Axiomatic Proof Techniques for Sequential and Parallel Programs

Jonas Kaiser

UdS, Graduate School of Computer Science

May 2, 2011

Introduction

General Idea

Define axioms and simple rules for the building blocks of our programming language such that the resulting logic is:

- Sound with respect to the semantics of the language.
- Compositional, i.e. assertions that are true for parts of the program can be used to derive assertions about the whole program.

Such a logic can be used to give strong guarantees about program behaviour. Plus it might serve as a form of documentation that is necessarily in sync with the code.

Note that neither of the two papers deals with more elaborate language constructs like jumps, labels, method or function calls and more interesting numeric types.

1969: C. A. R. Hoare

- *An Axiomatic Basis for Computer Programming*
- Considers only sequential programs (no concurrency)
- Origin of the so-called *Hoare Logic*
- Expressions are evaluated without side effects (both in assignments as well as in conditionals)
- Basic rule set very cumbersome to use, author already suggests to formulate derived rules for larger language constructs.
- Fairly practical motivations: reduce development times, reduce bug-fixing and testing costs, better portability and dealing with machine dependence, prevent mission critical failures (apparently including a world war …)

A little While-Language

We reason about programs expressed in the following small language:

$$\begin{array}{rcl} C & ::= & \textbf{skip} \mid x := e \mid C_1; C_2 \mid \\ & & \textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2 \mid \\ & & \textbf{while } B \textbf{ do } C \end{array}$$

Partial Correctness Statements

Notation originally used by Tony Hoare:

$$\vdash P \; \{ \; C \; \} \; Q$$

Conventional notation used today (and in this talk):

$$\vdash \{ \; P \; \} \; C \; \{ \; Q \; \}$$

Here P and Q are logical statements, while C is a program in our basic while-language. The statement should be read as

$$(\rho_b \vDash P) \wedge (C \; \rho_b = \rho_e) \Rightarrow (\rho_e \vDash Q)$$

The Axioms

Skip:

$$\frac{}{\vdash \{\ P\ \}\ \textbf{skip}\ \{\ P\ \}}$$

The Axioms

Skip:

$$\dfrac{}{\vdash \{\ P\ \}\ \textbf{skip}\ \{\ P\ \}}$$

Assignment:

$$\dfrac{}{\vdash \{\ Q[e/x]\ \}\ x := e\ \{\ Q\ \}}$$

The Axioms

Skip:

$$\overline{\vdash \{\ P\ \}\ \textbf{skip}\ \{\ P\ \}}$$

Assignment:

$$\overline{\vdash \{\ Q[e/x]\ \}\ x := e\ \{\ Q\ \}}$$

Remark. There are two common misconceptions of how this rule should look like. Both are unsound since they allow to derive facts which are clearly wrong:

$$\overline{\vdash \{\ P\ \}\ x := e\ \{\ P[x/e]\ \}} \qquad \overline{\vdash \{\ P\ \}\ x := e\ \{\ P[e/x]\ \}}$$

The Axioms

Skip:

$$\frac{}{\vdash \{\ P\ \}\ \textbf{skip}\ \{\ P\ \}}$$

Assignment:

$$\frac{}{\vdash \{\ Q[e/x]\ \}\ x := e\ \{\ Q\ \}}$$

Remark. There are two common misconceptions of how this rule should look like. Both are unsound since they allow to derive facts which are clearly wrong:

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

$$\vdash \{\ X = 0\ \}\ X := 1\ \{\ X = 0\ \}$$

The Axioms

Skip:

$$\frac{}{\vdash \{\ P\ \}\ \textbf{skip}\ \{\ P\ \}}$$

Assignment:

$$\frac{}{\vdash \{\ Q[e/x]\ \}\ x := e\ \{\ Q\ \}}$$

Remark. There are two common misconceptions of how this rule should look like. Both are unsound since they allow to derive facts which are clearly wrong:

