Departamento de Informática
Universidad de Valladolid
Campus de Segovia

TOPIC 8: ALGORITHM CORRECTNESS PROOFS

ALGORITHM CORRECTNESS PROOFS

- Preliminary concepts.
- Correctness.
- Preconditions and postconditions rules
- Correctness proofs in loopless code
- Loops
- Program termination

PRELIMINARY CONCEPTS.
Some Definitions

- The algorithm correctness proofs consists of a of formal techniques for determining whether a program works correctly or not.
    - **A program works correctly:** if it satisfies a given specification.
    - **Correctness proof tecniques:** consists in an inference process. Hence, there is an inference rule for each kind of executable sentence.
    - **Formal representation:** Hoare triples.

PRELIMINARY CONCEPTS
Assertions

- An assertion is a logical sentence about the states of a system. An assertion is denoted by means of square brackets {A}.
- An state represents the current set of values associated to the most relevant parameters that define a system .
- Preconditions y postconditions:
  - **Preconditions** involve any condition about the input data.
  - **Postconditions** involve any condition about the output data.
  - Example: Program working out the square root.
    - Precondition: The input must not be a negative number.
    - Postcondition: The square root of this number.

PRELIMINARY CONCEPTS

- A program is a sequence of sentence that converts the initial state into a final one.
- The initial state represents the previous state to execution.
- The final state represents the state once the code was executed.

PRELIMINARY CONCEPTS
Formal representation, Hoare triples

- If C is a piece of code, then any assertion {P} is called precondition of C if {P} just involves the initial state. Any assertion {Q} is called postcondition if just only involves the final state,.
- That definition is represented as: {P}C{Q} and is called Hoare triple.
- Example 1: $\{y \neq 0\}$ x:=1/y {x=1/y}
- Example 2: Sometimes there is no precondition:
  - { } a:=b {a=b}
  - This one represents the most general case, it means, any input generates an output satisfying the postcondition.
  - Here, { } represents the empty assertion which can be interpreted as “true for all possible states”.

PRELIMINARY CONCEPTS. Notation

- In most of the pieces of code, the final states depends on the initial ones.

- There are two possible notations for denoting this dependency:
  - **Subscripts representation**: This representation makes a distinction between the initial and the final state by means of subscripts. The 'α' subscript is related to the initial state and 'ω' to the final one.
  - **Shadow variables representation**: the shadow variables are that do not appear in the code and that are introduced to store initial values of certain storage location. Their definition have to appear on the precondition.

    {a=A,b=B} h:=a; a:=b; b:=h {a=B, b=A}

CORRECTNESS CONCEPTS

- If {P}C{Q} is a hoare triple representing a piece of code with a precondition {P} and postcondition {Q}, then {P}C{Q} is correct if every possible initial state satisfying {P} results in a final state satisfying {Q}.

- In this regard, it is possible to distinguish between:
  - **Partial Correctness**: If C is a piece of code with precondition {P} and postcondition {Q} then {P}C{Q} is said to be partially correct if the final state of C satisfies {Q}, provided the initial state satifies {P} although there is no final state due to the fact that the program does not terminate.
  - **Total Correctness**: if {P}C{Q} is partially correct and C terminates then C is said to be totally correct.

- A loopless partially correct code is always totally correct also. Then this distinction is only important when loops and recursion are present on the code.

PRECONDITION AND POSTCONDITIONS RULES

- If {R} and {S} are two assertions then it is said that {R} is stronger than {S} if {R}⇒{S} ({S} implies {R}). If {R} is stronger than {S} then it is said that {S} is weaker than {R}.
  - **Example 1:** {i>1} is stronger than {i>0} since {i>1} implies also {i>0}. Then {i>1} ⇒{i>0}.

- If an assertion {R} is stronger than an assertion {S} then every state satisfying {R}, satisfies also S but no viceversa. The strengthening of an assertion reduces the number of states that satisfy this one.
  - **Stronger** ⇒ more selective, less general.
  - **Weaker** ⇒ less selective, more general.
  - the weakest assertion is { } since it consideres all the possible states.
  - The strongest assertion is {False} since it represents that there is no state satifying the condition.

PRECONDITION STRENGTHENING

- If a piece of code C satisfies the precondition {P} then an strengthening of this precondition will be satisfied too.
- If {P}C{Q} is correct and $P_1 \Rightarrow P$ then it is also true that $\{P_1\}C\{Q\}$ is correct. This leads to the folowing inference rule:

$$\frac{P_1 \Rightarrow P \qquad \{P\}C\{Q\}}{\{P_1\}C\{Q\}}$$

  - Example: Let´s supposse the following Hoare triple is correct: $\{y \neq 0\}\ x:=1/y\ \{x=1/y\}$, then prove that $\{y=4\}\ x:=1/y\ \{x=1/y\}$ is also correct.

$$\frac{y=4 \Rightarrow y \neq 0 \qquad \{y \neq 0\}\ x:=1/y\ \{x=1/y\}}{\{y=4\}\ x:=1/y\ \{x=1/y\}}$$

EMPTY PRECONDITION STRENGTHENING

- The empty precondition may be strengthened to yield any precondition {P}.

- Example:
  - { } a:=b {a=b} can be used to justify {P} a:=b {a=b} where {P} is any precondition.

- As a general rule:
  - It is always advatageous to formulate the weakest precondition that assures a given postcondition. In this way any stronger precondition will be automatically satisfied.
  - As well as any program should be written such that they are as versatile as possible (they cover as many initial states as possible).

POSTCONDICITION WEAKENING

- According to the postcondition weakening principle if {P}C{Q} and {Q}$\Rightarrow${Q_1} are satisfied then {P}C{Q_1} is correct. This leads the following inference rule:

$$\frac{\{P\}C\{Q\} \qquad \{Q\} \Rightarrow \{Q_1\}}{\{P\}C\{Q_1\}}$$

- Example: Prove that, if { } max:=b {max=b} is correct, then { } max:=b {max$\geq$b} is also correct.

$$\frac{\{\ \}\ max:=b\ \{max=b\} \qquad max=b \Rightarrow max \geq b}{\{\ \}\ max:=b\ \{max \geq b\}}$$

CONJUNCTION RULE

- The following rule allow us to strengthen the precondition and postcondition simultaneously.

