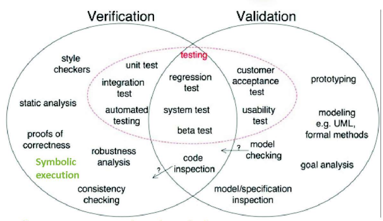
Software Systems Verification and Validation

Lecture 6: Correctness

Assoc. Prof. Andreea Vescan
Babeş-Bolyai University
Cluj-Napoca
2018-2019

Sales paradigm - SSVV

- Motivate the STUDENT - what you will learn!



- http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
  - **EVOZON** Presentation, topic: **Test automation**
  - **When:** Friday, April 12, 2019, hours 14:00-16:00;
  - **Where:** Room A2 (FSEGA Building)

Program verification methods - Correctness

- Lecture 1 - Verification and Validation
  - Verification
    - reviews products to ensure their quality → correctness
    - static and dynamic analysis techniques
  - A **correct program** is one that does exactly what it is intended to do, no more and no less.
  - A formally correct program is one whose correctness can be proved mathematically.
    - This requires a language for specifying precisely what the program is intended to do.
    - Specification languages are based in mathematical logic.
  - Until recently, correctness has been an academic exercise. – Now it is a key element of critical software systems.
- **Program verification - correctness**
  1. proof-based, computer-assisted, program-verification approach, mainly used for programs which we expect to terminate and produce a result
  2. model-based, automatic, property-verification approach, mainly used for concurrent, reactive systems (originally used in a post-development stage) - model checking (Lecture 9, Lecture 11)
  3. Developing correct algorithms from specification (Carroll Morgan, "Programming from Specification)

     Correctness-by-Construction.

     Originally intended as a mere means of programming algorithms that are correct by construction - -Dijkstra (1968), Hoare (1971),

     the approach found its way into commercial development processes of complex systems - Hall (2002), Hall and Chapman (2002)

     2012, The Correctness-by-Construction Approach to Programming, Authors: **Kourie**, Derrick G., **Watson**, Bruce W.

     2015, Experience with correctness-by-construction, B.W. Watson a, D.G. Kourie b, L. Cleophas b,*

     2016, Correctness-by-Construction and Post-hoc Verification: Friends or Foes?, Maurice H. ter Beek1(B) , Reiner H¨ahnle2, and Ina Schaefer3
- **Correctness Tools**
  - Theorem provers (PVS), Modeling languages (UML and OCL), Specification languages (JML), Programming language support (Eiffel, Java, Spark/Ada), Specification Methodology (Design by contract)
- **Methods for prooving program correctness**
  - Floyd's Method - Inductive assertions
  - Hoare - Semantics of Hoare triples
  - Dijkstra's Language- Guarded commands, Nondeterminacy and Formal Derivation of Programs

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
  - **EVOZON** Presentation, topic: **Test automation**
  - **When:** Friday, April 12, 2019, hours 14:00-16:00;
  - **Where:** Room A2 (FSEGA Building)

Floyd's Method - Inductive assertions [Flo67]

- **Aplicability**
  - Partial correctness of the program
  - Termination of the program
  - Total correctness = Partial correctness + Termination of the program
- **Uses**
  - The condition satisfied by the initial values of the program.
  - The condition to be satisfied by the output of the program.
  - Source code of the program.
- **Method**:
  - Cut the loops
  - Find an appropriate set of inductive assertions.
  - Construct the verification/termination conditions.
- **Theorem**: If all verification conditions are true, then the program is partially correct, i.e., whenever it terminates the result is correct.
- **Remark**. The method is useful when it is combined with termination.

