СООБЩЕНИЯ
ОБЪЕДИНЕННОГО
ИНСТИТУТА
ЯДЕРНЫХ
ИССЛЕДОВАНИЙ
Дубна

11 - 6738

ЛАБОРАТОРИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ
И АВТОМАТИЗАЦИИ

С.Х.Бычваров

КОНКРЕТНЫЙ ЯЗЫК ОПИСАНИЯ ПРАВИЛ УМОЛЧАНИЯ

1972

11 - 6738

С.Х.Бычваров

КОНКРЕТНЫЙ ЯЗЫК ОПИСАНИЯ ПРАВИЛ УМОЛЧАНИЯ

Неформальное описание правил умолчания в работах /4,5,6/ указывает на необходимость создания метода их описания. Основные причины разработки абстрактного языка описания правил умолчания /1/ следующие: а/ неоднозначность понимания; б/ отсутствие единого подхода; в/ существование языков программирования со сложной системой правил умолчания.

Абстрактный язык описания правил умолчания имеет ряд применений. Отметим только два из них: описание правил умолчания в потребительских руководствах и разработку конкретных языков описания правил умолчания.

В данной работе предлагается вариант конкретного языка описания правил умолчания. Программа на конкретном языке определяет алгоритм расширения допустимого множества элементов. Язык удобен для использования в системах автоматизации процесса создания программного обеспечения ЭВМ.

Синтаксис конкретного языка здесь описывается с помощью формализма, используемого при описании синтаксиса *PL/1* в работах /4,5,6/ с единственным расширением, взятым из работы /3/ - знак "::=".

1. Оператор SUBROUTINE

1.1. Синтаксис

оператор_ *SUBROUTINE* ::= *SUBROUTINE* идентификатор;
идентификатор ::= буква [буква | цифра] символ_разбивки
буква | символ_разбивки цифра] ...
буква ::= | *a* | *b* | *c* | *d* | *e* | *f* | *g* | *h* | *i* | *j* | *k* | *l* | *m* | *n* | *o* | *p* | *q* | *r* |
s | *t* | *u* | *v* | *w* | *x* | *y* | *z* | *A* | *B* | *C* | *D* | *E* | *F* | *G* | *H* | *I* | *J* | *K* | *L* |
M | *N* | *O* | *P* | *Q* | *R* | *S* | *T* | *U* | *V* | *W* | *X* | *Y* | *Z* |

цифра ::= | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
символ_разбивки ::= _

1.2. Семантика

Оператор *SUBROUTINE* является заглавным оператором подпрограммы. Идентификатор задает имя подпрограммы.

1.3. Примеры

```
SUBROUTINE Minor-structure;
SUBROUTINE Major-structure-level-one-variable;
```

2. Оператор ALT

2.1. Синтаксис

оператор_ *ALT* ::=*ALT* совокупность_множеств_альтернативных_ элементов ;
совокупность_множеств_альтернативных_элементов ::= | совокупность_множеств | множество |
совокупность_множеств ::= | [семейство_множеств]... семейство_ множеств |
семейство_множеств ::= | идентификатор | совокупность_множеств| [,идентификатор | , совокупность_множеств]...
множество ::= идентификатор [, идентификатор]...

Замечание: Знак " [" /" | "/ используется для обозначения символа " | " /" | "/ из конкретного языка описания правил умолчания.

2.2. Семантика

Оператор *ALT* определяет совокупность множеств альтернативных элементов. Идентификатор используется для обозначения элемента множества. Допустимое множество не может содержать такие два элемента, которые принадлежат некоторому множеству альтернативных элементов.

Везде, где необходимо, элемент будет рассматриваться как совокупность множеств, состоящая из единственного множества, причем последнее содержит только этот элемент.

Запятая ' , ' обозначает выполнение следующих действий:

а/ найти декартово произведение совокупностей;

б/ поставить в соответствие любой упорядоченной паре множеств их объединение.

Знак ' | ' обозначает операцию объединения совокупностей. Порядок выполнения операций ' , ' и ' | ' определяется расстановкой фигурных скобок и соблюдением старшинства операций /операция ' , ' старше, чем операция ' | '/.

