Program Analysis

INF4140 - Models of concurrency

Program Analysis, lecture 5

Høsten 2014

26.9.2014

Is my program correct?

Central question for this and the next lecture.

- Does the program behave as intended?
- Surprising behavior?

$$x := 5; \{\ x = 5\ \}\langle x := x + 1\rangle; \{\ x =?\ \}$$

- clear: $x = 5$ *immediately* after first assignment
- Will this still hold when the second assignment is executed?
  - Depends on other processes
- What will be the final value of x?

Today: Basic machinery for program reasoning
Next week: Extending this machinery to the concurrent setting

- Concurrent program: several threads operating on (here) *shared* variables
- Parallel updates to x and y:

$$\text{co } \langle x = x \times 3; \rangle \parallel \langle y := y \times 2; \rangle \text{ oc}$$

- Every concurrent execution can be written as a sequence of atomic operations (gives one history)
- Two possible histories for the above program
- Generally, if n processes executes m atomic operations each:

$$\frac{(n * m)!}{m!^n} \qquad \text{If n=3 and m=4: } \frac{(3 * 4)!}{4!^3} = 34650$$

How to verify program properties?

- *Testing* or *debugging* increases confidence in the program correctness, but does not guarantee *correctness*
  - Program testing can be an effective way to show the presence of bugs, but not their absence

- *Operational reasoning* (exhaustive case analysis) tries all possible executions of a program

- *Formal analysis* (assertional reasoning) allows to *deduce* the correctness of a program without executing it
  - *Specification* of program behavior
  - Formal argument that the specification is correct

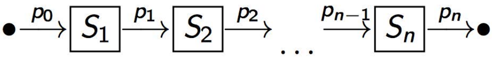
- A *state* of a program consists of the values of the program variables at a point in time, example: $\{ x = 2 \wedge y = 3 \}$
- The *state space* of a program is given by the different values that the declared variables can take
- Sequential program: one execution thread operates on its own state space
- The state may be *changed* by assignments ("imperative")

Example

$$\{ x = 5 \wedge y = 5 \}\mathtt{x} := \mathtt{x} * 2;\{ x = 10 \wedge y = 5 \}\mathtt{y} := \mathtt{y} * 2;\{ x = 10 \wedge y = 10 \}$$

Executions

- Given program S as sequence $S_1; S_2; \ldots; S_n;$, starting in a state p_0:



  where $p_1, p_2, \ldots p_n$ are the different states during execution
- Can be documented by: $\{p_0\}S_1\{p_1\}S_2\{p_2\}\ldots\{p_{n-1}\}S_n\{p_n\}$
- p_0, p_n gives an external specification of the program: $\{p_0\}S\{p_n\}$
- We often refer to p_0 as the *initial* state and p_n as the *final* state

Example (from previous slide)

$$\{\ x = 5 \land y = 5\ \}\ \mathtt{x} := \mathtt{x} * 2;\ \ \mathtt{y} := \mathtt{y} * 2;\ \{\ x = 10 \land y = 10\ \}$$

Want to express more general properties of programs, like

$$\{\ x = y\ \}\mathtt{x := x * 2;y = y * 2;}\{\ x = y\ \}$$

- If the assertion $x = y$ holds, when the program *starts*, $x = y$ will also hold when/if the program *terminates*
- Does not talk about particular *values* of x and y, but about *relations* between their values
- Assertions characterise *sets* of states

Example

The assertion $x = y$ describes *all* states where the values of $\mathtt{x}$ and $\mathtt{y}$ are equal, like $\{x = -1 \wedge y = -1\}$, $\{x = 1 \wedge y = 1\}$, ...

