Hoare calculi for parallel programs

Jens Frömmer

University of Kaiserslautern, Embedded Systems Group

froemmer@rhrk.uni-kl.de

8. September 2015

Overview

Correctness proofs for parallel programs

- Parallel computing
    - Performance
    - Efficiency
- Correctness proofs
    - Tests are incomplete
    - Complexity

Hoare Calculus

- Significant impact
- One of the first
- Basis for a variety of approaches

Overview of approaches based on the Hoare Calculus

- Common ground
- Similar intentions
- Based upon each other

Basis

1. Hoare Calculus
2. Towards a theory of parallel programming

Hoare Calculus

Notation

$$\frac{a}{b}$$

means if a is true then b is true, too.

Hoare Calculus

Hoare triple

The relation between a program Q, a precondition P and the result R of the program's execution build a *Hoare triple*:
$P\{Q\}R$
It is also denoted as:
$\{P\}S\{Q\}$
where P is the precondition and Q the postcondition of statement S.

Hoare Calculus

Axiom of Assignment

$$\overline{\{P[E\backslash x]\}x := E\{P\}}$$

Rules of Consequence

$$\frac{\{P\}S\{Q\} \qquad Q \supset R}{\{P\}S\{R\}} \qquad \frac{\{P\}S\{Q\} \qquad R \supset P}{\{R\}S\{Q\}}$$

Rule of Composition

$$\frac{\{P\}S_1\{Q_1\} \qquad \{Q_1\}S_2\{Q\}}{\{P\}S_1; S_2\{Q\}}$$

Hoare Calculus

Rule of Iteration

$$\frac{\{P\&B\}S\{P\}}{\{P\}\ \text{while}\ B\ \text{do}\ S\ \{P\&\neg B\}}$$

Rule of Nothing

$$\frac{}{\{P\}\ \text{skip}\ \{P\}}$$

Rule of Alternation

$$\frac{\{P\&B\}S_1\{Q\} \qquad \{P\&\neg B\}S_2\{Q\}}{\{P\}\ \text{if}\ B\ \text{then}\ S_1\ \text{else}\ S_2\ \{Q\}}$$

Towards a theory of parallel programming

- Arbitrary interleaving of **single units of action**
  - E.g. disjoint processes' instructions
- A critical region C is a **single unit of action**
  1. Exclusive access to shared resource
  2. Execution of C
  3. Free shared resource
- Limited resources
  - Arbitrary initial values
  - Final values are lost
- Shared resources for communication
  - Changes to shared resources have to be visible

Towards a theory of parallel programming

Parallel statement

$\{Q_1 \| Q_2 \| \ldots \| Q_n\}$

Shared resource

$\{\text{resource } r;\ \ Q_1 \| Q_2 \| \ldots \| Q_n\}$

Critical region

with r do C

Critical region with condition

with r when B do C

Towards a theory of parallel programming

Rule for Simultaneity

$$\frac{\text{r inv I},\ P_1\{Q_1\}R_1,\ P_2\{Q_2\}R_2,\ \ldots,\ P_n\{Q_n\}R_n}{I\&P_1\&\ldots\&P_n\ \{\text{resource r; } Q_1//\ldots//Q_n\}I\&R_1\&\ldots\&R_n}$$

Rule for Criticality

$$\frac{\text{r inv I},\ B\&I\&P\{C\}R\&I}{P\{\text{with r when B do C}\}R}$$

Proof techniques

1. Parallel Programming: An Axiomatic Approach
2. An Axiomatic Proof Technique for Parallel Programs I
3. The 'Hoare Logic' of Concurrent Programs
4. A Generalization of Owicki-Gries's Hoare Logic for a Concurrent While Language
5. Owicki-Gries Reasoning for Weak Memory Models

Parallel Programming: An Axiomatic Approach

- Obtain a shared resource by declaration
  - Use existing scope rules
  - Blocks claiming shared resources are single units of action
- Cooperating processes
  - Commutativity principle
- Communicating processes
  - Semi-commutativity
- Colluding processes
  - Possibly non-terminating head
  - Protected tail