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

$$\vdash \{\ X = 0\ \}\ X := 1\ \{\ X = 0\ \} \qquad \vdash \{\ X = 0\ \}\ X := 1\ \{\ 1 = 0\ \}$$

Syntax Rules

Sequencing:

$$\frac{\vdash \{\ P\ \}\ C_1\ \{\ R\ \} \qquad \vdash \{\ R\ \}\ C_2\ \{\ Q\ \}}{\vdash \{\ P\ \}\ C_1; C_2\ \{\ Q\ \}}$$

Syntax Rules

Sequencing:

$$\frac{\vdash \{\ P\ \}\ C_1\ \{\ R\ \} \qquad \vdash \{\ R\ \}\ C_2\ \{\ Q\ \}}{\vdash \{\ P\ \}\ C_1; C_2\ \{\ Q\ \}}$$

Conditional:

$$\frac{\vdash \{\ P \wedge B\ \}\ C_1\ \{\ Q\ \} \qquad \vdash \{\ P \wedge \neg B\ \}\ C_2\ \{\ Q\ \}}{\vdash \{\ P\ \}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \{\ Q\ \}}$$

Syntax Rules

Sequencing:

$$\frac{\vdash \{\, P \,\}\, C_1 \,\{\, R \,\} \qquad \vdash \{\, R \,\}\, C_2 \,\{\, Q \,\}}{\vdash \{\, P \,\}\, C_1; C_2 \,\{\, Q \,\}}$$

Conditional:

$$\frac{\vdash \{\, P \wedge B \,\}\, C_1 \,\{\, Q \,\} \qquad \vdash \{\, P \wedge \neg B \,\}\, C_2 \,\{\, Q \,\}}{\vdash \{\, P \,\}\, \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2 \,\{\, Q \,\}}$$

While:

$$\frac{\vdash \{\, P \wedge B \,\}\, C \,\{\, P \,\}}{\vdash \{\, P \,\}\, \textbf{while}\ B\ \textbf{do}\ C \,\{\, P \wedge \neg B \,\}}$$

Further Rules

Consequence:

$$\frac{\vdash P \Rightarrow P' \qquad \vdash \{\, P' \,\}\, C \,\{\, Q' \,\} \qquad \vdash Q' \Rightarrow Q}{\vdash \{\, P \,\}\, C \,\{\, Q \,\}}$$

Further Rules

Consequence:

$$\frac{\vdash P \Rightarrow P' \qquad \vdash \{\ P'\ \}\ C\ \{\ Q'\ \} \qquad \vdash Q' \Rightarrow Q}{\vdash \{\ P\ \}\ C\ \{\ Q\ \}}$$

Conjunction:

$$\frac{\vdash \{\ P_1\ \}\ C\ \{\ Q_1\ \} \qquad \vdash \{\ P_2\ \}\ C\ \{\ Q_2\ \}}{\vdash \{\ P_1 \wedge P_2\ \}\ C\ \{\ Q_1 \wedge Q_2\ \}}$$

Further Rules

Consequence:

$$\frac{\vdash P \Rightarrow P' \qquad \vdash \{\ P'\ \}\ C\ \{\ Q'\ \} \qquad \vdash Q' \Rightarrow Q}{\vdash \{\ P\ \}\ C\ \{\ Q\ \}}$$

Conjunction:

$$\frac{\vdash \{\ P_1\ \}\ C\ \{\ Q_1\ \} \qquad \vdash \{\ P_2\ \}\ C\ \{\ Q_2\ \}}{\vdash \{\ P_1 \wedge P_2\ \}\ C\ \{\ Q_1 \wedge Q_2\ \}}$$

Disjunction:

$$\frac{\vdash \{\ P_1\ \}\ C\ \{\ Q_1\ \} \qquad \vdash \{\ P_2\ \}\ C\ \{\ Q_2\ \}}{\vdash \{\ P_1 \vee P_2\ \}\ C\ \{\ Q_1 \vee Q_2\ \}}$$