- Definition: If C is a piece of code and $\{P_1\}C\{Q_1\}$ and $\{P_2\}C\{Q_2\}$ have been established then it could be concluded that $\{P_1 \wedge P_2\}C\{Q_1 \wedge Q_2\}$. Formally it can be expressed as follows:

$$\frac{\{P_1\}C\{Q_1\} \quad \{P_2\}C\{Q_2\}}{\{P_1 \wedge P_2\}C\{Q_1 \wedge Q_2\}}$$

- As a particular case:

$$\frac{\{\ \}C\{Q_1\} \quad \{P_2\}C\{Q_2\}}{\{P_2\}C\{Q_1 \wedge Q_2\}}$$

CONJUNCTION RULE

- Example:
  - Use the Hoare triples $\{\ \}i:=i+1\ \{i_\omega:=i_\alpha+1\}$, $\{i_\alpha>0\ \}i:=i+1\ \{i_\alpha>0\}$ to prove that:

$$\{i>0\ \}i:=i+1\ \{i>1\}$$

$$\{\ \}i:=i+1\{i_\omega:=i_\alpha+1\}$$
$$\underline{\{i_\alpha>0\ \}i:=i+1\ \{i_\alpha>0\}}$$
$$\{i_\alpha>0\ \}i:=i+1\ \{(i_\alpha>0)\wedge(i_\omega:=i_\alpha+1)\}$$

  - where: $i_\alpha:=i_\omega-1$ and since $i_\alpha>0$ then $i_\omega>1$. Hence:

$$\{i>0\ \}i:=i+1\ \{i>1\}$$

DISJUNCTION RULE

- The following rule allow us to weak the precondition and postcondition simultaneously.

- Definition: if C is a piece of code and $\{P_1\}C\{Q_1\}$ y $\{P_2\}C\{Q_2\}$ have been established then it can be concluded that $\{P_1 \vee P_2\}C\{Q_1 \vee Q_2\}$. Formally it can be expressed as follows:

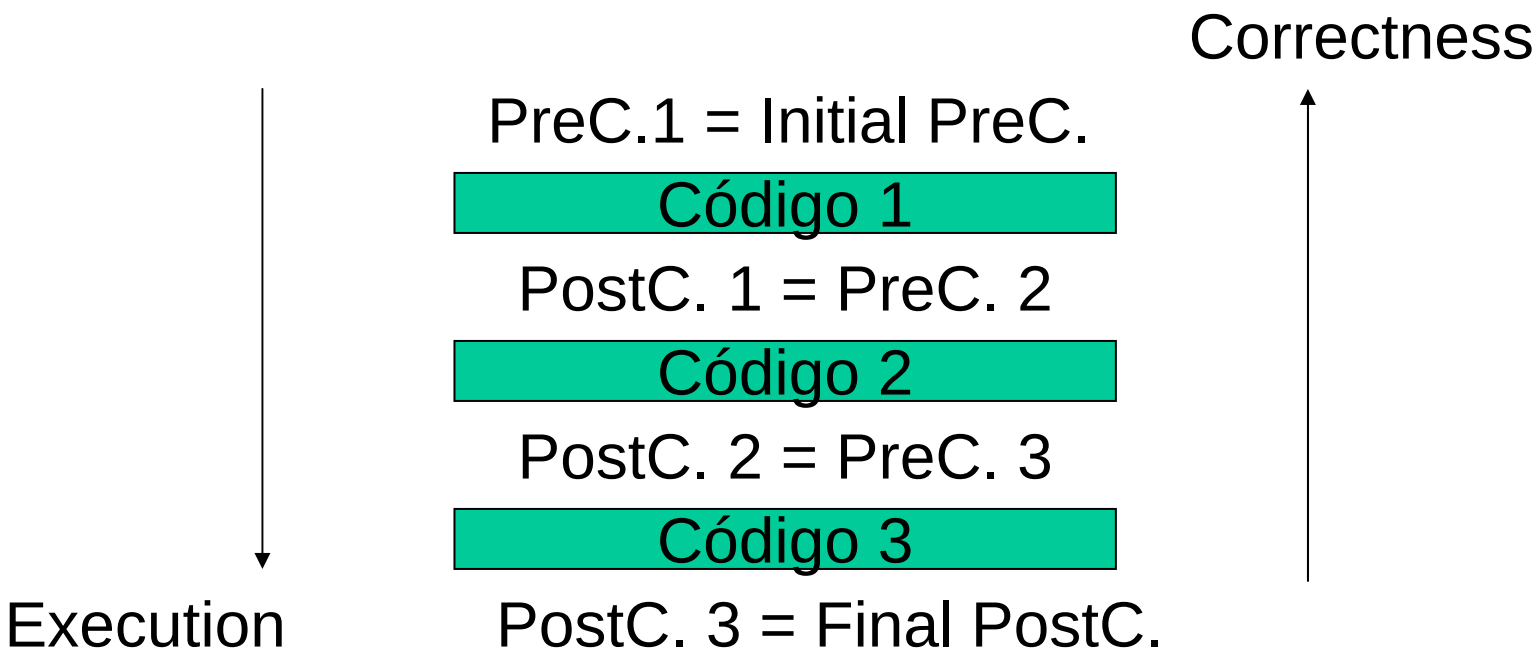
$$\frac{\{P_1\}C\{Q_1\} \quad \{P_2\}C\{Q_2\}}{\{P_1 \vee P_2\}C\{Q_1 \vee Q_2\}}$$

- As a particular case:

$$\frac{\{\ \}C\{Q_1\} \quad \{P_2\}C\{Q_2\}}{\{P_2\}C\{Q_1 \vee Q_2\}}$$

CORRECTNESS PROOFS IN LOOPLESS CODE

- **Correctness:** To prove the correctness of any piece of code it is necessary to use the postcondition as a starting point. Then, the initial precondition is derived from the last postcondition proving the code in the opposite order in which it is executed.



ASSIGNMENT STATEMENTS

- This inference rule requires that no involved variable share the same memory space with another. It means do not consider either pointers or arrays.

- The assignment statements are statements of the form V:=E, where V is a variable and E is an expression.

$$\{\ \}V:=E\{\ V=E\ _{\alpha}\}$$

- However that expression is not always correct (i.e. x:=1/y). In this case, it is necessary to strengthen the precondition.

