*Встречаются две подруги. Одна говорит другой:*

* *С моим мужем творится что-то странное. Приходит с работы, наливает полную ванну воды, берет удочку и весь вечер ловит рыбу.*
* *А почему ты не обратишься к врачу?*
* *Надо бы. Но так хочется свежей рыбки!*

# Анекдот

***Лекция 9.***

 **ДОКАЗАТЕЛЬСТВО СВОЙСТВ ПРОГРАММ**

*Понятие обоснования программ. Формализация свойств программ, триады Хоора. Правила для установления свойств оператора присваивания, условного и составного операторов. Правила для установления свойств оператора цикла, понятие инварианта цикла. Завершимость выполнения программы.*

**9.1. Обоснования программ. Формализация свойств программ.**

Для повышения надежности программных средств весьма полезно снабжать программы дополнительной информацией, с использованием которой можно существенно повысить уровень контроля ПС. Такую информацию можно задавать в форме неформализованных или формализованных утверждений, привязываемых к различным фрагментам программ. Будем называть такие утверждения *обоснованиями* программы. Неформализованные обоснования программ могут, например, объяснять мотивы принятия тех или иных решений, что может существенно облегчить поиск и исправление ошибок, а также изучение программ при их сопровождении. Формализованные же обоснования позволяют доказывать некоторые свойства программ как вручную, так и контролировать (устанавливать) их автоматически.

Одной из используемых в настоящее время концепций формальных обоснований программ является использование так называемых триад Хоора. Пусть *S* — некоторый обобщенный оператор над информационной средой *IS*, а *P* и *Q* — некоторые предикаты (утверждения) над этой средой. Тогда запись {*P*}*S*{*Q*} и называют *триадой Хоора*, в которой предикат P называют *предусловием*, а предикат *Q* — *постусловием* относительно оператора *S*. Говорят, что *оператор* (в частности, программа) *S обладает свойством* {*P*}*S*{*Q*}, если всякий раз, когда перед выполнением оператора *S* истинен предикат *P*, после выполнения этого оператора *S* будет истинен предикат *Q*.

Простые примеры свойств программ:

(9.1) { *n* = 0 } *n* := *n* + 1 { *n* = 1 },

(9.2) { *n* < *m* } *n* := *n* + *k* { *n* < *m* + *k* },

(9.3) { *n* < *m* + *k* } *n* := 3 \* *n* { *n* < 3 \* ( *m* + *k* ) },

1. { *n* > 0 } *p* := 1; *m* := 1;

 ПОКА *m* <> *n* ДЕЛАТЬ

*m* := *m* + 1; *p* := *p* \* *m*

 ВСЕ ПОКА

{ *p* = *n*! }.

Для доказательства свойства программы *S* используются свойства простых операторов языка программирования (мы здесь ограничимся пустым оператором и оператором присваивания) и свойствами управляющих конструкций (композиций), с помощью которых строится программа из простых операторов (мы здесь ограничимся тремя основными композициями структурного программирования, см. Лекцию 8). Эти свойства называют обычно правилами верификации программ.

**9.2. Свойства простых операторов.**

Для пустого оператора справедлива

*Теорема 9.1. Пусть P* — *предикат над информационной средой. Тогда имеет место свойство* {*P*}{*P*}*.*

Доказательство этой теоремы очевидно: пустой оператор не изменяет состояние информационной среды (в соответствии со своей семантикой), поэтому его предусловие сохраняет истинность и после его выполнения.

Для оператора присваивания справедлива

*Теорема 9.2. Пусть информационная среда IS состоит из переменной X и остальной части информационной среды RIS:*

 *IS* = (*X*, *RIS*).

*Тогда имеет место свойство*

 { *Q*(*F*(*X*, *RIS*), *RIS*) } *X* := *F*(*X*, *RIS*) { *Q*(*X*, *RIS*) } ,

*где F*(*X*, *RIS*)— *некоторая однозначная функция, Q* — *предикат.*

Доказательство. Пусть (*X*0, *RIS*0) — некоторое произвольное состояние информационной среды *IS*, и пусть перед выполнением оператора присваивания предикат *Q* ( *F*(*X*0, *RIS*0), *RIS*0) является истинным. Тогда после выполнения оператора присваивания будет истинен предикат *Q*(*X*, *RIS*), так как X получит значение *F*(*X*0, *RIS*0), а состояние *RIS* не изменяется данным оператором присваивания, и, следовательно, после выполнения этого оператора присваивания в  этом случае

 *Q* (*X*, *RIS*) = *Q* ( *F* (*X*0, *RIS*0), *RIS*0).