Robert W Floyd
(June 8, 1936 - September 25, 2001)

Floyd's Method - Inductive assertions [Flo67]

- Partial correctness - steps

Robert W Floyd
(June 8, 1936 - September 25, 2001)

- Cutting points are chosen inside the algorithm
  1. 1 point at the beginning of the algorithm, 1 point at the end;
  2. At least 1 point for each *loop* statement
- For each cutting point an assertion (invariant predicate) is chosen.
  1. Entry point - $\varphi(X)$;
  2. Ending point - $\psi(X,Z)$.
- Construction of the verification conditions
  1. Path from i to j - α;
  2. P_i and P_j are assertions in i and j;
  3. $R_\alpha(X,Y)$ - predicate that gives the condition for path α;
  4. $r_\alpha(X,Y)$ - function that gives the transformations of the variables Y from path α;
  5. $\forall X \forall Y (P_i(X,Y) \land R_\alpha(X,Y) \rightarrow P_j(X, r_\alpha(X,Y)))$.
- Theorem: If all the verification conditions are true then P is partial correct.

Floyd's Method - Inductive assertions [Flo67]

- Partial correctness - example

Robert W Floyd
(June 8, 1936 - September 25, 2001)

- Algorithm for $z = x^y$

$z := 1;\ u := x;\ v := y;$ A: $\varphi(X) ::= (v > 0 \land (y \geq 0))$
While ($v > 0$) execute B: $\eta(X,Y) ::= z * u^v = x^y$
If (v is even)
then $u := u * u;\ v := v/2;$
else $v := v - 1;\ z := z * u;$
endIf
endWhile
endAlg; C: $\psi(X,Z) ::= z = x^y$

Floyd's Method - Inductive assertions [Flo67]

- Termination - steps

Robert W Floyd
(June 8, 1936 - September 25, 2001)

- Cut the loops and find "good" inductive assertions.
- Choose a well-formed set M (i.e., an ordered set without infinite strictly decreasing sequences)
- To demonstrate that some termination conditions hold: passing from one cutting point to another the values of some functions in the well-ordered set decrease.
- In point i a function is chosen $u_i : D_X \times D_Y \rightarrow M$ and the termination condition on α is: $\forall X \forall Y (\varphi(X) \land R_\alpha(X,Y) \rightarrow (u_i(X,Y) > u_j(X, r_\alpha(X,Y))))$.
- **Remark**. If partial correctness was demonstrated then the termination condition can be: $\forall X \forall Y (P_i(X) \land R_\alpha(X,Y) \rightarrow (u_i(X,Y) > u_j(X, r_\alpha(X,Y))))$.
- Theorem: If all the termination conditions hold then the program P terminates.

Floyd's Method - Inductive assertions [Flo67]

- Termination - example

Robert W Floyd
(June 8, 1936 - September 25, 2001)

- Algorithm for $z = x^y$

```
z := 1;  u := x;  v := y;
While (v > 0) execute
If (v is even)
then u := u * u;  v := v/2;
else v := v - 1;  z := z * u;
endIf
endWhile
endAlg;
```

A: $\varphi(X) ::= (v > 0 \land (y \geq 0))$

B: $\eta(X, Y) ::= z * u^v = x^y$

C: $\psi(X, Z) ::= z = x^y$

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
  - **EVOZON** Presentation, topic: **Test automation**
  - **When:** Friday, April 12, 2019, hours 14:00-16:00;
  - **Where:** Room A2 (FSEGA Building)

- The Grand Verification Challenge Hoare 2003
- Develop a compiler which verifies that the program is correct

Hoare triples [Hoa69]

Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

- The meaning of a statement is described by a triple
  - $\{\varphi\}\ P\ \{\psi\}$, where φ is called the precondition and ψ is called the postcondition.

"when started in a state satisfying P, any terminating execution of S ends in a state satisfying Q "

- If P does not terminate, we make no guarantees.
  - Partial correctness
    - $\models_{par} \{\varphi\}P\{\psi\}$
    - only if P actually terminates.
  - Total correctness
    - $\models_{tot} \{\varphi\}P\{\psi\}$
    - the program P is guaranteed to terminate.

