Towards Concurrent Hoare Logic

November 2, 2012

Supervisor: Professor Tad Takaoka
tad.takaoka@canterbury.ac.nz
Robin Candy
rpc40@uclive.ac.nz
Department of Computer Science and Software Engineering
University of Canterbury, Christchurch, New Zealand

The author would like to thank Professor Tad Takaoka for his kindness and patience.

Abstract

How can we rigorously prove that an algorithm does what we think it does? Logically verifying programs is very important to industry. Floyd-Hoare Logic (or Hoare Logic for short) is a set of rules that describe a type of valid reasoning for sequential program verification. Many different attempts have been made to extend Hoare Logic for concurrent program verification. We combine ideas from a few of these extensions to formalise a verification framework for specific classes of parallel programs. A new proof rule to deal with the semantics of mesh algorithms is proposed within the verification framework. We use the framework and mesh proof rule to verify the correctness of Sung Bae's parallel algorithm for the maximum subarray problem.

Contents

1 Introduction

This chapter presents a brief overview of relevant research into concurrent Hoare Logic, the research objectives and an outline and justification of the research undertaken.

1.1 Background

Floyd-Hoare logic (or Hoare logic for short) is a set of rules that describe valid reasoning in sequential program verification. It was developed in the late 1960's by Floyd[10] (1967) and then by Hoare[11] (1969). The core idea is to formalise the way the state space of a program changes as the program is executed. Hoare had a desire[1] that his work be extended to verifying non-sequential programs too.

Since then Hoare logic has been extended to reason about many programming languages and semantic interpretations. Extensions in popular high level languages have been formalised (Java[14],C++[17] and C#[21]). The main roadblock in the extension of Hoare logic to parallel programs is the combinatorial explosion faced from analysing the order of execution. Hoare[12], Lamport[16] and Milner[19][20] manage this by reasoning about communicating sequential processes (CSP) instead of generalised parallel programs. Owicki and Gries[22] use the idea of critical sections to manage interference and minimise the effects combinatorial explosion. An approach by Stark[23] based on Owiki and Gries's method minimises the potential orders of execution by setting up a rely/guarantee framework in which relationships between processes are formalised. Takaoka[24] uses Hoare logic on generalised parallel programs using directed graphs and assertion matrices.

1.2 Project Objectives

The core objectives of the project are:

- To formalise and modify an existing framework to verify concurrent programs using Hoare logic.
- To formally verify Bae's parallel maximum subarray algorithm[2] using the formal framework.

[1] Hoare expressed the desire that others expand on his verification framework at the end of his celebrated paper:"An axiomatic basis for computer programming"[11].

[2] Bae's parallel maximum subarray algorithm appears as Algorithm 38 in his PhD thesis[3].

The first objective involves both exploring existing framework and developing rules for semantics not covered in the existing framework. The defined framework should be well suited to verifying mesh algorithms (as this is what we will use it for). It would be desirable if the approach were also general enough to be useful in verifying other algorithms.

The second objective is to use the formal framework to prove the partial correctness (correct but may or may not terminate) of Algorithm 1 (Bae's parallel maximum subarray algorithm). The key component of this objective is to treat the algorithm formally, using Hoare logic to reason about the semantics of mesh algorithms.

1.3 Justification

Verification of Bae's parallel maximum subarray algorithm (known also as Algorithm 1 for this thesis) has been previously given in two sources: Bae[3] and Weddell[25]. Bae's verification relied on finding invariants and providing an informal (with regard to program semantics) mathematical argument as to how the solution propagates. This style of verification relies on assumptions of how the program affects variables that are not formally verified. The verification in Weddell[25] is formal with regard to program semantics. It provides a summary[3] of parts of the proof without details. Restrictive assumptions are made about the order of execution in each parallel process (perfect synchronisation is assumed).

A proof of Algorithm 1 based on formal semantics and without restrictive assumptions would be useful in increasing confidence of its correctness. Algorithm 1 is one of the fastest maximum subarray algorithms of its type and scales well[3]. Weddell[25] argues it will be of use in radio and optical astronomy. It is therefore plausible that an increased confidence in its correctness would be useful in an applied setting also.

1.4 Report Outline

To have a formal framework to prove the partial correctness of mesh algorithms, we need to first define sequential Hoare logic. This is done in Chapter 2. We define the programming language used in this thesis and introduce Hoare logic. The axioms and rules for reasoning are also given in this chapter. This chapter is concluded with a discussion of the soundness and completeness of Hoare logic.

In Chapter 3 we formalise an approach for extending Hoare logic to concurrent semantics. The extension is consistent with the framework provided by the Chapter 2. We also discuss a graph theoretical approach to dealing with parallel semantics and how synchronisation can be simulated.

We use Chapter 4 to formalise the syntax and semantics of mesh algorithms. This formality is needed for the partial correctness proof of Alogrithm 1, which is presented at the end of the chapter. Many intermediate results are needed for the verification. We present the proof in a top down manner, with the low level Lemmas appearing at the end of the chapter.

[3]Takaoka has expanded parts the proof since the publication of [25].

We conclude in Chapter 5. The more contentious notation used is summarised in Appendix A.

2 Hoare Logic Framework

Sequential Hoare logic is discussed in this chapter. With the exception of notation, we use the original concepts proposed by Floyd[10] and Hoare[11]. In Section 2.1 we define the programming language used in this thesis. The remaining sections are devoted to introducing Hoare logic. Section 2.2 gives the fundamental structure of Hoare logic and presents two basic results. The axioms and rules for reasoning are given in Section 2.3. In Section 2.4 we present several useful results which we use in subsequent chapters. The chapter is concluded with Section 2.5 in which we discuss the soundness and completeness of Hoare logic.