Parallel Programming: An Axiomatic Approach

Relation

$Q_1 \sqsubseteq Q_2$ means Q_1 has the identical effects as Q_2 if Q_1 terminates
$Q_1 \equiv Q_2$ means $(Q_1 \sqsubseteq Q_2) \& (Q_2 \sqsubseteq Q_1)$

Disjoint processes

$Q_1 // Q_2 \equiv Q_1; Q_2$

Asymmetric parallel rule

$$\frac{P\{Q_1\}S \qquad S\{Q_2\}R}{P\{Q_1//Q_2\}R}$$

Symmetric parallel rule

$$\frac{P_1\{Q_1\}R_1 \qquad P_2\{Q_2\}R_2}{P_1 \& P_2\{Q_1//Q_2\}R_1 \& R_2}$$

Parallel Programming: An Axiomatic Approach

Semi-commute

Given action q_1 of Q_1 and action q_2 of Q_2, these actions are semi-commute if $q_2; q_1 \sqsubseteq q_1; q_2$

Communicating processes

If all q_1 and q_2 are semi-commute then Q_1 is a producer for the consumer Q_2 and the two processes are communicating.

Rule of two-way communication

$$\frac{P_1 \& S_2 \{Q_1\} S_1 \& R_1 \qquad P_2 \& S_1 \{Q_2\} S_2 \& R_2}{P_1 \& P_2 \{Q_1 // Q_2\} R_1 \& R_2}$$

Parallel Programming: An Axiomatic Approach

Colluding processes

$$\frac{P_1\{Q_1\}R_1 \qquad P_2\{Q_2\}R_2}{P_1\&P_2\{Q_1 \underline{\text{or}}\ Q_2\}R_1 \vee R_2}$$

Possibly non-terminating head and protected tail

$$\frac{\begin{array}{cc} P_1\{Q_1\}R_1 & R_1\{Q'_1\}R \\ P_2\{Q_2\}R_2 & R_2\{Q'_2\}R \end{array}}{P_1\&P_2\{Q_1\ \underline{\text{then}}\ Q'_1\ \underline{\text{or}}\ Q_2\ \underline{\text{then}}\ Q'_2\}R}$$

An Axiomatic Proof Technique for Parallel Programs I

- Based on *Towards a Theory of Parallel Programming*
- Different from *Parallel Programming: An Axiomatic Approach*
    - E.g. globally defined variables, idea of **interference-free** processes

An Axiomatic Proof Technique for Parallel Programs I

Rule of Await

$$\frac{\{P\&B\}S\{Q\}}{\{P\}\text{ await } B \text{ then } S\{Q\}}$$

Rule of Cobegin

$$\frac{\{P_1\}S_1\{Q_1\},\ \ldots,\ \{P_n\}S_n\{Q_n\}\text{ are interference-free}}{\{P_1\&\ldots\&P_n\}\text{cobegin } S_1//\ldots//S_n \text{ coend}\{Q_1\&\ldots\&Q_n\}}$$

An Axiomatic Proof Technique for Parallel Programs I

Interference-free (1)

Statement T with precondition $pre(T)$ does not interfere with $\{P\}S\{Q\}$ if:
$\{Q \& pre(T)\}T\{Q\}$ and
if S' in S then $\{pre(S') \& pre(T)\}T\{pre(S')\}$

Interference-free (2)

$\{P_1\}S_1\{Q_1\}, \{P_2\}S_2\{Q_2\}, \ldots, \{P_n\}S_n\{Q_n\}$ are interference-free if:
For all await or assignment statements T of process S_i, T does not interfere with $\{P_j\}S_j\{Q_j\}$ for all $j \neq i$.

