правило определяет, что если что-то входит в число уже вычисленных переменных, то это можно использовать как результат вычисления; второе правило определяет, что результат можно вычислять по частям; третье правило — вычисления можно выполнять последовательно: вначале вычисление некоторого про-

межуточного результата, а затем — из него уже окончательный результат.

Этих правил оказывается достаточно для доказательства $solv(u, v, M)$ для любой задачи, которая вообще может быть решена на семантической модели, представленной только предложениями вычислимости вида $x \underset{f}{\longmapsto} y$. Такими правилами как раз и пользуется синтезатор программ системы ПРИЗ. Если в формульном представлении семантической модели встречаются также формулы $P(w) \Rightarrow x \longmapsto y$, то дополнительно используется правило:

$$\frac{P_1(w) \vee \ldots \vee P_k(w),\ P_1(w) \Rightarrow x \underset{f_1}{\longmapsto} y,\ \ldots,\ P_k(w) \Rightarrow x \underset{f_k}{\longmapsto} y}{x \cup w \longmapsto y}, \qquad (4)$$

где $f = \underline{\text{if}}\, P_1(w)\ \underline{\text{then}}\, f_1\ \underline{\text{elif}} \ldots \underline{\text{elif}}\, P_k(w)\ \underline{\text{then}}\, f_k \underline{\text{fi}}$.

Но, к сожалению, не всегда удается проверить справедливость $P_1(w) \vee \ldots \vee P_k(w)$, поэтому приходится довольствоваться тем, что синтезируется только частичная программа. Такая программа иногда дает ответ, а иногда сообщает, что ответа нет.

Если в формульном представлении семантической модели встречаются также формулы вида

$$solv(u', v', M) \Rightarrow x \longmapsto y,$$

то поиск доказательства несколько усложняется.

Перед тем как использовать $x \longmapsto y$, приходится доказывать $solv(u', v', M)$ (или убедиться, что $solv(u', v', M)$ недоказуемо, и тогда отказаться от применения $x \longmapsto y$).

Таблица 1

Операторы программы, соответствующие правилам вывода

Правило вывода	Программа
$\dfrac{y \subseteq x}{x \longmapsto y}$	$y := \text{select}_{xy}(x)$
$\dfrac{x \underset{f_1}{\longmapsto} y,\ x \underset{f_2}{\longmapsto} z,\ w = y \cup z}{x \underset{f}{\longmapsto} w}$	$y := f_1(x);\ z := f_2(x)$
$\dfrac{x \underset{f_1}{\longmapsto} y,\ y \underset{f_2}{\longmapsto} z}{x \underset{f}{\longmapsto} z}$	$y := f_1(x);\ z := f_2(y)$
$\dfrac{P_1(w) \vee \ldots \vee P_k(w),\ P_1(w) \Rightarrow x \underset{f_1}{\longmapsto} y, \ldots,\ \ldots,\ P_k(w) \Rightarrow x \underset{f_k}{\longmapsto} y}{w \cup x \underset{f}{\longmapsto} y}$	$\underline{\text{if}}\, P_1(w)\ \underline{\text{then}}\, y := f_1(x)$ $\underline{\text{elif}} \ldots$ $\ldots \underline{\text{elif}}\, P_k(w)\ \underline{\text{then}}\, y := f_k(x)$ $\underline{\text{else}}\ \underline{\text{failure}}\ \underline{\text{fi}}$

Предположим, что для заданной задачи доказано $solv(u, v, M)$. Оказывается, что из доказательства очень просто получить желаемую программу, если каждому правилу вывода (1)—(4) сопоставить некоторое правило построения программы (табл. 1).

ГЛАВА 2

ЯЗЫК УТОПИСТ

Язык УТОПИСТ (Универсальные Текстовые ОПИСания Терминов) значительно отличается от распространенных сейчас языков программирования, его основное назначение — описание понятий и задач. Поскольку при описании задачи, хотя и редко, но все же приходится описывать действия, то язык УТОПИСТ имеет процедурную часть, пригодную для описания программ. Но основная его выразительность достигается за счет описаний. В УТОПИСТе описываются объекты. Описание объекта представлено в ЭВМ в виде семантической модели объекта. По существу, единственным различием между понятием и объектом является то, что модель понятия хранится постоянно в семантической памяти, в то время как модель объекта существует только временно, до конца решения задачи. При описании языка будем говорить про объекты, имея в виду, что сказанное справедливо и для понятий.

Язык УТОПИСТ позволяет описывать новые семантические модели, которые могут записываться в семантическую память для дальнейшего применения или использоваться непосредственно в качестве модели задачи.

Пример

```
ПУСТЬ'
        СХЕМА: (I1, I2, I3, U1, U2, U3, R1, R2, R3: ВЕЩ';
                УРАВ' U1=I1*R1;
                УРАВ' U2=I2*R2;
                УРАВ' U3=I3*R3;
                УРАВ' I1+I2+I3=0);
```

Данный текст на языке УТОПИСТ описывает модель, рассмотренную в качестве примера в гл. 1. Здесь отношения заданы уравнениями непосредственно в тексте на языке УТОПИСТ.

Язык УТОПИСТ позволяет описывать задачи, для решения которых должны быть составлены программы.

Пример. При условии что семантическая модель с именем СХЕМА уже имеется в библиотеке, задача на ней опишется следующим текстом:

```
ПУСТЬ'
        СХЕМА 1 : СХЕМА
                  R1 = 16, R2 = 32, R3 = 5.
                  U1 = 50, U2 = —28;
ДЕЙСТВИЯ'
        НА' СХЕМА 1 ВЫЧИСЛИТЬ' U3;
```

За счет того что при трансляции больший объем работы перекладывается на систему, реализующую язык, достигается легкость и удобство работы на языке УТОПИСТ.

Программа, написанная на языке УТОПИСТ, состоит из следующих частей: начала (ПРОГРАММА' идентификатор;), тела, конца (КОНЕЦ' +++).

Тело программы содержит описание данных и описание действий, определенные синтаксически следующим образом:

данные:
    ПУСТЬ' описание; ...
действия:
    { ДЕЙСТВИЯ' оператор; ... }
    { ПОДПРОГРАММА' идентификатор [(имя,...)]; оператор; ... }

Данные задаются в виде одного или нескольких описаний. Это декларативная часть текста программы, в которой описываются условия задачи — структура данных, где компоненты структуры, т. е. объекты, связаны между собой отношениями. Результатом трансляции описания данных является семантическая модель (см. гл. 1), называемая моделью задачи.

Действия — это последовательность операторов, записанная либо после слова ДЕЙСТВИЯ', либо после слова ПОДПРОГРАММА'. В первом случае результатом трансляции описания действий является управляющая программа на языке макроассемблера, а во втором — подпрограмма. Имя подпрограммы и ее формальные параметры, если они имеются, следуют слову ПОДПРОГРАММА'. Очевидно, что в одной программе не разрешается использовать оба варианта.

Так, описанные подпрограммы могут вызываться из программ, написанных на Фортране. Если результатом выполнения подпрограммы является одно значение, то такая подпрограмма может оформляться как функция. Тогда последним оператором программы на языке УТОПИСТ должен быть оператор ЗНАЧЕНИЕ'. Общая форма оператора

ЗНАЧЕНИЕ' имя;

Имя обозначает вещественный объект модели задачи, значение которого вычисляется.

Описания данных и действий могут чередоваться в тексте программы между собой. Кроме того, в текст программы могут вставляться директивы и комментарии. Директивами являются предложения, требующие немедленного выполнения в ходе трансляции; комментариями — строки, в первой позиции которых стоит символ «*». Эти строки на смысл программы не влияют.

Достоинством любого языка программирования является небольшое число основных понятий, из которых строится язык. Язык УТОПИСТ удовлетворяет этому требованию. Структура языка показана на рис. 6. Примеры конструкций языка приведены в приложении 1.

В описании языка выделен лексический уровень, показывающий, как из основных символов строятся лексемы. Правила построения всех лексем приведены в приложении 2. Для понимания описания синтаксиса достаточно знать лишь, что лексемами являются эле-

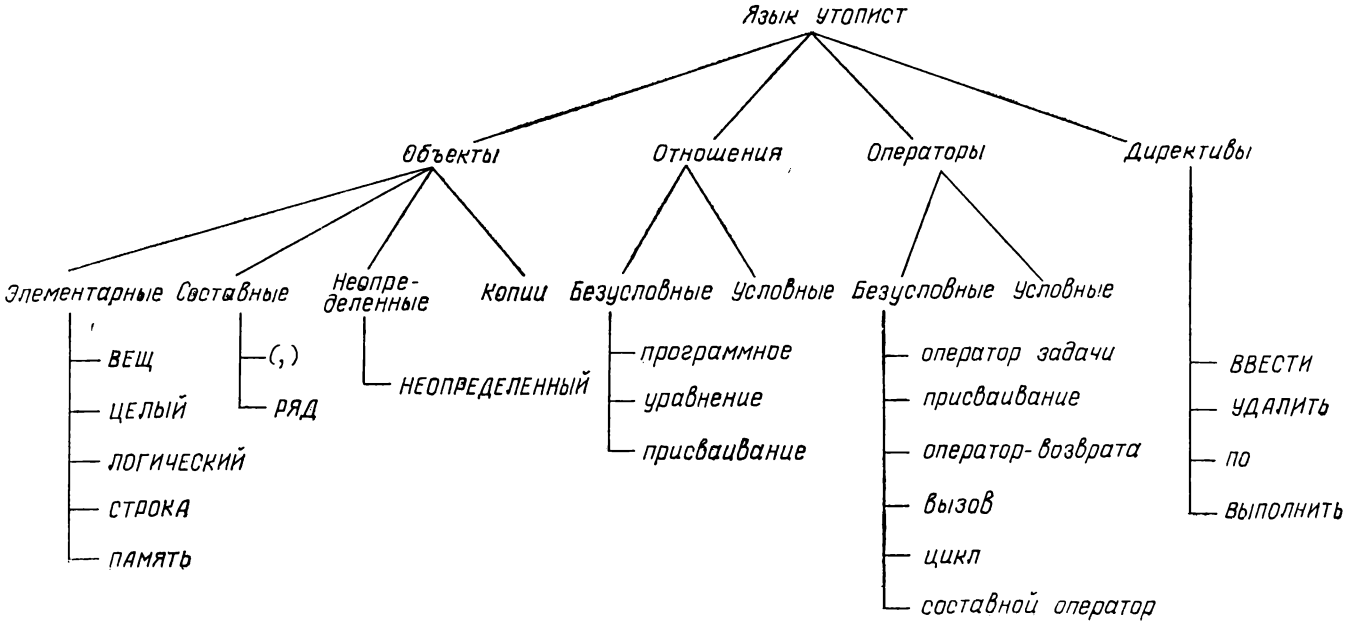


Рис. 6. Структура языка УТОПИСТ

ментарные единицы, из которых строятся все конструкции языка. Это идентификаторы, простые значения и служебные символы. В системе ПРИЗ ЕС реализован язык УТОПИСТ с русской и английской лексикой. В данной книге дано описание языка с русской лексикой. Соответствие ключевых слов русской и английской лексики приведено в приложении 3.

В описании синтаксиса нетерминальные символы состоят из строчных букв, может быть, разделяемых дефисом, например имя-объекта. Альтернативы правила изображаются отдельными строками или разделяются вертикальными черточками и заключаются в фигурные скобки, например

$$\left\{\begin{array}{l}\text{описание}\\ \text{оператор}\end{array}\right\} \quad \text{или} \quad \{\text{описание}|\text{оператор}\}$$

В фигурные скобки могут заключаться также синтаксические структуры языка с целью их выделения в описании синтаксиса. Применение квадратных скобок [,] означает, что заключенная в них конструкция или набор альтернатив могут и отсутствовать. Многоточие после символа или конструкции означает их повторение в количестве не менее одного раза.

Особый смысл имеет многоточие после запятой или точки с запятой: оно означает список из символов с этим разделителем, элементом которого является символ, предшествующий разделителю. Например, имя, ... означает то же, что имя [,имя] ...

2.1. ОПИСАНИЕ ОБЪЕКТОВ

Элементарные объекты. В синтаксисе языка УТОПИСТ определены следующие элементарные типы данных: вещественный (ВЕЩЕСТВЕННЫЙ'), целый (ЦЕЛЫЙ'), логический (ЛОГИЧЕСКИЙ') и текстовой (СТРОКА').

Кроме того, пользователю предоставлена возможность работать еще с такими объектами, о которых известно только число слов памяти, выделяемых для хранения значения описываемого объекта. Для этого в язык введен еще один элементарный тип — память (ПАМЯТЬ'). Объектами этого типа целесообразно заменить и такие составные объекты, которые в рамках задачи можно считать неделимыми, т. е. компонентами которых при описании задачи не пользуются.

В описаниях текстовых объектов и объектов типа память можно определить их длину (в словах). Если длина объекта не указана, то выделяется одно слово памяти.

Примеры описаний с элементарными типами:

```
В: ВЕЩЕСТВЕННЫЙ';
ИМЯ: СТРОКА'6;
К: ЛОГ';
ВЕКТОР: ПАМЯТЬ'10;
```

Если задача содержит несколько объектов одинакового типа, то в описании однотипных объектов перед описателем разрешается использовать список их имен:

```
X, Y, Z: ВЕЩ';
```

Составные объекты. В описании данных наряду с элементарными описателями можно употреблять составные описатели, состоящие из заключенных в скобки описаний его компонентов и отношений между ними. Эти компоненты могут быть снова составными и т. д.

Особенностью реализации такой возможности является образование в модели описываемого объекта структурных отношений, связывающих описываемый составной объект и его компоненты. Структурное отношение определяет составной объект по его компонентам или все его компоненты по заданному составному объекту.

Для примера приведено описание составного объекта КРУГ, состоящего из непосредственных компонентов РАДИУС, ПЛОЩАДЬ, ПЕРИМЕТР, ЦЕНТР. При этом компонент ЦЕНТР является также составным

```
КРУГ: (РАДИУС, ПЛОЩАДЬ, ПЕРИМЕТР: ВЕЩ';
       ЦЕНТР: (X, Y: ВЕЩ'));
```

В модели объекта ПРИМЕР на рис. 7 этому составному объекту соответствует подмодель КРУГ. Структура в виде дерева позволяет однозначно идентифицировать любой компонент с помощью составного имени, в котором последовательно записываются обозначения всех вершин на некотором пути. Например, ПРИМЕР. КРУГ.РАДИУС.

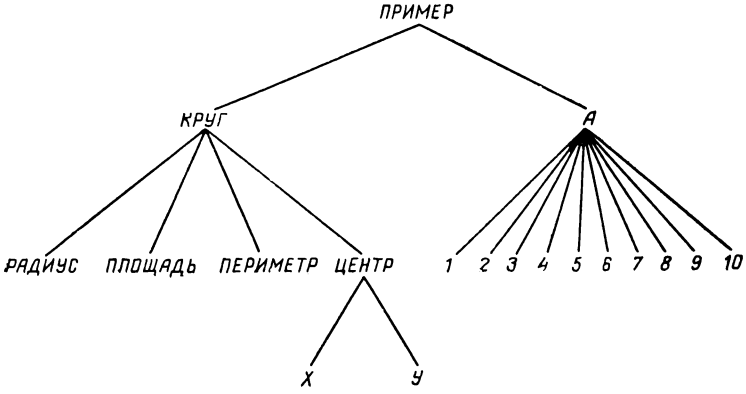


Рис. 7. Графическое представление составных объектов

Область действия имени определяется как при блочной структуре. Блокам соответствуют составные объекты. Областью действия описания является составной объект, непосредственным компонентом которого является объект, заданный данным описанием. Но если в каком-нибудь из внутренних блоков данное имя вновь описано, этот внутренний блок исключается из области действия рассматриваемого объекта, а соответствующее имя в рамках внутрен-

него блока обозначает другой объект. Поясним сказанное на примере

A: (X;ВЕЩ'; B: (X, Y; ВЕЩ'; ...) ; ...);

Во внутренних скобках X обозначает объект A.B.X, а во внешних — A.X.

Существуют и такие составные объекты, которые состоят из большого числа однотипных компонентов, в обозначении которых специальными именами нет никакой необходимости. С целью сокращения описания этих объектов в язык введено понятие «ряд». Рядом называется такой составной объект, непосредственные компоненты которого однотипны и различаются не по именам, а по номерам. Например, ряд может быть описан так:

A: МНОЖ' (1....10): ВЕЩ';

Тогда его непосредственные компоненты обозначаются.

A.(1), A.(2), ..., A.(10).

Неопределенные объекты. Для того чтобы иметь возможность при описании данных учитывать те объекты, которые до этого момента еще не определены или в разных задачах бывают разными, в язык введен особый тип описателя — неопределенный (НЕОПРЕДЕЛЕННЫЙ'). Описание такого объекта уточняется позже указанием имени ранее описанного объекта, копией которого он является (см. «Переделки»).

Неопределенные объекты и их компоненты разрешается использовать в описаниях отношений. При этом предполагается, что данный объект не остается неопределенным, если он не определен во время синтеза рабочей программы, такое описание рассматривается как ошибочное.

Копии. В отличие от других языков программирования в языке УТОПИСТ разрешается использовать в качестве описателя имя ранее описанного объекта. Это дает простую возможность расширения языка.

Очевидно, что реализуется это не легко, но зато пользователь освобождается от рутинного описания данных и получает возможность программировать на языке, который он сам по ходу дела строит. Термины этого языка могут иметь смысл, близкий к их обычному смыслу в данной области.

Если же описан объект X: типX, то X можно использовать в качестве нового описателя. Очевидно, что некоторые типы можно считать эквивалентными. Например, описание Y : X имеет тот же самый смысл, что описание Y: типX.

Копии позволяют в компактной форме восходящим способом строить описания сложных объектов, освобождая от необходимости представлять большой объект в виде одного большого многоуровневого описания. Например, описание объекта КРУГ (рис. 8) можно начинать с построения вспомогательного объекта ТОЧКА, который определяется через свои координаты *X* и *Y*. Поскольку не вся

информация об объекте должна содержаться явно в его описании, то описание круга зависит от понятия ТОЧКА. На базе уже существующих понятий экономно описывать новые объекты, в данном примере фигуру Ц1.

Если требуется определить новый объект, отличающийся от уже имеющегося наличием дополнительных компонентов и отношений, то можно в составном описателе в виде вставки включить описатель уже имеющегося объекта. Допустим, что в объект КРУГ надо включить компонент СЕКТОР. Для этого определим новый объект следующим образом:

КРУГ:(КОПИЯ' КРУГ; СЕКТОР : (ПЛ,ХОРДА : ВЕЩ'));

Переделки. Особенностью языка УТОПИСТ является наличие средств для преобразования типа объекта. Эти средства имеют весьма широкую область применения и разнообразные варианты исполнения.

В те описания, где в качестве описателя использовано имя ранее определенного объекта, разрешается добавлять так называемые переделки, определяющие некоторые изменения в типе, заданном именем.

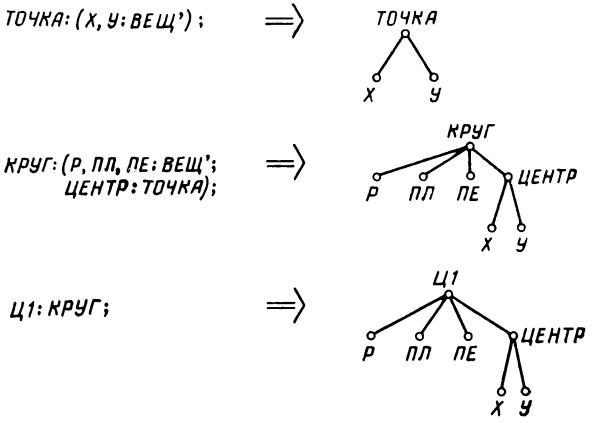


Рис. 8. Копирование объектов

В данной реализации пользователю дается возможность:

задавать исходные значения для некоторых компонентов;

определять тождественную связь;

определять тип неопределенного компонента;

изменять имя компонента.

Поскольку компоненты объекта упорядочены, то место переделок для непосредственных компонентов можно указать либо позиционным способом, либо ключевым, а если компонент имеет составное имя внутри данного объекта, то только ключевым способом.

Предположим, что у нас в модели МО уже описаны объекты А и В (рис. 9):

```
A:  (K:   НЕОПР';
     E:   (X, Y : ВЕЩ');
     M:    ПАМЯТЬ'2);
B:  (X, Y : ВЕЩ');
```

Опишем новый объект С

```
C:A   E.Y = 1.5;
```

В данном случае переделка указана ключевым способом и компоненту E.Y задается значение 1.5, т. е. в модель МО прибавляется объект С — копия объекта А, но дополненный отношением

ЗАДАНО' E.Y = 1.5;

Особого внимания заслуживает случай, когда переделка используется для доопределения неопределенного объекта. Например, указывая место переделки позиционным способом, напишем

E : A ТИП' B;

Этим описанием определяется новый объект Е, который отличается от объекта А тем, что он не содержит неопределенного компонента — его компонент К является копией объекта В.

Если переделка имеет вид

[<имя> =] <имя> ,

где первое имя по-прежнему обозначает компонент копируемого объекта, в то время как второе может быть любым именем, то возможны три варианта его исполнения.

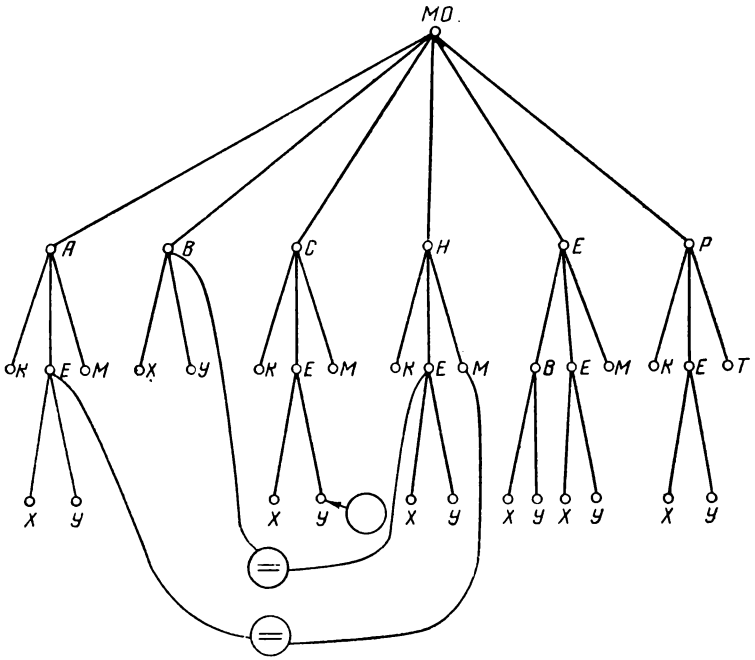


Рис. 9. Изменение типа при копировании

1. Компонент копируемого объекта не является типа НЕОПРЕДЕЛЕННЫЙ', а второе имя обозначает ранее описанный объект. Тогда устанавливается эквивалентность этих объектов. Так описываются взаимосвязанные объекты, в частности объекты, имеющие общие части.

Эквивалентную связь можно определять между компонентами описываемого объекта или между компонентом описываемого объекта и каким-то ранее описанным объектом или его компонентом. Например,

```
H : A E = B, M = A.E;
```

В модели создаются новый объект H и отношение эквивалентности между объектами МО.В и МО.Н.Е, структуры которых должны совпадать. Такое отношение отождествляет все соответствующие компоненты этих объектов, следовательно, и типы компонентов должны совпадать. Разрешается и тождество составного и элементарного объектов типа ПАМЯТЬ', если их длины совпадают. В примере такими объектами являются МО.А.Е и МО.Н.М.

2. Второй вариант отличается от первого тем, что компонент копируемого объекта является неопределенным. Тогда, кроме установления эквивалентности этих объектов, происходит еще доопределение неопределенного объекта точно так же, как при использовании ключевого слова ТИП.

3. Третий отличается тем, что объекта с заданным идентификатором нет. Тогда изменяется имя соответствующего компонента.

2.2. ОПИСАНИЕ ОТНОШЕНИЙ

Программные отношения. Программными называются отношения, которые задаются с помощью заранее написанных и уже отлаженных модулей, хранящихся в библиотеке модулей. Очевидно, что описание отношения должно содержать все необходимые сведения для вызова этого модуля. Поэтому в описании определяют имя модуля, вид оформления и все его фактические параметры — объекты модели, которые связаны с данным отношением. Кроме того, в каждом описании приводится следующая информация: тип и ранг отношения, способ использования связанных отношением объектов и при желании имя отношения.

Общая форма описания программного отношения:

```
[идент:]  { МОДУЛЬ' | МАКРО' }  идент  [[ТИП'=] { ПРОГ' | УРАВ' }]  [[РАНГ'=] целое]

[{ВХ'|ВЫХ'|СЛ'|СИЛ'|ОПР'|ВВХ'|ВВЫХ'} параметр,...]...;
```

Первым в описании отношения стоит имя отношения. Наличие имени отношения не обязательно, но дает возможность ссылаться на отношение, если это будет необходимо.

По ключевому слову МОДУЛЬ' данный модуль вызывается как подпрограмма. Но для повышения эффективности рабочей программы существует возможность реализовать отношения макроопределениями. В этом случае модуль оформляется в виде макроопределения и при описании отношения, которое реализуется таким модулем, вместо ключевого слова МОДУЛЬ' используется МАКРО'.

Тип отношения указывает, вызывается ли модуль непосредственно из модуля решения задачи (ТИП'=ПРОГ') или подключа-

ются требуемые для вычисления результата специальные системные программы, которые обращаются к модулю (ТИП'=УРАВ'). Более подробная информация об отношениях типа уравнения приведена ниже. Если тип опущен, то считается ТИП'=ПРОГ'.

Ранг отношения указывает число объектов, значения которых можно вычислить однократным применением данного отношения. Присутствие ранга также не обязательно; если ранг опущен, его значение полагается равным числу выходных переменных.

Примеры описания отношения, заданного модулем ПРОГ:

```
МОДУЛЬ'ПРОГ    ТИП' = ПРОГ' РАНГ' = 1    ВЫХ'Y;
МОДУЛЬ'ПРОГ    ПРОГ'1        ВЫХ'Y;
МОДУЛЬ'ПРОГ    ВЫХ'Y ;
```

Для автоматического синтеза программ необходимо указать способ использования каждого связанного отношением объекта. Объект, значение которого должно быть задано при любом применении некоторого отношения, называется входным; объект, значение которого вычисляется, — выходным. Способ использования объекта при описании отношения обозначается соответственным ключевым словом перед его именем. Например, отношение, представленное модулем ПРОГРАММА, вычисляющим Y по X, описывается следующим образом:

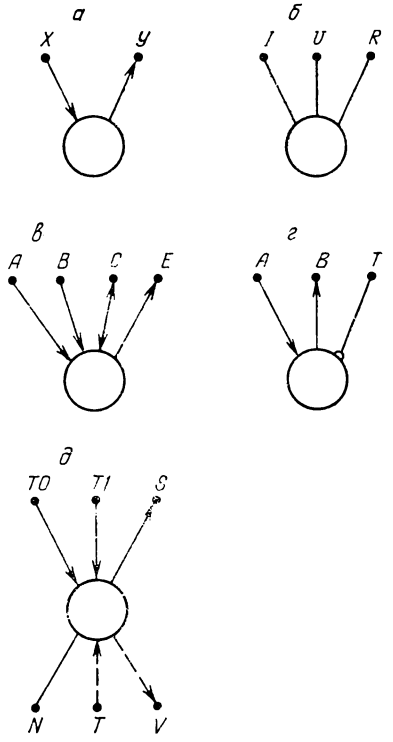


Рис. 10. Графическое представление отношений

```
МОДУЛЬ'ПРОГРАММА
ВХ'X    ВЫХ'Y;
```

Это описание порождает в семантической модели отношение, которое в формульном представлении выглядит так: X|—→У. В графическом представлении модели это отношение будет выглядеть так, как показано на рис. 10,*а*.

В языке УТОПИСТ предусмотрены средства описания таких отношений, параметры которых нельзя явно делить на входные и выходные. Каждый из них может быть и тем и другим в зависимости от конкретного применения. Такие параметры называются слабосвязанными (СЛ').

Отношения со слабосвязанными параметрами рассчитаны на то, чтобы одним таким отношением заменить несколько других. Например, определим активное сопротивление РЕЗИСТОР, компоненты которого связаны законом Ома:

```
РЕЗИСТОР:    (I, U, R : ВЕЩ';
   МОДУЛЬ'ОНМ1  ВХ'I, R  ВЫХ'U;
   МОДУЛЬ'ОНМ2  ВХ'U, R  ВЫХ'I;
   МОДУЛЬ'ОНМ3  ВХ'I, U  ВЫХ'R );
```

Эти отношения можно заменить одним

```
РЕЗИСТОР:    (I, U, R : ВЕЩ';
                МОДУЛЬ'ОНМ СЛ' I, U, R);
```

Модуль, реализующий это отношение, состоит из трех ветвей, вычисляющих соответственно I, U или R. Выполнение каждой из них зависит от задачи и определяется самой системой, реализующей язык.

Формульное представление последнего отношения, так же как и представление совокупности трех отношений, эквивалентной этому отношению, следующее:

$$I, R \mapsto U$$
$$U, R \mapsto I$$
$$I, U \mapsto R$$

Соответствующее графическое представление показано на рис. 10, *б*.

В приведенном выше примере число вычисляемых параметров равнялось 1, но если это не так, то в описании отношения необходимо указать еще число параметров, значения которых можно вычислить однократным применением соответствующего модуля. Например,

```
МОДУЛЬ' М 2 ВХ' А СЛ' В,С;

МОДУЛЬ' М РАНГ' = 2 ВХ' А СЛ' В,С;
```

В программных отношениях иногда требуется выделить так называемые побочные параметры, значения которых используются и изменяются в ходе применения отношения. Такие параметры обозначаются ключевым словом СИЛ' и называются сильносвязанными. Сильносвязанный параметр должен иметь значение уже до вызова модуля, а после вызова модуля это значение может измениться. Например, параметр подсчета числа вызовов некоторого модуля. Пример записи такого отношения:

```
МОДУЛЬ' ММ 2 ВХ' А,В ВЫХ'Е СИЛ'С;
```

Формульное представление этого отношения (рис. 10, *в*): $A,B,C \mapsto E,C$;

Применяемость отношения может зависеть от значения некоторого объекта, называемого определяющим для данного отношения. В описании отношения определяющий объект обозначается словом ОПР'. Дополнительная информация о применении такого отношения содержится ниже (см. «Условные отношения»). Формульное представление этого отношения: $T \Rightarrow A \mapsto B$, где T — определяющий объект (рис. 10, *г*).

Отношение с подзадачами — отношение, заданное такой программой, при выполнении которой надо решить какую-нибудь за-

дачу на той же модели, куда входит отношение. Отношение с подзадачей — это зависимость результата не от отдельных аргументов, а от характера связи между ними, так же как интеграл $y = \int_{T_0}^{T_1} F(x)\,dx$ зависит от функции F.

Стандартный пример дает отношение в механике между переменными V (скорость), T (время), S (путь), где S зависит от связи V с T и, конечно, пределов по времени T_0, T_1. Предположим, что связь V с T определяется моделью, быть может через серию промежуточных отношений.

В этом случае для определения зависимости $V = F(T)$ необходимо решить подзадачу ВЫЧИСЛИТЬ' V ПО' T на той же модели, куда входит отношение интеграла. В языке предусмотрена возможность описания подзадачи в несколько ином виде в описании отношения

МОДУЛЬ' ИНТ 2 ВХ' T0, T1 ВЫХ' S,N ВВХ'T ВВЫХ'V;

Здесь переменные типа ВВХ' являются входными для подзадачи, а переменные типа ВВЫХ' — искомыми для подзадачи.

В поисках решения задачи планировщик вначале не использует отношений с подзадачами. Если при этом задачу решить невозможно, то пытаются применить отношения с подзадачами. Такое отношение применимо только тогда, когда подзадача, поставленная в описании отношения, разрешима. При этом система составляет алгоритм решения подзадачи, синтезирует модуль для вычислений (вычисления V по T в данном случае) и присваивает ему уникальный номер. Для передачи этого номера в модуль ИНТ служит параметр N, который входит в описание подзадачи и непосредственно предшествует параметрам типа ВВХ' и ВВЫХ'.

Формульное представление этого отношения (рис. 10, *д*):

solv(T, V, модель) => T0, T1|→ S,N

Присваивание. На языке УТОПИСТ существуют две конструкции присваивания — ПРИСВ' и ЗАДАНО'.

Конструкция ПРИСВ' предназначена для вычисления арифметического или логического выражений и присваивания вычисленного значения некоторому объекту модели, а также для присваивания объектам исходных значений.

Общая форма конструкции ПРИСВ':

ПРИСВ' имя: = { выражение / значение };

При применении отношения присваивания выражение, находящееся справа от знака присваивания, вычисляется, полученный результат засылается в качестве значения объекту, имя которого стоит слева от знака присваивания. Этот объект не разрешается включить в состав операндов выражения, а его тип должен совпадать с типом значения выражения.

Результат трансляции этой конструкции — группа команд, необходимых для вычисления значения выражения и пересылки его, а также описания отношения присваивания. Входными параметрами отношения являются те объекты, которые входят в число операндов выражения, а выходным параметром — объект из левой части присваивания.

Например, наличие в программе конструкции

ПРИСВ' Z : = X + SIN' Y;

означает, что в модель прибавляется отношение

МАКРО' < системное имя > ВЫХ' Z ВХ' X, Y;

и соответствующее макроопределение (рис. 11).

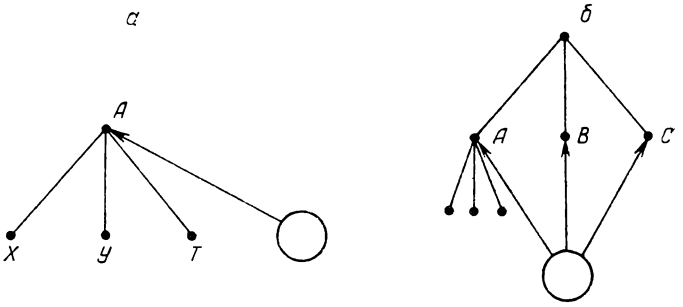


Рис. 11. Графическое представление присваиваний

Значением может быть целое или вещественное число со знаком или без знака, строка или логическое значение. Тип присваиваемого значения должен совпадать с типом объекта. Исключение составляет объект вещественного типа, которому можно присвоить и значение целого типа.

Составному объекту можно задавать и составное значение, например

A: (X, Y: ВЕЩ' ; T: СТРОКА');

ПРИСВ' A:= (5,2.5,' ИМЯ');

Эта конструкция означает, что в модель объекта А прибавляется отношение с выходным параметром А, как показано на рис. 11, *а*.

Важно отметить, что число значений должно равняться числу компонентов и значения располагаются в том же порядке, что и компоненты в описании объекта. Поэтому порядок компонентов в описании играет существенную роль.

Кроме ПРИСВ', присваивание может описываться конструкцией

ЗАДАНО' {имя = значение}, . . . ;

которая позволяет присваивать значения нескольким объектам, но не позволяет описывать вычисление выражениями.

Например, конструкция

ЗАДАНО' A = (1,2,3), B = 5, C = 6;

используется для присваивания исходных значений объектам A, B и C (рис. 11, *б*).

Уравнение. Примечательным в языке УТОПИСТ является возможность задавать отношения между объектами вещественного типа уравнениями, общая форма которых следующая:

УРАВ' арифметическое—выражение = арифметическое—выражение;

Семантику уравнения можем записать, придав уравнению вид

$$f(x_1, \ldots, x_n) = 0,$$

путем переноса обоих арифметических выражений в левую сторону. $x_1, \ldots, x_n$ обозначают объекты, входящие в уравнение. Если для некоторого x_i можно уравнению придать вид

$$a_i * x_i - b_i = 0,$$

где a_i, b_i — выражения, не содержащие x_i, то x_i входит в уравнение линейно.

Уравнение семантически эквивалентно совокупности присваиваний

ПРИСВ' $x_i := b_i/a_i$,

заданных для тех переменных, которые входят в уравнение линейно.

Например, уравнение

РАССТОЯНИЕ = ВРЕМЯ * СКОРОСТЬ

линейно относительно всех входящих в него объектов. Поэтому эквивалентное представление семантики через присваивания следующее:

РАССТОЯНИЕ : = ВРЕМЯ * СКОРОСТЬ
ВРЕМЯ : = РАССТОЯНИЕ /СКОРОСТЬ
СКОРОСТЬ: = РАССТОЯНИЕ / ВРЕМЯ

Многие физические и геометрические соотношения могут быть описаны на УТОПИСТе уравнениями, но при этом необходимо иметь в виду, что транслятор не решает уравнений совместно.

Например, по уравнениям

УРАВ' X + Y = 4;

УРАВ' X * Y = 4;

система не может вычислить ни X, ни Y, если хотя бы один из них заранее не задан. Еще хуже, если задать случайно какое-нибудь значение либо X, либо Y, то мы не сможем предвидеть, какое из двух возможных значений будет вычислено для другого из них. Такая система линейно-независимых уравнений однозначно определяет допустимое значение каждой переменной. Однако решение

таких систем уравнений не входит в стандартные возможности системы ПРИЗ ЕС.

Условные отношения. Иногда некоторое отношение истинно лишь при выполнении определенного условия. Например, для полупроводникового диода при положительной разности потенциалов справедлив закон

$$U1 - U2 = I * R,$$

а для отрицательной в идеальном случае ток отсутствует: $I=0$. Это обстоятельство можно выразить, добавив к отношению условие в виде

ЕСЛИ' лог — выражение ТО'

Например, условные отношения полупроводникового диода можно выразить в следующем виде:

```
ЕСЛИ' U1 БР' U2 ТО' УРАВ' U1 — U2 = I * R;
ЕСЛИ' U1 МР' U2 ИЛИ' U1 РАВ' U2 ТО' ПРИСВ' I: =0;
```

Заметим, что эти два отношения нельзя заменить одним, так как они качественно различны.

Для использования таких отношений при решении задачи система вставляет в расчетную программу фрагменты, проверяющие данные условия.

Если пользователь не использует вышеупомянутую конструкцию, то он сам должен обеспечить описание и вычисление некоторого логического компонента, от значения которого зависит применяемость отношения. В описании отношения этот объект обозначается ключевым словом ОПР' (см. выше). Например, условное отношение, применяемое при положительных значениях входного параметра, можно выразить следующим образом:

```
ЕСЛИ' X БР' 0 ТО' МОДУЛЬ' SQRT ВХ' X ВЫХ' Y;
```

или

```
T: ЛОГ';
ПРИСВ' T: = X БР' 0;
МОДУЛЬ' SQRT ОПР' T ВХ' X ВЫХ' Y;
```

Определяющему объекту T не соответствует ни один параметр модуля, реализующего условное отношение.

Шаблон имен. В модели одно конкретное отношение всегда связывает определенное число объектов или их компонентов. Но допустим, что придется описать отношение, вычисляющее Y по $X_1, X_2, \ldots, X_k$, где k в разных задачах может иметь разное значение. Например, Y является суммой, произведением, максимумом и т. д. от переменных $X_1, X_2, \ldots, X_k$, число которых варьируется от модели к модели. Было бы крайне неудобно в каждой задаче заново описывать данное отношение в зависимости от числа параметров.

Одним из способов решения этой проблемы является применение шаблона имен в описании отношения. Шаблон имен — это составное имя, содержащее ключевое слово ВСЕ':

$$\text{идент}\,[\cdot\ \left\{\begin{array}{l}\text{ВСЕ'}\\ \text{идент}\\ \text{(целое)}\end{array}\right\}]\ldots.$$

Ключевое слово ВСЕ' имеет смысл «все непосредственные компоненты». Следовательно, шаблон в виде X.ВСЕ' определяет список имен, куда входят все непосредственные компоненты объекта X. В одном шаблоне слово ВСЕ' можно употреблять не более двух раз.

Применение шаблона имен предполагает автоматическое построение описания отношения. Например, описание отношения

МОДУЛЬ' МАХ ВХ' X. ВСЕ' ВЫХ'Y ;

значит, что входными параметрами являются все непосредственные компоненты объекта X. Если в конкретной задаче их число равняется трем (*X1*, *X2*, *X3* на рис. 12, *а*), то системой автоматически строится новое описание

МОДУЛЬ' МАХ ВХ' X.X1, X.X2, X.X3 ВЫХ'Y ;

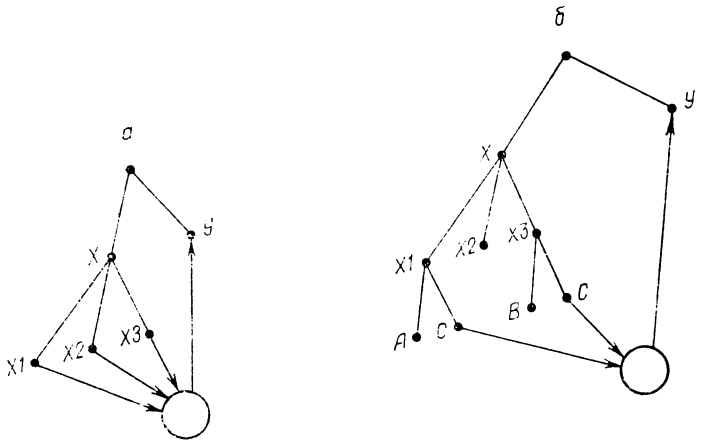


Рис. 12. Применение шаблона имен

С помощью шаблона имен можно задать не только непосредственные компоненты какого-то объекта, но и одноименные компоненты этих непосредственных компонентов. Для составления списка имен по шаблону X.ВСЕ'.C просматривают все непосредственные компоненты объекта X, при этом в список включают все подчиненные им компоненты, имеющие имя C. Предположим, что *X1* и *X3* содержат компонент с именем C (рис. 12, *б*), тогда описание

МОДУЛЬ' МАХ ВХ' X.ВСЕ'. C ВЫХ' Y ;

эквивалентно описанию

МОДУЛЬ' МАХ ВХ' X.X1.C, X.X3.C ВЫХ'Y ;

Надо отметить, что не все непосредственные компоненты объекта X должны содержать C, отношением связываются те, которые имеются в модели.