ASSIGNEMENT RULE

- **Assignement rule:** If C is a sentence of the form V:=E with postcondition {Q}, then the precondition {P} could be found by replacing every instance of V in Q by E. If Q_E^V is the expression thus obtained, then:

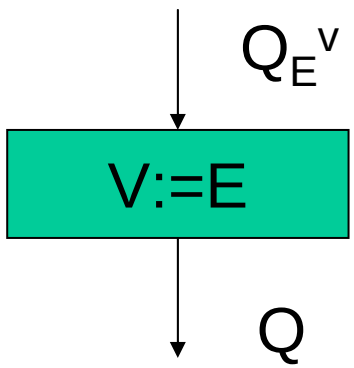
$$\{P\}V:=E\{Q\}$$
$$\{P\}=\{Q_E^V\} \Rightarrow \{ V=E_{\alpha},Q\}$$
$$\{Q_E^V\}V:=E\{Q\}$$

- Example: Find which is the precondition {P} such that {P} i:=2*i {i<6} is correct:

$$\{Q_E^V\}=\{i_{\omega}=2*i_{\alpha},\ i_{\omega}<6\}\Rightarrow\{2*i<6\} \Rightarrow\{i<3\}$$

ASSIGMENT RULE

$$\{Q_E^V\}V:=E\{Q\}$$



ASSIGMENT RULE
Examples

- **Example 1**: Find the precondition {P} such that {P} j:=i+1 {j>0} is correct:

  $\{Q_E^V\}=\{j_\omega=i_\alpha+1,\ j_\omega>0\}\Rightarrow\{i_\alpha+1>0\}$

- **Example 2**: Find the precondition {P} such that {P} y:=x^2 {y>1} is correct:

  $\{Q_E^V\}=\{y_\omega=x_\alpha * x_\alpha,\ y_\omega>1\}\Rightarrow\{x_\alpha^2>1\}$

- **Example 3**: Find the prostcondition {Q} such that {x>2} x:=x^2 {Q} is correct

  $\{Q\}\Rightarrow\{x_\alpha>2,\ x_\omega=x_\alpha^2\}\Rightarrow\{x_\omega>4\}$

- **Example 4**: Find the precondition {P} such that {P} x:=1/x {x≥0} is correct:

  $\{Q_E^V\}\Rightarrow\{x_\omega=1/x_\alpha,\ x_\omega\geq 0\}\Rightarrow\{x_\alpha>0\}$

CONCATENATION OF CODE

- Concatenation means that the pieces of code are executed secuencially, in such a way that the final state of the first piece of code becomes the initial state of the second piece of code and so on.

- **Concatenation rule:** Let C_1 and C_2 be two pieces of code, and let $C_1;C_2$ be their concatenation. If $\{P\}C_1\{R\}$ y $\{R\}C_2\{Q\}$ are both correct then we can conclude that:

$$\{P\}C_1\{R\}$$
$$\underline{\{R\}C_2\{Q\}}$$
$$\{P\}C_1;C_2\{Q\}$$

CONCATENATION OF CODE
Example 1

- **Example:** Prove that the following piece of code is correct:

$$\{\ \}c:=a+b;\ c:=c/2\ \{c=(a+b)/2\}$$

$$\{\ P\}c:=c/2\ \{c=(a+b)/2\}$$

$$P=\{c_{\omega}=c/2, c_{\omega}=(a+b)/2\}\Rightarrow\{c/2=(a+b)/2\}$$

$$\{\ c/2=(a+b)/2\}c:=c/2\ \{c=(a+b)/2\}$$

$$\{P\}\ c:=a+b\ \{c/2=(a+b)/2\}$$

$$P=\{c_{\omega}=a+b, c_{\omega}/2=(a+b)/2\}\Rightarrow\{(a+b)/2=(a+b)/2\}\Rightarrow\{\ \}$$

$$\{\ \}c:=a+b\ \{c/2=(a+b)/2\}$$

EXAMPLE 2

- **Example:** Prove that the following piece of code is correct:

$$\{\ \}s:=1;\ s:=s+r;\ s=s+r*r\ \{s=1+r+r^2\}$$

$$\{\ P\}s:=s+r*r\ \{s=1+r+r^2\}$$

$$P=\{s_\omega=s+r^2, s_\omega=1+r+r^2\}\Rightarrow\{s+r^2=1+r+r^2\}$$

$$\{\ s+r^2=1+r+r^2\}s:=s+r*r\ \{s=1+r+r^2\}$$

$$\{P\}\ s:=s+r\ \{s=1+r\}$$

$$P=\{s_\omega=s+r, s_\omega=1+r\}\Rightarrow\{s+r=1+r\}$$

$$\{s+r=1+r\ \}s:=s+r\ \{s=1+r\}$$

$$\{P\}\ s:=1\ \{s=1\}$$

$$P=\{s_\omega=1, s_\omega=1\}\Rightarrow\{1=1\}\Rightarrow\{\ \}$$

$$\{\ \}s:=1\ \{s=1\}$$

EXAMPLE 3

- **Example:** Prove that the following piece of code is correct: {a=A,b=B}h:=a; a:=b; b=h {a=B,b=A}

$$\{ P\}b:=h \{a=B,b=A\}$$

$$P=\{b=h, a=B, b=A\}\Rightarrow\{h=A, a=B\}$$

$$\{ h=A, a=B\}b:=h \{a=B, b=A\}$$

$$\{P\}\ a:=b \{h=A, a=B\}$$

$$P=\{a=b, h=A, a=B\}\Rightarrow\{h=A, b=B\}$$

$$\{h=A, b=B\}a:=b \{h=A, a=B\}$$

$$\{P\}\ h:=a \{h=A, b=B\}$$

$$P=\{h=a, h=A,b=B\}\Rightarrow\{a=A, b=B\}$$

$$\{a=A, b=B\}h:=a \{h=A, b=B\}$$

THE IF STATEMENT
with Else clause

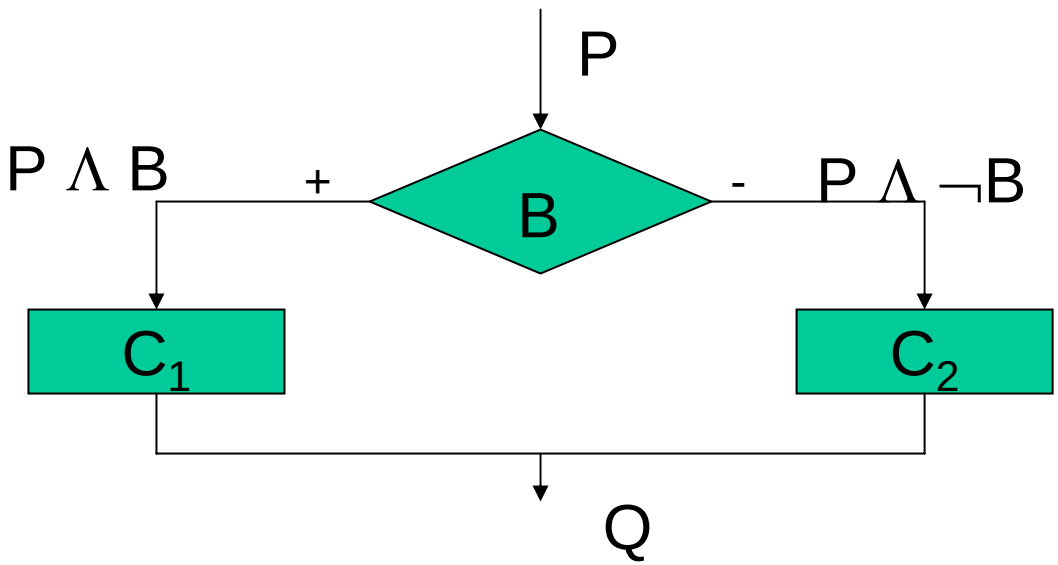
- If C_1 y C_2 are two piece of code of a program and B is a logical expression (condition) then the statement '**if B then C_1 else C_2**' may be interpreted as "if B is true then C_1 is executed otherwise C_2 is executed".
- The correctness proof of this statement involves:
  - If the initial state satifies B, as well as {P}, then C_1 is executed and hence it supposes to prove that $\{P \wedge B\}\ C_1\ \{Q\}$ is correct.
  - If the initial state satifies {not B, as well as {P}, then C_2 is executed and hence it supposes to prove that $\{P \wedge \neg B\}\ C_2\ \{Q\}$ is correct.

