Верификация параллельных программных и аппаратных систем

Курс лекций

Карпов Юрий Глебович
профессор, д.т.н., зав.кафедрой
“Распределенные вычисления и компьютерные сети”
Санкт-Петербургского политехнического университета

karpov@dcn.infos.ru

Лекция 2

Индуктивный метод Флойда

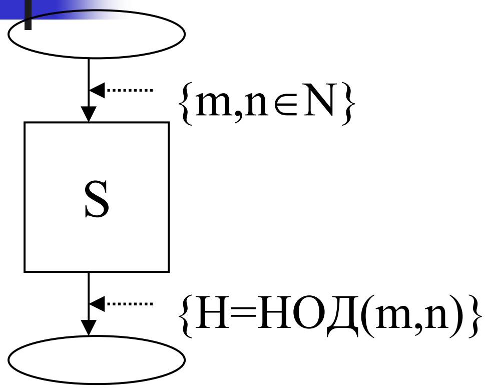
План курса

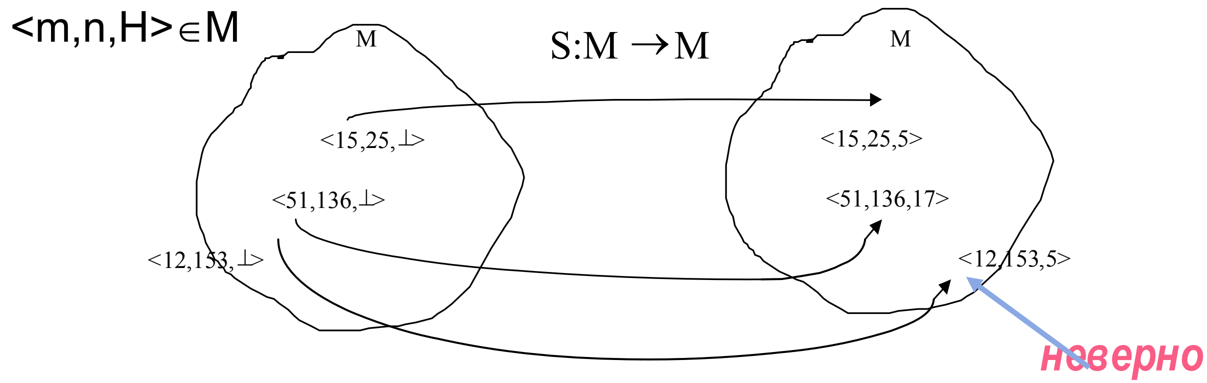
1. Введение
2. Метод Флойда-Хоара доказательства корректности программ
3. Исчисление взаимодействующих систем (CCS) Р.Милнера
4. Темпоральные логики
5. Алгоритм model checking для проверки формул CTL
6. Автоматный подход к проверке выполнения формул LTL
7. Структура Крипке как модель реагирующих систем
8. Темпоральные свойства систем
9. Система верификации Spin и язык Promela. Примеры верификации
10. Применения метода верификации model checking
11. BDD и их применение
12. Символьная проверка моделей
13. Количественный анализ дискретных систем при их верификации
14. Верификация систем реального времени (I)
15. Верификация систем реального времени (II)
16. Консультации по курсовой работе

Пример ошибки

- 09.09.2009. Компания Volvo объявила об отзыве по всему миру 26 тысяч автомобилей, у которых обнаружена неисправность в блоке управлением двигателем. Из-за этого дефекта мотор может не запуститься или заглохнуть после того, как автомобиль проедет несколько сотен метров
- Все владельцы неисправных машин получат письмо с предложением посетить сервисную станцию для бесплатного перепрограммирования блока управлением двигателем

Тестирование и верификация программ обработки данных





Как можно **гарантировать** правильность программы?

Нужно сформулировать, что программа ДОЛЖНА делать, и проверить, делает ли она это

Тестирование – проверка соответствия вход-выход в программе

"*Тестированием нельзя доказать правильность программы*"- Эдсгер Дейкстра

Полное тестирование программы нахождения НОД требует $2^{64}=10^{19}$ тестов

Мы хотим проверить правильность преобразования для ***ВСЕХ*** входных наборов

Поэтому необходимо **доказательство** правильности, т.е. **ВЕРИФИКАЦИЯ**

Например, сформулируем УТВЕРЖДЕНИЕ:

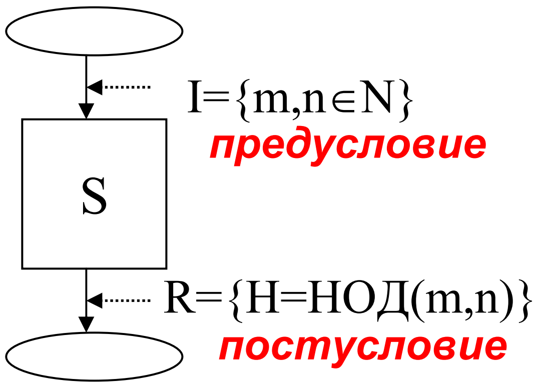
Если m и n – любые натуральные числа в начале программы, ***то*** при завершении программы S переменная H будет равна НОД от m и n

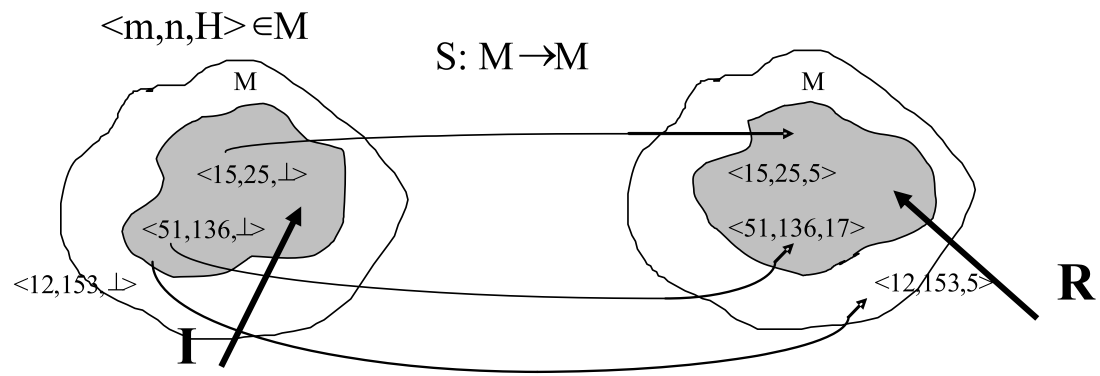
Спецификация программ обработки данных (1)

Программа - преобразователь на множестве всех ее возможных ***состояний***
(не всегда, иногда результат - файлы, операции ввода-вывода, и т.п.)

Состояние – вектор значений всех переменных программы

Такие программы называются ТРАНСФОРМАЦИОННЫМИ





Как ***конечным образом*** описать все исходные и “правильные” финальные состояния???

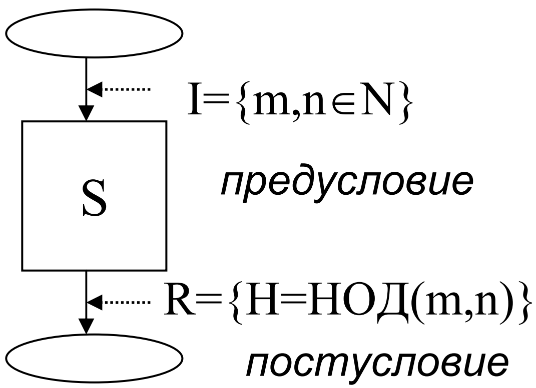
Средством описания ***подмножеств*** (бесконечных) множества является **Предикат.**

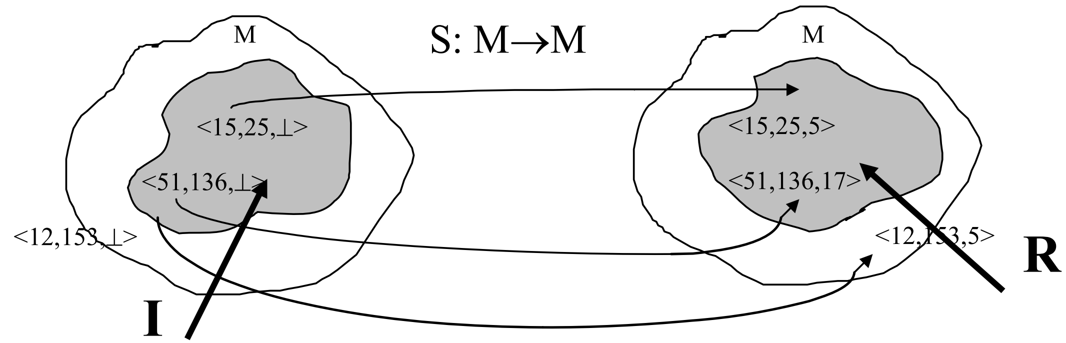
Например: x>y – все те пары x и y, в которых x > y (а их бесконечное число)

z+2x = y - определяет все тройки, в которых это отношение соблюдается

<51, 136,17> - удовлетворяет предикату H=НОД(m, n), а <12,153,5> - нет

Спецификация программ обработки данных (2)





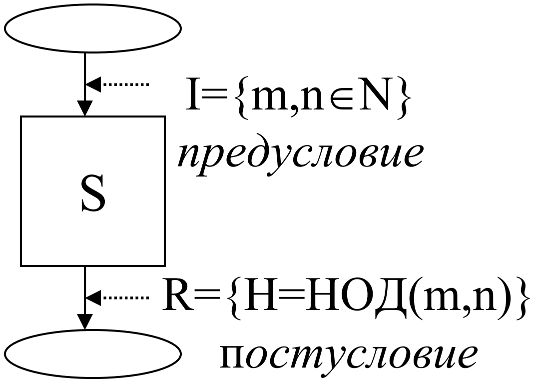
Формальная спецификация программы в двух формах:

