EESTI NSV TEADUSTE AKADEEMIA TOIMETISED. XVI KÖIDE
FÜÜSIKA * MATEMAATIKA. 1967, NR. 3

ИЗВЕСТИЯ АКАДЕМИИ НАУК ЭСТОНСКОЙ ССР. ТОМ XVI
ФИЗИКА * МАТЕМАТИКА. 1967, № 3

https://doi.org/10.3176/phys.math.1967.3.02

Б. ТАММ

ПРОБЛЕМЫ АВТОМАТИЗАЦИИ ПРОГРАММИРОВАНИЯ В ЭСТОНСКОЙ ССР

I

Вопросы математического обеспечения вычислительных машин возникли в нашей республике в начале 60-х годов, вскоре после ввода в эксплуатацию первых ЭВМ в Таллине и Тарту. Больше того, скоро выяснилось, что частичная автоматизация программирования, как применение операторного метода, программирование в символических и содержательных адресах, библиотеки стандартных подпрограмм, метод интерпретации и т. п., хотя и весьма важная на некоторых этапах, не может дать принципиального решения ни с точки зрения ЭВМ (т. е. их математического обеспечения), ни с точки зрения описания алгоритмов для машинного решения новых классов задач. Кроме того, мы были вынуждены работать над проблемами, соизмеримыми по объему, располагая силами (по вычислительной математике и технике) примерно на порядок меньшими, чем в крупных научных центрах.

В таком положении оказалось необходимым и, как выяснилось, своевременным поставить некоторые работы, специально посвященные исследованию самого программирования, а также алгоритмизации задач с целью их автоматизации. Основная цель этих работ состояла в создании систем автоматического программирования, использующих алгоритмические языки высших уровней и трансляторы, способные реализовать на малых ЭМВ любой имеющий смысл алгоритм, записанный на соответствующем языке программирования.

Ввиду большого дефицита хороших программистов в трудном положении оказались в первую очередь сами вычислительные центры. Следовательно, неотложной задачей стало резкое повышение производительности труда программистов. Учитывая характер основной массы задач, решавшихся тогда на ЭВМ (преимущественно они были связаны с научно-исследовательскими работами), легко было установить, что он сводится в основном к вычислению арифметических операторов. Поэтому одно из направлений поставленных работ состояло в разработке систем с процедурной ориентацией и, в первую очередь, систем типа АЛГОЛ [1, 2]. Задача сводилась к определению такого точного подмножества АЛГОЛа, которое, позволяя решать широкий класс задач, дало бы возможность упростить структуру и синтаксис языка и, что самое главное, позволило бы реализовать соответствующий транслятор на малых отечественных ЭВМ типа «Минск» и «Урал».

В результате трехлетней работы небольшая группа математиков Института кибернетики АН ЭССР во главе с М. Котли в 1965 году предложила систему МАЛГОЛ, удовлетворяющую упомянутым требованиям и запрограммированную вначале для М-3, а затем для «Минск-2» и «Минск-22». Она быстро прижилась в институте, вызвала интерес во многих вычислительных центрах страны, а теперь используется уже и в странах социализма.

Язык МАЛГОЛ отличается от языка АЛГОЛ главным образом шестью особенностями, которые существенно облегчают программирование [3, 4]. Особенности эти следующие:

1) отказ от блочной структуры языка;
2) отказ от описаний типа;
3) отказ от именующих и условных выражений;
4) запрещение использования рекурсивных процедур;
5) изменение формы записи процедур, определяющих функцию;
6) включение понятия подпрограммы.

Вторым большим достижением авторов МАЛГОЛа (кроме упрощения использования языка по сравнению с АЛГОЛом) является разработка и внедрение эффективно работающего транслятора, который вместе с библиотекой стандартных подпрограмм занимает всего лишь 8300 ячеек.

Хорошо известно, что системы типа АЛГОЛ совершенно непригодны для обработки строчной информации, для оперирования большими информационными массивами, особенно с меняющимися границами. По мере развития исследований по применению математических методов и вычислительной техники в экономике и планировании назрела необходимость автоматизации обработки информации такого характера. В 1964 году аспирантом Института кибернетики В. Куузиком была начата работа над созданием системы автоматического программирования для автоматической обработки указанного класса задач. Уже через два года новая система ВЕЛГОЛ (которая, если уж сравнивать, стоит гораздо ближе к таким известным системам, как LISP и IPL, чем к АЛГОЛ) проходила первые испытания на ЭВМ «Минск-2» [5, 6, 13]. Оказалось, что хотя система принципиально удовлетворяла предъявляемые к ней требования, программы, составленные транслятором в машинных кодах, работали очень медленно. После тщательной переделки транслятора с применением ряда изящных приемов техники программирования В. Куузику удалось повысить эффективность работы транслятора на целый порядок. В настоящее время заканчивается отладка ВЕЛГОЛ-II и еще в этом году она поступит на вооружение наших ВЦ.

Упомянутые две работы представляют собой наиболее существенные достижения республики в области развития систем с процедурной ориентацией, и поэтому во втором разделе данной статьи дается некоторый сравнительный анализ языков МАЛГОЛ и ВЕЛГОЛ.

Еще в 1959 году начала складываться группа сотрудников, которая интересовалась автоматизацией программирования с точки зрения инженера-автоматика. Внимание этой группы (в состав которой первоначально входили Ю. Прууден и автор) было обращено на использование цифровых ЭВМ при автоматизации умственного труда инженера. А необходимость в этом к тому времени уже назрела, в первую очередь в таких производственных процессах, где степень автоматизации была достаточно высока. Например, отсутствие эффективных методов подготовки исходной информации составляло основную трудность использования современных металлорежущих станков с цифровым программным управлением [7]. Процесс подготовки информации (т. е. изготовление управляющих станками программ) здесь сложен. Он содержит расчет параметров отрезков траектории движения центра режущего инструмента (эквидистанты обрабатываемого контура), вычисление переходных точек этих отрезков и технологических параметров, выбор окончательных данных для каждого кадра, кодирование этих данных и т. д. [8, 9]. Очевидно, что процесс этот весьма трудоемок и, помимо больших затрат времени, требует квалифицированной рабочей силы. Даже применение ЭВМ и обычных методов программирования не дает существенного эффекта.

При такого рода логико-информационных задачах вопрос состоит в том, чтобы разработать математический аппарат, позволяющий опытному специалисту-производственнику, не имеющему подготовки по вычислительной математике и технике, легко обращаться к ЭВМ для решения своей проблемы, т. е. необходим аппарат, который может служить посредником в системе человек—машина для решения определенного класса логических проблем. Такой аппарат, разумеется, должен иметь выразительные средства, позволяющие фиксировать мыслительный процесс специалиста при формировании им той или иной проблемы. Исходя из вербального характера мышления человека, естественно организовать эти выразительные средства в виде соответствующего искусственного языка с ориентацией на данную работу. Такой язык должен быть метаязыком для вычислительных и логических процедур, используемых машиной при решении задач данного класса, что позволяет программисту освободиться от уточнения многих деталей и сконцентрировать свое внимание на основной проблеме, которую он хочет решить.