Выражения. В языке УТОПИСТ допускаются арифметические и логические выражения, которые могут использоваться для описания вычислений, требуемых для получения некоторого значения. Так, арифметические выражения используются в присваиваниях, логические — в присваиваниях и в условиях конструкции ЕСЛИ' <логическое выражение> ТО' ... и в условии цикла.

Кроме того, арифметические выражения используются в уравнениях для задания отношений. Например, как показано выше, можно определить уравнение УРАВ'Y=X+1. Это не значит, что необходимо вычислить X+1 и это значение присвоить Y. Смысл данного отношения в том, что Y на единицу больше, чем X. И, в частности, если Y известен, то и X может быть вычислен как Y — 1.

Под арифметическим выражением понимается любая последовательность констант и вещественных объектов модели, разделенных знаками операций и круглыми скобками таким образом, чтобы получилось имеющее смысл математическое выражение. Допускается использование следующих пяти операций: + (сложение), — (вычитание), * (умножение), / (деление), ** (возведение в степень).

Символ «**», при записи которого используются две позиции, понимается всегда как знак одной операции. В качестве примеров арифметических выражений можно привести следующие:

```
X + 2 * Y
A ** 2 — B/C
(2 * РАДИУС) ** 2
```

Чтобы улучшить читабельность выражения, можно пользоваться пробелами, однако следует иметь в виду, что пробелы не должны использоваться внутри имен.

В языке УТОПИСТ предусмотрены средства для включения в выражение различных функций, вычисляемых по подпрограммам. В список имен этих функций входят SIN', COS', TG', ARCTG', ABS' (абсолютная величина), LN', EXP', SQRT' (квадратный корень). Аргумент любой функции может быть выражением. Поэтому допустима запись, например, такого вида:

```
SQRT' (Z + SQRT' (X**2 + Y**2) )
```

Если в выражении отсутствуют круглые скобки, определяющие порядок выполнения операций, то операции выполняются в следующей последовательности:

вычисление функций;
возведение в степень;
умножение и деление;
сложение и вычитание.

Операции с одинаковым приоритетом выполняются последовательно слева направо.

При составлении арифметических выражений необходимо придерживаться следующих правил:

все объекты, входящие в выражение, должны иметь тип ВЕЩ';

следует помнить, что вычисление выражения дает вещественный результат;

константы в выражении могут быть либо целые, либо вещественные;

два знака операции не могут стоять рядом, они должны быть разделены по крайней мере скобками. Запись вида X* — У неверна, а запись X*(—У) верна. Следовательно, отрицательное число должно заключаться в скобки. Это требование необходимо соблюдать и тогда, когда отрицательное число стоит первым в выражении. Например, A : = (—5)*B/(—3);

в отличие от обычной записи знак умножения никогда не опускается.

Логическое выражение образуется из операндов логического выражения и операций. Операндами логических выражений являются логические константы, логические объекты, отношения, а также логические выражения, заключенные в скобки. Над этими операндами могут быть произведены следующие логические операции: НЕ' (отрицание), И' (логическое умножение), ИЛИ' (логическое сложение).

Выполнение логических операций приведено в табл. 2.

Таблица 2

Значение операнда		Значение выражения в зависимости от операции		
A	B	НЕ' A	A И' B	A ИЛИ' B
ИСТИНА	ИСТИНА	ЛОЖЬ	ИСТИНА	ИСТИНА
ИСТИНА	ЛОЖЬ	ЛОЖЬ	ЛОЖЬ	ИСТИНА
ЛОЖЬ	ИСТИНА	ИСТИНА	ЛОЖЬ	ИСТИНА
ЛОЖЬ	ЛОЖЬ	ИСТИНА	ЛОЖЬ	ЛОЖЬ

Логические операции могут выполняться только над логическими величинами. Две логические операции могут быть записаны подряд лишь в том случае, если второй из них является операция НЕ'. Так, комбинации И' НЕ' и ИЛИ' НЕ' являются правильными, всякая другая комбинация логических операций недопустима.

Отношение образуется из двух арифметических выражений вещественного типа, соединенных операцией отношения. Операции отношения характеризуют соотношения между значениями арифметических выражений:

РАВ' (равно)
НР' (не равно)
МР' (меньше или равно)
БР' (больше или равно)

Кроме вышеприведенных операций, существует в языке еще операция сравнения кодов — СР'. Сравниваемые объекты могут быть любого типа, но результат сравнения логический.

Примеры записи логических выражений.

На языке УТОПИСТ	Математическая запись
А ИЛИ' В И' С	$a \vee в \wedge c$
TG' X БР' 1 + Y	$\operatorname{tg} x \geqslant 1 + y$
X БР' 1 ИЛИ' X МР'10	$x \geqslant 1 \vee x \leqslant 10$

Логические выражения вычисляются в соответствии с приоритетом операций:

1) вычисление функций;
2) возведение в степень;
3) умножение и деление;
4) сложение и вычитание;
5) операции отношения;
6) операция НЕ;
7) операция И;
8) операция ИЛИ.

Выражения, заключенные в скобки, вычисляются в первую очередь.

2.3. ДЕЙСТВИЯ

Оператор задачи. В языке УТОПИСТ предусмотрена возможность описывать задачи, не задавая явно алгоритма его решения. С этой целью в язык введен оператор задачи. Последний является семантически правильным только тогда, когда задача разрешима.

Оператор задачи имеет следующую форму:

НА' М ВЫЧИСЛИТЬ' У1, У2, ..., УК [ПО' Х1, Х2, ..., XN],

где X1, ..., XN, Y1, ..., YK — имена объектов, описанных моделью задачи с именем М. X1, ..., XN являются входными параметрами задачи, Y1, ..., YK — выходными. Список входных параметров может и отсутствовать, если исходные значения заданы уже в описании данных. Модель, на которой описывается задача оператором ВЫЧИСЛИТЬ', может являться моделью любого объекта, описанного в программе, или моделью любого понятия из библиотеки моделей.

Для решения описанной задачи система автоматически синтезирует алгоритм и оформляет его на языке Ассемблера в виде модуля решения задачи. При выполнении программы, исходный текст которой содержит оператор задачи, происходит обращение к этому модулю. При этом параметры передаются в соответствии с соглашениями, принятыми в системе ОС ЕС. После выполнения модуля решения задачи вычисленные значения выходных параметров и управление передаются вызывающей программе.

Другие операторы. В языке УТОПИСТ имеется ряд операторов, которые совпадают с операторами других языков.

1. Оператор присваивания предназначен для вычисления арифметического или логического выражения и присваивания вычисленного значения переменной программы. Общая форма оператора

имя: = выражение;

Переменной программы могут быть все объекты модели, которые определены ранее в программе или в библиотеке моделей.

Так как объекты модели могут иметь составное имя, то и соответствующие переменные программы могут иметь составное имя. Тип переменной должен совпадать с типом присваиваемого значения. Но переменной программы может быть и величина, которая не является объектом модели. В этом случае переменная определена только в описании действий и ей присваивается тип, вещественный или логический, в зависимости от вида присваиваемого значения. Поясним сказанное на примере.

```
ПРОГРАММА' ПРИМЕР;
    ПУСТЬ' A: (X, Y : ВЕЩ';
              УРАВ' X = 2 + Y;
              ЗАДАНО' X = 10);
              B: ВЕЩ';
                  ДЕЙСТВИЯ'
                            B : =1;
                            C: = 2;
                            A.X : = B + 2 * C;
```

Переменные программы B и A.X по имени являются объектами модели, но пока они никак не связаны с моделью. Чтобы значения, присвоенные объектам модели с помощью оператора присваивания, учитывались при решении задачи, следует в операторе ВЫЧИСЛИТЬ' после слова ПО' перечислить имена этих объектов. Пусть в данном примере необходимо вычислить A.Y по A.X, значение которого определено в описании действий. Тогда оператор должен выглядеть следующим образом:

НА' ПРИМЕР ВЫЧИСЛИТЬ' A.Y ПО' A.X;

При выполнении программы сначала вычисляется выражение B+2*C, и переменная A.X получит значение, равное 5. Затем это значение используется при решении уравнения X=2+Y, и окончательный результат равняется 3.

Но допустим, что в операторе ВЫЧИСЛИТЬ' объект A.X не задан

НА' ПРИМЕР ВЫЧИСЛИТЬ' A.Y;

тогда значением объекта берется значение, присвоенное ему в описании данных, т. е. 10, и A.Y получит значение, равное 8.

Отметим отличие оператора присваивания от описания отношений ПРИСВ' и ЗАДАНО'. Оператор выполняется всегда в определенном месте программы, а описание отношения используется тогда, когда это нужно для выполнения оператора ВЫЧИСЛИТЬ'.

2. Условный оператор предназначен для управления вычисле-

ниями в зависимости от выполнения некоторого условия. Общая форма оператора

ЕСЛИ' лог — выражение ТО' безусловный — оператор
[ИНАЧЕ' оператор];

При выполнении этого оператора вычисляется и анализируется значение логического выражения. Если значение выражения истинно, выполняется оператор, следующий за ключевым словом ТО'. После выполнения оператора, если он не является оператором выхода, управление передается оператору, следующему за данными условным оператором. Если значение выражения ложно, оператор за ключевым словом ТО' игнорируется и выполняется либо оператор за ключевым словом ИНАЧЕ', либо при его отсутствии оператор, непосредственно следующий за данным условным оператором.

Пример

```
ЕСЛИ' U1 БР' U2 ТО' I : = (U1 — U2) /R
                ИНАЧЕ' I : = 0;
```

В этом примере величина I вычисляется в зависимости от значений переменных U1 и U2. Если $U1 \geqslant U2$, значение переменной I вычисляется по формуле $(U1 - U2)/R$, в противном случае I принимает значение 0.

3. Любые операторы можно последовательно объединять в составной оператор:

[метка] : (оператор; ...);

Выполнение составного оператора заключается в последовательном выполнении (слева направо) входящих в него операторов до тех пор, пока не будет выполнен оператор ИЗ', выводящий за пределы данного оператора, или не будут выполнены все операторы, входящие в данный составной оператор.

4. Для передачи управления в языке имеется оператор выхода

ИЗ' метка;

Выполнение оператора выхода состоит в выходе из составного оператора, помеченного меткой. После этого начинается выполнение оператора, следующего за данным составным оператором.

Например,

```
M: (ЕСЛИ' X БР' 0 ТО' ИЗ' M;
      X : = ABS'X);
Y : = SQRT'X;
```

Здесь оператор ИЗ' вызывает передачу управления оператору, непосредственно следующему за составным оператором M, т. е. оператору присваивания Y : = SQRT'X.

5. Оператор цикла предназначен для повторения некоторого участка программы. Общая форма оператора

ПОКА' лог—выражение ЦИКЛ' оператор;

Выполнение оператора цикла начинается с вычисления значения логического выражения. Если это значение истинно, то выполняет-

ся оператор, записанный после слова ЦИКЛ'. Затем снова вычисляется и анализируется значение логического выражения. Если это значение опять истинно, то весь процесс повторяется. Если значение ложно, то управление передается оператору, следующему за оператором цикла. Это называется нормальным выходом из цикла. Но из цикла можно выйти и с помощью оператора выхода.

Рассмотрим пример употребления оператора цикла.

Присвоить переменной X значение произведения всех целых чисел от 1 до 15 можно с помощью следующих операторов:

```
X : = 1;
K : = 2;
ПОКА' K МР' 15 ЦИКЛ'
      (X : = X * K; K : = K + 1);
```

Внутри составного оператора, входящего в оператор цикла, могут быть другие операторы цикла.

6. При программировании на языке УТОПИСТ, как и при написании программ на других языках программирования, иногда возникает необходимость выделять в подпрограммы неоднократно выполняемые вычисления. Программа на УТОПИСТе может обращаться к подпрограмме, написанной на языке Ассемблера, на Фортране или на УТОПИСТе.

Обращение к подпрограмме осуществляется с помощью оператора вызова, общая форма которого следующая:

ВЫЗОВ' идент [(факт — параметр, ...)] ;

В операторе ВЫЗОВ' указываются имя подпрограммы и фактические параметры, со значениями которых должна выполняться подпрограмма. Список фактических параметров может и отсутствовать. В качестве фактического параметра могут быть использованы переменные программы, в том числе и объекты модели, описанные в данной программе, и значения. Например,

ВЫЗОВ' ПРОГ (ИСТИНА', 'ДАТА = 22.12.76', T.X.A);

Оператор ВЫЗОВ' передает управление подпрограмме с данным именем (в нашем примере с именем ПРОГ) и устанавливает соответствие между формальными и фактическими параметрами. Фактические параметры в операторе ВЫЗОВ' должны согласовываться с формальными по типу, длине, количеству и порядку следования. После выполнения подпрограммы вычисленные значения параметров и управление передаются вызывающей программе.

2.4. ДИРЕКТИВЫ

Директивы — это операторы, выполняемые во время трансляции. Для работы с библиотекой моделей предназначены директивы ВВЕСТИ', УДАЛИТЬ' и ПО'.

Выполнение директивы ВВЕСТИ' меняет состояние библиотеки моделей. По директиве

ВВЕСТИ' имя;

в библиотеку записывается объект, имя которого задано после слова ВВЕСТИ'. Если это имя составное, то именем объекта в архиве становится последний его компонент. Этот объект рассматривается как новая модель предметной области и ему выделяется в библиотеке отдельный раздел, имя и адрес которого включаются в каталог библиотеки.

Раздел библиотеки моделей имеет такую же структуру, как составной объект. Если в библиотеке уже имеется модель с данным именем, то об этом сообщается пользователю, и директива игнорируется.

Для дополнения уже существующей модели придется в директиве указать и ее имя. Например, для расширения модели цепей ЦЕПИ понятием РЕЗИСТОР пользуются следующей директивой:

ВВЕСТИ' РЕЗИСТОР В' ЦЕПИ;

а расширение понятия РЕЗИСТОР компонентом R реализуется директивой ВВЕСТИ' R В' ЦЕПИ.РЕЗИСТОР;

Директива УДАЛИТЬ' исключает понятие с данным именем. Например,

УДАЛИТЬ' ЦЕПИ;

освобождает в библиотеке целый раздел, так как в данном случае объектом является модель предметной области целиком, а директива

УДАЛИТЬ' ЦЕПИ. РЕЗИСТОР;

удаляет только понятие с именем РЕЗИСТОР.

Если имя встречается и в модели задачи (на втором уровне) и в библиотеке моделей, то удаляется объект из модели задачи.

Директива ПО' позволяет сократить имена библиотечных понятий, применяемых при описании данных и действий, за исключением оператора задачи. Запись этой директивы означает, что все используемые в дальнейшем объекты определены либо в самой программе, либо в модели, либо подмодели предметной области, имя которой указано в директиве. После появления очередной директивы ПО', ВВЕСТИ' или УДАЛИТЬ' связь с этой моделью исчезает.

Например, фрагмент текста с полными именами:

```
ПУСТЬ' ФИГУРА:
                    (КР: ГЕОМ.КРУГ;
                    КВ: ГЕОМ.КВАДРАТ;
                    . . . . . );
ДЕЙСТВИЯ'
    ГЕОМ.КРУГ.ДИАМЕТР: = 157.4;
    НА' ГЕОМ.КРУГ ВЫЧИСЛИТЬ' ПЛОЩАДЬ ПО' ДИАМЕТР;
    ФИГУРА.КР.ПЛОЩАДЬ: = ГЕОМ.КРУГ.ПЛОЩАДЬ;
    НА' ФИГУРА ВЫЧИСЛИТЬ' . . . ;
    . . . . . .
```

Тот же самый текст при использовании директивы ПО':

```
    ПО' ГЕОМ;
ПУСТЬ' ФИГУРА:
```

```
                (КР:КРУГ;
                КВ:КВАДРАТ;
                . . . . . );
ДЕЙСТВИЯ'
    КРУГ. ДИАМЕТР: = 157,4;
    НА' ГЕОМ.КРУГ ВЫЧИСЛИТЬ' ПЛОЩАДЬ ПО' ДИАМЕТР;
    ФИГУРА.КР.ПЛОЩАДЬ: = КРУГ.ПЛОЩАДЬ;
    НА' ФИГУРА ВЫЧИСЛИТЬ' . . .;
```

Кроме директив работы с библиотекой моделей, имеется еще одна директива — ВЫПОЛНИТЬ'. Директива ВЫПОЛНИТЬ' дает возможность во время трансляции сразу выполнить как пользовательские программы, так и некоторые системные программы. При этом предполагается, что вызываемые программы находятся в библиотеке загрузочных модулей, для которой задано DD-предложение на языке управления заданиями (см. гл. 4).

Директива имеет вид

ВЫПОЛНИТЬ' идентификатор; + + +

где идентификатор является именем вызываемого загрузочного модуля. После директивы должен стоять признак конца предложения «+ + +».

Вышеуказанные директивы могут располагаться непосредственно после любой точки с запятой, не охваченной скобками. После директив необходимо снова начать описание данных или действий, указав слово ПУСТЬ', ДЕЙСТВИЯ' или ПОДПРОГРАММА'.

Например,

```
ПРОГРАММА' ПРИМЕР;
ПУСТЬ' КРУГ: (ПЛ, Р,Д,ПЕ: ВЕЩ';
 МОДУЛЬ' КРУГМ 3 СЛ' ПЛ,Р,Д,ПЕ);
ВВЕСТИ' КРУГ В' ГЕОМ;
ПУСТЬ' КВАДРАТ:(. . .
        .
        .
        .
```

Формальное описание синтаксиса языка УТОПИСТ приведено в приложении 4.

Ниже приведен пример программы на языке УТОПИСТ:

```
* В ЭТОМ ПРИМЕРЕ ИСПОЛЬЗОВАНИЯ ЯЗЫКА УТОПИСТ
* ОПРЕДЕЛЯЮТСЯ ПОНЯТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ,
* КОТОРЫЕ ВВОДЯТСЯ В БИБЛИОТЕКУ МОДЕЛЕЙ
*И ЗАТЕМ ИСПОЛЬЗУЮТСЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ.
* ПРОГРАММА НАЧИНАЕТСЯ ВСЕГДА СЛОВОМ ПРОГРАММА',
* ПОСЛЕ КОТОРОГО ПИШЕТСЯ ИМЯ ПРОГРАММЫ.
        ПРОГРАММА' ПРИМЕР;
* СЛОВО ПУСТЬ' ОБОЗНАЧАЕТ НАЧАЛО ОПИСАНИЯ ДАННЫХ,
* ЛИН — ПЕРВОЕ ОПРЕДЕЛЯЕМОЕ ПОНЯТИЕ. ЭТО СОСТАВНОЕ
* ПОНЯТИЕ, КУДА ВХОДЯТ ОПРЕДЕЛЯЕМЫЕ ДАЛЕЕ ПОНЯТИЯ
* СОПР, ДВП.
            ПУСТЬ' ЛИН: (
* ТЕПЕРЬ ПРИВОДИТСЯ ОПРЕДЕЛЕНИЕ АКТИВНОГО СОПРОТИВЛЕ-
* НИЯ СОПР, КОТОРОМУ СООТВЕТСТВУЮТ ТОК I, НАПРЯЖЕНИЕ U,
* ОМИЧЕСКОЕ СОПРОТИВЛЕНИЕ R, СВЯЗАННОЕ ЗАКОНОМ ОМА
* I = U/R
            СОПР: (I, U, R : ВЕЩ';
                УРАВ' I = U/R ) ;
```

```
* СЛЕДУЮЩИМ ОПРЕДЕЛЯЕТСЯ ДВУХПОЛЮСНИК ДВП.
        ДВП: (I, U, R : ВЕЩ' );
* ДАЛЕЕ ОПРЕДЕЛЯЕМОЕ ПОНЯТИЕ ПАР ПРЕДСТАВЛЯЕТ
* ПАРАЛЛЕЛЬНОЕ СОЕДИНЕНИЕ ДВУХПОЛЮСНИКОВ R1, R2.
* ПАРАЛЛЕЛЬНОЕ СОЕДИНЕНИЕ РАССМАТРИВАЕТСЯ КАК
* ДВУХПОЛЮСНИК Z.
        ПАР: (R1, R2, Z : ДВП;
* ДЛЯ УДОБСТВА ДАЛЬНЕЙШЕГО ИСПОЛЬЗОВАНИЯ В ПОНЯТИЕ ПАР
* ВКЛЮЧАЮТСЯ ВЕЛИЧИНЫ I, U, R, КОТОРЫЕ РАВНЫ
* СООТВЕТСТВУЮЩИМ ВЕЛИЧИНАМ ДВУХПОЛЮСНИКА Z :
        I, U, R   :   ВЕЩ';
        УРАВ' I =  U/R;
        УРАВ' I =  Z.I;
        УРАВ' U =  Z.U;
        УРАВ' R =  Z.R;
* В ПОНЯТИЕ ПАР ВКЛЮЧАЮТСЯ ТАКЖЕ ОСНОВНЫЕ СООТНОШЕНИЯ
* МЕЖДУ ТОКАМИ, СОПРОТИВЛЕНИЯМИ И НАПРЯЖЕНИЯМИ
* ПАРАЛЛЕЛЬНО СОЕДИНЕННЫХ ДВУХПОЛЮСНИКОВ И
* СОПРОТИВЛЕНИЙ.
        УРАВ' R1.U = U;
        УРАВ' R2.U = U;
        УРАВ' R1.I + R2.I = I;
        УРАВ' R = R1.R + R2.R/(R1.R + R2.R) );
  +++
* АНАЛОГИЧНО ПОНЯТИЮ ПАР ОПРЕДЕЛЯЕТСЯ ПОНЯТИЕ ПОСЛ,
* ПРЕДСТАВЛЯЮЩЕЕ ПОСЛЕДОВАТЕЛЬНОЕ СОЕДИНЕНИЕ
* ДВУХПОЛЮСНИКОВ.
        ПОСЛ: (R1, R2, Z:ДВП;
                I, U,    R:ВЕЩ';
                УРАВ'  I = U/R;
                УРАВ'  I = Z.I;
                УРАВ'  U = Z.U;
                УРАВ'  R = Z.R;
                УРАВ'  R1.U + R2.U = U;
                УРАВ'  R1.I = I;
                УРАВ'  R2.I = I;
                УРАВ   R = R1.R + R2.R) ;
                ) ;
* ДИРЕКТИВОЙ ВВЕСТИ ОПРЕДЕЛЕННЫЕ ВЫШЕ ПОНЯТИЯ ВВОДЯТСЯ
* В БИБЛИОТЕКУ МОДЕЛЕЙ
        ВВЕСТИ' ЛИН; + + +
* ТЕПЕРЬ МОГУТ БЫТЬ ЗАПИСАНЫ ЗАДАЧИ С ИСПОЛЬЗОВАНИЕМ
* ВВЕДЕННЫХ ПОНЯТИЙ.
* ЧТОБЫ СДЕЛАТЬ ДОСТУПНЫМИ ПОНЯТИЯ, ВХОДЯЩИЕ В
* ЛИН, ИСПОЛЬЗУЕТСЯ СЛЕДУЮЩАЯ ДИРЕКТИВА
                ПО' ЛИН;
                 ПУСТЬ' СХЕМА: (R5, R6 : СОПР R = 5;
                                R3, R4 : СОПР R = 16;
                                X1: ПАР R5, R6;
                                X2: ПАР R3, R4;
                                X3: ПОСЛ X1.Z, X2.Z;
                ЗАДАНО' R5.I = 0.5); + + +
                ДЕЙСТВИЯ' НА' СХЕМА ВЫЧИСЛИТЬ'
                R3.I, R4.I, R5.U;
                ПУСТЬ' СХ:
                      (R3, R4, R5 : СОПР R = 155.0;
                          X1: ПОСЛ R4, R5;
                          X2: ПАР X1.Z, R3, I = 5.1) ;
                 ДЕЙСТВИЯ' НА' СХ ВЫЧИСЛИТЬ' R4.I;
* ПРОГРАММА КОНЧАЕТСЯ ВСЕГДА СЛОВОМ КОНЕЦ'
    КОНЕЦ' + + +
```

Примечания и ограничения.

1. Идентификаторы различаются по первым 8 символам, хотя могут иметь и большую длину.

2. Ключевые слова различаются по первым 3 символам, хотя могут иметь и большую длину. Апостроф является признаком ключевого слова. Таким образом, равноправны следующие пары:

```
СТРОКА'     СТР'
УРАВНЕНИЕ'  УРАВ'
ВЫЧИСЛИТЬ'  ВЫЧ'
```

3. Ограничена длина имени. Составное имя максимально может содержать 19 точек.

4. Вещественное число — это последовательность не более чем 16 символов.

5. Текст программы задачи разделяется с помощью сочетания символов +++ на части — предложения. Символы +++ можно вставить после описания компонентов или после оператора. Трансляция входного текста задачи происходит отдельно по предложениям. Символы +++ должны быть отперфорированы на перфокарте самыми последними.

6. Максимально допустимая длина предложения — 250 лексем.

7. Максимальное суммарное количество переменных и констант в описании действий — 250.

8. Максимальная длина внутримашинного описания модели задачи — 128 000 символов.

9. Максимальное количество отношений в описании данных — 1260.

10. Максимальная длина библиотеки моделей — 320 000 символов и максимальное количество моделей (разделов) в библиотеке — 100.

11. Метка в составном операторе не должна содержать символы русского алфавита.

12. Не рекомендуется использовать идентификаторы, начинающиеся символом ь.

13. Если текст значения типа строки имеет длину меньше, чем описано, то текст дополняется до заказанной длины пробелами справа.

14. Максимальное количество переменных в описании действий — 100.

15. Максимальное количество параметров в описании отношения — 25.

16. Максимальное количество параметров в описании подпрограммы и функции — 25.

17. Текст программы может быть отперфорирован в колонках с 1 по 71. Позиции 72—80 при трансляции не обрабатываются.

18. Если текстовое значение не помещается в одной строке, то его запись может быть продолжена на следующих строках. Другие лексемы нельзя продолжать на следующей строке.

19. В программе могут использоваться пробелы. При трансляции они игнорируются.

ГЛАВА 3

НЕКОТОРЫЕ СВЕДЕНИЯ О РЕАЛИЗАЦИИ СИСТЕМЫ ПРИЗ ЕС

3.1. ОБЩАЯ СТРУКТУРА И ПОРЯДОК РАБОТЫ СИСТЕМЫ

Система ПРИЗ разделяется на ряд самостоятельных программ, к которым относятся:

транслятор TRN;
генератор программ GEN;
ассемблер IEUASM;
редактор связей IEWL.

На рис. 13 приведена информационная схема, показывающая системные программы и наборы данных:

TEKST	— исходный текст на входном языке системы ПРИЗ;
MODLIB	— библиотека моделей;
MCRLIB	— макробиблиотека системы ПРИЗ;
MZT	— модель программы;
UPT	— управляющая программа;
ASMOD	— управляющая программа вместе с модулем решения задачи;
MACLIB	— макробиблиотека ассемблера;
LINMOD	— объектный модуль;
PROGLIB	— библиотека модулей;
MAIN	— загрузочный модуль, выполнением которого решается задача.

Так как система ПРИЗ — многоязыковая система программирования, то исходная программа (TEKST), вводимая в машину с дисплея или с перфокарт, может быть программой, написанной на языке УТОПИСТ, или на его расширении, или на макроязыке системы, а также программой, написанной на языке Ассемблера или на Форт-

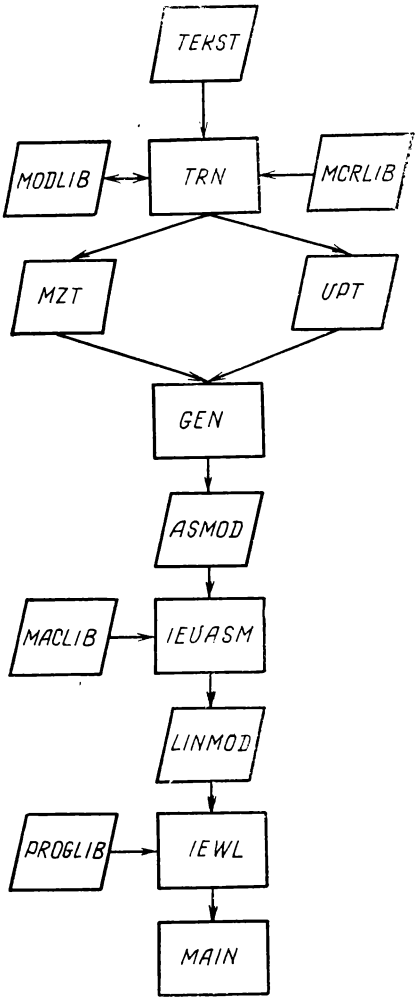


Рис. 13. Общая схема системы ПРИЗ

ране и содержащей специальные операторы языка УТОПИСТ — операторы задачи. От выбранного входного языка зависит, какие системные программы работают при переводе и выполнении данной исходной программы и какие наборы данных они для этого используют. При этом набор данных со сколь угодно сложной структурой, если эта структура точно описана, считается переменной системы. Например, если исходная программа написана на языке УТОПИСТ и содержит лишь описания данных, то из системных программ работает транслятор TRN, который для построения модели программы (модели предметной области или ее подмодели) не использует ни макротеку MCRLIB, ни библиотеку моделей MODLIB. Но однажды составленную модель целесообразно сохранить, чтобы не повторять описание понятий в каждой программе. Для этого, как правило, в программу включают директиву ВВЕСТИ, и модель записывается в библиотеку моделей MODLIB. После этого можно уже программировать на расширении языка УТОПИСТ, причем при трансляции программы транслятор автоматически использует библиотеку моделей в качестве входной переменной. Если программа предназначена и для изменения модели предметной области, то MODLIB придется считать и выходной переменной транслятора.

Такое применение моделей предметной области является гибким проблемно-ориентированным средством автоматизации программирования. Гибкость такой системы заключается в том, что модель в библиотеке может меняться и дополняться независимо от программ, которые эту модель используют.

Библиотека моделей является набором данных с прямой организацией, структура которой имеет сходство со структурой библиотечного набора данных. Как набор данных с библиотечной организацией, так и библиотека моделей MODLIB состоит из разделов, каждый из которых имеет свое имя. Кроме того, оба содержат каталог, размещающийся в начале набора данных. Различие состоит в организации разделов. Разделы библиотечного набора данных организованы последовательно, что не допускает выборочной обработки раздела. Но поскольку для системы ПРИЗ такая возможность необходима, то в библиотеке моделей используют разделы с прямой организацией. Для обслуживания библиотеки созданы специальные утилиты.

Библиотека моделей создается утилитой TYARH (см. гл. 4) с помощью специального режима базисного последовательного метода доступа — режима создания набора данных с прямой организацией. Выполнением этой утилиты создается и каталог. Каталог содержит имена разделов (моделей), их относительные адреса внутри библиотеки и длину моделей. Таким образом, через каталог легко найти адрес любой модели. Для повышения быстродействия системы дальнейшая обработка модели осуществляется с помощью программ виртуальной памяти, обеспечивающих работу с математической памятью большого размера, выполняя автоматически подкачку страниц с дисковой памяти.

Набор данных MODLIB не допускает объединения нескольких библиотек, поэтому если в программе ссылаются на модели, которые находятся в разных библиотеках, то вызову системы ПРИЗ должно предшествовать обращение к утилите KOPEER (см. гл. 4). Утилита KOPEER позволяет создать копию отдельной модели из одной библиотеки, включив ее в другую.

Обработка набора данных MODLIB в какой-то степени аналогична обработке библиотечных наборов данных в операционной системе ОС ЕС. Так как при внесении изменений в некоторый раздел его длина в общем случае может увеличиваться, то измененный раздел не записывается на старое место, а для него отводится память на новом месте. При удалении раздела уничтожается лишь название этого раздела в каталоге, пространство, которое было занято удаленным разделом, нельзя использовать до тех пор, пока набор данных не будет переорганизован. Но для выполнения уплотнения набора данных в системе ПРИЗ не требуется вмешательство пользователя. Когда в библиотеке недостаточно места для размещения нового раздела, то система сама обращается к специальной программе «сборки мусора», которая устраняет неиспользуемые участки внутри набора данных.

Важным обстоятельством с точки зрения расширения возможностей, позволяющих любому пользователю развить входной язык системы наиболее удобным для себя образом, следует считать возможность создания макроязыка. Подготовленные с этой целью макроопределения содержатся в наборе данных MCRLIB. В общем случае в системе имеются две макротеки — одна макротека пользователя, куда уже заранее внесены макроопределения, и временная макротека, созданная во время работы системы и используемая для запоминаний макроопределений, которые записаны непосредственно в исходную программу. Обе макротеки расположены на диске, состоят из набора макроопределений и каталога, содержащего имена отдельных макроопределений и их относительные адреса. Для создания и обслуживания макротеки также созданы специальные утилиты.

Основными составными частями программ, написанных на языке УТОПИСТ, являются описания объектов и операторы. Операторы применяются для указания различных действий, которые должны выполняться программой, а описания — для определения смысла, приписанного разным идентификаторам и именам, используемым в программе.

Таким образом, можно считать, что при использовании транслятора TRN преследуются две различные цели: во-первых, по описанию данных создать модель программы MZT, во-вторых, сформулировать управляющую программу на языке Ассемблера для выполнения операторов, заданных на языке УТОПИСТ (UPT). Кроме того, исходная программа может содержать директивы, т. е. операторы, выполняемые во время трансляции. Обращение к программам, реализующим действия, требуемые директивами, происходит также из программы TRN.

Обе выходные переменные транслятора — MZT и UPT — находятся на диске. MZT является набором данных с прямой организацией. Чтение и запись данных осуществляются программами виртуальной памяти. Набор данных MZT отличается от библиотеки моделей лишь тем, что в нем нет каталога, так как он содержит только одну модель — модель программы. Последовательный набор данных UPT состоит из записей фиксированной длины, каждая запись содержит макрокоманду или команду Ассемблера.

Если программа, написанная на языке УТОПИСТ, содержит оператор задачи т. е. описание задачи, для решения которой необходим автоматический синтез алгоритма, то транслятором TRN записывается в управляющую программу обращение к синтезируемому модулю, за которым непосредственно следует оператор задачи в виде комментария. Очевидно, что с этим трансляция входной программы с языка УТОПИСТ на язык Ассемблера еще не закончена. Для дальнейшей обработки оператора задачи в системе имеется большая программа — генератор программ GEN. Генератор GEN предназначен для автоматического составления модуля решения задачи, при этом он осуществляет поиск оператора задачи в управляющей программе, синтез алгоритма решения задачи, оформление модуля решения задачи на языке Ассемблера и его присоединение к управляющей программе. Для этого в качестве входных переменных генератора используют наборы данных UPT, MZT и MODLIB. Модель задачи, содержащая требуемую информацию для синтеза алгоритма, чаще всего является моделью программы или ее подмоделью в MZT, но может находиться и в библиотеке моделей MODLIB.

ASMOD является последовательным набором данных и находится на диске. Он состоит из записей фиксированной длины, длина каждой записи 80 байт. Такая структура обусловлена тем, что набор данных ASMOD определяет программу, дальнейшая обработка которой осуществляется Ассемблером, т. е. для Ассемблера она представляет собой исходную программу.

Две программы, входящие в систему ПРИЗ, — Ассемблер IEUASM и редактор связей IEWL — являются компонентами операционной системы ОС ЕС. Ассемблер получает из набора данных ASMOD программу на языке Ассемблера и создает эквивалентную ей программу на машинном языке вместе с информацией для редактора связей, т. е. выдает объектный модуль в набор данных LINMOD. Для этого Ассемблер использует еще набор данных MACLIB, содержащий макроопределения. Это библиотечный набор данных, в котором каждое макроопределение является отдельным разделом с именем, являющимся мнемоническим кодом макрокоманды. Макротека системы ПРИЗ объединена с макротекой операционной системы ОС ЕС.

LINMOD в дальнейшем используется как входной набор данных для редактора связей, который редактирует модули и объединяет их в единый загрузочный модуль, направляющийся в набор данных MAIN. При этом все вызываемые модули должны быть

доступны редактору связей. Для этого служит набор данных PROGLIB, содержащий как объектные, так и загрузочные модули (см. гл. 4). Выполнением созданного загрузочного модуля реализуются все действия, которые были заданы в программе, написанной на языке УТОПИСТ.

Рассмотрим функционирование системы на конкретном примере. Допустим, что библиотека моделей MODLIB содержит модель ЭЛЕКТР-ЦЕПИ, куда введено достаточно много содержательной информации об электрических цепях и составлены программы: автоматически вызываемые модули, реализующие отношения между понятиями модели. Поскольку термины языка, например РЕЗИСТОР, ДВУХПОЛЮСНИК и т. д., имеют смысл, близкий к их обычному смыслу в данной области, то входной язык можно считать довольно выразительным, но его синтаксис все же должен удовлетворять синтаксическим требованиям языка УТОПИСТ. Предположим, что для избавления от данного ограничения на этом же языке составлен еще ряд макроопределений, внесенный в набор данных MCRLIB. Пользователь имеет возможность описывать свои задачи на языке, который по форме приближается к естественному языку. Для примера приведем следующую программу:

```
ЗАДАЧА ПО ЭЛЕКТР_ЦЕПЯМ.
РЕЗИСТОР R1 СОЕДИНЕН ПАРАЛЛЕЛЬНО
С РЕЗИСТОРОМ R2.
ЗАДАНО R1.R = 5, R1.I = 2.
НАЙТИ R2.U.
```

При трансляции программы все четыре макрокоманды (в примере подчеркнуты) заменяются требуемой последовательностью описаний из макроопределений. После работы макропроцессора, входящего в состав транслятора TRN, программа имеет следующий вид:

```
ПРОГРАММА' ЭЦ;
ПО' ЭЛЕКТР — ЦЕПЯМ;
-----------------
ПУСТЬ'
   R1    : РЕЗИСТОР;
   R2    : РЕЗИСТОР;
   R0001 : ПАРАЛЛЕЛЬНО R1, R2;
-----------------------------
ЗАДАНО' R1.R = 5, R1.I=2;
-----------------------------
ДЕЙСТВИЯ'
      НА' ЭЦ ВЫЧИСЛИТЬ' R2.U;
      ВЫЗОВ' ВЫВОД (R2.U);
КОНЕЦ' +++
-----------------------------
```

В результате работы транслятора TRN набор данных MZT содержит модель программы:

```
ЭЦ (R1 (I ВЕЩ' ; U ВЕЩ' ; R ВЕЩ' ;
        МОД' ОНМ ПРО' 1 СЛ'I СЛ'U СЛ'R ) ;
```

```
   R2 (I ВЕЩ' ; U ВЕЩ' ; R ВЕЩ' ;
      МОД' ОНМ ПРО' 1 СЛ'I СЛ'U СЛ'R ) ;
   R0001 (R1 (I ВЕЩ' ; U ВЕЩ' ; R ВЕЩ' ) ;
          R2 (I ВЕЩ' ; U ВЕЩ' ; R ВЕЩ' ) ;
            I ВЕЩ' ;
            U ВЕЩ' ;
            МОД' НАПР ПРО' 2 СЛ' R1.U СЛ'R2.U СЛ'U ;
            МОД' ТОКИ ПРО' 1 СЛ' R1.I СЛ'R2.I СЛ'I ;
            МОД' ЬЬЬЬЬЬЬ ПРО'1 СЛ'R1 СЛ'ЭЦ. R1;
            МОД' ЬЬЬЬЬЬЬ ПРО'1 СЛ'R2 СЛ'ЭЦ. R2) ;
МАК' МАКRO # # # ПРО'2 ВЫХ'R1.R ВЫХ' R1.I ;
<МАКРО # # # &PM; BC 15, # 001 &SYSNDX;
#002 & SYSNDX DC E' 5' ;
#001 &SYSNDX MVC &PM(1), #002&SYSNDX;
BC 15, # 003 &SYSNDX;
#004 &SYSNDX DC E' 2';
#003 &SYSNDX MVC &PM(2),#004&SYSNDX>) ;
```

Управляющая программа UPT выглядит следующим образом:

```
                  SAVM UPT
                  CALL SYSPRZ, (# 001,   ¤   001), VL
        * НА      ЭЦ ВЫЧИСЛИТЬ   R2.U;
                  CALL ВЫВОД, (   ¤   001),VL
                  LOPU
        ¤ 001     DS F
        # 001     DC F' 1'
                  END
```

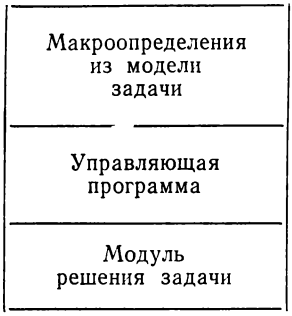


Рис. 14. Структура сгенерированной программы

Результатом работы генератора GEN является готовая программа в наборе данных ASMOD. Структура программы показана на рис. 14. В начале программы находятся те макроопределения, которые составлены системой для реализации отношений из модели задачи, т. е. макроопределения, являющиеся автоматически вызываемыми модулями. Затем следует управляющая программа. Генерируемый модуль решения задачи оформляется в соответствии с соглашениями, принятыми в ОС ЕС, как секция управляющей программы. Более подробно модуль решения задачи описан в п. 3.3.

Программируя на вложенном входном языке системы, пользователь пишет управляющую программу на языке Ассемблера или на языке Фортран (см. гл. 4). Модель задачи в любом случае берется из библиотеки моделей. В этом случае транслятор TRN не используется.

