Program Analysis (week 5)

INF4140 - Models of concurrency

Program Analysis, lecture 5

Høsten 2013

23.9.2013

Is my program correct?

Central question for this and the next lecture.

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

$$x = 5; \{x == 5\} < x = x + 1; > \{x == ?\}$$

- Know that $x == 5$ 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 *shared* variables
- Parallel updates to x and y:

$$\textbf{co} \;\; < x = x * 3; > \, || < y = y * 2; > \;\; \textbf{oc}$$

- 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

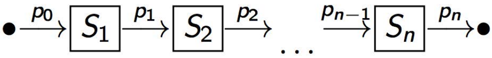
- A *state* of a program consists of the values of the program variables at a point in time, example: $\{x == 2 \land y == 3\}$
- The *state space* of a program is given by the different values that the declared variables can take
- Sequential program: one execution thread operates on its own state space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

Example

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

Executions

- Given the program $S : S_1; S_2; \ldots; S_n;$, 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)

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

Assertions

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 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 $\{x == -1 \wedge y == -1\}$, $\{x == 1 \wedge y == 1\}$, ...

Assertions

- 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

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

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

- Establish program properties by means of a system for formal reasoning
- Help in understanding how a program behaves
- Useful for program construction
- Look at logics for formal analysis

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:

$$\frac{H_1 \quad H_2 \quad \ldots \quad H_n}{C}$$

- 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 variables: $x, y, z, ...$
- Relation symbols: $\leq, \geq, \ldots$
- Function symbols: $+, -, \ldots$, and constants $0, 1, 2, \ldots, \mathit{true}, \mathit{false}$
- Equality: $==$

Formulas in 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 is a formula containing x, the following are formulas:

$\forall x : A(x)$	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

Typical axioms:

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

Typical rules:

$$\frac{A \quad B}{A \wedge B} \qquad \frac{A \quad A \Rightarrow B}{B} \qquad \frac{A}{A \vee B}$$

Example

$$\frac{x == 5 \quad y == 5}{x == 5 \wedge y == 5} \qquad \frac{x \geq 0 \quad x \geq 0 \Rightarrow y \geq 0}{y \geq 0}$$

$$\frac{x == 5}{x == 5 \vee y == 5}$$

Important terms

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

Program Logic (PL)

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

$$\{P\}\ S\ \{Q\}$$

  - S: program statement(s)
  - P and Q: assertions over program states (including $\neg, \wedge, \vee, \exists, \forall$)
  - P: Precondition
  - Q: Postcondition

Example

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

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 postconditions)
- Pre- and postconditions are given as assertions over the program state
- Use PL to find a mathematical argument that the program satisfies its specification

Interpretation of triples is related to code execution

$\{P\}\ S\ \{Q\}$ is *true* if

- the initial state of S satisfies P
- S terminates

then Q is *true* in the final state of S

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

Example

$\{x == y\}$ $\mathrm{x = x * 2; y = y * 2;}$ $\{x == y\}$ is *true*
if the initial state satisfies $x == y$ and the execution terminates, then the final state will satisfy $x == y$

Some *true* formulas:

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

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 assumes termination of $\{P\}\mathtt{S}\{Q\}$, but 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?

$\{P\}\ S;\ \{false\}$	S does not terminate
$\{P\}\ S;\ \{true\}$	does not say much...
$\{true\}\ S;\ \{Q\}$	Q holds after S in any case (provided S terminates)
$\{false\}\ S;\ \{Q\}$	S can not start

Proof system PL

The proof system consists of *axioms* and *rules*

- Axioms for basic statements:
  - `x = e`, **skip**,...
- Rules for composed statements:
  - `S1;S2`, **if**, **while**, **await**, **co...oc**, ...

Theorems in PL

- On triple form
- All axioms are theorems
- The conclusion of a rule is a theorem, given that all the assumptions are theorems:

$$\frac{H_1 \quad H_2 \quad \ldots \quad H_n}{C}$$

Soundness

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

- Example: we want

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

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

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

  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 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
- Variables may be *bound* by quantifiers (then that variable is not free)

Example

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

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?

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

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:

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

Axiom of assignment

Given the postcondition, we may construct the precondition:

Axiom for the assignment statement

$$\{P_{x \leftarrow e}\}\ x = e;\ \{P\}$$

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.

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

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

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

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

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

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

Axiom of skip

The skip statement does nothing

Axiom:

$$\{P\}\ \mathit{skip};\ \{P\}$$

Sequential composition

$$\frac{\{P\}\ S_1;\ \{R\} \quad \{R\}\ S_2;\ \{Q\}}{\{P\}\ S_1; S_2;\ \{Q\}}$$

Conditional

$$\frac{\{P \wedge B\}\ S;\ \{Q\} \quad (P \wedge \neg B) \Rightarrow Q}{\{P\}\ \text{if}\ (B)\ S;\ \{Q\}}$$

- Blue: proof obligations
- **for loop**: exercise 2.22!

Consequence

$$\frac{P' \Rightarrow P \quad \{P\}\ S; \{Q\} \quad Q \Rightarrow Q'}{\{P'\}\ S\ \{Q'\}}$$

while loop

$$\frac{\{I \wedge B\}\ S;\ \{I\}}{\{I\}\ \text{while}\ (B)\ S;\ \{I \wedge \neg B\}}$$

the **while** rule needs a *loop invariant*!

Backward construction over assignments:

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

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

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

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

$\{x == x_0\}\ \texttt{if}\ (\texttt{x} < 0)\ \texttt{x} = -\texttt{x}\ \{x \geq 0 \land (x == x_0 \lor x == -x_0)\}$

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

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

Verification of:

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

$$\frac{\{P \wedge B\}\ S\ \{Q\} \quad (P \wedge \neg B) \Rightarrow Q}{\{P\}\ \text{if}\ (B)\ S\ \{Q\}}$$

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

[And00] Gregory R. Andrews.
Foundations of Multithreaded, Parallel, and Distributed Programming.
Addison-Wesley, 2000.