INFERENCE RULE FOR AN IF STATEMENT WITH ELSE CLAUSE

$$\{P \wedge B\} C_1 \{Q\}$$

$$\underline{\{P \wedge \neg B\} C_2 \{Q\}}$$

$$\{P\} \text{if B then } C_1 \text{ else } C_2 \{Q\}$$



THE IF STATEMENT with Else clause

- **Example:** Prove that the following piece of code is correct:

  $\{\ \}$if a>b then m:=a else m:=b$\{(m \geq a) \wedge (m \geq b)\}$

- $\{\ a>b\}$m:=a $\{(m \geq a) \wedge (m \geq b)\}$

  $P=\{m=a, (m \geq a) \wedge (m \geq b)\} \Rightarrow \{a \geq a, a \geq b\}$

  $\{a=a\} \Rightarrow \{a \geq a\}, \{a=a, a \geq b\} \Rightarrow \{a \geq b\}$

  $\{\ a>b\} \Rightarrow \{a \geq b\}$ {Fortalecimiento PreC.}

  $\{\ a>b\}$m:=a $\{(m \geq a) \wedge (m \geq b)\}$

- $\{\ \neg(a>b)\}$m:=b $\{(m \geq a) \wedge (m \geq b)\}$

  $P=\{m=b,(m \geq a) \wedge (m \geq b)\} \Rightarrow \{b \geq a, b \geq b\}$

  $\{b=b\} \Rightarrow \{b \geq b\}, \{b \geq a, b=b\} \Rightarrow \{b \geq a\}$

  $\{\ \neg(a>b)\} \Rightarrow \{b \geq a\}$

  $\{\ b \geq a\}$m:=b $\{(m \geq a) \wedge (m \geq b)\}$

THE IF STATEMENT without Else clause

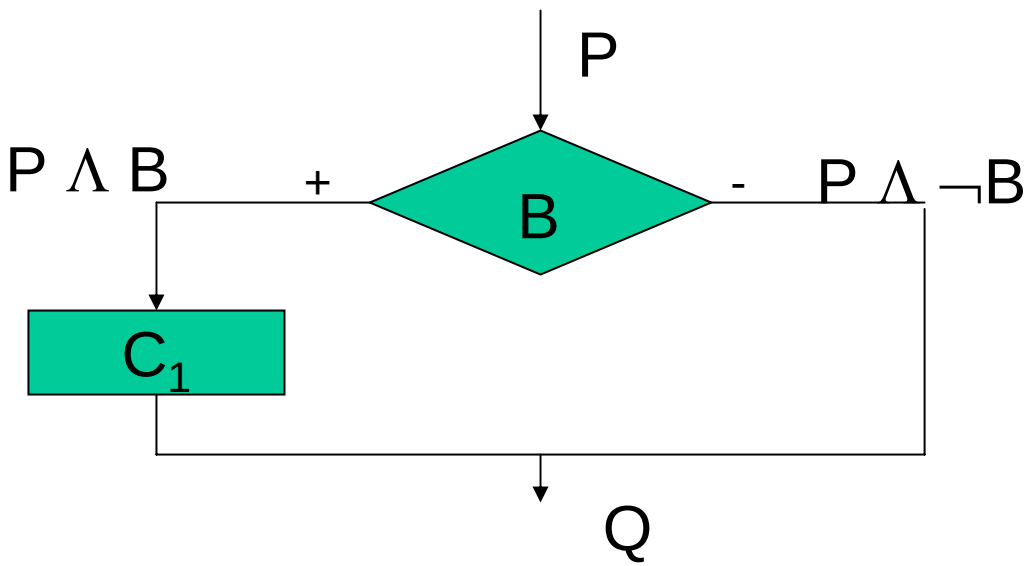
- If C_1 is a piece of code and B is some condition then the statement '**if B then C_1** ' may be interpreted as: "if B is true then C_1 is executed".
- The correctness proof of this statement involves:
  - If the initial state satifies B, as well as {P}, then C_1 is executed and therefore it supposes to prove that $\{P \wedge B\}\ C_1\ \{Q\}$ is correct.
  - If the initial state satifies {not B}, as well as {P}, then it supposes to prove that $\{P \wedge \neg B\} \Rightarrow \{Q\}$ is correct.

INFERENCE RULE FOR AN IF STATEMENT WITH ELSE CLAUSE

$$\{P \wedge B\} C_1 \{Q\}$$

$$\underline{\{P \wedge \neg B\} \Rightarrow \{Q\}}$$

$$\{P\} \text{if B then } C_1 \{Q\}$$



THE IF STATEMENT without Else clause

- **Example:** Prove that the following piece of code is correct:
  - { }if max<a then max:=a {(max≥a)}

- {¬(max<a)}⇒{(max≥a)}

- { max<a} max:=a {(max≥a)}

$$P=\{max=a,\ (max\geq a)\}\Rightarrow\{a\geq a\}$$

$$\{a\geq a\}\Rightarrow\{\ \},$$

Since {max<a} ⇒ { }

CORRECTNESS PROOFS IN CODES WITH LOOPS

- A loop invariant is any assertion that is, at the same time, a precondition and a postcondition of a piece of code.
- A loop variant is an expression that measures the progress made toward satisfying the exit condition.
- The iterative structures in Pascal that will be considered for the correctness proofs are:
  - 'while B do C', where B is some logical condition and C is a piece of code inside the loop.
  - 'repeat C until B', where B is some logical condition and C is a piece of code inside the loop.
- In this first step only the partial correctness of these structures will be studied.
- Termination proofs will be considered later.