В силу произвольности выбора состояния информационной среды теорема доказана.

Примером свойства оператора присваивания может служить пример 9.1.

1. **Свойства основных конструкций структурного программирования.**

Рассмотрим теперь свойства основных конструкций структурного программирования: следования, разветвления и повторения.

Свойство следования выражает следующая

*Теорема 9.3. Пусть P, Q и R* — *предикаты над информационной средой, а S*1 *и S2* — *обобщенные операторы, обладающие соответственно свойствами*

{*P*}*S*{*Q*} *и* {*Q*}*S*2{*R*}*.*

*Тогда для составного оператора*

*S*1*; S*2

*имеет место свойство*

{*P*} *S*1; *S*2{*R*} *.*

Доказательство. Пусть для некоторого состояния информационной среды перед выполнением оператора *S*1 истинен предикат *P*. Тогда в силу свойства оператора *S*1 после его выполнения будет истинен предикат *Q*. Так как по семантике составного оператора после выполнения оператора *S*1 будет выполняться оператор *S*2, то предикат *Q* будет истинен и перед выполнением оператора *S*2. Следовательно, после выполнения оператора *S*2 в силу его свойства будет истинен предикат *R*, а так как оператор *S*2 завершает выполнение составного оператора (в соответствии с его семантикой), то предикат *R* будет истинен и после выполнения данного составного оператора, что и требовалось доказать.

Например, если имеют место свойства (9.2) и (9.3), то имеет

место и свойство

{ *n* < *m* } *n* := *n* + *k*; *n* := 3\**n* { *n* < 3 \* ( *m* + *k* ) }.

Свойство разветвления выражает следующая

*Теорема 9.4. Пусть P, Q и R* — *предикаты над информационной средой, а S*1 *и S*2— *обобщенные операторы, обладающие соответственно свойствами*

{ *P, Q* } *S*1{*R*} *и* {¬*P, Q* } *S*2{*R*}*.*

*Тогда для условного оператора*

*ЕСЛИ P ТО S1ИНАЧЕ S*2 *ВСЕ ЕСЛИ*

*имеет место свойство*

{*Q*} *ЕСЛИ P ТО S*1 *ИНАЧЕ S*2 *ВСЕ ЕСЛИ* {*R*} *.*

Доказательство. Пусть для некоторого состояния информационной среды перед выполнением условного оператора истинен предикат *Q*. Если при этом будет истинен также и предикат *P*, то выполнение условного оператора в соответствии с его семантикой сводится к выполнению оператора *S*1. В силу же свойства оператора *S*1 после его выполнения (а в этом случае — и после выполнения условного оператора) будет истинен предикат *R*. Если же перед выполнением условного оператора предикат *P* будет ложен (а *Q*, по-прежнему, истинен), то выполнение условного оператора в соответствии с его семантикой сводится к выполнению оператора *S*2. В силу же свойства оператора *S*2 после его выполнения (а в этом случае — и после выполнения условного оператора) будет истинен предикат R. Тем самым теорема полностью доказана.

Прежде чем переходить к свойству конструкции повторения следует отметить полезную для дальнейшего

*Теорему 9.5. Пусть P, Q, P*1 *и Q*1— *предикаты над информационной средой, для которых справедливы импликации*

*P*1 ⇒ *P и  Q* ⇒ *Q*1*,*

*и пусть для оператора S имеет место свойство* {*P*}*S*{*Q*}*.Тогда имеет место свойство* {*P*1}*S*{*Q*1} *.*

Эту теорему называют еще теоремой об ослаблении свойств.

Доказательство. Пусть для некоторого состояния информационной среды перед выполнением оператора *S* истинен предикат *P*1. Тогда будет истинен и предикат *P* (в силу импликации *P*1⇒ *P*). Следовательно, в силу свойства оператора *S* после его выполнения будет истинен предикат *Q*, а значит и предикат *Q*1 (в силу импликации *Q*⇒ *Q*1). Тем самым теорема доказана.

