INF4140 - Models of concurrency

Fall 2016

September 23, 2016

Abstract

This is the "handout" version of the slides for the lecture (i.e., it's a rendering of the content of the slides in a way that does not waste so much paper when printing out). The material is found in [Andrews, 2000]. Being a handout-version of the slides, some figures and graph overlays may not be rendered in full detail, I remove most of the overlays, especially the long ones, because they don't make sense much on a handout/paper. Scroll through the real slides instead, if one needs the overlays.

This handout version also contains more remarks and footnotes, which would clutter the slides, and which typically contains remarks and elaborations, which may be given orally in the lecture.

1 Program analysis

26. 9. 2016

Program correctness

Is my program correct? Central question for this and the next lecture.

- Does a given 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 executions

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

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

- 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

States

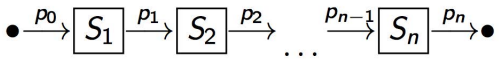
- *state* of a program consists of the values of the program variables at a point in time, example: $\{\ x = 2 \wedge y = 3\ \}$
- The state space of a program is given by the different values that the declared variables can take
- Sequential program: one execution thread operates on its own state space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

Example 1.

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

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 2 (from previous slide).

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

Assertions

Want to express more general properties of programs, like

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

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

Example 3. The assertion $x = y$ describes *all* states where the values of x and y are equal, like $\{x = -1 \wedge y = -1\}$, $\{x = 1 \wedge y = 1\}$, ...

Assertions

- state assertion P: *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 4.

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

Assertions may or may not say something correct for the behavior of a program (fragment). In this example, the assertions say something correct.

Formal analysis of programs

- establish program properties/correctness, 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 tools

Formal system

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

Logics and formal systems

Our formal system consists of:

- syntactic building blocks:
  - A set of *symbols* (constants, variables,...)
  - A set of *formulas* (meaningful combination of symbols)
- derivation machinery
  - A set of *axioms* (assumed to be true)
  - A set of *inference rules*

Inference rule[1]

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

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

[1] axiom = rule with no premises

Symbols

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

Formulas of first-order logic

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:[2]

$\forall x : A(x)$	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"

Examples of axioms and rules (no programs involved yet)

Typical axioms:

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

Typical rules:[3]

$$\frac{A \qquad B}{A \wedge B}\ \textsc{And-I} \qquad \frac{A}{A \vee B}\ \textsc{Or-I} \qquad \frac{A \Rightarrow B \qquad A}{B}\ \textsc{Impl-E/modus ponens}$$

Example 5.

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

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

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

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

[3] The "names" of the rules are written on the right of the rule, they serve for "identification". By some convention, "I" stands for rules *introducing* some logical connector, "E" for *eliminating* one.

Program Logic (PL)

- 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 6.

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

The proof system PL (Hoare logic)

- 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 a mathematical argument that the program satisfies its specification

Interpretation:
Interpretation ("semantics") of triples is related to program execution

Partial correctness interpretation
$\{\ P\ \}\ S\ \{\ Q\ \}$ is true/holds:

- If the initial state of S satisfies P (P holds for the initial state of S) and
- **if**[4] *S terminates,*
- *then* Q is *true* in the final state of S

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

Example 7. $\{x = y\}\ \texttt{x} := \texttt{x} * 2;\texttt{y} := \texttt{y} * 2;\ \{x = y\}$ is *true* if the initial state satisfies $x = y$ and, in case the execution terminates, then the final state satisfies $x = y$

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

Examples

Some true triples

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

Some non-true triples

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

Partial correctness

- The interpretation of $\{\ P\ \}\ S\ \{\ Q\ \}$ assumes/ignores termination of S, termination is not proven.
- The pre/post specification (P, Q) express *safety* properties

The state 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

Proof system PL

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[5]
- 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

[5]The terminology is standard from general logic. A "theorem" in an derivation system is a *derivable* formula. In an ill-defined (i.e., unsound) derivation or proof system, theorems may thus be not true.