2.1 Commands

Definition 1. A *command* C is a syntactically correct statement of a programming language that changes the program state when executed.

In this thesis we assume that program state is completely determined by the value of each variable in the program. The underlying logic we use is first-order logic[1] (sometimes called first-order predicate calculus) with an appropriate domain of discourse (usually program variables or integers depending on context). The programming language for this thesis is standard pseudocode `WHILE`-langauge defined below (and by Bergstra[6]).

Definition 2. Let V be a variable and E be a term in first-order logic. An *assignment command*

$$V := E$$

assigns the value of E to the variable V.

Definition 3. Let B be a quantifier free statement in first-order logic. Let C be a command. A *while command*

`WHILE` B `DO` C

checks whether B holds then executes C until B no longer holds. The statement B is known as a *boolean check*.

[1] Blass and Gurevich[7] justify this choice but also advocate exploration of other potential logics.

Definition 4. Let B be a statement in first-order logic. Let C_1 and C_2 be commands. A *strong conditional command*

$$\texttt{IF } B \texttt{ THEN } C_1 \texttt{ ELSE } C_2$$

executes C_1 if B holds and C_2 if B does not.

A *weak conditional command*

$$\texttt{IF } B \texttt{ THEN } C_1$$

executes C_1 if B holds. The statement B is known as a *boolean check*.

Definition 5. Let C_1, C_2 ... C_n be commands. A *composition command*

$$\texttt{BEGIN } C_1; C_2; \ldots C_n; \texttt{ END}$$

sequentially executes the commands $C_1, C_2 \ldots C_n$.

Definition 6. Let C be a command and i, n be some variables not modified by C. A *for loop command*

$$\texttt{FOR } i := 1 \ldots n \texttt{ DO } C$$

sequentially iterates through the assignments $i := 1 \ldots n$ and executes C after each assignment.

In Chapters 3 and 4 we define more commands to deal with concurrency.

2.2 Hoare Triple

Definition 7. Let P and Q be statements in first-order logic. Let C be a command. A *Hoare triple* has syntax[2]

$$\{P\}\ C\ \{Q\}$$

If P holds directly before the execution of C and C finishes executing then Q holds. The first statement P is called the *precondition* and the concluding statement Q the *postcondition*.

The Hoare triple is the core feature of Hoare logic. It allows us to rigorously define the expected behavior of a program. It is important to note that we only define partial correctness of a program; i.e. termination of execution cannot be guaranteed.

Proposition 8. *Let P_1, P_2, Q be statements in first-order logic. Let C be a command. Then*

$$\frac{\{P_1\}\ C\ \{Q\} \qquad \{P_2\}\ C\ \{Q\}}{\{P_1 \vee P_2\}\ C\ \{Q\}}$$

[2]Hoare's[11] original notation for the triple was: $P\ \{C\}\ Q$. This was changed to the current form (presumably) to be more consistent with set theory notation.

Proof. Assume $\{P_1 \vee P_2\}\ C\ \{Q\}$ does not hold. Then there exists a program state such that $P_1 \vee P_2$ holds directly before execution of C and Q does not hold after execution. But $P_1 \vee P_2$ implies P_1 or P_2 holds. Assume, with out loss of generality, P_1 holds. Then there exists a program state such that P_1 holds directly before execution and Q does not hold after execution. Contradicting the premise. Therefore $\{P_1 \vee P_2\}\ C\ \{Q\}$. □

Proposition 9. *Let P_1, P_2, Q be statements in first-order logic. Let C be a command. Then*

$$\frac{\{P\}\ C\ \{Q_1\} \qquad \{P\}\ C\ \{Q_2\}}{\{P\}\ C\ \{Q_1 \wedge Q_2\}}$$

Proof. Directly from the premises and definition 7, if P holds and the program finishes execution then both Q_1 and Q_2 must hold. Hence $\{P\}\ C\ \{Q_1 \wedge Q_2\}$. □

2.3 Hoare Logic Axioms

Let P, P_1, P_2, Q, Q_1, Q_2, B and R be statements in first-order logic. Let V be a variable and E be a term in first-order logic. Let C, C_1, C_2 be commands. With these assignments in mind we consider the Axioms and rules of inference of Hoare logic.

Axiom 1 (Null Rule)**.**

$$\{P\}\ \ \{P\}$$

Axiom 2 (Assignment Rule)**.**

$$\{P[E/V]\}\ V := E\ \{P\}$$

Axiom 3 (Strong Conditional Rule)**.**

$$\frac{\{P \wedge B\}\ C_1\ \{Q\} \qquad \{P \wedge \neg B\}\ C_2\ \{Q\}}{\{P\}\ \ \texttt{IF}\ B\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}}$$

Axiom 4 (Weak Conditional Rule)**.**

$$\frac{\{P \wedge B\}\ C\ \{Q\} \qquad P \wedge \neg B \rightarrow Q}{\{P\}\ \ \texttt{IF}\ B\ \texttt{THEN}\ C\ \{Q\}}$$

Axiom 5 (While Rule)**.**

$$\frac{\{P \wedge B\}\ C\ \{P\}}{\{P\}\ \ \texttt{WHILE}\ B\ \texttt{DO}\ C\ \{P \wedge \neg B\}}$$

