Оглавление

Введение

Теоретическое программирование – математическая дисциплина, изучающая синтаксические и семантические свойства программ, их структуру, процесс их составления, преобразования и выполнения.

ПРОГРАММИРОВАНИЕ

Искусство Ремесло Наука

Программирование **- как искусство -** материализация того, чего не было.

Программирование **- как ремесло -** технология.

Программирование **- как наука –** показывает, как можно достичь множества целей, применяя строго определенные правила построения к основным конструктивным элементам.

Вводный пример: Деление x/y

```
r:=x; q:=0;
 While r>y do
  Begin
   r:=r-y;
   q:=q+1;
  End;
```

В r хранится остаток, а в q – частное

Проверим: `Write(‘y*q+r=’,y*q+r)`

Программа стала выдавать много ошибочных сообщений.

Добавим условие:

Если не выполняется `{x=y*q+r}` , то выходим из программы.

В результате оказалось, что может быть y=0.

Перед `r:=x` будем писать `{y>0}` чтобы программа работала только с положительными числами.

В результате оказалось, что данный пример срабатывает следующим образом: `x=6, y=3, q=1, r=3`.

Перепишем программу так, что r должно быть явно меньше делителя.

```
{0≤x and 0≤y}
     r:=x;  q:=0;
     while r≥q do
     begin
          r:=r-y;
          q:=q+1;
     end;
{x=y*q+r and 0≤r<q}
```

Спецификация программ

Спецификация программ – это выражение на определенном языке, возможно естественном, которое точно описывает, что должно быть в результате выполнения программ.

В спецификации может быть указан размер программы.

Спецификации бывают **декларативными** и **описательными**. Описательная спецификация описывает как должна работать программа.

Для составления спецификации программ целесообразно использовать язык предикатов.

Спецификация программы на языке предикатов выглядит следующим образом:

{Q} S {R},

где Q – предусловие, S –программа, R – постусловие .

Если выполнение S началось в состоянии, удовлетворяющем Q, то имеется гарантия, что оно завершится в конечное время в состоянии удовлетворяющем R.

Пример спецификаций:

Записать в z произведение a и b, предполагая, что a и b больше либо равны нулю.

z:=a*b

$\{a \geq 0 \wedge b \geq 0\}$ S $\{z = a * b\}$

Понять {z=a*b} можно и так: программа не должна делать ничего кроме того, что должна делать.

Пример2: сспецификация на сортировку

Даны n >= 0 и массив b[0:n-1].

Менять можно только по два элемента.

a). R: $(\forall I\text{: } 0 \leq i < n-1\text{: } b[i] \leq b[i+1])$

b). $\{n \geq 0$ and $b = B\}$ S $\{ \forall i\text{: } 0 \leq i < n-1\text{: } b[i] \leq b[i+1]$ and b = перем.(B) }

где b является перемещением от B.

Пример3: обмен значениями переменных.

$\{x = X \wedge y = Y\}$ S $\{x = Y \wedge y = X\}$

Пример4: найти максимальное значение переменных в массиве B, который изменяется от 0 до n-1:

$\{n>0\}$ S $\{(\forall i\text{: } 0 \leq i < n\text{: } b[i] \leq x) \wedge (\exists\ i\text{: : } 0 \leq i < n\text{: } x = b[i])\}$

Пример5: придать x абсолютное значение x.

$\{x = X\}$ S $\{(X \leq 0 \wedge x = -X) \vee (X \geq 0 \wedge x = X)\}$

или

$\{(x \geq 0) \wedge (x = X \vee x = -X)\}$

Пример6: придать каждому значению массива b[0:n-1] значение суммы элементов этого массива.

$\{n > 0 \wedge b = B\}$ S $\{\forall i{:}0\le i<n{:}b[i] = \sum j{:}0\le j<n{:}B[j]\}$

Пример7: в массиве a[0:n-1] отсортирован список преподавателей Политеха, в b[0:m-1] отсортирован список, получающих пособие по безработице. Найти первого “жулика”, который преподает и получает пособие.

$\{n>0 \wedge m>0\}$ S $\{0\le p<n \wedge 0\le q<m \wedge a[p]=b[q] \wedge (i,j{:}0\le i\le p \wedge 0\le j\le q{:}a[i]=b[j]) = 1\}$

Преобразователь предикатов WP (wp(S,R))

Преобразователь предикатов **wp** определяет множество всех состояний, для которых выполнение команды S, начавшейся в таком состоянии, закончится через конечное время, в состоянии, удовлетворяющем R.

Пример:

1. wp(“i:=i+1”,i≥1)=(i≥0)
   {i≥0}i:=i+1{i≥1}

2. S=”if x≥y then z:=x else z:=y”
   R: z=max(x,y)
   wp(S,R)=T

3. S=”if x≥y then z:=x else z:=y”
   R: z=y
   wp(S,R)=(x≤y)

4. S=”if x≥y then z:=x else z:=y”
   R: z=y-1 (этого быть не может)
   wp(S,R)=F – предусловием является пустое множество состояний. Никакие значения переменных не сделают z меньше y.

5. S=”if x≥y then z:=x else z:=y”
   R: z=y+1
   wp(S,R)=(x=y+1)

wp(S,R) (weakest precondition) – слабейшее предусловие в контексте S по отношению к R, поскольку оно определяет множество всех состояний таких, что выполнение, начавшееся в любом из них, закончится при истинном R.

Пример:

Условие a>0 сильнее, чем условие a≥0.

Если зафиксируем S, то получим $wp_s(R)$, т.е. из одного предиката получаем другой.

Частичная и полная коррекция

{Q} S {R} – полная коррекция.
Q { S } R – частичная коррекция.
В частичной коррекции не используется выражение “через конечное время”.
Если выполнение команды S начинается в состоянии, удовлетворяющем Q, и если оно заканчивается, то конечное состояние будет удовлетворять R.

Пример:

`T {while T do skip} T` – в формате частичной корректности.
Представляет собой бесконечный цикл, который ничего не делает.
`{T} while T do skip {T}` – данное высказывание тождественно ложно.

Некоторые свойства WP

1. **Закон исключенного чуда:**
   wp(S,F)=F
   Никакое состояние не может удовлетворять F, т.к. F представляет собой пустое множество.
   Пример: `wp(a:=a+b,a>a) = False`

2. **Закон монотонности:**
   Если $Q \to R$, то $wp(S,Q) \to wp(S,R)$.
   Если R слабее Q, то wp(S,R) слабее wp(S,Q).
   Пример: $a>0 \Rightarrow a \geq 0 \quad wp(a:=a+1,a>0) \Rightarrow wp(a:=a+1,a \geq 0)$
   $a>-1 \Rightarrow a \geq -1$
3. **Дистрибутивность конъюнкции:**
   $wp(S,Q) \wedge wp(S,R) = wp(S,Q \wedge R)$
   Пример: $\underbrace{wp(a:=a+1,a>0)}_{a+b>0} \wedge \underbrace{wp(a:=a+b,b>0)}_{b>0}$ (Левая часть)
   $wp(a:=a+b,a>0 \wedge b>0) = a+b>0 \wedge a>0)$ (Правая часть)

4. **Дистрибутивность дизъюнкции:**
   $wp(S,Q) \vee wp(S,R) \to wp(S,Q \vee R)$
   Действует только в одну сторону.
   Пример: Q – выпадение орла, R – выпадение решки, S - бросание монетки.
   `wp(бросание, орел)∨ wp(бросание, решка)=T`
   `wp(бросание, решка∨орел)=T`
   Для детерминированных случаев закон действует в обе стороны.

Команды `skip` и `abort`

Будем определять понятия в терминах wp.
`wp(skip,R)=R`
skip – ничего не делает, пустой оператор.
Пример: `wp(skip,a>5)=a>5`

`wp(abort,R)=F`
abort – команда несвоевременного завершения.
Если когда-либо выполнение программы достигает точки, когда должна выполняться команда abort, то, очевидно, что команда ошибочна и требуется аварийное завершение.

Пример: wp(abort,a=b)=False

Композиция команд

Последовательное соединение – это один из способов составления больших командных сегментов из меньших.
S_1 – команда; S_2 – команда
S_1 ; S_2 – новая команда.
wp(“S_1 ; S_2 ”,R)=wp(S_1 ,wp(S_2,R))
Пример:

wp(“skip;skip”,R)= wp(“skip”,wp(“skip”,R))= wp(“skip”,R)=R
wp(“S_1 ; (S_2 ; S_3)”,R)= wp(“(S_1 ; S_2); S_3”,R)

Пример1:
wp(“i:=i+1; j:=i-1”, i*j=0) = wp(“i:=i+1”,wp(“j:=i-1”, i*j=0)) = wp(“i:=i+1”,i*(i-1)=0)=((i+1)(i+1-1)=0)=((i+1)*i=0)=(i=0,i =-1);
Пример2:
wp(“i:=i+1; j:=j-1”, i*j=0) = wp(“i:=i+1”,wp(“j:=j-1”, i*j=0)) = wp(“i:= i+1”, i *(j-1)=0) = ((i+1)(j-1)=0) = (i=-1, j=1);
Пример3:
wp(“x:=2+y; y:=x*y”, y+x=0) = wp(“x:=2+y”,wp(“y:=x*y”, y+x=0)) = wp(“x:=2+y”,x*y+x=0) = ((2+y)*y+2+y)=0) = (y=-1, y=-2);

Команда присваивания

Присваивание простым переменным

x:=e, где **e** может быть выражением и **e** вычислимо.
wp(“x:=e”,R)=domain(e) cand R^x_e ,
  где cand – короткое and.