1975: Susan Owicki and David Gries

- *An Axiomatic Proof Technique for Parallel Programs*
- Tries to reason about concurrent programs
- Extends basic While-Language with parallel constructs
- Replaces reasoning about dynamic execution behaviour with effects on static proofs of correctness

Extending the language

To reason about concurrency we need to modify our While-Language, thus:

$$C \quad ::= \quad \ldots \mid \textbf{await } B \textbf{ then } C \mid \\ \textbf{cobegin } C_1 // C_2 // \ldots // C_n \textbf{ coend}$$

Note that in the **await** construct, C may neither contain another **await** nor a **cobegin** construct. Also an **await** construct is atomic, including the evaluation of B.

New Proof Rules

Await:

$$\frac{\vdash \{\ P \wedge B\ \}\ C\ \{\ Q\ \}}{\vdash \{\ P\ \}\ \textbf{await}\ B\ \textbf{then}\ C\ \{\ Q\ \}}$$

New Proof Rules

Await:

$$\frac{\vdash \{\ P \wedge B\ \}\ C\ \{\ Q\ \}}{\vdash \{\ P\ \}\ \textbf{await}\ B\ \textbf{then}\ C\ \{\ Q\ \}}$$

Cobegin:

$$\frac{\vdash \forall i \in [1,n] : \{\ P_i\ \}\ C_i\ \{\ Q_i\ \} \qquad \vdash \textit{Noninterference}}{\vdash \{\ \bigwedge_{1\leq i\leq n} P_i\ \}\ \textbf{cobegin}\ C_1 // \ldots // C_n\ \textbf{coend}\ \{\ \bigwedge_{1\leq i\leq n} Q_i\ \}}$$

Noninterference I

- Assume shared variables (otherwise noninterference holds vacuously)
- Interference with respect to static proofs not dynamic executions
- Key idea: execution of a statement does not invalidate proofs of other code fragments that may run in parallel with the statement in question.
- For this we require invariants of the form:
  $\vdash \{\ P \wedge pre(C)\ \}\ C\ \{\ P\ \}$

Noninterference II

Definition (3.4). Given a proof $\{ P \} C \{ Q \}$ and a statement T with precondition $pre(T)$, we say that T does not interfere with $\{ P \} C \{ Q \}$ if the following two conditions hold:

- $\{ Q \wedge pre(T) \} T \{ Q \}$
- Let C' be any statement within C but not within an **await.** Then $\{ pre(C') \wedge pre(T) \} T \{ pre(C') \}$

Noninterference II

Definition (3.4). Given a proof $\{ P \} C \{ Q \}$ and a statement T with precondition $pre(T)$, we say that T does not interfere with $\{ P \} C \{ Q \}$ if the following two conditions hold:

- $\{ Q \wedge pre(T) \} T \{ Q \}$
- Let C' be any statement within C but not within an **await.** Then $\{ pre(C') \wedge pre(T) \} T \{ pre(C') \}$

Definition (3.5). $\{ P_1 \} C_1 \{ Q_1 \} \ldots \{ P_n \} C_n \{ Q_n \}$ are interference-free if the following holds. Let T be an **await** or assignment statement (which does not appear in an **await**) of process C_i. Then for all $j, j \neq i$, T does not interfere with $\{ P_j \} C_j \{ Q_j \}$

Translating semaphores

Since the **await** construct is too powerful to be used directly by the programmer, the paper provides a translation of semaphores to the augmented While-Language:

- $P(sem) \leadsto$ **await** $sem > 0$ **then** $sem := sem - 1$
- $V(sem) \leadsto$ **await true then** $sem := sem + 1$

Note that the **await** construct provides the atomicity that is required for the semaphore abstraction to work.