Axiom 6 (Compound Rule)**.**

$$\frac{\{P\}\ C_1\ \{R\} \qquad \{R\}\ C_2\ \{Q\}}{\{P\}\ \texttt{BEGIN}\ C_1; C_2\ \texttt{END}\ \{Q\}}$$

Axiom 7 (Precondition Strengthening)**.**

$$\frac{P \to R \qquad \{R\}\ C\ \{Q\}}{\{P\}\ C\ \{Q\}}$$

Axiom 8 (Postcondition Weakening)**.**

$$\frac{\{P\}\ C\ \{R\} \qquad R \to Q}{\{P\}\ C\ \{Q\}}$$

Remark 10. Strictly speaking Axiom 7 and Axiom 8 are not axioms as they can be shown directly from the definition of the Hoare triple. We include them as axioms as this is the typical presentation[16][15][2].

Axiom 9 (For Rule)**.** Let n be a positive integer. Let P_i be a sequence of conditions with $i \in \{0, 1, 2, \ldots n\}$. Then

$$\frac{\forall i \in \{1, 2, 3, \ldots n\}\ \{P_{i-1}\}\ k := i; C\ \{P_i\}}{\{P_0\}\ \texttt{FOR}\ k := 1 \ldots n\ \texttt{DO}\ C\ \{P_n\}}$$

Remark 11. We can give a formal justification of the previous rule. By sequencing rule (Axiom 6), assignment rule (Axiom 2) and premises we have

$$\{P_0\}\ k := 1; C; k := 2; C; \ldots k := n; C\ \{P_n\}$$

. Which is semantically equivalent to

$$\{P_0\}\ \texttt{FOR}\ k := 1 \ldots n\ \texttt{DO}\ C\ \{P_n\}$$

.

For the remaining sections and chapters if it is obvious from the context whether something is a command, statement, variable or expression we omit the formal declaration.

2.4 Basic Properties

Proposition 12 (And Rule)**.** *Let P_1, P_2, Q_1, Q_2 be statements in first-order logic. Let C be a command. Then*

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

Proof. By the first premise and precondition strengthening (Axiom 7) $\{P_1 \wedge P_2\}\ C\ \{Q_1\}$. Also by the second premise and precondition strengthening (Axiom 7) $\{P_1 \wedge P_2\}\ C\ \{Q_2\}$. Hence by Proposition 9

$$\{P_1 \wedge P_2\}\ C\ \{Q_1 \wedge Q_2\}$$

□

Proposition 13 (Or Rule)**.** *Let P_1, P_2, Q_1, Q_2 be statements in first-order logic. Let C be a command. Then*

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

Proof. By the first premise and postcondition weakening (Axiom 8) $\{P_1\}\ C\ \{Q_1 \vee Q_2\}$. Also by the second premise and postcondition weakening (Axiom 8) $\{P_2\}\ C\ \{Q_1 \vee Q_2\}$. Hence by Proposition 8

$$\{P_1 \vee P_2\}\ C\ \{Q_1 \vee Q_2\}$$

□

Proposition 14. *Let a command C be*

BEGIN
$M := E_1$;
IF $E_2 > M$ *DO* $M := E_2$;
IF $E_3 > M$ *DO* $M := E_3$;
...
IF $E_n > M$ *DO* $M := E_n$;
END

Then

$$\{True\}\ C\ \{M = \ \mathit{MAX}_{E \in \{E_1, E_2, \ldots E_n\}}\ E\}$$

Proof. When $n = 1$, by the assignment rule (Axiom 2)

$$\{M = M\}\ C\ \{M = E1\}$$

and the result holds.

Assume

$$\{True\}\ C\ \{M = \ \mathtt{MAX}_{E \in \{E_1, E_2, \ldots E_k\}}\ E\}$$

holds.

When $n = k{+}1$ if $E_{k+1} < M$ and $M = \ \mathtt{MAX}_{E \in \{E_1, E_2, \ldots E_k\}}\ E$ then $M = \ \mathtt{MAX}_{E \in \{E_1, E_2, \ldots E_{k+1}\}}\ E$. Furthermore

$$\{E_{k+1} = \ \mathtt{MAX}_{E \in \{E_1, E_2, \ldots E_{k+1}\}}\ E\}\ M := E_{k+1};\ \{M = \ \mathtt{MAX}_{E \in \{E_1, E_2, \ldots E_{k+1}\}}\ E\}$$

by the assignment rule (Axiom 2).

$$M = \ \mathtt{MAX}_{E \in \{E_1, E_2, \ldots E_k\}}\ E \wedge E_{k+1} > M \rightarrow E_{k+1} = \ \mathtt{MAX}_{E \in \{E_1, E_2, \ldots E_k + 1\}}\ E$$

so by previous result and precondition strengthening (Axiom 7)

$$\{M = \texttt{MAX}_{E\in\{E_1,E_2,\ldots E_k\}} E \wedge E_{k+1} > M\}\ M := E_{k+1};\ \{M = \texttt{MAX}_{E\in\{E_1,E_2,\ldots E_{k+1}\}} E\}$$

Hence by the weak conditional rule (Axiom 4)

$$\{M = \texttt{MAX}_{E\in\{E_1,E_2,\ldots E_k\}} E\}\ C\ \{M = \texttt{MAX}_{E\in\{E_1,E_2,\ldots E_{k+1}\}} E\}$$

and by the sequencing rule (Axiom 6) and induction hypothesis

$$\{True\}\ C\ \{M = \texttt{MAX}_{E\in\{E_1,E_2,\ldots E_n\}} E\}$$

□

Corollary 15. *Let a command C be*

BEGIN
$m := E_1$;
IF $E_2 < m$ *DO* $m := E_2$;
IF $E_3 < m$ *DO* $m := E_3$;
...
IF $E_n < m$ *DO* $m := E_n$;
END

Then

$$\{True\}\ C\ \{m = \textit{MIN}_{E\in\{E_1,E_2,\ldots E_n\}} E\}$$

The proof of Corollary 15 is omitted as it is effectively the same as the proof of Proposition 14.