R^x_e - вместо x подставляем e.
Можно записать так: wp(“x:=e”,R)= R^x_e .
Примеры:

1. wp(“x:=5”,x=5)=(5=5)=T
2. wp(“x:=5”,x≠5)=(5≠5)=F
3. wp(“x:=x+1”,x<0)=(x+1<0)=(x<-1)
4. wp(“x:=x*x”,x^4=10)=($(x*x)^4$=10)=(x^8=10)
5. wp(“x:=a/b”,p(x))=((b≠0)∧p(a/b))
6. wp(“x:=e”,y=c)=(y=c)
7. wp("x:=e",y=c)=(y=c)

Выполнение присваивания может изменять лишь ту переменную, которая стоит в его левой части.

Обмен местами значений двух переменных:
wp(“t:=x; x:=y; y:=t”,x=X ∧ y=Y) =
wp(“t:=x; x:=y”, wp(“y:=t”, x=X ∧ y=Y))=
wp(“t:=x; x:=y”, x=X ∧ t=Y)=

wp(“t:=x”,wp(“x:=y”, x=X $\wedge$ t=Y))=
wp(“t:=x”, y=X $\wedge$ t=Y)=(y=X $\wedge$ x=Y)

Кратные присваивания простым переменным

$x_1, x_2, \ldots, x_n := e_1, e_2, \ldots, e_n$.

Пример: x,y:=y,x – поменять местами значения двух переменных.

$\bar{x} := \bar{e}$

Алгоритм присваивания:

- сначала вычисляются все выражения с e_1 до e_n.
- в результате получаем некие значения $v_1, v_2, \ldots, v_n$.
- после этого x_1 присваиваем значение v_1,
  x_2 присваиваем значение v_2,
  ….
  x_n присваиваем значение v_n.

Пример:

x,x:=1,2
x:=2.

Определяем через wp:

wp(“ $\bar{x} := \bar{e}$ ”,R)=domain($\bar{e}$) cand $R^{\bar{x}}_{\bar{e}}$.

domain($\bar{e}$)=(∀i:domain(e_i))

Пример:

1. wp(“x,y:=y,x”,x=X,y=Y)=(x=X∧y=Y)$^{x,y}_{y,x}$=(y=X $\wedge$ x=Y)
2. wp(“z,y:=z*x,y-1”,y≥0 $\wedge$ z*x^y=c)=((y-1≥0) $\wedge$ (z*x*x^{y-1}=c))=
   =((y≥1) $\wedge$ (z*x^y=c))
3. wp(“s,i:=s+b[i],i+1”, i>0 $\wedge$ s=$\sum_j$: 0≤j<i : b[j])

Поиск решений – утверждений с помощью WP

Даны 3 переменных, между которыми выполняется соотношения i ≤ m < i+p.
Дан массив: b[i : i+p-1]
Нужно массив сократить так чтобы i:=m+1, но правая граница должна остаться на месте.
Нужно найти чему равно P.
Запишем задачу спецификаций
c – правая граница
{i+p=c} i,p:=m+1,x {i+p=c}
wp(“i,p:=m+1,x”,i+p=c) = (m+1+x=c)

$$\begin{cases} \text{i+p=c <= Q} \\ \text{m+1+x=c <= wp} \end{cases}$$

c+p=m+1+x
x=i+p-m-1
тогда i,p:=m+1, i+p-m-1.
Проблемы возникают когда ставятся 2 присваивания.
{T} a:=a+1;b:=x {a=b}
x попадает в зависимость от a.
Решение нужно искать в виде

wp(“a:=a+1; b:=x(a,b)”,a=b) = wp(“a:=a=1”,a=x(a,b))=
=(a+1=x(a+1,b))=(x(a,b)=a)

Присваивание элементу массива

b[i]:=e обозначим b:=(b;i:e) – функциональная запись массива (берем b и по индексу i присваиваем e).

wp(“b[i]:=e”,R)=wp(“b:=(b;i:e)”,R)=domain((b;i:e) cand $R^{b}_{(b,i:e)}$ =

=domain(“b[i]:=e”,R)=domain(e) cand inrange(b,i) cand $R^{b}_{(b,i:e)}$,

i в границах b

Примеры:

1. wp(“b[i]:=5”,b[i]=5)=wp(“b=(b;i:5)”,b[i]=5)=((b;i:5)[i]=5) = (5=5)=T

2. wp(“b[i]:=5”, b[i]= b[j]) =
((b;i:5)[i]=(b;i:5)[j])=
$(((i=j) \lor (5=5)) \land ((i \neq j) \lor (5=b[j])))=$
$(((i=j) \land T) \land ((i \neq j) \lor (5=b[j])))=$
$((i=j) \lor (i \neq j)) \land ((i=j) \lor (5=b[j]))=$
$(T \land ((i=j) \lor (5=b[j])))=$
$((i=j) \lor (5=b[j]))$

3. wp(“b[b[i]]:=i”,b[i]=i)=
((b;b[i]:i)[i]=i)=
$(i=b[i] \land i=i) \lor (i \neq b[i] \land b[i]=i) = (i=b[i])$

Команда выбора

Задача: вычисление абсолютной величины x.

```
If x ≥ 0 then z:=x else z:=-x
  if
        x ≥ 0 → z:=x
     □  x ≤ 0 → z:=-x
  fi
```

```
if
    B₁ → S₁
   □ B₂ → S₂
   ....
   □ Bₙ → Sₙ
fi
```

$B \to S$ – охраняемая команда.

B – служит охраной входа (→) - условие

S – выполняется только при истинности B.

Знак нестрогого неравенства ставится т.к. при x=0 выполняется одна из команд, но не известно какая.

Введем обозначения $BB=B_1 \vee B_2 \vee \dots \vee B_n$.
Все условия должны быть вычислены.
Аварийная остановка:
- если не выполнилось ни одно выражение
- какое-либо выражение не вычислимо.

Определение

$wp(IF,R)=domain(BB) \wedge BB \wedge$
$\quad \wedge (B_1 \Rightarrow wp(S_1,R) \wedge B_2 \Rightarrow wp(S_2,R) \wedge \dots . B_n \Rightarrow wp(S_n,R))$

$wp(IF,R) = (\exists\ i:1\le i \le n:B_i) \wedge (\forall\ i:1\le i \le n:B_i \Rightarrow wp(S_i;R))$

$wp(IF,R) = BB \wedge B_1 \Rightarrow wp(S_1,R) \wedge B_2 \Rightarrow wp(S_2,R)$

```
    if
        B₁→S₁
        B₂→S₂
    fi
    BB=B1∨B2
```

Находим абсолютные величины.

```
Wp("if x≥0→z:=x   x≤0→z:=-x fi",z=abs(x))=
   =(x≥0∨x≤0)∧                          //BB
   ∧(x≥0⇒wp("z:=x",z=abs(x))) ∧    //B₁→wp(S₁,R)
   ∧(x≤0⇒wp("z:=-x",z=abs(x)))=   //B₂→wp(S₂,R)
=( T ∧ ( x≥0⇒x=abs(x)) ∧ ( x≤0⇒-x=abs(x)) )=(T∧T∧T)=T
```

Пример: В цикле вычисляется число положительных элементов массива B.

```
   If
        b[i]>0 → P, i:=p+1, i+1.
       b[i]<0 → i:=i+1.
   fi
   {i<n ∧ p=N j: 0≤j<i:b[j]>0}

   Wp("if" ,i≤m ∧ p=(Nj:0≤j<i:b[j]>0))=
      =(b[i]>0∨b[i]<0) ∧               //BB
   ∧((b[i]>0)⇒wp("p,i:=p+1,i+1",R))∧
   ∧(b[i]<0⇒wp("i:=,i+1",R)=
   =(b[i]=0)∧((b[i]>0)⇒i+1≤m∧p+i=(Nj:0≤j<i+1:b[j]>0))∧  ∧(b[i]<0⇒
   i+1≤m∧p=(Nj:0≤j<i+1:b[j]>0)))=
   =(b[i]=0)∧((b[i]>0)⇒i+1≤m∧p=(Nj:0≤j<i:b[j]>0))∧      ∧(b[i]<0⇒
   i+1≤m∧p=(Nj:0≤j<i:b[j]>0)))=
   = (b[i]=0)∧(p=(Nj:0≤j<i:b[j]>0))∧(i≤m)
```

Пример: x=abs(x)

```
    if
         x≥0 → skip
        x≤0 → x:=-x
```

```
fi
if x<0 then x:=-x
```

Явное появление в тексте всех охран помогает читателю. Кроме этого каждая альтернатива представлена во всех деталях. И возможность упустить из вида какую-нибудь ситуацию уменьшается.