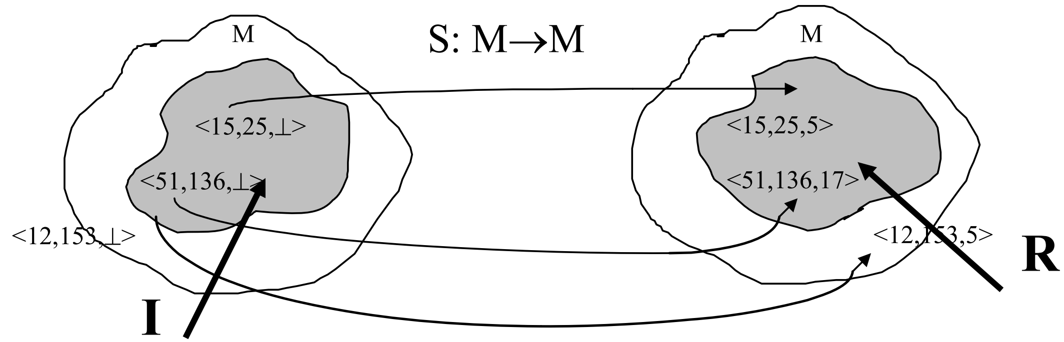
А: *"**Если** предусловие* **I** *истинно до начала работы программы* **S**,
И **S** *завершится,* ***ТО*** *после завершения* **S** *будет истинно постусловие* **R***"*
Это частичная (partial) корректность программы: { I } S { R }

В: *"**Если** предусловие* **I** *истинно до начала работы программы* **S**,
ТО **S** *завершится,* ***И*** *после завершения* **S** *будет истинно постусловие* **R***"*
Это полная (тотальная, total) корректность программы: [I] S [R]

Total correctness = Partial correctness + Termination

Свойство спецификации программ





Спецификация требований не связана с алгоритмом!

Примеры:

I$\equiv\{(\forall i\in 1,\dots,n)\ x_i\in\Re\}$; **R** $\equiv\{(\forall i\in 1,\dots,n\text{-}1)\ x_i\le x_{i+1}\}$ – все сортировки

{m,n∈N} S {(H=НОД(m,n)} – все программы нахождения НОД

PERM(A,a,N) =

$\exists$f: ($\forall$n: n≤N ⇒ f(n)≤N) ∧
($\forall$m,n: m≤N ∧ n≤N ∧ f(m)=f(n) ⇒ m=n) ∧
($\forall$n: n≤N ⇒ a[n] = A[f(n)] - условие перестановки массивов a и A

Простые примеры. Частичная корректность

{ I } S {R} – частичная корректность программы S относительно предусловия I и постусловия R

Пример: {X =1 } *S* {X = 2}

Если X=1 до начала программы S, *И S завершится, ТО после завершения* S *X*=2

{X =1 } *X:=X+1* {X = 2} утверждение истинно
{X =1 } *X:=X+2* {X = 2} утверждение ложно
{X=1 } *while True do Skip* {X=2} утверждение истинно

Простые примеры. Полная корректность

[I] S [R] –
полная корректность программы S относительно предусловия I и постусловия R

Пример: [X =1] *S* [X = 2]

Если до начала программы S *X=1, ТО S завершится И после завершения* S *X=2*

[X=1] *X:=X+1* [X=2] утверждение истинно
[X=1] *X:=2; while True do Skip* [X=2] утверждение ложно

Доказать завершение может быть сложно

Простые примеры

- {True} S {R} –

  когда S завершится, будет выполнено условие R
- {P} S {True} –

  истинно всегда, для любых P и S
- [True] S [R] –

  при любых исходных данных программа S завершается и ее результат удовлетворяет R
- [I] S [True] –

  если до начала программы S ее состояние удовлетворяет I, то она остановится (завершится)

Это только спецификация

Программа как преобразователь предикатов

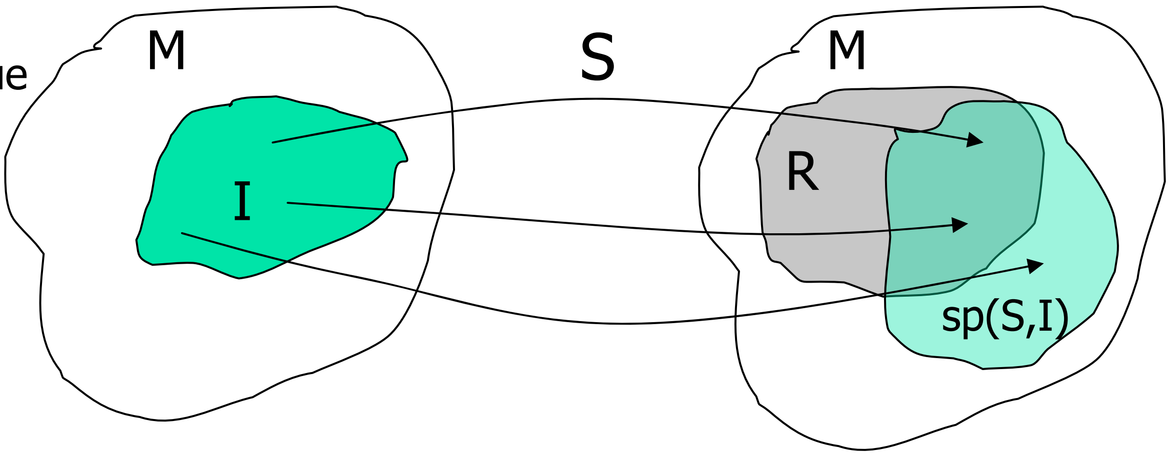
Программа преобразует данные, следовательно изменяется состояние программы. Поэтому и утверждения, которые были справедливы до начала программы, изменятся (и останутся справедливыми)

Программу можно рассматривать как *ПРЕОБРАЗОВАТЕЛЬ ПРЕДИКАТОВ* (Р. Флойд). По предусловию I в начале программы можно вычислить сильнейшее постусловие (strongest postcondition), sp(S,I)

Пример:

Пусть заданы предусловие (предикат) I и программа:

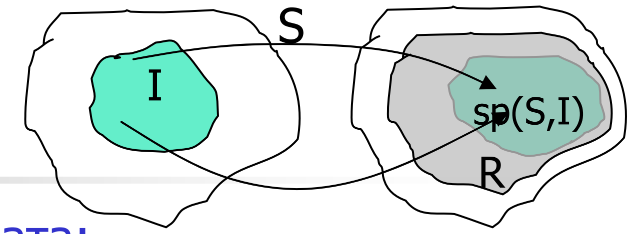
{I: x>z+5}
x:= x+3;
y:= 2;
sp(S,I) =?



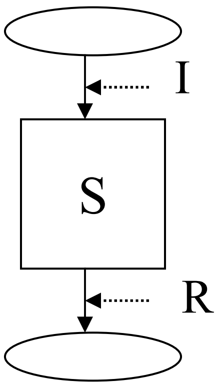
После *завершения* программы x возрастет на 3, а y станет равно 2

Поэтому: sp(S,I) ≡ x>z+8 & y=2

Корректность программы как преобразователя предикатов



Итак, у нас есть программа S и три предиката:

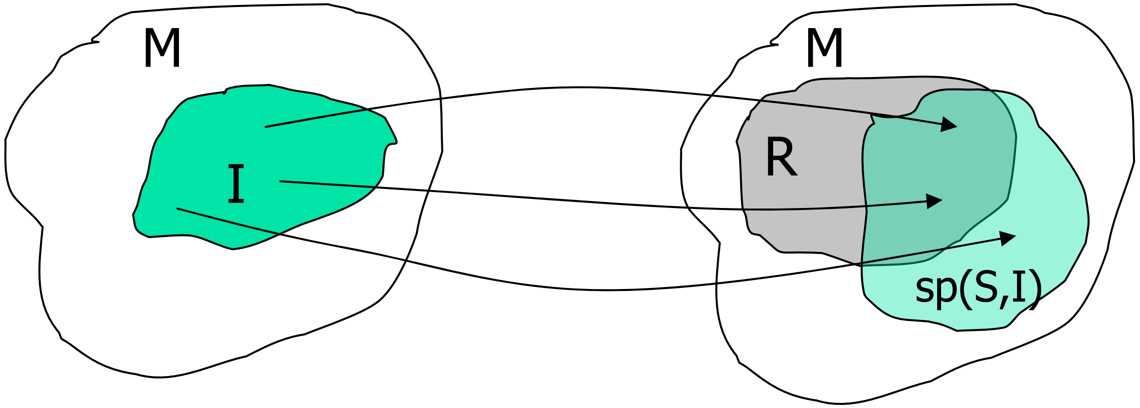


I – условие, которому должны удовлетворять программные переменные до выполнения программы (предусловие)

R – постусловие, которому должны удовлетворять программные переменные после выполнения программы (постусловие)

sp(S,I) – сильнейшее постусловие, которому точно удовлетворяют программные переменные после выполнения программы S после ее завершения, если они удовлетворяли предусловию I

Какое между ними должно быть соотношение, чтобы программа была корректна?



Программа S (частично) корректна тогда и только тогда, когда sp(S,I) целиком лежит в R

Как операторы языка преобразуют предикаты?

$I=\{x>z+5\}$
$x := x+3;$
sp (S, I) =?

Определим *формально*, как операторы языка преобразуют *любой* предикат

Пусть $I(x)$ – предусловие оператора $x := f(x)$

Обозначим x' значение *x после завершения* программы $x := f(x)$

Следовательно, $x' = f(x)$ (*именно равно, а не присвоить*!)

Тогда соотношения между старым x и *новым* x': $x = f^{-1}(x')$

Отсюда sp(S, I) = $I(x)$ (старое x не изменилось! I сформулировано для него)

Но нам нужно выражение исходного предиката I, которое сформулировано как функция от старого x, через новое значение x, т.е. через x′

Отсюда: $sp(S, I) = I(f^{-1}(x'))$

Это - ***формальная семантика*** оператора присваивания.

Для нашего примера: $x' := f(x) = x+3$; $f^{-1}(x')=x-3$; $sp(S, I)= x-3>z+5$

Школьный пример

- Задача для 7 класса:

“Если график функции $y = x^2 - 6x + 2$ сдвинуть на одну единицу вправо вдоль оси x, то график какой функции получится?”