- An assertion P can be viewed as a *set* of states where P is true:

$x = y$	All states where x has the same value as y
$x \leq y$:	All states where the value of x is less or equal to the value of y
$x = 2 \wedge y = 3$	Only one state (if x and y are the only variables)
true	All states
false	No state

Example

$$\{ x = y \}\mathrm{x} := \mathrm{x} * 2;\{ x = 2 * y \}\mathrm{y} := \mathrm{y} * 2;\{x = y\}$$

Then this must also hold for particular values of x and y satisfying the initial assertion, like $x = y = 5$

- Establish program properties, using a system for formal reasoning
- Help in understanding how a program behaves
- Useful for program construction
- Look at logics for formal analysis
- basis of analysis tool

Formal system

- *Axioms:* Defines the meaning of individual program statements
- *Rules:* Derive the meaning of a program from the individual statements in the program

Our formal system consists of:

- A set of *symbols* (constants, variables,...)
- A set of *formulas* (meaningful combination of symbols)
- A set of *axioms* (assumed to be true)
- A set of *inference rules* of the form:

Inference rule

$$\frac{H_1 \quad \dots \quad H_n}{C}$$

- Where each H_i is an *assumption*, and C is the *conclusion*
- The conclusion is true if all the assumptions are true
- The inference rules specify how to derive additional true formulas from axioms and other true formulas.

- (program + extra) variables: $x, y, z, \ldots$
- Relation symbols: $\leq, \geq, \ldots$
- Function symbols: $+, -, \ldots$, and constants $0, 1, 2, \ldots, \mathit{true}, \mathit{false}$
- Equality (also a relation symbol): $=$

Meaningful combination of symbols

Assume that A and B are formulas, then the following are also formulas:

$\neg A$	means "not A"
$A \vee B$	means "A or B"
$A \wedge B$	means "A and B"
$A \Rightarrow B$	means "A implies B"

If x is a variable and A, the following are formulas:[1]

$\forall x : A(x)$	means "A is true for all values of x"
$\exists x : A(x)$	means "there is (at least) one value of x such that A is true"

[1] $A(x)$ to indicate that, here, A (typically) contains x.

Examples of axioms and rules

Typical axioms:

- $A \vee \neg A$
- $A \Rightarrow A$

Typical rules:

$$\frac{A \qquad B}{A \wedge B}\ \text{And-I} \qquad \frac{A}{A \vee B}\ \text{Or-I} \qquad \frac{A \Rightarrow B \qquad A}{B}\ \text{Or-E}$$

Example

$$\frac{x = 5 \qquad y = 5}{x = 5 \wedge y = 5}\ \text{And-I} \qquad \frac{x = 5}{x = 5 \vee y = 5}\ \text{Or-I}$$

$$\frac{x \geq 0 \Rightarrow y \geq 0 \qquad x \geq 0}{y \geq 0}\ \text{Or-E}$$

Important terms

- **Interpretation**: describe each formula as either *true* or *false*
- **Proof**: derivation tree where all leaf nodes are axioms
- **Theorems**: a "formula" derivable in a given proof system
- **Soundness** (of the logic): If we can prove ("derive") some formula P (in the logic) then P is actually (semantically) true
- **Completeness**: If a formula P is true, it can be proven

- PL lets us *express* and *prove* properties about programs
- *Formulas* are of the form

"Hoare triple"

$$\{ P_1 \} \; S \; \{ P_2 \}$$

- S: program statement(s)
- P, P_1, P', Q ...: assertions over program states (including $\neg, \wedge, \vee, \exists, \forall$)
- In above triple P_1: Pre-condition, and P_2 post-condition of S

Example

$$\{ x = y \} \; \mathtt{x} := \mathtt{x} * 2; \mathtt{y} := \mathtt{y} * 2; \; \{ x = y \}$$

- Express and prove program properties
- $\{P\}\ S\ \{Q\}$
  - P, Q may be seen as a specification of the program S
  - Code analysis by proving the specification (in PL)
  - No need to execute the code in order to do the analysis
  - An *interpretation* maps triples to *true* or *false*
    - $\{\ x = 0\ \}\ x := x + 1;\ \{\ x = 1\ \}$ should be *true*
    - $\{\ x = 0\ \}\ x := x + 1;\ \{\ x = 0\ \}$ should be *false*

Reasoning about programs

- Basic idea: *Specify* what the program is supposed to do (pre- and post-conditions)
- Pre- and post-conditions are given as assertions over the program state
- Use PL for amathematical argument that the program satisfies its specification

Interpretation

Interpretation ("semantics") of triples is related to code execution

Partial correctness interpretation

$\{P\}\ S\ \{Q\}$ is *true*/holds, if the following is the case:

- If the initial state of S satisfies P (P holds for the initial state of S),
- and **if**[a] S *terminates*,
- *then* Q is *true* in the final state of S

[a] Thus: if S does not terminate, all bets are off...