Теорема о команде выбора

If - команда выбора, пусть Q предикат

1. $Q \Rightarrow BB$
2. $Q \wedge B_i \Rightarrow wp(S_i, R)$

тогда $Q \Rightarrow wp(If, R)$

Доказательство:

Из 2:

$\forall i: Q \wedge B_i \Rightarrow wp(S_i,R) = \forall I \ \neg Q \vee \neg B_i \vee wp(S_i,R) =$

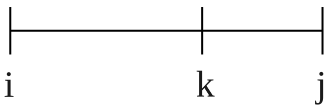
$\neg Q \vee \forall I \ \neg B_i \vee wp(S_i,R) = Q \Rightarrow \forall i \ B_i \Rightarrow wp(S_i,R).$

Складываем 1 и 2

$(Q \Rightarrow BB) \wedge (Q \Rightarrow \forall I \ B_i \Rightarrow wp(S_i,R)) =$

$=(Q \Rightarrow (BB \wedge \forall i \ B_i \Rightarrow wp(S_i,R)) = (Q \Rightarrow wp(If,R)).$

Пример – поиск методом половинного деления, нужно найти x в массиве b[0:n-1].



$b[k]>j$

$j:=k$

$Q: \ ordered(b[0:n-1]) \wedge 0 \le i<k<j<n \wedge x \in b[i:j]$

$\{Q\} if \ b[k] \le x \rightarrow i:=k \quad b[k] \ge x \rightarrow j:=k \ fi \ \{ x \in b[i:j]\}$

$Q \Rightarrow wp(If,R)$

1. $Q \Rightarrow BB = Q \Rightarrow (b[k] \le x) \vee (b[k] \ge x) = Q \Rightarrow T \equiv T$ первая посылка выполняется

2.1 $Q \wedge (b[k] \le x) \Rightarrow x \in b[k:j]$

$wp(\text{“}i:=k\text{”}, x \in b[i:j])$

2.2 $Q \wedge (b[k] \ge x) \Rightarrow x \in b[i:k]$

$wp(\text{“}j:=k\text{”}, x \in b[i:j])$

Q может использоваться в качестве предусловия.

Команда повторения

Общий вид: `While B do S`

```
do B → S od
do
    B1 → S1
    B2 → S2
  ....
```

$B_n \rightarrow S_n$

od

Повторять следующие действия пока возможно выбрать команду B_i , которая истинна, и выполнить соответствующее S_i.

do BB $\rightarrow$ if

$B_1 \rightarrow S_1$

...

$B_n \rightarrow S_n$

fi

od

Введем команду повторения через слабейшее предусловие:

$H_x(R)$ – предикат

Наш цикл завершиться за k или менее шагов при истинном R.

$H_0(R)=\neg BB \wedge R$

Не все охраны ложные, R – истина.

$H_k(R)= H_0(R)\vee wp(If, H_{k-1}(R))_{k>0}$

$H_1(R)= H_0(R)\vee wp(If, H_0(R))= H_0(R)\vee wp(If, \neg BB\wedge R)$

$Wp(DO ,R)=\exists k:0\le k: H_k(R)$

Пример: посчитать сумму элементов массива b[0:10]

```
b[0:10]
i, S:=1,b[0];
do i<11 → i, S:= i+1,S+b[i] od
```

$\{R:S=(\sum_k k : o \le k \le 11 : b[k])\}$

Сформируем предусловие

Введем предикат $P:1 \le i \le 11 \wedge S=(\sum_k k : o \le k \le i : b[k])$

Если P истина перед циклом, перед каждым шагом цикла и после него, то P истинно после цикла.

Нетрудно заметить, что

$P\wedge i\ge 11 \Rightarrow R$

{P=True}

i,S := 1,b[0];

{R} проверяем истинность P

do

  $i<11 \rightarrow \{P\wedge i<11\}$ i,S:=i+1,S+b[i]

  {P}

od

$\{I\ge 11 \wedge P\}$ цикл завершился

{R}

1. P истинно перед циклом
2. P истинно перед каждым шагом

Отсюда вывод: истинность P и ложность охран позволяет заключить, что искомый результат R получен.

Доказательство:

1. Проверим истинность P после инициализации переменных
   wp(“i,S:=1,b[0]”,P) = (1≤1≤11 ∧ b[0] = ($\sum_k k : o \le k \le 1 : b[k]$)) =
   = (1≤1≤11 ∧ b[0]=b[0]) = T
2. докажем истинность P после шага цикла, начавшемся при истинном P и истинной охране.
   wp(“i,S:=i+1,S+b[i]”,P)=
   =(1≤i+1≤11 ∧ S+b[i]=($\sum_k k : o \le k \le i+1 : b[k]$))=
   =(0≤i≤11 ∧ S=$\sum_k k : o \le k \le i : b[k]$) = T
3. P может служить предусловием
   P ∧ i≥11 ⇒ R
   (1≤i≤11 ∧ S=($\sum_k k : o \le k \le i : b[k]$) ∧ i≥11 ⇒ ($\sum_k k : o \le k \le 11 : b[k]$))=
   = (i=11 ∧ S=$\sum_k k : o \le k \le i : b[k]$) ⇒
   ⇒ S=($\sum_k k : o \le k \le 11 : b[k]$) = (вместо i подставляем 11)
   видим, левая и правая части равны.

Предикат P истинный перед и после выполнения каждого шага цикла, называется инвариантом цикла.
Введем функцию t, которая покажет, сколько шагов до конца цикла осталось.

t:=11-i

Это ограничивающая функция. Показывает верхнюю границу числа оставшихся шагов.

Составим программу, которая z ← a*b при b≥0

```
{b≥0}
x,y,z := a,b,0;
do
     y>0 ∧ чет(y) → y,x :=y÷2, x+x
     нечет(y)     → y,z :=y-1, z+x
od
{R:z=a*b}
```

Инвариант P: y≥0 ∧ z+x+y=a*b

1. Проверяем истинность P после инициализации переменных
   Wp(“x,y,z:=a,b,0”,P)=((b≥0)∧(0+a*b=a*b))=(b≥0)
2. Перед шагом и после шага цикла

а) z+x*y до шага	z+(y/2)*(x+x) после шага	Эти скобки раскрывать нельзя, но т.к. y – четное, то можно раскрыть.
б) z+x*y до шага	z+x+y-1*x= z+y*x после шага	

3. Истинность инварианта и ложность охран → постусловие.
   P ∧ ¬(y>0 ∧ чет(y)) ∧ ¬нечет(y) =
   = P ∧ (y≤0)=(y≤0 ∨ ¬чет(y) ∧ четн(y) ∧ P)=
   =(y≤0 ∧ чет(y) ∧ P)= (y≤0 ∧ чет(y) ∧ (y≥0 ∧ z+y*x=a*b))=
   =(y=0 ∧ z+y*x=a*b ⇒ z=a*b)

Ограничивающая функция t = y

Теорема о цикле, его инварианте и ограничивающей функции

Посылки:

1. $P \wedge B_i \Rightarrow wp(S_i, P)$ - сохранение инварианта

Т.е. если выполняется инвариант и нас пустила i-ая охрана, то выполнение i-ой команды составит истинность P.

2. $P \wedge BB \Rightarrow t>0$ (т.е. есть еще шаги)

Т.е. если выполняется инвариант и BB, то ограничивающая функция положительна.

3. С каждым шагом ограничивающая функция уменьшается.

$P \wedge B_i \Rightarrow wp(\text{“}t_1:=t;S_B\text{”}, t<t_1)$

Т.е. если нас пустила i-ая охрана, то выполнение i-ой команды уменьшит функцию.

Вывод: если эти посылки выполняются, то P можно использовать в качестве предусловия для цикла. $P \Rightarrow wp(\ DO\ , P \wedge \neg BB)$;

Аннотирование цикла

Перед циклом

{Q}

{inv P: инвариант}

{bound t: ограничивающая функция}

do

$B_1 \rightarrow S_1$

$B_2 \rightarrow S_2$

…

$B_n \rightarrow S_n$

od

{R : постусловие}

Список условий для проверки цикла

1) P истинно перед выполнением цикла
2) P является инвариантом цикла $\{P \wedge B_i\}\ S_i\ \{P\}$
3) Выполнение P и невыполнение BB должно дать R: $P \wedge BB \Rightarrow R$
4) Если цикл еще не закончен, то ограничивающая функция положительна: $P \wedge BB \Rightarrow t>0$
5) Каждый шаг цикла ведет к концу цикла $\{P \wedge B_i\}\ t_1:=t;S_1\ \{t<t_1\}$

Пример: найти сумму элементов массива.

{T}

i,S := 10,0;

{inv P:$0 \le i \le 10 \wedge S=(\sum_k k : i+1 \le k \le 10 : b[k])$}

{bound t:i}

do

$i \ne 0 \rightarrow$ i,S:=i-1, S+b[i]

od

$\{R:\ S=(\sum_{k} k : 1 \le k \le 10 : b[k])\}$

Процедуры

Процедуры в языках программирования используются в целях абстракции.