2.3. Пример

Оператор

```
ALT {INTERNAL, EXTERNAL|AUTOMATIC,BASED,DEFINED,
    {CONTROLLED, STATIC | EXTERNAL}| SECONDARY};
```

определяет следующие четыре множества альтернативных элементов:

```
{INTERNAL, EXTERNAL}

{AUTOMATIC, BASED, DEFINED,CONTROLLED,STATIC}

{AUTOMATIC, BASED, DEFINED, EXTERNAL}

{SECONDARY}  ,
```

где фигурные скобки использованы в смысле теории множеств.

3. Оператор ADT

3.1. Синтаксис

оператор_ *ADT*::= множество;

3.2. Семантика

Оператор *ADT* определяет множество дополнительных элементов. Идентификатор используется для обозначения элемента множества. Дополнительный элемент не добавляется/ в силу правил умолчания/.

3.3. Примеры

Операторы

```
ADT SECONDARY;  и
ADT ALIGNED, UNALIGNED, Dimension;
```

определяют следующие множества дополнительных элементов:

{ SECONDARY } и
{ ALIGNED, UNALIGNED, Dimension }

4. Оператор ADD

4.1. Синтаксис

оператор_ *ADD*::= совокупность_допустимых_множеств
ADD совокупность_одноэлементных_множеств;
совокупность_допустимых_множеств ::= [множество | совокупность_
множеств]
совокупность_одноэлементных_множеств ::=
{ идентификатор | совокупность_множеств_из_одного_эле-
мента }
совокупность_множеств_из_одного_элемента ::=
{ [идентификатор ⊥] ... идентификатор }

4.2. Семантика

Оператор *ADD* определяет правило /которое иногда будем называть правилом умолчания/ добавления новых элементов к допустимому множеству /расширения допустимого множества/. Идентификатор обозначает элемент множества.

Пусть совокупности допустимых и одноэлементных множеств упорядочены согласно следующим правилам /которые применяются в процессе получения совокупностей/:

а/ $\{a_1, a_2, \ldots, a_m\} \cup \{b_1, b_2, \ldots, b_n\} = \{a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_n\}$

б/ $\{a_1, a_2, \ldots, a_m\} \times \{b_1, b_2, \ldots, b_n\} = \{(a_1, b_1), \ldots, (a_1, b_n), (a_2, b_1), \ldots, (a_2, b_n), \ldots, (a_m, b_1), \ldots, (a_m, b_n)\}$,

где " × "обозначает операцию декартова произведения.

Оператор *ADD* называется простым, если совокупность допустимых множеств, а также совокупность одноэлементных множеств содержит только одно множество. Если левая часть оператора *ADD* пустая, то это обозначает, что совокупность допустимых множеств содержит только пустое множество. Оператор *ADD* определяет последовательность простых операторов следующим образом: при фиксированном множестве из совокупности допустимых множеств пробегаются все множества совокупности одноэлементных множеств. В последовательности простых операторов *ADD* результат данного простого оператора *(Y)* является аргументом *(X)* для следующего простого оператора этой последовательности.

Простой оператор *A ADD b* / где *A* множество, *b* элемент и $b \notin A$ / преобразует допустимое множество *X* в допустимое множество *Y* следующим образом: если $A \subset X$, $b \notin X$ и $X \cup \{b\}$ допустимое множество, то $Y = X \cup \{b\}$; в противном случае $Y = X$.

4.3. Пример

Оператор

```
ADD {AUTOMATIC | INTERNAL| STATIC};
```

определяет следующую последовательность простых операторов:

```
ADD  AUTOMATIC;
ADD  INTERNAL;
ADD  STATIC;
```

Пусть задан следующий оператор *ALT*:

```
ALT {AUTOMATIC,BASED,CONTROLLED,STATIC,DEFINED|
    INTERNAL, EXTERNAL| AUTOMATIC,BASED,DEFINED,
    EXTERNAL};
```

Тогда, например, множество $X = \{\}$ допустимое и оно преобразуется оператором *ADD* в множество $Y = \{AUTOMATIC, INTERNAL\}$; Аналогично, если $X = \{EXTERNAL\}$, то $Y = \{EXTERNAL, STATIC\}$ и т.д.