Генератор программ сам вводит входной текст в набор данных UPT и подыскивает модель задачи из набора данных MODLIB. Набор данных MZT не создается. Если входная программа написана на языке Ассемблера, то ее дальнейшая обработка ничем не отличается от обработки программы, написанной на языке УТОПИСТ. Но если применен язык Фортран, то после работы генератора происходит обращение к транслятору с языка Фортран.

3.2. ТРАНСЛЯТОР

Поступающий на вход системы текст задачи переводится на внутренний язык транслятором, который учитывает семантику понятий входного языка, заданную в модели предметной области. Анализ и переработка текста производятся по предложениям.

Транслятор системы ПРИЗ состоит из блоков лексического анализа — LEKS, синтаксического анализа — ANALY и блока семантических программ — SEM. Информационная схема транслятора изображена на рис. 15.

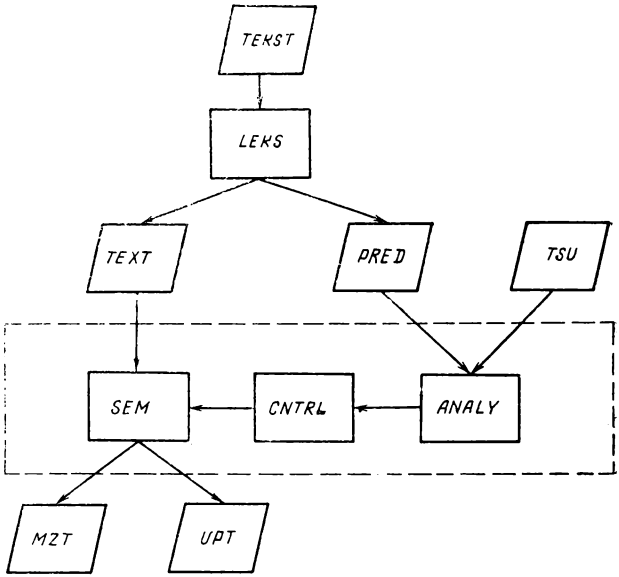


Рис. 15. Общая схема транслятора

Анализ входного текста (TEKST) распадается на два последовательных этапа. Задача начального этапа, лексического анализа состоит в разложении исходной последовательности символов на представление лексем языка. Для этого символы, составляющие исходную программу, считываются и группируются в отдельные лексемы, устраняются несущественные пробелы и комментарии. При этом конкретные представления лексем заменяются внутренними кодами.

Конструкции языка, которые будут выделяться как лексемы, следующие: ключевые слова, идентификаторы, числа, разделители, элементарные строки. Распознавание более сложных конструкций отложено до фазы синтаксического анализа.

Лексический анализатор можно рассматривать состоящим из ряда последовательно соединенных конечных преобразователей. Первый преобразователь в этой цепи устраняет из исходной прог-

раммы все несущественные пробелы, второй — ликвидирует комментарии, третий — ищет идентификаторы и т. д. Очевидно, что если изменяются, например, принятые в языке ключевые слова, то это отражается лишь на лексическом анализе.

В ходе лексического анализа исходная программа преобразуется в переменные TEXT и PRED, являющиеся входными для синтаксического анализатора. Переменная TEXT является внутримашинным описанием исходного текста. В переменной PRED конкретные лексемы уже заменены соответствующими внутренними кодами. Она состоит из пар вида (тип лексемы, указатель). Первой компонентой пары является синтаксическая категория — «число» или «идентификатор», второй — указатель, который указывает относительный адрес конкретной лексемы в переменной TEXT.

Задача второго этапа трансляции заключается в том, чтобы по описанию данных создать модель задачи и по описанию действий — управляющую программу. В общем случае для выполнения этой задачи в дополнение переменным TEXT и PRED транслятор TRN использует библиотеку моделей MODLIB, содержащую ранее созданные понятия. В данной реализации представление модели выбрано так, чтобы оно совпадало по своей структуре со входным текстом, но синтаксис был бы проще (см. приложение 5). Такой подход позволяет во многих случаях упростить семантические подпрограммы транслятора. Довольно много лексем языка переносятся из переменной TEXT в переменную MZT без изменений. Выбор текстового представления оправдывается тем, что обработка текста в основном происходит последовательно. Относительно редко приходится использовать отдаленные объекты, и, не считая отношений, структура модели является деревообразной. Работа с отношениями требует поиска объекта по имени, что тоже не представляет особых трудностей. В пользу этого представления говорит и то, что для хранения моделей предметных областей в виде списка потребовалось бы больше памяти.

В ходе составления модели задачи транслятор создает для каждой конструкции УРАВ, ПРИСВ и ЗАДАНО, а также в случае задания начальных значений в переделках соответствующее макроопределение, которое сохраняется также в переменной MZT. В тексте модели макроопределение ограничивается символами «<>».

В трансляторе применяется синтаксически управляемая переработка текста. Язык определен таблицей синтаксического управления, полученной путем специального представления синтаксиса исходного языка с включением ссылок на семантические программы. Таблица синтаксического управления является, по существу, псевдопрограммой, которая должна выполняться специальным интерпретатором. Таким интерпретатором и является синтаксический анализатор.

Для записи правил грамматики языка УТОПИСТ для синтаксического анализатора используется специальный метаязык К1 [3].

В правиле выделяют левую часть — нетерминальный символ и

правую часть — последовательность частичных определений, которые представляют собой цепочку основных конструкций языка.

Основные конструкции языка К1 следующие: конструкции терминального типа, нетерминальные, трансдукторы, параметры.

Терминальными конструкциями являются терминалы и интервалы. Последние в свою очередь делятся на простые и предварительные интервалы. С их помощью может быть представлена отдельная лексема входного языка (терминалы) или одна из групп лексем, коды которых составляют некоторый интервал чисел (интервалы).

Каждый нетерминальный символ является левой частью некоторого правила.

Трансдуктор можно рассматривать как вызов некоторой семантической программы. Семантические программы, связанные с трансдукторами, могут иметь параметры, которые записываются прямо в правилах вслед за соответствующим трансдуктором. Параметры могут быть общими или локальными.

Правила грамматики текста на языке К1 переводят в закодированную таблицу синтаксического управления TSU, которая вместе с переменной PRED является входной информацией синтаксического анализатора (см. рис. 15). Синтаксический анализатор состоит из двух самостоятельных программ — ANALY и SEM. Цель программы ANALY — синтаксический анализ, выявление синтаксических ошибок во входном тексте и выдача сообщений о них, а также продолжение трансляции после обнаружения ошибок. Появление в таблице синтаксического управления TSU трансдуктора указывает на необходимость выполнения одной из семантических подпрограмм. В таблице TSU нет явных ссылок на семантические подпрограммы. Связь номера трансдуктора с конкретной программой определена в программе SEM, которая, кроме управляющей части, содержит набор семантических подпрограмм. По полученному из переменной CNTRL информации программа SEM организует вызов соответствующей семантической подпрограммы. Кроме номера трансдуктора, переменная CNTRL содержит и необходимую для выполнения подпрограммы информацию — номер выхода из нее, регистр ветвления, регистры чтения и хранения входных символов, номер ошибки и т. д. Такой подход дает возможность отладить синтаксический анализатор до написания всех семантических подпрограмм. Удобным является и тот факт, что внесение изменений в семантические подпрограммы или их замена никак не влияют на программу анализа ANALY.

Семантические подпрограммы (семантики) делятся на обычные и специальные. Исходя из того, что текст задачи на входном языке состоит из двух самостоятельных частей — описания данных и описания действий, — обычные семантики в свою очередь делятся на семантики перевода условий задачи и семантики перевода действий. Кроме этих двух основных групп, имеются еще семантики выполнения директив. Директивы выполняются в ходе работы транслятора, при этом его состояние не меняется.

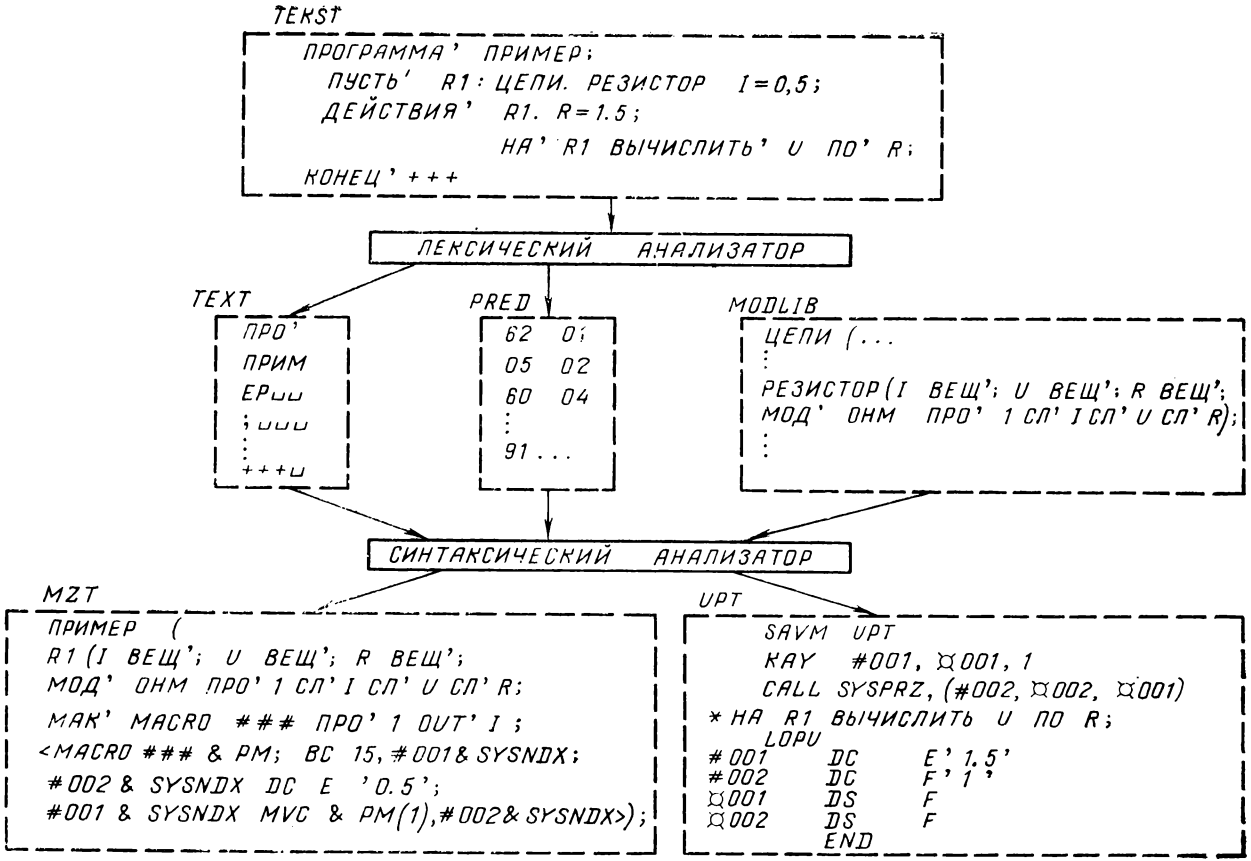


Рис. 16. Пример трансляции программы, написанной на языка УТОПИСТ

Специальные семантические подпрограммы (спецсемантики) влияют на синтаксический анализ. В состав спецсемантики входят программы продолжения переработки текста при обнаружении ошибки, программы для изменения состояния магазина и организация ветвлений и циклов в процессе синтаксического анализа.

Работу транслятора лучше всего проиллюстрировать, показывая содержимое внутренних переменных системы на различных стадиях трансляции. На рис. 16 показаны этапы перевода одной очень простой программы. Для простоты в примере предполагается, что исходная программа не содержит макровызовов. Применение макросредств системы рассматривается отдельно в гл. 4.

Первой программой, с которой начинается трансляция, является лексический анализатор LEKS. Считывание текста и пересылка его (по одной строке за цикл) в буфер ввода управляющей программы лексического анализатора выполняются соответствующей программой чтения (с перфокарт или дисплея). Анализ текста начинается с проверки первого символа. Если он «*», то строка воспринимается как комментарий и все остальные символы игнорируются. Но если это не так, то обращаются к модулям, преобразующим исходный текст, при этом анализатор пропускает все пробелы, стоящие в начале той строки, которую она обрабатывает, а также пробелы между лексемами. В приведенном примере первая лексема — ключевое слово, сравниваемое со списком ключевых слов. Если ключевое слово содержится в списке, то оно вносится в переменную TEXT, в которой каждому из ключевых слов отводится по одному слову. Затем его относительный адрес (1) и внутренний код (62), сформированный по специальной таблице, объединяются вместе и в переменной PRED также занимают одно слово.

Для идентификатора в переменной TEXT отведено два слова. Если идентификатор содержит более восьми символов, то учитываются только первые восемь из них, а если идентификатор содержит менее восьми символов, то он дополняется пробелами.

Составное имя в примере ЦЕПИ.РЕЗИСТОР на лексическом уровне не трактуют как лексический элемент, а как

<идентификатор> <разделитель> <идентификатор>.

Так как синтаксический анализ текста проводится по предложениям, то по появлению признака конца предложения «+,+» управление передается синтаксическому анализатору.

Стоящий первым в переменной PRED код лексемы ПРО' сообщает анализатору, что начинается новая программа, и указывает, что должен следовать идентификатор, который является именем программы. Разделитель же «;» после идентификатора будет означать конец конструкции. В ходе обработки этой конструкции задаются начальные значения различным счетчикам и переменным состояния. Кроме того, создается набор данных с прямой организацией MZT, куда включают пустую модель программы — ПРИМЕР.

Если же в самом начале вводимого текста обнаруживается лексема, отличная от лексемы ПРОГРАММА' (или ПОДПРОГРАММА'), то выполняется программа обработки ошибок, которая отпечатает сообщение о характере ошибки и место в тексте, где она была обнаружена. Ошибка в какой-то конструкции языка приведет к пропуску оставшейся части этой конструкции или же, если программа выполняется в режиме диалога, к запросу для ввода новой строки.

То, что идентификатору в описании данных следует разделитель «:», показывает, что это описание нового объекта, и поэтому происходит обращение к подпрограмме, которая отыскивает описание его типа. Сначала просматривается модель программы. Поскольку там объекта с именем ЦЕПИ.РЕЗИСТОР не существует, то обращаются к библиотеке моделей. Транслятор находит из библиотеки модель с именем ЦЕПИ, из нее — понятие РЕЗИСТОР и копирует его описание в MZT (с новым именем R1). При этом проверяют, не используется ли понятие ЦЕПИ.РЕЗИСТОР или его компоненты в качестве параметров какого-то отношения внутри описания. Если это так, то имя ЦЕПИ.РЕЗИСТОР заменяют именем R1. Это происходит и тогда, когда оно содержится в составном имени. Например, если в описании отношения было бы имя ЦЕПИ. РЕЗИСТОР. R, то оно заменилось бы именем R1.R.

Затем в описание объекта R1 добавляют отношение, задающее значение компоненту I:

```
МАКРО' MACRO # # #   ПРО' 1 ВЫХ' I;
```

Тогда будет сформировано и макроопределение, реализующее данное отношение. Оно содержит команду пересылки константы соответствующему объекту. Любая константа, появляющаяся в переделках, а также в конструкциях УРАВ, ПРИСВ или ЗАДАНО, переводится в предложение DC, в поле метки которого указано имя, генерируемое системой. Целые и вещественные константы определяются как константы типа F и E соответственно, которые определяют четырехбайтовые числа внутри макроопределения. Под текстовую константу отводится поле с числом байт, равным не числу написанных символов, а равным длине объекта, которому это значение присвоено. Рабочая память, необходимая при реализации арифметических и логических выражений для хранения промежуточных результатов, выделяется также внутри макроопределения.

Для перехода от описания действия к управляющей программе UPT составлен ряд макроопределений — макроопределения для реализации всех операторов входного языка и макроопределения, используемые при трансляции арифметических и логических выражений. При таком подходе генерируемая программа UPT состоит главным образом из последовательности макрокоманд, если необходимо, то с параметрами. Однако имеются и команды Ассемблера.

Первой макрокомандой должна быть макрокоманда SAVM, определяющая имя программы, базовые регистры и точку входа в

нее. Она обеспечивает сохранение регистров вызывающей программы и резервирование области сохранения.

При трансляции проверяют, что любое составное имя, которое используется в описании действий, предварительно определено либо в модели программы, либо в библиотеке моделей.

Знак присваивания «:=» указывает, что значение выражения, данного справа (в нашем случае просто число 1,5), следует присвоить объекту, указанному в левой части (R1.R). При этом заданный объект должен быть вещественного типа. Обработка констант, появляющихся в операторах, аналогична их обработке в описании данных с той лишь разницей, что под эти константы отводится отдельный участок памяти в конце программы. Каждый раз, когда встречается константа, просматривается этот участок, и константа включается туда только тогда, когда такой там не было. Итак, результатом трансляции оператора присваивания R1.R=1.5 является запись в управляющую программу предложения DC (с меткой #001) и макрокоманды КАУ с именем константы в качестве параметра.

По оператору задачи в управляющую программу включают обращение к синтезируемому модулю решения задачи

CALL SYSPRZ, (#002, ¤ 003, ¤ 001),VL

Однако для системы ПРИЗ это обозначает необходимость вызова программы GEN, составляющей данный модуль. Поэтому непосредственно после макрокоманды CALL записывается в виде комментария оператор задачи. Такое решение проблемы удовлетворяет двум требованиям: во-первых, при анализе текста управляющей программы системная программа GEN получит необходимую информацию для автоматического синтеза модуля и, во-вторых, без всяких переделок при выполнении этой программы обращаются к готовому модулю.

Так как исходная программа может содержать несколько описаний задач, то каждая задача имеет метку в виде целого числа, отличную от меток других задач в данной программе. Эта метка входит в качестве первого параметра в обращении к модулю.

По достижении конца программы (КОНЕЦ') транслятор формирует область данных, содержащую, кроме определенных программой констант, такие переменные программы с именами ¤ N, где N — порядковый номер переменной. При выделении памяти каждой переменной ставится в соответствие столько машинных слов, сколько определено в ее описании. Затем проводится проверка, не содержала ли программа оператора задачи. Если содержала, то управление передается генератору GEN, в противном случае обработка программы закончена.

3.3. ГЕНЕРАТОР ПРОГРАММ

Генератор системы ПРИЗ предназначен для реализации оператора задачи. Он воспринимает оператор задачи на входном языке и переводит его на внутренний язык системы, определяет после-

довательность модулей, которые необходимо выполнить для решения задачи, организует связь между модулями, объединяя их в программу решения задачи. Программа решения задачи составляется в окончательном виде до начала вычислений, т. е. генератор программ системы ПРИЗ является генератором компилирующего типа. Преимущество таких генераторов заключается прежде всего в том, что они позволяют составлять крупные программы хорошего качества, которые можно сохранять и использовать многократно для решения задач.

Генератор состоит из следующих блоков: обработки оператора задачи SKASK, преобразования модели PRD, планировщика PLAN, распределителя памяти MALU и компилятора RED. Работой этих блоков управляет управляющая программа генератора.

Информационные связи между отдельными частями генератора показаны на рис. 17.

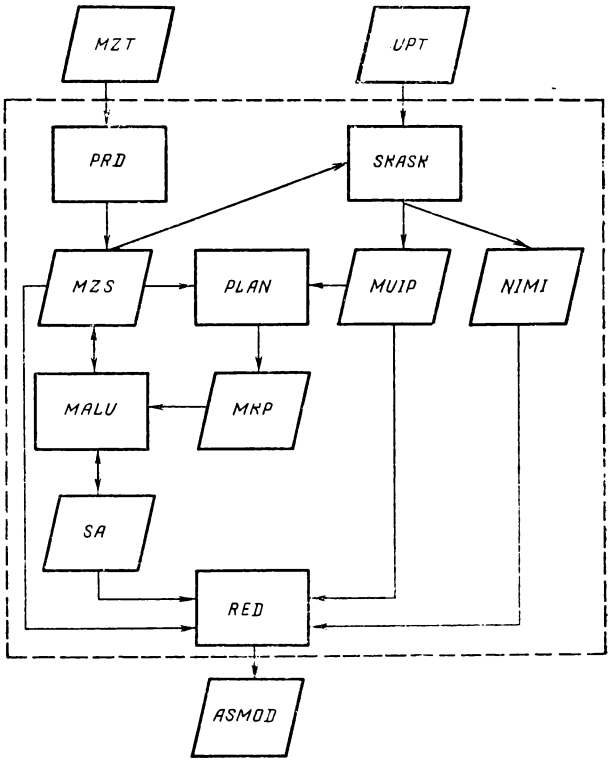


Рис. 17. Общая схема генератора:

MZT — текстовая модель; UPT — управляющая программа; MZS — списковая модель; MUIP — управляющая информация для планировщика; NIMI — номер генерируемого модуля; MKP — схема доказательства разрешимости задачи; SA — количество требуемой памяти в словах; ASMOD — генерируемая программа

Первыми выполняются те программы генератора, назначение которых — приведение системы в состояние готовности к автоматическому синтезу алгоритма решения задачи. При этом сначала осуществляется поиск оператора задачи в тексте управляющей программы. Когда из оператора задачи выделено имя модели, на которой задана задача, начинает работать подпрограмма поиска моделей. Модель задачи может быть моделью любого понятия в библиотеке моделей или моделью любого объекта, описанного в данной программе. Во всех случаях модель находится в виде текста на диске. Но наряду с неоспоримыми достоинствами такого представления модели оно имеет и существенный недостаток — значительная затрата машинного времени на планирование. Во избежание этого необходимо перейти к такому представлению модели, которое способствует эффективному планированию алгоритма решения задачи. Это делается программой PRD, на выходе которой модель представляется в виде списка и загружается в оперативную память. В виде списка представлены как входные, так и внутренние и выходные переменные планировщика, что обеспечивает удобную возможность оперировать адресами элементов, а не самими элементами, благодаря чему облегчается программирование и экономится память. В начале работы генератора вся доступная память, отведенная для хранения списковых переменных, разбивается на ячейки, которые связываются в список свободных ячеек. При образовании новых переменных из этого списка могут быть взяты свободные ячейки, аналогично они возвращаются в список после того, как потребность в них отпадает.

В модели задачи MZS каждой переменной и отношению сопоставляется множество в виде списка, элементами которого являются связи данного элемента со смежными элементами. В таком случае переменной ставят в соответствие множество связей данной переменной с отношениями, связанными с ней. Отношению — множество связей с переменными, связанными с данным отношением. Таким образом, все связи, имеющиеся между переменными и отношениями, описываются два раза. Это требует дополнительного объема памяти, но ускоряет работу при планировании решения задачи.

Используемые для представления множеств списки являются несимметричными циклическими списками. Список состоит из оглавления и элементов и идентифицируется ссылкой на оглавление списка. Оглавление содержит общие сведения этого списка, элемент — ссылку на оглавление списка соответствующего отношения или объекта, ссылку на следующий элемент, а также тип связи — способ использования объекта. Например, признак того, что связанный с отношением объект используется в качестве выходной переменной, записан в элемент списка.

Оглавлением списка, который представляет собой объект модели, является структура из шести полей:

название объекта;

указатель размещения (относительный адрес);

ссылка на эквивалентный объект;
тип объекта;
длина объекта;
ссылка на первый элемент.

Поскольку все объекты строятся в конечном счете из компонентов, принадлежащих к элементарным, то на этом этапе по типу различаются только составные и элементарные объекты. Таким образом, тип объекта в оглавлении может иметь одно из следующих значений — составной, целый, вещественный, логический, текстовой, неопределенный или память.

Оглавление списка, представляющего отношение, состоит из следующих полей:

названия отношения;
названия реализующего модуля;
типа отношения;
ранга (степени функциональности);
ссылки на первый элемент.

При переходе от текстовой модели к списковой выполняется еще ряд действий.

1. Строятся структурные отношения, которые в текстовой модели заданы неявно структурой объекта. Структурное отношение — это специальный тип отношения, связывающего составной объект с его непосредственными компонентами.

2. Между эквивалентными объектами и их компонентами формируются ссылки, которыми пользуются при планировании задачи и распределении памяти. При обработке очередного отношения эквивалентности проверяется, не объявлен ли связанный с ним объект уже ранее эквивалентным с каким-то другим объектом. Если это так, то указатели эквивалентности будут еще раз обрабатываться. Таким образом, все объекты модели, определенные как эквивалентные, будут связаны в цепочку. При составлении цепочек эквивалентности могут представиться следующие возможности:

один из объектов имеет указатель эквивалентности;
оба объекта имеют указатели эквивалентности;
повторное объявление эквивалентности, т. е. указатели эквивалентности обоих объектов входят в одну и ту же цепочку.

3. Если описание отношения содержит шаблон имени (см. гл. 2), то автоматически оформляется список имен связанных объектов.

4. Макроопределения выносятся из текста модели в начало набора данных ASMOD, при этом каждому макроопределению приписывают имя, которое заносится и в оглавление соответствующего отношения в MZS.

Результатом этого этапа является еще переменная MUIP, управляющая информация планировщика, созданная по оператору задачи. Переменная MUIP имеет стандартную списковую структуру и содержит два подсписка: список входных переменных и список выходных переменных задачи.

По описанию задачи (MZS, MUIP) планировщик выдает схему доказательства разрешимости задачи МКР или сообщение о

неразрешимости задачи. Доказательство состоит из шагов, соответствующих операторам программы.

Поиск доказательства заключается в определении того, какие шаги (операторы) и в какой последовательности должны выполняться для получения решения задачи.

Каждый шаг доказательства представлен в МКР своим подсписком, оглавление которого содержит адрес, указывающий на соответствующее отношение в MZS. В элементах списка ссылаются на предыдущий и следующий шаги и на входные и выходные переменные.

Следующий этап генерации программы — компиляция — должен обеспечить правильную связь между модулями, реализующими шаги в МКР.

Память для хранения передаваемых между модулями величин определяется программой распределения памяти до оформления модуля решения задачи. При этом память занимается не для всех объектов модели, а лишь для тех, которые участвуют в вычислениях при решении задачи, т. е. для тех, на которые ссылаются в доказательстве. Это позволяет строить и применять большие модели без лишнего расхода памяти.

Программа распределения памяти MALU определяет длину требуемого поля памяти и присваивает переменным относительные адреса внутри этого поля.

Эти относительные адреса и являются указателями размещения в оглавлении объектов в MZS. Кроме того, запоминается максимальный занятый адрес. Окончательный объем памяти не будет установлен до тех пор, пока не будут обработаны все операторы задачи в данной программе.

Выделенная память используется этими переменными все время, пока происходит решение данной задачи. После появления очередного оператора задачи связь между переменными и выделенными ячейками памяти исчезает, и эти ячейки становятся свободными для использования в других вычислениях.

Составному объекту выделяются последовательные ячейки памяти, однако при этом возможны случаи, когда некоторому компоненту уже ранее отведена память. В этом случае составному объекту отводится самостоятельный участок памяти, содержащий и память для данного компонента. Пересылка значения этого компонента из одной области в другую обеспечивается модулями, реализующими структурные отношения. Для хранения значений эквивалентных объектов программа распределения памяти выделяет одну и ту же область памяти.

Система формирует модуль решения задачи на языке Ассемблера, применяя стандартные макрокоманды операционной системы ОС ЕС и макрокоманды, разработанные для системы ПРИЗ. Генерируемые команды помещаются в выходной текст ASMOD, в который уже включены макроопределения из модели и управляющая программа UPT.

Поскольку исходная (управляющая) программа может содержать несколько операторов задачи, то для использования одной входной точки в начале модуля строится переключатель. При обращении к модулю в качестве первого параметра задается номер задачи, являющейся также номером соответствующего участка программы. Это обеспечивает обращение каждый раз к требуемому участку, реализующему оператор задачи.

Задача, которую требуется решить, представляется тройкой (M, U, V), где M — семантическая модель, U — множество входных переменных, V — множество выходных переменных. U, V состоят из объектов, описанных в модели M. Если задача разрешима, то система выдает программу вычисления соответствующей функции $f_{M,U,V}$. Кроме задачи, в операторе задачи задаются еще множество переменных X, которое является аргументом функции $f_{M,U,V}$, и множество переменных Y, которому присваивается значение $f_{M,U,V}(X)$. Таким образом, оператор задачи задает следующий оператор присваивания:

$$Y := f_{M,U,V}(X),$$

где X, Y — множества переменных из программы.

Оператор задачи можно описать несколько подробнее с помощью последовательности трех операторов присваивания. Соответственно можно и отрезок программы разбить на три части. В первой части происходит передача значений входных переменных из управляющей программы в сгенерированный модуль, т. е. переменным модели

$$U := X.$$

Затем следуют макрокоманды, реализующие вычисления

$$V := f_{M,U,V}(U).$$

В конце каждого участка вычисленные значения переменных отправляются в управляющую программу

$$Y : V.$$

После этого управление передается стандартным образом управляющей программе.

Для перехода от программы на языке Ассемблера (ASMOD) к программе, готовой для выполнения, естественно использовать универсальное программное обеспечение ЕС ЭВМ — Ассемблер и редактор связей.

Для пояснения структуры модуля решения задачи приведен пример на рис. 18. Допустим, что управляющая программа содержит два оператора задачи. Обращение к модулю решения задачи SYSPRZ осуществляется с помощью макрокоманды CALL. После того как выполнение вызванного участка модуля завершится, т. е. задача решена, управление возвращается управляющей программе, команде, следующей за макрокомандой CALL. Таким образом, если задачи, описанные в управляющей программе, не содержат

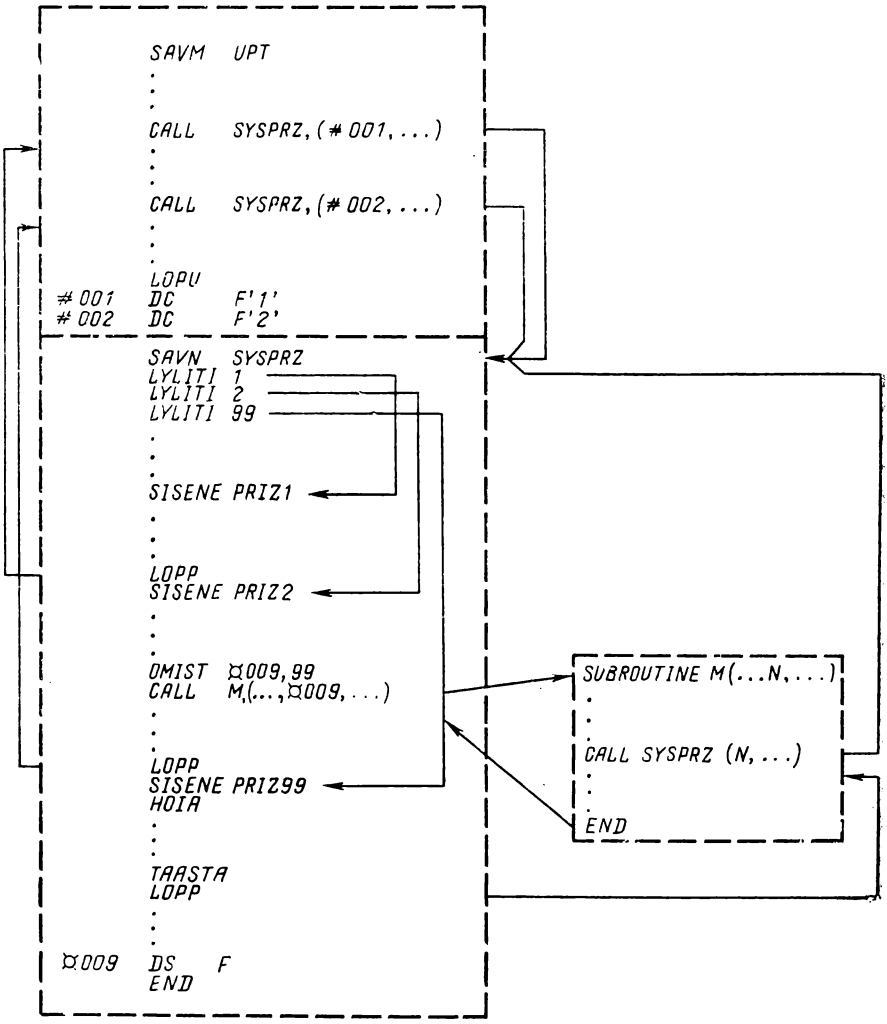


Рис. 18. Структура модуля решения задачи

подзадач, то участки модуля, реализующие отдельные операторы задачи, выполняются последовательно, т. е. к модулю SYSPRZ не обращаются повторно, пока не выполнится вызванный участок. Но модуль решения задачи может содержать автоматически вызываемый модуль, который в свою очередь вызывает модуль SYSPRZ, т. е. требует решения подзадачи. В результате этого выполнение вызванного участка (PRIZ2) прекращается, снова обращаются к модулю SYSPRZ, и управление передается к соответствующему участку (PRIZ99). Выполнение участка PRIZ2 будет продолжено с прерванной команды после выполнения модуля M.

Рассмотрим более детально модуль решения задачи. Как уже говорилось, модуль оформлен как секция управляющей программы.

Первым действием каждого модуля должно быть сохранение содержимого общих регистров. Для этого предназначена макрокоманда SAVN. Поскольку модуль решения задачи должен быть реентерабельным, то для сохранения регистров следует пользоваться временными областями памяти. Временные области памяти получены динамически макрокомандой SAVN по содержащейся в ней макрокоманде супервизора GETMAIN.

Затем строится переключатель. Всякий раз, когда обрабатывается новый оператор задачи, в переключатель записывается макрокоманда LYLITI, параметром которого является номер задачи. Любой участок программы, реализующий один оператор задачи, начинается макрокомандой SISENE с тем же номером в качестве параметра и завершается макрокомандой LOPP.

По макрокоманде LYLITI обращаются к участку программы, начинающемуся с соответствующей макрокоманды SISENE. Макрокоманда LOPP восстанавливает исходное значение регистров, освобождает область сохранения (по макрокоманде FREEMAIN) и передает управление в программу, использующую данный участок.

По соглашению номер задачи является первым параметром обращения к модулю решения задачи, при этом полагается, что его значение определено в управляющей программе либо транслятором (в случае программирования на независимом входном языке), либо пользователем (в случае программирования на вложенном входном языке). Но если в ходе решения задачи возникает потребность в выполнении другой задачи — подзадачи, т. е. автоматически вызываемый модуль также требует автоматический синтез программы, то этому участку программы генератор сам присвоит номер макрокомандой OMIST.

Перед тем как начинаются вычисления для решения подзадачи (PRIZ99), которые могут испортить еще нужное для участка PRIZ2 содержимое некоторых из ячеек памяти, выделенных для переменных модуля макрокомандой MALU, значения всех используемых переменных запоминаются на диске макрокомандой HOIA. При возврате в вызывающий модуль макрокомандой TAASTA восстанавливается содержимое тех ячеек, для которых проводилось запоминание.

Если при вызове системы ПРИЗ в предложении EXEC указан параметр RECU, то генератор не генерирует новый номер для подзадачи, а присваивает ей номер задачи, в которую она входит. Например, соответствующий фактический параметр модуля M приобретает значение, равное 2, а не 99. Таким образом, обращение к участку PRIZ2 повлечет за собой обращение к модулю M. В этом модуле происходит дальнейшее обращение к участку PRIZ2. Он снова обратится к модулю M. Следовательно, модуль будет использоваться рекурсивно. При этом, конечно, предполагается, что один

из операторов, замененный в примере точками, задает некоторое условие окончания этого непрерывного обращения.

Теперь остается посмотреть, как перевести схему доказательства решения задачи МКР, состоящую из последовательных шагов, в форму программы. Если при доказательстве проверяется несколько альтернатив, т. е. в состав модели задачи входят условные отношения, то в программе могут возникнуть разветвления.

Поскольку каждому шагу доказательства соответствует какой-то автоматически вызываемый модуль, то необходимо обеспечить передачу управления от модуля к модулю и передачу величин между ними, а также между модулем решения задачи и вызывающей программой.

Для каждого (кроме первого) входного параметра обращения к модулю решения задачи в начале соответствующего участка программы формируется макрокоманда PARM, предназначенная для передачи значений входных переменных в сгенерированный модуль (U:=X). Параметрами макрокоманды являются адрес, начиная с которого переписывается значение параметра, порядковый номер параметра и длина параметра в байтах. Если фактический параметр окажется состоящим более чем из 256 символов, то формируется несколько команд пересылки. Аналогично в конце участка записаны макрокоманды PART, выполнением которых вычисленные значения пересылаются в вызывающую программу (У:=V).

Затем организуются обращения к модулям, реализующие шаги доказательства МКР ($V:=f_{M,U,V}(U)$). Обращения к модулям могут быть пяти типов в зависимости от типа соответствующего отношения. Напомним, что отношения в модели задачи могут быть типа программа или типа уравнение, при этом модули, задающие эти отношения, оформлены либо как подпрограммы, либо как макроопределения. От этого зависит способ обращения к ним. По-иному обращаются к модулям, которыми заданы структурные отношения. В вызываемый модуль, как правило, необходимо передать значения каких-то параметров. Количество и порядок фактических параметров, указанных в обращении к модулю, соответствуют количеству и порядку параметров отношения в модели задачи.

Простейшим путем передачи управления подпрограмме является использование макрокоманды супервизора

CALL<имя точки входа> , (<параметр> ,..., <параметр>) , VL

Первый операнд команды определяет внешнее имя, которому будет передаваться управление. Параметры, разделенные запятыми, заключаются в круглые скобки и записываются в качестве второго операнда. Имеется и третий операнд VL, благодаря чему знаковый бит последнего адреса устанавливается равным 1. Этим признаком можно пользоваться, когда число аргументов является величиной переменной, для того чтобы легко обнаружить конец списка.

Если требуется выполнить модуль, реализуемый макроопределением, то в генерируемую программу записывается соответствующая этому модулю макрокоманда. В макрокоманде указаны имя модуля

и значения параметров. При трансляции макрокоманда заменяется макрорасширением. Макрорасширение — это не что иное, как текст модуля, скорректированный в соответствии с фактическими параметрами макрокоманды и вставленный на ее место в модуль решения задачи. В поле названия макрокоманды стоит пробел, а в поле операции — имя соответствующего модуля. Используемая макрокоманда является позиционной — значения параметров записываются в том же порядке, в каком перечислены параметры прототипа.

При построении очередного обращения всегда проверяется, имеет ли данное отношение слабосвязанные параметры. Если это так, то обращению предшествует макрокоманда TABLIK, которая передает модулю информацию о том, какие из слабосвязанных параметров придется считать входными, какие — выходными. Определяемый макрокомандой TABLIK информационный вектор имеет следующий вид:

```
BAL 2, * + <N>
DC B'0000000 {0}'
             {1}
.
.
.                  } m строк,
DC B'0000000 {0}'
             {1}
```

где m — число параметров;

$$N = m + 4.$$

Первый оператор DC соответствует первому параметру и т. д. 0 в последнем бите указывает, что параметр является входным, 1 указывает на выходной параметр. Адрес начала информационного вектора загружается в регистр 2. При выполнении программы через этот регистр системная функция OUTPUT (см. гл. 4) получит требуемую информацию для передачи управления тому участку программы, который вычисляет искомое значение.

Отличительной чертой реализации отношений типа уравнения является то, что система сама организует решение уравнений. В расчетную программу вставляется программа численного метода решения уравнений, вызывающая модуль, реализующий отношение как подпрограмму-функцию. Эта подпрограмма вычисляет левую часть уравнения $f(x_1, x_2, \ldots, x_k) = 0$ и загружает результат в регистр 0 (см. гл. 4). Необходимая для программы решения уравнений информация задается следующим информационным вектором:

```
BAL 2,* + 12
DC  V (<имя модуля>)
DC  A (<имя искомого параметра>)
```

В данной версии реализации языка уравнения решаются методом хорд. В режиме диалога пользователю предоставлена возможность самому определить начальные приближения, погрешность интерполяции и число итераций. Кроме того, пользователь может

включить в систему какой-то другой метод решения уравнений. Для этого программу решения уравнений VYRLA в системной библиотеке заменят новой.

Если уравнение задано макроопределением, то обращаться к нему из программы решения уравнений не так просто. Для преодоления этой проблемы создана макрокоманда VYRLAH.

Обращение к модулю структурного типа реализуется как обращение к модулю с именем YHENDA или ERALDA в зависимости от того, необходимо ли вычислить составной объект на основе его компонентов или наоборот. Информационный вектор здесь имеет следующий вид:

```
BAL   2, * + <N>
DC    F'<длина переменной>'
.                                  } m + 1 строк,
.
.
DC    F'<длина переменной>'
```

где m — число компонентов;

$N = 4 * (m + 1) + 4$.

По соглашению первый оператор DC всегда соответствует составному объекту.

Вернемся к примеру на рис. 16. В результате работы генератора программа в наборе данных ASMOD выглядит следующим образом:

```
                MACRO
                MACR1     &PM
                BC        15,#001&SYSNDX
#002&SYSNDX     DC        E'0.5'
#001&SYSNDX     MVC       &PM(1), #002&SYSNDX
                MEND
                SAVM      UPT
                KAY       #001,  ¤      001,1
                CALL      SYSPRZ, (#002, ¤ 002, ¤ 001)
* НА R1 ВЫЧИСЛИТЬ U ПО R;
                LOPU
#001            DC        E'1.5'
#002            DC        E'1'
¤ 001           DS        F
¤ 002           DS        F
                SAVN      SYSPRZ
                LYLITI    1
                SISENE    PRIZ1
                PARM      ¤ +8,2,4
                MACR1     (¤ +0)
                TABLIK    (0,1,0)
                CALL      OHM,(¤ +0, ¤ +4, ¤ +8)
                PART      ¤ +4,1,4
                LOPP
                MALU      3
                END
```

Если бы понадобилось, чтобы приведенная для примера программа на языке УТОПИСТ прошла все стадии обработки вплоть до выполнения, то потребовалась бы процедура PRIZD или PRIZCLG.