An Axiomatic Proof Technique for Parallel Programs I

Reasons for non-termination:

1. Deadlock
2. Infinite Loop

An Axiomatic Proof Technique for Parallel Programs I

Deadlock-free

Let S be a statement with proof $\{P\}S\{Q\}$.
Let the awaits of S which do not occur within *cobegins* of S be A_j : await B_j then ...
Let the *cobegins* of S which do not occur within other *cobegins* of S be T_k : cobegin $S_1^k // S_2^k // \ldots // S_{n_k}^k$ coend

$$D(S) = \left[\bigvee_j (pre(A_j) \wedge \neg B_j)\right] \vee \left[\bigvee_k D_1(T_k)\right]$$

$$D_1(T_k) = \left[\bigwedge_i (post(S_i^k) \vee D(S_i^k))\right] \wedge \left[\bigvee_i D(S_i^k)\right]$$

Then $D(S) = false$ implies that in no execution of S can S be blocked.

An Axiomatic Proof Technique for Parallel Programs I

Rule of Iteration with Termination

$$\frac{\{P\&B\}S\{P\},\ t >= 0,\ \{P\&B\ \&t = c\}S\{t < c\}}{\{P\}\text{ while } B \text{ do } S\{P\&\neg B\}}$$

Adaption of interference-free (1)

A statement T with precondition $pre(T)$ does not interfere with $\{P\}S\{Q\}$ if:
$\{Q\&pre(T)\}T\{Q\}$ and
if S' in S then $\{pre(S')\&pre(T)\}T\{pre(S'\}$ and
if t is the integer function used in a proof of correctness of a loop within S, then $\{t = c\&pre(T)\}T\{t \leq c\}$

An Axiomatic Proof Technique for Parallel Programs I

Adapted Rule of Cobegin

$$\frac{\{P_1\}S_1\{Q_1\},\ \ldots,\ \{P_n\}S_n\{Q_n\} \text{ are interference-free} \quad \{P_1\}S_1\{Q_1\},\ \ldots,\ \{P_n\}S_n\{Q_n\} \text{ are deadlock-free}}{\{P_1\&\ldots\&P_n\}\text{cobegin } S_1//\ldots//S_n \text{ coend}\{Q_1\&\ldots\&Q_n\}}$$

allows to prove termination!

The 'Hoare Logic' of Concurrent Programs

- Includes proof method of Owicki and Gries

Redefined Hoare triple

The meaning of the triple $\{P\}S\{Q\}$ is redefined. If P is true and execution starts at any point in S then P stays true until S terminates and Q becomes true on termination of S.

Atomic action

$\langle x := x + 1\rangle$

Control locations

$\langle x\rangle := \langle x + 1\rangle$

The 'Hoare Logic' of Concurrent Programs

Value-variables

- Uniquely identifier for each occurrence of a statement or expression
- Class of *value*-variables

$$\langle e \rangle \rightarrow \langle value('\langle e \rangle') := e \rangle$$

Hoare triple

$$\frac{P\{S\}Q}{\{P\}\langle S \rangle\{Q\}}$$

Rule of Consequence

$$\frac{\{P\}S\{Q\},\ Q \supset R}{\{P\}S\{R\}}$$

(weakening postcondition)

The 'Hoare Logic' of Concurrent Programs

Program locations

$at('S') = true$ iff. control is at the beginning of S
$in('S') = true$ iff. control is somewhere in S
$after('S') = true$ iff. control is at the point immediately following S

Rule of Composition

$$\frac{\{P\}S\{Q\} \qquad \{R\}T\{U\} \qquad Q \wedge at('T') \supset R}{\{[in('S') \supset P] \wedge [in('T') \supset R]\}\,[S;T]\,\{U\}}$$

$'[S;T]' = \ 'S' \oplus' T'$
$at('[S;T]') \equiv at('S')$
$after('[S;T]') \equiv after('T')$
$after('S') \equiv at('T')$