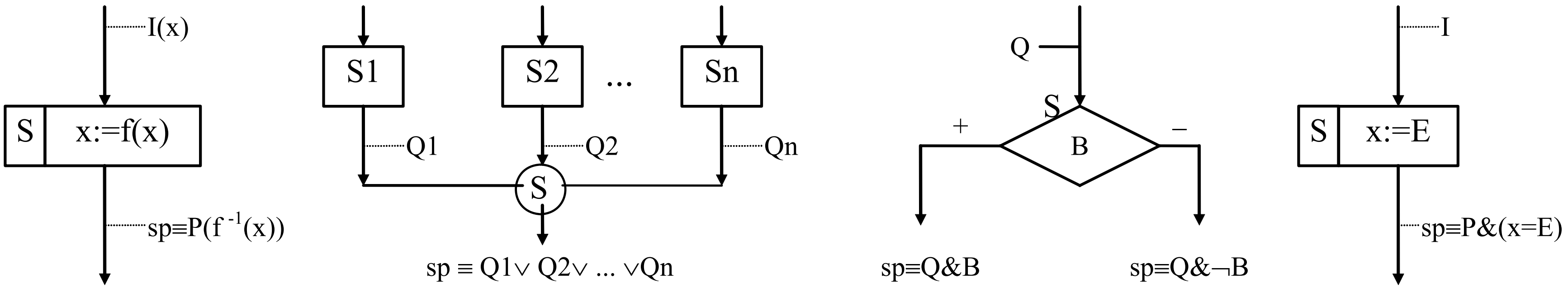
I={$y = x^2 - 6x + 2$}
$x := x+1$;
sp (S, I) =?

sp (I (f(x))) = I($f^{-1}(x)$) = {$y = (x-1)^2 - 6(x-1) + 2 = x^2 - 8x + 9$ }

Поэтому после сдвига графика на 1 вправо по оси OX, получится график функции

$y = x^2 - 8x + 9$

Сильнейшее постусловие программы

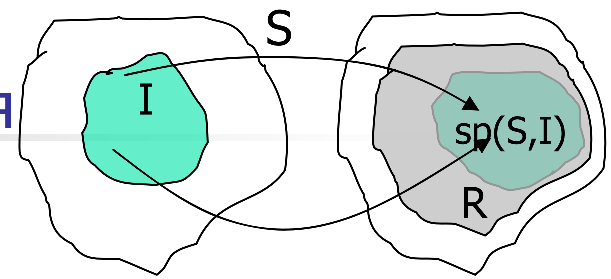


Формальная семантика операторов программы:

Сильнейшее постусловие sp(S,I) – это исходный предикат I, преобразованный оператором (программой) S

Сильнейшее *постусловие – поскольку предикату* sp(S, I) *удовлетворяют только те заключительные состояния программы* S, *в которые могут перейти все начальные состояния* **S**, *удовлетворяющие* **I**

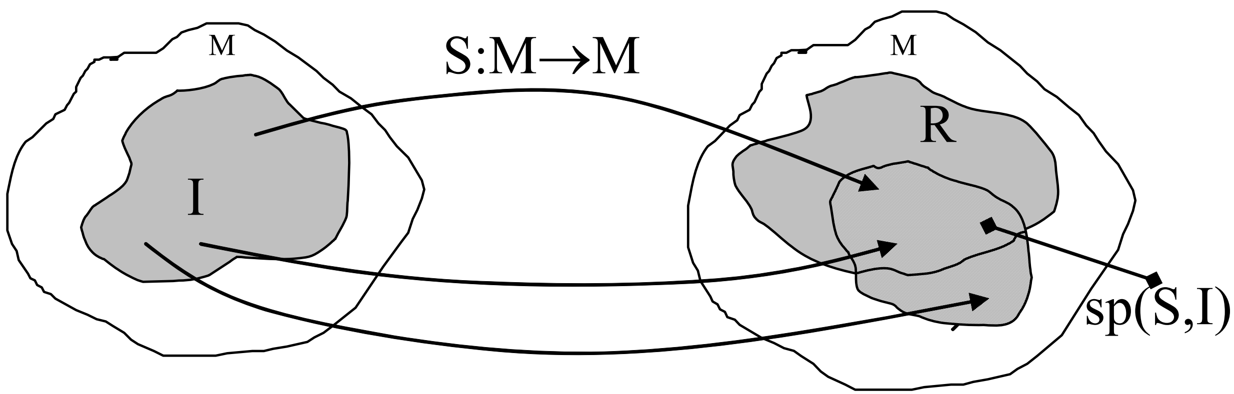
Проверка корректности программы на основе сильнейшего постусловия



Спецификация требований к программе S:

Пусть заданы S, предусловие **I** и постусловие **R** (ASSERTIONS)

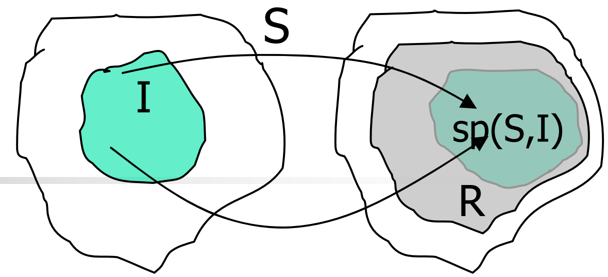
{I} S {R}: ***если*** I *истинно до начала* S, ***то*** R *истинно после завершения* S (I – предусловие, R – постусловие)



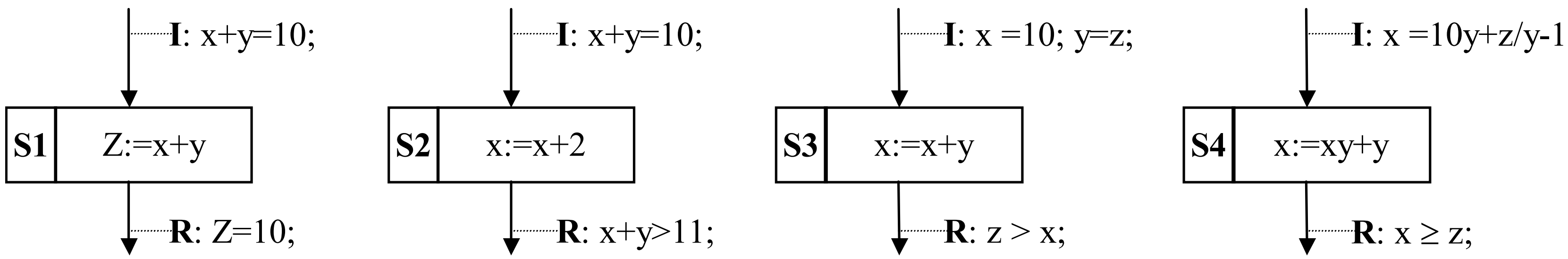
Теорема. *Утверждение* {I}S{R} *истинно тогда и только тогда, когда* R *является логическим следствием* sp(S,I).

Итак, программа S корректна, если sp (S,I) ⇒ R

Проверка корректности простейших программ



Примеры



Post(S, I) $\Rightarrow$(?) R

S1**: sp**(S1, I) $\equiv$ (x+y=10) & (z=x+y); Post(S1, I) $\Rightarrow$ Z=10 S1 корректна

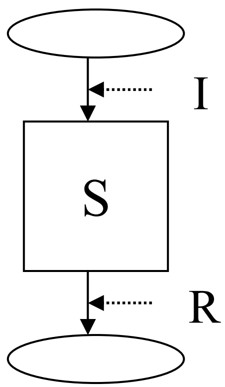
S2**:** sp(S2, I) $\equiv$ (x-2+y=10); Post(S2, I) $\Rightarrow$x+y >11 S2 корректна

S3**:** sp(S3, I) $\equiv$ (x-y=10) & (y=z); $\neg$[Post(S3, I) $\Rightarrow$ z>x] S3 некорректна

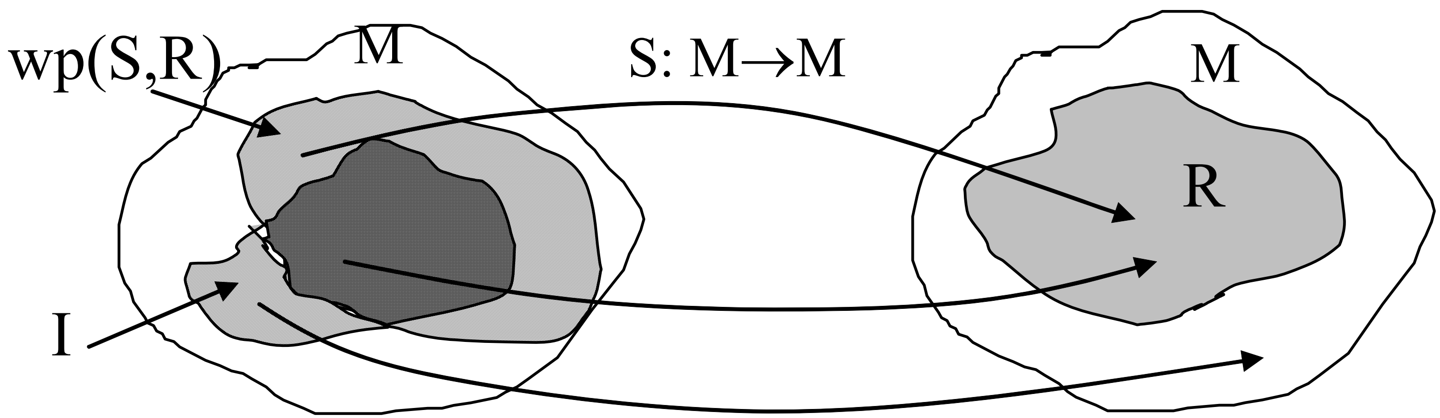
S4: sp(S4, I) $\equiv$ (x-y)/y=10y+z/y-1 $\equiv$ x = $10y^2$+z ; sp(4, I) $\Rightarrow$ x$\geq$z

S4 корректна

Слабейшее предусловие программы

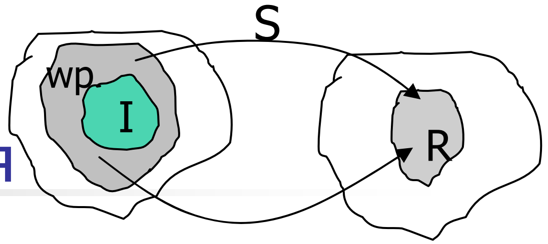


Формальная семантика программы – слабейшее предусловие: wp(S,R)– *предикат, которому удовлетворяют* ***все те*** *начальные состояния программы* S, *которые неизбежно приведут к такому заключительному состоянию* S, которое будет *удовлетворять* заданному предикату R



Программа S корректна, если I ⇒ wp(S,R)

Вычисление слабейшего предусловия



wp(S,R) = ?

x:= x+3;

z:= 2;

{R: x>z}

Неформально:

После *завершения* программы *x* возрастет на 3, а z станет равно 2. Для того, чтобы после завершения программы *x стало больше z,* нужно, чтобы *до начала программы* было *x*>-1

Определим *формально,* как по *любому* постусловию-предикату R и программе S построить слабейший предикат wp(S,R).