Абстракция – действие, состояние в выборе для дальнейшего изучения или использования небольшого числа свойств объекта и изъятие из рассмотрения остальных свойств, которые в данный момент не нужны.

Абстракция относительно процедур:

Что процедура делает – важно

Как она это делает – изымаем.

Построение процедуры – расширение языка путем включения новой операции.

Описание процедуры

Proc <идентификатор> (<спецификация параметров>,…< спецификация параметров >);

Способы передачи параметров:

1. by value передается только значение
2. by result
3. by value-result
4. by reference
5. by name текстовая подстановка
6. стек

Пример:

```
Prog
 B[1]:=1
 B[2]:=1
 I:=1
 P(B[I])

Proc P(x)
 I:=1
 x:=x+2
 B[I]:=10
 I:=2
```

```
x:=x+2
```

После выполнения

	B[1]	B[1]
1	1	1
2	-	-
3	5	1
4	12	1
5	10	3

<спецификация параметров>:

value <спецификация параметров>: тип

result <спецификация параметров>: тип

proc <идентификатор> (<спецификация параметров>)

{P : предусловие }   <тело> {Q : постусловие }

будем считать, что правильность процедуры доказана.

Аннотирование процедур

```
{Pre:P}
{Post:Q}
```

Proc <идентификатор> (<спецификация параметров>)

Этого достаточно, тело не рассматриваем $\Rightarrow$ использовать глобальные переменные нельзя.

В P могут входить параметры с атрибутами

value
value result

В Q: **value result**
result

Рассмотрим процедуру поиска элемента массива

```
{Pre: n=N ∧ x=X ∧ X∈B[0:N-1] ∧ b=B}
{Post: 0≤i<N ∧ B[i]=X}

Proc search (  value n,x:integer;
          value b; array of integer;
          result i: integer );
```

Тело:

```
{i:=0}
{инвариант: 0≤i<N ∧ X∈B[i:N-1]}
{ограничение  N-i}
do b[i]≠x → i:=i+1 od
```

$$\begin{cases} proc\_P(value\_\bar{x}; value\_result\_\bar{y}; result\_\bar{z}) \\ \{P\}B\{Q\} \end{cases}$$

Вариант, который будем рассматривать дальше

Описание вызова:

P($\bar{a},\bar{b},\bar{c}$) x,y,z – формальные параметры – параметры
a,b,c - фактические параметры – аргументы

a_i – входные аргументы

b_i – входные/выходные аргументы

c_i – выходные аргументы

Вызов нашей процедуры:

```
search(50,t,c,position[i])
```

Алгоритм вызова процедуры

1. определяются значения аргументов $\bar{a}$ и $\bar{b}$ и записываются в параметре $\bar{x}$ и $\bar{y}$, причем в $\bar{a}$ может быть выражение.
2. определяются переменные, описываемые $\bar{b}$ и $\bar{c}$.
3. выполняется тело процедуры
4. значение выходных параметров $\bar{y}$ и $\bar{z}$ записываются в выходные аргументы $\bar{b}$ и $\bar{c}$.

Формальное определение вызова процедуры

$\bar{x}, \bar{y} := \bar{a}, \bar{b}$; B ; $\bar{b}, \bar{c} := \bar{y}, \bar{z}$.

Wp(“p($\bar{a},\bar{b},\bar{c}$)”,R)=wp(“$\bar{x}, \bar{y} := \bar{a}, \bar{b}$; B; $\bar{b}, \bar{c} := \bar{y}, \bar{z}$. ”,R)

Теорема 1 (о вызове процедуры)

Proc P(value $\bar{x}$, value result $\bar{y}$, result $\bar{z}$)

$\{P\}\ B\ \{Q\}$ //B: $\bar{y}, \bar{z} := \bar{u}, \bar{v}$ сокращено тело

$\{PR: P^{\bar{x},\bar{y}}_{\bar{a},\bar{b}} \wedge (\forall \bar{u}, \bar{v} : Q^{\bar{y},\bar{z}}_{\bar{u},\bar{v}} \Rightarrow R^{\bar{b},\bar{c}}_{\bar{u},\bar{v}})\}$

$P(\bar{a}, \bar{b}, \bar{c})$

$\{R\}$

$PR \Rightarrow wp(p(\bar{a}, \bar{b}, \bar{c}), R)$

PR можно использовать в качестве предусловия.

Пример: $\bar{x}, \bar{y} = \bar{y}, \bar{x}$

```
Proc swap (value result y₁, y₂);
```

$\{P: y_1=X \wedge y_2=Y\}$

B

$\{Q: y_1=Y \wedge y_2=X\}$

Доказать, что если $\{a=X \wedge b=Y\}$ то после вызова процедуры swap (a,b) будет $\{a=Y, b=X\}$

Доказательство:

$PR = (a=X \wedge b=Y \wedge (\forall u_1,u_2: Q: u_1=Y \wedge u_2=X \Rightarrow u_1=Y \wedge u_2=X)) = a=X \wedge b=Y$

Пример:

$\{a=A \wedge b=B\}$ – предусловие, та же процедура swap(a,b)

$\{a=B \wedge b=A\}$

$\{y_1=X \wedge y_2=Y\}\ B\ \{y_1=Y \wedge y_2=X\} = \{y_1=A \wedge y_2=B\}\ B\ \{y_1=B \wedge y_2=A\}$

Пример:

$\{i=I \wedge (\forall k: b[k]=B[k])\}$

swap(i, b[i])

$\{i:=B[I] \wedge B[I]=I \wedge \forall k : I\neq k : b[k]=B[k]\}$

Перепишем процедуру

$\{P: y_1=I \wedge y_2=B[I]\}\ y_1,y_2:=u_1,u_2\ \{Q: y_1=B[I] \wedge y_2=I\}$

$PR=P^{y_1,y_2}_{i,b[i]} \wedge (\forall u_1,u_2\ Q^{y_1,y_2}_{u_1,u_2} \Rightarrow R^{i,b[i]}_{u_1,(b:i=u_2)}) =$

$= i=I \wedge b[i]=B[I] \wedge (\forall u_1,u_2\ u_1=B[I] \wedge u_2=I \Rightarrow$

$\Rightarrow u_1=B[I] \wedge (b:i:u_2)[I]=I \wedge \forall k\neq(b:i:u_2)[k]=B[k])=$ //раскроем квантор всеобщности

$= i=I \wedge b[i]=B[I] \wedge (B[I]=B[I] \wedge I=I \Rightarrow B[I]=B[I] \wedge$

$\wedge (b:i:I)[I]=I \wedge \forall k:k\neq I:(b:i:I)[k]= B[k])\Rightarrow(i=I \wedge b[i]=B[I] \wedge I=I \wedge \forall k: k\neq I:b[k]B[k])=i=I \wedge (\forall k: b[k]=B[k])$

Построение программ.

Основные принципы:

1. программа и доказательства ее правильности должны строиться одновременно
2. строя программу надо пользоваться и теорией и здравым смыслом
3. изучите свойства объектов, с которыми должна работать программа
4. никогда не отвергайте как очевидный никакой фундаментальный принцип
5. программирование – это целенаправленная деятельность
   Целенаправленная деятельность значит –
   {?} S {R: z=max(x,y)} и {T} S {?}
6. уточните и разъясните себе предусловие и постусловие

Пусть есть задача

Определено предусловие {T} S {?}

Нужно найти максимальное из 2-х чисел.

{T} S { R: z=max(x,y)}

напишем программу исходя из предусловий и постусловий.

R: $z:=x \wedge x \ge y \vee z=y \wedge y \ge x$ расписали что значит максимум

S - ?

Выскажем две гипотезы

S

→ z:=x

→ z:=x+1

Возьмем первую гипотезу, т.к. она проще и следует из R.

Найдем условие, такое что из z:=x получаем R.

Wp(“z:=x”,R) = $x \wedge x \ge y \vee z=y \wedge y \ge x = x \ge y \vee x=y = x \ge y$

Программа z:=x при условии x≥y приведет к R.

```
if
     x≥y → z:=x
fi
```

Какие еще охраны можно написать?

Можно еще написать команду z:=y, подставив получим x≤y.

Теперь можно написать программу S.

z:=y

wp(“z:=y”,R)=y≥x

S:

```
if
      x≥y → z:=x
      y≥x → z:=y
fi
```

Теперь можно доказать, что это правильно (по теореме о команде выбора)

1. $Q \Rightarrow BB$

   $T \Rightarrow (x \ge y) \vee (y \ge x) = (F \vee T) = T$

2.1 $Q \wedge B_1 \Rightarrow wp(S_1, R)$

   $T \wedge x \ge y \Rightarrow wp(\text{“}z:=x\text{”}, z=max(x,y)) = (x \ge y \Rightarrow x=max(x,y)) = T$

2.2 $Q \wedge B_2 \Rightarrow wp(S_2, R)$

$T \wedge y \geq x \Rightarrow wp(“z:=y”,\ z=max(x,y))=(y \geq x \Rightarrow y=max(x,y))=T$

При прочих равных условиях делайте охраны в командах выбора настолько, сильными насколько возможно, с тем, чтобы некоторые ошибки приводили к аварийному останову.