Hoare triples [Hoa69]

- Partial correctness

Rules

- Assignment
- Sequencing
- Conditional
- Loop

Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

Hoare triples [Hoa69]

- Partial correctness

Assignment

General Form: for any expression E

- $\{P\}\ X := E\{Q\}\ provided[P \Rightarrow \langle X \leftarrow E\rangle\,(Q)]$
    - Consider the triple $\{P\}X := Y + 2\{Q\}$
        - Given predicate Q, for what predicate P does this hold?
        - for any P such that $[P \Rightarrow \langle X \leftarrow Y + 2\rangle\,(Q)]$
    - Examples
        - $\{P_0\}\ X := Y + 2\ \{X \leq Y + 2\}$
          $P_0 \equiv true$
        - $\{P_1\}\ X := Y + 2\ \{X < 0\}$
          $P_1 \equiv (Y + 2 < 0)$
        - $\{P_2\}\ X := Y + 2\ \{Y < 0\}$
          $P_2 \equiv (Y < 0)$
        - $\{P_3\}\ X := X + 2\ \{X \text{ is even}\}$
          $P_3 \equiv (X \text{ is even})$

Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

Hoare triples [Hoa69]

- Partial correctness

Sequencing

- We can conclude
  $\{P\}\ S; T\{Q\}$
  if we can find a predicate R such that
  $\{P\}\ S\{R\}\ and\ \{R\}T\{Q\}$

  Examples
    - $\{P_0\}\ X := 2 * X;\ X := X + 1\{X > 0\}$
      $P_0 \equiv (2 * X + 1 > 0)]$
    - $\{P_1\}\ X := Y;\ Y := 3\ \{X + Y < 5\}$
      $\{P_1 \equiv (Y + 3 < 5)]$

Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

Hoare triples [Hoa69]

- Partial correctness

Conditional

Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

- We can conclude
  $\{P\}$ *IF* (C) *THEN* S *ELSE* T *END*$\{Q\}$
  provided we can show
  $\{P \wedge C\}\ S\{Q\}$ *and* $\{P \wedge \neg C\} T\{Q\}$
  - Examples
    - $\{?\}\ \{((x > y) \Rightarrow Q_0) \wedge ((x \leq y) \Rightarrow Q_1)\}$
      IF $(x > y)$ *THEN* $\quad Q_0 : \{(m|x - y) \wedge (m|y)\}$
      $x := x - y$
      ELSE $Q_1 : \{(m|x) \wedge (m|y - x)\}$
      $y := y - x$
      END
      $Q : \{(m|x) \wedge (m|y)\}$
    - So our final proof obligations are
      $[(x > y) \Rightarrow (m|x - y) \wedge (m|y)$ *and*
      $[(x \leq y) \Rightarrow (m|x) \wedge (m|y - x)]$

Hoare triples [Hoa69]

- Partial correctness

Loop

Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

- How can we conclude
  $\{P\}$ *WHILE* (G) *DO* S *END* $\{Q\}$
  At the end of the loop (assuming it terminates), we know $\neg G$
  But in general we dont know how often S is executed...
- Suppose we have a predicate J that is preserved by S
  $\{J\}S\{J\}$ such a J is called a loop invariant
  Then, at the end of the loop, we can conclude
  $J \wedge \neg G$
  To establish the postcondition, we need J such that
  $[J \wedge \neg G \Rightarrow Q]$
- We can conclude
  $\{P\}$ *WHILE* (G) *DO* S *END* $\{Q\}$
  provided we can find a loop invariant J such that

$[P \Rightarrow J]$	J holds at loop entry
$[J \wedge \neg G \Rightarrow Q]$	J establishes Q at loop exit
$\{G \wedge J\}S\{J\}$	J is preserved by each iteration

- Exponentiation using multiplication
  - $\{(A > 0) \wedge (B \geq 0)\}$ S $\{R = A^B\}$

$\{(A > 0) \wedge (B \geq 0)\}$
$R :=?; b := 0$ R:=1
WHILE $(b \neq B)$ DO $J : R = A^b$
$R :=?; R := R * A;$
$b := b + 1$
END
$\{R = A^B\}$

Hoare triples [Hoa69]

Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

- The meaning of a statement is described by a triple
  - $\{\varphi\}\ P\ \{\psi\}$, where φ is called the precondition and ψ is called the postcondition.