С другой стороны, этот аппарат должен быть понятен вычислительной машине и допускать эффективное решение всех задач данного класса. Это достигается иерархическим построением транслятора с применением различных уровней (внутримашинных) промежуточных языков. Нельзя не отметить, что при построении такого транслятора, который должен быть способным синтезировать все алгоритмы решения задач данного класса, необходимо исходить из принципов эвристического программирования.

Учитывая указанные положения, группа сотрудников Института кибернетики разрабатывала системы автоматического программирования САП-I (1962 г.) и САП-II (1964 г.) — первые такого рода системы в СССР — и АПРОКС (1967 г.). Более подробный анализ этих систем дан в третьем разделе статьи.

Помимо указанных двух основных направлений в автоматизации программирования и отмеченных конкретных работ, проведены еще некоторые исследования, результаты которых вносят существенный вклад в развитие теории и техники программирования.

Вычисление всевозможных интегралов — одна из наиболее распространенных и трудоемких задач при решении многих математических и технических проблем. После доказательства Бахваловым (в 1964 г.) теоремы о невозможности конструирования для больших классов функций квадратурных формул, которые были бы лучше существующих, один из узловых вопросов теории приближенного интегрирования состоял в конструировании наилучшей в некотором смысле квадратурной формулы для данной функции $f(x)$ на данном отрезке $[a, b]$ с заданной точностью ε. Эта задача, разумеется, должна быть решена на машинном уровне. Развернутая программа (по сути дела транслятор), составленная сотрудником Института кибернетики Р. Пукк, позволяет вычислительной машине найти оптимальную для данной функции квадратурную формулу и интегрировать эту функцию с заданной точностью. При этом программа сама определяет стратегию интегрирования (по мере изменения характера функции) и минимизирует число обращений к подынтегральной функции, т. е. машинное время. Входным языком является здесь обыкновенная математическая запись интегралов [10] *.

В вычислительном центре Тартуского государственного университета под руководством А. Корьюса разработано конкретное представление языка АЛГОЛ-60 и соответствующий транслятор для ЭВМ «Урал-4». Этот язык отличается от эталонного выбором основных символов и резервированных слов, обусловливаемых периферийными уст-

* Работа была выполнена под руководством профессора А. Кронрода (ИТМВТ, Москва).

ройствами «Урал-4». По своей грамматической структуре он не отличается от АЛГОЛа. Транслятор этой системы находится сейчас в стадии отладки [11].

Совсем иное по сравнению с предыдущими системами приложение имеет разработанный сотрудником Института кибернетики Х. Салумом язык ЦИМОД (ЦИфровое МОДелирование) для записи блок-схем цифровых устройств с целью моделирования их работы (во времени) на цифровых ЭВМ.* Записанная на языке ЦИМОД блок-схема вводится в моделирующую ЭВМ и интерпретируется, результатом чего является информация о различных состояниях отдельных логических элементов схемы в разные моменты условного времени моделирования. Кроме того, можно получить нужные для проектировщика временные характеристики работы отдельных элементов или узлов. Основным понятием языка является:

<запись схемы> : : = <заголовок> # <микропрограмма> #

Заголовок представляет собой совокупность массивов перечней элементов схемы и возможных в схеме элементарных операций переработки информации.

Эта работа еще не завершена в смысле практического применения ввиду отсутствия эффективно работающей интерпретирующей программы моделирования для какой-нибудь из ЭВМ [12].

II

Языки МАЛГОЛ и ВЕЛГОЛ базируются на знаковой системе второго международного телеграфного кода.

Идентификаторы строятся из букв и цифр, причем МАЛГОЛ различает первые шесть, ВЕЛГОЛ — первые пять знаков. МАЛГОЛ использует буквы латинского, а ВЕЛГОЛ — русского алфавита, входящие в буквенный регистр телеграфного кода. Резервированных идентификаторов в МАЛГОЛе — 24 (английские), в ВЕЛГОЛе — 20 (русские) *.

Числа в обоих языках могут состоять не более чем из девяти значащих десятичных цифр. Для десятичной дроби вместо запятой используется точка, причем в МАЛГОЛе она должна находиться между цифрами (напр., 0.25), в ВЕЛГОЛе это необязательно (напр., .25). В МАЛГОЛе используется еще десятичный показатель степени. В этом случае число имеет вид:

г ю t, что означает $г \cdot 10^t$,

например, $200 = 200.0 = 2Ю2 = 0.0002Ю6 = 20000Ю - 2$ и т. д.

Там же используются еще восьмеричные числа, которые записываются при помощи идентификатора OCTAL′ и представляются в машине в виде восьмеричного целого числа. Например, OCTAL′ 135 0000 представляется в машине как 0000 0135 0000.

В ВЕЛГОЛе константами могут служить (кроме чисел) еще символы, заключенные между апострофами. Значением такого символа является он сам. Введение такого обобщенного понятия значения (где таковым может быть как число, так и символ) необходимо для обработки текстовой информации отдельно от чисел. Так, если $1 + 2 = 3$, то значение 1 + '2' не определено, потому что не существует смешанных действий с числами и символами, а запись 1 '+' 2 представляет три константы. При вводе в ЭВМ числами являются числовые константы и элементы совокупностей, специфицированные как массивы; символами являются символьные константы ('. . .') и элементы совокупностей, специфицированные как коды. На выходе ЭВМ печатаются только символы, для чего все числа переводятся в символы, т. е. в код.

---

* Работа выполнена под научным руководством академика В. Глушкова (ИК АН УССР).

* В настоящее время разрабатывается новая редакция языка ВЕЛГОЛ на базе латинского алфавита.

Переменные (массивы, коды)

а) ВЕЛГОЛ. Значением переменной может быть как число, так и символ. Для обозначения совокупностей переменных при их идентификаторах используются индексы, число которых определяет размерность массива (кода). Индексом может служить произвольное, имеющее одно целочисленное значение выражение. Индексная шкала состоит из двух выражений, разделенных двоеточием. Эти выражения A и B отражают границы изменения индекса, причем $A \leqslant B$. Индексные шкалы пишутся в фигурных скобках. Индексная шкала, соответствующая последнему размеру кода, не берется в фигурные скобки в случае, если символы кода, относящиеся к области изменения данной шкалы, рассматриваются как одно число. Это позволяет преобразовать цифровой текст в числа.

Пусть значениями элементов кода ТЕКСТ [{1 : 4}, {1 : 6}] являются символы

—	6	—	М	А	Я
3	5	—	Р	У	Б
1	.	5	—	К	Г
—	3	.	4	2	5

тогда

ТЕКСТ [2 , 1 : 2] дает число 35
ТЕКСТ [4 , 1 : 4] „ „ 3.4
ТЕКСТ [{2 : 4} , 2 : 3] „ числа 5, 0.5, 3
ТЕКСТ [{1 : 3} , 2 : 2] „ „ 6, 5, 0

Если же вместо последней индексной шкалы стоит одинокий индекс, то символы остаются символами. Например, значением

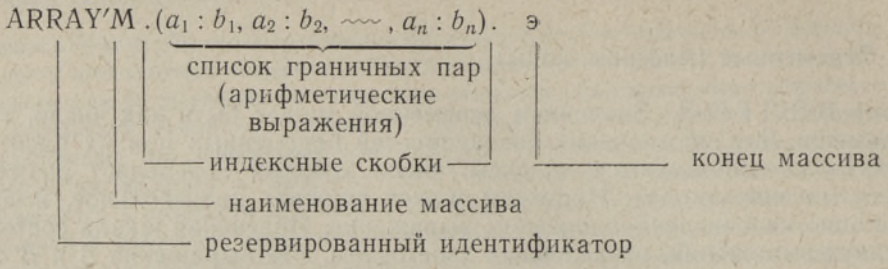
ТЕКСТ [{1 : 3} , 2] являются символы «6», «5» и «.»
ТЕКСТ [1 , 1] — символ « — »
ТЕКСТ [1 , 1 : 1] — число 0

б) МАЛГОЛ. Переменной может быть только численная величина, которой присваивается наименование при помощи идентификатора. Массивы — переменные с индексами — тоже численные. Индексы находятся в индексных скобках .( ).. Общий вид массива:

$$M.(v_1, v_2, \sim\sim, v_n).$$

где М — идентификатор массива, v_i — арифметические выражения, $\sim\sim$ обозначает три точки.

