Axiomatic Semantics
The guarded command language

Semantics
- A programming language specification consists of a syntactic description and a semantic description.
- Syntactic description: symbols we can use in a language
- Semantic Description: what phrases in a programming language mean.
- Semantics may be given as
  - Denotational
  - Axiomatic
  - Operational
- Here we concentrate on axiomatic descriptions: the meaning is defined by a logical calculus called program logic which provides a tool for the derivation of programs and assertions of the form \( \{ Q \} p \{ R \} \)

Example:
- Read in 2 integers and output their product divided by their sum. You are guaranteed that there are 2 integer values in the input stream.
- Precondition = \{ Input stream contains two integer values \}
- Postcondition = \{ Product / Sum is output \}
- \( \{ Q \} p \{ R \} \) states that a program \( p \) once started in a state satisfying \( \{ Q \} \) will lead to a situation characterised by \( \{ R \} \)
- \( \{ Q \} \) may also be written as the weakest precondition of \( p \) that achieve postcondition \( R \) i.e. \( wp(p, R) \)

\[ wp(S, R) \] represents the set of all states such that execution of \( S \) beginning in any one of them is guaranteed to terminate in a finite amount of time satisfying \( R \).
- Examples:
  - \( wp(i = i + 1, i <= 1) \)
  - S: if \( x >= y \) then \( z = x \) else \( z = y \), R: \( z = max(x, y) \)
    calculate \( wp(S, R) \)
  - Let \( S \) be as above and \( R: z = y \), calculate \( wp(S, R) \)
    - S: if \( x >= y \) then \( z = x \) else \( z = y \), R: \( z = y - 1 \)
    calculate \( wp(S, R) \)
  - Let \( S \) be as above, \( R: z = y + 1 \)

Some properties of \( wp \)
- Law of excluded miracle: \( wp(S, F) = F \)
- Distributivity of conjunction: \( wp(S, Q) \land wp(S, R) = wp(S, Q \land R) \)
- Law of monotonicity: if \( Q \Rightarrow R \) then \( wp(S, Q) \Rightarrow wp(S, R) \)
- Distributivity of disjunction: \( wp(S, Q) \lor wp(S, R) \Rightarrow wp(S, Q \lor R) \)
- Non-deterministic:
  - Execution of a command is non-deterministic if it need not always be exactly the same each time it is begun in the same state
    - e.g. \( \{ x = 4 \} x := 14 \| x := x + 1 \) \? \)

- Command \( S \) is usually designed to establish the truth of a postcondition \( R \). We may not be interested in \( wp(S, R) \). If we can find a stronger precondition \( Q \) that represents a subset of the set \( wp(S, R) \) and can show \( Q \Rightarrow wp(S, R) \) then we are content with \( Q \) as the postcondition.
- When we write \( \{ Q \} p \{ R \} \) we denote Total Correctness
- \( Q \{ p \} R \) denotes partial correctness.
### Exercises

- Determine
  - \( \text{wp}(i := i + 1, i > 0) \)
  - \( \text{wp}(i := i + 2, j := j - 2, i + j = 0) \)
  - \( \text{wp}(x := x^j; i := i - 1, x^* j = c) \)
  - \( \text{wp}(a[i] = 1, a[i] = a[i]) \)
  - \( \text{wp}(a[i] = i, a[i] = i) \)

### Skip & Abort

- Skip
  - Execution of the skip command does nothing.
  - It is equivalent to the empty command;
  - It is the identity transformer
  - \( \text{wp}(\text{skip}, R) = R \)

- Abort
  - \( \text{wp}(\text{abort}, R) = \text{False} \)
  - Abort should never be executed as it may only be executed in a state satisfying \( \text{False} \).