5. Оператор END

5.1. Синтаксис

оператор_ *END* ::= *END* [идентификатор];

5.2. Семантика

Оператор *END* замыкает подпрограмму или основную программу. Идентификатор задает имя замыкаемой подпрограммы или основной программы.

5.3. Примеры

```
END;

END Major-structure-level-one-variable;

END MASTER;
```

6. Подпрограмма

6.1. Синтаксис

подпрограмма ::= оператор_*SUBROUTINE* оператор_ *ALT*
[оператор_ *ADT*] [оператор_ *ADD*]... оператор_*END*

6.2. Семантика

Операторы *ALT* и *ADT* определяют необходимую информацию для выполнения операторов *ADD*. Операторы *ADD* выполняются последовательно и преобразуют результат предыдущего оператора *ADD* /если таковой имеется/. Подпрограмма корректна, если она преобразует любое допустимое множество в множество, к которому нельзя добавить недополнительный элемент, указанный в операторе *ALT*. Критерии корректности обоснованы в /1/.

6.3. Пример

```
SUBROUTINE Major-structure-level-one-variable;
ALT {AUTOMATIC,BASED,CONTROLLED,STATIC,DEFINED |
     INTERNAL,EXTERNAL | AUTOMATIC,BASED,DEFINED,
     EXTERNAL | SECONDARY} ;
ADT  SECONDARY;
ADD {AUTOMATIC | INTERNAL | STATIC} ;
END Major-structure-level-one-variable;
```

Эта подпрограмма, например, преобразует допустимое множество {*STATIC*} в множество {*STATIC , INTERNAL* } .

7. Оператор PROCEDURE

7.1. Синтаксис

оператор_ *PROCEDURE*::= *PROCEDURE* [идентификатор] ;

7.2. Семантика

Оператор *PROCEDURE* является заглавным оператором основной программы. Идентификатор задает имя основной программы.

7.3. Пример

```
PROCEDURE MASTER;
```

8. Оператор вызова

8.1. Синтаксис

```
оператор_вызова   ::= [ ненулевая_целая_десятичная_константа ]
      идентификатор [ совокупность_множеств ] ;
ненулевая_целая_десятичная_константа  ::= [ цифра ] ...
      ненулевая_цифра [ цифра ]...
ненулевая_цифра  ::= { 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 }
```

8.2. Семантика

Идентификатор определяет имя подпрограммы, совокупность множеств определяет совокупность управляющих множеств, а ненулевая целая целая десятичная константа - номер уровня.

Оператор вызова служит для обращения к подпрограмме. При выполнении определенных условий он вызывает передачу управления на ту подпрограмму, имя которой указано в операторе. Если оператор вызова выполняется применительно к допустимому множеству X , то существует множество из совокупности управляющих множеств, которое является подмножеством X.

Отсутствие совокупности множеств обозначает, что совокупность управляющих множеств содержит только пустое множество, а отсутствие ненулевой целой десятичной константы обозначает, что номер уровня равняется 1.

8.3. Примеры

```
Minor-structure {{ I-N | N_I_N } , Minor-structure } ;

02 Major-structure-parameter { Parameter } ;

02 Major-structure-level-one-variable;
```

9. Основная программа

9.1. Синтаксис

основная_программа ::= оператор_*PROCEDURE* { оператор_вызова }...
оператор_ *END*

9.2. Семантика

Основную программу можно изобразить в виде дерева. Между его вершинами/за исключением корня/ и операторами /вызова/ основной программы можно установить взаимно однозначное соответствие, которое обладает следующими свойствами:

а/ пусть кусок основной программы $n_1 s_1 ; n_2 s_2 ; \ldots ; n_k s_k ;$ /где n_i - номер уровня, а s_i - оператор вызова / такой, что $k \geq 2 ; n_1 < n_i$ / $2 \leq i \leq k-1$ / и $n_1 \geq n_k$. Тогда поддерево с корнем s_1 содержит только вершины $s_1 , s_2 , \ldots , s_{k-1}$.

б/ Если в основной программе оператор s_i встречается раньше оператора s_j , то в дереве они либо принадлежат одному и тому же пути, либо путь, содержащий оператор s_i , находится левее пути, содержащего оператор s_j .