Soundness

If a triple $\{ P \}\ S\ \{ Q \}$ is a *theorem* in PL (i.e., derivable), the triple holds

- 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:[6]
All theorems in PL hold

$$\vdash \{ P \}\ S\ \{ Q \} \quad \text{implies} \quad \models \{ P \}\ S\ \{ Q \} \tag{1}$$

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

Substitution
$P[e/x]$ means, all free occurrences of x in P are replaced by expression e.

Example 8.
$$\begin{array}{lll} (x = 1)[(x+1)/x] & \Leftrightarrow & x + 1 = 1 \\ (x + y = a)[(y+x)/y] & \Leftrightarrow & x + (y + x) = a \\ (y = a)[(x+y)/x] & \Leftrightarrow & y = a \end{array}$$

Substitution propagates into formulas:
$$\begin{array}{lll} (\neg A)[e/x] & \Leftrightarrow & \neg(A[e/x]) \\ (A \wedge B)[e/x] & \Leftrightarrow & A[e/x] \wedge B[e/x] \\ (A \vee B)[e/x] & \Leftrightarrow & A[e/x] \vee B[e/x] \end{array}$$

Free and "non-free" variable occurrences
$P[e/x]$

- 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 9 (Substitution).
$$\begin{array}{lll} (\exists y : x + y > 0)[1/x] & \Leftrightarrow & \exists y : 1 + y > 0 \\ (\exists x : x + y > 0)[1/x] & \Leftrightarrow & \exists x : x + y > 0 \\ (\exists x : x + y > 0)[x/y] & \Leftrightarrow & \exists z : z + x > 0 \end{array}$$

Correspondingly for $\forall$

The assignment axiom – Motivation
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:

$$\begin{array}{rlr} \{ e = 5 \} & \texttt{x := e} & \{ x = 5 \} \\ \{ (x = 5)[e/x] \} & \texttt{x := e} & \{ x = 5 \} \end{array}$$

[6] technically, we'd need a semantics for reference, otherwise it's difficult to say what a program "really" does.

Axiom of assignment

"Backwards reasoning:" Given a postcondition, we may construct the precondition:

Axiom for the assignment statement

$$\{\ P[e/x]\ \}\ x := e\ \{\ P\ \}\quad \textsc{Assign}$$

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

To prove the triple $\{\ P\ \}x := e\{\ Q\ \}$ in PL, we must show that the precondition P implies $Q[e/x]$

$$\frac{P \Rightarrow Q[e/x] \qquad \{\ Q[e/x]\ \}\ x := e\ \{\ Q\ \}}{\{\ P\ \}\ x := e\ \{\ Q\ \}}$$

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

- $Q[e/x]$ is the largest set of states such that the assignment is guaranteed to terminate with Q
- largest set corresponds to weakest condition $\Rightarrow$ weakest-precondition reasoning
- We must show that the set of states P is within this set

Examples

$$\frac{\mathit{true} \Rightarrow 1 = 1}{\{\ \mathit{true}\ \}\ x := 1\ \{\ x = 1\ \}}$$

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

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

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

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

Axiom of skip

The skip statement does nothing

Axiom:

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

PL inference rules

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

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

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

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

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

Sequential composition and consequence

Backward construction over assignments:

$$\frac{\dfrac{x = y \Rightarrow 2x = 2y}{\{\ x = y\ \}\ x := 2x\ \{\ x = 2y\ \}} \qquad \{\ (x = y)[2y/y]\ \}\ y := 2y\ \{\ x = y\ \}}{\{\ x = y\ \}\ x := 2x; y := 2y\ \{\ x = y\ \}}$$

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

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

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 10.

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

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:

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

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

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

  $x = x_0 \wedge x < 0 \Rightarrow (-x \geq 0 \wedge (-x = x_0 \vee -x = -x_0))$

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

References

[Andrews, 2000] Andrews, G. R. (2000). *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison-Wesley.