Expresses *partial correctness* (termination of S is assumed)

Example

$\{x = y\}$ $\mathtt{x := x * 2; y := y * 2;}$ $\{x = y\}$ is *true*
if the initial state satisfies $x = y$ and, in case the execution terminates, then the final state satisfies $x = y$

Some true formulas:

$$\{ x = 0 \} \; x := x + 1; \; \{ x = 1 \}$$
$$\{ x = 4 \} \; x := 5; \; \{ x = 5 \}$$
$$\{ \texttt{true} \} \; x := 5; \; \{ x = 5 \}$$
$$\{ y = 4 \} \; x := 5; \; \{ y = 4 \}$$
$$\{ x = 4 \} \; x := x + 1; \; \{ x = 5 \}$$
$$\{ x = a \wedge y = b \} \; x = x + y; \; \{ x = a + b \wedge y = b \}$$
$$\{ x = 4 \wedge y = 7 \} \; x := x + 1; \; \{ x = 5 \wedge y = 7 \}$$
$$\{ x = y \} \; x := x + 1; y := y + 1; \; \{ x = y \}$$

Some formulas that are not *true*:

$$\{ x = 0 \} \; x := x + 1; \; \{ x = 0 \}$$
$$\{ x = 4 \} \; x := 5; \; \{ x = 4 \}$$
$$\{ x = y \} \; x := x + 1; y := y - 1; \; \{ x = y \}$$
$$\{ x > y \} \; x := x + 1; y := y + 1; \; \{ x < y \}$$

- The interpretation of $\{ P \} S \{ Q \}$ assumes/ignores termination of S, termination is not proven.
- The assertions (P, Q) express *safety* properties
- The pre- and postconditions *restrict* possible states

The assertion *true* can be viewed as all states. The assertion *false* can be viewed as no state. What does each of the following triple express?

$$\{ P \}\ S;\ \{ \mathit{false} \}$$
$$\{ P \}\ S;\ \{ \mathit{true} \}$$
$$\{ \mathit{true} \}\ S;\ \{ Q \}$$

$$\{ \mathit{false} \}\ S;\ \{ Q \}$$

- The interpretation of $\{ P \} S \{ Q \}$ assumes/ignores termination of S, termination is not proven.
- The assertions (P, Q) express *safety* properties
- The pre- and postconditions *restrict* possible states

The assertion *true* can be viewed as all states. The assertion *false* can be viewed as no state. What does each of the following triple express?

$\{ P \}\ S;\ \{ \mathit{false} \}$ S does not terminate

$\{ P \}\ S;\ \{ \mathit{true} \}$

$\{ \mathit{true} \}\ S;\ \{ Q \}$

$\{ \mathit{false} \}\ S;\ \{ Q \}$

- The interpretation of $\{ P \} S \{ Q \}$ assumes/ignores termination of S, termination is not proven.
- The assertions (P, Q) express *safety* properties
- The pre- and postconditions *restrict* possible states

The assertion *true* can be viewed as all states. The assertion *false* can be viewed as no state. What does each of the following triple express?

$\{ P \} S; \{ \mathit{false} \}$	S does not terminate
$\{ P \} S; \{ \mathit{true} \}$	trivially true
$\{ \mathit{true} \} S; \{ Q \}$	
$\{ \mathit{false} \} S; \{ Q \}$	

- The interpretation of $\{ P \} S \{ Q \}$ assumes/ignores termination of S, termination is not proven.
- The assertions (P, Q) express *safety* properties
- The pre- and postconditions *restrict* possible states

The assertion *true* can be viewed as all states. The assertion *false* can be viewed as no state. What does each of the following triple express?

$\{ P \} S; \{ \mathit{false} \}$	S does not terminate
$\{ P \} S; \{ \mathit{true} \}$	trivially true
$\{ \mathit{true} \} S; \{ Q \}$	Q holds after S in any case (provided S terminates)
$\{ \mathit{false} \} S; \{ Q \}$	

- The interpretation of $\{ P \} S \{ Q \}$ assumes/ignores termination of S, termination is not proven.
- The assertions (P, Q) express *safety* properties
- The pre- and postconditions *restrict* possible states

The assertion *true* can be viewed as all states. The assertion *false* can be viewed as no state. What does each of the following triple express?