INVARIANT OF A PIECE OF CODE

– **Example:** Prove that $\{r=2^i\}$ is an invariant of the following piece of code: i:=i+1; r:=r*2.

$$\{ P\}r:=r*2 \{r=2^i\}$$

$$P=\{r=r*2, r=2^i\}\Rightarrow\{r*2=2^i\}\Rightarrow\{r=2^{i-1}\}$$

$$\{r=2^{i-1}\}\ r:=r*2\ \{r=2^i\}$$

$$\{P\}\ i:=i+1\ \{r=2^{i-1}\}$$

$$P=\{i=i+1, r=2^{i-1}\}\Rightarrow\{r=2^{i+1-1}\}\Rightarrow\{r=2^i\}$$

$$\{r=2^i\}i:=i+1\ \{r=2^{i-1}\}$$

SOME HINTS TO FIND THE INVARIANT.

- The loop invariant is a precursor of the postcondition, which means that it must somehow be similar to the postcondition.
- The loop invariant contains all variables that change from iteration to iteration.
- The loop invariant is only a formalization of the programmer goals.

EXAMPLE

- Find the loop invariant of this piece of code:

```
sum:=0;
j:=0;
while  j<>n do begin
          sum:=sum+a;
          j:=j+1
end;
```

EXAMPLE I

- If the loop invariant is **I={sum=j*a}**

{ P} j:=j+1; {sum=j*a}

P={j=j+1,sum=j*a}⇒{sum=(j+1)*a}

{sum=(j+1)*a} j:=j+1 {sum=j*a}

{P} sum:=sum+a; {sum=(j+1)*a}

P={sum=sum+a, sum=(j+1)*a}⇒{sum+a=(j+1)*a}

{sum=j*a} sum:=sum+a; {sum=(j+1)*a}

Therefore {I}C{I}

{P} j:=0 {sum=j*a}

P={j=0,sum=j*a}⇒{sum=0}

{sum=0} j:=0 {sum=j*a}

{P} sum:=0 {sum=0}

P={sum=0,sum=0}⇒{ }

{ } sum:=0 {sum=0}

{(¬ B∧I)} ⇒ {(sum=j*a) ∧(j=n)} ⇒{sum=n*a}

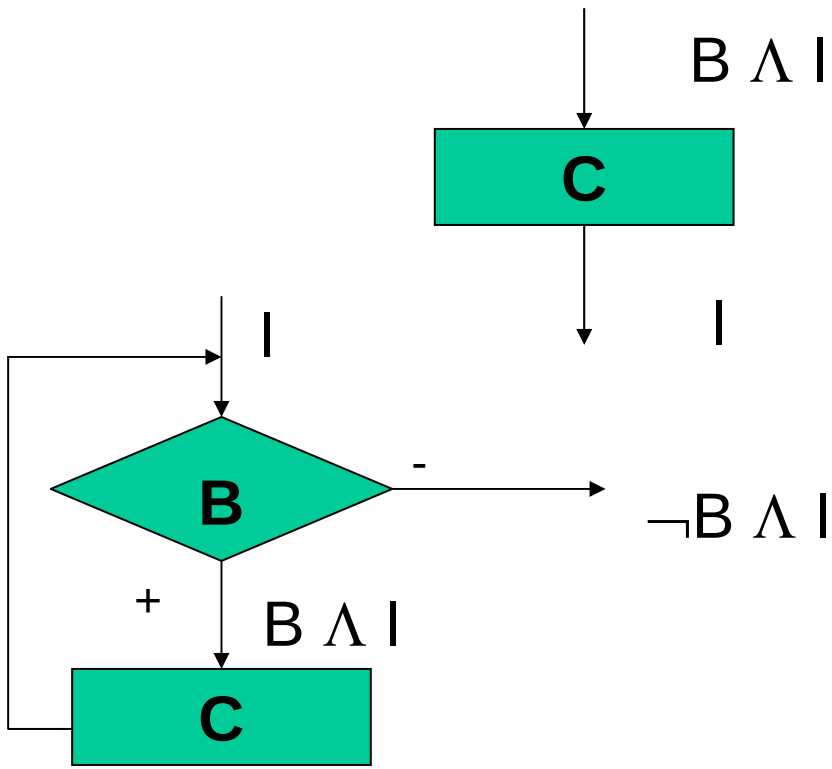
INFERENCE RULE FOR A WHILE LOOP.

- If C is a piece of code such that $\{B \wedge I\}\ C\ \{I\}$ is correct then we can conclude that:

$$\frac{\{B \wedge I\}\ C\ \{I\}}{\{I\}\ \text{while } B \text{ do } C\ \{(\neg B \wedge I)\}}$$

INFERENCE RULE FOR A WHILE LOOP

$$\frac{\{B \wedge I\}\ C\ \{I\}}{\{I\}\ \text{while } B \text{ do } C\ \{(\neg B \wedge I)\}}$$



EXAMPLE I CORRECTNESS PROOF OF A WHILE LOOP

- Prove the correctness of the following piece of code.

```
i:=1;
sum:=0;
while not ( i=n+1) do begin
        sum:=sum+i*i;
        i:=i+1
end:
```

- This piece of code works out the following summatory $\Sigma_{j=1}^{n} i^2$.

EJEMPLO I

if $\{(sum=\Sigma_{j=1}^{n} j^2) \wedge (i=n+1)\}$ is the postcondition and

I= $\{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}$ the invariant, then:

$$\{ P\}\ i:=i+1; \{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}$$

$$P=\{i=i+1, sum= \Sigma_{j=1}^{i-1} j^2, i \leq n+1\} \Rightarrow \{sum= \Sigma_{j=1}^{i} j^2, i+1 \leq n+1\}$$

$$\{ i+1 \leq n+1\} \Rightarrow \{i \leq n\} \Rightarrow \{i<n+1\}$$

$$\mathbf{\{(sum=\Sigma_{j=1}^{i} j^2) \wedge (i <n+1)\}\ i:=i+1\ \{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}}$$

$$\{P\}\ sum:=sum+i^2; \{(sum=\Sigma_{j=1}^{i} j^2) \wedge (i < n+1)\}$$

$$P=\{sum=sum+i^2, sum= \Sigma_{j=1}^{i} j^2, i < n+1\} \Rightarrow \{sum+i^2= \Sigma_{j=1}^{i} j^2, i<n+1\}$$

$$\{sum+i^2= i^2+\Sigma_{j=1}^{i-1} j^2, i<n+1\} \Rightarrow \{sum= \Sigma_{j=1}^{i-1} j^2, i< n+1\}$$

$$\mathbf{\{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i<n+1)\}\ sum:=sum+i^2;\ \{(sum=\Sigma_{j=1}^{i} j^2) \wedge (i < n+1)\}}$$