The 'Hoare Logic' of Concurrent Programs

Rule of Iteration

$$\frac{\{P\}b\{Q\} \qquad \{R\}S\{P\} \qquad [Q \wedge at('S') \wedge value('b') = true \supset R]}{\{[in('b') \supset P] \wedge [in('S') \supset R]\}\text{while } b \text{ do } S\{Q \wedge value('b') = false\}}$$

$'W' =' b' \oplus' S'$

$at('W') \equiv at('b')$

$after('b') \equiv at('S') \oplus after('W')$

$after('S') \equiv at('b')$

The 'Hoare Logic' of Concurrent Programs

Rule of Cobegin

B denotes **cobegin** $S_1 \| \ldots \| S_n$ **coend**

$$\frac{\{P\}S_1\{P\}, \ldots, \{P\}S_n\{P\}}{\{P\}B\{P\}}$$

$in('B') \equiv [[in('S_1') \vee after('S_1')] \wedge \ldots \wedge [int('S_n') \vee after('S_n')] \wedge \neg [after('S_1') \wedge \ldots \wedge after('S_n')]]$

$at('B') \equiv at('S_1') \wedge \ldots \wedge at('S_n')$

$after('B') \equiv after('S_1') \wedge \ldots \wedge after('S_n')$

$\forall i :' S_i' \textit{ part of } 'B'$

$\forall i \neq j :' S_i' \|' S_j'$

The 'Hoare Logic' of Concurrent Programs

Safety property

A safety property for a program S demands that a certain predicate Q is always *true*. Given that Q is *true* after S terminates, Q is always *true* if some predicate P satisfies:

$$\text{The initial condition implies that } P \text{ is } \mathit{true}. \quad (1)$$

$$\{P\}S\{\mathit{true}\} \quad (2)$$

$$P \supset Q \quad (3)$$

The proof of the second property requires to prove that each sub-statement S_i of **cobegin**-statement S is correct and to prove $\forall j \neq i : \{P_i \wedge P_j\}S_i\{\mathit{true}\}$

A Generalization of Owicki-Gries's Hoare Logic for a Concurrent While Language

- Interpret program by its potential computations
  - Actions of environment
  - Actions of program
  - Invariant properties

Labelled transition relation

$$\langle p, s \rangle \xrightarrow{l} \langle q, s' \rangle$$

A Generalization of Owicki-Gries's Hoare Logic for a Concurrent While Language

Relations describing the programming language's meaning

$\langle p, s \rangle \xrightarrow{E} \langle p, s' \rangle$

$\langle x := t, s \rangle \xrightarrow{P} \langle \varepsilon, s\,[t/x] \rangle$

...

$\langle \text{await } D \text{ then } p, s \rangle \xrightarrow{P} \langle \varepsilon, s' \rangle$ if $s \models D$ and $p = \varepsilon$ or $\exists n \geq 1$.
$\forall i$ with $1 \leq i \leq n . p_0 = p, p_n = \varepsilon, s_0 = s, s_n = s'$ implies
$\langle p_{i-1}, s_{i-1} \rangle \xrightarrow{P} \langle p_i, s_i \rangle$

...

$\langle p \| q, s \rangle \xrightarrow{P} \langle p' \| q, s' \rangle$ if $\langle p, s \rangle \xrightarrow{P} \langle p', s' \rangle$

$\langle p \| q, s \rangle \xrightarrow{P} \langle p \| q', s' \rangle$ if $\langle q, s \rangle \xrightarrow{P} \langle q', s' \rangle$

A Generalization of Owicki-Gries's Hoare Logic for a Concurrent While Language

Potential computation

A potential computation (pc) from p_0 is any finite or infinite sequence $\langle p_i, s_i\rangle \xrightarrow{l_i} \langle p_{i+1}, s_{i+1}\rangle$ for each defined i, $l_i \in \{P, E\}$