3.4. ПЛАНИРОВЩИК

Планировщик является программой, которая по вычислительной модели задачи (MZS) и спискам ее входных и выходных переменных (MUIP) ведет поиск доказательства разрешимости задачи. К началу работы планировщика модель задачи приведена уже к такому виду, который обеспечивает быстрый перебор при поиске доказательства. MZS, по существу, — семантическая модель, представленная в виде списковой структуры с двухсторонними ссылками. MZS оформлена как абстрактная структура данных, для работы с которой используются соответствующие функции. Таким образом, основная часть программы планировщика оказывается машинно-независимой и написанной на весьма содержательном уровне в терминах соответствующих функций. Эти функции следующие:

1. Выделение подмножества ALAMH (K, T): K — задаваемое множество в виде списка; T — тип элементов подмножества; ALAMH — подмножество, в которое входят все элементы множества K с типом T.

2. Объединение множеств YHEND (K, L): K, L — задаваемые множества; YHEND — объединение множеств K и L.

3. Разность множеств VAHE (K, L): K, L — задаваемые множества; VAHE — разность множеств K и L.

4. Пересечение множеств YHIS (K, L) : K, L — задаваемые множества; YHIS — пересечение множеств K и L.

5. Проверка включения одного множества в другом множестве HSEES (K, L): K, L — задаваемые множества;

$$\text{HSEES} = \begin{cases} 1, \text{ если } K \subseteq L \\ 0, \text{ если } K \not\subseteq L \end{cases}$$

6. Проверка включения элемента в множестве ESEES (N, K): K — задаваемое множество; N — задаваемый элемент;

$$\text{ESEES} = \begin{cases} 1, \text{ если } N \in K \\ 0, \text{ если } N \notin K \end{cases}$$

7. Проверка пустого множества TYHI (K): K — задаваемое множество;

$$\text{TYHI} = \begin{cases} 0, \text{ если } K = 0 \\ 1, \text{ если } K \neq 0 \end{cases}$$

Работа планировщика происходит следующим образом (рис. 19). Сначала программой TOSTA создаются начальное состояние процесса JW, совпадающее с множеством входных переменных задачи, и множество искомых переменных задачи — MV. Далее начинается собственно планировка.

На первом этапе планировки происходит синтез линейного участка (программа VETKA) путем нахождения транзитивного замыкания для входных переменных задачи (см. гл. 1).

Зная заданные переменные (JW), можно по модели выбирать модули (JSV), входные переменные которых заданы. Применяя такие модули, можно вычислять новые переменные, которые добавляются к множеству заданных переменных. Этот процесс продолжается, пока не будут найдены все искомые переменные или не станет ясно, что дальше ничего нового вычислить нельзя. Если найденное транзитивное замыкание включает искомые переменные (MV), то задача разрешима программой, состоящей из одного линейного участка. Тогда работа планировщика завершается выполнением программы KONST, которая для каждого выбранного модуля, начиная с последнего, устанавливает его необходимость и исключает из доказательства разрешимости лишние шаги (дело в том, что выбирались все модули, которые вычисляли что-то новое, но не все они нужны).

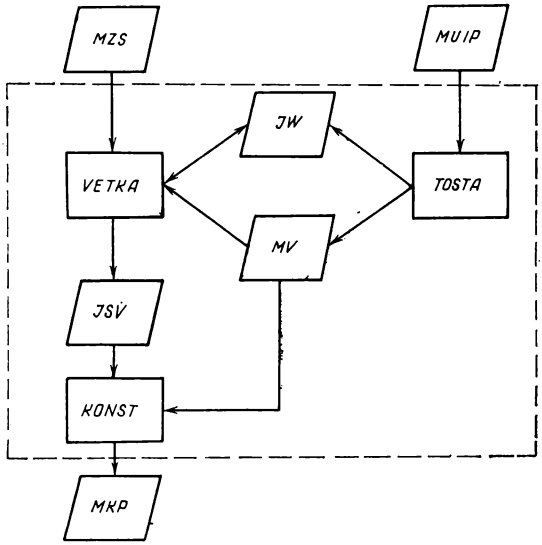


Рис. 19. Общая схема планировщика

К синтезу ветвящейся программы переходят в том случае, если задача неразрешима линейным участком. При этом сразу после составления доказательства для линейного участка выбираются все применимые в достигнутом состоянии условные отношения. В доказательство включают разбор случаев (формула (4) из гл. 1), а в программу — условные операторы. В каждой ветви программы процесс повторяется сначала, т. е. с синтеза линейного участка. Этим процессом управляет головная программа планировщика — PLAN.

В случае подзадач планировщик используется рекурсивно для каждой подзадачи. Если вся задача оказывается неразрешимой, то выдается сообщение о семантической ошибке в операторе задачи.

3.5. ОБРАБОТКА ОПИСАНИЙ ТИПОВ

Понятие типа является одним из важнейших в языках программирования высокого уровня. С каждым объектом связывается один определенный тип, который задается в ее описании. При разработке компилятора информация о типе определяет форму представления объектов и объем машинной памяти, который должен быть от-

веден для них. Таким образом, объекты обрабатываются по-разному в зависимости от типа. Поэтому в большинстве языков имеются только некоторые элементарные типы.

Но было бы желательно, чтобы пользователь смог вводить новые типы в дополнение к типам, уже определенным семантикой самого языка. В некоторых языках можно определить новые структурные типы, определяемые в терминах, ранее определенных составляющих типов. Очевидно, что первичными компонентами структуры должны быть элементарные типы.

Отличительной чертой реализации такой возможности в системе ПРИЗ является то, что в языке УТОПИСТ нет специальной конструкции для описания нового типа — каждый описанный объект можно применять в качестве описателя типа, поскольку тип объекта определен однозначно его вычислительной моделью.

Реализуется это копированием — способом, подобным макротехнике. В том месте, где должен стоять описатель типа, записывается имя ранее определенного объекта (макровызов) и, если необходимо, добавляют переделки (параметры макровызова). При трансляции имя объекта, применяемого в качестве описателя типа, заменяется его описанием. Поскольку возможны переделки, то любой объект может быть использован для задания множества новых типов.

Пусть дано описание

$$y{:}x,$$

где y — имя определяемого объекта;

x — имя ранее определенного объекта.

Для поиска объекта x сначала просматривается модель программы. Если его там нет, то поиск продолжается в семантической памяти. Найденная подмодель

$$x{:}(x_1{:}t_1;\ \ldots;\ x_m{:}t_m;\ a_1;\ \ldots;\ a_n),$$

где x_i — идентификатор;

t_i — описатель типа;

a_j — описание отношения,

представляет собой объект x. Копия $y{:}x$ описывает новый объект — точную копию объекта x. Поэтому и вычислительная модель нового создаваемого объекта является копией уже имеющейся модели. Только имя переменной, соответствующей новому объекту, задается новое. Обозначая через a_j' отношения, полученные из a_j заменой имен $x.x_i$ на $y.x_i$, получим новый объект

$$y{:}(x_1{:}t_1;\ \ldots;\ x_m{:}t_m;\ a'_1;\ \ldots;\ a'_n).$$

Если в составном описателе после ключевого слова КОПИЯ' указано имя ранее определенного объекта, то к копии его вычислительной модели добавляются еще новые компоненты или отношения, заданные в том же описателе.

Пусть на множестве типов (моделей) T определены операторы $P_\alpha{:}T{\to}T$, соответствующие переделкам при копировании.

Согласно семантике языка УТОПИСТ преобразование типа при копировании представимо в виде суперпозиции этих операторов. Имея определение x:тип$_x$, можно по переделкам $\alpha_1, \alpha_2, \ldots, \alpha_r$ получить новый описатель типа

$$p_{\alpha_r}(\ldots(p_{\alpha_j}(\ldots p_{\alpha_1}(\text{mun}_x)\ldots)\ldots).$$

Первая фаза реализации этой конструкции совпадает с реализацией описания объекта, тип которого без изменений определен ранее описанным объектом. По описанию

$$y{:}x\ \alpha_1,\ \alpha_2,\ \ldots,\ \alpha_r$$

в текстовую модель задачи вначале прибавляется новый объект

$$y{:}(x_1{:}t_1;\ \ldots;\ x_m{:}t_m;\ a'_1;\ \ldots;\ a'_n).$$

На второй фазе выполняются действия, требуемые переделками, т. е. операторами преобразования. Семантические подпрограммы, обрабатывающие преобразования, запрограммированы в предположении, что к моменту передачи им управления в модели задачи уже существует подмодель, представляющая собой тип, подвергающийся преобразованию.

Одним из семантических действий является определение вида оператора преобразования. Оператор устанавливают по синтаксису входного текста.

1. Если в переделках задано значение, то преобразование следует интерпретировать как присвоение значения компоненту, указанному либо позиционным способом, либо ключевым. Транслятором конструируется макроопределение для пересылки заданного значения в область памяти, зарезервированную для соответствующего компонента. Поскольку память выделяется генератором, то его адрес является параметром макроопределения. Макроопределение записывается в текстовую модель задачи. К модели добавляют также отношение $a_{\text{задано}}$, представляющее данное макроопределение

$$p_\alpha(\text{mun}_x) = (x_1{:}t_1;\ \ldots;\ x_m{:}t_m;\ a'_1;\ \ldots;\ a'_n;\ a_{\text{задано}};\ \langle\text{макроопр.}\rangle).$$

2. Если в переделках задано новое имя, т. е. объекта с заданным идентификатором в модели программы нет, то придется изменить имя соответствующего компонента. В этом случае оба объекта представлены моделью с одной и той же структурой. Различие состоит только в имени одного компонента.

3. Особой обработки требует переделка, которая используется для доопределения неопределенного объекта. Неопределенный объект — это объект, вычислительная модель которого при описании остается неопределенной и задается в зависимости от конкретной ситуации лишь тогда, когда его используют при решении задачи. Она задается при копировании объекта, компонентом которого является данный неопределенный объект.

Реализуется это во многом аналогично реализации конструкции обычного описания объекта. Пусть, например, в качестве описания типа компонента x_1 задан объект z. Тогда преобразование модели связано с копированием модели $z:(z_1{:}s_1;\ \ldots;\ z_l{:}s_l;\ b_1;\ \ldots;\ b_k)$; и подстановкой этой модели вместо компонента x_1:

$$p_\alpha(\text{mun}_x)=(z_1:s_1;\ \ldots;\ z_l:s_l;\ b'_1;\ \ldots;\ b'_k;\ x_2:t_2;\ \ldots;\ x_m:t_m;\ a_1'\ldots a'_n).$$

Имея неопределенный объект, можно называть по имени его компоненты, которые еще не описаны, т. е. учтена возможность при описании отношений $a_1,\ldots,\ a_n$ использовать объектами $z_1,\ldots,$ z_l. Во время обработки текста эти объекты еще не описаны, а определяются при копировании. Если при создании списковой модели выясняется, что какой-то объект, связанный отношением, не определен, то сообщается об ошибке. Пользователь должен обеспечить определение всех неопределенных объектов, используемых при решении задачи.

4. По отношению к эквивалентным объектам справедливо семантическое требование, согласно которому либо оба объекта имеют одинаковую структуру, либо второй является элементарным объектом типа ПАМЯТЬ'. Два объекта с разной длиной не могут быть объявлены эквивалентными.

При надобности в ходе обработки преобразований типа транслятором конструируются отношения эквивалентности, которые включаются в текстовую модель задачи. Каждое такое отношение связывает два объекта. По этим отношениям эквивалентность объектов модели выясняется во время преобразования модели задачи из текстового вида в списковый. Отношения эквивалентности будут заменены указателями, которые формируются не только между связанными данным отношением объектами, но и между всеми подчиненными им компонентами.

ГЛАВА 4

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ В СИСТЕМЕ ПРИЗ ЕС

4.1. ПРОГРАММИРОВАНИЕ В СИСТЕМЕ ПРИЗ ЕС

Обработка заданий. На языке управления заданиями ОС ЕС программист обращается с запросами на выполнение некоторой работы непосредственно к операционной системе. Задание состоит из одного или нескольких шагов задания.

Применение заданий, которые требуют много управляющих операторов и используются регулярно, можно значительно упростить посредством использования каталогизированных процедур. Каталогизированная процедура — это набор управляющих операторов задания, который находится в библиотеке процедур. Каталогизированная процедура вызывается из библиотеки по имени в операторе EXEC во входном потоке.

Входной поток заданий состоит из предложений языка управления заданиями и, возможно, из исходных данных.

Для работы с системой ПРИЗ созданы свои каталогизированные процедуры, которые помещаются в библиотеку процедур во время генерации системы (см. п. 4.6). Каталогизированная процедура состоит из серии шагов процедуры. Каждый шаг содержит необходимый набор утверждений управления заданием для трансляции исходной программы, генерации модуля решения задачи, ассемблирования, редактирования или выполнения задания.

Ниже приводятся каталогизированные процедуры системы ПРИЗ:

PRIZC — изменение модели предметной области (см. ниже);

PRIZCLG — трансляция исходной программы, написанной на языке УТОПИСТ или на его расширении, генерация модуля решения задачи, создание и исполнение загрузочного модуля в пакетном режиме (см. ниже);

PRIZD — действия те же самые, что при процедуре PRIZCLG, но в режиме диалога;

GENCCLG — генерация модуля решения задачи, создание и исполнение загрузочного модуля, управляющая программа, написанная на языке Ассемблера, вводится с перфокарт, модель задачи — из библиотеки моделей;

GNFCCLG — действия те же самые, что при процедуре GENCCLG, но управляющая программа написана на языке Фортран;

PRIZMODL — создание библиотеки моделей;

PRIZMINI — создание макротеки системы ПРИЗ;

PRIZMACL — работа с макротекой (включает, заменяет и удаляет макроопределения).

Этим процедурам требуются следующие наборы данных:

TEKST — исходная программа на входном языке системы ПРИЗ;

MODLIB — библиотека моделей предметных областей;

MCRLIB — библиотека макрогенератора системы;

ASMOD — программа на языке макроассемблера, соответствующая исходной программе;

MACLIB — библиотека макроопределений ОС;

MACROLIB — библиотека макроопределений системы;

LINMOD — объектный модуль, соответствующий исходной программе;

PRIZLIB — библиотека системы ПРИЗ;

LDLIB — библиотека загрузочных модулей пользователя;

OBLIB — библиотека объектных модулей пользователя;

MAIN — загрузочный модуль, соответствующий исходной программе;

SYSOUT — выходной набор данных, распечатки и сообщения системы;

UPT, MZT, INF, MCR, FORTPR — внутрисистемные наборы данных.

В зависимости от выбранного режима работы системы ПРИЗ используется одна из названных процедур.

Чтобы воспользоваться каталогизированной процедурой во входном потоке, вслед за предложением JOB необходимо поместить предложение EXEC, указывающее на процедуру. При исполнении такого предложения EXEC каталогизированная процедура будет вызвана из библиотеки процедур и будет считаться как бы находящейся во входном потоке.

Утверждения управления заданием идентифицируются начальными литерами //или/* в колонках 1 и 2 перфокарты и могут иметь 3 поля — имя, операция, операнды (табл. 3).

Таблица 3

Формат	Утверждения, которые могут быть использованы в данном формате
// Имя операция операнды	JOB, EXEC, DD
// Операция операнды	EXEC, DD
/*	Разделитель

Перед кодом операции и после него должно располагаться по одному или более пробелу.

При использовании каталогизированной процедуры программист должен подготовить следующие управляющие утверждения задания:

утверждение JOB;

утверждение EXEC, указывающее каталогизированную процедуру с соответствующими параметрами;

утверждение DD, указывающее место, где находится входная информация пользователя, если она имеется;

утверждение DD, которое определяет управляющие предложения редактора связей во входном потоке, необходимые при использовании объектных модулей.

Любое утверждение управления может быть отперфорировано в колонках с 1 по 71. Его можно продолжать на следующей перфокарте. В этом случае в 72-й колонке предыдущей перфокарты должен быть отперфорирован любой допустимый символ, не совпадающий с пробелом, а утверждение, имеющее продолжение, должно заканчиваться запятой, следующей за последним перфорируемым параметром. На карте продолжения должны быть отперфорированы символы // в колонках 1 и 2, а колонки с 3 по 15 — содержат пробелы. Продолжение утверждения перфорируется в колонках с 16 по 71.

Программирование на независимом входном языке. Система ПРИЗ позволяет строить независимые языки в виде расширений языка УТОПИСТ. Несмотря на то что синтаксическая структура языка заранее определена, можно таким образом получать языки очень высокого уровня, удобные для описания задач.

Входной язык УТОПИСТ предназначен для описания сложных структур данных и решаемых на них задач. Благодаря наличию в языке богатого набора выразительных средств язык УТОПИСТ может быть использован как инструмент содержательного анализа предметной области.

Проблемно-ориентированные входные языки пакетов (ПОЯ), создаваемые на базе языка УТОПИСТ, дают специалистам предметных областей, не имеющим программистского опыта, возможность решать свои задачи на пакетах. Тем самым система ПРИЗ расширяет круг пользователей ЭВМ за счет предоставления им возможности описания своих задач на содержательном уровне.

В качестве входного языка можно брать также некоторое подмножество языка УТОПИСТ, расширенное новыми понятиями. Этим можно значительно упростить входной язык, выкинув ненужные конструкции. В частности, для многих пакетов достаточно из процедурной части языка использовать лишь оператор задачи.

Макросредства системы позволяют создать и языки, синтаксически независимые от языка УТОПИСТ.

Независимый входной язык пакета следует строить в том случае, если класс решаемых задач настолько разнообразен, что по условиям каждой задачи приходится строить новую модель задачи, пригодную для решения этой конкретной задачи. Такими, например, являются задачи имитационного моделирования. Построение входного языка начинается с содержательного анализа рассматриваемой прикладной области, класса решаемых задач и создания на основе этого анализа семантической модели предметной области.

Если создана модель предметной области, то уместно хранить ее в библиотеке моделей, чтобы не повторять те же описания в

каждом задании. Эту модель или ее компоненты можно использовать для описания объектов из этой же предметной области.

Имя, используемое в программе, считается именем библиотечного понятия, если оно не определено в описании данных этой программы. Имена библиотечных понятий используются в описании копии.

Поясним сказанное на примере. Предположим, что создана семантическая модель геометрии ГЕОМ, которая находится в библиотеке моделей. В модель включены такие понятия, как круг, квадрат, треугольник и т. д. Если теперь, программируя на независимом входном языке системы ПРИЗ, нам хочется сказать, что рассматриваемый объект К является кругом, то в описании данных придется писать

```
ПУСТЬ' К: ГЕОМ.КРУГ;
```

Имена можно сократить, используя директиву ПО':

```
ПО' ГЕОМ;
ПУСТЬ' К:КРУГ;
```

По этому описанию создают модель, содержащую подробное описание объекта К — описания всех его компонентов и отношений между ними

```
К: (РАДИУС, ДИАМЕТР, ПЛОЩАДЬ, ПЕРИМЕТР: ВЕЩ';
   МОДУЛЬ' КРУГМ      3        СЛ' РАДИУС, ДИАМЕТР,
                                   ПЛОЩАДЬ, ПЕРИМЕТР);
```

Посмотрим пример использования одноименных объектов

```
      ПРОГ' ОДНОИМЕННЫЕ;
*ОПРЕДЕЛЯЕМ ОБЪЕКТ КРУГ
      ПУСТЬ' КРУГ: (РАДИУС, ПЛОЩАДЬ: ВЕЩ';
                  МОДУЛЬ' КР СЛ' РАДИУС, ПЛОЩАДЬ;
                  ЗАДАНО' РАДИУС = 5);
*ОБЪЕКТ КРУГ1 ЯВЛЯЕТСЯ КОПИЕЙ БИБЛИОТЕЧНОГО ПОНЯТИЯ
*КРУГ
            КРУГ1:ГЕОМ.КРУГ РАДИУС=5;
      ДЕЙСТВИЯ'
*ЗАДАЧА С ИСПОЛЬЗОВАНИЕМ ОБЪЕКТА КРУГ, ОПИСАННОГО В
*ДАННОЙ ПРОГРАММЕ
      НА' КРУГ ВЫЧИСЛИТЬ' ПЛОЩАДЬ;
*ЗАДАЧА С ИСПОЛЬЗОВАНИЕМ ОБЪЕКТА КРУГ1,ЯВЛЯЮЩЕГОСЯ
*КОПИЕЙ ПОНЯТИЯ КРУГ ИЗ БИБЛИОТЕКИ МОДЕЛЕЙ
      НА' КРУГ1 ВЫЧИСЛИТЬ' ПЛОЩАДЬ, ДИАМЕТР, ПЕРИМЕТР;
*ЗАДАЧА С ИСПОЛЬЗОВАНИЕМ ПОНЯТИЯ КРУГ ИЗ БИБЛИОТЕКИ
*МОДЕЛЕЙ

      ГЕОМ.КРУГ.РАДИУС = 10;
      НА' ГЕОМ.КРУГ ВЫЧИСЛИТЬ' ПЛОЩАДЬ ПО' РАДИУС;
      КОНЕЦ' +++
```

По первому оператору задачи нельзя вычислить, например, диаметр, так как он не определен в объекте КРУГ. Третий оператор задачи вообще не использует описания данных. Моделью задачи является подмодель из библиотеки моделей.

Напишем программу для вычисления площади круга, который вписан в квадрат (рис. 20, *а*). Диагональ квадрата равняется 5 см. В этом случае при описании данных необходимо указать, что имеются два объекта — круг и квадрат, причем сторона квадрата равняется диаметру круга, и, кроме того, еще известно значение диагонали. В операторе задачи придется определить, что именно надо вычислить, и на какой фигуре:

```
ПРОГРАММА' ПРИМЕР1;
        ПО' ГЕОМ;
        ПУСТЬ' К1: КВАДРАТ ДИАГОНАЛЬ = 5;
                К2: КРУГ ДИАМЕТР = К1.СТОРОНА;
        ДЕЙСТВИЯ'
        НА' ПРИМЕР1 ВЫЧИСЛИТЬ' К2.ПЛОЩАДЬ;
КОНЕЦ' +++
```

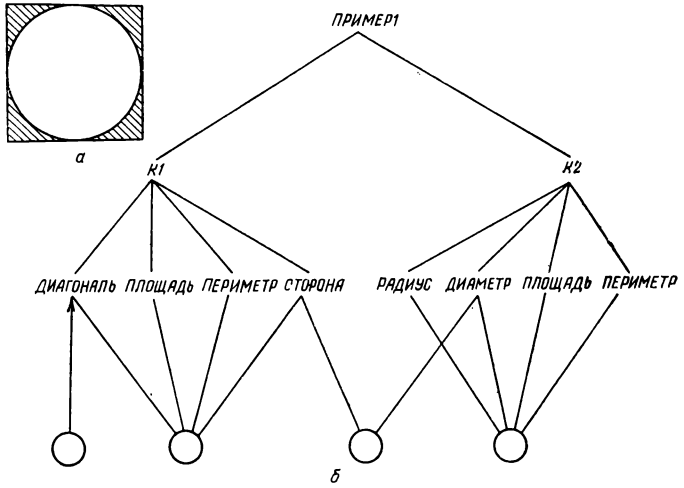


Рис. 20. Графическое представление фигуры

После трансляции модель, соответствующая данной программе, имеет вид, показанный на рис. 20, *б*. По этой модели и оператору задачи генератором синтезируется модуль решения задачи, при выполнении которого диагонали квадрата присваивается значение, равное 5, затем вычисляется сторона квадрата; так как сторона квадрата равняется диаметру круга, то по этому же значению вычисляется площадь круга.

В вышеприведенном примере исходное значение задали отношением в семантической модели, но его можно определить и оператором присваивания в действиях. В этом случае программа выглядит следующим образом:

```
ПРОГРАММА' ПРИМЕР2;
        ПО' ГЕОМ;
        ПУСТЬ' К1:КВАДРАТ;
                К2:КРУГ ДИАМЕТР = К1.СТОРОНА;
        ДЕЙСТВИЯ'
```

```
            К1.ДИАГОНАЛЬ : = 5;
    НА' ПРИМЕР2 ВЫЧИСЛИТЬ' К2.ПЛОЩАДЬ ПО' К1.ДИАГОНАЛЬ;
КОНЕЦ' +++
```

Важно заметить, что в данном случае в операторе задачи необходимо указать имя объекта, значение которого задано. Только тогда это значение станет доступным модулю решения задачи.

Посмотрим еще другой пример. Допустим, что надо вычислить площадь заштрихованной фигуры. Для этого в описании данных определим искомый объект К3, значение которого равняется разности площади квадрата и круга:

```
ПРОГРАММА' ПРИМЕР3;
   ПО' ГЕОМ;
   ПУСТЬ'     К1:КВАДРАТ ДИАГОНАЛЬ = 5;
              К2:КРУГ ДИАМЕТР = К1.СТОРОНА;
              К3:ВЕЩ';
              ПРИСВ' К3 : =К1.ПЛОЩАДЬ — К2.ПЛОЩАДЬ;
   ДЕЙСТВИЯ'
              НА'ПРИМЕР3 ВЫЧИСЛИТЬ' К3;
КОНЕЦ' +++
```

Можно было бы не добавлять объект К3, а необходимые вычисления выполнять оператором присваивания, после решения соответствующей задачи:

```
ДЕЙСТВИЯ'
    НА' ПРИМЕР3 ВЫЧИСЛИТЬ' К1.ПЛОЩАДЬ, К2.ПЛОЩАДЬ;
    ПЛ: = К1.ПЛОЩАДЬ — К2.ПЛОЩАДЬ;
```

Более сложные примеры уже описаны в литературе. Например, применение системы ПРИЗ при проектировании силовых полупроводниковых диодов описано в [4].

Задача, заданная программой, написанной на языке УТОПИСТ, решается последовательным использованием программ системы ПРИЗ, начиная с трансляции и кончая выполнением создаваемого загрузочного модуля. Для этого предназначены процедуры PRIZCLG и PRIZD (см. ниже).

Схема процедуры PRIZCLG показана на рис. 21, демонстрирующем системные программы и наборы данных (см. выше). Транслятор TRN переводит текст с независимого входного языка на внутренний язык системы. На внутреннем языке исходной программе соответствуют модель MZT, которую можно записать и в библиотеку моделей MODLIB, и программа на языке макроассемблера. Если описание действий обозначено ключевым словом ДЕЙСТВИЯ, то программе присваивается имя UPT. Генератор GEN составляет модуль решения задачи и оформляет модуль на языке макроассемблера ASMOD, который соответствует исходной программе на языке УТОПИСТ. Затем этот модуль транслируется, редактируется и выполняется.

Предложения во входном потоке для использования процедуры PRIZCLG следующие:

```
// название задания        JOB MSGLEVEL=(2,0)
// название шага           EXEC PRIZCLG,
```

```
//          LIBD    = имя диска, где находятся библиотеки
                      пользователя,

//        MODLIB    = имя библиотеки моделей,
//        MCRLIB    = имя библиотеки макрогенератора,
//        LDLIB     = имя библиотеки загрузочных модулей,
//        OBLIB     = имя библиотеки объектных модулей
//TRN. SYSIN DD *
                    предложения на входном языке
/*
//LKED. SYSIN DD *
  LIBRARY OBJECT    — имена объектных модулей, реализующих отношения
                      модели
/*
//GO.SYSIN DD *
                    входная информация рабочей программы
/*
//
```

Оператор JOB инициирует задание. Оператор EXEC указывает имя процедуры, которая должна быть выполнена, и значения ее параметров. Если некоторый параметр опущен, то его значение принимается из следующего набора: LIBD=PRZLIB, MODLIB= =PRIZ.MODLIB, MCRLIB=NULLFILE, LDLIB=PRIZ.LINKLIB, OBLIB=NULLFILE.

Из операторов DD первый является обязательным и определяет место исходной программы во входном потоке, остальные включаются при необходимости.

Программист может указать режимы работы транслятора и генератора в поле PARM предложения EXEC при вызове процедуры. Указания заключаются в апострофы, разделяются запятыми, могут следовать в любом порядке

```
PARM. TRN = '[MACRO] [,MZT] [,UPT]',
PARM. GEN = '[PRINT] [,TRACE] [,RECU]',
```

Эти указания означают следующее:

MACRO — исходная программа содержит макровызовы, которые необходимо при трансляции заменять требуемыми макрорасширениями;

MZT — выводится распечатка семантической модели, соответствующей той части исходной программы, где приведены описания данных;

UPT — выводится результат трансляции описаний действий (на языке Ассемблера);

PRINT — печатаются значения входных и выходных параметров всех задач, а также обозначения на входном языке;

TRACE — отладочный параметр, при этом печатаются значения входных и выходных параметров всех автоматически вызываемых модулей и их обозначения;

RECU — в модуле задачи содержатся описания рекурсивных модулей.

Если в операторе EXEC режимы не указаны, то по умолчанию действуют режимы, установленные при генерации системы, т. е.

исходная программа не должна содержать макровызовы, и на печатающее устройство выводится только распечатка исходной программы вместе с сообщениями системы. Выводятся конструкции языка, в которых имеются ошибки, и сообщения об ошибках.

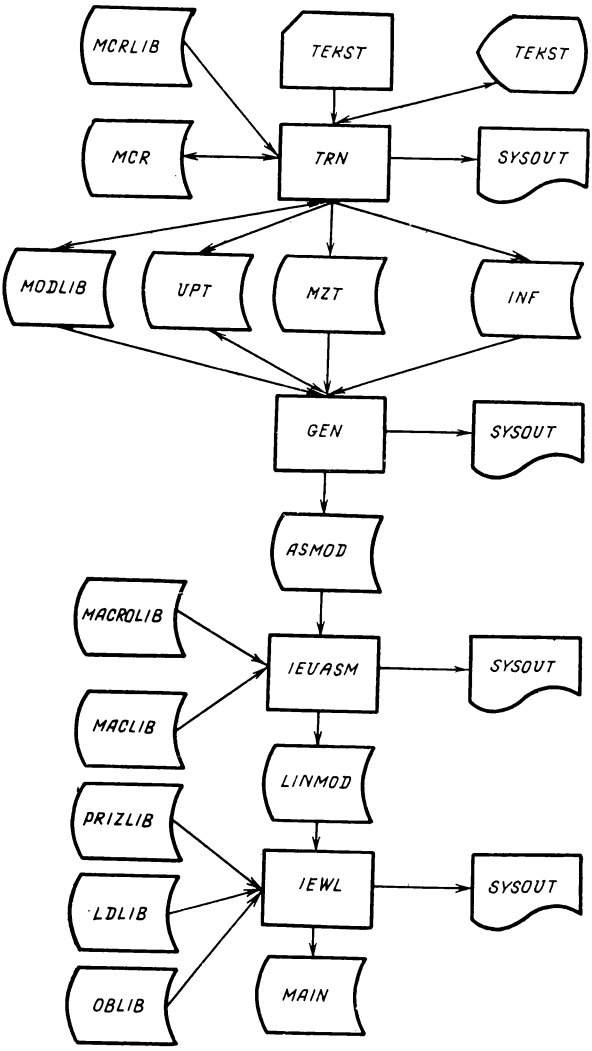


Рис. 21. Схема процедуры PRIZCLG

Программирование на вложенном входном языке. Система ПРИЗ позволяет строить языки пакетов программ, вложенные в базовый язык программирования, которым является Фортран или язык Ассемблера. Один и тот же язык пакета может использоваться с обоими базовыми языками.

Вложенный входной язык пакета следует разрабатывать в том случае, если предметная область может быть описана единой вычислительной моделью, на которой решаются все задачи пакета. При этом модель предметной области может меняться во времени независимо от задач. Такие классы задач встречаются при проектировании крупных агрегатов, технологических процессов, энергоустановок и т. д. В этом случае задачи относятся к одному крупному объекту, который имеет относительно устойчивую структуру (модель), но параметры объекта меняются.

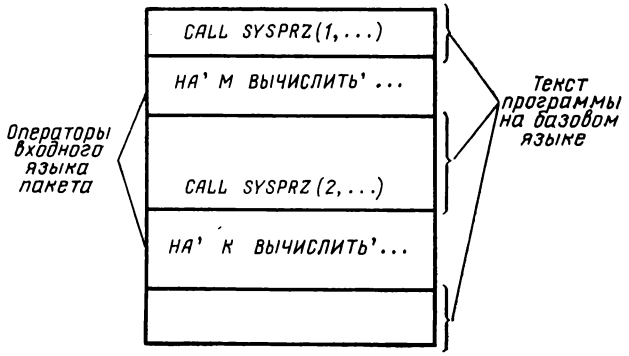


Рис. 22. Структура программы с использованием вложенного входного языка

Структура программы с использованием вложенного языка пакета показана на рис. 22. Текст программы на базовом языке содержит операторы входного языка пакета (операторы задачи). Перед каждым оператором входного языка пакета записано обращение к системе ПРИЗ (см. ниже).

Оператор вложенного входного языка вместе с вызовом процедуры SYSPRZ в тексте программы на базовом языке определяет следующие действия:

выборку значений из программы на базовом языке и присваивание этих значений некоторым величинам предметной области;

вычисление значений новых величин предметной области;

передачу значений величин предметной области в программу на базовом языке и присваивание этих значений некоторым величинам в этой программе.

При построении вложенного входного языка следует стремиться к тому, чтобы все вычисления, относящиеся к предметной области пакета, можно было выполнять на модели предметной области. Для этого модель предметной области должна содержать:

все величины из предметной области, которые могут встретиться при решении задач в данном пакете;

все отношения, наблюдаемые между величинами предметной области. Последние представляются в виде отдельно программи-

руемых модулей или в виде уравнений или присваиваний, задаваемых на языке УТОПИСТ.

На базовом языке описывается общий ход решения задачи. Если в пакете не предусмотрены специальные средства ввода-вывода, то на базовом языке описывается ввод-вывод для задач, решаемых в пакете. Через базовый язык осуществляется также связь с модулями (подпрограммами), не включенными в пакет, и связь с другими пакетами.

Для трансляции, редактирования и выполнения программ, написанных на расширенном языке Ассемблера, предназначена процедура GENCCLG.

Предложения во входном потоке для использования процедуры следующие:

```
// название задания JOB MSGLEVEL = (2,0)
// название шага EXEC GENCCLG,
//              PARM.GEN = 'CARD, ASM',
//              LIBD = имя диска, где находятся библиотеки пользователя,
//              MODLIB = имя библиотеки моделей,
//              LDLIB = имя библиотеки загрузочных модулей,
//              OBLIB = имя библиотеки объектных модулей
//GEN.SYSIN DD *
/*                    текст программы
//LKED.SYSIN DD *
  LIBRARY OBJECT              /имена объектных модулей, реализующих отношения модели/
/*
//GO.SYSIN DD *
                      входная информация для рабочей программы
/*
//
```

Если некоторый параметр опущен, то по умолчанию его значение принимается таким, как и при процедуре PRIZCLG (см. выше).

Программист может указать варианты работы генератора при вызове процедуры PARM.GEN='CARD, ASM[,PRINT][,TRACE][,RECU]'

Схема процедуры GENCCLG показана на рис. 23. Для этой процедуры транслятор TRN не используется. В качестве модели задачи применяется некоторая семантическая модель из библиотеки моделей предметных областей (MODLIB), а в качестве головной программы (UPT) — модуль, введенный с перфокарт. В результате работы системы составляется программа на языке макроассемблера (ASMOD), которая транслируется (LINMOD), редактируется (MAIN) и выполняется.

Аналогичная процедура (рис. 24) составлена для случая использования расширенного языка Фортран. Предложения во входном потоке для использования процедуры следующие:

```
// название задания JOB MSGLEVEL = (2,0)
// название шага EXEC GNFCCLG,
//              PARM.GEN = 'CARD,FORT'
//UPTR.SYSIN DD *
// ADD LIST = ALL
```

текст исходной программы

```
·/ ENDUP
/ /LKED.SYSIN DD *
  INCLUDE ASSEM
/*
/ /
```

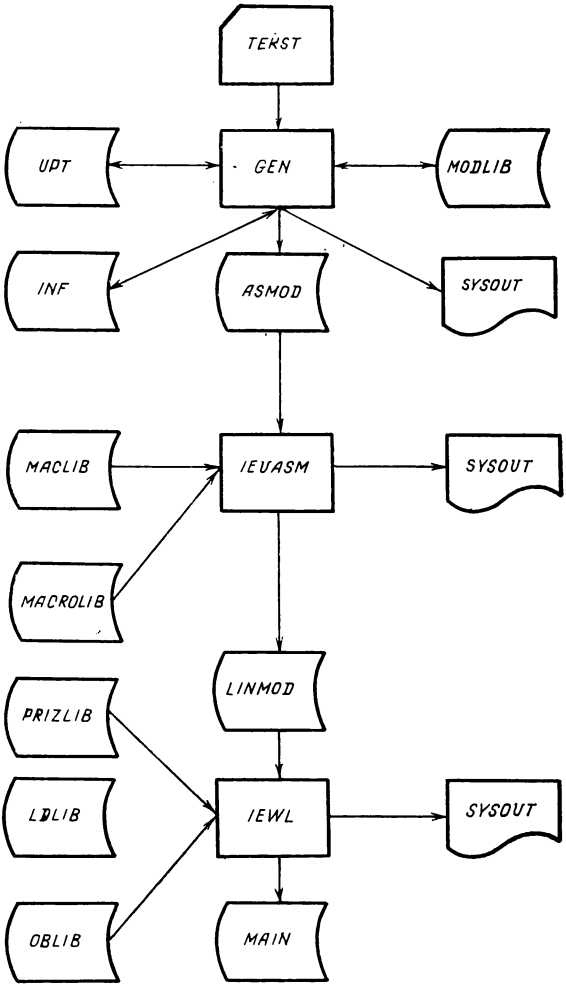


Рис. 23. Схема процедуры GENCCLG

Обе процедуры можно использовать и в том случае, когда исходная программа находится в некоторой библиотеке пользователя. Для этого придется изменить значение параметра в поле PARM — слово CARD заменяется словом DISC, и добавить оператор DD, определяющий местонахождение исходной программы.

Для использования процедуры GENCCLG следует пользоваться следующим входным потоком:

```
/ / название задания JOB MSGLEVEL = (2,0)
/ / название шага EXEC GENCCLG,
/ /              PARM. GEN= 'DISC, ASM'
/ /GEN.UPTLIB DD DSN = ULIB (NAME), DISP=SHR,
/ /                          UNIT=SYSDA, VOL=SER=UPTD
/*
/ /
```

где

ULIB — имя библиотеки, где находится исходная программа;
NAME — имя раздела, содержащего данную программу;
UPTD — имя диска, где находится библиотека.

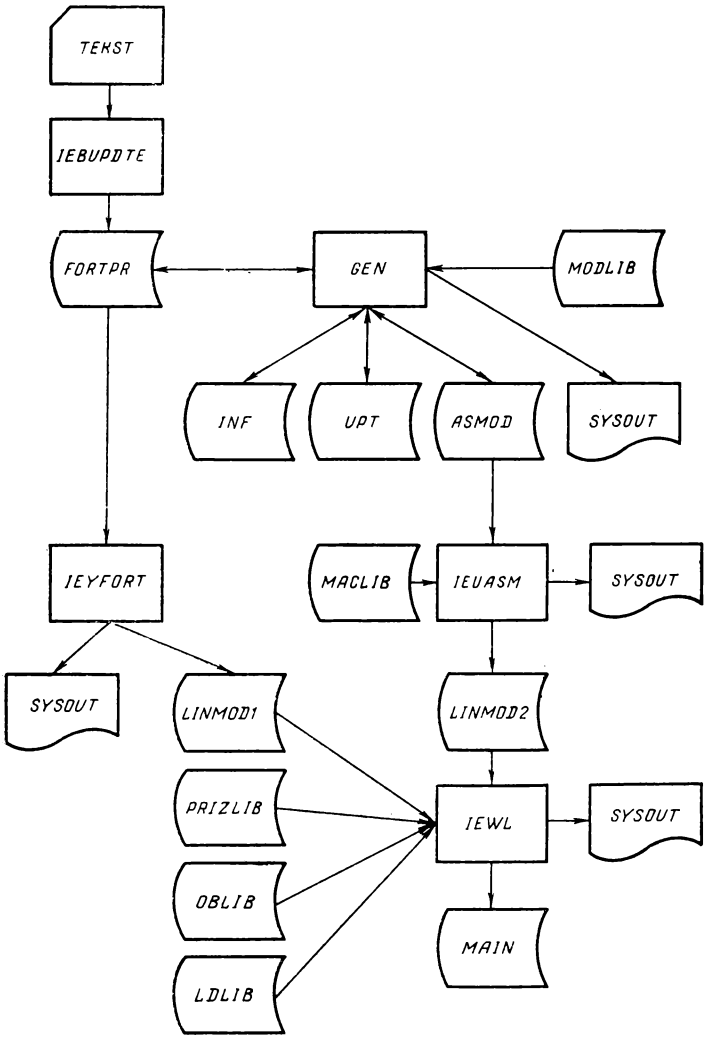


Рис. 24. Схема процедуры GNFCCLG

В случае использования процедуры GNFCCLG предложения во входном потоке выглядят несколько по-другому:

```
/ / название задания JOB MSGLEVEL = (2,0)
/ / название шага EXEC GNFCCLG,
/ /                PARM. GEN = 'DISC, FORT'
/ /UPTR.SYSUT1                DD DSN = ULIB, UNIT = SYSDA,
/ /                           DISP=SHR, VOL=SER=UPTD
/ /LKED. SYSIN                DD *
·/ REPRO                      NAME=NIMI, NEW=PS
·/ ENDUP
/*
/ /UPTR. SYSIN1               DD*
        INCLUDE ASSEM
/*
/ /
```

Работа с семантической памятью. Для работы с семантической памятью в пакетном режиме предназначена процедура PRIZC (рис. 25). Для работы в диалоговом режиме можно использовать процедуру PRIZD. При разработке нового проблемно-ориентированного языка пакета программ выбранные понятия описываются на языке УТОПИСТ или на его расширении. Эти описания (TEKST) преобразуются транслятором TRN в новые семантические модели MZT и по директиве INSERT записываются в библиотеку моделей предметных областей (MODLIB). При этом можно использовать и макробиблиотеку MCRLIB системы.