Значения переменной с индексами изменяются не только «по массиву», но могут быть использованы и при изменении значений элементов массива с помощью оператора присваивания. Сказанное относится также к ВЕЛГОЛу (см. ниже). Отметим еще, что оператор, описывающий массив, должен стоять перед операторами, использующими элементы массива. Общий вид оператора описания n-мерного массива в МАЛГОЛе следующий:



Выражения. Наименьший элемент программы ВЕЛГОЛ — операнд. Это константа, переменная, элемент или часть массива (кода) и т. д. Выражение состоит из операндов, знаков, арифметических операций и круглых скобок и является правилом вычисления одного или нескольких значений. Ввиду характера МАЛГОЛ его выражения бывают только арифметическими. Обе системы используют четыре основные арифметические действия плюс возведение в степень (МАЛГОЛ) и извлечение абсолютной величины (ВЕЛГОЛ). МАЛГОЛ предназначен для выполнения операций со сложными арифметическими выражениями, поэтому здесь разработан комплекс правил для их записи. Это относится и к строгому порядку выполнения арифметических операций — к их старшинству. В ВЕЛГОЛе, где важное значение имеют различные операции между множествами численных величин, допускаются следующие виды арифметических операций:

1) между двумя численными значениями — результатом является одно численное значение;

2) между одним численным значением и множеством численных значений — результатом является множество численных значений;

3) между двумя множествами численных значений — результатом является множество численных значений, содержащее столько значений, сколько содержится в меньшем из двух множеств.

Примеры:

A [{1 : 3}] + 5 — к каждому элементу из A прибавляется 5; всего 3 ответа.

A [{1 : 6}] × B [{1 : 2}, {1 : 3}] — умножение находящихся на взаимно соответствующих позициях величин; всего 6 ответов.

A [{1 : 6}] + B [1,1] — имеет 6 ответов (случай *2*)).

A [{1 : 6}] + B [1, {1 : 1}] — имеет один ответ (случай *3*)).

A [{1 : 100}] × (A[{1 : 50}] + B[1, {1 : 1}]) — имеет также один ответ, потому что находящаяся в круглых скобках сумма дает множество, содержащее только одно значение; оно и является меньшим множеством (случай *3*)).

Операторы. В МАЛГОЛе различают восемь типов операторов (рис. 1, *а*), а в ВЕЛГОЛе — пять типов, которые в свою очередь разделяются на более мелкие (рис. 1, *б*). Несмотря на то, что наименования операторов в обоих языках не совпадают (кроме одного), они выполняют многие аналогичные задачи. Единственный совпадающий по наименованию оператор — оператор присваивания, который в обоих языках предназначен для присвоения переменной нового значения. Для реализации этого оператора в МАЛГОЛе используется знак присваивания, а в ВЕЛГОЛе — знак равенства.

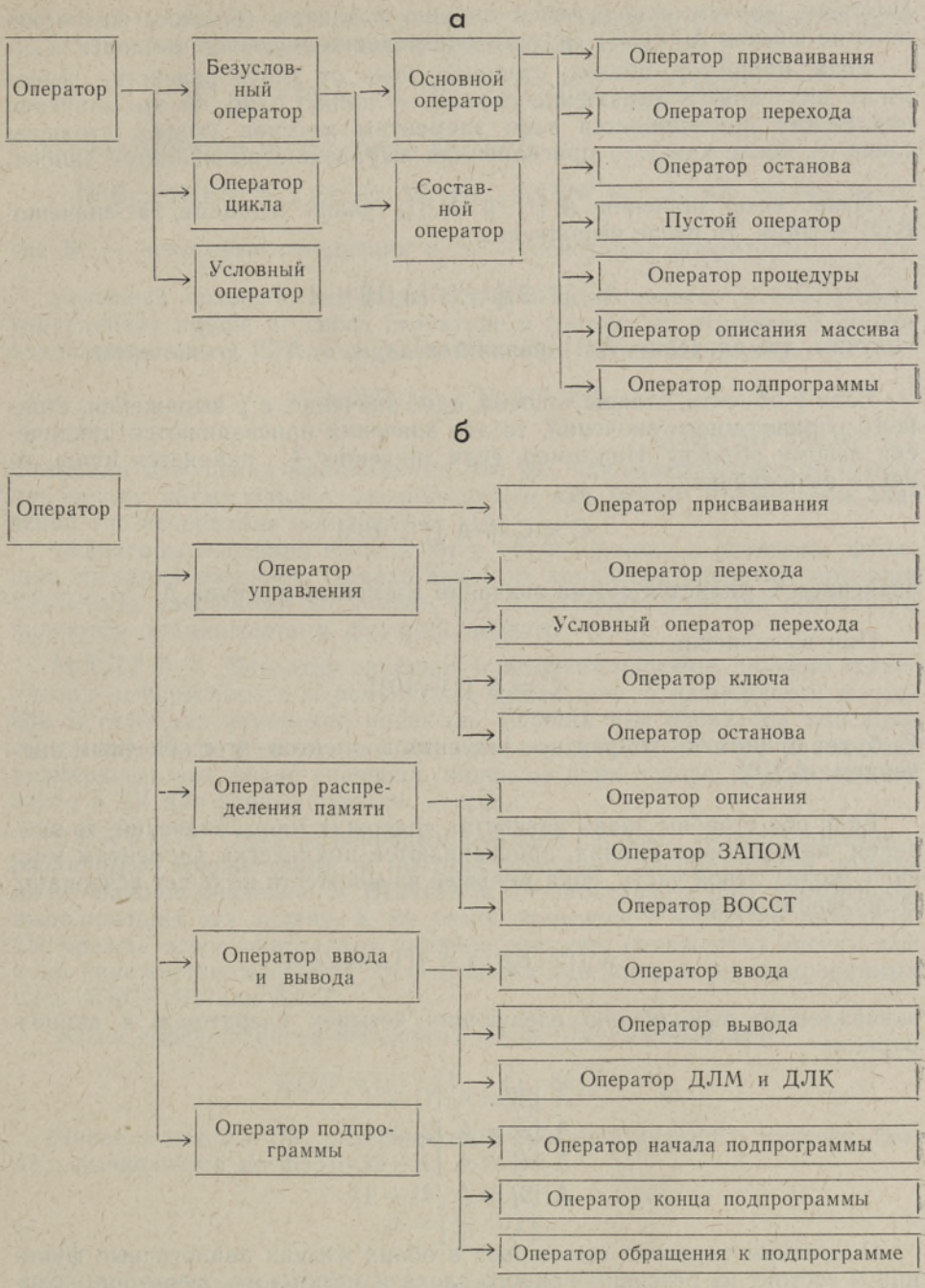


Рис. 1.