Построение циклов исходя из инвариантов и ограничений

Увеличить K, сохраняя j=(K mod 10)

```
if
       j < 9 → K,j:=K+1,j+1
       j = 9 → K,j:=K+1,0
fi
```

Правило:

Сначала строится охрана B такая, что B: $P \wedge \neg B$, затем строится тело цикла, таким образом, чтобы оно уменьшало ограничивающую функцию при сохранении инварианта цикла.

```
{inv P}
{bound t}
 do
     B → уменьшить t, сохраняя P
 od
{P∧¬B}
```

Пусть нужно суммировать элементы массива

$n \geq 0$ b[0:n-1]

R: $S = \sum j:\ 0 \leq j \leq n : b[j]$

P: $0 \leq i \leq n \wedge S = \sum j:\ 0 \leq j < i : b[j]$

t: n-I

1. i,s:=0,0 – инициализация переменной, после которой P истина.
2. Нужно найти охраны.

$P \wedge ? \Rightarrow R$

$P \wedge I = n \Rightarrow R$

B: $i \neq n$

уменьшение ограничивающей функции

i:=i+1

$S:=S+b[i] = \sum j:\ 0 \leq j < i+1 : b[j]$

S,i:= S+b[i],i+1

```
do
      i≠n → i,S:=i+1,S+b[i]
od
```

Докажем правильность программы

1. P истинно перед началом цикла

$0 \le 0 \le n \wedge 0 = \sum j:\ 0 \le j < 0 : b[j] = T$

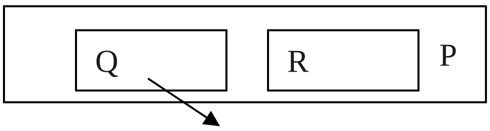
2. P является инвариантом цикла $wp(S_i,P)=P$
$Wp(“i,S:=i+1,S+b[i]”,P)=$
$=(0 \le i+1 \le n \wedge S+b[i] = \sum j:\ 0 \le j < i+1 : b[j])$
Это условие следует из P и $n \ne i$, отсюда следует что инвариант сохранился.

3. $P \wedge \neg BB \Rightarrow R \quad P \wedge i=n \Rightarrow R$
$(0 \le n \le n \wedge S) = (\sum j:\ 0 \le j < n : b[j]) = (S = \sum j:\ 0 \le j < n : b[j])$
4. $P \wedge i \ne n \Rightarrow t > 0$
$P \wedge i \ne n \Rightarrow n-i > 0 [n > i]$
$0 \le i \le n \wedge i \ne n \Rightarrow n > i$

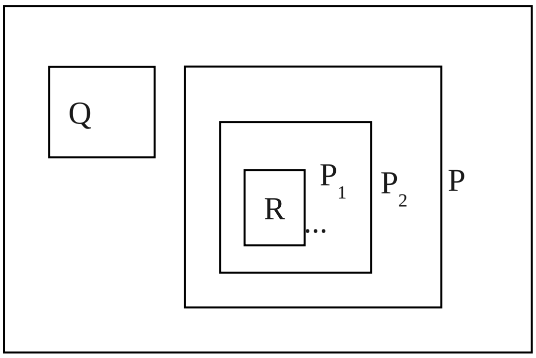
5. Каждый шаг цикла ведет к его концу
$\{P \wedge i \ne n\}\ t_1 := t;\ S_i\ \{t < t_1\}$
$\{P\ i \ne n\}\ t_1 := n-i;\ i,S := i+1, S+b[i]\ \{t < t_1\}$
$wp(“t_1:=n-i;\ i:=i+1”,\ n-i < t_1) =$
$wp(“t_1:=n-i”, n-i-1 < t_1) = (n-i-1 < n-i) = (1 > 0) = T$

Построение инварианта

$\{Q\}\ do\ B \to S\ od\ \{R\}$



начальное состояние



Начиная с R наращиваем до Q.

Ослабление предикатов (точнее предиката R):

1. Устранение конъюнктивного члена

$A \wedge B \wedge C \to A \wedge C$

$n \ge 0$

$R:\ 0 \le a^2 \le n < (a+1)^2$

$0 \le a \le \sqrt{n} < (a+1)$

$R:\ 0 \le a^2 \wedge a^2 \le n \wedge n < (a+1)^2$

```
      P: 0≤a² ∧ a²≤n
P ∧ ¬B⇒R
B = n≥(a+1)²
      t: n-a
      a:=0
      do
            n≥(a+1)² → a:=a+1
      do
wp("a:=a+1",P) = 0≤(a+1)² ∧ (a+1)²≤n
P∧n ≥ (a+1)² ⇒ 0≤(a+1)² ∧ (a+1)²
```

2. Замена константы переменной

Нужно найти суммы элементов массива

`b[0:n-1]   n>0`

`R: S=∑ j: 0≤j<n : b[j]` константу n заменяем на i.

`P: S=∑ j: 0≤j<i : b[j]∧ 0≤i≤n`

Инициализация переменных, чтобы P было истинно перед циклом

`i, S :=0, 0`

ограничивающая функция `t: n-i`

```
i≠n
do
      i≠n → i,S := i+1, S+b[i]
od
```

3. Расширение области значения переменной.

Найдем значение вхождения X в массив b.

`b[0:n-1]  n>0`

iv – const – наименьшее i, удовлетворяющее условию `0≤i ∧ x=b[i]`

```
R: i=iv
P: 0≤i≤iv
i:=0;
do
   x≠b[i] → i:=I+1
od
```

4. Комбинирование предусловия и постусловия

```
Q: ∀i: 0≤i<n : b[i]=B[i]
R: ∀i: 0≤i<n : b[i]=B[i]+P*i
```

R вместо ? не подойдет т.к. n-константа

P: $0\le j\le n \wedge (\forall i:\ 0\le i<j : b[i]=B[i]+P*i) \wedge (\forall i: j\le j<n : b[i]=B[i])$

Цикл:

```
j:=0
do
      j≠n → j, b[j] := j+1, b[j]+P*j
od
```

Построение ограничивающей функции

Функцию можно выражать как свойство инварианта и задачи.

Лексикографический порядок.

пара(i, j)<(h, k) если i<h или если $i=h \wedge j<k$

Пример:
(3,5,5)<(5,5,5)<(5,6,0)<(5,6,1)

Теорема об ограничивающей функции

Пара (i, j), где i, j – выражение содержащее переменные, используемые в цикле.
Если выполняется:

1) (i, j) – на каждом шаге уменьшается.
2). $\min i \le i \le \max i$;
$\min j \le j \le \max j$;

где min i, max i, min j, max j – некоторые const.
Если 1), 2) – выполняются, то выполнение цикла должно завершиться.

Ограничивающая функция t будет иметь вид:
t: (i - min i)*(1+max j – min j)+j – min j

Пример: сумма элементов двумерного массива

b[0: n-1][0:m-1]
m – строка
n – столбец
$\{0<m \wedge 0<n\}$
i, j:=n-1, m-1

do
$j \ne 0 \rightarrow$ S,j := S+b[i][j] , j-1
$i \ne 0 \wedge j=0 \rightarrow$ S, i, j :=S+s[i][j], i-1,m-1
od

1) выход из цикла, когда i и j =0 одновременно

$0 \le i \le n$
$0 \le j \le m$
2)
$\{P \wedge B\}<i_0,j_0> := <i,j>; S_2\{<i_0,j_0> > <i,j>\}$
$wp(“i_0,j_0:=i,j;\ i,j:=i-1,m-1”,\ <i_0,j_0> > <i,j>) =$
$= wp(“i_0,j_0:=i,j;\ <i_0,j_0> > <i-1,m-1>)=$
$=<i,j> > <i-1,m-1> = T$
Подставим для получения t:
t: (i-0)*(m)+j –0 = i*m+j;

Задача: поезда на станции
Можно выводить поезд, содержащий 1 вагон.
$0 \le$ число вагонов $\le$ начальное число вагонов
начальное число вагонов $\le$ число поездов ≤ 0
Чем больше поездов, тем ближе мы к окончанию работы.

```
do
	сортировочная непустая →
	выбираем поезд →
	if
		в поезде 1 вагон → удалить поезд
		в поезде больше 1го вагона → разбить поезд на две части
	fi
od
```

Правильность представления данных.

Доказательство правильности представления данных

Переход от абстрактной программы к конкретной

Метод перехода от абстрактной программы к конкретной, а также доказательство правильности такого перехода.

В абстрактной программе нет типов данных.

Записывается программа в общем виде и делается переход к конкретной программе, доказывается, что она соответствует абстрактной программе.

Абстрактная программа намного короче конкретной.

t- абстрактная переменная, ее тип T. (например объект типа множества)

c_1, c_2, …, c_n – конкретное представление t.

Операции с t_1 возможны следующие: P_1, P_2, …, P_m.