A Generalization of Owicki-Gries's Hoare Logic for a Concurrent While Language

Redefined Hoare triple

$\{\Gamma, A\}p\{B, \Delta\}$ with
$\models \{\Gamma, A\}p\{B, \Delta\}$ iff $E[\Gamma] \cap O[A] \cap [\![P]\!] \subseteq \Lambda[B] \cap P[\Delta]$ where
$E[\Gamma]$ set of all pcs which only include Γ-invariant environment changes
$P[\Gamma]$ set of all pcs which only include Γ-invariant program changes
$O[A]$ set of all pcs which initial state satisfy A
$\Lambda[A]$ set of all pcs that either do not terminate or satisfy A in the first termination state
$[\![p]\!]$ set of all actual computations of p

A Generalization of Owicki-Gries's Hoare Logic for a Concurrent While Language

Rule of $\|$

$$\frac{\Gamma \rightarrow C \qquad \{\Gamma, A\}p\{C, \Sigma \cup \Delta\} \qquad \{\Sigma, B\}q\{E, \Gamma \cup \Delta\} \qquad \Sigma \rightarrow E}{\{\Gamma \cup \Sigma, A \wedge B\}p\|q\{C \wedge E, \Delta\}}$$

- Setting $\Gamma = L$ and $\Delta = \emptyset$ one obtains the usual Hoare rules
- Rule of $\|$ ensures that the environment invariants of the one are program invariants of the other proof
- Freedom of interference in the sense of Owicki-Gries

Owicki-Gries Reasoning for Weak Memory Models

- Stronger definition of non-interference that is sound under C11's release/acquire and TSO
- Program's semantic is given by its set of consistent executions

Execution

An execution G is a triple $\langle A, L, E \rangle$ where A is a finite set of nodes that does identify G, L labels each node and E is a set of edges.

Owicki-Gries Reasoning for Weak Memory Models

Labels

$\langle S \rangle$ (skip), $\langle R, x, v_r \rangle$ (read), $\langle W, x, v_w \rangle$ (write),
$\langle U, x, v_r, v_w \rangle$ (update)

Edges

For every triple $\langle a, b, x \rangle \in E \subseteq (A \times A) \cup (A \times A \times Loc)$ there is
$a \in G.W_x \cup G.U_x$, $b \in G.S \cup G.R_x \cup G.U_x$ and
$G.val_w(a) = G.val_r(b)$ (for $b \notin G.S$).

Owicki-Gries Reasoning for Weak Memory Models

Composition

If $G = \langle A, L, E \rangle$ and $G' = \langle A', L', E' \rangle$ are two executions with disjoint sets of nodes then

$$G \| G' \text{ is given by } \langle A \cup A', L \cup L', E \cup E' \rangle$$
$$G; G' \text{ is given by } (G \| G') \cup (O \times I)$$

with the terminal nodes O of G and the initial nodes I of G'

Owicki-Gries Reasoning for Weak Memory Models

Gadgets

A *read gadget* is an execution like $\langle \{a\}, \{a \mapsto \langle R, x, v\rangle\}, \emptyset\rangle$. *write*, *update* and *skip gadgets* are defined in the same way. $\mathcal{RG}(x, v)$, $\mathcal{WG}(x, v)$, $\mathcal{UG}(x, v_r, v_w)$ and $\mathcal{SG}$ denote the sets of all *read*, *write*, *update* and *skip gadgets*.

Map instructions to sets of executions

$[\![\text{skip}]\!] = \mathcal{SG}$
$[\![\text{if } e(x) \text{ then } c_1 \text{ else } c_2]\!] = \bigcup\{\mathcal{RG}(x, v); [\![c_i]\!] | \text{value } v, i \in \{1, 2\}, [\![e]\!](v) = 0 \text{ iff } i = 2\}$
...

Owicki-Gries Reasoning for Weak Memory Models

Consistent executions

An execution $G = \langle A, L, E \rangle$ is complete if for every read or update there is a previous write or update. It is coherent if E_{all} is acyclic and there is a modification order for each location. It is consistent if it becomes complete and consistent by adding some edges.