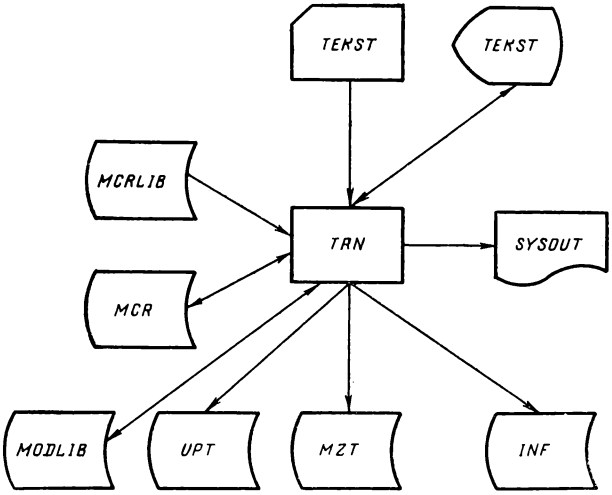


Рис. 25. Схема процедуры PRIZC

Выполнением процедуры PRIZC можно не только расширять модель предметной области, но и изменять состояние библиотеки моделей. Директивой DELETE можно исключить из библиотеки целую модель или удалить из нее только объект с данным именем (см. п. 2.4).

Предложения во входном потоке для использования процедуры следующие:

```
// название задания JOB MSGLEVEL = (2,0)
// название шага EXEC PRIZC
//              LIBD = имя диска, где находятся библиотеки пользователя,
//              MODLIB = имя библиотеки моделей;
//              MCRLIB = имя библиотеки макрогенератора,
//TRN.SYSIN           DD *
                  текст программы на входном языке
/*
//
```

Пользователь может указать с помощью параметров поля предложения EXEC, нужно ли ему работать с макрогенератором (MACRO), получить распечатку модели задачи (MZT) или головной программы (UPT).

Диалоговый режим* системы ПРИЗ предназначен для интерактивной работы пользователя при описании и решении задач из прикладных областей. Данный режим позволяет: вводить и корректировать входной текст задачи с помощью дисплея, следить и управлять за ходом решения задач, многократно дополнять и исправлять семантическую модель задачи как при трансляции, так и во время управления диалоговыми подпрограммами.

Для работы в диалоговом режиме применяется управляющая процедура PRIZD. Предложения во входном потоке для использования процедуры PRIZD следующие:

```
// название задания JOB MSGLEVEL = (2,0)
// название шага EXEC  PRIZD
//         MODLIB     = имя библиотеки моделей,
//         MCRLIB     = имя библиотеки макрогенератора,
//         LDLIB      = имя библиотеки загрузочных модулей,
//         LDLIB2     = имя библиотеки загрузочных модулей,
//         CRT        =номер устройства дисплея,
//         R          =размер раздела.
//
```

Если некоторый параметр опущен, то по умолчанию его значение принимается таким, как и при процедуре PRIZCLG (см. выше). Программист может при вызове процедуры указать режимы работы системы в поле PARM предложения EXEC.

Параметры заключаются в апострофы и группируются по программным компонентам, которыми они управляют (1 группа — указания для транслятора; 2 — указания для генератора; 3 — указания для Ассемблера; 4 — указания для загрузчика). В одной группе они разделяются запятыми и могут следовать в любом порядке. Группы разделяются знаками «;»:

```
PARM='[COPY,][MACRO,][MZT,][UPT];[PRINT,][TRACE,][RECU];
   [параметры для ассемблера]  ;  [параметры для загрузчика]'
```

---

* Система ПРИЗ работает равноправно как с английской, так и с русской лексикой. Но по традиции ЕС ЭВМ все утилиты системы ПРИЗ работают только с английской лексикой.

Если некоторая группа указаний отсутствует, то эта группа заменяется знаком «;». Если отсутствует некоторая группа указаний с конца, то список указаний можно заканчивать прямо символом «,», опуская точки с запятой.

Например,

```
/ /        PARM = ' ; ; ; NCAL'
/ /        PARM = 'MACRO; PRINT,TRACE; LET,LIST'
```

Вышеприведенные указания означают следующее:

COPY — распечатка протокола диалога;
MACRO — работа с макрогенератором;
MZT — распечатка модели задачи;
UPT — распечатка головной программы;
PRINT — распечатка обозначений и значений входных и выходных параметров всех задач;
TRACE — распечатка входных и выходных параметров всех автоматически вызываемых модулей;
RECU — в модуле задачи содержатся описания рекурсивных модулей.

Если в операторе EXEC режимы не указаны, то по умолчанию действуют режимы, установленные при генерации системы, т. е. ввод исходной программы и вывод сообщений системы только через экран.

Если пользователь хочет использовать директивы EXEC' для выполнения своих подпрограмм (из своей библиотеки загрузочных модулей), то в поток управляющих карт задания добавляются следующие перфокарты:

```
/ / EXEC   PRIZD ,...
/ /STEPDLIB    DD
/ /            DD DSN  = <имя библиотеки пользователя> , ...
                         .
                         .
                         .
```

Все модули пакета, вызываемые автоматически системой ПРИЗ, должны быть помещены в библиотеку загрузочных модулей. Карта с описанием этой библиотеки включается в поток после управляющих карт STEPLIB:

```
/ /LKDLIB    DD
/ /          DD
/ /          DD DSN  = <имя библиотеки пользователя> , ...
```

Карты остальных наборов данных, используемых программами пользователя, описываются после управляющих карт LKDLIB.

Процедура PRIZD управляет последовательным выполнением следующих фаз работы системы ПРИЗ: диалог для описания и корректировки задач, компиляция, пуск программы (рис. 26).

I фаза работы для ввода и корректировки текста задачи начинается сообщением системы «ПРИЗ К ВАШИМ УСЛУГАМ». В ответ на это сообщение надо ввести предложение начала программы ЗДЧ' идентификатор; , после чего можно приступить к опи-

санию задачи. Теперь можно использовать при вводе с экрана описаний предложение начала описаний (ПУСТЬ) или начала действий (ДЕЙСТВИЯ) при вводе действий (см. дополнительно п. 2.1).

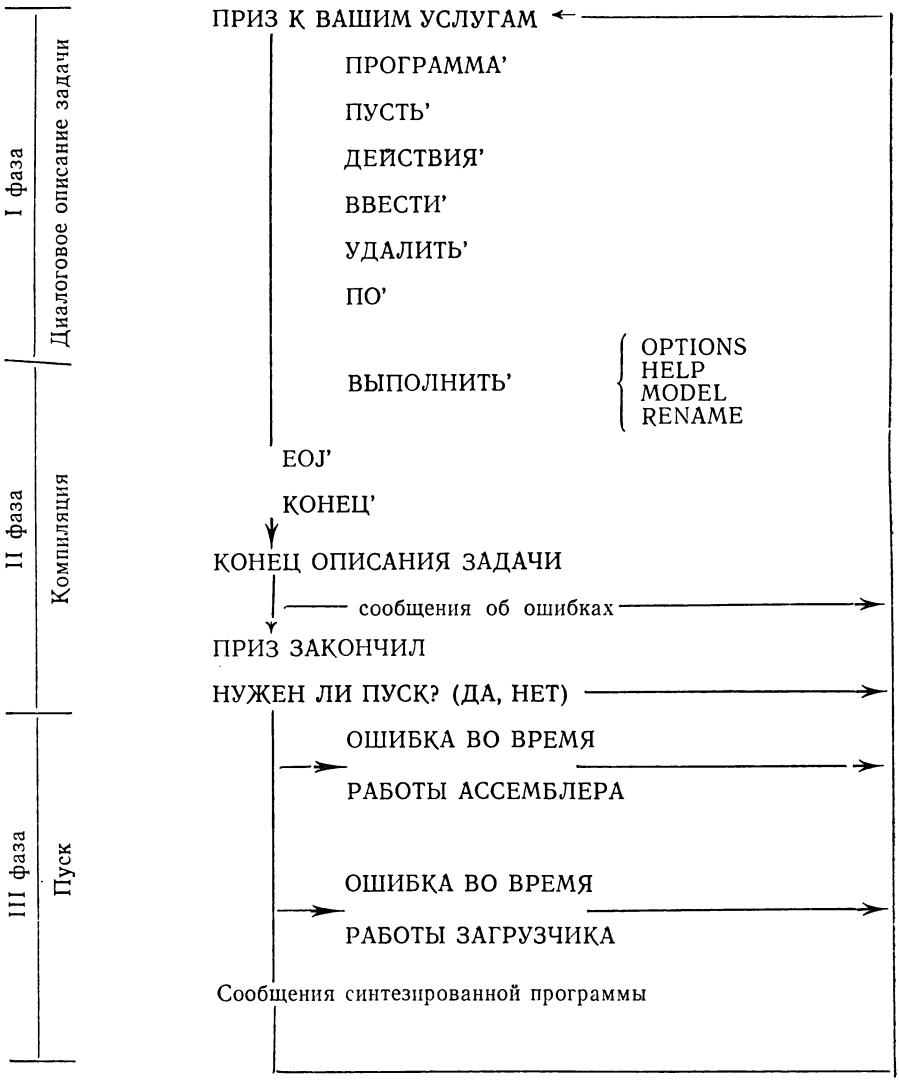


Рис. 26. Диалоговый режим

Основные директивы при диалоговой работе на этой фазе следующие:

ВВЕСТИ, УДАЛИТЬ, ПО, ВЫПОЛНИТЬ, EOJ,

Директива ВВЕСТИ предназначена для изменения состояния библиотеки моделей путем включения в нее новых моделей и их компонент. По директиве

ВВЕСТИ' имя;

в библиотеку записывается новый объект, имя которого задано после слова ВВЕСТИ. Если это имя составное, то именем объекта в архиве становится последний его компонент.

По директиве

ВВЕСТИ' имя В' имя-модели;

в библиотеку моделей в состав существующей модели (имя модели указано после слова В) добавляется новый объект, имя которого задано после слова ВВЕСТИ.

Пример

```
ВВЕСТИ' СТАТИСТИКА;
ВВЕСТИ' ДИСПЕРСИЯ В' СТАТИСТИКА;
```

Директива УДАЛИТЬ' в отличие от директивы ВВЕСТИ' предназначена для исключения из библиотеки моделей.

Пример

```
УДАЛИТЬ' СТАТИСТИКА;
УДАЛИТЬ' ГЕОМЕТРИЯ.КРУГ;
```

Директива ПО предназначена для сокращения составных имен библиотечных понятий, применяемых при описании данных и действий.

Пример

Вместо описаний

```
ПУСТЬ'X: СУБД.ПРОЕКЦИЯ...
      Y: СУБД.СОЕДИНЕНИЕ...
```

можно использовать текст

```
ПО' СУБД;
 ПУСТЬ' X:ПРОЕКЦИЯ...
       Y: СОЕДИНЕНИЕ ...
```

Директива ВЫПОЛНИТЬ предназначена для непосредственного выполнения модулей из библиотеки загрузочных модулей PRIZLIB или из прикрепленной к ней библиотеки пользователя (см. выше предложения STEPLIB процедуры PRIZD).

Система ПРИЗ сама предлагает к использованию следующие системные модули: OPTIONS, HELP, MODEL, RENAME.

1. Директивой ВЫПОЛНИТЬ' OPTIONS; +++ выполняется программа OPTIONS, которая дает возможность изменить режимы работы системы ПРИЗ. Напомним, что начальные режимы работы системы указываются при вызове процедуры PRIZD.

В начале работы программа OPTIONS выдает на экран сообщение

```
SPECIFY PARAMETERS (MZT, UPT, COPY, MACRO)
   PARAMETER = (YES,NO) ,...)
```

В ответ на это сообщение нужно ввести желаемые новые режимы работы.

Пример

COPY = YES, MACRO = YES, MZT = NO

После ввода этих указаний программа OPTIONS кончает работу, и можно продолжать диалог для ввода текста задачи.

2. Директивой ВЫПОЛНИТЬ' HELP; +++ выполняется программа HELP, которая дает возможность просмотреть содержание библиотеки моделей, содержание модели задачи или содержание макробиблиотек.

В начале работы программа HELP выдает на экран дисплея сообщение

YOU CAN PRINT : ARCHIVES, MODEL, MACROLIB OR END

А. Если в ответ на это сообщение вводится ARCHIVES, то программа сообщает дальше

```
ARCHIVES CONTAIN
имя 1   имя 2   имя 3
YOU       CAN PRINT:
INTO      NAME
NEXT
UP
SHOW      NAME. NAME. ... NAME
WHAT      IS      NAME. NAME. ... NAME
HELP
END
```

Здесь имя 1, имя 2, ... являются именами моделей (понятий) из библиотеки. Предлагаемые в этом сообщении возможности дальнейшей работы означают:

INTO имя-понятия — программа выбирает данное понятие и выводит его компоненты. Имя данного понятия должно быть заранее выведено программой. Если данного понятия нет, то выводится сообщение

NAME NOT FOUND;

NEXT — программа выводит компоненты следующего понятия на том же иерархическом уровне. Если на этом уровне понятий больше нет, выводится сообщение

DOES NOT EXIST;

UP — программа выводит компоненты высшего уровня. Если данное сообщение было дано на самом высшем уровне, выводится сообщение

DOES NOT EXIST;

SHOW имя-понятия — программа выбирает данное понятие и выводит его компоненты. В отличие от команды INTO здесь «имя-понятия» может стоять как составное имя. Если данного понятия нет, то выводится сообщение

NAME NOT FOUND;

WHAT IS имя-понятия — программа выбирает данное понятие и выводит его значение;

HELP — программа выводит

```
YOU CAN PRINT:
INTO NAME
NEXT
UP
SHOW NAME. NAME. . . .NAME
WHAT IS NAME.NAME. . . . NAME
HELP
END
```

упоминая возможности работы;

END — программа выводит заново сообщение

YOU CAN PRINT: ARCHIVES, MODEL, MACROLIB OR END.

Б. Если в ответ на это сообщение вводится MODEL, то программа выдает следующее сообщение:

```
MODEL NAME IS
имя-модели-задачи
YOU CAN PRINT:
INTO NAME
NEXT
UP
SHOW NAME. NAME. . . . NAME
WHAT IS NAME. NAME. . . . NAME
HELP
END
```

предлагая аналогичные возможности работы, как и при работе с библиотекой моделей.

В. Если в ответ на это сообщение вводится MACROLIB, то программа выдает сообщение

```
YOU CAN PRINT:
MACRO
&MACRO
MACRO NAME
&MACRO NAME
END
```

предлагая пять разных режимов работы.

MACRO — программа выдает на экран дисплея каталог макробиблиотеки

```
MACRO CONTAIN
  имя-макро 1, имя-макро 2, . . . ;
```

&MACRO — программа выдает на экран каталог временной макробиблиотеки;

MACRO имя-макро — программа выводит из макробиблиотеки на экран макроопределение, имя которого задано после слова MACRO;

&MACRO имя-макро — программа выводит из временной макробиблиотеки на экран макроопределение, имя которого задано после слова &MACRO (если текст макроопределения не помещается на экран дисплея, то для выдачи продолжения текста требуется повторное нажатие клавиши ввода);

END — конец работы с макробиблиотеками.

Выводятся сообщения:

END OF MACROLIB
YOU CAN PRINT: ARCHIVES,MODEL,MACROLIB OR END.

Г. Если в ответ на это сообщение вводится END, то программа HELP кончает работу и выводит END OF HELP.

Теперь можно продолжать работу для ввода текста задачи.

3. Директивой ВЫПОЛНИТЬ' MODEL; +++ выполняется программа MODEL, которая дает возможность познакомиться с содержанием библиотеки моделей или с содержанием модели задачи в полном объеме. В отличие от программы HELP программой MODEL выводится внутреннее представление модели, содержащее, кроме имен компонент, описания всех отношений и некоторую системную информацию.

В начале работы программа MODEL выводит на экран дисплея сообщение

YOU CAN PRINT: MODEL,ARCHIVES OR END.

А. Если в ответ на это сообщение вводится MODEL, то программа выдает на экран модель задачи.

Б. Если вводится ARCHIVES, то программа выдает новое сообщение

MODEL NAME IS

На это сообщение надо ответить именем модели из библиотеки моделей.

В. Если вводится END, то программа заканчивает работу и система готова заново обрабатывать текст задачи в УТОПИСТе.

4. Директивой ВЫПОЛНИТЬ' RENAME; +++ выполняется программа RENAME, которая дает возможность переименовать компоненты любого уровня модели задачи или модели из библиотеки моделей.

Предупреждение. В описаниях отношений параметры (компоненты моделей) не переименовываются.

В начале работы программа RENAME выводит сообщение

YOU CAN PRINT: MODEL OR ARCHIVES

А. Если в ответ на это сообщение вводится MODEL, то программа готова к переименованию в модели задачи.

Б. Если вводится ARCHIVES, то программа готова к переименованию в библиотеке моделей.

В обоих случаях программа выдает следующий вопрос:

THE OLD NAME IS

В ответ на данный вопрос надо ввести полное (составное) имя компонента, которое будет изменяться. Если программа не найдет такого компонента, то выдаются следующие сообщения:

NAME NOT FOUND
END OF RENAME

и программа заканчивает работу.

После успешного поиска имени следует сообщение

THE NEW NAME IS

Теперь надо ввести новый идентификатор, который заменяет последний компонент составного имени.

Программа заканчивает работу сообщением

END OF RENAME

после чего система ПРИЗ готова заново обрабатывать текст задачи на языке УТОПИСТ.

Директива EOJ предназначена для окончания сеанса работы с системой ПРИЗ ЕС. В ответ на эту директиву система выводит сообщение END OF JOB и заканчивает работу.

Первая фаза работы системы ПРИЗ ЕС заканчивается вводом директивы КОНЕЦ, после чего система выдает сообщение КОНЕЦ ОПИСАНИЯ ЗАДАЧИ и переходит ко второй фазе работы — компиляции.

II фаза работы — это проверка разрешимости введенной задачи и создание исходного модуля. На этой фазе работы выводятся сообщения о корректной или некорректной постановке задачи. При корректной постановке задачи, которая ведет к синтезу алгоритма решения задачи, система выдает сообщение об успешном завершении второй фазы работы ПРИЗ ЗАКОНЧИЛ и переходит к выполнению третьей фазы работы системы.

При некорректной постановке задачи выводятся сообщения об ошибках и передается управление к первой фазе работы. Затем на экране заново появляется текст ПРИЗ К ВАШИМ УСЛУГАМ. После этого можно дополнять предыдущую модель задачи, начиная с описания со словом ПУСТЬ', можно описывать новую модель, вводя ПРОГРАММА' <идентификатор>, или можно описывать новую задачу на предыдущей модели, начиная со слова ДЕЙСТВИЯ'.

Третья фаза работы (пуск) начинается сообщением системы НУЖЕН ЛИ ПУСК? (ДА, НЕТ) (см. рис. 26). Отрицательный ответ на это сообщение — НЕТ — вызывает переход к первой фазе работы, позволяя после сообщения ПРИЗ К ВАШИМ УСЛУГАМ продолжать работы со слов ПУСТЬ, ПРОГРАММА или ДЕЙСТВИЯ. Положительным ответом ДА передается управление программным компонентам операционной системы — ассемблеру и загрузчику. Если работа ассемблера прерывается сбоем, то на экран выдается сообщение ОШИБКА ВО ВРЕМЯ РАБОТЫ АССЕМБЛЕРА, и возвращаются к первой фазе работы системы. Если работа загрузчика прерывается сбоем (или возникают ошибки пользователя при редактировании связей и выполнении программы), то выдается сообщение ОШИБКА ВО ВРЕМЯ РАБОТЫ ЗАГРУЗЧИКА, и возвращаются к первой фазе работы системы.

Если третья фаза работы завершается нормально, то после выполнения составленной программы выводится сообщение ПРИЗ К ВАШИМ УСЛУГАМ, после чего можно продолжать работу на первой фазе работы системы.

Сеанс работы с системой ПРИЗ ЕС заканчивается при вводе на первой фазе работы системы директивы EOJ'; +++.

4.2. ПРОГРАММИРОВАНИЕ МОДУЛЕЙ

Все программы в системе программирования ПРИЗ оформляются в виде модулей. Модули могут быть подпрограммами, макроопределениями или головными программами. Программирование модулей, их трансляция и отладка выполняются с помощью программирования на Фортране или на языке Ассемблера.

В табл. 4 дана сводка всех вариантов оформления и применения модулей. Допустимые в системе ПРИЗ варианты отмечены в таблице знаком «+», недопустимые варианты — знаком «—».

Таблица 4

Варианты оформления и применения модулей

Оформление		Применение					
		вызывающая программа		явно вызываемый модуль		автоматически вызываемый модуль	
вид модуля	язык программирования	содержит вызовы модулей	содержит описания задач	содержит вызовы модулей	содержит описания задач	содержит вызовы модулей	содержит описания задач
Головная программа	Язык Ассемблера	+	+	—	—	—	—
	Фортран	+	+	—	—	—	—
Подпрограмма	Язык Ассемблера	+	+	+	—	+	+
	Фортран	+	+	+	—	+	+
	ПЛ/1	+	—	+	—	—	—
Макроопределение	Язык Ассемблера	+	+	+	—	+	+

Далее приводятся дополнительные сведения, необходимые при использовании модулей в системе ПРИЗ.

Автоматически вызываемые модули. Особенностью системы ПРИЗ является то, что в ней возможен автоматический вызов модуля без явной ссылки на модуль в тексте программы на исходном языке.

Автоматически вызываемые модули описывают отношения в семантических моделях. Каждый такой модуль может описывать несколько отношений, т. е. может иметь несколько вхождений в семантическую модель.

Для каждого вхождения модуля устанавливается соответствие между параметрами и связанными переменными отношения. Это соответствие устанавливается позиционно — по порядку перечисления параметров в заголовке подпрограммы (или макроопределения) и по порядку связанных переменных в описании отношения.

Пример

Имеются тексты модулей

```
SUBROUTINE          P (A,B,C)
.
.
.
END
SUBROUTINE          Q (S,H)
.
.
.
END
```

и текст семантической модели

```
ПУСТЬ'
        M: (X, Y, K, H : ВЕЩ'; ...
        МОДУЛЬ' P ВХ' X, Y ВЫХ' K;
        МОДУЛЬ' Q ВЫХ' H ВХ' K; ...);
```

Эти тексты определяют одно вхождение модуля P в модель M и одно вхождение модуля Q в модель M. При вхождении модуля P устанавливаются следующие соответствия между переменными модели и параметрами модуля:

```
X ... A
Y ... B
K ... C
```

При вхождении модуля Q устанавливается следующее соответствие:

```
H ... S
K ... H
```

Через модуль решения задачи, составленный по модели M, может произойти автоматический вызов модулей P и Q, который на языке Ассемблера будет записан

```
CALL  P, (X, Y, K), VL
CALL  Q, (H, K), VL
```

Автоматически вызываемый модуль может быть программой на языке Ассемблера или на языке Фортран, а также макроопределением. Обработка подпрограмм или макроопределений, которые являются автоматически вызываемыми модулями, ничем не отличается от обработки подпрограмм или макроопределений в соответствующих системах программирования ОС ЕС. При программировании автоматически вызываемых подпрограмм необходимо учитывать то, что фактические параметры таким подпрограммам передаются из модуля решения задачи по наименованию в полном соответствии с соглашениями о связях.

Автоматически вызываемая подпрограмма должна быть доступной редактору связей.

Макроопределения в качестве автоматически вызываемых модулей должны быть доступны Ассемблеру во время генерации рабочей программы. При программировании автоматически вызываемых макроопределений необходимо учитывать то, что модуль решения задачи, который содержит соответствующие макрокоманды, использует общие регистры 7—12, содержимое которых не должно изменяться.

Описание задачи в модуле. Модуль, записанный на языке Фортран или на языке Ассемблера, может содержать описание задачи, для решения которой необходим автоматический синтез алгоритма.

Описание задачи в головной программе состоит:

из вызова процедуры SYSPRZ, записанного в том месте программы, куда должна вставляться программа решения задачи;

из оператора задачи, записанного непосредственно после вызова процедуры SYSPRZ.

Описание задачи в автоматически вызываемом модуле состоит только из вызова процедуры SYSPRZ, записанного в том месте программы, куда должна быть вставлена программа решения задачи. Оператор задачи составляется в этом случае автоматически по модели задачи.

Модуль может содержать несколько описаний задач. Каждая задача имеет номер в виде целого числа, отличный от номеров других задач в данном модуле.

В головной программе номера задач задаются программистом в виде констант с фиксированной точкой (от 1 до 16). В автоматически вызываемых модулях номера задач определяются системой автоматически, но не более трех в одном модуле.

Первым параметром процедуры является номер задачи. Если программа написана на Фортране, то номер задачи должен быть явно записан в виде константы. Следующими параметрами являются собственные переменные вызывающего модуля. Последовательность этих параметров должна соответствовать последовательности переменных в описании задач.

Пример

```
CALL SYSPRZ (1,Y1,Y2, X1)
```

Если задача описывается в автоматически вызываемом модуле, то все параметры процедуры SYSPRZ (в том числе и номер задачи) должны быть формальными параметрами модуля.

Оператор задачи для каждой задачи в головной программе записывается в виде комментария непосредственно за вызовом процедуры SYSPRZ. Форма оператора задачи следующая:

* НА имя-модели ВЫЧИСЛИТЬ список-переменных-1 ПО список-переменных-2;

имя-модели является именем модели, на которой должен синтезироваться алгоритм решения задачи;

список-переменных-1 является списком тех компонент модели,

которые являются выходными параметрами задачи, их значения передаются в модуль в качестве результатов решения задачи;

список-переменных-2 является списком тех компонент названной модели, которые являются входными переменными задачи, их значения передаются из модуля в качестве исходных данных для решения задачи.

Пример задачи в головной программе

```
          CALL SYSPRZ,(N,Y1,Y2,X1),VL
*НА ГЕОМ.КРУГ ВЫЧИСЛИТЬ ПЛОЩАДЬ,ДИАМЕТР ПО РАДИУС;
          .
          .
          .
N         DC F'1'
```

Содержание этой задачи может быть выражено последовательностью операторов

```
РАДИУС :      = X1;
ПЛОЩАДЬ :     = f1 (РАДИУС);
ДИАМЕТР  :    = f2 (РАДИУС);
 Y1      :    = ПЛОЩАДЬ;
 Y2      :    = ДИАМЕТР;
```

где функции f_1, f_2 определены моделью ГЕОМ.КРУГ.

Пример задачи в автоматически вызываемом модуле

```
SUBROUTINE     PROGR  (X,Y,N,Y1,Y2,X1)
.
.
.
CALL SYSPRZ    (N,A,B,C)
.
.
.
.
END
```

Содержание задачи в данном случае будет определено только после того, как модуль PROGR будет использован для описания отношения в некоторой модели.

Использование слабосвязанных переменных. Модуль, соответствующий отношению, которое выражает связь между значениями слабосвязанных переменных, и используемый для вычисления значения любой из них по заданным значениям других переменных, состоит из ряда ветвей. Каждый раз выполняется одна из ветвей в зависимости от того, какие формальные параметры заданы в качестве входных и выходных. Это касается отношения типа ПРОГ.

При конкретном выполнении данного модуля системной информацией определяется, какие формальные параметры модуля являются входными и какие — выходными. Системная информация об использовании параметров доступна модулю типа подпрограмма через служебную программу OUTPUT.

Программа OUTPUT оформлена в виде подпрограммы FUNCTION с одним формальным параметром. Логическая функ-

ция OUTPUT(N) определяет, является ли формальный параметр с порядковым номером N в модуле выходным. Если N-й формальный параметр является выходным, то логическая функция принимает значение «истина» (общий регистр 0 содержит 1), в противном случае — «ложь» (общий регистр 0 содержит 0).

Пример

Программа с параметрами X1, X2, X3, вычисляющая значение любого из параметров по заданным значениям остальных двух, например, может иметь вид

```
      SUBROUTINE   PROG(X1,X2,X3)
      LOGICAL OUTPUT
      IF (OUTPUT (1)) GOTO1
      IF (OUTPUT (2)) GOTO2
      X3 = X1 + X2
      RETURN
    1 X1 = —X2 + X3
      RETURN
    2 X2 =—X1 + X3
      RETURN
      END
```

Уравнения. Отношение может быть задано уравнением

$$f(X_1, X_2, \ldots, X_k) = 0,$$

где $X_1, X_2, \ldots, X_k$ — вещественные переменные.

Такое отношение используется для вычисления значения переменной X_i $(i=1, 2, \ldots, k)$ при заданных значениях переменных $X_1, X_2, \ldots, X_{i-1}, X_{i+1}, \ldots, X_k$*.

Отношению, заданному уравнением, соответствует модуль, вычисляющий выражение $f(X_1, X_2, \ldots, X_k)$. При выходе из модуля вычисленное значение должно оставаться в нулевом регистре с плавающей точкой. Для этого, программируя модуль на языке Фортран, надо его оформить в виде подпрограммы FUNCTION.

Во время решения задачи система ПРИЗ находит ту переменную, значение которой необходимо вычислить, и определяет ее значение методом хорд.

Пример

Допустим, что отношение определяет следующую зависимость:

$$X + Y + Z = 0$$

Модуль, соответствующий данному отношению, может быть записан следующим образом:

а) на языке Фортран

```
      FUNCTION      F(X,Y,Z)
      F = X + Y + Z
      RETURN
      END
```

б) на языке Ассемблера

```
      START
      BALR    12,0
```

* Так как уравнения решаются методом хорд, не всякое уравнение бывает разрешимым.

```
USING  *, 12
L      3, 0 (0,1)
LE     0, 0 (0,3)
L      3, 4 (0,1)
AE     0, 0 (0,3)
L      3, 8 (0,1)
AE     0, 0 (0,3)
BR     14
END
```

в) макроопределением

```
MACRO
MACROF  & X, & Y, & Z
LE      0, & X
AE      0, & Y
AE      0, & Z
MEND
```

4.3. МАКРОСИСТЕМА

Назначение и возможности макросредств. Макросредства ПРИЗ ЕС предназначены для обеспечения синтаксической независимости входных языков пакетов программ, создаваемых в системе программирования ПРИЗ, от языка УТОПИСТ. Последний является языком с расширяемой семантикой и весьма простой синтаксической структурой. Расширения языка УТОПИСТ наследуют его синтаксическую форму. Применяя макросистему, пользователь получает возможность создавать языки, синтаксически независимые от языка УТОПИСТ, приблизив их, например, к естественному языку конкретной прикладной области.

Макросистему ПРИЗ ЕС можно использовать также для реализации некоторого имеющегося языка в конкретной системе программирования ПРИЗ ЕС, как это в качестве примера сделано с языком ELEQ (см. приложение 6).

Кроме того, макросредства просто позволяют заменить длинный повторяющийся входной текст компактным текстом или именем.

Макропроцессор строит результирующий текст на языке УТОПИСТ из входного текста, состоящего из последовательности текста на языке УТОПИСТ и из макровызовов, используя макроопределения из библиотеки макроопределений или макроопределения, заданные непосредственно во входном тексте. Результирующий текст получается из входного текста путем замены каждого находящегося в нем макровызова макрорасширением, т. е. текстом, полученным из соответствующего макровызову макроопределения.

Прежде чем использовать макровызов, необходимо составить соответствующее данному макровызову макроопределение. Составленное макроопределение можно записать в библиотеку макроопределений с помощью процедуры обслуживания библиотеки или задать во входном тексте до того, как в нем встретится соответствующий макровызов. Если макроопределение включено непосредственно во входной текст, то оно имеет приоритет над макроопре-

делением с тем же именем, находящимся в библиотеке, т. е. при генерации макрорасширения используется макроопределение, которое включено во входной текст.

Для наибольшего быстродействия макропроцессор используется как препроцессор и включен в состав транслятора системы программирования ПРИЗ ЕС.

Общая схема использования макросистемы показана на рис. 27.

Структура макроопределения включает заголовок, описывающий макровызов, тело, задающее макрорасширение, и операторы MACRO, MBEGIN и MEND:

<макроопределение> : : =MACRO<заголовок>MBEGIN
<тело>MEND

Начальный оператор MACRO является первым в макроопределении и указывает только начало макроопределения.

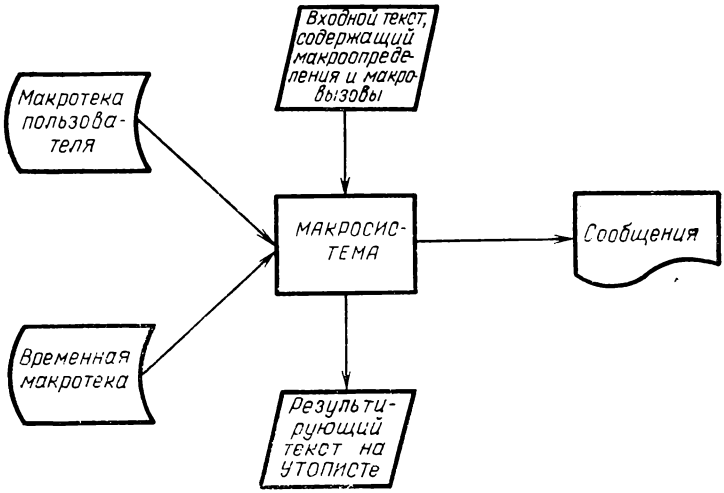


Рис. 27. Схема использования макросистемы

Заголовок является прототипом макровызова, он определяет имя макроопределения, его формальные параметры и входной текст, который считается частью макровызова. Данный текст должен удовлетворять лексическим правилам языка УТОПИСТ.

Заголовок макроопределения имеет следующую структуру:

<заголовок> : : =
[{ <текст> / <формальный параметр> }]<имя макроопределения>[{ <текст> / <формальный параметр> } ...]
<имя макроопределения> : : = &<идентификатор>&
<формальный параметр> : : = &<идентификатор> <разделитель>
<разделитель> : : = {,|:|;|=|+|—|*|/|(|)|.|␣|'}
<идентификатор> : : = буква [{ буква / цифра } ...]
<текст> : : = последовательность символов

Имя макроопределения используется для установления связи между макровызовом и макроопределением, в этом смысле имя должно быть уникальным.

Формальный параметр состоит из символа амперсенда &, за которым следует последовательность из букв и цифр, начинающаяся с буквы, т. е. идентификатора. По разделителю, записанному после идентификатора, макропроцессор определяет конец фактического параметра в макровызове (см. ниже). Примеры записи параметров:

```
&A,
&B1+
&ПАРАМЕТР:
```

Оператор MBEGIN указывает начало тела макроопределения.

Тело макроопределения содержит текст на языке УТОПИСТ или на его расширении и параметры, заданные в заголовке. Запись параметра в теле макроопределения идентична записи его в заголовке (8 первых символов).

Конечный оператор MEND является всегда последним в макроопределении.

Для создания уникальных имен в тексте при многократном использовании макровызовов используется системный параметр &SGEN, который в момент начала анализа входного текста задачи имеет значение 0000. Перед началом расширения каждого макровызова его значение увеличивается на единицу и остается таким на все время расширения данного макровызова, вместо &SGEN подставляется четырехзначное число (нули не опускаются). Уникальное имя записывается соединением символов с &SGEN. Например, в макроопределении MAC&SGEN:МНОЖ ЗАПИСЬ=СТУДЕНТ;

системный параметр &SGEN заменяется при генерации значением 0004 в том случае, если макровызов является четвертым по порядку:

```
MAC0004:МНОЖ ЗАПИСЬ=СТУДЕНТ;
```

Запись макроопределения:
в первой строке записывается оператор MACRO;
начиная со второй строки, записывается заголовок;
с новой строки — оператор MBEGIN;
тело макроопределения записывается с новой строки;
в последней строке записывается — оператор MEND.
Замечания:
MBEGIN не может быть первым словом в заголовке;
имя макроопределения должно содержаться в первой строке заголовка.

Ограничение: тело макроопределения не должно содержать макровызовы.

Примеры макроопределений.

Пример 1

```
    MACRO
(1) &P1 ЭТО &ПОДМНОЖЕСТВО& ПОЛУЧАЕМОЕ ИЗ МНОЖЕСТВА &P2
    С ИСПОЛЬЗОВАНИЕМ УСЛОВИЯ &P3;
      MBEGIN
    УСЛ&SGEN: ЛОГ';
(2) ПРИСВ' УСЛ&SGEN : = ( &P3);
    С &SGEN:ПОДМНОЖ МНОЖ = &P2,ЕСТЬ = &P1,УСЛОВ = УСЛ&SGEN;
      MEND
```

Это макроопределение реализует семантику понятия подмножества.

(1) — заголовок макроопределения, где

&ПОДМНОЖЕСТВО& — имя макроопределения;

и формальные параметры:

&P1 — имя результирующего множества;

&P2 — имя исходного множества;

&P3 — условие выбора элементов (могут быть составные объекты) для подмножеств.

(2) — тело макроопределения содержит текст на языке УТОПИСТ с заданными в заголовке макроопределения параметрами. Имена УСЛ&SGEN и С&SGEN заменяются при каждом использовании макровызова уникальными именами (УСЛ0001, С0001, УСЛ0002, С0002, ...).

Пример 2

```
      MACRO
&P1: &ПИ& * &P2;
      MBEGIN
&P1:=3.1415926535*&P2;
      MEND
```

Это макроопределение заменяет имя параметра ПИ с его значением.

Макровызовы порождают в результирующем тексте некоторый фрагмент текста на языке УТОПИСТ или на его расширении — макрорасширение. Формирование макрорасширения, соответствующего макровызову, описывается макроопределением.

Макровызов содержит имя, может содержать фактические параметры и участки произвольного текста

<макровызов> : : =
[{ <текст> | <фактический параметр> }...] <имя макровызова> [{ <текст> | <фактический параметр> }]

<имя макровызова> : : = <идентификатор>

<фактический параметр> : : = <текст 1> <разделитель>

<текст> — не должен содержать разделитель, который в данном макровызове стоит после <текст 1>.

Пример

Макровызов: X—ЭТО ПЕРЕМЕННАЯ ТИПА ВЕЩ,

где ПЕРЕМЕННАЯ — это имя макро, X, ВЕЩ — фактические параметры.

Данный макровызов при подходящем макроопределении порождает следующий текст на языке УТОПИСТ:

```
X:ВЕЩ';
```

Структура макровызова описывается в заголовке соответствующего макроопределения. Имя макровызова используется для нахождения соответствующего макроопределения. Именем макровызова должно быть имя макроопределения. Соответствие устанавливается по первым восьми символам имени.

По способу установления соответствия фактических и формальных параметров макровызов является позиционным. Соответствие устанавливается по месту, которое параметр занимает в макровызове. При этом ни один параметр не может быть опущен.

Примеры макровызовов:

1. Использование макровызова «ПОДМНОЖЕСТВО»

```
ОТЛИЧНИКИ ЭТО ПОДМНОЖЕСТВО, ПОЛУЧАЕМОЕ ИЗ МНОЖЕСТВА
  СТУДЕНТЫ С ИСПОЛЬЗОВАНИЕМ УСЛОВИЯ СТУДЕНТ. УСПЕВАЕ-
  МОСТЬ БР'5;
```

В этом примере

```
ПОДМНОЖЕСТВО — имя макровызова,
ОТЛИЧНИКИ                          )
СТУДЕНТЫ                           } фактические параметры,
СТУДЕНТ.УСПЕВАЕМОСТЬ БР'5          )
ЭТО,,ПОЛУЧАЕМОЕ ИЗ МНОЖЕСТВА,     )
С ИСПОЛЬЗОВАНИЕМ УСЛОВИЯ, ;       } тексты.
```

Этот макровызов заменяется следующим текстом на языке УТОПИСТ:

```
УСЛ0001:ЛОГ';
ПРИСВ'УСЛ0001:=(СТУДЕНТЫ. УСПЕВАЕМОСТЬ БР'5);
С0001:ПОДМНОЖ МНОЖ=СТУДЕНТЫ, ЕСТЬ=ОТЛИЧНИКИ,
УСЛОВ=УСЛ0001;
```

2. Использование макровызова «ПИ»

```
РЕЗ: = ПИ * (4Д — 3Н);
```

В этом примере

```
ПИ — имя макровызова,
РЕЗ                 )фактические параметры,
(4Д РЕЗ             )
: =(4Д — 3Н)сты
```

Этот макровызов заменяется следующим текстом на языке УТОПИСТ:

```
РЕЗ : = 3.1415926535 * (4Д — 3Н);
```

Библиотека макроопределений и процедура обслуживания библиотеки. Для использования одних и тех же макровызовов в разных заданиях целесообразно заранее записывать макроопределения в библиотеку макроопределений.

Библиотека макроопределений (PRMAKTEK) представляет собой файл с прямым доступом, хранящийся на диске. Библиотека состоит из двух частей: в первой части содержится таблица имен всех имеющихся в библиотеке макроопределений с указанием их

расположения в файле. Во второй части библиотеки хранятся макроопределения в несколько преобразованном виде: вместо параметров в теле макроопределения подставляется % <порядковый номер>, где порядковый номер — это номер, соответствующий порядку расположения параметра в заголовке макроопределения.

Все работы по ведению библиотеки осуществляются процедурой обслуживания библиотеки PRIZMACL. С помощью процедуры можно включать новые макроопределения, заменять макроопределения и удалять ненужные макроопределения.