"when started in a state satisfying P, any terminating execution of S ends in a state satisfying Q "

- If P does not terminate, we make no guarantees.

- Partial correctness
  - $\models_{par} \{\varphi\}P\{\psi\}$
  - only if P actually terminates.
- Total correctness
  - $\models_{tot} \{\varphi\}P\{\psi\}$
  - the program P is guaranteed to terminate.

- The "total correctness" interpretation also requires termination
  "when started in a state satisfying P, any execution of S must terminate in a state satisfying Q "

Hoare triples [Hoa69]

- Termination

Rules

- Assignment
- Sequencing
- Conditional
- Loop

Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

- Assignment
  $\{P\}\ X := E\ \{Q\}$ provided $[P \Rightarrow (X \leftarrow E)(Q)]$
- Sequencing
  $\{P\}\ S; T\{Q\}$ provided
  $\{P\}\ S\ \{R\}$ and $\{R\}T\ \{Q\}$ for some R
- Conditional
  $\{P\}$ IF (G) THEN S ELSE T END $\{Q\}$ provided
  $\{P \wedge G\}\ S\ \{Q\}$ and $\{P \wedge \neg G\}\ T\{Q\}$
- Note: Same as the rules for partial correctness!

- Total correctness rule for loops
- Consider
  $\{P\}$ WHILE (G) DO S END $\{Q\}$
- How do we show that the loop terminates?
- One method
  find an integer expression V such that
  the value of V is nonnegative (that is $V \geq 0$), and
  the value of V (strictly) decreases in every iteration that is,
  $\{V = K\}\ S\ \{V < K\}$
- Such an expression is called a "loop variant"

fppt.com

Hoare triples [Hoa69]

- Termination

Charles Antony Richard Hoare
(11 January 1934, Colombo, Sri Lanka)

Exponentiation using multiplication

- $\{(A > 0) \wedge (B \geq 0)\}$ S $\{R = A^B\}$
- Recall loop invariant $J : R = A^b\ \wedge (B \geq b);$
  $\{(A > 0) \wedge (B \geq 0)\}$
  $R := 1; b := 0$
  WHILE $(b \neq B)$ DO $J:\ R = A^b \wedge (B \geq b);$
  $R := R * A;$
  $b := b + 1$
  END
  $\{R = A^B\}$

fppt.com

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
  - **EVOZON** Presentation, topic: **Test automation**
  - **When:** Friday, April 12, 2019, hours 14:00-16:00;
  - **Where:** Room A2 (FSEGA Building)

fppt.com

Edsger Wybe Dijkstra [Dij75]

- Guarded command

Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

- "guarded command" - a statement list prefixed by a boolean expression: only when this boolean expression is initially true, is the statement list eligible for execution
- $<$ *guarded command* $>::=<$ *guard* $>\rightarrow<$ *guarded list* $>$
- $<$ *guard* $>::=<$ *boolean expression* $>$
- $<$ *guarded list* $>::=<$ *statement* $>\{;<$ *statement* $>\}$
- $<$ *guarded command set* $>::=$
  $<$ *guarded command* $>\{\Box<$ *guarded command* $>\}$
- $<$ *alternative construct* $>::=$ **if** $<$ *guarded command set* $>$ **fi**
- $<$ *repetitive construct* $>::=$ **do** $<$ *guarded command set* $>$ **do**
- $<$ *statement* $>::=<$ *alternative construct* $>$ |
  $<$ *repetitive construct* $>$ | *"other statements"*

Edsger Wybe Dijkstra [Hoa69]

- Nondeterminacy

Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

- Example 1
  if$x \geq y \rightarrow m := x$
  $\Box y \geq x \rightarrow m := y$
  fi
- Example 2 - compute k s.t. for fixed value n and fixed function *f(i) (defined for* $0 \leq i < n$*), k will eventually satisfy* $0 \leq k < n$ *and* $(\forall i : 0 \leq i < n : \ f(k) \geq f(i))$.