Redefined Hoare triple

A Hoare triple $\{P\}c\{Q\}$ is valid if Q holds at the terminal edge of complete and coherent execution $G \cup E'$ in $G \cup E'$ for every execution G in $\mathcal{WG}(P); [\![c]\!]; \mathcal{SG}$ where $\mathcal{WG}(P)$ denotes all possible initializations with respect to P.

Owicki-Gries Reasoning for Weak Memory Models

Owicki-Gries judgement

$\mathcal{R};\mathcal{G} \models \{P\}c\{Q\}$ where the rely component $\mathcal{R}$ consists of assertions that are required to be stable under parallel execution. The guarantee component $\mathcal{G}$ consists of guarded assignments.

Rule of Parallel

$$\frac{\mathcal{R}_1;\mathcal{G}_1 \models \{P_1\}c_1\{Q_1\} \qquad \mathcal{R}_2;\mathcal{G}_2 \models \{P_2\}c_2\{Q_2\} \qquad Q_1 \wedge Q_2 \vdash Q \qquad \mathcal{R}_1;\mathcal{G}_1 \text{ and } \mathcal{R}_2;\mathcal{G}_2 \text{ are non-interfering}}{\mathcal{R}_1 \cup \mathcal{R}_2 \cup \{Q \nearrow (\mathcal{R}_1^R \vee \mathcal{R}_2^R \vee Q)\};\mathcal{G}_1 \cup \mathcal{G}_2 \models \{P_1 \wedge P_2\}c_1 \| c_2\{Q\}}$$

Owicki-Gries Reasoning for Weak Memory Models

Non-interfering

$\mathcal{R}_1; \mathcal{G}_1$ and $\mathcal{R}_2; \mathcal{G}_2$ are non-interfering if every $R \nearrow C \in \mathcal{R}_i$ is stable under every $\{P\}c \in \mathcal{G}_j$ for $i \neq j$. If applied to atomic assignments or assignments of values, the new non-interference check coincides with the one of Owicki and Gries.

Conclusions

- Hoare's proposal in *Parallel Programming: An Axiomatic Approach* is not of importance for later ones
- Owicki-Gries's system is either the basis of or included in the other reviewed approaches
- Crucial point: non-interference
- Owicki-Gries's rule for the parallel-statement is considered to be non-compositional

Conclusions

- Adding the concept of control locations yields in a system that does include the one of Owicki-Gries, but is compositional
- Stirling's proposal applies Jones' idea of rely and guarantee conditions
- Lahav and Vafeiadis presume a weak memory model instead of sequential consistency

References

Stephen D Brookes, Charles AR Hoare & Andrew W Roscoe (1984)
A theory of communicating sequential processes
Journal of the ACM 31(3), pp. 560–599

CAR Hoare (1976)
Parallel programming: an axiomatic approach
Springer

CAR Hoare (1969)
An axiomatic basis for computer programming
*Communications of the ACM*12(10), pp. 576–580

References

CAR Hoare (2002 (1971))
Towards a theory of parallel programming
The origin of concurrent programming, Springer pp. 231–244

2011
Information technology – Programming languages – C++
International Organization for Standardization, Geneva, CH

Cliff B Jones (1981)
Development methods for computer programs including a notion of interference
Oxford University Computing Laboratory

References

Thomas Kleymann (1999)
Hoare logic and auxiliary variables
Formal Aspects of Computing 11(5), pp. 541–566

Ori Lahav & Viktor Vafeiadis (2015)
Owicki-Gries Reasoning for Weak Memory Models

Leslie Lamport (1980)
The 'Hoare logic'of concurrent programs
Acta Informatica 14(1), pp. 21–37

References

Susan Owicki & David Gries (1976)
An axiomatic proof technique for parallel programs I
Acta informatica 6(4), pp. 319–340

Colin Stirling (1988)
A generalization of Owicki-Gries's Hoare logic for a concurrent while language
Theoretical Computer Science 58(1), pp. 347–359