$\{ P \}\ S;\ \{ \mathit{false} \}$	S does not terminate
$\{ P \}\ S;\ \{ \mathit{true} \}$	trivially true
$\{ \mathit{true} \}\ S;\ \{ Q \}$	Q holds after S in any case (provided S terminates)
$\{ \mathit{false} \}\ S;\ \{ Q \}$	trivially true

A proof system consists of *axioms* and *rules*
here: structural analysis of programs

- Axioms for basic statements:
  - $x := e$, `skip`,...
- Rules for composed statements:
  - $S_1;S_2$, `if`, `while`, `await`, `co`...`oc`, ...

Formulas in PL

- formulas = triples
- theorems = derivable formulas
- hopefully: all derivable formulas are also "really" (= semantically) true
- derivation: starting from axioms, using derivation rules
- $$\frac{H_1 \quad H_2 \quad \ldots \quad H_n}{C}$$
- axioms: can be seen as rules without premises

If a triple $\{ P \}\ S\ \{ Q \}$ is a *theorem* in PL, the triple is actually true!

- Example: we want

$$\{ x = 0 \}\ x := x + 1\ \{ x = 1 \}$$

  to be a theorem (since it was interpreted as *true*),
- but

$$\{ x = 0 \}\ x = x + 1\ \{ x = 0 \}$$

  should *not* be a theorem (since it was interpreted as *false*)

Soundness: All theorems in PL are true

If we can use PL to prove some property of a program, then this property will hold for all executions of the program

(Textual) substitution

$P_{x \leftarrow e}$ means, all free occurrences of x in P are replaced by expression e.

Example

$$
\begin{array}{lcl}
(x = 1)_{x \leftarrow (x+1)} & \Leftrightarrow & x + 1 = 1 \\
(x + y = a)_{y \leftarrow (y+x)} & \Leftrightarrow & x + (y + x) = a \\
(y = a)_{x \leftarrow (x+y)} & \Leftrightarrow & y = a
\end{array}
$$

Substitution propagates into formulas:

$$
\begin{array}{lcl}
(\neg A)_{x \leftarrow e} & \Leftrightarrow & \neg (A_{x \leftarrow e}) \\
(A \wedge B)_{x \leftarrow e} & \Leftrightarrow & A_{x \leftarrow e} \wedge B_{x \leftarrow e} \\
(A \vee B)_{x \leftarrow e} & \Leftrightarrow & A_{x\ \leftarrow e} \vee B_{x \leftarrow e}
\end{array}
$$

$P_{x \leftarrow e}$

- Only *free* occurrences of x are substituted
- Variable occurrences may be *bound* by quantifiers, then that occurrence of the variable is not free (but bound)

Example (Substitution)

$$(\exists y : x + y > 0)_{x \leftarrow 1} \quad \Leftrightarrow$$
$$(\exists x : x + y > 0)_{x \leftarrow 1} \quad \Leftrightarrow$$
$$(\exists x : x + y > 0)_{y \leftarrow x} \quad \Leftrightarrow$$

Correspondingly for $\forall$

$P_{x \leftarrow e}$

- Only *free* occurrences of x are substituted
- Variable occurrences may be *bound* by quantifiers, then that occurrence of the variable is not free (but bound)

Example (Substitution)

$$
\begin{aligned}
(\exists y : x + y > 0)_{x \leftarrow 1} &\Leftrightarrow \exists y : 1 + y > 0 \\
(\exists x : x + y > 0)_{x \leftarrow 1} &\Leftrightarrow \\
(\exists x : x + y > 0)_{y \leftarrow x} &\Leftrightarrow
\end{aligned}
$$

Correspondingly for $\forall$

$P_{x \leftarrow e}$

- Only *free* occurrences of x are substituted
- Variable occurrences may be *bound* by quantifiers, then that occurrence of the variable is not free (but bound)