Примечание. Разрабатываемая в настоящее время редакция ВЕЛГОЛ-2 имеет несколько измененную структуру операторов.

Оператор присваивания МАЛГОЛа мало отличается от соответствующего оператора АЛГОЛа. В ВЕЛГОЛе же ввиду того, что изменение значений переменных двух различаемых там множеств (массивы и коды) играет важную роль, оператор присваивания разработан более

детально, причем используется хорошо развитая (больше, чем в семействе языков АЛГОЛ) система обозначения массивов (кодов).

Если, например, объект, стоящий слева от знака равенства, имеет много значений, а выражение справа — только одно, то это значение циклически присваивается всем элементам массива (кода), стоящим слева. А после каждого присваивания выражение вычисляется заново.

Итак, если значения A [1] и B [1] равны единице, а значение B [2] — двум, то после выполнения

$$A\ [\{1:2\}] = B\ [A\ [1]] + 1$$

получим, что значение A [1] равняется двум, а A [2] равно трем.

Если у объекта, стоящего слева, одно значение, а у выражения, стоящего справа, много значений, то эти значения присваиваются циклически левому объекту. Например, если значение C равняется нулю, то после выполнения

$$C = C + A\ [\{1:10\}]$$

значением C является сумма значений элементов массива A.

При выполнении же

$$C = A\ [\{1:10\}]$$

C будет принимать подряд все значения элементов A с конечным значением A [10].

Если обе стороны знака равенства содержат много значений, то значения, находящиеся справа, присваиваются циклически элементам массива (кода) левой части, пока меньшее из множеств не будет исчерпано. Выполнение оператора

$$A\ [\{2:5\}] = A\ [\{1:4\}] + 1$$

эквивалентно выполнению следующих четырех операторов в данном порядке

$$A\ [2] = A\ [1] + 1$$
$$A\ [3] = A\ [2] + 1$$
$$A\ [4] = A\ [3] + 1$$
$$A\ [5] = A\ [4] + 1$$

Операторы перехода выполняют в обоих языках аналогичные функции и состоят из резервированного слова и метки того оператора, который будет выполняться после оператора перехода:

GOTO′ L (МАЛГОЛ)
НА MI (ВЕЛГОЛ)

Условный оператор перехода МАЛГОЛа похож на таковой АЛГОЛа и имеет в программах важное значение. Он может рассматриваться как

составной оператор, части которого могут существовать и самостоятельно в качестве так наз. неполных операторов перехода:

$$\text{IF}' \sim\sim \text{THEN}' \quad \text{и} \quad \text{IF}' \sim\sim \text{THEN}' \sim\sim \text{ELSE}'$$

а полный оператор может иметь вид [1–3]

$$\text{IF}'B_1\_\text{THEN}'\,S_1\_\text{ELSE}'\;\text{IF}'\;B_2\_\text{THEN}'\,S_2\_\text{ELSE}'\,S_3\;\text{ЭS}$$

где B_i — некоторые отношения и S_i — некоторые операторы.

Условный оператор перехода ВЕЛГОЛа (относится к операторам управления) проще по своей структуре и состоит из отношения, резервированного слова НА и следующей за ним метки. Например,

$$\text{A MP } (\text{K} + 5) \text{ НА M5}$$

(здесь MP значит $\leqslant$)

Операции отношения выполняются так же, как арифметические операции между двумя одинокими значениями или между множеством значений (до исчерпания меньшего из них).

Операторы описания выполняют в обоих языках в основном функцию описания массивов. Однако ввиду их характера оператор описания ВЕЛГОЛа разработан подробнее, чем МАЛГОЛа: функции последнего являются подмножеством функций первого.

МАЛГОЛ. Оператор описания описывает массивы, дающие транслятору информацию о переменных с индексами: о размерности массивов, о границах изменения индексов. (Общий вид оператора описания см. выше). В соответствии с описанием массива транслятор определяет размерность его, число значений, индексов и на основе этого выделяет место в ЗУ для записи данного массива.

ВЕЛГОЛ. Одна из основных функций системы — оперирование с различными множествами, и поэтому проблема распределения памяти имеет важное значение. В ВЕЛГОЛе разработана группа операторов, выполняющих эту задачу. К их числу относится и оператор описания. Он прежде всего определяет наличие массива (множество чисел) или кода (множество символов). Это делается при помощи резервированных слов МАССИВ или КОД.

Затем следует список индексных шкал. Двухмерный код:

$$\text{КОД A}[\{1:5\}\ ,\{1:60\}]$$

Число ячеек p в ЗУ, резервируемое для описываемого массива (кода), вычисляется по формуле

$$p = \left[\frac{n}{m} + 1\right],$$

где n — число элементов массива (кода),

$$m = \begin{cases} 1 & \text{— в случае массива} \\ 6 & \text{— в случае кода.} \end{cases}$$

Важное свойство оператора описания состоит в том, что он позволяет использовать массивы (коды), значения индексов которых могут выйти за пределы заранее декларированной области. Это позволяет

пользоваться элементами соседних массивов и кодов без явного обращения к ним, а также описывать массивы (коды) произвольной (наперед незаданной) длины.

Например, если дано описание

МАССИВ ТАБ [{1 : 10} , {1 : 20}]

то ТАБ [11, {1 : 20}]

обозначает содержимое двадцати первых ячеек ЗУ, следующих за данным массивом, а

ТАБ [—1, {1 : 10}]

обозначает содержимое ячеек, находящихся перед описываемым массивом и имеющих порядковые номера 30...40 (считая в обратную сторону от первого элемента массива). Такие удобные приемы могут быть использованы и по отношению к размерности массивов (кодов).

Ряд операторов как в МАЛГОЛе, так и в ВЕЛГОЛе сравнительно просты и похожи друг на друга или же, например оператор цикла в МАЛГОЛе, подобны аналогичным операторам АЛГОЛа. Все же хочется сделать несколько замечаний, касающихся процедур и подпрограмм.

Прежде всего надо сказать, что в МАЛГОЛе в отличие от АЛГОЛа, помимо процедур, используются и подпрограммы. Процедуры — это по сути дела подпрограммы, но с той существенной разницей, что в них все идентификаторы и метки операторов (кроме названий процедур и элементарных функций), входящие в описание процедуры, строго локализированы внутри данной процедуры. В описании (точнее — в заголовке) процедуры необходимо задать список формальных параметров; в описании подпрограммы этого, конечно, не требуется:

PROCEDURE′ P $(t_1, t_2, \sim\sim, t_n)$ ∋ BEGIN′ S_1 ∋ S_2 ∋ ~~ ∋ S_n END′ ∋

SUBROUTINE′A ∋ BEGIN′S_1 ∋ S_2 ∋ ~~ ∋ S_n END′ ∋

При обращении к подпрограмме наименование ее достаточно записать в соответствующие места последовательности операторов, а обращение к процедуре происходит при помощи оператора обращения, состоящего из идентификатора процедуры и списка фактических параметров.