### Sequential Composition

- A way of composing larger programs from smaller segments
- If \( S_1 \) and \( S_2 \) are commands then \( S_1; S_2 \) is a new command
- \( \text{wp}(S_1; S_2, R) = \text{wp}(S_1, \text{wp}(S_2, R)) \)

### Assignment

- \( x := e \)
- \( x \) is a simple variable, \( e \) is an expression and the types of \( x \) and \( e \) are the same
- \( \text{wp}(x := e, R) = \text{domain}(e) \text{ and } R_x^e \)
- Domain\( (e) \) is a predicate that describes the set of all states in which \( e \) may be evaluated i.e. is well defined.
- Usually we write: \( \text{wp}(x := e, R) = R_x^e \)

### Examples:

- \( \text{wp}(x := 5, x = 5) \)
- \( \text{wp}(x := 5, x != 5) \)
- \( \text{wp}(x := x+1, x < 10) \)
- \( \text{wp}(x := x^2, x^2 = 10) \)
- \( \text{wp}(x := a/b, p(x)) \)
- \( \text{wp}(x := b[i], x = b[i]) \) for \( b \), an array with indexes 0..100

### Multiple Assignment

- Multiple assignment has the form
  - \( x_1, x_2, x_3, \ldots, x_n := e_1, e_2, e_3, \ldots, e_n \)
  - where \( x_i \) are distinct simple variables and \( e_i \) are expressions.

- Definition:
  - \( \text{wp}(x_1, x_2, \ldots, x_n := e_1, e_2, e_3, \ldots, e_n, R) \)
  - \( = \text{domain}(e_1, e_2, e_3, \ldots, e_n) \text{ and } R_{x_1, x_2, x_3, \ldots, x_n} \)
  - Examples:
    - \( x, y := y, x; \)
    - \( x, y, z := y, z, x; \)
    - \( \text{wp}(x, y := z, x, y < 0 \land x^* x'^* = c) \)

- Execution of an expression may change only the variables indicated and evaluation of an expression may change no variables.
- This prohibits functions with side effects and allows us to consider expressions as conventional mathematical entities i.e. we can use associativity, commutativity of addition etc.

- Example: Swapping two variables:
  - \( \text{wp}(x := x; y := y; x = X \land y = Y) \)
The if statement

If \( B_1 \rightarrow S_1 \)

[ ] \( B_2 \rightarrow S_2 \)

... [ ] \( B_n \rightarrow S_n \)

fi

- Each \( B_i \rightarrow S_i \) is a guarded command and each \( S_i \) may be any command e.g. skip, about, sequential composition etc.
- If any guard \( B_i \) is not well defined in the state in which execution begins, abortion may occur. This is because nothing is assumed by the order of evaluation of the guards.
- At least one guard must be true to avoid abortion.
- If at least one guard \( B_i \) is true, then 1 guarded command \( B_i \rightarrow S_i \) is chosen and \( S_i \) is executed.

\[
\text{W}p(\text{If}, R) = \text{domain}(BB) \land BB \land (B_1 \Rightarrow wp(S_1, R)) \land ... \land (B_n \Rightarrow wp(S_n, R))
\]

where \( BB = B_1 \lor B_2 \lor ... \lor B_n \)

\[
\text{wp}(\text{If}, R) = (\exists i : 1 \leq i \leq n : B_i) \land (\forall i : 1 \leq i \leq n : B_i \Rightarrow wp(S_i, R))
\]

Example: \( A \equiv (\text{if } x \geq 0 \rightarrow z := x) \) \[ x \leq 0 \rightarrow z := -x \]

\( wp(A, z = \text{abs}(x)) = \text{True} \)

Exercises:

- Complete and Simplify:
  1. \( wp(S, a \geq 0 \land b > 0) \) where
  \( S = \) if \( a > b \rightarrow z := a - b \)
  [ ] \( b > a \rightarrow b := b - a \)
  fi

  2. \( wp(S, x \leq y) \) where
  \( S = \) if \( x > y \rightarrow x, y := y, x \)
  [ ] \( x < y \rightarrow \text{skip} \)
  fi

The Iterative Command

- Do \( B \rightarrow S \) od
  - where \( B \rightarrow S \) is a guarded command. This is equivalent to a while loop.
  - Do \( x>0 \rightarrow x := x - 1 \) od
  = while \( x>0 \) \{ \( x := x - 1 \) \}

We can generalize in the guarded command language to:

Do \( B_1 \rightarrow S_1 \) [ ] \( B_2 \rightarrow S_2 \)
...
[ ] \( B_n \rightarrow S_n \)
odo where \( n>0 \), and \( B_i \rightarrow S_i \) is a guarded command.