  $k := 0;\ j := 1;$
  do $j \neq n \rightarrow$ **if** $f(j) \leq f(k) \rightarrow j := j + 1$
  $\Box f(j) \geq f(k) \rightarrow k := j;\ j := j + 1$
  fi
  od

Edsger Wybe Dijkstra [Hoa69]

- Weakest pre-conditions

Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

- Hoare - introduced sufficient pre-conditions such that the mechanism will not produce the wrong result but may fail to terminate.
- Dijkstra - introduced necessary and sufficient pre-conditions such that the mechanism are guaranteed to produce the right result.
  = weakest pre-conditions
- wp(S, R), where S denotes a statement list, R some condition on the state of the system.
- wp - called a " predicate transformer" - because it associates a pre-condition to any post-condition R.

Edsger Wybe Dijkstra [Hoa69]

- Properties of wp

Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

1. Law of the Excluded Miracle
   For any S, for all states: $wp(S, F) = F$
2. For any S and any two post-conditions, such that for all states $P \Rightarrow Q$, for all states:
   $wp(S, P) \Rightarrow wp(S, Q)$
3. For any S and any two post-conditions P and Q, for all states:
   $wp(S, P)$ **and** $wp(S, Q) = wp(S, P$ **and** $Q)$
4. For any deterministic S and any post-conditions P and Q, for all states:
   $(wp(S,\ P)$ **or** $wp(S,\ Q)) \Rightarrow wp(S, P$ **or** $Q)$

Edsger Wybe Dijkstra [Hoa69]

Assignment and concatenation operator

Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

- Assignment
  The semantics of $x := E$ are given by:
  $wp(\text{"}x := E\text{"}, R) = R^x_E$, R^x_E -denotes a copy of the predicate defining R in which each occurrence of the variable x is replaced by E.
- Concatenation operator ;
  The semantics of the ; operator are given by:
  $wp(\text{"}S1\ ; S2\text{"}, R) = wp(S1, wp(S2, R))$.

Edsger Wybe Dijkstra [Hoa69]

The Alternative Construct

Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

- Let IF denote: $\mathbf{if} B_1 \rightarrow SL_1 \square ... \square B_n \rightarrow SL_n \mathbf{fi}$
  Let BB denote: $(\exists i :\ 1 \le i \le n :\ B_i)$, then, by definition
  $wp(IF, R) = (BB\ and\ (\forall i :\ 1 \le i \le n :\ B_i \Rightarrow wp(SL_i, R)))$.
- Theorem 1
  From $(\forall i : 1 \le i \le n :\ (Q\ \mathbf{and}\ B_i) \Rightarrow wp(SL_i, R)$ for all states we can conclude that $(Q\ \mathbf{and}\ BB) \Rightarrow wp(IF, R)$ holds for all states.)
- Let t denote some integer function, and $wdec(S, t)$
- Theorem 2
  From $(\forall i : 1 \le i \le n : (Q\ \mathbf{and}\ B_i) \Rightarrow wdec(SL_i, t))$ for all states we can conclude that $(Q\ \mathbf{and}\ BB) \Rightarrow wdec(IF, t)$ hold for all states.
- By definition,
  $wdec(S, t) = (tmin(X) \le t(X) - 1) = (tmin(X) < t(X))$.

Edsger Wybe Dijkstra [Hoa69]

The Alternative Construct - example

- The formal requirements for performing $m := max(x, y)$ is:
  $R: \ (m = x \textbf{ or } m = y) \textbf{ and } m \geq x \textbf{ and } m \geq y.$
- Assignment $m := x$ for $m = x$?
  $wp(``m := x", R) = (x = x \textbf{ or } x = y) \textbf{ and } x \geq x \textbf{ and } x \geq y = x \geq y$
- Theorem 1: $\textbf{if} x \geq y \rightarrow m := x \textbf{fi}$
- But $B \neq T$, so we weakening BB means looking for alternatives which might introduce new guards.
- Alternative: $"m := y"$ that introduces the new guard
  $wp(``m" := y, R) = y \geq x$
  $\textbf{if} x \geq y \rightarrow m := x$
  $\Box y \geq x \rightarrow m := y$
  fi

Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

Edsger Wybe Dijkstra [Hoa69]

- The Repetitive Construct

- Let DO denote: $\textbf{do} B_1 \rightarrow SL_1 \Box ... \Box B_n \rightarrow SL_n \textbf{do}$
  Let $H_0 = (R \textbf{ and non } BB)$
  and for $k > 0, H_k(R) = (wp(IF, H_{k-1}(R))) \textbf{ or } H_0(R)$
  then, by definition: $wp(DO, R) = (\exists k : k \geq 0 : H_k(R))$.
- Theorem 3
  If we have all the states
  $(P \textbf{ and } BB) \Rightarrow (wp(IF, P) \textbf{ and } wdec(IF, t) \textbf{ and } t \geq 0)$ we can conclude that we have for all states $P \Rightarrow wp(DO, P \textbf{ and non } BB)$
- T is the condition satisfied by all states, and $wp(S, T)$ is the weakest pre-condition guaranteeing proper termination of S.
- Theorem 4
  From $(P \textbf{ and } BB) \Rightarrow wp(IF, P)$ for all states, we can conclude that we have for all states
  $(P \textbf{ and } wp(DO, T) \Rightarrow wp(DO, P \textbf{ and non } BB))$

Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

Edsger Wybe Dijkstra [Hoa69]

- The Repetitive Construct - example

- The greatest common divisor: $x = gcd(X, Y)$
- Choose an invariant relation and variant function.
  establish the relation P to be kept invariant
  do "decrease t as long as possible under variance of P" **od**
- invariant relation (established by $x := X; y := Y$):
  $P : gcd(X, Y) = gcd(x, y) \textbf{ and } x > 0 \textbf{ and } y > 0$
- $(P \textbf{ and } B) \Rightarrow wp(``x, y : E1, E2", P))$
  $= (gcd(X, Y) = gcd(E1, E2) \textit{ and } E1 > 0 \textbf{ and } E2 > 0)$.
  - $gcd(X, Y) = gcd(E1, E2)$ is implied by P
  - invariant for $(x, y) : wp(``x := x - y, P) = (gcd(X, Y) = gcd(x - y, y) \textbf{ and } x - y > 0 \textbf{ and } y > 0)$, and guard $x > y$
  - decrease of the variant function $t = x + y$
    $wp(``x := x - y", t \leq t_0) = (x \leq t_0)$
    $tmin = x, wdec("x := x - y", t) = (x < x + y) = y > 0$

Edsger Wybe Dijkstra
(May 11, 1930 - August 6, 2002)

- x:=X; y:=Y
  $\textbf{do } x > y \rightarrow x := x - y \textbf{ od}$
- But P **and** BB - are not allowed to conclude $x = gcd(X, Y)$
  the alternative $y := y - x$ requires a guard $y > x$
- x:=X; y:=Y
  do $x > y \rightarrow$ x:=x-y
  $\Box y > x \rightarrow y := y - x$
  od

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
  - **EVOZON** Presentation, topic: **Test automation**
  - **When:** Friday, April 12, 2019, hours 14:00-16:00;
  - **Where:** Room A2 (FSEGA Building)

Developing correct programs from specification[Mor98]

- Refinement
  - Input data: X $\qquad \varphi(X)$
    Output data: Z $\qquad \psi(X, Z)$
  - Abstract program
    $Z : [\varphi, \psi]$
  - Refinement
    $P_1 \prec P_2 \prec \ldots \prec P_{n-1} \prec P_n$
  - Rules of refinement
    - Assignment rule
    - Sequential composition rule
    - Alternation rule
    - Iteration rule

Carroll Morgan
https://my.cse.unsw.edu.au/staff/staff_details.php?ID=carrollm

Developing correct programs from specification[Mor98]