Допустим, задана подпрограмма

SUBROUTINE′ SUM ∋ BEGIN′ ~~

Для двукратного обращения к ней достаточно писать

~~ S_k ∋ S_{k+1} ∋ S_{k+2} ∋ SUM ∋ S_{k+3} ∋ S_{k+4} ∋ SUM ∋ S_{k+5} ∋ ~~

Для процедуры PROCEDURE′ P$(t_1, t_2, \sim\sim, t_k)$ ∋ BEGIN′ ~~ оператор обращения имеет вид P $(p_1, p_2 \sim\sim, P_k)$

Процедуры в МАЛГОЛе существуют совершенно независимо от программы, в которой они используются. Поэтому однажды составленные процедуры могут быть использованы в любых написанных на языке МАЛГОЛ программах (ср. с библиотекой стандартных подпрограмм), а набор таких стандартных процедур во многом повышает мощность системы. С другой стороны, некоторые части основной программы часто

необходимо использовать повторно (без локализации идентификаторов и меток). Такие части описываются при помощи резервированного слова SUBROUTINE′ и они являются подпрограммами МАЛГОЛа.

Функции в МАЛГОЛе по своему характеру весьма близки к процедурам. Они разделяются на элементарные функции и функции, определенные процедурами. Элементарные функции представляют собой по существу простые стандартные процедуры, где характер функции (процедуры) определяется также с помощью резервированного слова, но выполнение самой функции (процедуры) не требует декларации фактических параметров. Вместо этого в круглых скобках дается арифметическое выражение — аргумент, для которого необходимо найти значение функции:

ABS'(F), SQRT'(F), SH'(F).

Функции, определенные процедурами, пишутся как

$$f(t_1, t_2, \sim, t_n),$$

где f — идентификатор, определяющий процедуру функции, и $t_1, t_2, \sim, t_n$ — фактические параметры.

Сам автор МАЛГОЛа дает функции следующее определение (из которого также следует ее близость с процедурой): функция — это определенное предписание, в результате применения которого к данной совокупности величин (фактических параметров) получается некоторое значение. Внешний вид функции МАЛГОЛа существенно отличается от такового функции АЛГОЛа, ибо в МАЛГОЛе нет понятия типа, которое в АЛГОЛе используется при описании функций.

В ВЕЛГОЛе вся программа делится на подпрограммы. Даже основная программа может рассматриваться как одна из подпрограмм. Они подобны процедурам МАЛГОЛа (особенно стандартным) в смысле, что все идентификаторы (кроме резервированных слов и наименований подпрограмм) тут локализированы. Следовательно, связь между подпрограммами происходит только при помощи формальных параметров, которые в каждом конкретном случае заменяются фактическими параметрами. Сама подпрограмма начинается с оператора ПОДПР и заканчивается оператором КОНЕЦ, а оператор обращения состоит из резервированного слова ПП, за которым следует наименование подпрограммы и (в круглых скобках) список фактических параметров, например:

ПП УПРАЖНЕНИЕ (К, 'А' , ТЕКСТ [ )

Мощность ВЕЛГОЛа во многом зависит от развитости и состава системы ее подпрограмм (написанных в самом ВЕЛГОЛе и предназначенных для решения конкретных классов задач), в связи с чем ВЦ ЭРСПО и Институт кибернетики АН ЭССР с участием Госплана республики ведут работу по разработке этой системы для плановых расчетов. Первые уже составленные подпрограммы (их около тридцати) относятся к вводу и выводу информации, к агрегированию выборки информации и к самому счету.

III

Разделение языков программирования на языки с процедурной ориентацией и языки с проблемной ориентацией является, разумеется, весьма условным и зависит от того, что мы подразумеваем под проблемой. Для математика, целью которого является решение алгоритмов чис-

ленного анализа, языки типа АЛГОЛ могут стать проблемно ориентированными. Ученые, инженеры, экономисты и т. п., использующие численные процедуры как одно из средств решения других комплексных задач, нуждаются для описания своих проблем в гораздо более удобных языках [14, 15].

Исследование специализированных систем автоматического программирования (в мировом масштабе) началось позже, чем исследование соответствующих систем общего назначения, и поэтому пока нет никакой общей теории этих систем. Нет даже общепринятой терминологии ни в Советском Союзе, ни за рубежом [16]. Число практически работающих систем автоматического программирования с проблемной ориентацией в нашей стране весьма скромное.

Однако нам представляется, что алгоритмические языки с проблемной ориентацией могут служить мощным средством моделирования широкого класса логико-информационных процессов, а вместе с соответствующими трансляторами — средствами автоматического управления этими процессами [8, 17, 18].

Анализ потоков информации показывает, что как различные множества информации, так и отдельные этапы ее переработки можно отнести к определенным классам и придать им точную количественную оценку или характеристику. Это обстоятельство создает предпосылки для создания моделей этих процессов и автоматизации управления ими.

Обширный класс логико-информационных процессов поддается описанию по нижеприведенной схеме. Каждый такой процесс имеет на входе некоторые массивы информации $\mathfrak{Q}$, не приспособленные к автоматической переработке. Эти массивы можно разделить на два множества $\mathfrak{A}$ и $\mathfrak{B}$. Тогда

$$\mathfrak{Q} = \mathfrak{A} + \mathfrak{B},$$

где $\mathfrak{A}$ — множество входной информации, представленное носителями, не допускающими прямого ввода в ЭВМ;

$\mathfrak{B}$ — множество входной информации, находящееся в ЭВМ или на носителях, допускающих непосредственный ввод в ЭВМ.

Одно из множеств $\mathfrak{A}$ или $\mathfrak{B}$ может оказаться и пустым. В зависимости от $\mathfrak{A}$ и $\mathfrak{B}$ подготовка первичной информации к автоматической переработке осуществляется при помощи двух различных множеств операторов. Эти операторы суть некоторые точные предписания, находящиеся в полном соответствии с используемым в дальнейшем языком программирования.

Обозначим эти операторы и соответствующие множества через $g_1, \ldots, g_n \in \mathfrak{G}$ и $f_1, \ldots, f_m \in \mathfrak{F}$. Образуем теперь новые множества из всех возможных составных операторов множеств $\mathfrak{G}$ и $\mathfrak{F}$:

$$g_1, \ldots, g_i, \ldots, g_n, g_1g_2, \ldots, g_1g_n, g_2g_1, \ldots, g_ng_1, \ldots,$$
$$g_1g_2 \ldots g_i, \ldots, g_i \ldots g_2g_1, \ldots, g_ng_{n-1} \ldots g_2g_1$$

и

$$f_1, \ldots, f_i, \ldots, f_m, f_1f_2, \ldots, f_1f_m, f_2f_1, \ldots, f_mf_1, \ldots,$$
$$f_1f_2 \ldots f_i, \ldots, f_i \ldots f_2f_1, \ldots, f_mf_{m-1} \ldots f_2f_1.$$

Обозначая элементы первого из новых множеств через v_i, а элементы второго — через w_i, можем написать

$$v_1, \ldots, v_k \in \mathfrak{V} \quad \text{и} \quad w_1, \ldots, w_l \in \mathfrak{W}.$$

Теперь можем писать

$$v(\mathfrak{A}) = \mathfrak{A}_1 \qquad v \in \mathfrak{V}$$

и

$$w(\mathfrak{B}) = \mathfrak{B}_1 \qquad w \in \mathfrak{W},$$

где $\mathfrak{A}_1$ и $\mathfrak{B}_1$ — множества информации, подготовленные к автоматической переработке;

w — некоторая совокупность подпрограмм;

v — набор «вручную» используемых предписаний.