Example (Substitution)

$$
\begin{array}{lcl}
(\exists y : x + y > 0)_{x \leftarrow 1} & \Leftrightarrow & \exists y : 1 + y > 0 \\
(\exists x : x + y > 0)_{x \leftarrow 1} & \Leftrightarrow & \exists x : x + y > 0 \\
(\exists x : x + y > 0)_{y \leftarrow x} & \Leftrightarrow &
\end{array}
$$

Correspondingly for $\forall$

$P_{x \leftarrow e}$

- Only *free* occurrences of x are substituted
- Variable occurrences may be *bound* by quantifiers, then that occurrence of the variable is not free (but bound)

Example (Substitution)

$$
\begin{array}{rcl}
(\exists y : x + y > 0)_{x \leftarrow 1} & \Leftrightarrow & \exists y : 1 + y > 0 \\
(\exists x : x + y > 0)_{x \leftarrow 1} & \Leftrightarrow & \exists x : x + y > 0 \\
(\exists x : x + y > 0)_{y \leftarrow x} & \Leftrightarrow & \exists z : z + x > 0
\end{array}
$$

Correspondingly for $\forall$

Given by backward construction over the assignment:

- Given the postcondition to the assignment, we may derive the precondition!

What is the precondition?

$$\{\ ?\ \}\ x = e\ \{\ x == 5\ \}$$

If the assignment `x = e` should terminate in a state where `x` has the value 5, the expression `e` must have the value 5 before the assignment:

$$\{e == 5\} \quad \texttt{x} = \texttt{e} \quad \{x == 5\}$$
$$\{(x == 5)_{x \leftarrow e}\} \quad \texttt{x} = \texttt{e} \quad \{x == 5\}$$

Given the postcondition, we may construct the precondition:

Axiom for the assignment statement

$$\{\ P_{x \leftarrow e}\ \}\ x := e\ \{\ P\ \} \quad \text{Assign}$$

If the assignment $x = e$ should lead to a state that satisfies P, the state before the assignment must satisfy P where x is replaced by e.

Proving an assignment

In order to prove the triple $\{P\}x = e\{Q\}$ in PL, we must show that the precondition P implies $Q_{x\leftarrow e}$

$$\frac{P \Rightarrow Q_{x\leftarrow e} \qquad \{\ Q_{x\leftarrow e}\ \}\ x := e\ \{\ Q\ \}}{\{\ P\ \}\ x := e\ \{\ Q\ \}}$$

The blue implication is a logical proof obligation. In this course we only convince ourself that these are true (we do not prove them formally).

- $Q_{x\leftarrow e}$ is the largest set of states such that the assignment is guaranteed to terminate with Q
- We must show that the set of states P is within this set

Examples

$$\frac{true \Rightarrow 1 == 1}{\{true\}\ x = 1;\ \{x == 1\}}$$

$$\frac{x == 0 \Rightarrow x + 1 == 1}{\{x == 0\}\ x = x + 1;\ \{x == 1\}}$$

$$\frac{(x == a \wedge y == b) \Rightarrow x + y == a + b \wedge y == b}{\{x == a \wedge y == b\}\ x = x + y;\ \{x == a + b \wedge y == b\}}$$

$$\frac{x == a \Rightarrow 0 * y + x == a}{\{x == a\}\ q = 0;\ \{q * y + x == a\}}$$

$$\frac{y > 0 \Rightarrow y \geq 0}{\{y > 0\}\ x = y;\ \{x \geq 0\}}$$

The skip statement does nothing

Axiom:

$$\{\ P\ \}\ \texttt{skip}\ \{\ P\ \} \quad \text{Skip}$$

$$\frac{\{\ P\ \}\ S_1\ \{\ R\ \} \qquad \{\ R\ \}\ S_2\ \{\ Q\ \}}{\{\ P\ \}\ S_1; S_2\ \{\ Q\ \}}\ \text{Seq}$$

$$\frac{\{\ P \wedge B\ \}\ S_1\ \{\ Q\ \} \qquad P \wedge \neg B \Rightarrow Q}{\{\ P\ \}\ \texttt{if}\ B\ \texttt{then}\ S\ \{\ Q\ \}}\ \text{Cond}'$$

$$\frac{\{\ I \wedge B\ \}\ S\ \{\ I\ \}}{\{\ I\ \}\ \texttt{while}\ B\ \texttt{do}\ S\ \{\ I \wedge \neg B\ \}}\ \text{While}$$

$$\frac{\{\ P\ \}\ S\ \{\ Q\ \} \qquad P' \Rightarrow P \qquad Q \Rightarrow Q'}{\{\ P'\ \}\ S\ \{\ Q'\ \}}\ \text{Consequence}$$

- Blue: logical proof obligations
- the rule for `while` needs a *loop invariant*!
- `for`-loop: exercise 2.22!