Возможны следующие режимы обслуживания макробиблиотеки (в процедуре указывается соответствующий параметр):

1) без параметров или параметр ADD — добавляет макроопределения в существующую библиотеку макроопределений;
2) параметр BEGIN — обновление библиотеки: записывает макроопределения с начала файла, старая билиотека не сохраняется;
3) параметр REPLACE — замена макроопределений: на перфокарте задаются имена макроопределений, которые подлежат замене, каждое на отдельной перфокарте, в конце списка имен помещается перфокарта ·/END. После этого идут тексты заменяющих макроопределений;
4) параметр DELETE — удаление макроопределений из библиотеки: на перфокартах задается список имен удаляемых макроопределений аналогично параметру REPLACE.

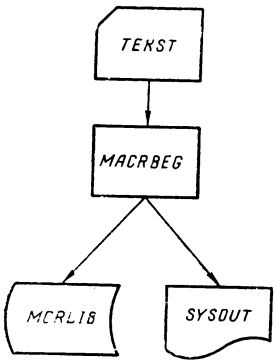


Рис. 28. Схема процедуры PRIZMACL

Схема процедуры PRIZMACL показана на рис. 28.

Предложения во входном потоке для использования процедуры следующие:

```
// название задания     JOB   MSGLEVEL = (2,0)
// название шага EXEC     PRIZMACL
//          MCRLIB — имя библиотеки макроопределений.
// MKR.SYSIN DD *
   [имена макроопределений, которые надо заменить или удалить]
    [текст макроопределений]
/*
//
```

С помощью параметров поля PARM предложения EXEC можно указать режим работы программы обслуживания

PARM.MKR = '[ADD] [,REPLACE] [,DELETE] [,BEGIN]'

В ходе работы процедуры обслуживания выдаются тексты макроопределений, а также сообщения об ошибках, если они имеются.

При успешном завершении записи макроопределения в библиотеку выдается следующий текст:

МАКРО <имя> ВВЕДЕНО В БИБЛИОТЕКУ МАКРООПРЕДЕЛЕНИЙ

Для создания библиотеки макроопределений пользователя предназначена процедура PRIZMINI. Для использования процедуры необходимы следующие управляющие карты:

```
// название задания        JOB    MSGLEVEL = (2,0)
// название шага EXEC          PRIZMINI,
//        MCRLIB  — имя библиотеки макроопределений,
//        LIBD    — имя диска, где находится библиотека.
```

4.4. ВВОД-ВЫВОД

В языке УТОПИСТ отсутствуют специальные операторы ввода-вывода. Обмен информацией между носителями данных и основной памятью обеспечивается расширениями языка, задаваемыми дополнительными объектами.

В этих целях система предлагает пользователям объект ввода ВВОД, объект вывода ВЫВОД, работу с шаблонами и автоматическую генерацию формата. Кроме того, для каждого пакета необходимо разработать свой язык ввода-вывода, отвечающий характерным требованиям предметной области пакета. Этот язык будет расширением языка УТОПИСТ или УТОПИСТ-макро. Конечно, при разработке соответствующих языков должны быть учтены особенности диалоговых и пакетных режимов пользования системой.

Объект ВВОД предназначен для ввода значений простых и составных объектов в диалоговом режиме работы с системой. Данный объект является макроопределением и его использование предусмотрено вместе с макросистемой ПРИЗа.

Форма применения объекта ВВОД в тексте задачи на языке УТОПИСТ-макро следующая:

ВВОД имя-объекта;

Примеры

```
ВВОД ГЕОМ.ТРЕУГОЛЬНИК.СТОРОНА1;
ВВОД ГЕОМ.ТРЕУГОЛ2;
```

По описанию задачи, где встречался объект ВВОД, система генерирует часть программы, которая при выполнении выдает на экран дисплея сообщение по следующей форме:

имя-объекта?

Например, в результате вышеприведенных примеров мог появиться следующий текст диалога:

```
1. вопрос: ГЕОМ.ТРЕУГОЛЬНИК.СТОРОНА?
   ответ:  64.17
2. вопрос: ГЕОМ.ТРЕУГОЛ2?
   ответ: 30.0,40.0, 50.0
```

Здесь следует обратить внимание на то, что значения компонентов составных объектов вводятся в строгой последовательности все вместе, отдельные значения разделяются запятыми. Значения вводятся так же, как ввод на языке УТОПИСТ.

Пример

667, 66.7, 'БЕЛОЕ' , ИСТИНА'

Если при вводе данных встретилась ошибка, то вопрос повторяется.

Объект ВЫВОД предназначен для вывода значений простых и составных объектов в диалоговом режиме работы. Объект ВЫВОД является макроопределением, как и объект ВВОД, и предусматривает использование вместе с макросистемой ПРИЗа.

Форма применения объекта ВЫВОД в тексте задачи следующая:

ВЫВОД имя — объекта;

Примеры

ВЫВОД ВАЛ233
ВЫВОД ГЕОМ.КРУГ.ДИАМЕТР;

По описанию задачи, где встречается объект ВЫВОД, система генерирует часть программы, которая при выполнении выдает на экран дисплея следующее сообщение:

имя-объекта : значение

Например, вышеприведенные примеры использования объекта ВЫВОД могли бы вызвать на экран дисплея следующий текст:

```
ВАЛ233.МАТЕРИАЛ     : 40ХМ
ВАЛ233.ДЛИНА        : 405.5
ВАЛ233.ДИАМЕТР      : 21.75
ГЕОМ.КРУГ.ДИАМЕТР   : 4.12
```

Аналогичный текст распечатывается и на АЦПУ.

Работа с шаблонами. Данный вариант ввода-вывода предназначен только для диалогового режима работы и служит для ввода-вывода значений составных объектов.

Операции ввода-вывода выполняются здесь в следующей последовательности.

а. При работе в диалоговом режиме на языке УТОПИСТ вводится директива ВЫПОЛНИТЬ' MENU; +++, которая вызывает выполнение программы MENU. Программа MENU предназначена для создания, удаления, вывода на экран, исправления и модификации шаблонов обмена через экран. В начале выполнения эта программа выдает на первую строку следующее сообщение:

OPTIONS : (ADD,DELETE,DISPLAY,REPLACE,MODIFY,END)

необходимо выбрать один из перечисленных операторов и ввести с именем шаблона, после этого

в строках со 2-й по 11-ю можно описать поля шаблона по следующим правилам:

поле типа ЦЕЛЫЙ' описывается символами «9». Длина поля определяется количеством символов «9», а конец поля — символом «'», например 9999';

поле типа ВЕЩЕСТВЕННЫЙ' описывается как и поле типа ЦЕЛЫЙ', используя дополнительно символ «.» для указания десятичной точки, например 999.99';

поле типа СТРОКА' описывается символами «X», например XXXX'.

Пример

```
ADD                ДЕТАЛИ
НАИМЕНОВАНИЕ       XXXXXXXXXXX'
ДЛИНА              999.999'
ВЕС                99.999'
```

б. На языке УТОПИСТ значения объектов вводятся с использованием понятия ВВОДИТЬ:

ид: ВВОДИТЬ имя — объекта;

Пример

ВВОД1: ВВОДИТЬ ДЕТАЛЬ1 ;

Во время выполнения программы на экран выводится сообщение TABLE NAME?

На это сообщение нужно ответить (на ту же строку) именем созданного заранее шаблона. После этого на экран выдается образец шаблона. Пользователь заполняет поля соответствующими значениями и вводит эти данные с экрана.

Пример

```
НАИМЕНОВАНИЕ  ВАЛ № 132
       ДЛИНА    152,50
       ВЕС      00.512
```

в. На языке УТОПИСТ значения объектов выводятся с использованием понятия ВЫВОДИТЬ:

ид: ВЫВОДИТЬ имя — объекта [,РЕЗ = имя — логического];

Пример

ВЫВОД1: ВЫВОДИТЬ ДЕТАЛЬ2;

Чтобы по данному предложению выполнялся вывод, надо организовать вычисление компонентов предложения ВЫВОД1.РЕЗ, например, так:

ДЕЙСТВИЯ' НА' ПРИМЕР1 ВЫЧИСЛИТЬ' ВЫВОД1.РЕЗ;

Во время выполнения программы необходимо на сообщение TABLE NAME ответить именем созданного заранее шаблона. После этого данные выводятся по этому шаблону.

Автоматическая генерация формата. Во многих случаях пользователь системы ПРИЗ, создавая свои программы ввода-вывода,

имеет следующие трудности. После того как он запрограммировал в своих модулях нужные форматы данных, структуры моделируемых на языке УТОПИСТ объектов изменились. Пользователю приходится тогда менять форматы, снова транслировать модули и т. д. В помощь программисту система ПРИЗ предусматривает в таких случаях следующий порядок работы:

программирование модулей ввода-вывода на Фортране;

включение в каждый модуль ввода-вывода следующих двух строк:

```
INTEGER  NAME(2), FORM(60), LENGTH
         .
         .
         .
CALL  FORMAT (NAME, FORM, LENGTH)
```

где NAME — имя объекта на языке УТОПИСТ;

FORM — массив, содержащий формат;

LENGTH — длина объекта NAME.

Параметр с именем NAME входной, остальные — выходные.

Семантика используемого модуля FORMAT состоит в том, что модуль, получив имя объекта на языке УТОПИСТ, синтезирует по описанию объекта соответствующий фортрановский формат, вычисляет длину объекта и передает оба значения вызывающей программе.

Пример вывода данных:

```
WRITE (6, FORM) (M(I), I = 1, LENGTH),
```

где M — выдаваемый массив.

Разработка языка ввода-вывода. Для разработки языка ввода-вывода необходимо придерживаться требований, предъявляемых к разработке пакетов программ (п. 4.5). Здесь мы приведем только простейший пример.

Допустим, что нам необходимо напечатать значение какого-нибудь вещественного объекта, для этого опишем объект типа вывода следующим образом:

```
ПЕЧАТВЕЩ: (ПАР,РЕЗ:ВЕЩЕСТВЕННЫЙ';
           МОД' ПЕЧАТЬ ВХ' ПАР ВЫХ' РЕЗ);
```

Модуль ПЕЧАТЬ может быть запрограммирован, например, следующим образом:

```
      SUBROUTINE    ПЕЧАТЬ (ПАР,РЕЗ)
        WRITE         (6,10) ПАР
   10   FORMAT        (1X, F5.1)
        RETURN
        END
```

Теперь используем ПЕЧАТВЕЩ при вычислении значения объекта Y в следующей задаче:

```
ПРОГРАММА' ПР,
ПУСТЬ' X, Y:ВЕЩ';
УРАВ' X = Y + 5;
ЗАДАНО' X = 10;
```

```
ВЫВОД: ПЕЧАТВЕЩ  Y;
ДЕЙСТВИЯ'
        НА' ПР ВЫЧИСЛИТЬ' ВЫВОД.РЕЗ;
КОНЕЦ'; + + +
```

Модель этой задачи приведена на рис. 29. Так как требуется вывести объект Y, то в переделках определяется тождественная связь между объектами Y и ВЫВОД.ПАР. При выполнении программы сперва объекту X присваивается начальное значение, равное 10, затем по уравнению вычисляется значение объекта Y и, наконец, обращаются к модулю ПЕЧАТЬ, который выводит вычисленный результат.

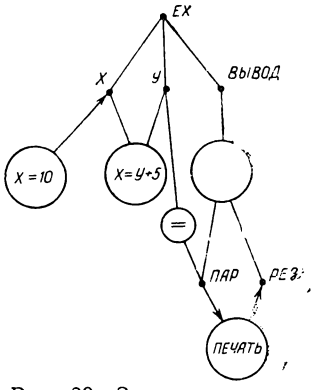


Рис. 29. Задача вывода

Описанные выше возможности и принципы реализуют обмен данных в виде текстов. Аналогично можно запрограммировать и включить в систему возможности выдачи результатов в графическом виде.

Программа ввода-вывода, предназначенная для работы в системе ПРИЗ, может обладать независимостью от устройств, так как вся система работает под управлением операционной системы ОС.

4.5. ПАКЕТЫ ПРОГРАММ

Пакеты программ, создаваемые в системе ПРИЗ. Пакетом прикладных программ, создаваемым с помощью системы программирования ПРИЗ, является совокупность программ, совместимых между собой и обеспечивающих решение задач из некоторой прикладной области, называемой предметной областью пакета. Каждый создаваемый пакет снабжается входным языком высокого уровня, являющимся расширением языка УТОПИСТ.

Пакет прикладных программ состоит из тела пакета и организующей программы. Тело пакета содержит библиотеку модулей, куда входят все прикладные программы, заранее запрограммированные для выполнения отдельных этапов решения задач из предметной области пакета, а также библиотеку семантических моделей, описывающих понятия из предметной области пакета. Эти семантические модели вместе образуют модель предметной области.

Организующая программа управляет работой пакета при решении задач, задаваемых на входном языке пакета. По отношению к пакетам, строящимся в системе ПРИЗ, в роли организующей программы выступают системные программы ПРИЗа, перечисленные в п. 3.1.

Организующая программа

управляет работой пакета, осуществляет связь с операционной системой;

воспринимает описание задачи на языке пакета и переводит его на внутренний язык системы;

определяет последовательность модулей, которые необходимо выполнить для решения задачи;

организует связь между модулями, объединяя их в программу решения задачи или вызывая модули по очереди для выполнения действий по решению задачи.

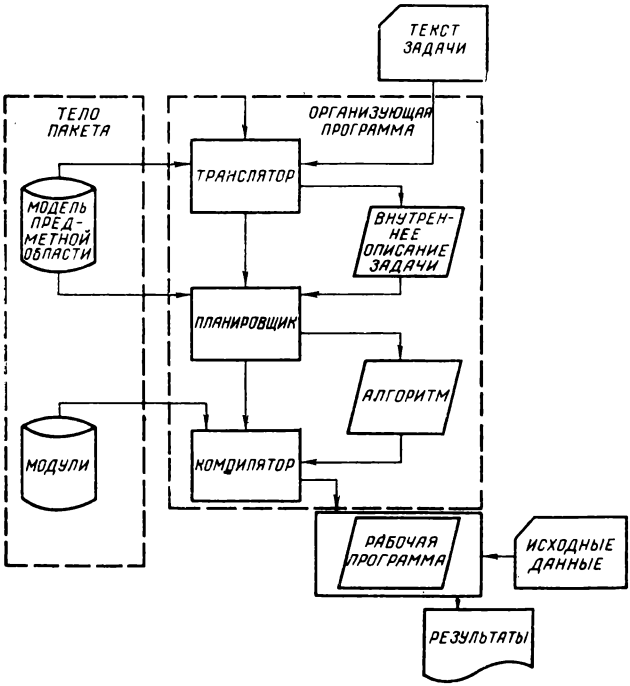


Рис. 30. Схема пакета программ

На рис. 30 показана схема пакета программы, в которой выделены отдельные компоненты организующей программы, выполняющие перечисленные выше функции. Информационные связи показаны тонкими линиями. Поступающий на вход пакета текст задачи переводится на внутренний язык транслятором, который учитывает семантику понятий языка пакета, заданную в модели предметной области. По описанию задачи на внутреннем языке планировщик определяет последовательность выполнения модулей и выдает на внутреннем языке алгоритм решения задачи. Наконец, компилятор составляет программу решения задачи в виде, готовом для выполнения.

Язык пакета предназначен для описания задач, решаемых в пакете.

В системе ПРИЗ языки пакетов создаются как расширения об-

щего ядра (языка УТОПИСТ). Элементарные виды и многие синтаксические конструкции создаваемого языка содержатся уже в ядре. Этот способ обеспечивает единство основной синтаксической структуры входных языков разных пакетов, создавая благоприятные условия для применения разных пакетов совместно.

Этапы разработки пакетов программ. В разработке пакета программ должны принимать участие как программисты, так и специалисты по предметной области пакета. Организация совместной работы специалистов из разных областей всегда является трудной проблемой. В данном случае трудности значительно увеличиваются еще сложностью самой цели. Если в прикладном программировании целью является создание программы для решения некоторой задачи, то в данном случае целью служит создание совокупности программ для решения совокупности задач. Даже если решены все проблемы, связанные с программированием, остаются трудности с описанием класса подлежащих решению задач и трудности, связанные с необходимостью принимать проектные решения, правильные для всех задач из предметной области пакета. Такие решения приходится принимать при разработке входного языка пакета. Именно поэтому проектирование входного языка рассмотрено в данной книге подробнее других этапов.

Разработка пакета программ настолько сильно отличается от разработки отдельной программы, что опыт программиста, накопленный при решении прикладных задач, бывает явно недостаточен для построения хорошо функционирующего пакета.

При построении пакета совершенно необходимым являются хорошие знания в предметной области и методах решения задач. Носителями этих знаний являются специалисты по предметной области. Но построение пакета требует формализации этих знаний в такой степени, которая часто непривычна для этих специалистов. В данной книге предлагается нисходящий метод построения входного языка, при котором строгая формализация и принятие многих мелких решений откладываются на поздние этапы. Разработка языка начинается с определения основных понятий и их содержательного описания. За этим следует проверка множества выбранных понятий на полноту, которая проводится путем описания типовых задач с помощью выбранных понятий. На этом же этапе уточняется семантика понятий. Лишь затем составляются формальные описания понятий. Естественно, такой процесс не доходит сразу до конца. Приходится возвращаться к введению новых понятий, опять их испытывать и затем точно описывать, возможно, изменяя уже существующие описания.

В форме инструкции можно дать лишь общую схему работ. Поэтому особенно рекомендуем внимательно изучить все примеры, имеющиеся по построению пакетов.

Здесь перечисляются работы, которые должны быть выполнены при построении пакета программ. Только первая из них — составление технического задания — однозначно соответствует одному из этапов проведения ОКР. Остальные работы могут по-разному

распределяться между этапами технического и рабочего проектирования и изготовления системы.

1. Составление технического задания. Этот этап включает оценку ожидаемой эффективности разработки, точную постановку цели в виде требований на готовый продукт (пакет программ), разработку рекомендации концептуального характера и составление графика работ на все последующие этапы. Этап выполняется согласно требованиям на составление технических заданий НИР и ОКР.

2. Описание класса решаемых задач. Класс подлежащих решению задач описывается содержательно в терминах соответствующей специальности. Приводится классификация решаемых задач, близкие по методам решения и сложности задачи объединяются в один тип задач. Для каждого типа задач задаются

пример;

числовые характеристики, определяющие размерность задачи;

относительная частота появления задач данного типа;

методы решения;

требования к вводу-выводу.

Для создаваемого пакета целиком определяется режим работы с пакетом: пакетный режим, диалог в ходе трансляции, диалог в ходе счета.

3. Модульный анализ. На этом этапе выделяются подзадачи, решение которых обеспечит решение задач данного класса, разрабатываются алгоритмы решения подзадач. Целью модульного анализа является определение такой совокупности модулей, из которых можно составлять программы для решения всех задач разрабатываемого пакета.

4. Разработка входного языка пакета. Определяется тип языка:

независимый, в виде расширения языка УТОПИСТ;

включенный в язык программирования Фортран или в язык Ассемблера.

Определяются основные понятия языка, и их семантика описывается в виде текстов на языке УТОПИСТ.

5. Программирование, документирование и отладка модулей. Выполняются согласно принятой технологии программирования.

6. Введение в пакет описания предметной области и комплексная отладка пакета. Этот этап является непосредственным продолжением этапа 4 и заключается во введении в библиотеку моделей понятий входного языка, заданных на языке УТОПИСТ.

7. Составление эксплуатационной документации пакета. Эксплуатационная документация пакета содержит общее описание пакета с указанием назначения и возможностей пакета, описание входного языка пакета и все инструкции, необходимые для решения задач в пакете под управлением операционной системы ОС ЕС.

8. Опытная эксплуатация и приемка пакета. Выполняется согласно общим требованиям к опытной эксплуатации и приемке программного продукта.

Разработка входного языка пакета. Построение входного языка пакета начинается составлением таблицы, содержащей те понятия предметной области, которые могут использоваться при решении задач. При составлении таблицы величин следует туда заносить все величины предметной области, так как размер таблицы не влияет на размер программ, создаваемых в пакете.

Для каждого понятия указываются

содержательное название;

общепринятое обозначение;

идентификатор, применяемый для обозначения данного понятия в пакете;

подчиненные ему понятия;

тип;

максимальный объем памяти, требуемый для хранения значений.

Типы величин описываются средствами языка УТОПИСТ и могут быть следующие: вещественные, целые, логические, строковые, память, ряд, составные, неопределенные.

Составление списка понятий является наиболее сложным и ответственным этапом разработки языка. Здесь необходимы хорошие знания из предметной области, а также достаточно полное представление о возможностях, которые язык УТОПИСТ дает для совместного использования понятий.

При первоначальном составлении списка понятий рекомендуется полностью отвлечься от ограничений, налагаемых языком УТОПИСТ, и включать в список все те содержательные понятия, которые необходимы при описании задач данной предметной области.

После определения величин описываются отношения между величинами. Для этого составляется схема модели предметной области, по ней проверяется разрешимость задач данного пакета и составляется таблица модулей, входящих в тело пакета. Для каждого модуля указываются идентификатор, тип, ранг, список параметров. В списке параметров для каждого параметра указывается его тип связи с модулем.

На основании таблицы понятий, схемы модели предметной области и таблицы модулей составляется описание модели предметной области на языке УТОПИСТ.

Обеспечение совместимости пакетов программ. С помощью системы программирования ПРИЗ могут быть построены интегрированные системы программирования, включающие многие пакеты программ, которые совместно применимы для решения задач. Так могут объединяться пакеты общего назначения — по статистике, оптимизации, линейной алгебре — с пакетами специальными, например по проектированию силовых полупроводниковых приборов.

Для совместного использования пакетов необходимо обеспечить их программную, информационную и функциональную совместимость.

Модули совместно используемых пакетов должны быть одновременно доступны редактору связей при сборке программы решения задачи. Модели предметных областей совместно применяемых пакетов должны помещаться в одной и той же библиотеке. Системой программирования ПРИЗ обеспечивается полная программная совместимость пакетов программ, если имена модулей и общих областей в разных пакетах не совпадают и если в модулях не используется непомеченных общих областей.

Для обеспечения информационной совместимости пакетов необходимо, чтобы понятия, которые в разных пакетах соответствуют друг другу и могут связываться между собой отношением эквивалентности, имели сопоставимые виды, между которыми допускается отношение эквивалентности.

При составлении интегрированной системы программирования из нескольких пакетов для каждого пакета определяются внешние понятия, т. е. понятия, которые могут связываться с понятиями других пакетов. Для внешних понятий составляется таблица соответствий, в которой для каждого внешнего понятия указаны соответствующие понятия из других пакетов.

По составленной таблице проверяется сопоставимость видов соответствующих понятий. Если соответствующие друг другу внешние понятия пакетов имеют несопоставимые виды и переопределять понятия в пакетах нецелесообразно, можно информационную совместимость обеспечить методом трансляции данных по следующей схеме. Для совокупности соответствующих внешних понятий определяется новое понятие, куда в качестве компонент включаются понятия всех тех видов, которые требуют преобразования. В этом же понятии определяются отношения, по которым происходит преобразование данных при передаче их из одного пакета в другой.

Пример

В разных пакетах используется понятие силы, измеряемой в разных единицах. Вводится новое понятие СИЛЫ, куда включаются компоненты во всех используемых единицах измерения. Вводятся уравнения, определяющие отношения между величинами в разных единицах. Новое понятие силы вводится в библиотеку на самом верхнем уровне так, что оно может одинаково использоваться в любом пакете:

```
ПАКЕТ1: (. . .Г : ВЕЩ'; . . .);
ПАКЕТ2: (. . .КГ : ВЕЩ'; . . .);
ПАКЕТ3: (. . .Т : ВЕЩ'; . . .);
СИЛЫ: (Г,КГ,Т : ВЕЩ';
        УРАВ'Г = 1000*КГ;
        УРАВ'КГ = 1000*Т);
```

Если теперь в некоторой задаче будут определены силы F1, F2 и F3 из разных пакетов, то они могут быть связаны между собой единым понятием силы:

```
F1:ПАКЕТ1.Г;
F2:ПАКЕТ2.КГ;
```

```
F3:ПАКЕТ3.Т;
F:СИЛЫ  F1,F2,F3;
```

При такой записи все три силы F1, F2 и F3 будут между собой связаны и при необходимости передачи значений будет происходить правильное их преобразование.

Функциональная совместимость пакетов должна обеспечиваться разработчиками пакетов на содержательном уровне. Для этого необходимо при построении пакетов достаточно точно определять содержание понятий и программных модулей.

Пример построения пакета. На основе соглашений, приведенных выше для построения пакетов программ, рассмотрим пример построения пакета программ для управления базами данных. Данный пример является также некоторым введением и предварительным описанием пакета СУБД (см. гл. 5).

Составление технического задания. Приведем основные требования к создаваемому пакету программ для управления базами данных. Разрабатываемый пакет программ должен соответствовать двум основным критериям:

пакет должен быть расширением системы ПРИЗ, обеспечивая другим создаваемым в системе ПРИЗ пакетам сервис обработки данных (это достигается интеграцией пакетов);

пакет должен функционировать отдельной системой для обслуживания баз данных в диалоговом режиме.

Разработка должна соответствовать основным концепциям реляционных баз данных, но одновременно она должна быть несколько шире, чем эта модель, позволяя описывать записи иерархической структуры и реализуя на языках баз данных некоторые возможности РПГ. При разработке языков баз данных следует придерживаться классической структуры, создавая соответственно язык описания данных, язык манипулирования данными и язык запросов. Эти языки создаются как расширения языка УТОПИСТ. Для проектирования баз данных необходимо ограничиваться максимальным объемом дисковой памяти 7 Мб, а схемы баз данных хранить на средствах системы ПРИЗ.

Разработку пакета планируется провести в следующей последовательности:

используя результаты модульного анализа задач обработки данных, разработать входные языки баз данных;

запрограммировать модули пакета;

провести комплексную отладку пакета и разработать процедуру генерации пакета;

составить эксплуатационную документацию пакета;

провести опытную эксплуатацию пакета, загружая базы данных данными кадрового учета, и провести приемку пакета.

Описание класса решаемых задач и модульный анализ. Модульный анализ задач обработки данных несколько проще, чем анализ большинства других менее «популярных» областей, для которых требуется проектировать пакеты программ. Здесь много решений можно найти уже в учебной литературе.

В настоящее время данные обрабатываются чаще всего с использованием баз данных. Для их обслуживания создаются программные комплексы — системы управления базами данных (СУБД). Существуют три основные модели организации баз данных — реляционные, иерархические и сетевые. В техническом задании на данный пакет программ была поставлена задача на проектирование системы управления базами данных расширенного реляционного типа. Такой постановкой задачи уже многое определено. В реляционных базах данных обработка данных осуществляется операциями α-алгебры. Учитывая, что мы смотрим на экземпляры отношения как на множества записей, выделим следующие операции нашего языка: подмножество, проекция, соединение, кванторы общности и существования, теоретико-множественные операции, операции ввода-вывода и др. Для языков баз данных предусмотрены пакетный и диалоговый режимы работы.

Разработка входного языка пакета. Изложим понятия языка баз данных в виде следующей табл. 5.

Таблица 5

Таблица понятий баз данных

Наименование	Идентификатор	Параметры	Вид	Объем памяти
Множество:	МНОЖ			
запись		ЗАПИСЬ	неопр	
имя множества		ИМЯ	строка	1
ключ записи		КЛЮЧ	неопр	
номер записи		НОМЕР	цел	1
Подмножество:	ПОДМНОЖ			
исходное множество		ОТ	МНОЖ	
результирующее множество		ЕСТЬ	МНОЖ	
условие		УСЛОВ	лог	1
номер подзадачи		Н	цел	1
Проекция:	ПРОЕКЦИЯ			
исходное множество		ОТ	МНОЖ	
результирующее множество		ЕСТЬ	МНОЖ	
формат записи		ФОРМАТ	строка	2
номер подзадачи		Н	цел	1
Квантор общности:	ВСЕ			
множество		ОТ	МНОЖ	
условие		УСЛОВ	лог	1
результат		РЕЗ	лог	1
номер подзадачи		Н	цел	1

Для описанных выше понятий требуется запрограммировать ряд модулей, реализующих семантику данных понятий. Эти модули (табл. 6) следующие:

последовательный выбор записей из множества;

выбор записей из множества по ключу;

вычисление подмножества;
вычисление проекции множества;
вычисление квантора общности.

Таблица 6

Модули пакета

Идентификатор	Тип	Ранг	Список параметров
SETNXT	Программа	1	ВХ МНОЖ.ИМЯ, МНОЖ.НОМЕР ВЫХ МНОЖ.ЗАПИСЬ
SETKEY	Программа	1	ВХ МНОЖ.ИМЯ, МНОЖ.КЛЮЧ ВЫХ МНОЖ.ЗАПИСЬ
SUBSEM	Программа	2	ВХ ОТ.ИМЯ ВЫХ ЕСТЬ.ИМЯ, Н ВВХ ОТ.НОМЕР ВВЫХ ОТ.ЗАПИСЬ, УСЛОВ
PROJM	Программа	2	ВХ ОТ.ИМЯ, ФОРМАТ ВЫХ ЕСТЬ.ИМЯ, Н ВВХ ОТ.НОМЕР ВВЫХ ЕСТЬ.ЗАПИСЬ
ALLM	Программа	2	ВХ. ОТ.ИМЯ ВЫХ. РЕЗ, Н ВВХ ОТ.НОМЕР ВВЫХ УСЛОВ

Программирование, документирование и отладка модулей. Описанные выше модули создаются на языке Ассемблера и Фортране.

Введение в пакет описания предметной области и комплексная отладка пакета. Мы будем опираться на результаты разработки входного языка пакета. Ввод в пакет описаний понятий баз данных описывается следующим текстом:

```
ЗАДАЧА'   БД;
ПУСТЬ'      МНОЖ:   (ЗАПИСЬ:НЕО';
                     ИМЯ : СТР';
                     КЛЮЧ : НЕО';
                     НОМЕР : ЦЕЛ';
            МОД'     SETNXT ВХ' ИМЯ, НОМЕР ВЫХ' ЗАПИСЬ;
            МОД'     SETKEY ВХ' ИМЯ, КЛЮЧ ВЫХ' ЗАПИСЬ);
ПОДМНОЖ:    (ОТ: МНОЖ;
            ЕСТЬ : МНОЖ;
            Услов : ЛОГ';
            Н : ЦЕЛ';
            МОД' SUBSEM    2    ВХ' От. ИМЯ ВЫХ' ЕСТЬ.ИМЯ, Н
                          ВВХ'ОТ.НОМЕР ВВЫХ'ОТ.ЗАПИСЬ,УСЛОВ);
ПРОЕКЦИЯ:   (ОТ : МНОЖ;
            ЕСТЬ : МНОЖ;
            ФОРМАТ : СТР';
            Н :    ЦЕЛ';
```

```
                МОД' PROJM      2 ВХ' ОТ.ИМЯ, ФОРМАТ ВЫХ'ЕСТЬ.ИМЯ,Н
                                  ВВХ' ОТ.НОМЕР ВВЫХ' ЕСТЬ.ЗАПИСЬ);
ВСЕ               (ОТ: МНОЖ;
                  УСЛОВ : ЛОГ';
                  РЕЗ : ЛОГ';
                     Н : ЦЕЛ';
                МОД' ALLM       2ВХ' ОТ.ИМЯ ВЫХ' РЕЗ, Н
                                  ВВХ' ОТ.НОМЕР ВВЫХ'УСЛОВ);
ВВЕСТИ' БД; +++
```

Следующим этапом работ является испытание языка на большом числе примеров. При испытаниях следует использовать средства системы ПРИЗ — библиотеку моделей и транслятор УТОПИСТ. В ходе испытаний для тестовых задач должны получаться полностью правильные программы их решения, которые после редактирования связей могут быть действительно использованы для вычислений.

Для испытания языка не нужны модули, так как они требуются только на этапе редактирования связей. Комплексную отладку всего пакета можно начинать сразу после отладки первых модулей пакета и загрузки первого набора данных.

Составление эксплуатационной документации пакета. Содержимое эксплуатационной документации данного пакета программ почти полностью излагается в гл. 5.

Опытную эксплуатацию данного пакета программ целесообразно провести с данными кадрового учета организации разработчика.

4.6. СОСТОЯНИЕ И ГЕНЕРАЦИЯ СИСТЕМЫ

Здесь приводятся описания расположения компонент системы ПРИЗ ЕС на машинных носителях информации, перевода системы в рабочее состояние и проверка правильности работы системы.

Состояние системы. В состав ПРИЗ ЕС входят транслятор, генератор, Ассемблер, редактор связей. Рабочий экземпляр стандартной версии системы ПРИЗ ЕС образуется во время процесса, называемого генерацией системы. Генерация системы представляет собой последовательность выполнения нескольких шагов задания ОС ЕС, управляющие предложения которого находятся на дистрибутивной магнитной ленте PRIZDS.

Система ПРИЗ распространяется на дистрибутивной магнитной ленте PRIZDS, содержащей

управляющие предложения генерации PRIZINIT;
библиотеку ПРИЗа PRIZLIB;
процедуры ПРИЗа PRIZPROC;
тестовую задачу PRIZTEST;
таблицу синтаксического управления PLAAN;
макробиблиотеку ПРИЗа MACROLIB;
вспомогательную библиотеку ABILIB (рис. 31).

Библиотека PRIZLIB, содержащая программные секции транслятора и генератора системы ПРИЗ, является разгрузочной копией набора данных, организованного разделами и полученного путем

копирования библиотеки с устройства прямого доступа на магнитную ленту. Копирование выполняется утилитой IEHMOVE.

Все библиотеки системы ПРИЗ находятся на отдельном магнитном диске PRZLIB.

Процедуры ПРИЗа находятся в системной библиотеке ОС SYS1. PROCLIB, куда их записывают утилитой IEBGENER.

Для переписи процедур ПРИЗа с магнитной ленты PRIZDS в библиотеку SYS1.PROCLIB требуется 10 дорожек свободной памяти на резидентном диске.

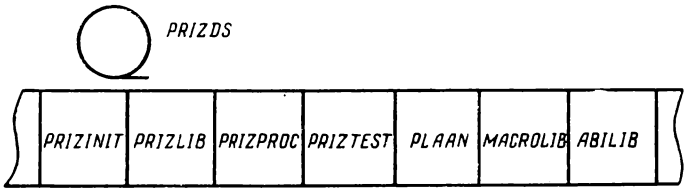


Рис. 31. Содержание ленты PRIZDS

Генерация системы проходит в трех стадиях (предусмотрен также режим генерации в двух стадиях, см. ниже Руководство оператора). Первая стадия заключается в перфорации управляющих предложений генерации с магнитной ленты. Результатом данной стадии является колода перфокарт, представляющая собой задание ОС ЕС, которое должно быть задано на выполнение. Это задание состоит из нескольких шагов, в каждом из которых выполняется некоторая утилита ОС ЕС. Выполнение данного задания представляет собой вторую стадию генерации системы. На этой стадии осуществляются:

копирование библиотеки PRIZLIB системы ПРИЗ;

перепись управляющих процедур системы ПРИЗ в библиотеку процедур SYS1, PROCLIB;

копирование набора данных PLAAN;

копирование макробиблиотеки MACROLIB;

создание библиотеки моделей;

копирование библиотеки ABILIB;

перфорирование тестовой задачи.

Выполнение тестового задания представляет собой третью стадию генерации системы.

Перфорация процедуры генерации системы ПРИЗ ЕС с магнитной ленты PRIZDS осуществляется утилитой IEBGENER (рис. 32). Процедура генерации PRIZINIT является первым набором данных на дистрибутивной магнитной ленте PRIZDS (см. рис. 31). Управляющие предложения для утилиты приведены ниже:

```
// /PRIZ1        JOB     MSGLEVEL=1
// /             EXEC    PGM=IEBGENER
// /SYSPRINT     DD      SYSOUT=A
```

```
/ /SYSIN        DD      DUMMY
/ /SYSUT1       DD      DSNAME=PRIZINIT,UNIT=5010,LABEL=(1,SL),
/ /             DISP=OLD,VOL=SER=PRIZDS,DCB=(RECFM=FB,
/ /             LRECL=80,BLKSIZE=80)
/ /SYSUT2       DD      UNIT=SYSCP, DCB=BLKSIZE=80
/ /
```

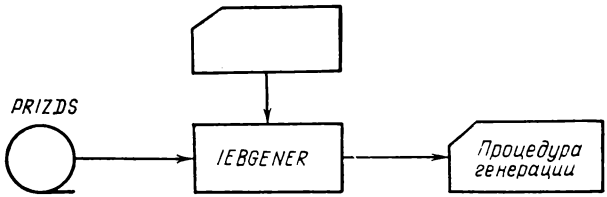


Рис. 32. Первая стадия генерации

Вторая стадия генерации представляет собой последовательность выполнения семи шагов задания ОС ЕС (рис. 33):

1) перепись библиотеки PRIZLIB системы ПРИЗ с магнитной ленты PRIZDS на магнитный диск системы ПРИЗ;

2) перенос процедур ПРИЗа с магнитной ленты в библиотеку SYS1.PROCLIB;

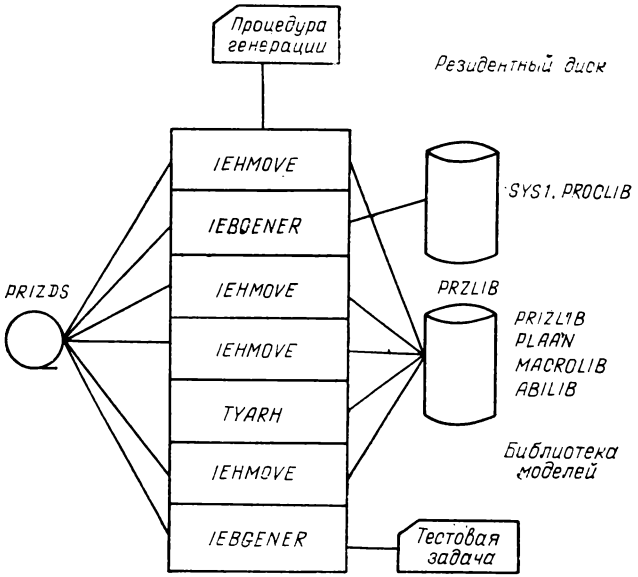


Рис. 33. Вторая стадия генерации

3) копирование таблицы синтаксического управления PLAAN с магнитной ленты PRIZDS на магнитный диск системы ПРИЗ;

4) копирование макробиблиотеки MACROLIB системы ПРИЗ с магнитной ленты PRIZDS на магнитный диск системы ПРИЗ;

5) создание библиотеки моделей;

6) копирование вспомогательной библиотеки ABILIB системы ПРИЗ с магнитной ленты PRIZDS на магнитный диск системы ПРИЗ;

7) перфорация тестовой задачи с магнитной ленты PRIZDS.

Тестовая задача, выполняемая после создания системы ПРИЗ ЕС на магнитных дисках, демонстрирует возможности системы ПРИЗ и служит минимальным тестом этих возможностей. Кроме того, она представляет собой пример написания задачи в языке УТОПИСТ (рис. 34).

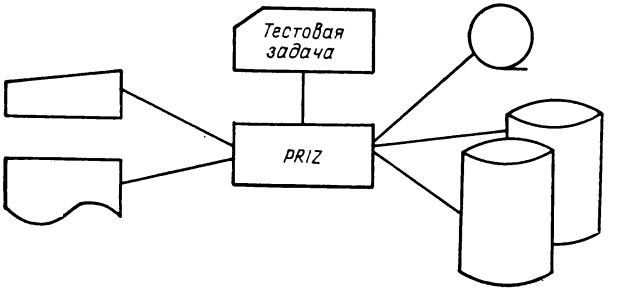


Рис. 34. Третья стадия генерации

Колода перфокарт тестовой задачи состоит

из операторов языка управления заданиями, вызывающих каталогизированные процедуры ПРИЗа для трансляции, генерирования, редактирования связей и выполнения;

из тестовой программы на языке УТОПИСТ;

из операторов DD (не входящих в состав каталогизированных процедур ПРИЗа) для наборов данных, используемых при выполнении задачи.

Модули для решения тестовых задач находятся во вспомогательной библиотеке ABILIB.

Эта задача проверяет работоспособность системы ПРИЗ ЕС.

Рекомендации по операторской работе. Режим генерации системы ПРИЗ в трех стадиях предусматривает следующий порядок работ:

монтирование системных магнитных дисков, магнитного диска системы ПРИЗ и магнитной ленты PRIZDS;

выполнение загрузки операционной системы ОС, обеспечивая состояние готовности лентопротяжного механизма с магнитной лентой PRIZDS и системного выходного перфоратора;

размещение колоды карт первого задания генерации системы ПРИЗ в устройство ввода перфокарт и выполнение задания *;

снятие колоды перфокарт с устройства выходного перфоратора, размещение ее на устройстве ввода с перфокарт и выполнение задания;

снятие тестовой задачи с устройства выходного перфоратора, размещение ее на выполнение в устройство ввода с перфокарт и выполнение задания.

Режим генерации в двух стадиях предусматривает следующий порядок работ:

монтирование системных магнитных дисков, магнитного диска системы ПРИЗ и магнитной ленты PRIZDS;

выполнение загрузки операционной системы ОС, обеспечивая состояние готовности лентопротяжного механизма с магнитной лентой PRIZDS;

освобождение очереди заданий, остановка процедуры системного ввода RDR и выдача команды *

S RDR,UNIT, PRIZDS, DSN=PRIZINIT, LABEL=1

UNIT — это адрес устройства, на которое установлена дистрибутивная лента, например 280;

после завершения задания PRIZ система готова к работе. Для проверки работоспособности системы запустить для тестового примера

S RDR, UNIT, PRIZDS, DSN=PRIZTEST, LABEL=4

---

* После начала выполнения задания PRIZ следует запрос на разрешение записи в системные библиотеки

*00 IEC1070 E,DDD,SER,JJJ,AMEND, SYS1. PROCLIB

Ответить на это нужно командой

R 00,'U'

ГЛАВА 5

ППП ДЛЯ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

В этой главе дается описание пакета прикладных программ для управления базами данных — системы управления базами данных, называемой DABU. Данный пакет является одним из первых созданных в системе ПРИЗ ЕС пакетов прикладных программ и служит одновременно тестом для проверки работы и возможностей системы ПРИЗ ЕС. Кроме того, в пакете программ DABU реализованы некоторые новые решения из области баз данных (в частности, автоматический синтез поисковых программ, возможность одновременной реализации разных моделей баз данных и др.), представляющих научный интерес для специалистов в этой области. Специально эти вопросы здесь не выделены, а излагаются в ходе описания пакета.