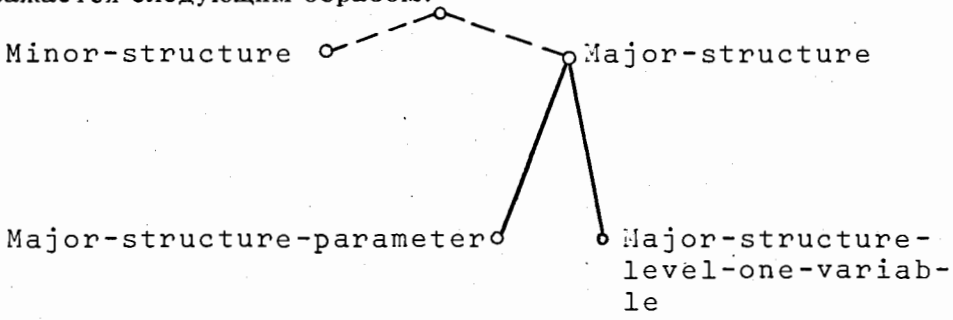
Пусть дано допустимое множество X . Находим самый левый путь дерева такой, что для всех принадлежащих ему операторов вызова соответствующая совокупность управляющих множеств содержит множество, которое является подмножеством X. Выполнение основной программы является последовательным выполнением операторов вызова указанного пути. Положим по определению, что такой путь всегда существует для допустимого множества.

9.3. Примеры

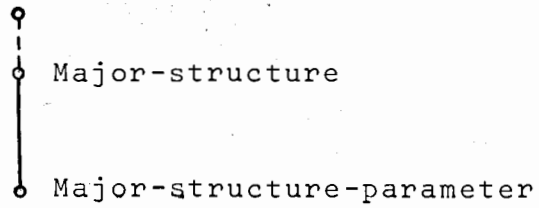
а/ кусок основной программы

```
    :
    Minor-structure {{ I_N | N_I_N } , Minor-structure };
    Major-structure {{I_N | N_I_N } , Major-structure } ;
       02 Major-structure-parameter { Parameter } ;
       02 Major-structure-level-one-variable;
       01 ...
```

изображается следующим образом:



Если допустимое множество $X = \{ I - N$, *Major-structure* , *Parameter* , *ALIGNED* , *Dimension* $\}$, то искомый путь будет:



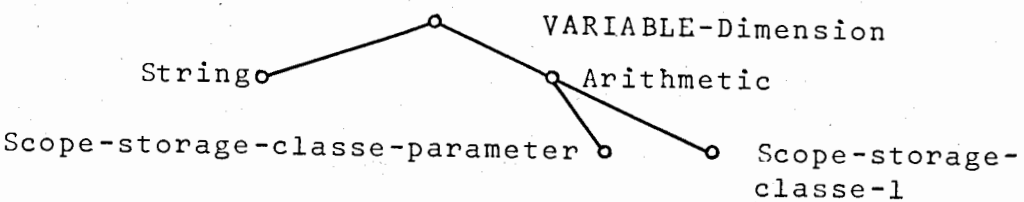
При выполнении основной программы будут последовательно исполнены следующие операторы:

```
Major-structure {{I_N |N_I_N } , Major-structure } ;

    02 Major-structure-parameter { Parameter } ;
```

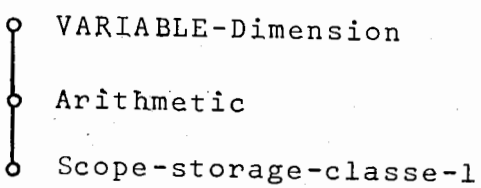
б/ кусок основной программы

```
.
.
.
VARIABLE-Dimension;

02 String {{ I_N | N_I_N } , {CHARACTER | BIT |
    VARYING | PICTURE_C }} ;

02 Arithmetic;

       03 Scope-storage-classe-parameter
          { Parameter } ;

       03 Scope-storage-classe-1;

01 ...
```

изображается следующим поддеревом:



Если допустимое множество $X = \{\ \}$, то искомый путь будет:



При выполнении основной программы будут последовательно исполнены следующие операторы:

```
VARIABLE-Dimension;

    02 Arithmetic;

    03 Scope-storage-classe-1;
```

10. Программа

10.1. Синтаксис

программа ::= основная_программа { подпрограмма } ...