Con lo que queda demostrado que $\{B \wedge I\}C\{I\}$

$$\{P\}\ sum:=0\ \{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}$$

$$P=\{sum=0, sum= \Sigma_{j=1}^{i-1} j^2, i \leq n+1\} \Rightarrow \{0= \Sigma_{j=1}^{i-1} j^2, i \leq n+1\}$$

$$\mathbf{\{(0= \Sigma_{j=1}^{i-1} j^2) \wedge\ (i \leq n+1)\}\ sum:=0\ \{(sum= \Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}}$$

$$\{P\}\ i:=1\ \{(0= \Sigma_{j=1}^{i-1} j^2) \wedge\ (i \leq n+1)\}$$

$$P=\{i=1, 0= \Sigma_{j=1}^{i-1} j^2, i<n+1\} \Rightarrow \{0= \Sigma_{j=1}^{0} j^2\ , 1 \leq n+1\} \Rightarrow \{0 \leq n\}$$

$$\mathbf{\{n \geq 0\ \}\ i:=1\ \{(0= \Sigma_{j=1}^{i-1} j^2) \wedge\ (i<n+1)\}}$$

$$\{(\neg\ B \wedge I)\} \Rightarrow \{(i=n+1) \wedge ((sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1))\} \Rightarrow$$

$$\{(sum=\Sigma_{j=1}^{n} j^2\ ) \wedge (i=n+1)\}$$

EJEMPLO I

If $\{(sum=\Sigma_{j=1}^{n} j^2) \wedge (i=n+1)\}$ is the postcondition

And I= $\{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}$ the invariant, then:

$$\mathbf{\{n \geq 0\}}\ i:=1\ \{(0= \Sigma_{j=1}^{i-1} j^2) \wedge (i<n+1)\}$$

$$\{(0= \Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}\ sum:=0\ \mathbf{\{(sum= \Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}}$$

while not (i=n+1) do

$$\mathbf{\{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i<n+1)\}}\ sum:=sum+i^2;\ \{(sum=\Sigma_{j=1}^{i} j^2) \wedge (i <n+1)\}$$

$$\{(sum=\Sigma_{j=1}^{i} j^2) \wedge (i <n+1)\}\ \ i:=i+1\ \ \mathbf{\{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}}$$

$$\frac{\{B \wedge I\}\ C\ \{I\}}{\{I\}\ \text{while } B \text{ do } C\ \{(\neg B \wedge I)\}}$$

$$\{(\neg B \wedge I)\} \Rightarrow \{(i=n+1) \wedge ((sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1))\} \Rightarrow$$

$$\{(sum=\Sigma_{j=1}^{n} j^2) \wedge (i=n+1)\}$$

EXAMPLE II, CORRECTNESS PROOF OF A WHILE LOOP

- Prove that the following piece of code is correct:

```
f:=1;
t:=0;
i:=0;
while  ( i ≤ n) do begin
        t:=t+f;
        f:=f*r;
        i:=i+1
end;
```

- It is supposed that it works out the following expression:

$$t=\Sigma_{j=0}^{n} r^j \text{ for } i>n$$

EXAMPLE II

If $\{(t=\Sigma_{j=0}^{n} r^j) \wedge (i>n)\}$ is the postcondition and

I= $\{(t=\Sigma_{j=0}^{i-1} r^j) \wedge (f=r^i) \wedge (i \leq n+1)\}$ the invariant

$$\{P\}\ i:=i+1;\ \{(t=\Sigma_{j=0}^{i-1} r^j) \wedge (f=r^i) \wedge (i \leq n+1)\}$$

$$P=\{i=i+1, t=\Sigma_{j=1}^{i-1} r^j,\ f=r^i,\ i \leq n+1\} \Rightarrow \{t=\Sigma_{j=1}^{i} r^j,\ f=r^{i+1},\ i+1 \leq n+1\}$$

$$\{i+1 \leq n+1\} \Rightarrow \{i \leq n\} \Rightarrow \{i<n+1\}$$

$$\mathbf{\{(t=\Sigma_{j=1}^{i} r^j) \wedge (f=r^{i+1}) \wedge (i<n+1)\}\ i:=i+1\ \{(t=\Sigma_{j=1}^{i-1} r^j) \wedge (f=r^i) \wedge (i \leq n+1)\}}$$

$$\{P\}\ f:=f*r;\ \{(t=\Sigma_{j=1}^{i} r^j) \wedge (f=r^{i+1}) \wedge (i<n+1)\}$$

$$P=\{f=f*r,\ t=\Sigma_{j=1}^{i} r^j,\ f=r^{i+1},\ i<n+1\} \Rightarrow \{t=\Sigma_{j=1}^{i} r^j,\ f*r=r^{i+1},\ i<n+1\} \Rightarrow$$

$$\Rightarrow \{t=\Sigma_{j=1}^{i} r^j,\ f=r^i,\ i<n+1\}$$

$$\mathbf{\{(t=\Sigma_{j=1}^{i} r^j) \wedge (f=r^i) \wedge (i<n+1)\}\ f:=f*r;\ \{(t=\Sigma_{j=1}^{i} r^j) \wedge (f=r^{i+1}) \wedge (i<n+1)\}}$$

$$\{P\}\ t:=t+f;\ \{(t=\Sigma_{j=1}^{i} r^j) \wedge (f=r^i) \wedge (i<n+1)\}$$

$$P=\{t=t+f,\ t=\Sigma_{j=1}^{i} r^j,\ f=r^i,\ i<n+1\} \Rightarrow \{t+f=\Sigma_{j=1}^{i} r^j,\ f=r^i,\ i<n+1\} \Rightarrow$$

$$\Rightarrow \{t+r^i=r^i+\Sigma_{j=1}^{i-1} r^j,\ f=r^i,\ i<n+1\} \Rightarrow \{t=\Sigma_{j=1}^{i-1} r^j,\ f=r^i,\ i<n+1\}$$

$$\mathbf{\{(t=\Sigma_{j=1}^{i-1} r^j) \wedge (f=r^i) \wedge (i<n+1)\}\ f:=f*r;\ \{(t=\Sigma_{j=1}^{i} r^j) \wedge (f=r^i) \wedge (i<n+1)\}}$$