Свойство повторения выражает следующая

*Теорема 9.6. Пусть I, P, Q и R* — *предикаты над информационной средой, для которых справедливы импликации*

*P*⇒ *I и*( *I,* ¬*Q* )⇒ *R,*

*и пусть S* — *обобщенный оператор, обладающий свойством* {*I*}*S*{*I*}*.*

*Тогда для оператора цикла*

*ПОКА Q ДЕЛАТЬ S ВСЕ ПОКА*

*имеет место свойство*

{*P*} *ПОКА Q ДЕЛАТЬ S ВСЕ ПОКА* {*R*} *.*

Предикат I называют *инвариантом* оператора цикла.

Доказательство. Для доказательства этой теоремы достаточно доказать свойство

{I} ПОКА Q ДЕЛАТЬ S ВСЕ ПОКА {I,¬Q}

(по теореме 9.5 на основании имеющихся в условиях данной теоремы импликаций). Пусть для некоторого состояния информационной среды перед выполнением оператора цикла истинен предикат *I*. Если при этом предикат *Q* будет ложен, то оператор цикла будет эквивалентен пустому оператору (в соответствии с его семантикой) и в силу теоремы 9.1 после выполнения оператора цикла будет справедливо утверждение ( *I*, ¬*Q*). Если же перед выполнением оператора цикла предикат *Q* будет истинен, то оператор цикла в соответствии со своей семантикой может быть представлен в виде составного оператора

*S; ПОКА Q ДЕЛАТЬ S ВСЕ ПОКА*

В силу свойства оператора *S* после его выполнения будет истинен предикат *I*, и возникает исходная ситуация для доказательства свойства оператора цикла: предикат *I* истинен перед выполнением оператора цикла, но уже для другого (измененного) состояния информационной среды (для которого предикат *Q* может быть либо истинен либо ложен). Если выполнение оператора цикла завершается, то, применяя метод математической индукции, мы за конечное число шагов придем к ситуации, когда перед его выполнением будет справедливо утверждение ( *I*, ¬*Q*). А в этом случае, как было доказано выше, это утверждение будет справедливо и после выполнения оператора цикла. Теорема доказана.

Например, для оператора цикла из примера (9.4) имеет место свойство

{ *n* > 0, *p* = 1, *m* = 1}

*ПОКА m* <> *n ДЕЛАТЬ*

 *m* := *m* + 1; *p* := *p*\**m*

*ВСЕ ПОКА* { *p* = *n*! }.

Это следует из теоремы 9.6, так как инвариантом этого оператора цикла является предикат *p* = *m*! и справедливы импликации

( *n* > 0, *p* = 1, *m* = 1) ⇒ *p* = *m*! и  ( *p* = *m*!, *m* = *n*) ⇒ *p* = *n*!

**9.4. Завершимость выполнения программы.**

Одно из свойств программы, которое нас может интересовать, чтобы избежать возможных ошибок в ПС, является ее *завершимость,* т.е. отсутствие в ней зацикливания при тех или иных исходных данных. В рассмотренных нами структурированных программах источником зацикливания может быть только конструкция повторения. Поэтому для доказательства завершимости программы достаточно уметь доказывать завершимость оператора цикла. Для этого полезна следующая

Теорема 9.7. Пусть F — целочисленная функция, зависящая от состояния информационной среды и удовлетворяющая следующим условиям:

1. *если для данного состояния информационной среды истинен предикат Q, то ее значение положительно;*
2. *она убывает при изменении состояния информационной среды в результате выполнения оператора S.*

 *Тогда выполнение оператора цикла*

*ПОКА Q ДЕЛАТЬ S ВСЕ ПОКА*

*завершается*.

Доказательство. Пусть *IS* — состояние информационной среды перед выполнением оператора цикла и пусть *F* (*IS*) = *k*. Если предикат *Q*(*IS*) ложен, то выполнение оператора цикла завершается. Если же предикат *Q*(*IS*) истинен, то по условию теоремы *k* > 0. В этом случае будет выполняться оператор *S* один или более раз. После каждого выполнения оператора *S* по условию теоремы значение функции *F* уменьшается, а так как перед выполнением оператора *S* предикат *Q* должен быть истинен (по семантике оператора цикла), то значение функции *F* в этот момент должно быть положительно (по условию теоремы). Поэтому в силу целочисленности функции *F* оператор *S* в этом цикле не может выполняться более *k* раз. Теорема доказана.

Например, для рассмотренного выше примера оператора цикла

условиям теоремы 9.7 удовлетворяет функция *f* (*n*, *m*) = *n* − *m*. Так как перед выполнением оператора цикла *m* = 1, то тело этого цикла будет выполняться ( *n* – 1 ) раз, т.е. этот оператор цикла завершается.