В дальнейшем в ход процесса вмешивается язык программирования. Оба процесса, для которых в Институте кибернетики созданы системы автоматического программирования с проблемной ориентацией — САП-II и АПРОКС, полностью поддаются анализу по общей схеме, приведенной выше. Однако ввиду того, что системы предназначены для автоматизации подготовки информации существенно различных процессов (САП — для функциональных металлорежущих станков с цифровым программным управлением; АПРОКС — для автоматов вырезки деталей корпусов судов), каждая из них имеет свое лицо, соответствующее проблеме, которую она моделирует.

В случае САПа в множество $\mathfrak{A}$ входит информация, представленная на рабочих чертежах или эскизах деталей, подлежащих обработке ($\mathfrak{B} = \emptyset$). Это — результат конструкторской работы, непригодный для непосредственного использования в ЭВМ. В случае же АПРОКСа в $\mathfrak{A}$ входит информация, заложенная на картах раскроя и эскизах деталей, а в $\mathfrak{B}$ — результаты вычисления контуров деталей в локальных системах координат, находящихся в ЗУ ЭВМ.

Алгоритмические языки САП и АПРОКС разработаны на уровне проблемной ориентации или на «субчеловеческом» уровне искусственных языков [8, 17]. Они позволяют легко и однозначно описывать алгоритмы любых задач тех классов, для которых они предназначены.

Система САП предусмотрена для программирования изготовления так называемых плоских деталей с траекторией их обработки на плоскости, параллельной одной из трех координатных плоскостей, — плоскости $X-Y$. В то же время она непосредственно в процессе работы допускает переход к обработке в других плоскостях, параллельных любой из двух остальных координатных плоскостей, — к плоскостям $X-Z$ и $Z-Y$. Этим достигается программирование таких объемных деталей, траектории обработки которых состоят из частей, находящихся на взаимно параллельных и перпендикулярных плоскостях. Система позволяет программировать детали с контуром обработки, состоящим из отрезков прямых и дуг окружностей, выбирать точность аппроксимации последних, выбирать скорости подач и тип рабочего инструмента.

Система АПРОКС предназначена для автоматизации подготовки управляющей информации для автоматов обработки судокорпусных деталей как в режиме газовой, так и газоэлектрической резки, для одно- и многорезаковой резки, а также для разметки и точного вычерчивания деталей.

Эта система позволяет:

— программировать вырезку деталей, участки контуров которых определены либо координатами их опорных точек, либо коэффициентами образующих их кривых;

— осуществлять вырезку участков различных деталей одним совмещенным резом; при этом автоматически уточняется месторасположение деталей, имеющих вырезаемые совмещенным резом участки;

— назначать маршрут вырезки детали в зависимости от последовательности обработки ее участков; определять расположение перемычек;
— определять технологические параметры и комплексы обработки;
— определять маршрут и технологические параметры вырезки детали, а также месторасположение перемычек на ее участках частично или полностью по маршруту вырезки другой детали (тождественной данной детали, симметричной ей или отличающейся от нее по размерам), имеющей такой же порядок нумерации элементов контура.

Алгоритмы всех решаемых задач записываются в обеих системах в виде языковой программы, но изобразительные средства (словарь и внешняя форма записи языковой программы) и синтаксические конструкции этих языков сильно отличаются друг от друга.

Ввиду характера уже описанного множества $\mathfrak{A}_1$ в *языковой программе* САП вся числовая информация записывается в виде *элементарных обозначений*, например [8, 19]:

X02/ = /11000/
Y03/ = /06580/
R06/ = /01350/ и т. д.

Для однозначного определения решаемой задачи, как правило, необходимо подсчитать еще некоторое количество численных величин (относящееся к геометрической информации), заданных неявно. Эти величины фиксируются в языковой программе посредством *составных* (сложных) *обозначений*, например:

тк09/ = /пр12/пр13/
пр10/ = /тк03/YX K02/
кр03/ = /тк05/R06/,

и вычисляются при помощи канонических процедур транслятора.

Элементарные и составные обозначения образуют первую часть языковой программы — *группу обозначений*, где каждое обозначение является самостоятельной (независимой от соседних обозначений) инструкцией для электронной машины.

Вышеописанная запись обозначений была получена исходя из требований человека, а не ЭВМ. Для преобразования информации, записанной на языке «субчеловеческого» уровня, в информацию на машинном уровне требуется несколько промежуточных вспомогательных языков. Эта процедура называется в САП кодированием и расстановкой информации и осуществляется первым блоком транслятора. Если в первоначальной записи составные обозначения требуют по нескольку (до пяти) ячеек каждое и содержат избыточную информацию, то в кодах обозначений, т. е. в окончательном языке записи обозначений в ОП, избыточная информация ликвидирована и обозначения записаны друг за другом, как в ЗУ с ячейками изменяющейся длины.

Для осуществления вычислений элементов траектории движения объекта по некоторому универсальному алгоритму все геометрические элементы независимо от способа их определения (на входном языке) должны быть приведены к стандартной форме, т. е. канонизированы. Эта процедура производится вторым блоком транслятора.

Этим заканчивается обработка численной входной информации, отделенной в случае САПа (в отличие от АПРОКСа) от информации, описывающей траектории обработки.

Второй частью языковой программы САП, описывающей траекторию и режим обработки, является *строка обхода*. Она состоит из последовательности команд, составленных, в свою очередь, из слов языка САП:

$$\langle\text{команда}\rangle ::= \langle\text{основная команда}\rangle, \mid \langle\text{вспомогательная команда}\rangle.$$

Основными называются команды, определяющие геометрические элементы на траектории движения центра режущего инструмента; все остальные (технологические, уточняющие и т. п.) команды — вспомогательные.

Надо еще раз подчеркнуть, что, исходя из принципа проблемной ориентации, синтаксис и семантика языка приспособлены в первую очередь к потребителю — человеку, неквалифицированному в области вычислительной техники. Поэтому задаваемое им описание траектории обработки должно как можно меньше отличаться от такового, написанного на естественном языке. Для реализации этого язык построен таким образом, что команды строки обхода не могут рассматриваться как автономные, независимые друг от друга инструкции для ЭВМ (аналогично тому, как слова предложения любого естественного языка не могут без их взаимных связей служить для определения точного смысла этого предложения). Для вычисления любой требуемой эквидистанты заданного контура и опорных точек на ней, а также для формирования выходной информации в виде приращений транслятор одновременно оперирует тремя соседними основными командами (и вспомогательными командами, заключенными между ними).

Все это осуществляется в IV блоке транслятора, куда подается как строка обхода, так и канонизированная числовая информация (выход III блока).

Последний, пятый блок формирует кадры — определенные порции информации — для специальной управляющей машины процесса — интерполятора [8].

Структура и правила составления языковой программы (ЯП) системы АПРОКС существенно отличаются от уже описанной САП. Прежде всего здесь $\mathfrak{B} \neq \emptyset$, а множество $\mathfrak{A}_1$ содержит гораздо меньше цифровых данных, чем в случае САП, и в ЯП нецелесообразно отделять числовую информацию от буквенной или строчной.