Deadlock avoidance

Theorem (6.5). Let S be a statement with proof $\{\ P\ \}\ S\ \{\ Q\ \}$. Let the **await**s of S which do not occur within **cobegin**s of S be

- A_j : **await** B_j **then** $\ldots$

Let the **cobegin**s of S which do not occur within other **cobegin**s of S be

- T_k : **cobegin** $S_1^k // S_2^k // \ldots // S_{n_k}^k$ **coend**

Define

- $D(S) = [\bigvee_j (pre(A_j) \wedge \neg B_j)] \vee [\bigvee_k D_1(T_k)]$
- $D_1(T_k) = [\bigwedge_i (post(S_i^k) \vee D(S_i^k))] \wedge [\bigvee_i D(S_i^k)]$

Then

$D(S) =$ **false** $\wedge\ S$ is a program $\Rightarrow S$ is deadlock free

Proof. *By induction on level of nesting of* ***cobegins****.*

Termination and Total correctness

To ensure that we get proofs of total rather than partial correctness, we need to be able to show termination. A program may fail to terminate properly due to one of the two following cases.

- An infinite loop, i.e. the loop guard is always **true**
- The program deadlocks due to cyclic dependencies between guards

Termination and Total correctness

The first can be fixed by requiring that each loop iteration steps down a well-founded chain. The new **while** rule is then:

$$\frac{\vdash [\ P \wedge B\ ]\ C\ [\ P\ ] \qquad \vdash wdec(P \wedge B, C, t) \qquad \vdash (P \wedge t \leq 0) \Rightarrow \neg B}{\vdash [\ P\ ]\ \textbf{while}\ B\ \textbf{do}\ C\ [\ P \wedge \neg B\ ]}$$

Termination and Total correctness

The first can be fixed by requiring that each loop iteration steps down a well-founded chain. The new **while** rule is then:

$$\frac{\vdash [\ P \wedge B\ ]\ C\ [\ P\ ] \qquad \vdash wdec(P \wedge B, C, t) \qquad \vdash (P \wedge t \leq 0) \Rightarrow \neg B}{\vdash [\ P\ ]\ \textbf{while}\ B\ \textbf{do}\ C\ [\ P \wedge \neg B\ ]}$$

The second is solved by augmenting the **cobegin** rule (using our earlier condition):

$$\frac{\vdash \forall i \in [1,n] : [\ P_i\ ]\ C_i\ [\ Q_i\ ] \qquad \vdash \textit{Noninterference} \qquad \vdash \forall i \in [1,n] : [\ P_i\ ]\ C_i\ [\ Q_i\ ]\ \textit{is deadlock-free}}{\vdash [\ \bigwedge_{1 \leq i \leq n} P_i\ ]\ \textbf{cobegin}\ C_1 // \ldots // C_n\ \textbf{coend}\ [\ \bigwedge_{1 \leq i \leq n} Q_i\ ]}$$

Termination and Total correctness

The first can be fixed by requiring that each loop iteration steps down a well-founded chain. The new **while** rule is then:

$$\frac{\vdash [\ P \wedge B\ ]\ C\ [\ P\ ] \qquad \vdash wdec(P \wedge B, C, t) \qquad \vdash (P \wedge t \leq 0) \Rightarrow \neg B}{\vdash [\ P\ ]\ \textbf{while}\ B\ \textbf{do}\ C\ [\ P \wedge \neg B\ ]}$$

The second is solved by augmenting the **cobegin** rule (using our earlier condition):

$$\frac{\vdash \forall i \in [1,n] : [\ P_i\ ]\ C_i\ [\ Q_i\ ] \qquad \vdash \textit{Noninterference} \qquad \vdash \forall i \in [1,n] : [\ P_i\ ]\ C_i\ [\ Q_i\ ]\ \textit{is deadlock-free}}{\vdash [\ \bigwedge_{1 \leq i \leq n} P_i\ ]\ \textbf{cobegin}\ C_1 // \ldots // C_n\ \textbf{coend}\ [\ \bigwedge_{1 \leq i \leq n} Q_i\ ]}$$

All other rules carry over unchanged (just replace $\{\}$ with $[]$).

Thank you for listening

Questions?