Пусть R(*x*) – предикат *после* выполнения оператора *x* := f(*x*). Обозначим *x*'- значение *x после завершения* оператора *x* := f(*x*).

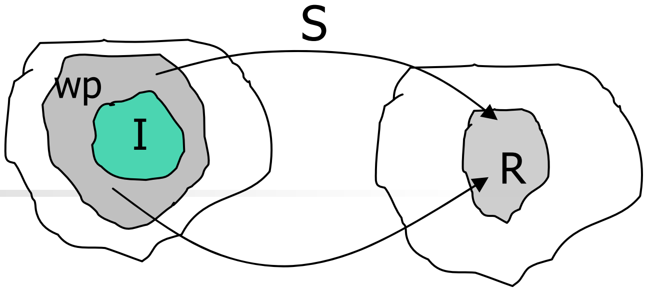
Следовательно, *x*' = f(*x*).

Предикат wp(S,R) выражен через новое значение *x*, т.е. через *x*'.

Отсюда wp(S,R) = R(*x*') = R(f(*x*)).

Для примера: wp(S, R) $\equiv$ $x' > z' \equiv x+3 > 2 \equiv x > -1$, т.е. получено то соотношение, которое ранее было найдено рассуждениями

Доказательство корректности простейших программ



Примеры

I: x+y=10; → **S1** | Z:=x+y → **R**: Z=10;

I: x+y=10; → **S2** | x:=x+2 → **R**: x+y>11;

I: x =10; y=z; → **S3** | x:=x+y → **R**: z > x;

I: x =10y+z/y-1; → **S4** | x:=xy+y → **R**: $x \geq z$;

$$I \Rightarrow (?)\ wp(S, R)$$

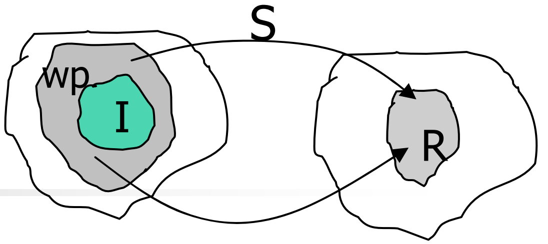
S1: $wp(S1, R) \equiv (x+y=10)$; $x+y=10 \Rightarrow wp(S1, R)$ S1 корректна

S2: $wp(S2, R) \equiv (x+2+y>11)$; $x+y=10 \Rightarrow wp(S2, R)$ S2 корректна

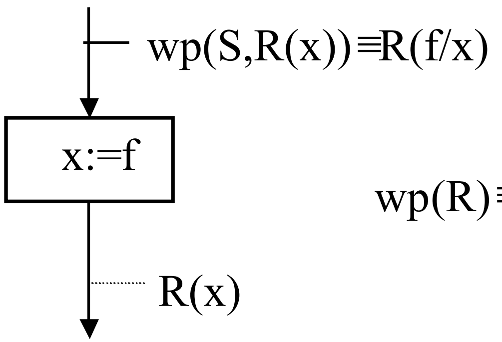
S3: $wp(S3, R) \equiv (z>x+y)$; $(x=10)\&(y=z) \not\Rightarrow (z>x+y)$ S3 некорректна

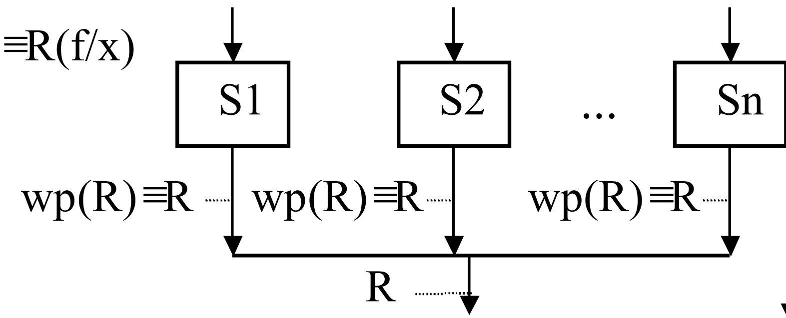
S4: $wp(S4, R) \equiv (xy+y>z)$; $x = 10y+z/y-1 \Rightarrow (xy+y>z)$ S4 корректна

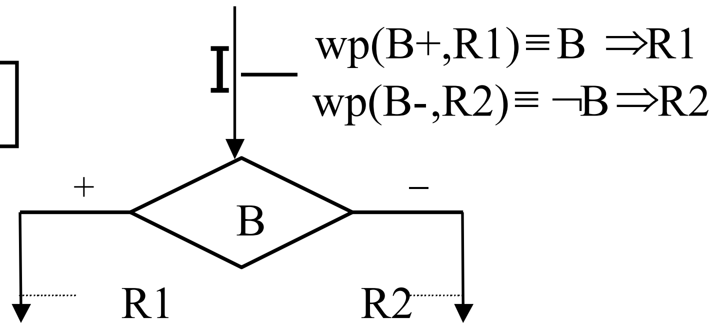
Формальная семантика операторов на основе слабейшего предусловия



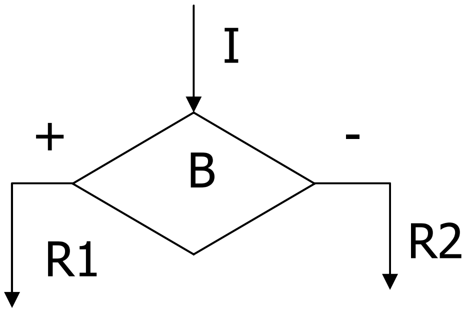
Правила построения wp(S,R)







Корректность условного оператора



Доказательство с помощью сильнейшего постусловия:

$I \;\&\; B \Rightarrow R1; \quad I \;\&\; \neg B \Rightarrow R2$

Доказательство с помощью слабейшего предусловия:

$I \Rightarrow (B \Rightarrow R1)\,;\; I \Rightarrow (\neg B \Rightarrow R2)$

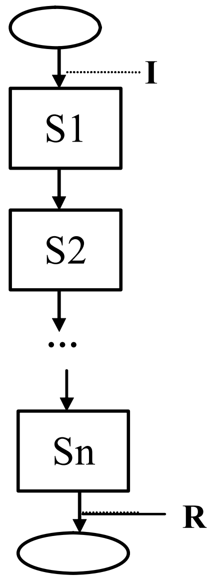
Являются ли два эти доказательства эквивалентными??

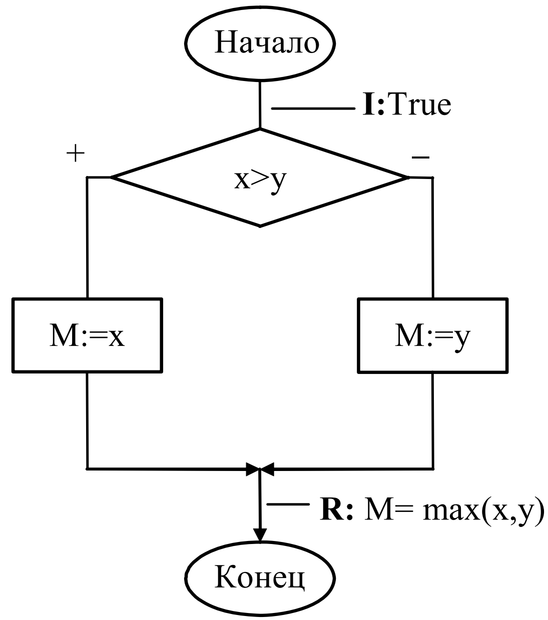
Сведем обе формулы к каноническому представлению - СКНФ

$I \;\&\; B \Rightarrow R1 =$ по свойству импликации
$\neg(I \;\&\; B) \vee R1 =$ по формуле де Моргана
$\neg I \vee \neg B \vee R1$

$I \Rightarrow (B \Rightarrow R1) =$ по свойству импликации
$\neg I \vee (\neg B \vee R1) =$ по ассоциативности дизъюнкции
$\neg I \vee \neg B \vee R1$

Доказательство корректности ациклических программ: "протаскивание" предикатов





Для доказательства программы нужно доказать теоремы:

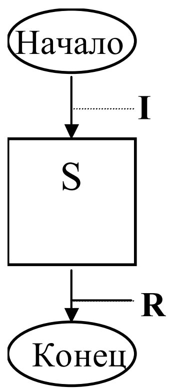
sp(S,I)⇒R: $[(x>y)\&(M=x)\Rightarrow M=\max(x,y)]\ \&\ [\neg(x>y)\&(M=y)\Rightarrow M=\max(x,y)]$