- Rules of Refinement

- Assignment rule: $[\varphi(v/e), \psi] \prec v := e$
- Sequential composition rule ($\gamma - middlepredicate$)
  $[\eta_1, \eta_2] \prec [\eta_1, \gamma]$
  $\qquad\qquad [\gamma, \eta_2]$
- Alternation rule, $G = g_1 \vee g_2 \vee \ldots \vee g_n$
  $[\eta_1, \eta_2] \prec$
  if $g_1 \rightarrow [\eta_1 \wedge g_1, \eta_2]$
  $\quad \Box g_2 \rightarrow [\eta_1 \wedge g_2, \eta_2]$
  $\quad \vdots$
  $\quad \Box g_n \rightarrow [\eta_1 \wedge g_n, \eta_2]$
  fi
- Iteration rule $G = g_1 \vee g_2 \vee \ldots \vee g_n$
  $[\eta, \eta \wedge \neg G] \prec$
  do $g_1 \rightarrow [\eta \wedge g_1, \eta \wedge TC]$
  $\quad \Box g_2 \rightarrow [\eta \wedge g_2, \eta \wedge TC]$
  $\quad \vdots$
  $\quad \Box g_n \rightarrow [\eta \wedge g_n, \eta \wedge TC]$
  do

Developing correct programs from specification[Mor98]

- Examples

- See the file with the examples
  - One example is discussed during lecture.
- Book [Mor98]

Outline

- Correctness
- Floyd's Method -Inductive assertions, Partial correctness, Termination
- Hoare Logic, Semantics of Hoare triples, Partial correctness, Total correctness
- Dijkstra's Language, Guarded commands, Nondeterminacy, Formal Derivation of Programs
- Developing correct programs from specification, Refinement, Rules of Refinement, Examples
- Static analysis, JML- Java Modeling Language, ESC/Java2- Extended Static Checker for Java
- Questions
- Next lecture
  - **EVOZON** Presentation, topic: **Test automation**
  - **When:** Friday, April 12, 2019, hours 14:00-16:00;
  - **Where:** Room A2 (FSEGA Building)

Program verification methods - Correctness

- Lecture 1 - Verification and Validation
  - Verification
    - reviews products to ensure their quality → correctness
    - static and dynamic analysis techniques
  - A **correct program** is one that does exactly what it is intended to do, no more and no less.
  - A formally correct program is one whose correctness can be **proved** mathematically.
    - This requires a language for specifying precisely what the program is intended to do.
    - Specification languages are based in mathematical logic.
  - Until recently, correctness has been an academic exercise. – Now it is a key element of critical software systems
- **Program verification - correctness**
  - proof-based, computer-assisted, program-verification approach, mainly used for programs which we expect to terminate and produce a result
  - model-based, automatic, property-verification approach, mainly used for concurrent, reactive systems (originally used in a post-development stage) - model checking (Lecture 10)
- **Correctness Tools**
  - Theorem provers (PVS), Modeling languages (UML and OCL), Specification languages (JML), Programming language support (Eiffel, Java, Spark/Ada), Specification Methodology (Design by contract)
- **Methods for prooving program correctness**
  - Floyd's Method - Inductive assertions
  - Hoare - Semantics of Hoare triples
  - Dijkstra's Language- Guarded commands, Nondeterminacy and Formal Derivation of Programs

Program verification methods - Correctness

- **Software engineering problem:** building/maintaining **correct** systems.
  - How?
    - Specification
    - Tools
- Formal Methods in Software Engineering
  - Formal languages guarantee
    - Precision (no ambiguity)
    - Certainty (modeling errors)
    - Automation (automatic verification tools).

- JML- Java Modeling Language
  - Demo JML
- ESC/Java2- Extended Static Checker for Java
  - Demo ESCJava2

Remark. ESC/Java tool - Topic of Laboratory 6!