10.2. Семантика

Выполнение программы заключается в выполнении основной программы и соответствующих подпрограмм. Положим по определению, что все элементы допустимого множества указаны в соответствующих подпрограммах /к которым обращаются операторы искомого пути/.

10.3. Пример

Рассмотрим следующую простую программу:

```
PROCEDURE MASTER;
Minor-structure {{I_N | N_I_N } , Minor-structure } ;
Major-structure {{ I_N |N_I_N } , Major-structure } ;

    02 Major-structure-parameter {Parameter } ;

    02 Major-structure-level-one-variable;

END MASTER;
```

```
SUBROUTINE  Minor-structure;
ALT | ALIGNED, UNALIGNED | I_N, N_I_N | Minor-
      structure | VARIABLE | Dimension | INTERNAL | ;
ADT   ALIGNED, UNALIGNED, Dimension;
ADD | VARIABLE | INTERNAL | ;
END   Minor-structure;

SUBROUTINE  Major-structure;
ALT | ALIGNED, UNALIGNED | I_N, N_I_N | Major-
      structure | VARIABLE | Dimension | ;
ADT   ALIGNED, UNALIGNED, Dimension;
ADD VARIABLE ;
END Major-structure;

SUBROUTINE  Major- structure-parameter;
ALT | CONNECTED, CONTROLLED | Parameter | INTERNAL |;
ADT   CONNECTED, CONTROLLED;
ADD INTERNAL;
END Major-structure-parameter;

SUBROUTINE Major-structure-level-one-variable;
ALT | AUTOMATIC, BASED, CONTROLLED, STATIC,DEFINED |
      INTERNAL, EXTERNAL | AUTOMATIC, BASED,
      DEFINED, EXTERNAL | SECONDARY |;
ADT   SECONDARY ;
ADD | AUTOMATIC | INTERNAL | STATIC | ;
END Major- structure-level-one-variable;
```

Если допустимое множество X равно

{ Major-structure, I_N, SECONDARY },

то после выполнения оператора

Major-structure {{ I_N | N_I_N } , Major-structure };

/при этом происходит обращение к подпрограмме *Major–structure* / множество X преобразуется в множество

{ Major-structure, I_N, SECONDARY, VARIABLE } ,

а после выполнения оператора

02 Major- strucructure-level-one-variable;

множество

{ Major-structure, I_N, SECONDARY, VARIABLE }

преобразуется в множество Y, которое равно

{Major-structure, I_N,SECONDARY,VARIABLE,AUTOMATIC,
INTERNAL }

Следовательно, данная программа преобразует допустимое множество X в множество Y. Аналогично, например, она преобразует множество

{ I_N, Minor-structure, ALIGNED, Dimension }

в множество

{ I_N, Minor-structure,ALIGNED,Dimension,VARIABLE,
INTERNAL }

/в результате выполнения оператора

Minor-structure {{ I_N | N_I_N } , Minor-structure } /.

Работа доложена на Всесоюзном симпозиуме "Теория языков и методы построения систем программирования" /Алушта, 26 июня-1 июля, 1972 г./.

Автор выражает благодарность Н.Н.Говоруну за содействие, Е.А.Жоголеву и В.П.Ширикову за полезные обсуждения, а также Г.Л.Мазному за редакционные замечания.

Литература

1. *С.Х.Бычваров. ОИЯИ, 11-6737, Дубна, 1972.*
2. *С.Х.Бычваров. ОИЯИ, 11-6429, Дубна, 1972.*

3. Алгоритмический язык АЛГОЛ-60, пересмотренное сообщение. Москва, "Мир", 1965.
4. Универсальный язык программирования PL/1 /Перевод с английского под редакцией В.М.Курочкина. Москва, 1968/.
5. IBM SYSTEM/360 Operating System PL/1. Language Specifications Form C 28-6571-4, IBM, 1965.
6. PL/1 Language Specifications Order Number GY 33-6003-2 IBM, 1970.

Рукопись поступила в издательский отдел 26 сентября 1972 года.