**9.5. Пример доказательства свойства программы.**

На основании доказанных правил верификации программ можно доказывать свойства программ, состоящих из операторов присваивания и пустых операторов и использующих три основные композиции структурного программирования. Для этого, анализируя структуру программы и используя заданные ее пред- и постусловия, необходимо на каждом шаге анализа применять подходящее правило верификации. В случае применения композиции повторения потребуется подобрать подходящий инвариант цикла.

В качестве примера докажем свойство (9.4). Это доказательство будет состоять из следующих шагов.

(Шаг 1). *n* > 0 ⇒ ( *n* > 0, *p* — любое, *m* — любое).

(Шаг 2). Имеет место

{ *n* > 0, *p* — любое, *m* — любое} *p* := 1 { *n* > 0, *p* = 1, *m* — любое }.

—— По теореме 9.2.

(Шаг 3). Имеет место

{ *n* > 0, *p* = 1, *m* — любое } *m* := 1 { *n* > 0, *p* = 1, *m* = 1 }.

—— По теореме 9.2.

(Шаг 4). Имеет место

{ *n* > 0, *p* — любое, *m* — любое} *p* := 1; *m* := 1 { *n* > 0, *p* = 1, *m* = 1 }.

 —— По теореме 9.3 в силу результатов шагов 2 и 3.

Докажем, что предикат *p* = *m*! является инвариантом цикла, т.е. { *p* = *m*! } *m* := *m* + 1; *p* := *p*\**m* { *p* = *m*! }.

(Шаг 5). Имеет место { *p* = *m*! } *m* := *m* + 1 { *p* = ( *m* – 1 )! }.

—— По теореме 9.2, если представить предусловие в виде { *p* = ( (*m* + 1) − 1)! }.

(Шаг 6). Имеет место { *p* = (*m* − 1)! } *p* := *p*\**m* { *p* = *m*! }.

—— По теореме 9.2, если представить предусловие в виде { *p*\**m*= *m*! }.

(Шаг 7). Имеет место инвариант цикл

{ *p* = *m*! } *m* := *m* + 1; *p* := *p*\**m* { *p* = *m*! }.

—— По теореме 9.3 в силу результатов шагов 5 и 6.

(Шаг 8). Имеет место

{ *n* > 0, *p* = 1, *m* = 1} *ПОКА* *m* <> *n* *ДЕЛАТЬ*

 *m* := *m* + 1; *p* := *p*\**m*

*ВСЕ ПОКА* { *p* = *n*! }.

—— По теореме 9.6 в силу результата шага 7 и имея в виду, что ( *n* > 0, *p* = 1, *m* = 1 )⇒ *p* = *m*!; ( *p* = *m*!, *m* = *n* )⇒ *p* = *n*!.

(Шаг 9). Имеет место

{ *n* > 0, *p* — любое, *m* — любое } *p* := 1; *m* := 1;

*ПОКА m* <> *n ДЕЛАТЬ*

 *m* := *m* + 1; *p* := *p*\**m*

*ВСЕ ПОКА* { *p* = *n*! }.

—— По теореме 9.3 в силу результатов шагов 3 и 8.

(Шаг 10). Имеет место свойство (9.4) по теореме 9.5 в силу результатов шагов 1 и 9.

**Упражнения к лекции 9.**

*9.1. Что такое триада Хоора?*

*9.2. Что такое свойство программы?*

*9.3. Пусть заданы описания*

*const n= <конкретное целое значение>*;

*var k, m*: *integer;*

*x*: *array*[1*..n*] *of integer;*

*Доказать свойство программы:*

{ *n* > 0 }

*m* := *x*[1];

*k* := 1;

*ПОКА k* < *n ДЕЛАТЬ*

*k* :*= k +* 1;

*ЕСЛИ x*[*k*] *< m ТО*

*m* := *x*[*k*]

*ВСЕ ЕСЛИ*

*ВСЕ ПОКА*;

{ *n* > 0& *m <= x*[*i*] *для всех i,* 1 *<= i <= n*}

**Литература к лекции 9.**

[9.1] С.А. Абрамов. Элементы программирования. — М.: Наука, 1982. С. 85–94.

[9.2] М. Зелковец, А. Шоу, Дж. Гэннон. Принципы разработки программного обеспечения. — М.: Мир, 1982. С. 98–105.