либо

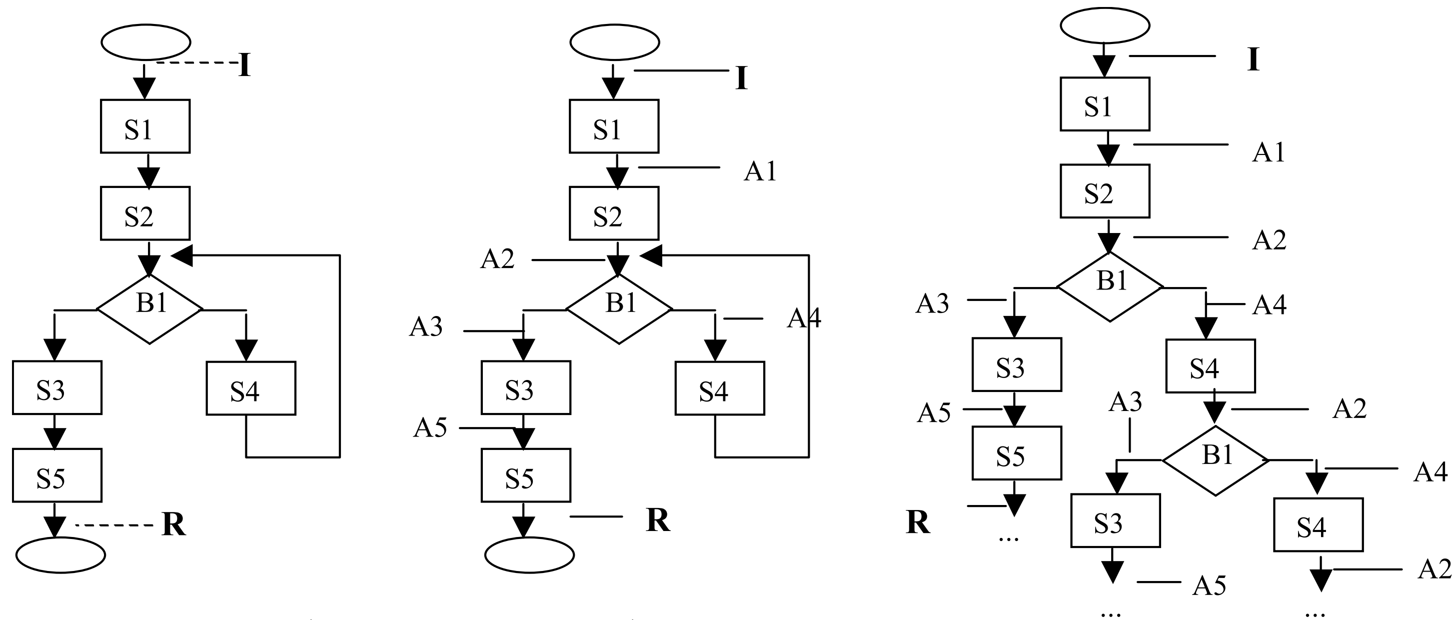
I ⇒ wp(S,R) : $[True \Rightarrow (x>y \Rightarrow x=\max(x,y))\ \&\ [True \Rightarrow (\neg(x>y) \Rightarrow y=\max(x,y))]$

$\equiv [x>y \Rightarrow x=\max(x,y)]\ \&\ [\neg(x>y) \Rightarrow y=\max(x,y)]$

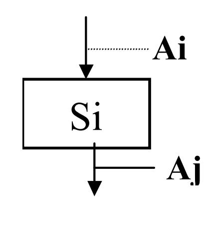
Индуктивный метод Р.Флойда доказательства частичной корректности программ (1967)



$\{\mathbf{I}\}S\{\mathbf{R}\}$?



Роберт Флойд: каждая стрелка блок-схемы снабжается утверждением, и для каждого блока Si доказывается теорема {Ai} Si {Aj}



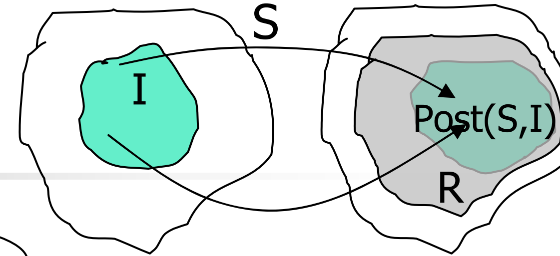
Индуктивное утверждение:

если **I** истинно, то каждое утверждение, **встретившееся** на пути вычислений, истинно

По индукции следует $\{\mathbf{I}\}S\{\mathbf{R}\}$ – но только ***если программа завершается*** (т.е. **R когда-нибудь** встретится на каждом пути)

Ai - Assertions

Пример: программа нахождения максимума



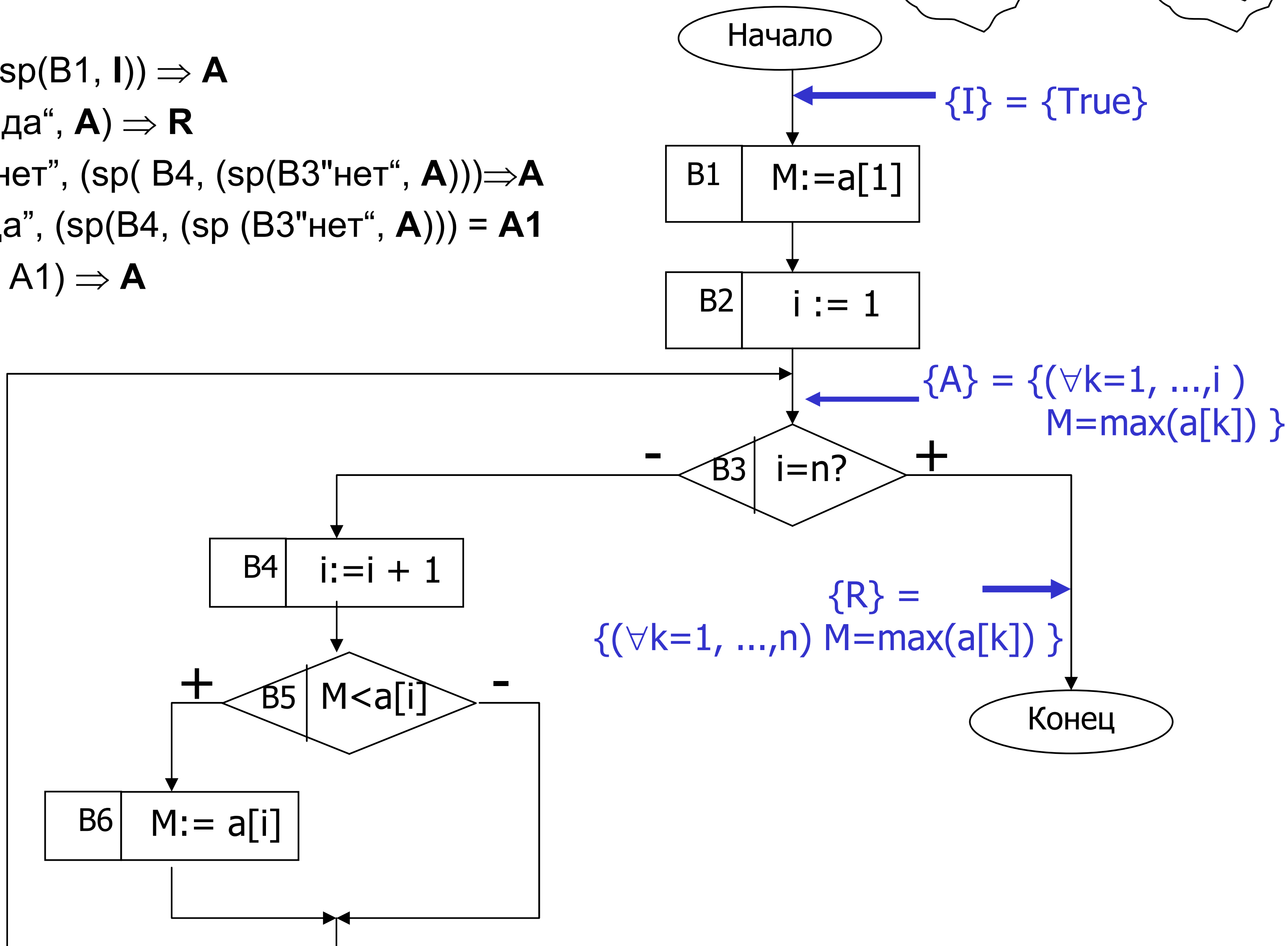
T1: sp (B2, sp(B1, **I**)) ⇒ **A**

T2: sp (B3 "да", **A**) ⇒ **R**

T3: sp(B5"нет", (sp(B4, (sp(B3"нет", **A**)))⇒**A**

T4: sp(B5"да", (sp(B4, (sp (B3"нет", **A**))) = **A1**

T5: sp(B6, A1) ⇒ **A**



Инвариант цикла: задача о кофейных зернах

Задача. В банке находится несколько черных и белых кофейных зерен

- Пока это возможно, повторяем следующий процесс:
  случайно выбираем из банки два зерна;

    если они одного цвета, то выбрасываем их, а в банку кладем черное;

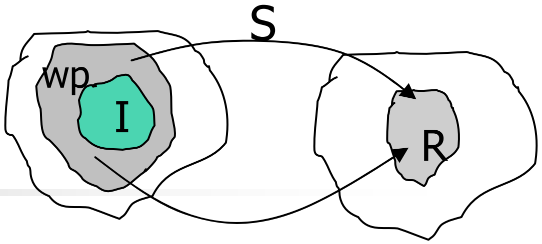
    если они разного цвета, то черное выбрасываем, белое кладем обратно.