Конкретное представление

```
Class T;
Begin
     Описание c₁, c₂, …, cₙ;
     Procedure P1 (формальные параметры); Q₁;
     …
     Procedure Pm (формальные параметры); Qm;
     Q (призвание начальных значений c₁, c₂, …, cₙ)
End
```

Qi – тело Pi процедуры

Для описания переменной

Var t1, t2, … : T

t2.Pj – вызов Pj процедуры для t2

Опишем множество целых чисел.

```
Class НМЦ;
     Begin
     Var
          m: integer;
          b: array[0..99] of integer;

     proc добавить(value i:integer);

     Proc содержит(value l: integer; result is: boolean);

     Proc удалить(value i:integer)

     m:=0;
end
```

Операции (в программе)

Абстрактные	Конкретные
$t_i=f_j(t_i,a_1, a_2,... a_{nj})$ t инициализир. пустое мн. {}	$t_i.P_j(t_i,a_1, a_2,... a_{nj})$ var t:НМЦ

X=i $\in$t (принадлежит ли i мн. t)	t.содержит(i,x)
t:=t$\vee${i}	t.добавить(i,)
t:=t \ {i}	t.удалить(i)

Доказать, что абстрактные выражения эквивалентны конкретным:

Критерий правильности предусловия данных состоит в том, что каждая P_j моделирует соответствующую f_j.

Для доказательства используем функции:

$K_T(P(t))$ - отображает предикатное условие с абстрактными переменными в его конкретное представление в контексте типа T.

$K_{нмц}(h=H) \equiv m=M \wedge \forall k: 0\le k<m : b[k]=B[k]$

Надо доказать

- для инициализации $\{T\}\ Q\ \{K_T(h=h_0)\}$

- для процедур

$\{K_T(h=H)\}\ P_j(a_1,a_2,\ .\ .\ .\ ,a_{nj})\{K_T(h=f_j(H,a_1,a_2,\ .\ .\ .,a_{nj}))\}$

инициализация m:=0

$\{T\}\ Q\ \{K_{нмц}(h=\varnothing)\}$

$\{T\}\ Q\ \{m=0\}$

Q: m:=0

wp(“m:=0”,m=0) = 0=0 = T

Инвариант класса.

Какое должно быть отношение между переменными класса.

$I = 0\le m<160 \wedge (N_{i,j} : i\ne j \wedge 0\le i<m \wedge 0\le j<m : b[i] = b[j] = 0$

$\{I\}\ Q_{содерж.}\ \{I \wedge K_{нмц}(x=i\in h)\}$

Само множество не меняется.

Преобразуем:

$\{I\}\ Q_{содерж.}\ \{I \wedge x= \exists k: 0\le k<m: b[k]=i)\}$

Заменим константу переменной:

Inv:

$0\le j\le m \wedge x:=\exists k: 0\le k<j: b[k]=i$

огр: m-j

охр: $j\ne m$

$j,x:=j+1,x\vee b[j]=i$

инициализация: j,x:=0,F

```
 j,x:=0,F
     do
          j≠m → j,x:=j+1,x ∨ b[j]=i
     od
```

Процедура добавления элемента

$\{I\ (K_{нмц}(h\vee\{i\}))\} \wedge K_{нмц}(h=H)\}$

$Q_{добавления}$

$\{I \wedge K_{нмц}(h=H\vee\{i\})\}$

$\{(i\in b \wedge 0\le m\le 100 \vee i\notin b \wedge 0\le m+1\le 100) \wedge m=M \wedge b=B\}$

$\{0 \le m \le 100 \wedge (i \in b \wedge m \le M \wedge b=B \vee i \notin b \wedge m=M+1 \wedge b[0:M-1]=B[0:M-1] \wedge b[0:m-1]=i)\}$

$Q_{добавления}$ содержит (i,x)
```
if
      x=T → SKIP
      x=F → m,b[m]:=m+1,i
fi
```

Удаление

$\{ I \wedge K_{НМЦ}(h=H)\}\ Q_{удаления}\ \{K_{НМЦ}(h=H \setminus \{i\})\}$

$\{I \wedge m=M \wedge b=B\}\ \ Q_{удаления}\ \{\forall k\colon 0 \le k<m\colon b[k]=i \wedge (m=M \wedge b=B \vee m=M-1 \wedge \forall k\colon 0 \le k<m\colon (b[k]=B[k] \vee b[k]=B[M-1] \wedge B[k] = i))\}$

$Q_{удаления}$ содержит (i,x);
```
If
      x=F → SKIP
      x=T → Sц
fi
```

Inv:

$0 \le j \le m \wedge \forall k\colon 0 \le k<j\colon (b[j] \ne i \wedge (b[k]=B[k] \vee b[k]=B[M-1] \wedge B[k]=i))$

Если это все доказано, то соответственно доказана и правильность представления данных.

X – абстрактная программа

X' – конкретная программа

X' получаем из X

Преобразование алгоритмов

Язык логических схем
<u>ЯЛС</u>

Описание

ЯЛС создан для описания дискретных процессов.
ЯЛС обеспечивает:

- простоту описания дискретных процессов;
- удобство равносильных преобразований алгоритмов;

Обозначение операторов в ЯЛС:

D – действующий ($x:=f_2(x_2y_5)$)
V – варьирующийся ($x:=f_i(x_iy_i)$)
P – логический
F – формирование объектов
Ф – ввод объектов

Элементарные выражения

$\lfloor_{v}\ \lfloor_{v}\quad {}_{v}\rfloor\quad [^{v}_{v}\quad]^{v}_{v}$

$U_0\lfloor_{M}$ - начало схемы

$Я_M$ - конец схемы

$Q\lfloor_{v}$ $\quad Q \in \{D, V, F, Ф\}$

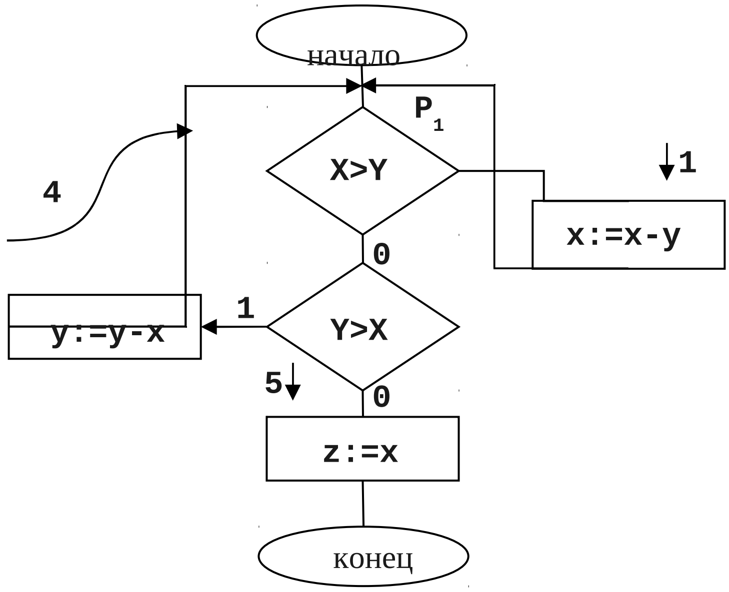
$P[^{v_1}_{v_2}$ - если P истина, то v_2 иначе v_1

${}_{M}\rfloor$ - куда происходит переход

M – целое
v, v_1, v_2 – целочисленное выражение

Логической схемой называется конечная строка элементарных выражений.
После логической схемы в [ ] пишется расшифровка всех операторов.

Пример: Найдем наибольший общий делитель



Преобразуем:

$$U_0 \lfloor_1 \rfloor_1 P_1 \lfloor^{2}_3 \rfloor_3 P_2 \lfloor^{4}_5 \rfloor_5 D_1 \lfloor_6 \rfloor_6 Я_1 \;\; \rfloor_2 D_2 \lfloor_1 \rfloor_4 D_3 \lfloor_6$$

[P_1 x>y; P_2 y>x; D_1 z:=x; D_2 x:=x-y; D_3 y:=y-x]

Равносильные преобразования алгоритмов

Основные понятия

Два алгоритма называются равносильными, если равносильность между исходными данными приводит к равносильности между их результатами.

Комплекс – алгоритм с исходными данными и результатами.

Преобразование комплекса, приводящее к возникновению нового, алгоритм которого равносилен исходному, называется равносильным преобразованием алгоритма.

Свойства отношения равносильности алгоритма:

1. Рефлексивность
   A = A
2. Симметричность
   A = B $\to$ B = A
3. Транзитивность
   если A = B , B = C то A = C

В дальнейшем будем рассматривать однородные комплексы.

На однородные комплексы наложены определенные ограничения, такие как единственный выход из цикла.

Запишем все операторы в виде нормальной последовательности формул.