Then $\{B \wedge I\}C\{I\}$ is proved

EXAMPLE II

$$\{P\}\ i:=0;\ \{(t=\Sigma_{j=1}^{i-1} r^j)\wedge (f=r^i)\wedge(i \leq n+1)\}$$

$$P=\{i=0,\ t=\Sigma_{j=1}^{i-1} r^j,\ f=r^i,\ i\leq n+1\}\Rightarrow \{t=\Sigma_{j=1}^{-1} r^j,\ f=r^0,\ 0 \leq n+1\} \Rightarrow$$

$$\Rightarrow\{t=0,\ f=1,\ 0 \leq n+1\}$$

$$\mathbf{\{(t=0\ )\wedge(f=1)\wedge(0<n+1)\}\ i:=0;\ \{(t=\Sigma_{j=1}^{i-1} r^j)\wedge\ (f=r^i)\wedge(i \leq n+1)\}}$$

$$\{P\}\ t:=0;\ \{(t=0)\wedge\ (f=1)\wedge(-1\leq n)\}$$

$$P=\{t=0,\ t=0,\ f=1,\ -1 \leq n\}\Rightarrow \{0=0,\ f=1,\ -1 \leq n\}$$

$$\mathbf{\{(f=1)\wedge(-1\leq n)\}\ t:=0;\ \{(t=0)\wedge\ (f=1)\wedge(-1 \leq n)\}}$$

$$\{P\}\ f:=1;\ \ (f=1)\wedge(-1\leq n)\}$$

$$P=\{f=1,\ f=1,\ -1 \leq n\}\Rightarrow \{1=1,\ -1 \leq n\}$$

$$\mathbf{\{(-1\leq n)\}\ f:=1;\ \{(f=1)\wedge(-1 \leq n)\}}$$

$$\{(i>n)\} \Rightarrow\{i=n+1\}$$

$$\{(\neg\ B\wedge I)\} \Rightarrow \{(i=n+1)\wedge((t=\Sigma_{j=0}^{i-1} r^j)\wedge(\ f=r^i)\wedge(i\leq n+1))\} \Rightarrow$$

$$\{((t=\Sigma_{j=0}^{n} r^j)\wedge(\ f=r^{n+1})\wedge(i=n+1))\}$$

EXAMPLE II

If $\{(t=\Sigma_{j=0}^{n} r^j) \wedge (i>n)\}$ is the postcondition and

I= $\{(t=\Sigma_{j=0}^{i-1} r^j) \wedge (f=r^i) \wedge (i \leq n+1)\}$ the invariant

$\mathbf{\{(-1 \leq n)\}}$ f:=1; $\{(f=1) \wedge (-1 \leq n)\}$

$\{(f=1) \wedge (-1 \leq n)\}$ t:=0; $\{(t=0) \wedge (f=1) \wedge (-1 \leq n)\}$

$\{(t=0) \wedge (f=1) \wedge (0<n+1)\}$ i:=0; $\mathbf{\{(t=\Sigma_{j=1}^{i-1} r^j) \wedge (f=r^i) \wedge (i \leq n+1)\}}$

while i $\leq$n do

$\mathbf{\{(t=\Sigma_{j=1}^{i-1} r^j) \wedge (f=r^i) \wedge (i <n+1)\}}$ f:=f*r; $\{(t=\Sigma_{j=1}^{i} r^j) \wedge (f=r^i) \wedge (i <n+1)\}$

$\{(t=\Sigma_{j=1}^{i} r^j) \wedge (f=r^i) \wedge (i <n+1)\}$ f:=f*r; $\{(t=\Sigma_{j=1}^{i} r^j) \wedge (f=r^{i+1}) \wedge (i <n+1)\}$

$\{(t=\Sigma_{j=1}^{i} r^j) \wedge (f=r^{i+1}) \wedge (i <n+1)\}$ i:=i+1 $\mathbf{\{(t=\Sigma_{j=1}^{i-1} r^j) \wedge (f=r^i) \wedge (i \leq n+1)\}}$

$$\frac{\{B \wedge I\}\ C\ \{I\}}{\{I\}\ \text{while } B \text{ do } C\ \{(\neg B \wedge I)\}}$$

$\{(i>n)\} \Rightarrow \{i=n+1\}$

$\{(\neg B \wedge I)\} \Rightarrow \{(i=n+1) \wedge ((t=\Sigma_{j=0}^{i-1} r^j) \wedge (f=r^i) \wedge (i \leq n+1))\} \Rightarrow$

$\{((t=\Sigma_{j=0}^{n} r^j) \wedge (f=r^{n+1}) \wedge (i=n+1))\}$

INFERENCE RULE FOR A REPEAT LOOP

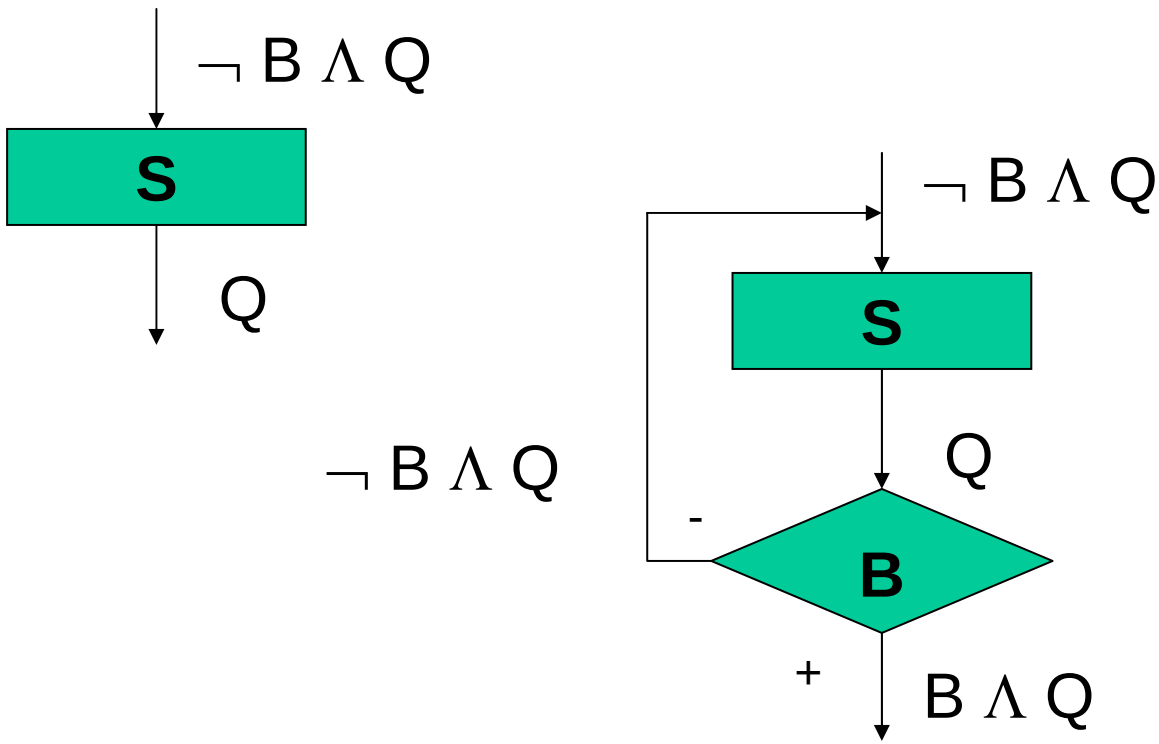
- If C is a piece of code such that {I} C {Q} is correct, where I={$\neg$ B $\wedge$ Q}, then we can coclude that:

$$\frac{\{I\}\ C\ \{Q\}}{\{I\}\ \text{repeat}\ \ C\ \ \text{until}\ B\ \{(B \wedge Q)\}}$$