Языковая программа АПРОКС содержит две части. Первая часть состоит из *операторов размещения*, определяющих месторасположение деталей по их замеренным координирующим точкам на листе, и из *операторов совмещенного реза*, поправляющих априорное расположение координирующих точек деталей, имеющих участок, вырезаемый совмещенным резом вместе с участком другой детали. Вторая часть ЯП служит для определения маршрута разрезки листа и технологических параметров резки. В случае отсутствия операторов совмещенного реза операторы размещения могут быть записаны во вторую часть и тогда первая часть может оказаться пустой.

О простоте изобразительных средств языка, а также совпадении синтаксических правил с правилами естественного так наз. вербального мышления говорят хотя бы следующие примеры. Оператор размещения

ДЕТ К 22 = ДЕТ К 4 СИММ Y (720; 605/1134; 180)

означает, что деталь с порядковым номером 22 (на карте раскроя) определена геометрическими данными детали с порядковым номером 4 и симметрична ей относительно оси Y. В скобках указаны координаты то-

чек, координирующих локальную систему координат детали № 22 с общей системой координат листа раскроя. Оператор совмещенного реза

СОВМЕЩ {ДЕТ К5, УЧ(ТК 10/ТК 11) → ДЕТ К4, УЧ(ТК 5/ТК 6)}

означает, что участок детали № 5, находящийся между точками № 10 и 11, и участок детали № 4, находящийся между точками № 5 и 6, вырезается одним совмещенным резом и что для этого месторасположение детали № 5 подлежит уточнению по указанному участку детали № 4.

Вторая часть ЯП определяется *операторами управления движением* и *технологическими операторами*. Такие операторы управления движением, как

ОТ : ТК (A; B)
или ДО : ДЕТ К N

не нуждаются в пояснении, а оператор подхода

ДО : ДЕТ К 10, ТК(УЧ 3/УЧ 4) [ПРОБ : НА]

означает, что резаку необходимо подойти к детали № 10, точнее — к такой ее точке, которая является пересечением участков № 3 и 4; после подхода в этой же точке необходимо пробить металл.

Если только отметить, что через S(N) обозначается скорость движения режущего инструмента, а ПОВТ означает повторение, то уловить смысл следующего фрагмента ЯП не представляет трудности даже для неподготовленного читателя:

. . . . . .
РЕЗ : {S (2100)}
ПОВТ : ДЕТ К2 СИММ Х (350; 1120/1650; 1120)
ОТ : ТК (УЧ 6/УЧ 7) [ПРОБ : ВНЕ]
ДО : ТК (УЧ 1/УЧ 2) [ПОДЪЕМ, ПЕРЕХ]
ДО : ДЕТ К 6
. . . . . .

Для преобразования языковой программы в машинные коды и дальше, учитывая данные множества $\mathfrak{B}_1$, в кадры для интерполятора транслятор системы (как и в случае САПа) использует несколько уровней промежуточных языков. После ввода ЯП она просматривается дешифратором, который выделяет слова, знаки и цифры. На основе этого определяется класс каждого оператора, а затем производится уже формирование всех операторов (рис. 2). Результатом первого этапа составления маршрута является последовательность строго кодированных ячеек ЗУ — массив $\{v\}$. Далее следует формирование так наз. условных кадров. Это происходит на основе массива $\{v\}$ с учетом геометрических данных о деталях карты раскроя и их участках (множества $\mathfrak{B}_1$). Каждый условный кадр состоит из ряда ячеек, где записаны код участка и число перемычек на участке, расположение резака относительно участка, числовые величины опорных точек, дополнительные условные знаки и т. п.

Условные кадры, как и геометрические данные стандартных вырезов и технологические таблицы, служат исходными данными для второй части транслятора, формирующей кадры для управления интерполятором.

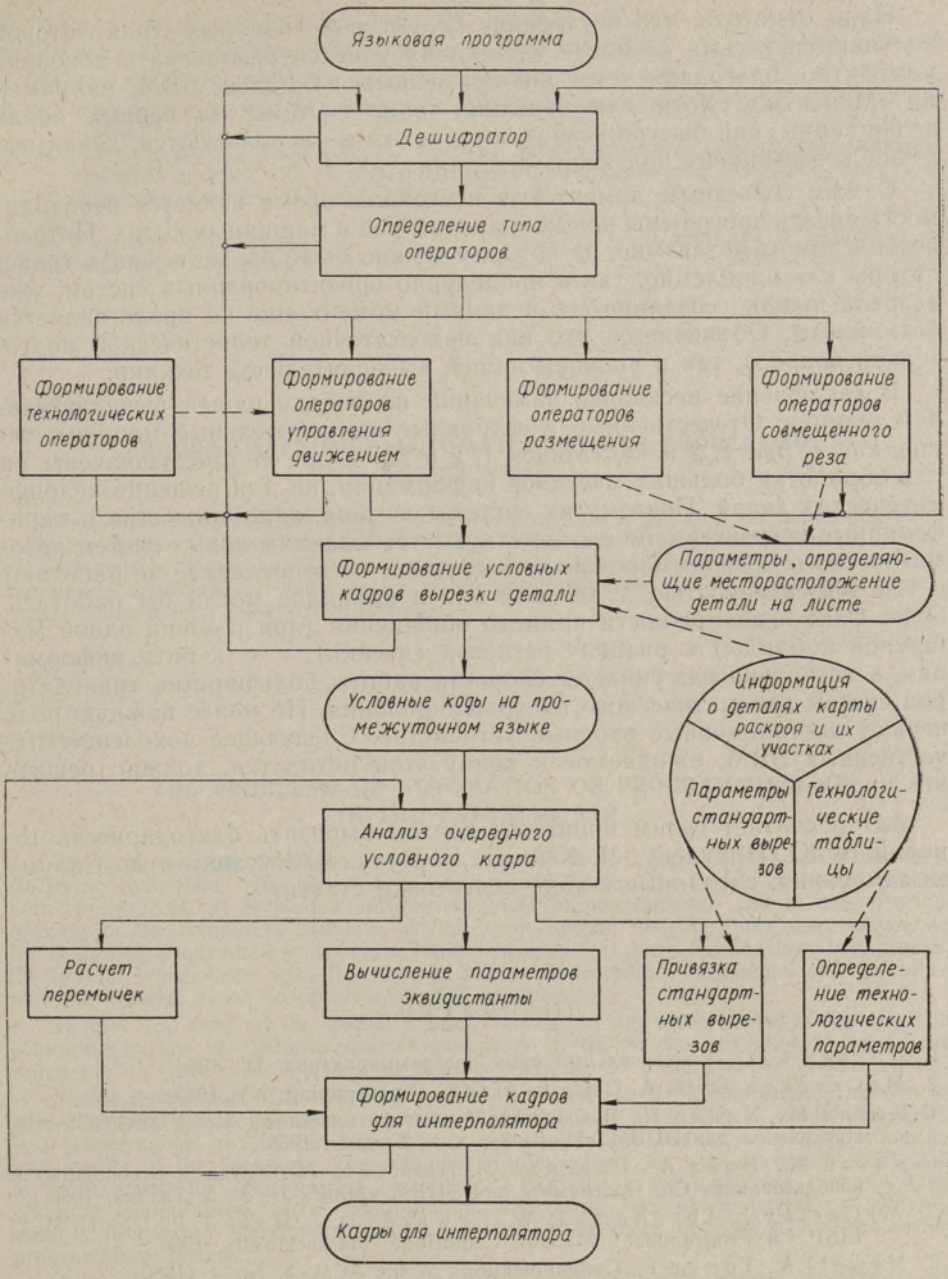


Рис. 2.