$\alpha_{t_1} := \varphi_1(\alpha_1, \alpha_2, ...\alpha_n)$

$\alpha_{t_2} := \varphi_2(\alpha_1, \alpha_2, ...\alpha_n)$

………………………

$\alpha_{t_N} := \varphi_N(\alpha_1, \alpha_2, ...\alpha_n)$

X = $< \alpha_{S_1}, \alpha_{S_2}, ...\alpha_{S_m} >$ - входной кортеж

Y = $< \alpha_{r_1}, \alpha_{r_2}, \ldots\ldots \alpha_{r_m} >$ - выходной кортеж

```
u:=x-y
v:=x+z
v:=v*u

X=<x,y,z>
Y=<v>
Z=<u>
```

N=0 , то Y⊆X (Y включено в X)
Z – рабочий кортеж

Выявление равносильностей алгоритмов

```
u:=x-y
v:=x+z
v:=v*u

v:=(x+z) * (x-y)
```

x = $< \delta_{l+1}, \delta_{l+2}, \ldots\ldots \delta_{l+m} >$
y = $< \beta_1, \beta_2, \ldots\ldots \beta_M >$

$$\beta_1 := \theta_1(\delta_{l+1}, \delta_{l+2}, \ldots\ldots \delta_{l+m})$$
$$\beta_2 := \theta_2(\delta_{l+1}, \delta_{l+2}, \ldots\ldots \delta_{l+m})$$
…………………………….
$$\beta_M := \theta_M(\delta_{l+1}, \delta_{l+2}, \ldots\ldots \delta_{l+m}) \ ,$$

где θ - функция получения ответа
β - выходные переменные
Очередность элементов может быть любой.

Пусть существуют два однородных комплекса K_1 и K_2 и на них наложены определенные ограничения.

K_1

$$\beta_1 := \theta_1(\delta_{l'+1}, \delta_{l'+2}, \ldots\ldots \delta_{l'+m})$$
$$\beta_2 := \theta_2(\delta_{l'+1}, \delta_{l'+2}, \ldots\ldots \delta_{l'+m})$$
…………………………….
$$\beta_M := \theta_M(\delta_{l'+1}, \delta_{l'+2}, \ldots\ldots \delta_{l'+m})$$

K_2

$$\beta_1 := \theta'_1(\delta_{l''+1}, \delta_{l''+2}, \ldots\ldots \delta_{l''+m})$$
$$\beta_2 := \theta'_2(\delta_{l''+1}, \delta_{l''+2}, \ldots\ldots \delta_{l''+m})$$
…………………………….
$$\beta_M := \theta'_M(\delta_{l''+1}, \delta_{l''+2}, \ldots\ldots \delta_{l''+m})$$

В каждом комплексе выберем нетождественные функции (не повторяющиеся функции)
Для K_1 это θ_{ai} , а для K_2 это θ'_{bi}

Для равносильности K_1 и K_2 необходимо и достаточно:

1. Число отобранных функций должно быть одинаково
2. $\theta_{ai} = \theta'_{bi}$ и т.д. $\theta_{an} = \theta'_{bn}$

Алгоритмы:

K_1

```
u:=x+sin2z+cos2z
v:=y*u
w:=v
```

K_2

u:=x+$\frac{2}{\pi}$ (arccos t + arcsin t)

v:=y*u

x_1 = <x,y,z> входной кортеж x_2 = <x,y,t>

y_1 = <u,v,w> выходной кортеж y_2 = <u,v>

K_1 выразим

$$\beta_1 = \delta_2 + \sin^2 \delta_4 + \cos^2{}_4$$

$$\beta_2 = \delta_3 * (\delta_2 \sin^2 \delta_4 + \cos^2{}_4)$$

$$\beta_3 = \delta_3 * (\delta_2 \sin^2 \delta_4 + \cos^2{}_4)$$

K_2 выразим

$$\beta_1 = \delta_2 + \frac{2}{\pi}(\arcsin \delta_1 + \arccos \delta_1)$$

$$\beta_1 = \delta_3 * (\delta_2 + \frac{2}{\pi}(\arcsin \delta_1 + \arccos \delta_1))$$

Тождественные функции:

K_1

$$\delta_2 + \sin^2 \delta_4 + \cos^2{}_4 \quad =$$

$$\delta_3 * (\delta_2 \sin^2 \delta_4 + \cos^2{}_4) \quad =$$

Вывод: K_1 равносилен K_2

K_2

$$\delta_2 + \frac{2}{\pi}(\arcsin \delta_1 + \arccos \delta_1)$$

$$\delta_3 * (\delta_2 + \frac{2}{\pi}(\arcsin \delta_1 + \arccos \delta_1))$$

Операции над комплексами

Сумма:

K = K_1 + K_2

$\Theta_{ai} = \Theta'_{bi}$

G’ - область задания K_1

G’’ - область задания K_2

G = G′ + G′′

Произведение:

K = K_1 * K_2

$X_2 \cap Z_1 = 0$ Z_1 – рабочий кортеж K_1

$$\theta = \begin{cases} \theta_1 \\ \theta_2 \end{cases}$$ - соединяем два алгоритма

$x=x_1+(x_2-y_1)$ $\qquad$ y_1 – результат в θ_1

$y=y_2+(y_1-z_2)$

$K_1K_2 \neq K_2K_1$

$K_1(K_2K_3) = (K_1K_2)K_3$

Пример:

K_1:=a:=2*b+c

K_2:=e:=3*c-a

$x_1 = \langle b,c \rangle$

$x_2 = \langle c,a \rangle$

$y_1 = \langle a \rangle$

$y_2 = \langle e \rangle$

$x = \langle b,c \rangle$

$y = \langle e,a \rangle$

Единичный комплекс

Если x = y, то комплекс единичный $K = \Lambda$

Обратный комплекс

Если $K_n * K_{-n} = \Lambda$, то K_{-n} обратный справа

Равносильные преобразования алгоритмов заданных на ЯЛС

1)

$\rfloor_i \rfloor_j = \rfloor_j \rfloor_j$

2)

$\varepsilon(\,,\,)\underset{k}{\perp} \quad \varepsilon(\,,\,)$ - схема

Пример:

$\varepsilon(W_1,W_2) = \varepsilon(W'_1,W'_2)$ $\quad \varepsilon$ - схема

Преобразуем схемы – убираем метки правых знаков перехода, если

$= \quad , \llcorner_k \quad \ulcorner^k$

нет соответствующих левых знаков перехода.

3)

Если после левого знака перехода стоит правый, то его убираем.

$$R \rfloor_i = R \rfloor_i$$

$$U_0 \lfloor_1 \rfloor_1 D_1 \lfloor_2 \rfloor_2 P_1 \lfloor^4_3 \rfloor_4 D_2 \lfloor_5 \rfloor_3 \rfloor_5 D_3 \lfloor_6 \rfloor_6 Я$$

Поменяем местами

Преобразуем:

$$U_0 \rfloor_1 D_1 \rfloor_2 P_1 \lfloor_3 \rfloor_4 D_2 \rfloor_5 \rfloor_3 D_3 \rfloor_6 Я$$

Далее:

$$U_0 D_1 P_1 \lfloor_3 D_2 \rfloor_3 D_3 Я$$

Замыкание оператора

Замыкание – элементарное выражение + правый знак перехода слева от него.

$$\rfloor_2 D_1 \lfloor_3$$

4)

$$\varepsilon(T_1,T_2) = \varepsilon(T_1,T_{2,})$$

5)

TW= Λ, если в схеме нет левого знака перехода, соответствующего правым знакам перехода в TW.

$$\begin{cases} \varepsilon(,\ R \rfloor_k) = \varepsilon(,) \\ R\,Q = R\,R \end{cases}$$

$$D_1 \lfloor_i D_2 \lfloor_j = D_1 \lfloor_i$$

$$P_1 \lfloor^j_i P_2 \lfloor^m_l = P_1 \lfloor^j_i$$

Пример:

$$U_0 D_1 \lfloor_1 \rfloor_2 D_2 \lfloor_3 \rfloor_1 P_3 \lfloor^5_4 \rfloor_3 D_6 \lfloor_2 \rfloor_4 D_7 \rfloor_5 D_8 Я$$

$$U_0 D_1 \lfloor_1 \rfloor_1 P_3 \lfloor^5_4 \boxed{\rfloor_2 D_2 \lfloor_3 \rfloor_3 D_6 \lfloor_2} \rfloor_4 D_7 \rfloor_5 D_8 Я$$

Нет перехода

$$U_0 D_1 P_3 \lfloor^5_4 \rfloor_4 D_7 \rfloor_5 D_8 Я$$

$$U_0 D_1 P_3 \lceil^5 D_7 \rfloor_5 D_8 Я$$

Выражение называется **совершенным**, если у одинаковых операторов:

1. одинаковые внешние левые знаки перехода (выходящие за данное выражение)

2. внутренним левым знакам перехода соответствуют правые знаки перехода у одинаковых операторов

$$\rfloor_1 P_1 \lfloor^{11}_{7} \rfloor_2 P_1 \lfloor^{11}_{6} \rfloor_7 \rfloor_6 D_1 \lfloor_{10}$$

$$\rfloor_4 D_1 \lfloor_{2} \rfloor_3 \rfloor_1 P_1 \lfloor^{10}_{4} \rfloor_5 D_1 \lfloor_{3} \rfloor_2 P_1 \lfloor^{10}_{5}$$

6)

Любой правый знак перехода, принадлежащий одному из одинаковых операторов совершенного выражения, можно отнести к другому.