Backward construction over assignments:

$$\dfrac{\dfrac{x = y \Rightarrow 2 * x = 2 * y}{\{\ x = y\ \}\ x := x * 2\ \{\ x = 2 * y\ \}} \qquad \{\ (x = y)_{y \leftarrow 2y}\ \}\ y := y * 2\ \{\ x =}{\{\ x = y\ \}\ x := x * 2; y := y * 2\ \{\ x = y\ \}}$$

Sometimes we don't bother to write down the assignment axiom:

$$\dfrac{\dfrac{(q * y) + x = a \Rightarrow ((q + 1) * y) + x - y = a}{\{\ (q * y) + x = a\ \}\ x := x - y;\ \{\ ((q + 1) * y) + x = a\ \}}}{\{\ (q * y) + x = a\ \}\ x := x - y; q := q + 1\ \{\ (q * y) + x = a\ \}}$$

Logical variables

- Do *not* occur in program text
- Used only in *assertions*
- May be used to "freeze" initial values of variables
- May then talk about these values in the postcondition

Example

$$\{ x = x_0 \} \ \textit{if} \ (x < 0) \ \texttt{then} \ x := -x \ \{ x \geq 0 \wedge (x = x_0 \vee x = -x_0) \}$$

where $(x = x_0 \vee x = -x_0)$ states that

- the final value of x equals the initial value, *or*
- the final value of x is the negation of the initial value

Example: if statement

Verification of:

$$\{\ x = x_0\ \}\ \texttt{if}\ (x < 0)\ \texttt{then}\ x := -x\ \{\ x \geq 0 \wedge (x = x_0 \vee x = -x_0)\ \}$$

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad (P \wedge \neg B) \Rightarrow Q}{\{\ P\ \}\ \texttt{if}\ \ B\ \texttt{then}\ \ S\ \{\ Q\ \}}\ \text{Cond}'$$

- $\{\ P \wedge B\ \}\ S\ \{\ Q\ \}$:
  $\{\ x = x_0 \wedge x < 0\ \}\ x := -x\ \{\ x \geq 0 \wedge (x = x_0 \vee x = -x_0)\ \}$
  Backward construction (assignment axiom) gives the implication:
  $x = x_0 \wedge x < 0 \Rightarrow (-x \geq 0 \wedge (-x = x_0 \vee -x = -x_0))$
- $P \wedge \neg B \Rightarrow Q$:
  $x = x_0 \wedge x \geq 0 \Rightarrow (x \geq 0 \wedge (x = x_0 \vee x = -x_0))$

Verification of:

$$\{\ x = x_0\ \}\ \texttt{if}\ (x < 0)\ \texttt{then}\ x := -x\ \{\ x \geq 0 \wedge (x = x_0 \vee x = -x_0)\ \}$$

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad (P \wedge \neg B) \Rightarrow Q}{\{\ P\ \}\ \texttt{if}\ \ B\ \texttt{then}\ \ S\ \{\ Q\ \}}\ \text{Cond}'$$

- $\{\ P \wedge B\ \}\ S\ \{\ Q\ \}$:
  $\{\ x = x_0 \wedge x < 0\ \}\ x := -x\ \{\ x \geq 0 \wedge (x = x_0 \vee x = -x_0)\ \}$
  Backward construction (assignment axiom) gives the implication:
  $x = x_0 \wedge x < 0 \Rightarrow (-x \geq 0 \wedge (-x = x_0 \vee -x = -x_0))$

- $P \wedge \neg B \Rightarrow Q$:
  $x = x_0 \wedge x \geq 0 \Rightarrow (x \geq 0 \wedge (x = x_0 \vee x = -x_0))$