Remark 16. In both Proposition 14 and Corollary 15 we may omit the first lines of the commands and replace E_1 with the variable in question (M or m respectively).

2.5 Soundness and Completeness

A logic is given credibility if it can never prove anything that is not true and can prove everything that is true (within the domain the logic is defined for).

Definition 17. A Hoare logic[3] is *sound* if every Hoare triple that can be proven is true (by semantic meaning of a Hoare Triple).

Definition 18. A Hoare logic is *complete* if every Hoare triple that is true can be proven.

[3]We say *a* Hoare logic as opposed to just Hoare logic. We do this as the choices of programming language, underlying logic and Hoare style rules can make Hoare Logics behave very differently from eachother.

In 1978 Cook[8] proved the completeness and soundness of a typical Hoare logic. A sound and complete underlying logic and a ALGOL-like language was used for the proof. Since then Apt[2][1] has summarised soundness and completeness of many other Hoare logics (including nodeterministic Hoare logic).

A Hoare logic cannot be complete if the underlying logic is incomplete. This is very relevant as Peano arithmetic and ZFC are the main ways we reason about integers and are incomplete.

3 Parallel Hoare Logic

In this chapter we formalise an approach for extending Hoare logic to concurrent semantics (Section 3.1). The extension is consistent with the framework provided by the previous chapter. In Section 3.2 we informally discuss how synchronisation of parallel processes can be simulated with no specific synchronisation semantics. We present a simplified version of the graph theoretic approach of Takaoka[24] in 3.3. Section 3.4 is concerned with other approaches to parallel extensions of Hoare logic and the way they interact with the graph theoretical approach.

3.1 Preliminaries

In this thesis we consider concurrent execution as multiple commands simultaneously executing. We make no assumptions about the relative order in which they execute. We do however assume that each assignment and boolean check in each command is atomic. This means no variable can simultaneously be assigned a value and referenced by any command. Except for local or auxiliary variables, all variables are assessable from any concurrently executing command.

Definition 19. Let C be a command. A *local variable* V in C is a variable such that no other command (outside C) may reference V. We refer to C as the territory for the local variable V.

Remark 20. In light of Definition 19 we may even have two or more local variables in a program state with the same label. To quash ambiguity we must define when we are using such local variables and ensure none have overlapping territory.

Definition 21. Let $\{C_1, C_2 \dots C_n\}$ be a set of commands. A *cobegin command*

$$\texttt{COBEGIN } C_1||C_2 \dots ||C_n \texttt{ COEND}$$

concurrently executes each member of $\{C_1, C_2 \dots C_n\}$. The cobegin command does not stop execution unless(/until) each process terminates. The set of commands $\{C_1, C_2 \dots C_n\}$ are the *concurrent commands* of the cobegin command.

Remark 22. In Definition 21 all the subscripts on the commands (including n) are **not** program variables. They are therefore not accessible from the program. They are simply part of the labels for the concurrent commands.

Definition 23. Let C be a cobegin command. A *shared variable* V in C is a variable such that V is assigned a value in a concurrent command of C and V is referenced (i.e. assigned or read) by at least one **other** concurrent command of C.

Definition 24. Let C be a cobegin command. An *interference point* of C is a boolean check or an assignment statement that uses a shared variable.

Remark 25. Without interference reasoning about parallel programs is very similar to reasoning about sequential programs. It is therefore a key goal of concurrent Hoare logic to manage interference to simplify the verification process.

3.2 Synchronisation

We don't support specific synchronisation procedures in the pseudocode `WHILE` -langauge we present for this thesis. However, a slight synchronisation can be simulated using multiple cobegin statements.

Example 26. Consider the following `COBEGIN` command:

```
                COBEGIN
C_mix           C_mix           C_mix
C_bake    ||    C_mix     ||    C_wait
C_wait          C_ice           C_eat
C_eat           C_eat
                COEND
```

Suppose we want to ensure all concurrent commands wait for each other to finish C_{mix} before continuing. Suppose also that we wish C_{bake} to finish execution before C_{ice} begins execution. Lastly, suppose C_{eat} should not begin execution untill all other processes have finished. Instead of using formal synchronisation procedures for the concurrent commands, we may rewrite the cobegin command as:

$$
\begin{array}{lll}
 & \texttt{COBEGIN} & \\
C_{mix} \quad || \quad & C_{mix} \quad || & \quad C_{mix} \\
 & C_{mix} & \\
 & \texttt{COEND} &
\end{array}
$$

$$
\begin{array}{lll}
 & \texttt{COBEGIN} & \\
C_{bake} & || & C_{wait} \\
 & \texttt{COEND} &
\end{array}
$$

$$
\begin{array}{lll}
 & \texttt{COBEGIN} & \\
C_{wait} & || & C_{ice} \\
 & \texttt{COEND} &
\end{array}
$$

$$
\begin{array}{lll}
 & \texttt{COBEGIN} & \\
C_{eat} \quad || \quad & C_{eat} \quad || & \quad C_{eat} \\
 & \texttt{COEND} &
\end{array}
$$

3.3 Program Graphs

A more formal treatment of the concepts in this section can be found in Takaoka[24].