Назначение и область применения. Пакет прикладных программ для управления базами данных DABU представляет собой комплекс средств для создания и использования баз данных, допускающих сложные запросы и интегрированную обработку данных совместно с другими пакетами прикладных программ. DABU применим также для построения небольших баз данных, например системы учета кадров, системы описания оборудования завода и др.

Особенности системы. Данный пакет прикладных программ реализует множественную модель базы данных. Структура данных описывается в терминах множеств объектов, над которыми определены операции, применяемые для описания запросов. С одной стороны, понятие множества в данной СУБД близко к понятию отношения, введенного Коддом для описания реляционных баз данных (РБД). С другой стороны, реализованная в данной системе множественная модель шире, чем классическая реляционная модель баз данных. Существенно, что элементом множества может быть объект любого типа, описанный на языке УТОПИСТ. Объекты на языке УТОПИСТ строятся в большинстве случаев в виде древовидной структуры — это первое отличие от классической реляционной модели баз данных, где элементы кортежа располагаются последовательно, на одном уровне.

Входные языки пакета охватывают все операции α-алгебры для реляционных баз данных. Кроме операции α-алгебры, в языках системы реализован небольшой генератор отчетов (подобный РПГ) — это второе отличие системы от реляционной модели.

Ядро системы DABU содержит небольшое количество основных понятий, таких, как множество, подмножество, операции с множествами. Обработка множеств в системе DABU аналогична обработке на языке СЕТЛ. Третья особенность системы состоит в

том, что множества могут обрабатываться не только операциями реляционной модели, но и из них можно строить иерархическую структуру и применять к ним операции иерархической модели.

Реализация иерархической модели предусматривает стыковку системы DABU с СУБД иерархического типа, но описание ее здесь не приводится.

Для работы с базами данных система DABU имеет диалоговые языки, которые являются расширениями языка УТОПИСТ. Особенностью диалога с пакетом СУБД DABU являются высокий уровень запросов и автоматизация синтеза программ обработки запросов.

5.1. ЯЗЫКИ БАЗ ДАННЫХ

Языки баз данных пакета прикладных программ DABU основываются на входном языке системы ПРИЗ ЕС УТОПИСТ и являются его расширениями. Языки разделены на те классы задач, какие встречаются в большинстве систем управления базами данных. Для данного пакета разработаны следующие языки:

язык описания данных;

язык запросов;

язык манипулирования данными;

расширения языков программирования средствами описания и манипулирования данными;

макроязык запросов.

Язык описанных данных (ЯОД) является расширением некоторого подмножества языка УТОПИСТ понятием множества — МНОЖ. На языке описания данных задается описание схемы базы данных, состоящее из описаний объектов и описаний множеств:

```
Описание объекта : : =
    ид:   ( { ид, . . . : описатель;          } . . . );
            { описание-объекта                }

описатель : : =  { ВЕЩЕСТВЕННЫЙ'      }
                 { ЦЕЛЫЙ'              }
                 { ЛОГИЧЕСКИЙ'         }
                 { ПАМЯТЬ' (целое)     }
                 { СТРОКА' (целое)     }
```

Описание объекта задает новый тип объекта (записи) путем описания имени и типа каждого из его полей. Целое при описателе ПАМЯТЬ' или СТРОКА' обозначает число ячеек, выделяемых под значение данного поля.

Примеры

```
ЛИЦО:  (ФИО: (ФАМИЛИЯ: СТРОКА'3;
              ИМЯ,ОТЧЕСТВО: СТРОКА' 2;);
        ВОЗРАСТ: ЦЕЛЫЙ';
        ПОЛ: ЛОГИЧЕСКИЙ');
МАТЕРИАЛ:(МАРКА:СТРОКА' 2;
          ТВЕРДОСТЬ,ПРОЧНОСТЬ: ВЕЩЕСТВЕННЫЙ';):
```

Описание — множества : : =
{ид: МНОЖ ЗАПИСЬ = имя — объекта [,КЛЮЧ = имя — текста]
{ид: имя — множества } ;

Имя-объекта и имя-множества являются именами, встретившимися ранее в крайней левой позиции в описании объекта (в описании множества).

Примеры

ГРУППА: МНОЖ ЗАПИСЬ = ЛИЦО;
НОМЕНКЛАТУРА : МНОЖ ЗАПИСЬ = МАТЕРИАЛ;

Полное описание схемы базы данных задается следующим текстом:

ПРОГРАММА' ид;
ПУСТЬ' описание — объекта ... +++
ПУСТЬ' описание — множества...+++
ВВЕСТИ' имя;
КОНЕЦ' ; +++

Идентификатор в заголовке схемы БД становится именем схемы БД. Это же имя должно быть указано в предложении ВВЕСТИ', которое включает заданное описание схемы БД в библиотеку описаний.

Пример

```
ПРОГРАММА' СХЕМАБД;
ПУСТЬ' СОТРУДНИК: (ИМЯ: СТРОКА' 3;
                   ВОЗРАСТ: ЦЕЛЫЙ';
                   ПОЛ: ЛОГИЧЕСКИЙ';);
ПУСТЬ' ОТДЕЛ: МНОЖ ЗАПИСЬ = СОТРУДНИК;
ВВЕСТИ' СХЕМАБД;
КОНЕЦ'; +++
```

Описанные выше конструкции являются достаточными для описаний разных отношений между данными. Ключ одного объекта (располагаемый всегда в начале объекта) может быть использован в качестве компонента другого объекта, т. е. так же, как в РБД.

Язык запросов (ЯЗ) является расширением языка УТОПИСТ следующими понятиями:

подмножество — ПОДМНОЖ;
объединение — ОБЪЕДИН;
соединение — СОЕДИН;
проекция — ПРОЕКЦИЯ;
ввод множества — ВВОД;
вывод множества — ВЫВОД;
квантор общности — ВСЕ;
квантор существования — НЕКОТОРЫЙ;
сумма (по всем элементам множества) — СУММА;
максимум — МАКС;
минимум — МИН;
арифметическое среднее — СРЕДН;
число элементов — ПОДСЧЕТ;

итоги — ИТОГИ;
вывод вещественного — ВЫВОДВЕЩ.

Каждому понятию соответствует предложение некоторой формы, где это понятие может быть употреблено. В таком предложении указываются свойственные данному понятию характеристики — параметры. Форма предложений следующая:

```
подмножество : : =
ид: ПОДМНОЖ МНОЖ= имя—множества, ЕСТЬ= имя—множества.
УСЛОВ=имя—логического;
объединение : : =
ид: ОБЪЕДИН  МНОЖ= имя—множества, И=имя—множества,
             ЕСТЬ= имя—множества;
соединение : : =
ид: СОЕДИН   МНОЖ=имя—множества, И=имя—множества,
             ЕСТЬ= имя—множества;
проекция : : =
ид: ПРОЕКЦИЯ МНОЖ=имя—множества, ЕСТЬ= имя—множества.
             ФОРМАТ='имя—объекта';
ввод—множества : : =
ид: ВВОД МНОЖ=имя—множества,
          ИЗ='имя—набора';
вывод множества : : =
ид: ВЫВОД МНОЖ=имя—множества,
            ФОРМАТ='имя—объекта'
        [,ПЕЧАТЬ = имя—логического]
        [,ЭКРАН  = имя—логического]   ;
   ∀ — квантор : : =
ид: ВСЕ ОТ = имя—множества,
          УСЛОВ = имя—логического
       [,РЕЗ = имя—логического]   ;
   ∃ — квантор : : =
ид: НЕКОТОРЫЙ ОТ = имя—множества,
    УСЛОВ = имя—логического
     [,РЕЗ = имя—логического]   ;
сумма : : =
ид: СУММА ЭЛЕМ =имя—вещественного,
ОТ = имя—множества
[,РЕЗ = имя—вещественного]   ;
максимум : : =
ид: МАКС ЭЛЕМ = имя—вещественного,
          ОТ    = имя—множества
       [,РЕЗ   = имя—вещественного]  ;
минимум : : =
ид:МИН ЭЛЕМ = имя—вещественного,
         ОТ    = имя—множества
      [,РЕЗ   = имя—вещественного]  ;
арифм.среднее : : =
ид: СРЕДН ЭЛЕМ = имя—вещественного,
          ОТ                    = имя—множества
          [,РЕЗ                 = имя—вещественного]  ;
число—элементов : : : =
ид: ПОДСЧЕТ МНОЖ                = имя—множества
          [,РЕЗ                 = имя—вещественного]  ;
итоги : : =
ид:ИТОГИ МНОЖ                   = имя—множества,
                ПО              = имя—переменного,
                  В             = имя—переменного
               [,РЕЗ            = имя—логического]  ;
вывод—вещественного : : =
```

```
ид: ВЫВОДВЕЩ ПЕРЕМ = имя—вещественного
                [,ПЕЧАТЬ  = имя—логического]
                [,ЭКРАН   = имя—логического]  ;
```

Из языка УТОПИСТ используется следующая конструкция для описания условий выборки элементов для подмножеств и кванторов:

```
условие : : =
ПРИСВ' имя: = логическое—выражение;
```

В начале запроса описывается подсхема, т. е. часть схемы БД (или схема БД), на которой решается конкретный запрос:

```
подсхема : : =
                  { ид: имя—схемы—БД;                  }
[ПО' имя;]ПУСТЬ'  { ид: описание—объекта ... +++       }
                  { ид: описание—множества ...+++      }
```

Текст запроса на языке УТОПИСТ имеет следующую форму:

```
запрос: : =
ПРОГРАММА'ид;
                  { подсхема базы данных          }
                  { подмножество                  }
                  { объединение                   }
                  { соединение                    }
                  { проекция                      }
                  { ввод—множества                }
                  { вывод—множества               }
                  {   ∀ —квантор                  }
                  {   Э —квантор                  }
                  { сумма                         }  ...+++
                  { максимум                      }
                  { минимум                       }
                  { арифм.среднее                 }
                  { число—элементов               }
                  { итоги                         }
                  { вывод—вещественного           }
                  { условие                       }
                  {                               }
ДЕЙСТВИЯ'
            НА' имя ВЫЧИСЛИТЬ' имя,...;
            КОНЕЦ', +++
```

Пример

```
ПРОГРАММА' ПРИМ;
ПУСТЬ' БД: СХЕМАБД;
      В:ВВОД МНОЖ = БД.ОТДЕЛ,
         ИЗ = '1ОТД';
           С:СУММА ЭЛЕМ = БД.СОТРУДНИК.ОКЛАД,
           ОТ = БД.ОТДЕЛ;
ДЕЙСТВИЯ'
           НА' ПРИМ ВЫЧИСЛИТЬ' С.РЕЗ;
КОНЕЦ'; +++
```

Предложения ЯЗ. Для повышения наглядности и облегчения составления запросов все конструкции ЯЗ иллюстрируются дополнительными схемами (семантическими моделями), из которых пользователь строит свой запрос.

Ниже приведены элементы этих схем.

Семантика предложений ЯЗ (иллюстрированных рисунками) следующая:

X:МНОЖ ЗАПИСЬ=У; определяет множество X, элементами которого являются объекты У (рис. 35).

X: ПОДМНОЖ МНОЖ=У, ЕСТЬ=Т, УСЛОВ=С; определяет подмножество Т множества У, элементами которого являются те элементы множества У, которые удовлетворяют условию С (рис. 36).

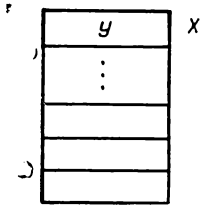


Рис. 35. Предложение МНОЖ

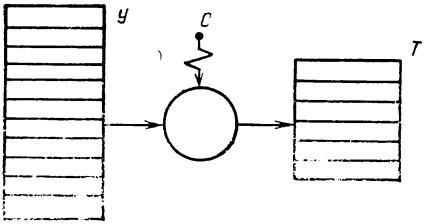


Рис. 36. Предложение ПОДМНОЖ

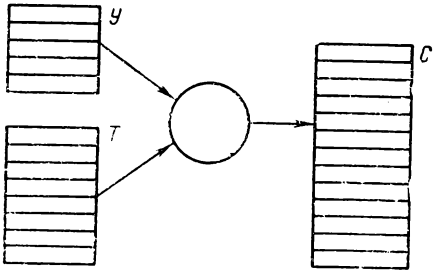


Рис. 37. Предложение ОБЪЕДИН

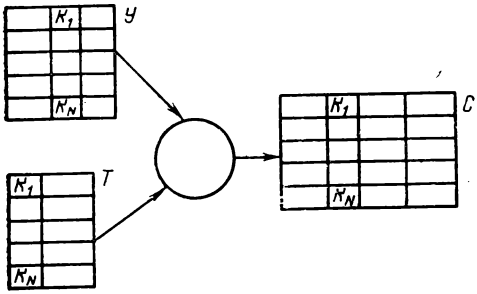


Рис. 38. Предложение СОЕДИН

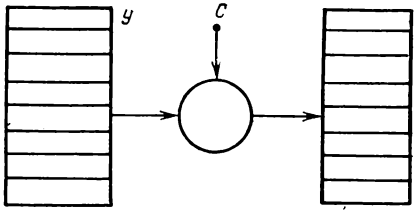


Рис. 39. Предложение ПРОЕКЦИЯ

X: ОБЪЕДИН МНОЖ=У, И=Т, ЕСТЬ=С; определяет множество С, являющееся объединением множеств У и Т (рис. 37).

X: СОЕДИН МНОЖ=У, И=Т, ЕСТЬ=С; определяет мно-

жество С, которое получается из множеств У и Т описанным ниже путем.

Каждому объекту в множестве У, имеющему поле $К_i$, соответствует в множестве Т точно один объект, имеющий ключ $К_i$. Создаваемое новое множество С имеет такое же количество элементов, как множество У. Каждый объект нового множества С содержит те же поля, что и объект множества У, но туда вложены еще, начиная с поля $К_i$, поля объекта из множества Т, т. е. так, как это делается в РБД (рис. 38).

Х: ПРОЕКЦИЯ МНОЖ=У, ЕСТЬ=Т, ФОРМАТ='С'; определяет множество Т, которое получается из множества У, используя при этом проекцию элементов по формату объекта С (рис. 39).

Х: ВВОД МНОЖ=У, ИЗ='Т'; определяет то, что множество У вводится из набора данных Т (рис. 40).

Х: ВЫВОД МНОЖ=У, ФОРМАТ='Т'; определяет то, что множество У может быть выдано на АЦПУ или на экран, используя при этом преобразование по формату объектов Т (рис. 41).

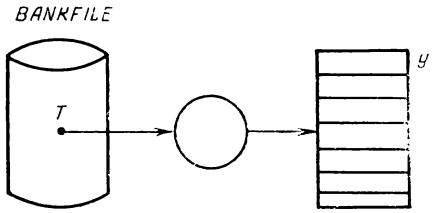


Рис. 40. Предложение ВВОД

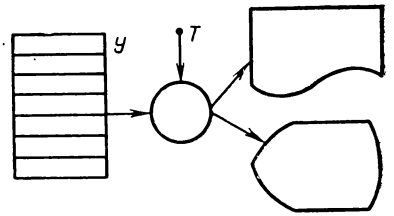


Рис. 41. Предложение ВЫВОД

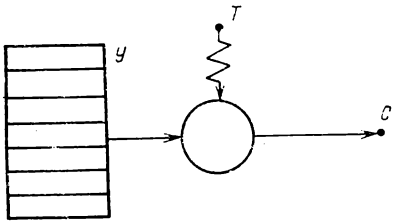


Рис. 42. Предложение ВСЕ

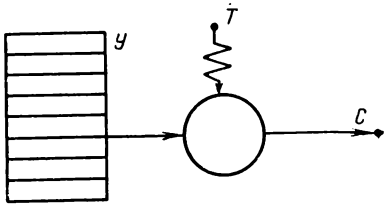


Рис. 43. Предложение НЕКОТОРЫЙ

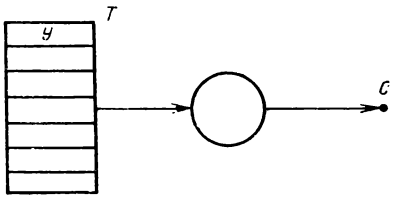


Рис. 44. Предложение СУММА

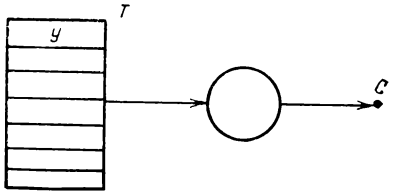


Рис. 45. Предложение МАКС

X: ВСЕ ОТ=У, УСЛОВ=Т, РЕЗ=С; определяет, что переменная С получит значение «ИСТИНА», если для всех элементов множества У справедливо условие Т, и С получает значение «ЛОЖЬ» в противоположном случае (рис. 42).

X: НЕКОТОРЫЙ ОТ=У, УСЛОВ=Т, РЕЗ=С; определяет, что переменная С получит значение «ИСТИНА», если хотя бы для одного элемента множества У справедливо условие Т, и С получает значение «ЛОЖЬ» в противоположном случае (рис. 43).

X: СУММА ЭЛЕМ=У, ОТ=Т, РЕЗ=С; определяет, что переменная С получит значение суммы элементов У множества Т (рис. 44).

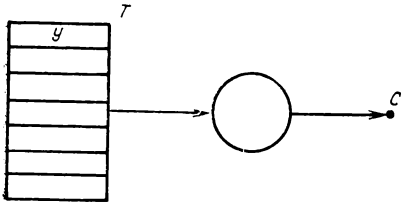


Рис. 46. Предложение МИН

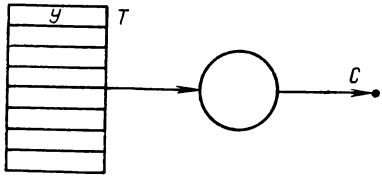


Рис. 47. Предложение СРЕДН

X: МАКС ЭЛЕМ=У, ОТ=Т, РЕЗ=С; определяет переменную С, являющуюся максимумом из элементов У множества Т (рис. 45).

X: МИН ЭЛЕМ=У, ОТ=Т, РЕЗ=С; определяет переменную С, являющуюся минимумом из элементов У множества Т (рис. 46).

X: СРЕДН ЭЛЕМ=У, ОТ=Т, РЕЗ=С; определяет, что переменная С получит значение среднего арифметического элементов У множества Т (рис. 47).

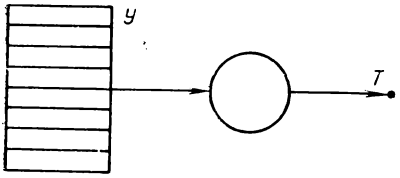


Рис. 48. Предложение ПОДСЧЕТ

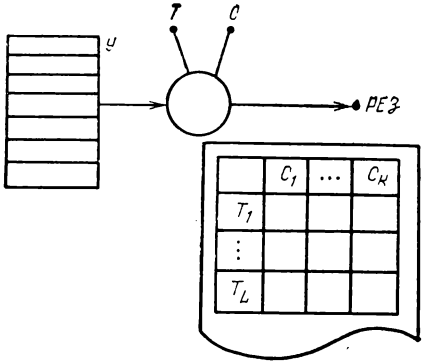


Рис. 49. Предложение ИТОГИ

X: ПОДСЧЕТ МНОЖ=У, РЕЗ=Т; определяет переменную Т, являющуюся числом элементов множества У (рис. 48).

X: ИТОГИ МНОЖ=У, ПО=Т, В=С; определяется итоговый подсчет элементов Т множества У в разрезе параметра С. Результат выполнения выводится на АЦПУ в виде таблицы, где столбцы разделены по значению параметра С и ряды по значению элементов Т (рис. 49).

X: ВЫВОДВЕЩ ПЕРЕМ=У; определяет то, что вещественная переменная У может быть выдана на АЦПУ или на экран (рис. 50).

Примеры запросов. В примерах 1 — 4 показаны некоторые возможности составления запросов. В примере 1 описан запрос для нахождения минимальной заработной платы, получаемой сотрудниками института (рис. 51).

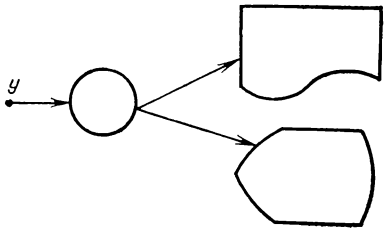


Рис. 50. Предложение ВЫВОДВЕЩ

Предложение ПРОГРАММА' идентифицирует запрос с именем ПРИМ1.

Следующее предложение определяет, что для задачи используется схема базы данных БД, которая является подсхемой для данной задачи и получает также имя БД (схема БД должна быть заранее записана в библиотеку описаний).

Предложением ВВОД вводится множество сотрудников БД.ИНСТ из набора данных 'КИБР'.

Предложение МИН определяет, что из множества сотрудников БД.ИНСТ выделяется сотрудник, который получает наименьшую

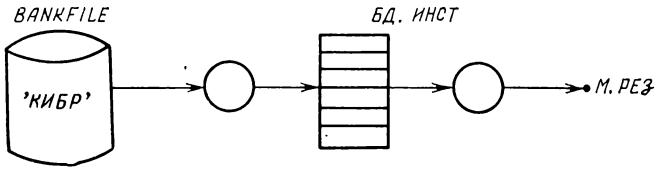


Рис. 51. Задача 1

заработную плату, и его минимальная заработная плата присваивается результату предложения МИН (см. описание предложения МИН).

В предложении ДЕЙСТВИЯ' НА' определяется задача: на модели запроса ПРИМ1 вычислить результат, т. е. минимальную заработную плату сотрудника.

Пример 1

```
* ЗАПРОС НА ВЫДЕЛЕНИЕ МИНИМАЛЬНОЙ ЗАРПЛАТЫ
        ПРОГРАММА' ПРИМ1;
* ОПРЕДЕЛЕНИЕ ПОДСХЕМЫ
        ПУСТЬ' БД:БД;
* ДОПОЛНЕНИЕ ПОДСХЕМЫ
*           А) ВВЕСТИ МНОЖЕСТВО
              В:ВВОД МНОЖ= БД.ИНСТ,ИЗ ='КИБР';
*           Б) НАЙТИ МИНИМАЛЬНУЮ ЗАРПЛАТУ
              М:МИН ЭЛЕМ=БД.СОТРУДНИК.ОКЛАД, ОТ=БД.ИНСТ;
* ПОСТАНОВКА ЗАДАЧИ
          ДЕЙСТВИЯ' НА' ПРИМ1 ВЫЧИСЛИТЬ'М.РЕЗ;
КОНЕЦ'; +++
```

В примере 2 описан запрос, который состоит в том, что из числа всех сотрудников института нужно выделить сотрудников не старше 28 лет (рис. 52).

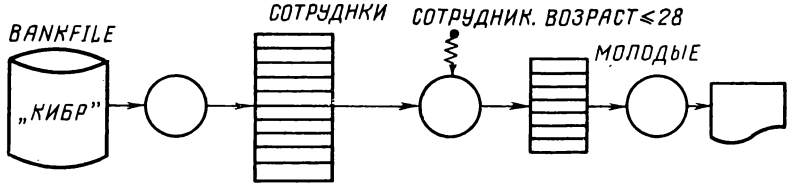


Рис. 52. Задача 2

Запрос получает имя ПРИМ2.

В этом запросе не предполагается, как в предыдущем, что схема базы данных заранее записана в библиотеку описаний. В данном случае подсхема строится в начале запроса перед предложениями ЯЗ.

Конструкция СОТРУДНИК:СОТР определяет, что новый объект СОТРУДНИК (левый в конструкции) получает структуру объекта СОТР (правая часть) из библиотеки описаний, куда эта структура заранее записана. Объект СОТРУДНИК описывает структуру данных сотрудника института (запись сотрудника).

В следующем предложении определены новые множества СОТРУДНКИ и МОЛОДЫЕ, состоящие из объектов СОТРУДНКИ.

Предложением ВВОД вводится множество сотрудников из набора данных 'КИБР'.

Предложение ПОДМНОЖ определяет, что из множества СОТРУДНКИ образуется подмножество МОЛОДЫЕ.

В следующем предложении дано условие выборки объектов (сотрудников) из множества СОТРУДНКИ в множество МОЛОДЫЕ.

В предложении ВЫВОД указывается, что надо выдать набор данных МОЛОДЫЕ, используя при этом преобразование по формату объектов СОТРУДНИК.

В предложении ДЕЙСТВИЯ' НА' определяется задача: на модели ПРИМ2 вычислить результат, т. е. распечатать данные всех сотрудников с возрастом менее 28 лет.

Пример 2

```
* ЗАПРОС НА ВЫДЕЛЕНИЕ СОТРУДНИКОВ НЕ СТАРШЕ 28 ЛЕТ
            ПРОГРАММА' ПРИМ2;
* ПОСТРОЕНИЕ ПОДСХЕМЫ
*           А) ОПРЕДЕЛЕНИЕ ЗАПИСИ СОТРУДНИКА
            ПУСТЬ' СОТРУДНИК: СОТР;
*           Б) ОПРЕДЕЛЕНИЕ МНОЖЕСТВ
              СОТРУДНКИ,МОЛОДЫЕ: МНОЖ ЗАПИСЬ=СОТРУД-
              НИК;
*           В) ВВОД МНОЖЕСТВА
              В:ВВОД СОТРУДНКИ*, ИЗ='КИБР';
```

```
*              Г) ОПРЕДЕЛЕНИЕ ПОДМНОЖЕСТВА
                  П: ПОДМНОЖ МНОЖ = СОТРУДНКИ, ЕСТЬ=МОЛО-
                  ДЫЕ;
                ПРИСВ'П.УСЛОВ: =СОТРУДНИК.ВОЗРАСТ МР'28;
*              Д) ВЫВОД ПОДМНОЖЕСТВА
                     РЕЗУЛЬТАТ: ВЫВОД МОЛОДЫЕ, ФОРМАТ 'СО-
                     ТРУДНИК';
* ПОСТАНОВКА ЗАДАЧИ
                ДЕЙСТВИЯ' НА' ПРИМ2 ВЫЧИСЛИТЬ'
                РЕЗУЛЬТАТ.ПЕЧАТЬ;
 КОНЕЦ'; +++
```

В примере 3 описан запрос для нахождения данных конкретного сотрудника (рис. 53).

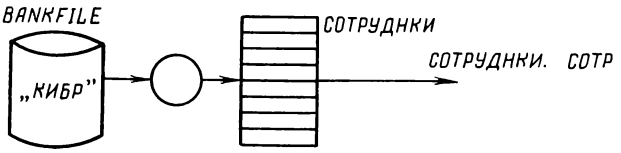


Рис. 53. Задача 3

Предложение ПРОГРАММА' идентифицирует запрос с именем ПРИМ3.

Следующее предложение СОТР:СОТР определяет, что новый объект СОТР (левый в конструкции) получает структуру объекта СОТР (правая часть) из библиотеки описаний.

В следующем предложении определяется новая переменная ФИО длиной 9 слов. Далее этой переменной присваивается значение 'СИДОРОВ ИВАН ПЕТРОВИЧ'.

В следующем предложении указывается, что эта переменная ФИО является ключом для выборки элементов (сотрудников) из множества сотрудников.

Предложением ВВОД вводится множество сотрудников из набора данных 'КИБР'.

*— РАЗРЕШЕНО ОПУСКАТЬ КЛЮЧЕВЫЕ СЛОВА ПРЕДЛОЖЕНИЯ.

В предложении ДЕЙСТВИЯ' НА' определяется задача: на модели запроса ПРИМ3 вычислить данные сотрудника.

Пример 3

```
* ЗАПРОС НА ВЫДЕЛЕНИЕ ДАННЫХ КОНКРЕТНОГО СОТРУДНИКА
            ПРОГРАММА' ПРИМ3;
* ПОСТРОЕНИЕ ПОДСХЕМЫ
*     А) ОПРЕДЕЛЕНИЕ ЗАПИСИ СОТРУДНИКА
          ПУСТЬ' СОТР:СОТР;
*           Б) ОПРЕДЕЛЕНИЕ ПЕРЕМЕННОГО ФИО
                ФИО: СТРОКА'9;
          ПРИСВ' ФИО: ='СИДОРОВ ИВАН ПЕТРОВИЧ';
*           В) ОПРЕДЕЛЕНИЕ МНОЖЕСТВА СОТРУДНИКОВ
                СОТРУДНКИ: МНОЖ ЗАПИСЬ=СОТР, КЛЮЧ=ФИО;
*           Г) ВВОД МНОЖЕСТВА
                В: ВВОД СОТРУДНКИ, ИЗ ='КИБР';
```

```
* ПОСТАНОВКА ЗАДАЧИ
            ДЕЙСТВИЯ' НА' ПРИМ3 ВЫЧИСЛИТЬ' СОТРУДНКИ.СОТР;
  КОНЕЦ'; +++
```

Пример 4 содержит запрос на использование понятия ИТОГИ. Запрос состоит в том, что требуется найти итоговый подсчет сотрудников по разным должностям в разрезе отделов учреждения (рис. 54).

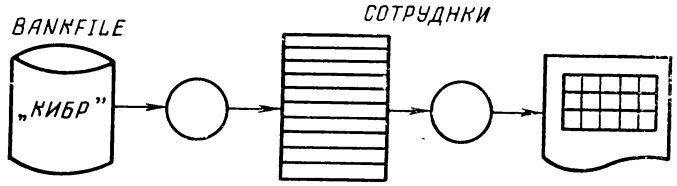


Рис. 54. Задача 4

Предложение ПРОГРАММА' идентифицирует запрос с именем ПРИМ4.

Конструкция СОТР:СОТР определяет, что новый объект СОТР (левый в конструкции) получает структуру объекта СОТР (правая часть) из библиотеки описаний, куда эта структура заранее записана.

В следующем предложении определено множество СОТРУДНКИ, состоящее из объектов СОТР.

Предложением ВВОД вводится множество сотрудников СОТРУДНКИ из набора данных 'КИБР'.

В предложении ИТОГИ определяется итоговый подсчет сотрудников СОТРУДНКИ по разным должностям СОТР.ДОЛЖНОСТЬ в разрезе отделов СОТР.ОТДЕЛ.

В предложении ДЕЙСТВИЯ' НА' определяется задача: на модели запроса ПРИМ4 вычислить результат.

Пример 4

```
* ЗАПРОС НА ВЫДЕЛЕНИЕ ИТОГОВОГО ПОДСЧЕТА
            ПРОГРАММА' ПРИМ4;
* ПОСТРОЕНИЕ ПОДСХЕМЫ
*               А) ОПРЕДЕЛЕНИЕ ЗАПИСИ СОТРУДНИКА
                      ПУСТЬ' СОТР:СОТР;
*               Б) ОПРЕДЕЛЕНИЕ МНОЖЕСТВА СОТРУДНИКОВ
                      СОТРУДНКИ: МНОЖ ЗАПИСЬ=СОТР;
*               В) ВВОД МНОЖЕСТВА
                      В:ВВОД СОТРУДНКИ, ИЗ='КИБР';
*               Г) ОПРЕДЕЛЕНИЕ ИТОГОВОГО ПОДСЧЕТА
                      И:ИТОГИ МНОЖ=СОТРУДНКИ, ПО=СОТР.ДОЛЖ-
                        НОСТЬ, В=СОТР.ОТДЕЛ;
* ПОСТАНОВКА ЗАДАЧИ
                      ДЕЙСТВИЯ' НА' ПРИМ4 ВЫЧИСЛИТЬ' И.РЕЗ;
  КОНЕЦ'; +++
```

Язык манипулирования данными (ЯМД) предназначен для изменения состояния баз данных. В состав этого языка входят все необходимые операции манипулирования, реализующие

ввод, замену и удаление объектов;
создание и удаление набора данных;
ввод и вывод множеств объектов (входят также в ЯЗ).

ЯМД является расширением языка УТОПИСТ и содержит следующие новые понятия:

ввод объекта — ВКЛЮЧИТЬ;
замена объекта — ЗАМЕНИТЬ;
удаление объекта — УДАЛИТЬ;
создание набора данных — СОЗДАТЬ;
удаление набора данных — УНИЧТОЖИТЬ.

Как на языке ЯЗ, так и здесь каждое предложение языка манипулирования данными имеет свойственные характеристики — параметры. Форма этих предложений следующая:

```
ввод—объекта : : =
ид: ВКЛЮЧИТЬ ЭЛЕМ = имя—объекта,
                 В = имя—множества
               [,РЕЗ = имя—логического] ;
замена—объекта : : =
ид: ЗАМЕНИТЬ ЭЛЕМ = имя—объекта,
                 В = имя—множества
               [,РЕЗ = имя—логического] ;
удаление—объекта : : =
ид: УДАЛИТЬ ЭЛЕМ = имя—объекта,
                ИЗ = имя—множества
               [,РЕЗ = имя—логического] ;
создание—множества : : =
ид: СОЗДАТЬ МНОЖ = имя—множества,
                 В = имя—описания—набора,
            ФОРМАТ = имя—формата
              [,РЕЗ = имя—логического] ;
```

где соответственно должны быть предварительно определены

```
описание—набора : : =
ид: НАБОР имя—текста, имя—целого, имя—целого, имя—текста, имя—целого;
```

и

```
формат : : =
ид: ФОРМАТ 'формат—фортрана';
формат—фортрана : : =
```

Формат описывается как в явном утверждении FORMAT в фортрановской программе.

Пример

```
Ф:ФОРМАТ '(1X, 2F5,2, 2A4, I4)';
    удаление—множества : : =
    ид: УНИЧТОЖАТЬ МНОЖ = имя—множества,
            НАБОР='имя—набора'
            [,РЕЗ ='имя—логического] ;
```

Из языка УТОПИСТ используется следующая конструкция для присваивания значений объектам:

$$\text{присваивание} : : = \text{ЗАДАНО' имя—объекта} = \left\{ \begin{array}{l} \text{простое значение} \\ \text{(значение, ...)} \end{array} \right\}$$

Пример присваивания значений объекту АВТО:

ЗАДАНО' АВТО=('МОС44—11','ГАЗ—24','ЧЕРНЫЙ',1972,'СИДОРОВ',2700);

Эта же конструкция применима для присваивания значений объектам типа «описание — набора» и «формат».

Примеры

```
С: СОЗДАТЬ МНОЖ = МАШИНЫ;
ЗАДАНО' С.В= ('ГАИ',54, 300, 'ММХ2',2);
ЗАДАНО' С.ФОРМАТ = ('(1Х,2А4, 2А4, 2А4, 14,2А4,16)');
```

Текст задачи для манипулирования данными на языке УТОПИСТ имеет следующую форму:

```
манипулирование : : =
ПРОГРАММА' ид;
                подсхема базы данных
                ввод — объекта
                замена — объекта
                удаление — объекта
                создание — множества
                удаление — множества
                ввод — множества
                вывод — множества
                присваивание
ДЕЙСТВИЯ'
   НА' имя ВЫЧИСЛИТЬ' имя,...;
КОНЕЦ'; +++
```

Пример

```
* ЗАДАЧА : ИЗМЕНИТЬ СОСТОЯНИЕ БАЗЫ ДАННЫХ АВТОМАШИН
        ПРОГРАММА' ПРИМЕР;
* ОПИСАНИЕ ПОДСХЕМЫ
        ПУСТЬ' АВТО:МАШИНА;
           МАШИНЫ: МНОЖ ЗАПИСЬ= АВТО;
* МАНИПУЛЯЦИИ
             В:ВВОД МАШИНЫ, ИЗ' = 'ГАИ';
             А1: ВКЛЮЧИТЬ АВТО, В = МАШИНЫ;
             А2: ЗАМЕНИТЬ АВТО, В = МАШИНЫ;
     ЗАДАНО'А1.АВТО=('ММХ11—49','ЗА3698','БЕЛЫЙ',1979,'БЛИК',270);
     ЗАДАНО'А2.АВТО=('МОС44—11','ВА32101','КРАСНЫЙ',1973,'УМОВ',
     70000);
* ПОСТАНОВКА ЗАДАЧИ
  ДЕЙСТВИЯ' НА' ПРИМЕР ВЫЧИСЛИТЬ' А1.РЕЗ,А2.РЕЗ;
  КОНЕЦ'; +++
```

Каждому приведенному выше понятию ЯМД соответствуют своя семантика и предложение некоторой формы, где это понятие может быть употреблено.

Семантика предложений ЯМД следующая:

Х: ВКЛЮЧИТЬ ЭЛЕМ=У, В=Т;* определяет, что элемент У вводится в множество Т.

Х: ЗАМЕНИТЬ ЭЛЕМ=У, В=Т; определяет, что элемент У заменяется в множестве Т.

---

* При описании семантики предложений не используются их компоненты РЕЗ. Они определяют совершение или несовершение операции манипулирования и на них ссылаются только в предложении ДЕЙСТВИЯ'НА' для вычисления результата.

X: УДАЛИТЬ ЭЛЕМ=У, ИЗ=Т; определяет, что элемент У удаляется из множества Т.

X: СОЗДАТЬ МНОЖ=У, В=Т, ФОРМАТ=С; определяет, что создается пустое множество У, которому соответствует описание данных Т и формат элемента множества С.

Описание набора данных Т состоит в свою очередь из компонентов, которые характеризуют физическую организацию набора данных. Например, для настоящей физической организации баз данных это описание состоит из 5 компонентов:

Т.КОМП1 — имя набора данных длиной 4 символа;

Т.КОМП2 — максимальное число элементов множества;

Т.КОМП3 — длина элементов множества (в словах);

Т.КОМП4 — ключ набора данных длиной 4 символа;

Т.КОМП5 — длина ключа элемента множества (в словах).

X: УНИЧТОЖИТЬ МНОЖ=У, НАБОР='Т'; определяет, что из базы данных удаляется набор данных Т, содержащий множество У.

5.2. ОБЩАЯ СТРУКТУРА СИСТЕМЫ

Пакет прикладных программ СУБД DABU работает под управлением операционной системы ОС ЕС, используя для трансляции описаний данных и запросов системные программы ПРИЗ (рис. 55). Для работы с базами данных предназначены входные языки пакета СУБД.

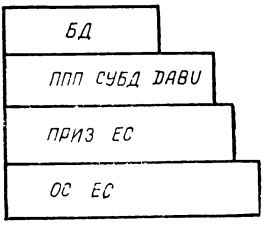


Рис. 55. Архитектура системы

При работе с ограниченным числом баз данных все библиотеки программ и базы данных можно поместить на один системный диск системы ПРИЗ ЕС. При больших объемах информации желательно выделить для баз данных отдельные тома. Других технических устройств, кроме перечисленных, для работы с системой ПРИЗ ЕС не требуется. Необходимый объем оперативной памяти для работы — 140 К.

Программные компоненты. В состав системы DABU входят стандартные программные компоненты системы ПРИЗ (транслятор, генератор, ассемблер и редактор связей) и программы пакета СУБД. Программы СУБД реализованы в виде двух наборов подпрограмм (рис. 56). Первый из них выполняет функции, описываемые понятиями языков ЯОД, ЯЗ и ЯМД, а второй предназначен для обслуживания наборов данных на магнитных дисках.

Базы данных (BANKFILE или другие наборы данных) и описание схем базы данных (PRIZ. MODLIB) хранятся в отдельных наборах данных на магнитных дисках.

Функциональная схема системы показана на рис. 56.

Кратко работа системы состоит в следующем. С экрана дисплея или с перфокарт вводится текст описания схемы БД или текст

запроса. Схемы БД транслируются и записываются в дисковый файл PRIZ. MODLIB (PRIZ. MODLIB является, по существу, семантической памятью, обслуживаемой системой ПРИЗ). В этом файле располагаются понятия языка запросов и языка манипулирования данными.

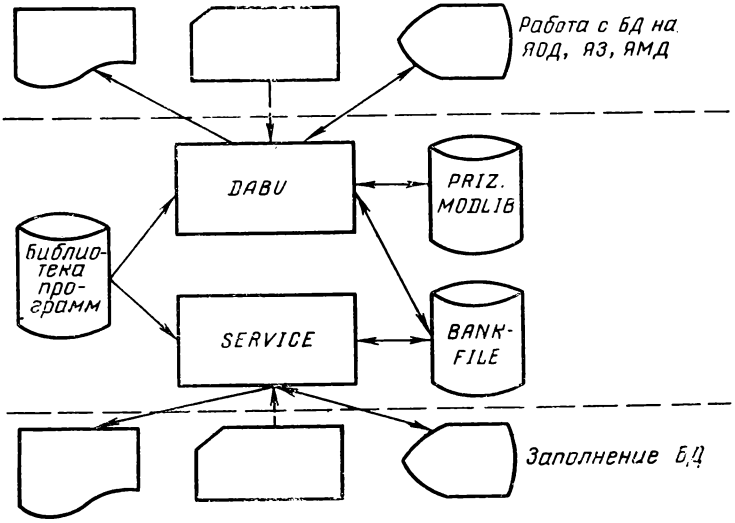


Рис. 56. Функциональная схема системы

Запрос к данным строится из описаний схем БД и предложений ЯЗ. По описанию запроса система синтезирует поисковую программу, работает ядро системы DABU. Базы данных располагаются в дисковом файле BANKFILE. Для их обслуживания применяется ЯМД или комплекс обслуживающих программ SERVICE. При заполнении и обслуживании системы предпочтительнее работать с комплексом программ SERVICE, который создан для повышения удобств и быстродействия обслуживания.