Все условные кадры просматриваются по очереди и в зависимости от их характера либо производится вычисление эквидистантных отрезков, либо расчет и распределение перемычек, либо привязка к контурам стандартных вырезов, либо определение технологических параметров. Все этапы, разумеется, сопровождаются контрольными просчетами. Последовательность кадров интерполятора составляет управляющую программу, которая вводится в управляемый объект.

Надо отметить, что внутренние структуры подобных трансляторов получаются весьма сложные, однако их удается реализовать довольно компактно, благодаря чему они применимы на малых ЭВМ, например на «Минск-22». Хотя для практика такие системы на первых порах неожиданны, они быстро осваиваются и охотно используются, давая высокий коэффициент полезного действия.

Самым серьезным моментом в настоящее время является необходимость писать программы новых трансляторов в машинных кодах. Потребность в таком метаязыке, на котором можно было бы записывать трансляторы как проблемно, так и процедурно ориентированных систем, уже назрела, однако создание его в данный момент еще не представляется возможным. Объясняется это как недостаточной теоретической подготовкой вопроса, так и уровнем нашей вычислительной техники.

В заключение несколько замечаний по поводу нашей вычислительной техники. Отечественные электронные вычислительные машины, такие, как «Урал 1, 2 и 4», «Минск 1, 2 и 22» и др., не предназначены ни для обработки больших массивов информации, ни для решения неарифметических задач. Поэтому их системы команд, запоминающие и периферийные устройства не соответствуют требованиям новых систем автоматического программирования, вследствие чего последние не работают с той эффективностью, с которой они в принципе могли бы работать. Один факт невозможности прямого обращения (при помощи одной машинной команды) к разряду регистра (ячейки), т. е. к биту информации, в несколько раз снижает скорость работы большинства трансляторов систем автоматического программирования. Не менее важную роль играют дистанционные входные устройства. Следующее поколение отечественных ЭВМ, ожидаемое к концу этой пятилетки, должно решить эти проблемы.

Автор считает своим приятным долгом выразить благодарность товарищам Ю. Пруудену, Л. Хейнла, Р. Пукк, В. Куузику и Х. Салуму за замечания, сделанные в ходе подготовки статьи.

ЛИТЕРАТУРА

1. Лавров С. С., Универсальный язык программирования, М., 1964.
2. McCracken D. D., A Guide to ALGOL Programming, NY, 1962.
3. Котли М., Ханко П., Руководство по алгоритмическому языку MALGOL, Сб. Программы для ЭЦВМ «Минск-2», № 4, Таллин, 1966.
4. Котли М., Вийл А., Рахенди М., Транслятор MALGOLa и руководство к использованию, Сб. Программы для ЭЦВМ «Минск-2», № 5, Таллин, 1966.
5. McCarthy J., Abrahams P. W., Edwards D. J., Hart T. P., Levin M. I., LISP 1,5 Programmer's Manual, Cambridge, Massachusetts, 1966.
6. Newell A., Tonge F., Communications of the ACM, **3**, No. 4 (1960).
7. Bates E. A., Automatic Programming for Numerically Controlled Tools — APT III, Computers Application, 140—156 (1962).
8. Тамм Б. Г., Прууден Ю. И., Прууден Э. В., Автоматизация подготовки программ для металлорежущих станков с помощью ЭВМ, Таллин, 1966.
9. Тамм Б. Г., Автоматика и телемеханика, **22**, № 8, 1038—1054 (1961).
10. Пукк Р., Ж. выч. матем. и математич. физ., **5**, № 2, 185—198 (1965).
11. Корьюс А., Тр. Вычислительного центра Тартуск. гос. ун-та, № 5 (1965).
12. Салум Х., Изв. АН ЭССР. Сер. физ.-матем. и техн. наук, **14**, № 3, 464—472 (1965).
13. Куузик В., Ж. выч. матем. и математич. физ., **5**, № 3, 571—574 (1965).
14. Fenves S. J., Appl. Mech. Revs, **18**, No. 3, 175—180 (1965).

15. Barton R. S., AFIPS Conf. Proc., **23**, 169—178 (1963).
16. Zemanek H., Communs ACM, **9**, No. 3, 139—143 (1966).
17. Ющенко Е. Л., Адресное программирование, Киев, 1963.
18. Куликович А. Е., Ющенко Е. Л., Кибернетика (АН УССР), № 2, 3—8 (1965).
19. Прууден Ю. И., Тамм Б. Г., Система автоматического программирования обработки деталей, Сб. Методы подготовки информации для станков с программным управлением, Таллин, 1963, с. 55—70.

Институт кибернетики
Академии наук Эстонской ССР

Поступила в редакцию
19/IV 1967

B. TAMM

PROGRAMMEERIMISE AUTOMATISEERIMISE PROBLEEMID EESTI NSV-s

Need probleemid kerkisid vabariigis 60-ndate aastate algul ning on taandunud põhiliselt protseduur- ja probleemorientatsiooniga automaatsete programmeerimissüsteemide väljatöötamisele. Meie vabariigis loodud neisse kahte klassi kuuluvate programmeerimissüsteemide analüüsile on pühendatud artikli põhilised osad. Lisaks sellele kommenteeritakse lühidalt teisi sel alal tehtud töid.

B. TAMM

THE PROBLEMS OF AUTOMATION OF PROGRAMMING IN THE ESTONIAN SSR

The programming problems in Estonia arose in the early sixties after the first electronic computers were installed in Tallinn and Tartu. They were connected mainly with two aspects: a) lack of a sufficient number of programmers and b) the effective running of the computers. Because of that some special investigations were organized for designing programming systems. These investigations were divided into two groups, the first of which dealt with the systems based on procedural languages and the other with the systems based on problem-oriented languages.

In the first part of the paper, a brief review of the works done in Estonia and belonging to the two aforementioned groups is given. At the end of that part some comments on three other works (not clearly belonging to these groups) are presented.

In the second part of the paper, a comparative analysis of the two already introduced procedure-oriented programming systems MALGOL and VELGOL is made. MALGOL is a proper subset of ALGOL, but with sufficiently simplified features both in semantics and syntax of the language. The compiler of the system, as well as that of VELGOL is programmed for Minsk series computers. VELGOL is developed for economic and planning computations where manipulation with large sets of numerical or symbolic data is necessary; therefore it may be classified as more or less a list processing programming system.

In the third part of the paper, problem-oriented programming languages are discussed. Two of these: SAP for numerically controlled machine tools and APROKS for flame cutters (as well as the systems mentioned above) were worked out at the Institute of Cybernetics of the Academy of Sciences of the Estonian SSR and successfully introduced into the industry. The paper gives a brief review of these two systems and a more general or theoretical point of view for analysing programming systems with problem-oriented languages.