Definition 27. Let C be the `COBEGIN` command:

$$\texttt{COBEGIN } C_1||C_2 \ldots ||C_n \texttt{ COEND}$$

Let $A = \{a_1 \ldots a_n\}$ and $B = \{b_1 \ldots b_n\}$ be sets of control points for $C_1 \ldots C_n$. We say A is *connected* to B if there exists an atomic action (assignment, boolean check) in a C_i such that:

- Directly before the execution of the action each C_i is at control point a_i.
- Directly before the execution of the action each C_i is at control point b_i.

We denote A being connected to B by: $\mathfrak{C}(A, B)$.

Definition 28. Let C be the `COBEGIN` command:

$$\texttt{COBEGIN } C_1||C_2 \ldots ||C_n \texttt{ COEND}$$

Let A_i be the set of control points (any gap between atomic statements) in each C_i. A *program graph* $G = (V, E)$ is a directed graph with vertex set V and edge set E such that:

$$V = A_1 \times A_2 \times \ldots A_n$$

$$E = \{(A, B) \in V^2 : \mathfrak{C}(A, B)\}$$

Remark 29. A program graph's vertices may be thought of as the program's current point of execution. Edges may be thought of as the atomic actions that, when executed, can progress a control point. A path from the initial set of control points to the final set of control points can be thought of as an execution of the whole program.

Definition 30. Let C be the `COBEGIN` command:

$$\texttt{COBEGIN } C_1||C_2 \ldots ||C_n \texttt{ COEND}$$

Let G be a program graph for C. We associate an condition P_α to each vertex $\alpha = \{a_1, a_2 \ldots a_n\}$ in G. For each edge e in G let C_e be the action associated with e. For a given set of vertex conditions G is a *proof graph* for C if for each edge $e = (\alpha, \beta)$ in G:

$$\{P_\alpha\}\ C_e\ \{P_\beta\}$$

Remark 31. For a boolean check B, the edge failing that check gets associated with the action $\neg B$.

Remark 32. For Definition 30 we have to have a way of verifying Hoare triples for boolean checks. As boolean checks make no changes to the program state space we treat them in the underlying logic. This means if B is a boolean check:

$$\{P\}\ B\ \{Q\} \Leftrightarrow P \wedge B \rightarrow Q$$

Remark 33. If C is a command and G is a graph associated with C. Suppose G is a provable graph and the condition P is associated with the initial control points. Suppose also, that the condition Q is associated with the final control points. Then

$$\{P\}\ C\ \{Q\}$$

That is if we have a provable graph then we have partial correctness. We are not equipped with the preliminaries to prove this result so we instead treat it as an informal axiom.

Example 34. Let C be the following `COBEGIN` command:

```
                COBEGIN
WHILE x < y     DO      ||      WHILE x < y DO
  x := x + 1;                     y := y - 1;
                COEND
```

A proof graph for C is given in Figure 3.1

3.4 Program Graphs Revisited

There has been a considerable amount of research into proving concurrent program correctness by reasoning about interference. Owicki and Gries[22] use critical sections to manage interference. They require that each variables which may cause interference be declared at the start of each critical section. Lamport[16] and Milner[19][20] use communication between local processors to minimise interference. We may combine these ideas with the graphical approach from the previous section. We do this by only considering interference points instead of all control points.