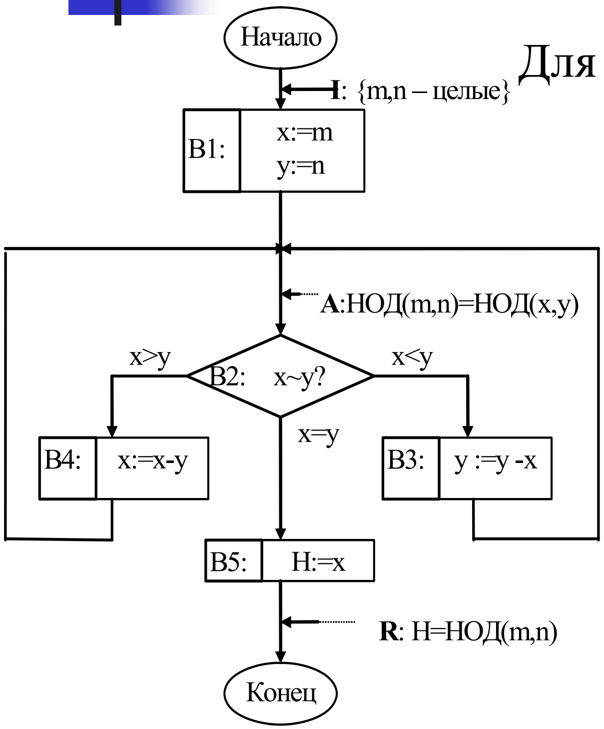
  В конце концов в банке останется всего одно зерно (ДОКАЗАТЬ).
- **ВОПРОС:** какого цвета это оставшееся зерно, если известно, сколько было вначале в банке черных и белых зерен?
- Попытка решить задачу, используя тестовые примеры, обычно не приводит к решению. Нужно задачу, алгоритм ПОНЯТЬ!
- Если решение каким-то образом найдено, то обосновать его почти невозможно без привлечения понятия *инварианта цикла,* т.е. такого свойства, которое будет истинным вначале и остается истинным после выполнения каждого шага цикла
- Инвариант: четность белых не меняется. Поэтому :

  Решение : *было четное число белых – останется черное,*
  было нечетное число белых – останется белое

  Гарантия завершения: на каждом шаге число зерен уменьшается на 1

Доказательство частичной корректности программы НОД





Для доказательства программы нужно доказать четыре теоремы:

1. $\mathbf{I} \Rightarrow$ wp(B1, **A**)
2. $\mathbf{A} \Rightarrow$ wp ($B2_{x<y}$, wp(B3,**A**))
3. $\mathbf{A} \Rightarrow$ wp ($B2_{x>y}$, wp(B4,**A**))
4. $\mathbf{A} \Rightarrow$ wp ($B2_{x=y}$, wp(B5,**R**))

или:

1. m,n – целые $\Rightarrow$ НОД(m,n)=НОД(m,n)
2. НОД(m,n)=НОД(x,y)$\Rightarrow$[x<y$\Rightarrow$НОД(m,n)=НОД(x,y-x)]
3. НОД(m,n)=НОД(x,y)$\Rightarrow$[x>y$\Rightarrow$НОД(m,n)=НОД(x-y,y)]
4. НОД(m,n)=НОД(x,y)$\Rightarrow$[x=y$\Rightarrow$ x=НОД(m,n)]

Программа снабжается кроме **I** и **R** ***только одним*** дополнительным утверждением: ***инвариантом цикла*** A

Программа ***частично корректна:*** при отрицательном m либо n программа входит в бесконечный цикл

Формулировка утверждений требует проникновения в суть алгоритма

Аксиоматический метод А.Хоара (1969)

Метод Флойда сводит доказательство программы к доказательству отдельных теорем

Антони Хоар предложил свести верификацию к строгому доказательству теорем на основе аксиом и правил вывода, рассматривая формальную логическую систему, обеспечивающую порождение точных утверждений о выполнении программы

А.Хоар выбрал язык программирования, который оперирует только с целыми и имеет следующие операторы:

x:= f - оператор присваивания
S1; S2 - последовательность операторов
if B **then** S1 **else** S2 **fi** - условный оператор
while B **do** S **od** - оператор цикла

Логика программирования

$$\frac{H1;\ H2;\ \ldots,\ Hn}{C}$$

Логическая система состоит из:

- Формул, построенных из символов по определенным правилам
- Выделенных формул, называемых аксиомами (они предполагаются ИСТИННЫМИ)
- Правил вывода одних истинных формул из других истинных формул
- **Истинная формула исчисления называется теоремой**

Логика программирования – это логическая система для доказ св-в программ:

- Формула – это просто предикат или {P}S{Q}, где P и Q – предикаты, S - программа
- Аксиомы (для указанного языка): – одна аксиома присваивания $\{P_{X \leftarrow E}\}$ X:=E {P}, аксиомы арифметики, аксиомы прикладной области (например, НОД(x,x) = x)
- Правила вывода:

Правило композиции:

$$\frac{\{P\}\ S1\ \{P1\};\ \{P1\}\ S2\ \{Q\}}{\{P\}\ S1;\ S2\ \{Q\}}$$

Правило следствия:

$$\frac{I \Rightarrow P;\ \{P\}\ S\ \{Q\};\ Q \Rightarrow R}{\{I\}\ S\ \{R\}}$$

Правило итерации:

$$\frac{\{P \& B\}\ S\ \{P\}}{\{P\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{P \& \neg B\}}$$

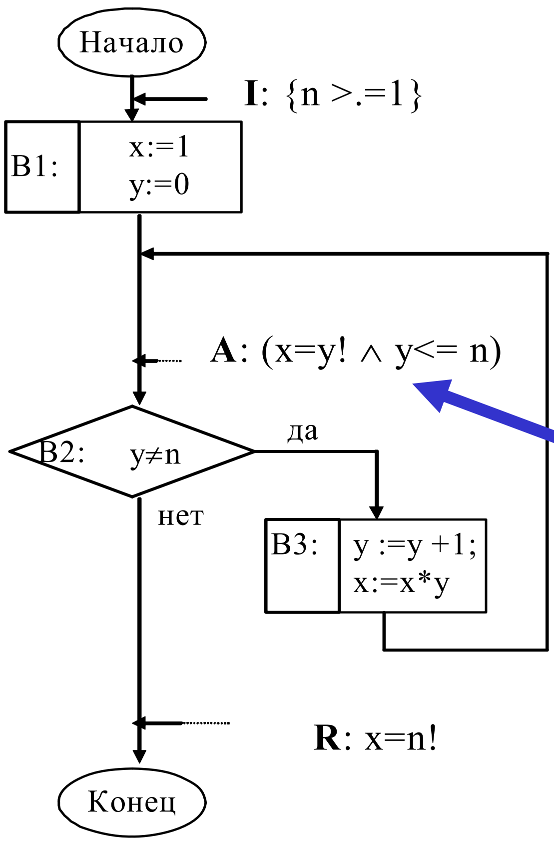
Правило выбора 1:

$$\frac{\{P \& B\}\ S\ \{Q\};\ \ P \& \neg B \Rightarrow Q}{\{P\}\ \mathbf{if}\ B\ \mathbf{then}\ S\ \mathbf{fi}\ \{Q\}}$$

Правило выбора 2:

$$\frac{\{P \& B\}\ S1\ \{Q\};\ \ \{P \& \neg B\}\ S2\ \{Q\}}{\{P\}\ \mathbf{if}\ B\ \mathbf{then}\ S1\ \mathbf{else} S2\ \mathbf{fi}\ \{Q\}}$$

Пример аннотированной программы: n!



{I: n≥1}
x:=1;
y:=0;
{A: x=y! ∧ y≤n }
  while (y≠n)
    do
      y:=y+1;
      x:=x*y
  od
{ R: x=n!}

Инвариант

Примеры доказательства методом Хоара

Примеры:

- $\{X = 1\}\ X := X+1\ \{X = 2\}$ теорема
- $X > 5 \Rightarrow X \geq 3$ теорема
- $\{X = 1\}\ X := X+1\ \{X = 3\}$ не теорема

Доказательство – это цепочка формул, каждая из которых либо аксиома, либо теорема, полученная из теорем и аксиом по правилам вывода

Доказать:

$\{ x > 3 \}$
if x < 25 **then** x := 2 * x **fi**
$\{ x \geq 8 \}$

Структура:

$\{ P \}$
if B **then** S **fi**
$\{ Q \}$

Доказательство:

1. $\{ 2*x>=8 \}$ x := 2*x $\{ x \geq 8 \}$ - Аксиома $\{R\}\ S\ \{Q\}$
2. $(x > 3)\ \&\ (x < 25) \Rightarrow (2*x>=8)$ $P\&B \Rightarrow R$

   доказываем из аксиом арифметики (x – целое!)
3. $\{P\ \&\ B\}\ S\ \{Q\}$ - по Правилу следствия,

   поскольку $\{R\}\ S\ \{Q\}$ и $P\&B \Rightarrow R$
4. $x>3\ \&\ x >= 25 \Rightarrow x \geq 8$ доказываем $P\&\neg B \Rightarrow Q$
5. { P } **if** B **then** S **fi** { Q } по Правилу выбора 1

Правило следствия:

$$\frac{I \Rightarrow P;\ \{ P \}\ S\ \{ Q \};\ Q \Rightarrow R}{\{ I \}\ S\ \{ R \}}$$

Правило выбора 1:

$$\frac{\{P\&B\}\ S\ \{Q\};\ \ P\&\neg B \Rightarrow Q}{\{ P\}\ \textbf{if}\ B\ \textbf{then}\ S\ \textbf{fi}\ \ \{Q \}}$$

Семантика параллельного выполнения

Оператор синхронизации:

await (B) S

{ I } S { R } называются
“Хоаровскими тройками”