Фазы работы. При работе с пакетом программ СУБД используются стандартные фазы работы системы ПРИЗ:

для описания схем баз данных используется только фаза трансляции, работает расширенный транслятор УТОПИСТа;

для выполнения запросов и манипулирования данными применяются все фазы работы системы ПРИЗ, работают последовательно расширенный транслятор УТОПИСТа, генератор, редактор связей и выполнения программы;

для обслуживания баз данных предназначена только фаза выполнения комплекса программ SERVICE.

Структура базы данных. В системе СУБД DABU логические и физические уровни базы данных почти полностью отделены. Опи-

санные на логическом уровне схемы базы данных имели уже три варианта реализации физического уровня (мониторов данных).

В первоначальной стадии разработки системы СУБД DABU был использован монитор данных, работающий с наборами данных в оперативной памяти. При втором варианте монитор работает с библиотечными наборами данных, которые содержали данные в текстовом формате, и при последнем варианте монитор работает с одним большим набором данных прямого доступа BANKFILE (конечно, можно работать и с несколькими наборами данных с разными именами). BANKFILE содержит все наборы данных конкретной базы (банка) данных и каталог наборов. Эти наборы можно обрабатывать как прямым, так и последовательным методами доступа.

При наличии у пользователя монитора данных, работающего эффективнее, чем перечисленные мониторы, его можно, не нарушая логической организации, включить в систему DABU.

5.3. РАБОТА С БАЗАМИ ДАННЫХ

Здесь описываются работы, проводимые для проектирования конкретного банка данных, даются указания для обслуживания баз данных и для оформления запросов.

Технология построения баз данных охватывает очень широкий круг вопросов, начиная с описания записей и заканчивая наличием необходимого числа терминалов для создаваемого банка данных. Описанная дальше технология охватывает только вопросы, определяющие последовательность и содержание проводимых в системе DABU работ по созданию программного обеспечения конкретного банка данных. Работы в этих целях рекомендуется проводить в следующей последовательности:

определение задачи, выбор структур данных;
описание выбранных структур данных на языке ЯОД;
разработка входных форм, ввод и корректировка данных;
составление запросов.

Определение задачи осуществляется в целях выяснения пригодности системы DABU для описания информационной модели предметной области пользователя. На этом этапе определяется, какие записи и какие наборы данных требуются для описания предметной области, а также их взаимосвязи. Сравнение полученных структур с возможными структурами в системе DABU дает ответ на вопросы, можно ли систему DABU использовать прямо, требуются ли переделки или систему использовать невозможно.

Описание структур данных выполняется по приведенным в описании ЯОД (см. п. 5.1) требованиям.

Управляющие предложения для трансляции этих структур (схемы базы данных) в пакетном режиме приведены ниже:

```
/ /JOB1  JOB  MSGLEVEL=(2,0)
/ /      EXEC PRIZC
/ /TRN.SYSIN DD *
```

```
                ПРОГРАММА' имя;
                КОНЕЦ'; +++        } схема базы данных
/ /
```

Для работы в диалоговом режиме необходимо использовать следующие управляющие предложения:

```
/ /JOB2  JOB  MSGLEVEL=(2,0)
/ /      EXEC PRIZD,CRT=NNN
/ /
```

Номер устройства NNN должен соответствовать реальному номеру дисплея, например 063.

Возможности диалоговой работы на ЯОД рассмотрим на примере схемы БД, описывающей объект «сотрудник» и множество «сотрудники».

Пример

```
ПРОГРАММА' СХЕМА1;
ПУСТЬ' СОТРУДНИК : (ФИО: (ФАМИЛИЯ: СТР'3;
                          ИМЯ:      СТР'3;
                          ОТЧЕСТВО: СТР'3);
                    ВОЗРАСТ: ЦЕЛ';
                    ПОЛ:  ЛОГ'
                    ОТДЕЛ: СТР' 2;
                    ОКЛАД: ВЕЩ';
                    АДРЕС: СТР'5;);
      СОТРУДНИКИ: МНОЖ ЗАПИСЬ= СОТРУДНИК;
ВВЕСТИ' СХЕМА1; +++
КОНЕЦ'; +++
```

В этом примере используется директива ВВЕСТИ', которая включает новую схему БД в файл PRIZ. MODLIB. Директива ВВЕСТИ' предназначена также для дополнения схем баз данных путем включения в них новых компонентов. Например, ввод в рассматриваемый объект «сотрудник» нового параметра «год рождения» описывается так:

```
ПУСТЬ' ГОДРОЖД: ЦЕЛЫЙ';
ВВЕСТИ' ГОДРОЖД В' СХЕМА1.СОТРУДНИК;
```

Кроме директивы ВВЕСТИ', для работы с файлом PRIZ. MODLIB используются директивы УДАЛИТЬ' и ВЫПОЛНИТЬ' HELP. Директива УДАЛИТЬ' служит для удаления из PRIZ. MODLIB ненужных схем или их частей. Например, удаление из объекта «сотрудники» компоненты «возраст» описывается так:

```
УДАЛИТЬ' СХЕМА1.СОТРУДНИК.ВОЗРАСТ;
```

Директивой ВЫПОЛНИТЬ' HELP выполняется программа-информатор HELP, которая позволяет ознакомиться с файлом PRIZ. MODLIB путем вывода на экран его содержимого.

Программа HELP понимает следующие команды:

INTO X — выдать описание объекта (понятие или схема БД) X;

NEXT — выдать описание следующего объекта;

UP — выдать описание объекта ближайшего верхнего уровня;
WHAT IS X — выдать значение объекта X;
END — окончить работу.

По окончании работы программы-информатора можно продолжать работу с транслятором для построения новых схем и запросов (при работе на ЯЗ или на ЯМД).

Ввод данных. Для каждого типа записи целесообразно разработать свою входную форму в виде бланка записи (рис. 57). Такие бланки должны быть удобными для заполнения, перфорации и ввода с них данных через экран дисплея.

Для ввода данных в пакетном режиме можно применить процедуру обслуживания, которая управляет комплексом программ SERVICE, или язык манипулирования данными. Процедура обслуживания обеспечивает обслуживание баз данных в полном объеме.

Обращение к процедуре приведено ниже:

```
/ /JOB3  JOB  MSGLEVEL=(2,0)
/ /      EXEC SERVICE,PARM.P1= 'XXXX'
/ /P1.SYSIN DD *
              <карты данных >
/ /
```

Параметры поля «xxxx» могут иметь следующие значения:

INIT,LOAD,ADD,REPL,DELE,STAT,CATA,

где INIT — вызывает инициализацию набора данных BANKFILE на диске;
LOAD — используется для начальной загрузки конкретного набора данных;
ADD — применяется для дополнения новых записей;
DELE — используется для удаления записей из набора данных;
REPL — предназначено для замены записей в наборе данных;
STAT — используется для распечатки состояния конкретного набора данных;
CATA — применяется для распечатки каталога набора данных BANKFILE.

Первый раз эта процедура выполняется для создания пустой базы данных путем кодирования параметра процедуры INIT. Далее осуществляются загрузка данных путем кодирования параметра LOAD и исправление неверных записей с помощью параметров процедуры REPL,DELE,ADD,STAT и CATA.

Для работы на ЯМД (см. п. 5.1) должен быть предварительно инициализирован набор данных BANKFILE (инициализация проводится средствами программы SERVICE).

Предполагается, что основное обслуживание наборов данных ведется с помощью программы SERVICE, а на ЯМД вводятся в базу данных только мелкие исправления. Предложения ЯМД можно задавать вперемежку с предложениями ЯЗ, задачи на обоих языках имеют общие управляющие карты, которые приведены ниже.

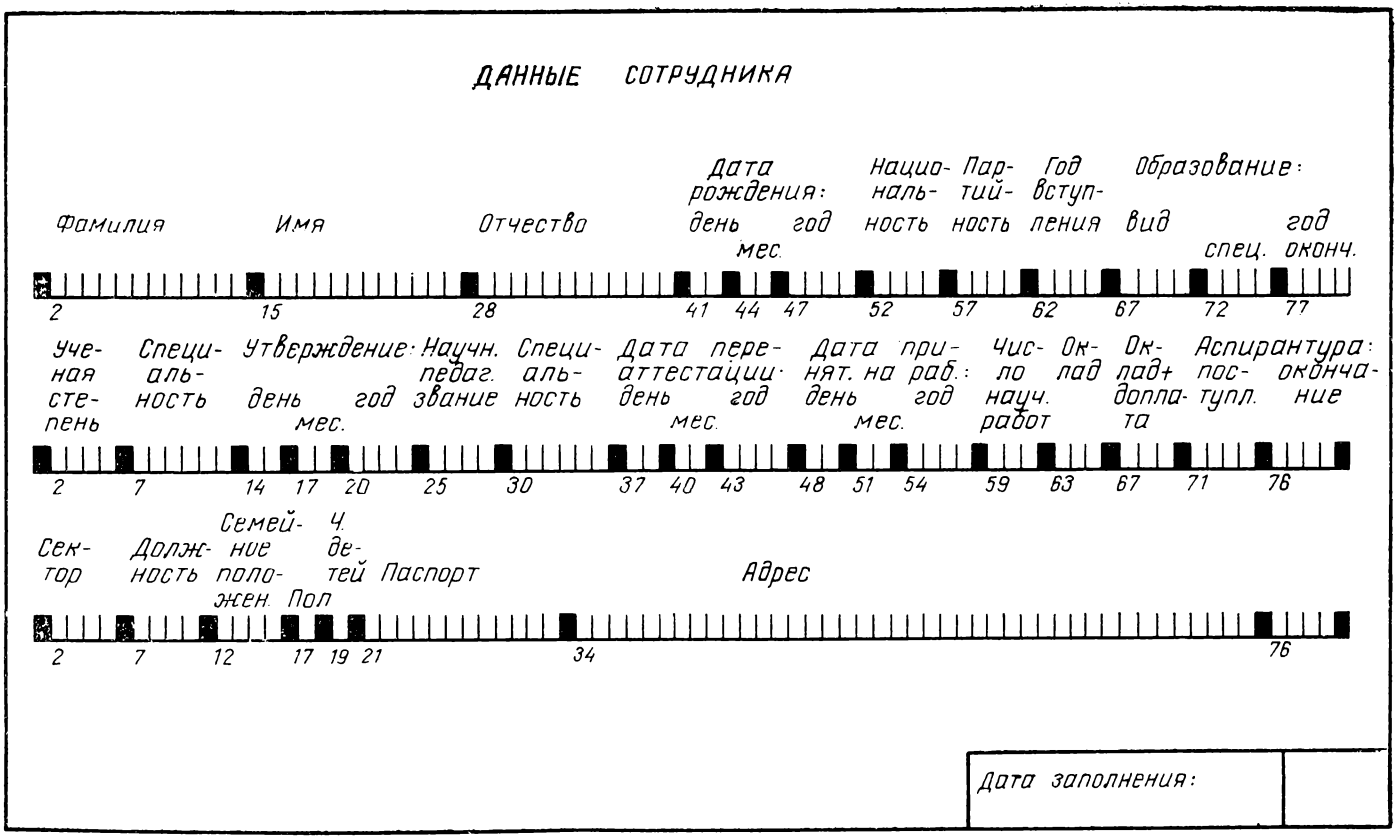


Рис. 57. Бланк «Данные сотрудника»

Для ввода данных в диалоговом режиме также применяются две вышеприведенные возможности работы. Обращение к программе обслуживания SERVICE осуществляется при работе на языках ЯОД, ЯЗ, ЯМД директивой EXEC' SERVICE. После этой директивы программа SERVICE сама управляет действиями пользователя. На экран дисплея выдается сообщение «DATA BASE FILE?». В ответ на это пользователь должен ввести идентификатор конкретного набора данных. Далее выдается сообщение «DB COMMANDS: FIRST,KEY=, HELP,END», с которым программа предлагает пользователю 4 варианта работы:

при ответе FIRST выдается из набора данных первая запись;

при ответе KEY=XXXX, где XXXX ключ записи, выдается из набора данных запись, имеющая данный ключ;

при ответе HELP программа предлагает ряд возможностей, облегчающих поиск записей;

при ответе END программа возвращает управление расширенному транслятору УТОПИСТа.

После выполнения одного из трех первых вариантов система выдает следующие сообщения:
«DB COMMANDS:FIRST,NEXT,KEY=,WRITE,HELP,END».

При этом добавились два новых варианта работы:

при ответе NEXT выдается из набора данных следующая запись;

при ответе WRITE можно исправлять на экране неверную запись и вводить исправленную запись в базу данных.

Управляющие предложения для диалоговой работы на ЯМД точно такие, как и при диалоговой работе на ЯЗ.

Составление запросов. К составлению запросов можно приступить после того, как в библиотеку PRIZ.MODLIB введены структуры данных (схемы БД) и в BANKFILE загружены наборы данных. Запросы составляются на языке ЯЗ. Управляющие предложения для трансляции и выполнения запросов в пакетном режиме приведены ниже:

```
/ /JOB4 JOB  MSGLEVEL=(2,0)
/ /      EXEC  PRIZCLG,LDLIB=BANKLOAD
/ /TRN.SYSIN  DD  *
                ПРОГРАММА' имя;   ┐
                    .             │
                    .             │ текст запроса
                    .             │
                КОНЕЦ'; +++       ┘
/ /GO.FT19F001 DD    DSN=BANKFILE,DISP=OLD,
/ /            VOL= SER=PRZLIB,UNIT=SYSDA
/ /
```

Для работы в диалоговом режиме можно использовать следующие управляющие предложения:

```
/ /JOB5  JOB    MSGLEVEL=(2,0)
/ /      EXEC     PRIZD,CRT=NNN
/ /FT19F001  DD    DSN=BANKFILE,DISP=OLD,
/ /            VOL=SER=PRZLIB,UNIT=SYSDA
/ /
```

Приведенный выше порядок работ с базами данных не обязателен. Имеется возможность трансляции запросов без выполнения, т. е. отладки запросов, не имея на диске наборов данных. В данном случае надо удалить из колоды управляющих карт карту с описанием набора данных BANKFILE.

Как было показано, все языки баз данных системы DABU являются расширениями языка УТОПИСТ. Это позволяет транслировать тексты задач на этих языках одним транслятором УТОПИСТа. Из этого следует, что управляющие карты для трансляции и выполнения запросов можно применить и для решения задач на ЯОД и ЯМД.

5.4. СОСТОЯНИЕ И ГЕНЕРАЦИЯ СУБД

Состояние и генерация системы управления базами данных — это расположение компонентов системы на машинных носителях информации, перевод системы в рабочее состояние и проверка правильности работы системы.

Состояние системы. В состав СУБД DABU входят следующие программные компоненты:

библиотеки программных средств системы ПРИЗ ЕС;

библиотека загрузочных модулей системы DABU.

Система DABU распространяется на дистрибутивной магнитной ленте PRIZDS, содержащей

управляющие предложения генерации BANKINIT;

библиотеку загрузочных модулей BANKLOAD;

текст инструкции по использованию дистрибутивной ленты BANKINST.

Библиотека BANKLOAD, содержащая программные секции системы, является загрузочной копией библиотечного набора данных, полученного путем копирования библиотеки с устройства прямого доступа на магнитную ленту. Копирование выполняется утилитой IEHMOVE.

Библиотека BANKLOAD копируется на магнитный диск PRZLIB (магнитный диск системы ПРИЗ ЕС). Для копирования библиотеки на этом диске требуется 10 цилиндров свободной памяти.

Генерация системы. Рабочий экземпляр стандартной версии СУБД DABU образуется во время процесса, называемого генерацией системы. Генерация системы представляет собой последовательность выполнения нескольких шагов задания ОС ЕС, управляющие предложения которого находятся на дистрибутивной магнитной ленте PRIZDS.

Генерацию системы DABU можно выполнить только при наличии стандартной версии системы ПРИЗ ЕС. Генерация системы DABU осуществляется по управляющим директивам процедуры генерации. Процедура генерации BANKINIT является девятым набором данных на дистрибутивной магнитной ленте PRIZDS (рис. 58). Она содержит пять последовательно выполняемых шагов задания ОС ЕС (рис. 59).

Шаги задания генерации следующие:

копирование библиотеки BANKLOAD системы DABU с магнитной ленты PRIZDS на магнитный диск PRZLIB;

ввод понятий языка манипулирования данными в библиотеку моделей (системы ПРИЗ ЕС) PRIZ.MODLIB;

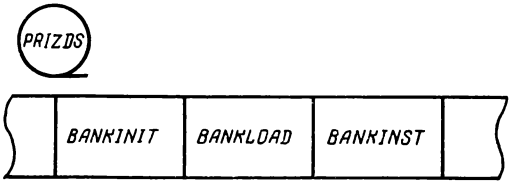


Рис. 58. Содержимое ленты PRIZDS

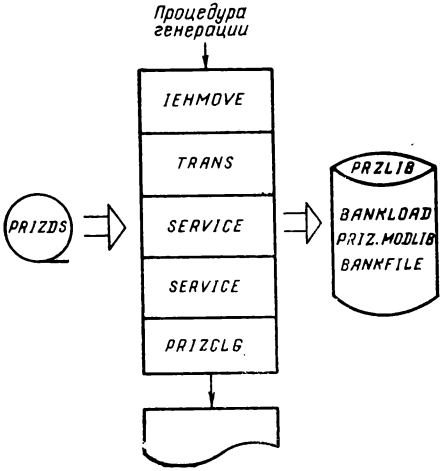


Рис. 59. Генерация системы

создание на магнитном диске PRZLIB пустой базы данных BANKFILE;

загрузка базы данных BANKFILE данными для тестовой задачи;

выполнение тестовой задачи.

От оператора требуется выполнение следующих действий:

монтирование магнитного диска PRZLIB и магнитной ленты PRIZDS;

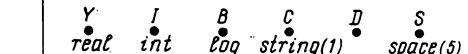
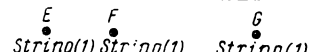
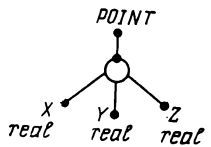
освобождение очереди заданий, остановка процедуры системного ввода RDR и выдача команды

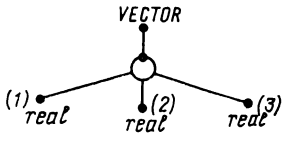
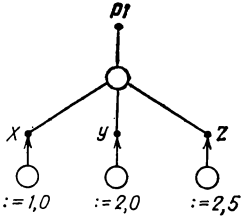
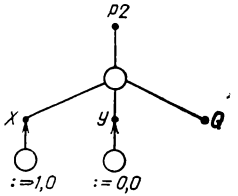
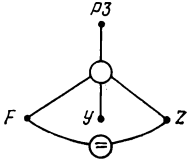
```
S  RDR,UNIT,PRIZDS,DSN=BANKINIT,LABEL=9
```

UNIT — это адрес устройства, на котором установлена дистрибутивная лента, например 280.

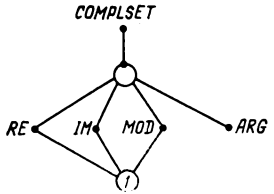
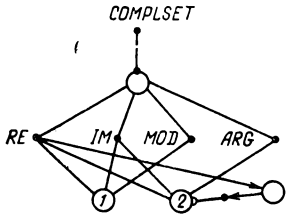
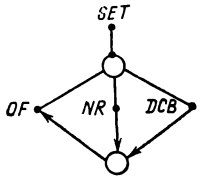
Приложение 1

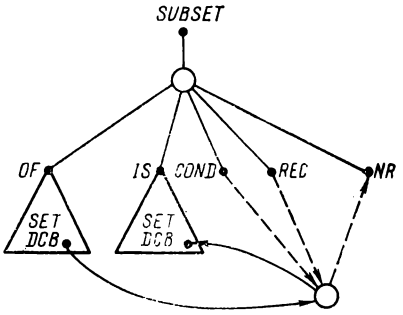
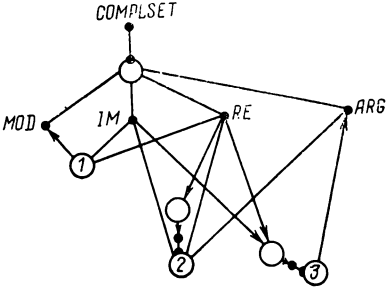
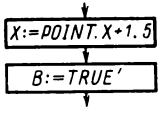
Примеры конструкций языка УТОПИСТ

№ п/п	Наименование	Текст на языке УТОПИСТ	Семантическая модель
1	2	3	4
1	Описание-компонента с элементарными типами	Описания X : R E A L ' ;	
2	Последовательность описаний-компонент	Y:REAL';I:INT';B:LOG'; C:STRING' 10;D:UNDEFINED' ;S:SPACE'5;	
3	Сокращенная запись однотипных описаний-компонент	E,F,G:STRING';	
4	Составной объект	POINT: (X,Y,Z:REAL' ;) ;	
5	Копирование объекта	A : POINT;	

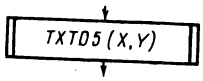
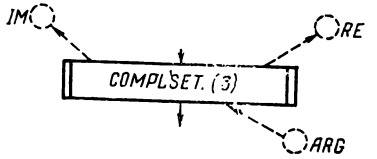
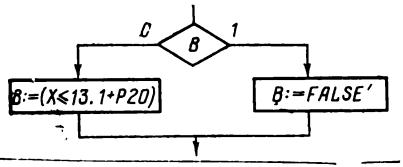
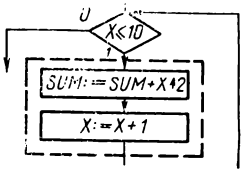
6	Ряд	VEKTOR : ROW (1 . . . 3) : REAL'	
7	Переделки	P1: POINT␣1.0,2.0,2.5;	
		P2: POINT␣X=1.0,Y :=0.0,Q;	
		P3: POINT␣F,F=P3.Z;	

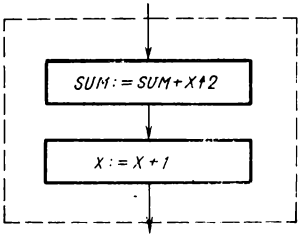
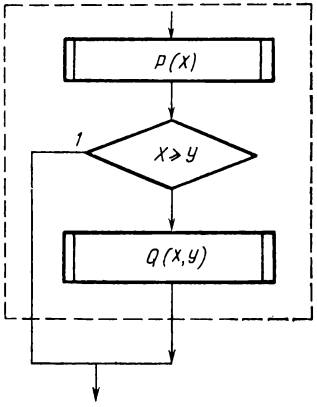
Продолжение

№ п/п	Наименование	Текст на языке УТОПИСТ	Семантическая модель
1	2	3	4
8	Уравнение	COMPLSET : (RE,IM,MOD,ARG : REAL'; EQN'MOD∗∗2=RE∗∗2 + IM∗∗2);	
9	Условное отношение	COMPLSET: (RE,IM,MOD,ARG:REAL'; EQN'MOD∗∗2=RE∗∗2+IM ∗∗2; IF'RE␣NE'0 ␣THEN' EQN'TG'ARG=IM/RE);	
10	Отношение, выраженное программой	SET : (OF:UNDEFINED'; NR:INT'; DCB:SPACE'20; MODULE'A␣IN'DCB,NR ␣OUT'OF;);	

11	Отношение, выраженное программой, содержащей подзадачи	SUBSET:(OF,IS:SET; COND:LOG'; REC:UNDEFINED'; NR:INT'; MODULE'B IN' OF.DCB OUT'IS.DCB ARG'NR RES'COND,REC;);	
12	Присваивание	COMPLSET:(RE,IM,MOD,ARG:REAL' EQN'MOD**2=RE**2+IM **2; IF'RE␣'NE'0 ␣THEN' EQN'TG'ARG=IM/RE; IF'RE␣EQ'0 ␣AND'NOT'(IM ␣LE'0) ␣THEN' ASSING'ARG:=3.1415026525/2;);	
		Действия	
1	Присваивание и последовательность операторов	X:=POINT.X+1.5; B:=TRUE';	

Продолжение

№ п/п	Наименование	Текст на языке УТОПИСТ	Семантическая модель
1	2	3	4
2	Вызов	CALL'TXT05(X,Y);	
3	Оператор постановки задачи	ON'COMPLSET.(3) ␣COMPUTE'ARG ␣FROM'RE,IM;	
4	Условный оператор	IF'B␣THEN'B:=FALSE'; ELSE'B:=X ␣LE'13.1+P20;	
5	Оператор цикла	WHILE'X␣ LE'10␣DO' (SUM:=SUM+X**2; X:=X+1;);	

6	Составной оператор	(SUM : = SUM+X**2;X : = X+1;);	
7	Метка и оператор возврата	M: (CALL'P(X) ; IF'X␣GE'Y␣THEN'EXIT'M; CALL'Q(X,Y););	

Лексический уровень языка УТОПИСТ

идентификатор:

буква [{ буква / цифра }] ...

целое:

цифра ...

число:

целое. [целое]

строка:

'любой—кроме—кавычки ...'

знакосочетание:

{ : = } присваивание
{ * * } возведение в степень

символы:

{ . / , / : / + / — / (/) /// * / = / ; / ' }

цифра:

{ 0/ 1/ 2/ 3/ 4/ 5/ 6/ 7/ 8/ 9}

буква:

{ А/ Б/ В/ Г/ Д/ Е/ Ж/ З/ И/ Й/ К/ Л/ М/ Н/ О/ П/ Р/ С/ Т/ У/
Ф/ Х/ Ч/ Ш/ Щ/ Ь/ Ю/ Я/ Ц/ #/ —}

ключевое—слово:

{ ПРОГРАММА' | ПУСТЬ' | ДЕЙСТВИЯ' | ПОДПРОГРАММА' |
КОНЕЦ' | УРАВ' | ПРИСВ' | ЗАДАНО' | МАКРО' | МНОЖ' |
ЦЕЛЫЙ' | ВЕЩ' | ЛОГ' | ПАМЯТЬ' | СТРОКА' | ДЛИНА' |
НЕОПР' | МОДУЛЬ' | ТИП' | РАНГ' | КОПИЯ' | ВХ' | ВЫХ' |
СЛ' | СИЛ' | ОПР' | ВВХ' | ВВЫХ' | ВСЕ' | ИСТИНА' | ЛОЖЬ' |
SIN' | COS' | TG' | ARCTG' | ABS' | LN' | EXP' |
И' | ИЛИ' | СР' | НЕ' | РАВ' | НР' | МР' | БР' | НА' |
ВЫЧИСЛИТЬ' | ИЗ' | ПОКА' | ЦИКЛ' | ЕСЛИ' | ТО' | ИНАЧЕ' |
ВЫЗОВ' | ЗНАЧЕНИЕ' | ПО' | ВВЕСТИ' | В' | УДАЛИТЬ' |
ВЫПОЛНИТЬ' }

Примечания

1. На уровень синтаксиса пробелы не выносятся, их роль — разделение лексем там, где они не разделяются другими способами (например, между двумя именами).
2. Понятие комментария в языке УТОПИСТ отсутствует. Комментарии вводятся на отдельных перфокартах или на отдельных строках, распознаваемых по символу «*» в первой позиции.

Соответствие английской и русской лексики

Английская лексика	Русская лексика	Английская лексика	Русская лексика
ABS	ABS	LE	МР
ACTIONS	ДЕЙСТВИЯ	LENGTH	ДЛИНА
ALL	ВСЕ	LET	ПУСТЬ
AND	И		
ARCTG	ARCTG	LN	LN
ARG	ВВХ		
ASSIGN	ПРИСВ	LOG	ЛОГ
CALL	ВЫЗОВ		
COMPUTE	ВЫЧИСЛИТЬ	MCR	МАКРО
COND	ОПР	MODULE	МОДУЛЬ
COPY	КОПИЯ'	NE	НР
COS	COS	NOT	НЕ
CV	СР	ON	НА
DELETE	УДАЛИТЬ	OR	ИЛИ
DO	ЦИКЛ	OUT	ВЫХ
ELSE	ИНАЧЕ	PROBL	ПРОГРАММА
END	КОНЕЦ	PROC	ПРОГ
EQ	РАВ	RANK	РАНГ
EQN	УРАВ	REAL	ВЕЩ
EXEC	ВЫПОЛНИТЬ	RES	ВВЫХ
EXIT	ИЗ	ROW	МНОЖ
EXP	EXP	SIN	SIN
FALSE	ЛОЖЬ	SPACE	ПАМЯТЬ
FROM	ПО	STRING	СТРОКА
GE	БР	SUBROUTINE	ПОДПРОГРАММА
GIVEN	ЗАДАНО		
		TG	TG
IF	ЕСЛИ	THEN	ТО
IN	ВХ		
INSERT	ВВЕСТИ	TRUE	ИСТИНА
INT	ЦЕЛЫЙ	TYPE	ТИП
INTO	В	UNDEFINED	НЕОПР
IO	СИЛ		
		VALUE	ЗНАЧЕНИЕ
		WEAK	СЛ
		WHILE	ПОКА

Синтаксис языка УТОПИСТ

Текст-программы:

ПРОГРАММА' идент[;]
{ ПУСТЬ' описание; ... }
{ ДЕЙСТВИЯ' оператор; ... }
{ ПОДПРОГР' идент [(имя, ...)] ; оператор; ... }
КОНЕЦ'[;]+++

описание:

{ описание — компонент }
{ описание — отношения }

описание — компонент:

идент, ... : описатель

описатель:

{ элем — описатель }
{ составной — описатель }
{ описатель — ряда }
{ вхождение — объекта }
{ НЕОПР' }

элем — описатель:

{ ЦЕЛЫЙ' }
{ ВЕЩ' }
{ ЛОГ' }
{ СТРОКА' [[ДЛИНА'=] целое] }
{ ПАМЯТЬ' [[ДЛИНА'=] целое] }

составной — описатель

({ описание } ; ...)
({ КОПИЯ'имя })

описатель — ряда:

МНОЖ' (целое ... целое) : описатель

вхождение — понятия:

название — понятия [подстановка — параметров, ...]

название — понятия:

имя

имя:

идент [. { (целое) }] ...
{ идент }
{ ВСЕ' }

подстановка — параметров:

[параметр [{ ТИП' }]] { новый — идент }
{ = } { имя }
{ значение }

параметр:

имя

новый — идент:

идент

значение:

{ простое — значение }
{ (значение, ...) }

простое — значение:

{ [[{ + }] { число }] }
{ { − } { целое } }
{ строка }
{ лог—значение }

лог — значение:

{ ИСТИНА' }
{ ЛОЖЬ' }

описание — отношения:

[метка:] [условие] [( )] { программный — модуль / присваивание / генератор / уравнение } [( )]

условие:

ЕСЛИ' лог — выражение ТО'

программный — модуль:

{ МОДУЛЬ' / МАКРО' } идент [[ТИП' =] { ПРОГ' / УРАВ' }] [[РАНГ' =] целое]

[роль — параметра параметр, ...] ...

роль — параметра:

{ ВХ' / ВЫХ' / СЛ' / СИЛ' / ВВХ' / ВВЫХ' / ОПР' }.

присваивание:

ПРИСВ' имя: = { значение / выражение }

генератор:

ЗАДАНО' {имя=значение} , ...

уравнение:

УРАВ' ариф — выражение = ариф — выражение

выражение:

{ ариф — выражение / лог — выражение }

ариф — выражение:

ариф — терм [бинарная — ариф — операция ариф — терм]...

ариф — терм:

{ имя / целое / число / (ариф — выражение) / (— ариф — терм) / унарная — ариф — операция ариф — терм }

бинарная — ариф — операция:

{ + / — / / / * / ** }

унарная — ариф — операция:

{SIN' | COS' | TG' | ARCTG' | ABS' | LN' | EXP' | SQRT'}

лог — выражение:

лог — терм [{ И' / ИЛИ' / СР' } лог — терм]

лог — терм : ...

[НЕ'] { имя / лог — значение / (лог — выражение) / лог — отношение }

лог — отношение:

ариф — выражение { РАВ' / НР' / МР' / БР' } ариф — выражение

оператор:

{ безусловный — оператор / условный — оператор }

условный — оператор:

условие безусловный — оператор [ИНАЧЕ' оператор]

безусловный — оператор:

{ оператор — присваивания
  оператор — возврата
  вызов
  оператор — цикла
  составной — оператор
  оператор — задачи
  присваивание — значения — функции }

оператор — присваивания:

имя : = { [{ + / — }] выражение / { целое / число } }

оператор — возврата:

ИЗ' метка

метка:

идент

вызов:

ВЫЗОВ' идент [(факт—параметр, . . .)]

факт — параметр:

{ имя
  простое — значение }

оператор — цикла:

ПОКА' лог — выражение ЦИКЛ' оператор

составной — оператор:

[метка:] (оператор; . . .)

оператор — задачи:

НА' имя ВЫЧИСЛИТЬ' имя, . . . [ПО' имя, . . .]

присваивание — значения — функции:

ЗНАЧЕНИЕ' имя

Синтаксис текстовой модели задачи

<описание> : : = { <идентификатор> | <целое> } <описатель вида> ;

<целое> : : = <цифра> ...

<описатель вида> : : = { <стандартный объект> | (<описание> ...) }

<стандартный объект>: : = { <переменная> | <отношение> }

<переменная> : : = { ЦЕЛ' | ВЕЩ' | ЛОГ' | СТР' [<целое>] | ПАМ' [<целое>] }

<отношение> : : = { МОД' | МАК' } <идентификатор> { ПРО' | УРА' } <целое>
[<список связей>] [<текст макроопределения>]

<список связей> : : = {<тип связи> <имя>} ...

<тип связи> : : = {ВХ'/ВЫХ'/СЛ'/СИЛ'/ОПР'/ВВХ'/ВВЫ'}

<имя> : : = <идентификатор> [· { <идентификатор> | <ВСЕ'> }] ...

<идентификатор> : : = <символ> ...

<текст макроопределения> : : = текст на макроассемблере

Текст задачи на языке ELEQ

```
C   СЕКЦИЯ — УРАВНЕНИЙ — СЛУЖИТ ДЛЯ ОПИСАНИЯ ЭЛЛИПТИ-
C   ЧЕСКОГО УРАВНЕНИЯ И ЕГО КОЭФФИЦИЕНТОВ В ФОРМЕ:
                (A*U'X)'X+(B*U'Y)'Y=C;
C               EQUATION;
                ((ALPHA*U'X)'X+(ALPHA*U'Y)'Y=R;
                ALPHA=0.  1*(X—20)**2;
                R=15;
C   СЕКЦИЯ — ГРАНИЦ — СЛУЖИТ ДЛЯ ОПИСАНИЯ ДВУХМЕРНОЙ
C   ОГРАНИЧЕННОЙ ОБЛАСТИ, НА КОТОРОЙ РЕШАЕТСЯ ЭЛЛИПТИ-
C   ЧЕСКОЕ УРАВНЕНИЕ, ЗДЕСЬ ОПРЕДЕЛЯЕТСЯ НАБОР ОТРЕЗКОВ,
C   ,ФОРМИРУЮЩИХ ГРАНИЦУ ОБЛАСТИ И ИМЕНА МЕТОК ГРАНИЧ-
C   НЫХ УСЛОВИЙ.
                BOUNDARY;
                (0,40) Y=0,B1;  (40) Y=0: :A,B2;
                (0) Y=0: :A,B2;  (0,40) Y=A,B3;
C   СЕКЦИЯ — ГРАНИЧНЫХ — УСЛОВИЙ — ПРЕДСТАВЛЯЕТ СОБОЙ
C   ПОСЛЕДОВАТЕЛЬНОСТЬ ГРАНИЧНЫХ УСЛОВИЙ ЗАДАЧИ ДИРИХЛЕ.
                BCOND;
                B1:  U=0;  B2:  U=40/A*Y;
                B3:  U=40/A*Y; B4: U=40;
C   СЕТОЧНАЯ — СЕКЦИЯ — ЗАДАЕТ ШАГИ СЕТКИ, ПОКРЫВАЮ-
C   ЩЕЙ ЗАДАННУЮ ОБЛАСТЬ, В УЗЛАХ КОТОРОЙ РЕШАЮТСЯ
C   КОНЕЧНОРАЗНОСТНЫЕ УРАВНЕНИЯ.
                MESH;
                STEP(X)=1.0;  STEP(Y)=2.0;
C   СЕКЦИЯ — АЛГОРИТМА — ЗАДАЕТ ОСТАЛЬНЫЕ ПАРАМЕТРЫ ДЛЯ
C   РЕШЕНИЯ УРАВНЕНИЯ МЕТОДОМ ВЕРХНЕЙ РЕЛАКСАЦИИ:
C   INITIAL — НАЧАЛЬНАЯ ИНИЦИАЛИЗАЦИЯ ИТЕРАТИВНОГО ПРО-
C   ЦЕССА РЕШЕНИЯ
C   ORF — ЗНАЧЕНИЕ ВЕРХНЕЙ РЕЛАКСАЦИИ,
C   ERROR — ВЕЛИЧИНА ДОПУСТИМОЙ ОШИБКИ,
C   MAXITER — МАКСИМАЛЬНОЕ ЧИСЛО ИТЕРАЦИЙ.
                ALGORITHM;
                INITIAL=50;  MAXITER=30;
                ORF=1.85;  ERROR=0.01;
C   СЕКЦИЯ — ТАБЛИЦ — ЗАДАЕТ ШАГИ РАСПЕЧАТКИ ТАБЛИЦЫ
C   РЕШЕНИЯ ПО ОСЯМ X И Y
                TABLE;
                TAKE(X)=1;  TAKE(Y)=2;
C   СЕКЦИЯ — ПОВТОРОВ — ДЛЯ ПОВТОРНОГО РЕШЕНИЯ С НОВЫ-
C   МИ ЗНАЧЕНИЯМИ ПАРАМЕТРОВ.
                REPEAT;
                A=60;
                FINISH;
```

Текст задачи на языке ELEQ — УТОПИСТ

```
A IS  A PARAMETER OF ELEQ;
  EQUATION;  (A*T'X)'X+(B*T'Y)'Y=R;
    EQN'A=0.1*(X—20)*(X—20);
    EQN'B=0.1*(X—20)*(X—20);
    EQN'R=15;
  BOUNDARY;
    (0,40)YN=0,B1;
    (0,40)YV=0: :A,B2;
    (0)YV=0: :A,B3;
    (0,40)YN=A,B4;
```

```
    BCOND;
      EQN'B1=0;
      EQN'B2=40/A*Y;
      EQN'B3=40/A*Y;
      EQN'B4=40;
OTHERS PARAMETERS OF THE PROBLEM.
    MESH;
      STEP(X)=1.0; STEP(Y)=2.0;
    TABLE;
      TAKE(X)=1; TAKE(Y)=2;
    ALGORITHM;
      INITIAL=50; MAXITER=30;
      ORF=1.85; ERROR=0.01;
      QUANT=4;
    REPEAT;
      A=60
```

ЛИТЕРАТУРА

1. Knuth D. E. Semantics of context-free languages. — Math. Sept. Theory, 1968, № 2, v. 2, p. 127—144.
2. Меристе М. Б. Методы реализации атрибутных схем в системах построения трансляторов. — Программирование, 1980, № 5.
3. Основы разработки трансляторов. Ростов-на-Дону, РГУ, 1974.
4. Григоренко В. П., Саан Ю. П., Сотникова Н. С. Опыт применения системы ПРИЗ-32 при построении пакетов прикладных программ САПР. — Программирование, 1979, № 1.
5. Левин Д. Я. Система СЕТЛ. Вып. 1. Программирование в системе СЕТЛ. Препринт ВЦ СО АН СССР, 1978, № 138.
6. Codd E. F. A Relational Model of Data for Large Shared Data Banks. — Communications of the ACM, 1970, № 6, v. 13, p. 377—387.

ОГЛАВЛЕНИЕ

Мильви Иоханнесовна Кахро
Ахто Пээтерович Калья
Энн Харальдович Тыугу

ИНСТРУМЕНТАЛЬНАЯ СИСТЕМА ПРОГРАММИРОВАНИЯ ЕС ЭВМ (ПРИЗ)

Рецензент *В. М. Брябрин*

Зав. редакцией *И. Г. Дмитриева*

Редактор *Т. А. Петрова*

Мл. редактор *Г. В. Корниенко*

Техн. редактор *Р. Н. Феоктистова*

Корректоры *Г. В. Хлопцева, И. П. Елкина*

Худож. редактор *Э. А. Смирнов*

Обложка художника *А. И. Гольдмана*

ИБ № 1046

Сдано в набор 16.01.81. Подписано в печать 21.08.81. А 01715. Формат 60×90¹/₁₆. Бум. тип. № 3. Гарнитура «Литературная». Печать высокая. П. л. 10. Усл. п. л. 10. Усл. кр.-отт. 10,25. Уч.-изд. л. 9,56. Тираж 15 000 экз. Заказ 641. Цена 50 коп.

Издательство «Финансы и статистика»,
Москва, ул. Чернышевского, 7.

Великолукская городская типография управления издательств, полиграфии и книжной торговли Псковского облисполкома,
г. Великие Луки, ул. Полиграфистов, 78/12

Кахро М. И. и др.

К30 Инструментальная система программирования ЕС ЭВМ (ПРИЗ)/М. И. Кахро, А. П. Калья, Э. Х. Тыугу. — М.: Финансы и статистика, 1981. — 158 с., ил.

50 к.

Описывается система программирования ПРИЗ, расширяющая возможности систем программирования операционной системы ОС ЕС ЭВМ и позволяющая повышать эффективность труда программистов. Приводятся принципы построения данной системы, рассматриваются расширяемый язык УТОПИСТ и технология программирования в этой системе.

Книга предназначена для программистов широкого профиля, может быть использована аспирантами и студентами вузов, изучающими системы обработки информации.

К $\frac{30502-107}{010(01)-81}$ 85—81(С) 2405000000 ББК 32.973
6Ф7.3