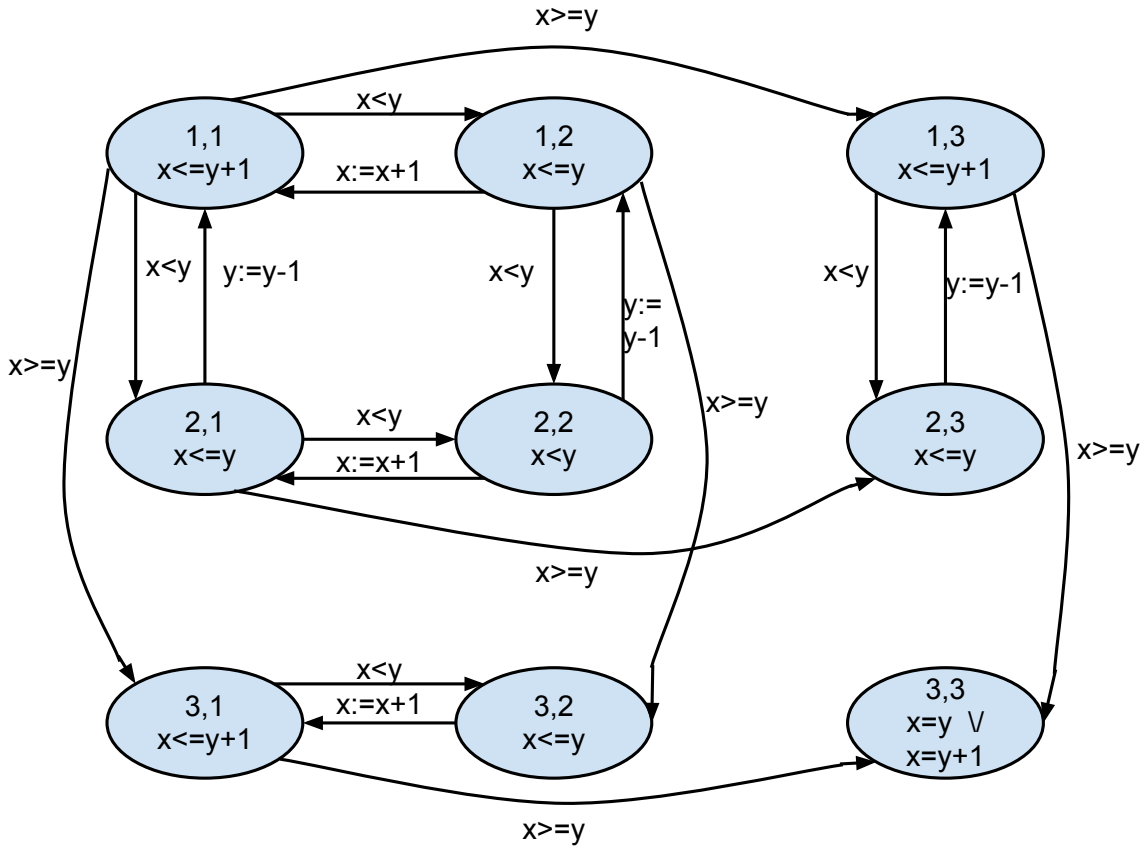

Figure 3.1: Proof graph for Example 34.

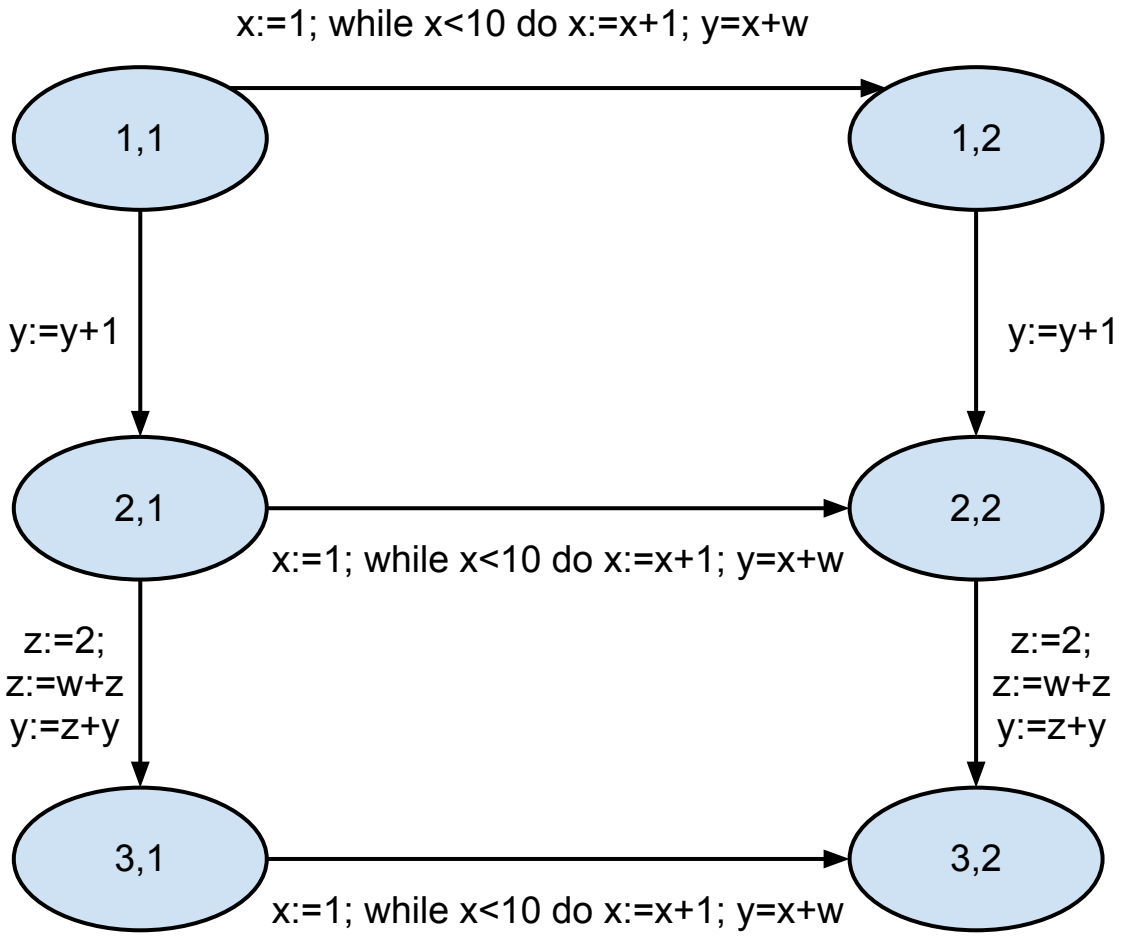

Figure 3.2: Interference graph for Example 35.

Example 35. Let C be the parallel command

```
                    COBEGIN
x := 1                              y := y + 1
WHILE x < 10 DO          ||         z := 2
  x := x + 1                        z := w + z
  y := x + w                        y := z + y
                    COEND
```

An interference graph for the previous command is given in Figure 3.2. By the former program graph definition (Definition 28) this program would have 12 or more vertices.