Правило для **await:**

$$\frac{\{P\&B\}\ S\ \{Q\}}{\{P\}\ \textbf{await}\ (B)\ S\ \{Q\}}$$

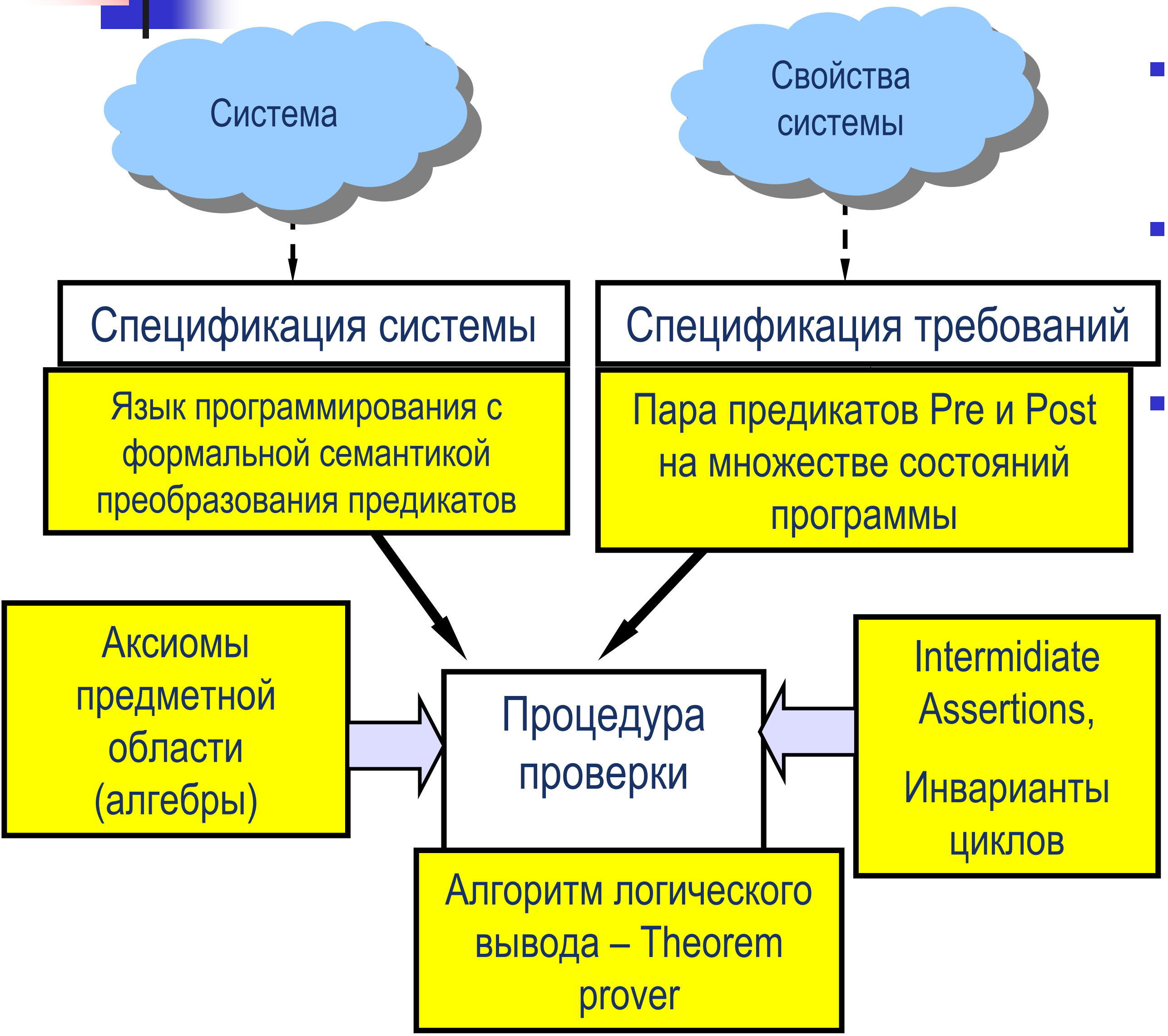
Оператор параллельного выполнения:

co S1 || S2 || … || Sn **oc**

Правило для **co-oc:**

$$\frac{\{P_i\}\ S_i\ \{Q_i\}\quad \textit{и все } S_i \textit{ не влияют друг на друга}}{\{P_1\&\ldots\&P_n\}\ \textbf{co}\ S_1 \,||\, S_2 \,||\, \ldots \,||\, S_n\ \textbf{oc}\ \{Q_1\&\ldots\&Q_n\}}$$

Индуктивный метод верификации программ (HOL, Isabelle , …)



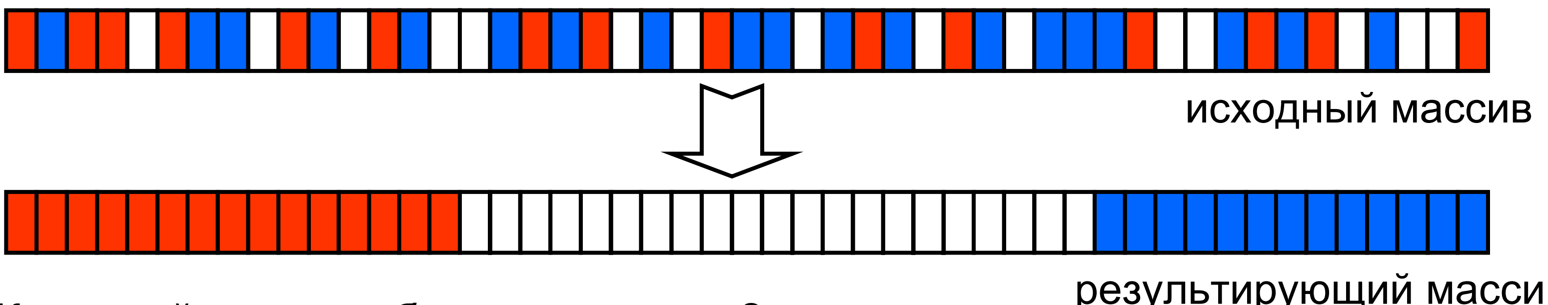
- Спецификация – на языке программирования
  - Для операторов языка определена ***семантика преобразования предикатов***
- Свойства программы
  - Описываются как пара вход-выходных утверждений
  - Добавляются инварианты циклов и промежуточные утверждения, которые отражают идею алгоритма
- Процедура проверки
  - Не очень эффективная (частично автоматизированная) система доказательства теорем, работающая с помощью пользователя

Пример верификации с помощью theorem prover (с использованием системы Isabelle)

- Сообщение СМИ 01.10.2009
  - Компания NISTA (Australia) Research Centre of Excellence, объявила сегодня о завершении работы по формальному доказательству корректности ядра ОС. Доказанная ОС - The Secure Embedded L4 (seL4) microkernel содержит 7,500 строк C code. Было формально доказано 10,000 промежуточных теорем, доказательство заняло 200,000 строк. Доказательство было поддержано интерактивной системой доказательства теорем Isabelle
  - Это результат 4-х летнего труда группы из 12 исследователей NICTA под руководством Dr Klein, PhD студентов и нескольких других работников
  - Этот экстраординарный результат открывает путь к разработке нового поколения ПО с беспрецедентным уровнем надежности. Это одно из самых длинных из когда-либо выполненных формальных доказательств с помощью формальных средств theorem-proving

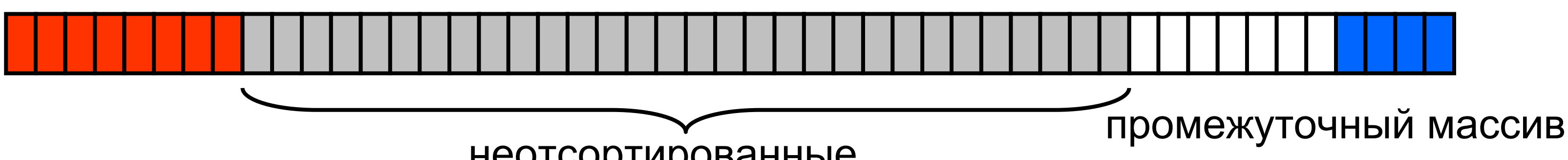
Разработка корректных программ

- Э.Дейкстра: “*Программы нужно сразу строить корректными*”. В книге “Дисциплина программирования” приведено множество примеров такого подхода к разработке
- “Голландский национальный флаг” – сортировка массива из трех типов объектов – “красных”, “белых” и “синих”, которые должны в результате стоять именно в этом порядке. Программа должна справляться со всеми случаями вырождения: отсутствием одного или нескольких цветов



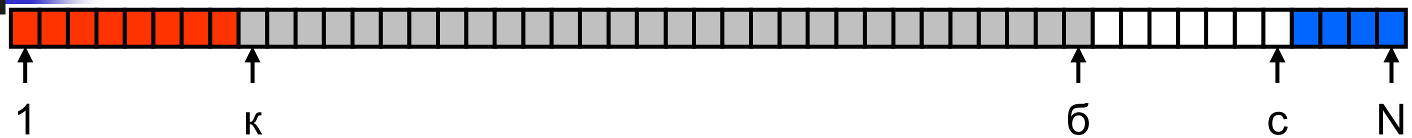
Как подойти к разработке программы?

Выбрать ИНВАРИАНТ!



Крайними состояниями инварианта являются исходный и результирующий массивы

Формализация инварианта



$$
\begin{aligned}
A \equiv\ & [\ \forall i: (1 \le i < к)\ ]\quad M[i] \text{ - красный} \\
\&\ & [\ \forall i: (б < i \le с)\ ]\quad M[i] \text{ - белый} \\
\&\ & [\ \forall i: (с < i \le N)\ ]\quad M[i] \text{ - синий} \\
\&\ & [\ \forall i: (к \le i \le б)\ ]\quad M[i] \text{ - неотсортированный} \\
\&\ & (к \ge 1)\ \&\ (б \le с)\ \&\ (с \le N)
\end{aligned}
$$

Очевидно, что:
$A\ \&\ (к > б) \Rightarrow$ массив M отсортирован
(массив неотсортированных пуст)

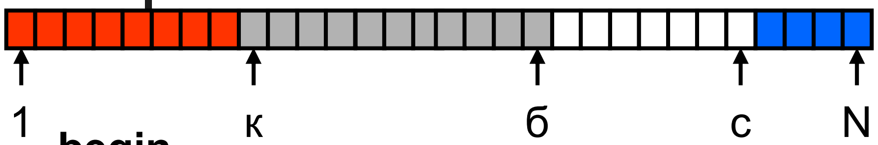
Разработка программы сводится к построению цикла, который при сохранении инварианта уменьшает зону неотсортированных

```
begin
  { [∀i: (1≤i≤N) ] M[i] - неотсортированные}
  к,б,с:=1,N,N;
  /* инвариант A стал истинным */
do
  <пока зона не отсортированных непуста> →
  <уменьшить зону не отсортированных, не изменяя A>
od
  { R≡ массив M отсортирован }
end
```