Note: Non Determinism is allowed.

- Let \( BB = B_1 \lor B_2 \lor ... \lor B_n \)
- \( H_i(R) = \neg BB \land R \)
  - Represents the set of states in which execution of DO terminates in 0 iterations with \( R \) true, as the guards are initially false
- \( wp(DO,R) = \exists k : 0 < k : H_k(R) \)
  - Represents the set of states in which execution of DO terminates in a bounded number of iterations with \( R \) true.

Example: What does the following calculate? How can we prove it?
\[
\text{Is} = 1, b[0] = 0;
\text{do}
\ i := 11 \rightarrow i := i + 1, s = b[i] \text{ od}
\text{for}
\ R : s = \sum k : 0 < k < 11 : b[k] \]

Exam: The state

If \( B_1 \rightarrow S_1 \)

[ ] \( B_2 \rightarrow S_2 \)

... [ ] \( B_n \rightarrow S_n \)

fi

Example:

\[ T = \{ x \geq 0 \} \lor \{ x < 0 \} \]
if \( x > 0 \)
\{ \text{abs}(x) \}
\{ z = \text{abs}(x) \}
[ ] \( x > 0 \)
\{ \text{abs}(x) \}
\{ z = \text{abs}(x) \}
fi

\[ \{ x \geq 0 \land x \leq 0 \land z = \text{abs}(x) \} \lor \{ x \leq 0 \land z = \text{abs}(x) \} \]
= \{ z = \text{abs}(x) \}
**Invariant (P)**: Predicate that is true throughout the program

**Guard Bi, BB:**
- True on entry into the loop
- May be true or false at the exit point of the loop => re-evaluate guard
- The guard is always false after the loop terminates

**Postcondition (R):** The postcondition should imply the Invariant and the negation of the guard i.e. \( P \land \neg BB \Rightarrow R \)

**Precondition (Q):** Should imply the Invariant with initialisations.

---

**Loop Template**

\[ \{ Q \} \Rightarrow \{ P \} \]

Do BB

\[ \{ P \land BB \} \]

"Loop Body"

\[ \{ P \} \]

Od

\[ \{ P \land \neg BB \} \Rightarrow \{ R \} \]

---

**Program Verification**

- Given a precondition, a postcondition and some code verify that the code when executed in a state satisfying the given precondition achieves the given postcondition.

\[ \{ Q \} : \{ \text{Array b has values} \} \]

\[ i,s := 1,b[0] \]

Do \( i < N \)

\[ i,s := i+1,s+b[i]; \]

Od

\[ \{ R \} : \{ s = \sum k: 0 \leq k < 11 : b[k] \} \]

---

**Loop Termination**

- To show that a loop terminates we introduce an integer function, \( t \) where \( t \) is a function of the program variables i.e. an upper bound on the number of iterations still to be performed.
- \( t \) is called the **variant function** and it is a measure of the amount of work yet to be completed by the loop.
- Each iteration of the loop decreases \( t \) by at least one
- As long as execution of the loop has not terminated then \( t \) is bounded below by 0. Hence the loop must terminate.
- In our last example \( t: 11-i \)

---

**Checklist for loops**

- Show that \( P \) is true before the execution of a loop begins
- Show that \( P \land \neg BB \Rightarrow R \) i.e. when the loop terminates the desired result is true.
- Show that \( \{ P \land Bi \} \Rightarrow \{ P \} \) for \( 1 \leq i < n \) i.e. execution of each guarded command terminates with \( P \) true so that \( P \) is an invariant of the loop.
- Show that \( P \land BB \Rightarrow (t > 0) \) so that the bound function i.e. “the amount of work yet to be done” is bounded from below as long as the loop has not terminated.
- Show that \( \{ P \land Bi \} \Rightarrow \{ t < n \} \) for \( 1 \leq i < n \) so that each loop iteration is guaranteed to decrease the bound function. In general \( t \) can only provide an upper bound on the number of iterations to be performed. Hence, it is called the bound function or the variant function.