4 Mesh Hoare Logic

In this chapter we formalise the syntax and semantics of mesh algorithms (Section 4.1). We also present a partial corectness proof of Sung Bae's parallel maximum subarray algorithm (Algorithm 1). The algorithm is presented in a similar way as by Bae[3]. It has some subtle differences to the versions of the same algorithm presented by Weddell[25] and the earlier Bae[5] (the update operation is split into two parts). Section 4.2 contains the final proof rules required for the verification and Section 4.3 contains the proof of partial correctness.

4.1 Mesh Algorithms

Definition 36. Let I be some countable index set (of any dimension). Let A be a sequence of variables indexed by I. We call A an *array*. For each $i \in I$ we call the variable $A[i]$ an *array variable*. If I is multidimensional we index the elements of A by the components of the vectors in I.

Remark 37. In this thesis we assume that array variables can not index arrays. We make this assumption so we can treat array variables with the same axioms as other types of variables. Justification for such an assumption is given by the examples in [18].

Definition 38. Let C be a command and I be some countable index set (of any dimension) not modified by C. An *indexed cobegin command*

$$\texttt{COBEGIN } ||_{i \in I} C \texttt{ COEND}$$

concurrently executes C for each value of i. Each value of i is a local variable in C labeled as i (Remark 20). If i is a vector (i.e. the index set was multidimensional), then the components of i are also local variables in C.

Remark 39. Let $I = \{i_0, i_1 \dots i_n\}$ be a countable index set. Informally the semantics in Definition 38 can be thought of as:

$$\texttt{COBEGIN } i := i_0; C || i := i_1; C || \dots || i := i_n; C \texttt{ COEND}$$

With each assignment to i being for a local variable i. In Remark 20 we discussed conditions where this ambiguity is acceptable although we do not give a formal syntax for local variable name sharing.

Definition 40. Let the indexed cobegin command M be given by `COBEGIN` $||_{(i,j)\in I} C$ `COEND` where I is some countable $2-$dimensional index set of integers. M is a *mesh algorithm* if it has the following two properties:

- Assignments in C are made only to array variables indexed by (i, j).
- References to array variables in C must have index in the range $(i \pm 1, j)$ or $(i, j \pm 1)$.
- M has no shared variables (Definition 23).

Remark 41. Informally we may think of each concurrent command in M from Definition 40 as a mesh cell. Then, each array variable indexed by (i, j) are the cell's registers. The second condition ensures arrays only access registers in adjacent cells. Mesh algorithms are interference free by the last property.

Example 42. Algorithm 1 is Bae's parallel mesh algorithm for the maximum subarray problem. Given a $2-$Dimensional array a the algorithm finds the sum of the maximum rectangular subarray in a. Informally we make think of the solution as propagating down and to the right in the mesh of cells. After a certain number of steps the solution is in the bottom right cell.

Let I be the index set given by $\{1, 2, 3, \dots N\}$ for some positive integer N. Under these conditions Algorithm 1 contains three mesh algorithms. The mesh algorithm on lines $1-12$ of Algorithm 1 is known as the setup phase. The two other mesh algorithms (lines $15-26$ and 27-34) are embedded in a for loop and can be thought of as update operations.

To verify these `COBEGIN` commands are mesh algorithms, we observe only array variables with index (i, j) are in assignment statements. Only array variables in the the same cell or adjacent are referenced. Lines $27-34$ are the only places other array variable indexes are referenced in the algorithm and no array is used for assignment and reference in this block. Hence all three `COBEGIN` blocks are mesh algorithms.

4.2 Mesh Hoare Logic Rules

Proposition 43 (Simplified For Rule)**.** *If k does not appear in P_i for $i \in \{0, 1, 2, \dots n\}$ then*

$$\frac{\{P_{k-1}\}\ C\ \{P_k\} \qquad \{k = i\}\ C\ \{k = i\}}{\{P_0\}\ \texttt{FOR}\ k := 1 \dots n\ \texttt{DO}\ C\ \{P_n\}}$$

Proof. Let $i \in \{0, 1, 2, \dots n\}$. By the Proposition 12) and the premises

$$\{P_{k-1} \wedge k = i\}\ C\ \{P_k \wedge k = i\}$$

Which by post condition weakening (Axiom 4) implies

$$\{P_{k-1} \wedge k = i\}\ C\ \{P_i\}$$

Algorithm 1 Sung Bae's Mesh Algorithm for the Maximum Subarray Problem

```
1:  COBEGIN ||_{(i,j)∈(I∪{0})²}
2:    r[i][j] := 0;
3:    sum[i][j] := 0;
4:    min[i][j] := 0;
5:    max[i][j] := −∞;
6:    rL[i][j] := 0;
7:    minL[i][j] := 0;
8:    sumL[i][j] := 0;
9:    maxL[i][j] := −∞;
10:   sumU[i][j] := 0;
11:   maxU[i][j] := −∞;
12: COEND
13: FOR step := 1 ... 2N − 1 DO
14: BEGIN
15:   COBEGIN ||_{(i,j)∈I²}
16:     IF j <= step DO
17:     BEGIN
18:       r[i][j] := rL[i][j] + a[i][j];
19:       min[i][j] := minL[i][j];
20:        IF min[i][j] > sumL[i][j] DO min[i][j] := sumL[i][j];
21:       sum[i][j] := sumU[i][j] + r[i][j];
22:        IF sum[i][j] − min[i][j] > max[i][j] DO max[i][j] := sum[i][j] − min[i][j];
23:        IF maxU[i][j] > max[i][j] DO max[i][j] := maxU[i][j];
24:        IF maxL[i][j] > max[i][j] DO max[i][j] := maxL[i][j]
25:     END
26:   COEND
27:   COBEGIN ||_{(i,j)∈I²}
28:     rL[i][j] := r[i][j − 1]
29:     minL[i][j] := min[i][j − 1];
30:     sumL[i][j] := sum[i][j − 1];
31:     maxL[i][j] := max[i][j − 1];
32:     sumU[i][j] := sum[i − 1][j];
33:     maxU[i][j] := max[i − 1][j]
34:   COEND
35: END
```