π - совокупность правых знаков перехода.

$$\varepsilon(\rfloor_k Q \lfloor_i, Q_i) = \varepsilon(Q_i, Q_k)\lfloor_i$$

$$\rfloor_l Q \lfloor_i \rfloor_k Q \lfloor_i = \rfloor_l \rfloor_k Q \lfloor_i$$

7)

$$\varepsilon(P \lfloor^{i}_{i}, \rfloor_i) = \varepsilon(,)\rfloor_i$$

$$\rfloor_i P \lfloor^{k}_{k} \rfloor_k = \rfloor_i \rfloor_k$$

$$0 \lfloor^{j}_{i} = 0 \lfloor^{i}_{i} \qquad 0 \lfloor^{j}_{i} \rfloor_i = \rfloor_i \qquad 0 \lfloor^{j}_{i} = P \lfloor^{i}_{i}$$

$$1 \lfloor^{j}_{i} = 1 \lfloor^{j}_{j} \qquad 1 \lfloor^{j}_{i} \rfloor_j = \rfloor_j \qquad 0 \lfloor^{j}_{i} = P \lfloor^{j}_{j}$$

8)

$$P_1 \; P_2 \rightarrow \qquad P_1 \lfloor^{j}_{i} = P_2 \lfloor^{j}_{i}$$

9)

$$P \lfloor^{j}_{i} = P \lfloor^{i}_{j}$$

10)

$$(P_1 \wedge P_2) \lfloor^{j}_{i} = P_1 \lfloor_{C} P_2 \lfloor^{j}_{i}$$

11)

$$(P_1 \wedge P_2) \lfloor^{j}_{i} = P_1 \lceil^{j} P_2 \lfloor^{j}_{i}$$

12)

$$\begin{cases} \varepsilon(P\lfloor^{j}_{i},\ \rfloor_{i} P\lfloor^{i}_{k}) = \varepsilon(P\lfloor^{j}_{i},\ \rfloor_{i} P\lfloor^{i}_{k}) \\ \varepsilon(P\lfloor^{j}_{i},\ \rfloor_{j} P\lfloor^{l}_{k}) = \varepsilon(P\lfloor^{l}_{i},\ \rfloor_{j} P\lfloor^{l}_{k}) \end{cases}$$

13)

$$D_1\lfloor_i = D_2\lfloor_i$$

если D_1 и D_2 отвечают равносильным комплексам с одинаковыми входными и выходными кортежами.

14)

$$D_1\lfloor_i = D_1\ D_2\lfloor_i$$

если D – произведение D_1 и D_2

15)

$$D\lfloor_i = D\ \ D\lfloor_i$$

если не пересекаются кортежи:

входной и выходной

рабочий и выходной

16)

$$\varepsilon(D_1 D_2\lfloor_i,\ \rfloor_i) = \varepsilon(\ ,\ )\ \rfloor_i$$

если D_2 обратен справа D_1

17)

$$D_1 D_2\lfloor_k = D_2\ D_1\lfloor_k$$

если элементы выходного кортежа D_1 являются несущественными аргументами D_2 и наоборот, а их рабочие кортежи не пересекаются ни с входом ни с выходомкортежей другого.

18)

$$\underset{\psi(y,z)}{D}\ P\lfloor^{j}_{k} = \underset{\psi(k(x),z)}{P'}\ulcorner^{m}\ D\lfloor_{i}\ \rfloor_{m}\ D\lfloor_{j}$$

D y:=x+1

P y>5

P' x+1 > 5

Программирование таблиц решений

Таблица решений – непроцедурная форма записи программы.

Состоит из:

1. условий;
2. данных;
3. действий;

В наглядной форме определяет, какие условия должны быть выполнены прежде чем можно будет переходить к тому или иному действию.

Пример:
C1 - выходной
C2 - отпуск
C3 - лето
Y1 - отдыхать
Y2 – плавать

При выполнении каких условий мы будем выполнять те или иные действия?

C1	Yes	Yes	No	No	No	*
C2	-	-	Yes	Yes	Yes	*
C3	Yes	No	Yes	No	-	*
Y1	X	X	X	X		
Y2	X		X			

*** - иначе для всей таблицы

Полнота таблицы решений

Одним из преимуществ ТР является то, что они дают возможность систематически проверить каждую комбинацию значений и условий так, чтобы быть уверенным, что не пропущена ни одна из них.

Для того, чтобы учесть полная ТР или нет, нужно подсчитать кол-во столбцов.
$2^3=8$ (2-возможных варианта, 3-условия)

ТР с правилом иначе охватывает все правила, но может не рассмотреть случай из правильных.

Двусмысленность ТР

Использование (-) может приводить к двусмысленности.

C1		yes	-
C2		-	no
C3		-	yes
C4		yes	yes
Y1			
Y2		x	
Y3			x
Y4		x	x

Если входные данные отвечают YNYY, то подойдут оба столбца.

Если одни правила приводят к разным действиям, то такая двусмысленность называется противоречием.

Можно переделать (избавиться от двусмысленности).

Y1	X	X
Y2	X	X
Y3	X	X
Y4	X	X

Двусмысленность в таком виде – избыточность.

Виды

- таблицы с ограниченным входом
- таблицы с расширенным входом (условия не только yes-no)

Преобразование ТР

1. Прямое программирование
2. Метод ключа
3. Метод маски
4. Метод ветвления

1.

 if
 C1=Y $\wedge$ C3=Y $\rightarrow$ Y1,Y2
 C1=Y $\wedge$ C3=N $\rightarrow$ Y1
 fi

 “-“ – слишком громоздко.

2. Кодируем Y$\rightarrow$1 N$\rightarrow$0 (yes – 1 , no – 0)

По входным данным строим ключ:

Key:=C1+C2*2+C3*4

Запишем программу на ЯЛС.
D1
D2: Y1Y2
D3: Y1

$$U_0 \llcorner_1 \lrcorner_1 D_1 \llcorner_{key+2} \lrcorner_2 \lrcorner_6 Я \underbrace{\lrcorner_3 \lrcorner_4 \lrcorner_5}_{\text{Только отдыхаем}} D_3 \llcorner_2 \lrcorner_7 \lrcorner_8 \lrcorner_9 D_2 \llcorner_2$$

2 – ни одно из условий не выполняется
6 – выполняется лето

Чтобы определить все ключи, нужно расшифровать все “-“

3. Метод маски

Нужно построить 2 предварительных таблицы:

1) Таблица “-“, состоит из 1 и 0. На месте “-“ – 0

2) Таблица YES. На месте Y стоят 1.

Таблица “-“

1	1	1	1	1
0	0	1	1	1
1	1	1	1	0

Таблица YES

1	1	0	0	0
0	0	1	1	0
1	0	1	0	0

Выбор действия:

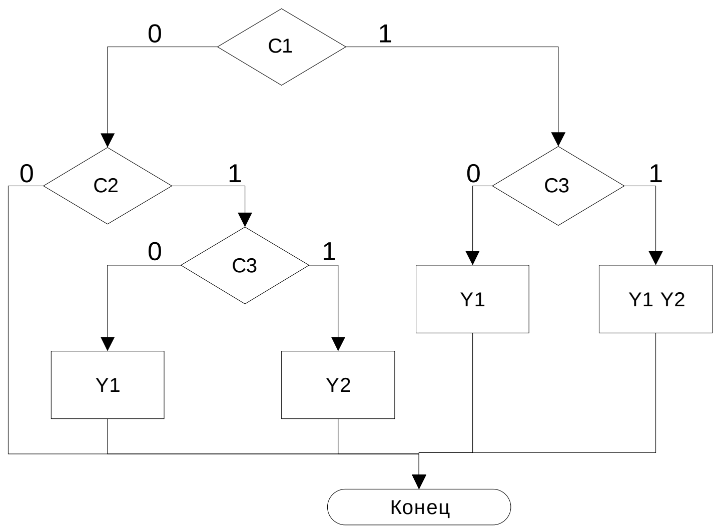
1) Формируем ключ
   NYY→ `key = 5`
   0 1 1 011

будний день отпуск лето
0+1+4=5

2) Выполняем & с очередным столбцом таблицы ‘-‘
   1ст. 011 & 101 = 001
   2ст. 011 & 111 = 011
3) Сравниваем со столбцом таблицы YES
   Если совпадает, то совершаем действие, связанное с этим столбцом.
   Если не совпадает, то проверяем следующий столбец.

4. Метод ветвлений
   Условия проверяются по очереди. На каждом очередном шаге выбирается столбец с большим числом ”-“

Проверяем условие C1



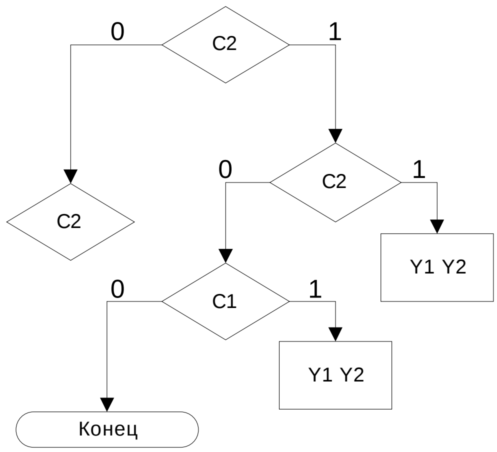


Таблица решений – более общий способ описания логики, чем любая блок-схема.