- Things to do:
  1) make a ***formal model***
  2) ***specify properties*** for the model
  3) ***verify/check*** the properties

- Formal methods and JML (Java Modeling Language):
  1) formal model is ***Java programming language***
  2) the properties are specified in ***JML***
  3) Properties may be
     - ***Tested*** using ***jmlrac***
     - ***Checked*** using ***ESC2Java***

fppt.com

What is JML?

- Gary T. Leavens's JML group at the University of Central Florida
- http://www.eecs.ucf.edu/~leavens/JML//index.shtml

- a behavioral interface specification language
- used to specify the behavior of Java modules
- combines
  - design by contract approach
  - the model-based specification approach
  - some elements of the refinement calculus

Tools for using JML

- Runtime assertion checkers (e.g. **jmlc/jmlrac**)
- Static checkers (**ESC2Java**)

- Test generation (e.g. jmlunit)
- Formal verification tools (e.g. KeY)
- Design tools (e.g. AutoJML)

fppt.com

Tools for JML

Runtime assertion checking with jmlc/jmlrac

- Special compiler inserts runtime tests for all JML assertions. Any assertion violation results in a special exception.
- checks specs at run-time
- only **tests** correctness of **specs**.
- **Find violations at runtime.**

Extended static checking with ESC/Java

- Automatically tries to prove simple JML assertions at compile time.
- checks specs at compile-time
- **proves** correctness of **specs**.
- **Warn about likely runtime exceptions and violations.**

JML web page
- http://www.eecs.ucf.edu/~leavens/JML//index.shtml

ESC/Java2 web page
http://www.kindsoftware.com/products/opensource/ESCJava2/download.html

fppt.com

Design by contract

Contract?

Method contract

Precondition

Specifies "caller's responsibility"

- Constraints on parameter values and target object's state.
- Valid object's states, in which a method can be called.

Intuitively

- Expression that must hold at the entry to the method.

Postcondition

Specifies "implementation's responsibility"

- Constraints on the method's return value and side effects.
- Relation between initial and final state of the method.

Intuitively

- Expression that must hold at the exit from the method.

Class contract

Invariant

- Specifies caller's responsibility at the entry to a method and implementation's responsibility at the exit from a method.
- Valid states of class instances (values of fields).

Intuitively

- Expression that must hold at the entry and exit of each method in the class.

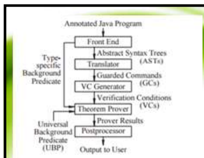
Tools for JML

Runtime assertion checking with jmlc/jmlrac

- Special compiler inserts runtime tests for all JML assertions. Any assertion violation results in a special exception.
- checks specs at run-time
- only **tests** correctness of **specs**.
- **Find violations at runtime.**

***jmlc* and *jmlrac* – by example**

- Demo 01: Factorial
- Demo02: Integer sqrt



- Unsound ?
- Incomplete ?

Tools for JML

Extended static checking with ESC/Java

- Automatically tries to prove simple JML assertions at compile time.
- checks specs at compile-time
- **proves** correctness of **specs**
- **Warn about likely runtime exceptions and violations.**

ESC/Java2 – by example

- Demo 01: Fast exponentiation
- Demo 02: MyArray
- Demo 03: MySet

Next Lecture

- **EVOZON** Presentation, topic: **Test automation**
- **When:** Friday, April 12, 2019, hours 14:00-16:00;
- **Where:** Room A2 (FSEGA Building)

Questions

- Thank You For Your Attention!

References

- [Flo67] Robert W. Floyd, Assigning meanings to programs, In *Proceedings of Symposia in Applied Mathematics*, pages 19–32, 1967.
- [Hoa69] C. A. R. Hoare, An axiomatic basis for computer programming, *Commun. ACM*, 12(10):576–580, 1969.
- [Dij75] E. Dijkstra, Guarded commands, nondeterminacy and formal derivation of programs.
- CACM, 8(18):453–457, 1975.
- [Fre10] M. Frentiu, Verificarea si validarea sistemelor soft, Presa Universitara Clujeana, 2010.
- [Mor98] C. Morgan, Programming from Specifications, Prentice Hall International Series in Computing Science, 1998