INFERENCE RULE FOR A REPEAT LOOP

$$\underline{\{I\}\ C\ \{Q\}}$$

{I} repeat C until B {(B∧Q)}



EXAMPLE I, CORRECTNESS PROOF OF A REPEAT LOOP

- Prove the correctness of the following piece of code:

```
z:=0;
u:=x;
repeat
        z:=z+y;
        u:=u-1
until  u=0
```

- This piece of code works out the product of two integer numbers.

EXAMPLE1 I

- If {(z=x*y) $\wedge$ (u=0)} is the postcondition and {Q}= {(z+u*y=x*y) $\wedge$ (u$\geq$0)} the invariant, then:

{ P} u:=u-1;{(z+u*y=x*y) $\wedge$ (u$\geq$0)}

P={u=u-1, z+u*y=x*y, u $\geq$ 0}$\Rightarrow${z+(u-1)*y=x*y, u-1$\geq$0}

{ u-1 $\geq$ 0} $\Rightarrow$ {u $\geq$ 1} $\Rightarrow$ {u >0}

{(z+(u-1)*y=x*y) $\wedge$ (u > 0) } u:=u-1 {(z+u*y=x*y) $\wedge$ (u $\geq$ 0)}

{P} z:=z+y; {(z+(u-1)*y=x*y) $\wedge$ (u >0) }

P={z=z+y, z+(u-1)*y=x*y, u>0} $\Rightarrow$ {z+y+(u-1)*y=x*y, u >0}

{z+y+(u-1)*y=x*y} $\Rightarrow$ {z+y+u*y-y=x*y} $\Rightarrow$ {z+u*y=x*y}

{(z+u*y=x*y) $\wedge$ (u > 0)} z:=z+y; {(z+(u-1)*y=x*y) $\wedge$ (u > 0)}

Con lo que queda demostrado que {$\neg$ B $\wedge$ Q} C {Q}

{P} u:=x; {(z+u*y=x*y) $\wedge$ (u > 0)}

P={u=x, z+u*y=x*y, u>0}$\Rightarrow${z+x*y=x*y, u>0} $\Rightarrow$ {z=0, x>0}

{z=0, x>0} u:=x; {(z+u*y=x*y) $\wedge$ (u > 0)}

{P} z:=0 {(z=0) $\wedge$ (x>0)}

P={z=0, z=0, x>0}$\Rightarrow${0=0 , x>0}$\Rightarrow${x>0}

{x>0 } z:=0 {(z=0) $\wedge$ (x>0)}

{(B$\wedge$Q)} $\Rightarrow$ {(u=0) $\wedge$ ((z+u*y=x*y) $\wedge$ (u $\geq$0))} $\Rightarrow$

{(z=x*y) $\wedge$ (u=0)}

EXAMPLE I

If $\{(z=x*y) \wedge (u=0)\}$ is the postcondition and

I= $=\{\neg B \wedge Q\}= \{(z+u*y=x*y) \wedge (u>0)\}$ the invariant, then:

$Q= \{(z+u*y=x*y) \wedge (u \geq 0)\}$

{x>0} z:=0 $\{(z=0) \wedge (x>0)\}$

$\{z=0, x>0\}$ u:=x; $\{(z+u*y=x*y) \wedge (u > 0)\}$

repeat

$\{(z+u*y=x*y) \wedge (u > 0)\}$ z:=z+y; $\{(z+(u-1)*y=x*y) \wedge (u > 0)\}$

$\{(z+(u-1)*y=x*y) \wedge (u > 0)\}$ u:=u-1; **$\{(z+u*y=x*y) \wedge (u \geq 0)\}$**

until u=0

$\underline{\{ I\}C\{Q\}}$

{ I} repeat C until B $\{(B \wedge Q)\}$

$\{(B \wedge Q)\} \Rightarrow \{(u=0) \wedge ((z+u*y=x*y) \wedge (u \geq 0))\} \Rightarrow$

$\{(z=x*y) \wedge (u=0)\}$

DOCUMENTATION

```
*****************************************
* This function works out the product of two      *
* Integer numbers.                                *
* parameters: x>0                                 *
* Result: z                                       *
*****************************************
```

BEGIN {Prec: x>0; Dec: u-1}
    z:=0; u:=x;
    REPEAT {z+u*y=x*y Λ u > 0}
    z:=z+y; u:= u-1
    UNTIL (u=0);
END { Post: z=x*y Λ u=0}

TOTAL CORRECTNESS

- A program is **partially correct** if it can be proved that, if the program terminates, it terminates satisfying the postcondition. If, in addition, we can prove that the program does indeed terminate then this one is **totally correct**.
- For establishing the termination condition we have to study how change the variables involved into the loop variant.
- As soon as the entry condition of the loop fails, after a finite number of iteration, then the program does terminate.

Example

- Prove that the following piece of code terminates for n≥0.

```
i:=0;
f:=1;
while i<>n do
        begin
            i:=i+1;
            f:=f*r;
        end;
```

- Let us suppose that “n” is initialized to 0.
- The only variable in the entry condition that changes inside the loop is “i”. Hence the loop variant is “i”.
  - If n=0 then the entry condition is not satisfied and the loop is not executed. **The program does terminate.**
  - If n>0 then as soon i=n, after “n” iterations increasing in one unit the value of i, the entry condition fails and the loop terminates. **The program does terminate.**

CORRECTNESS OF A RECURSIVE ALGORITHM

- PARTIAL CORRECTNESS:
  - Prove the correctness of the case base by means of an **inductive hypothesis**.
  - Prove the correctness of the recurrent cases by means of the **inductive step**.
- TOTAL CORRECTNESS:
  - Prove that, In each recursive invocation, some parameters are changing toward the case base condition.