Программа частично корректна, если A – инвариант цикла

Программа полностью корректна, если зона неотсортированных уменьшается на каждом шаге цикла

Разработка программы, корректной по построению



$A \equiv [\,\forall i: (1 \le i < к)\,]$ M[i] - красный
$\&\ [\,\forall i: (б < i \le с)\,]$ M[i] - белый
$\&\ [\,\forall i: (с < i \le N)\,]$ M[i] - синий
$\&\ [\,\forall i: (к \le i \le б)\,]$ M[i] - неотсортированный
$\&\ (к \ge 1)\ \&\ (б \le с)\ \&\ (с \le N)$

```
begin
{ [∀i: (1≤i≤N)]  M[i] – неотсортированный }
 к,б,с:=1,N,N;
{A ≡ инвариант цикла}
do
 к ≤б →
   if    M[б] = белый    → б:=б-1;
      [] M[б] = красный → поменять местами (M[б], M[к]); к:=к+1;
      [] M[б] = синий → поменять местами (M[б], M[с]); с, б:=с-1, б-1;
   fi
od
{R≡массив M отсортирован}
end
```

Охраняемые команды Дейкстры:

```
begin
{ I }
 S0
{A}
do
 G →
   if    G1 → S1;
      [] G2 → S2;
      [] G3 → S3
   fi
od
{R}
end
```

Для верификации программы нужно доказать:

1. sp (S0, I) $\Rightarrow$ A или I $\Rightarrow$ wp(S0, A)
2. $A\&\neg G \Rightarrow R$, т.е. $A\&(к<б) \Rightarrow R$
3 -5. sp(Si, A&G&Gi)$\Rightarrow$ A или A$\Rightarrow$wp(Si, A&G&Gi)
6. ***Завершение программы***: нужно доказать, что **б-к** уменьшается при каждом прохождении цикла

Guarded Commands – охраняемые команды

Предложены Э. Дейкстрой, использованы и в других приложениях, например, в языке Холзманна Promela (Protocol Meta Language)

В условном операторе при истинности нескольких защит выбирается случайно одна любая. Если все защиты ложны, то это ошибка

Оператор цикла завершается, если ни одна из защит не является истинной

Guarded Command: :: G → Op,

Оператор присваивания: x, y,z := E1,E2,E3

Условный оператор: ***if***

:: <command1>

:: …

:: <command n>

fi

Оператор цикла:

do

:: <command1>

:: …

:: <command n>

od

Пример: Алгоритм Эвклида в этом языке запишется так:

```
x, y := m, n
do
   :: x>y → x:=x-y
   :: y>x → y:=y-x
od
print x
```

Общая характеристика дедуктивной верификации

- Дедуктивная верификация имеет две составных части:
  - верифицируемые свойства представляются в виде предикатов – логических формул, а программа – как преобразователь предикатов;
  - доказательство корректности программы сводится к доказательству выполнимости логической формулы дедукцией (логическим выводом) в подходящем логическом исчислении (со сформулированными аксиомами и правилами вывода)
- Семантика программ (то, как конструкции программы преобразуют предикаты) может быть представлена очень точно:
  - неограниченные структуры данных (целые, списки, деревья и т.п.);
  - неограниченные программные конструкции (рекурсия, циклы)

  Это позволяет представить и проверить тонкие свойства программ
- Достижение этой точности имеет свою цену: трудно предсказать, какой объем ресурсов (времени, памяти) может потребоваться для выполнения задачи верификации конкретной программы: в общем случае, проблема доказательства свойств программ, работающих с бесконечными структурами, алгоритмически неразрешима

Заключение

- Тестированием нельзя доказать корректность программы
- Тестирование – проверка работы программы на одном конкретном входном ее состоянии. Предикат (ASSERTION) описывает бесконечное множество возможных состояний. Рассматривая программу как преобразователь предикатов, можем рассуждать о корректности программы
- Доказательство корректности программы методом Флойда-Хоара сводится к расстановке утверждений в теле программы и доказательству теорем логики первого порядка, полученных из преобразованных предикатов в некоторой дедуктивной системе
- Спецификация программы – это пара предикатов <*входной-выходной*> для каждого блока. Как минимум, дополнительно требуется один предикат в каждом цикле (*инвариант цикла*) вдобавок к начальному и заключительному. Формулировка утверждений требует проникновения в существо, основные идеи алгоритма, часто требует полного понимания (ultimate understanding) программы
- Формальная семантика языка задается сильнейшим постусловием либо слабейшим предусловием для каждого типа операторов и правилами вывода

Заключение (2)

- Индуктивный метод Флойда и аксиоматический метод Хоара доказывают только частичную корректность программы; полная (total) корректность требует дополнительного доказательства завершаемости
- Метод Хоара – это формализация метода Флойда: сведение доказательства к формальной модели исчисления с аксиомами и правилами вывода, т.е. к аксиоматической системе
- Метод не может быть полностью автоматизирован из-за неполноты логики
- Сегодня есть мощные инструменты, частично автоматизирующие доказательство (например, HOL(Higher Order Logic), Isabelle и PVS (Prototype Verification System))
- Реальное доказательство требует "ручного сопровождения" – определения тысяч детальных лемм
- Пример: Isabelle Verification of BDD Normalization (алгоритм нормализации Брайанта доказан в 2005 г.)
- Пример: доказательство корректности алгоритма деления с плавающей точкой в процессоре AMD5K86 потребовало введение 1600 определений и лемм [1]
- В книге "*Дисциплина программирования*" Э.Дейкстра предложил методику разработки программ совместно с доказательством их корректности, т.е. корректных по построению

[1] Brock B. et.al. *ACL2 theorems about commercial microprocessors*. Proc. of the Formal Methods in Computer-Aided Design, Nov.1996, p.275-293

Персоналии: Роберт Флойд

- Robert Floyd (1936-2001) – выдающийся ученый в области информатики (США)
- Закончил школу в 14, получил Бакалавра в 17, стал профессором в 27 (Carnegie Mellon U), полным профессором в 33 (Стандфордского университета).
- Получил премию Тьюринга в 1978 за вклад в методологию в следующих областях CS:
  - создание эффективного надежного ПО
  - терия синтаксического анализа (парсинг)
  - семантика языков программирования
  - автоматическая верификация программ
  - автоматический синтез программ
  - анализ алгоритмов

(Wikipedia)

Персоналии: Антони Хоар

- Sir Charles Antony Richard Hoare (род. 1934), известный, как Антони Хоар или C.A.R. Hoare – выдающийся английский ученый в области информатики
  - наиболее известен созданием Quicksort- наиболее широко используемых алгоритмов сортировки (в возрасте 26 лет)
  - Реализовал Алгол-60
- Разработал также
  - "хоаровскую логику программ" для проверки корректности программ
  - Формальный язык CSP (Communicating Sequential Processes), на основе которого создан язык программирования Occam (след лекция)
- Получил премию Тьюринга 1980 г. за "фундаментальный вклад в определение и разработку языков программирования"

(Wikipedia)

Персоналии: Эдсгер Дейкстра

- Edsger W.Dejkstra (1930 - 2002) – выдающийся голландский ученый в области информатики
  - разработал первую ОС THE для работы с параллельными процессами
  - вместе с двумя другими учеными построил одну из первых ЭВМ X1 (1956) за год. Для разводки печатных плат разработал алгоритм поиска кратчайшего пути на графе, известный как "алгоритм Дейкстры"
  - участвовал в разработке Алгол-60 и компилятора для него
- Разработал также
  - структурное программирование
  - нотацию ПОЛИЗ при трансляции
  - алгоритм банкира для синхронизации процессов при захвате ресурсов
  - семафоры для синхронизации процессов
  - с 1970 года основные интересы – формальная верификация, математическое доказательство корректности. Книга Дисциплина программирования – об этом
- Стал лауреатом премии Тьюринга 1972 г. за "фундаментальный вклад в разработку языков программирования"

(Wikipedia)

Задачи

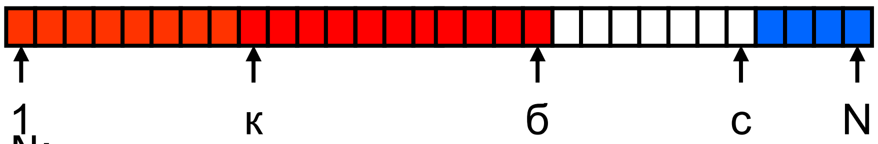
1. Доказать корректность:
   - $\{X=x \wedge Y=y\}$ X:=X+Y; Y:=Y-X; X:=X-Y $\{Y=x \wedge X=y\}$
   - {X=R+Y*Q} R:= R-Y; Q:= Q+1 {X= R+Y*Q}
2. Построить программу суммирования элементов массива и доказать ее корректность
3. Разработать программу определения минимального элемента массива и доказать ее корректность
4. Доказать корректность следующей программы возведения *x* в степень *y*:

```
function exponential (x,y);
  int x,y;
  begin
     int i, z;
       z=1; i=1;
       while (i≠y) do
          z*=x; i=i+1;
      return z;
 end
```

5. Построить и доказать программу “*Национальный флаг королевства Маврикий*”. Флаг Маврикия имеет четыре полосы: красная, синяя, желтая, зелёная.

Задачи (продолжение)

6. Проверить корректность второго варианта программы "*Голландский национальный флаг"*, анализируя самый левый элемент зоны не отсортированных элементов:




```
begin
  к,б,с:=1,N,N;
do
  к ≤ б →
     if   M[к] = красный → поменять местами (M[к], M[б]); к:=к+1;
      [] M[к] = белый → б:=б-1;
      [] M[к] = синий → поменять местами (M[к], M[с]); с:=с-1;
                         поменять местами элементы (M[к], M[б]); б:=б-1;
     fi
od
End
```

7. Доказать методами Флойда и Хоара программу вычисления факториала
8. Построить программу вычисления факториала по спецификации {X=n} S {Y=n!}

Спасибо за внимание