as k does not appear in P_i. Therefore by assignment rule (Axiom 2) and sequencing rule (Axiom 6)

$$\{P_{i-1}\}\ k := i; C\ \{P_i\}$$

The result yields by observing the choice of i was arbitrary and an application of Axiom 9. □

Axiom 10 (Mesh Rule)**.** Let C be the concurrent command of a mesh command

$$\texttt{COBEGIN}\ ||_{(i,j)\in I} C\ \texttt{COEND}$$

for some countable index set I. Let P be a condition. Let $\{Q(i,j) : (i,j) \in I\}$ be a set of conditions.

$$\frac{\{P \wedge (i,j) \in I\}\ C\ \{Q(i,j)\}}{\{P\}\ \ \texttt{COBEGIN}\ ||_{(i,j)\in I} C\ \texttt{COEND}\ \ \{\forall (l,k) \in I \quad Q(l,k)\}}$$

4.3 Verification of Sung Bae's Algorithm 38

Remark 44. Throughout the verification of Algorithm 1 we make use of invariants. Consider one of the following conditions P. At the end of step z, for each index $(y,x) \in I$, $P(x,y,z)$ holds. At the start of step z, for each index $(y,x) \in I$, $P(x,y,z-1)$ holds.

For a given x, y and z, we refer to the subarray with top left corner $a[y+x-z][1]$ and bottom right corner $a[y][x]$ as the strip of $a[y][x]$. We refer to the subarray $a[y][1]$ to $a[y][x]$ as the row of $a[y][x]$. A concurrent process (or cell) is considered active at z if $x \leq z$.

We assume if a summation sign has invalid boundries it defaults to 0 (e.g. $\sum_{i=1}^{0} = 0$). Similarly we assume `MIN` defaults to 0 and `MAX` defaults to $-\infty$ for out of bounds cases and when $x \leq z(+1)$ does not hold. We make the final assumption that if an array is being accessed with a negative index, the array gives the value at index 0 (e.g. $a[-10][3] = a[0][3]$). An informal description of each invariant is presented under each.

$$Pr(y,x,z) \Leftrightarrow x \leq z \rightarrow r[y][x] = \sum_{\alpha=1}^{x} a[y][\alpha]$$

If cell (y,x) is active at z, then $r[y][x]$ is the sum of all entries in the row of $a[y][x]$.

$$Psum(y,x,z) \Leftrightarrow x \leq z \rightarrow sum[y][x] = \sum_{\beta=y+x-z}^{y} \sum_{\alpha=1}^{x} a[\beta][\alpha]$$

If cell (y,x) is active at z, then $sum[y][x]$ is the sum of all entries in the strip of $a[y][x]$.

$$Pmin(y,x,z) \Leftrightarrow x \leq z \rightarrow min[y][x] = \ \texttt{MIN}_{\,\overline{\alpha}\in\{1,2,\ldots x-1\}} \sum_{\beta=y+x-z}^{y} \sum_{\alpha=1}^{\overline{\alpha}} a[\beta][\alpha]$$

If cell (y, x) is active at z, then $min[y][x]$ is the minimum prefix sum in the strip of $a[y][x]$.

$$Pmax(y, x, z) \Leftrightarrow x \leq z \rightarrow max[y][x] = \texttt{MAX}_{\underline{\beta},\overline{\beta}\in\{y+x-z\ldots y\},\underline{\alpha},\overline{\alpha}\in\{1,2\ldots x\}} \sum_{\beta=\underline{\beta}}^{\overline{\beta}} \sum_{\alpha=\underline{\alpha}}^{\overline{\alpha}} a[\beta][\alpha])$$

If cell (y, x) is active at z, then $max[y][x]$ is the maximum subarray of the strip of $a[y][x]$.

$$PrL(y, x, z) \Leftrightarrow x \leq z + 1 \rightarrow rL[y][x] = \sum_{\alpha=1}^{x-1} a[y][\alpha]$$

If cell (y, x) is active at $z + 1$, then $rL[y][x]$ is the sum of the row of the cell to the left of $a[y][x]$ $(a[y][x - 1])$.

$$PsumL(y, x, z) \Leftrightarrow x \leq z + 1 \rightarrow sumL[y][x] = \sum_{\beta=y+x-z-1}^{y} \sum_{\alpha=1}^{x-1} a[\beta][\alpha]$$

If cell (y, x) is active at $z+1$, then $sumL[y][x]$ is the sum of the strip of the cell to the left of $a[y][x]$ $(a[y][x - 1])$.

$$PminL(y, x, z) \Leftrightarrow x \leq z + 1 \rightarrow minL[y][x] = \texttt{MIN}_{\overline{\alpha}\in\{1,2,\ldots x-2\}} \sum_{\beta=y+x-z-1}^{y} \sum_{\alpha=1}^{\overline{\alpha}} a[\beta][\alpha]$$

If cell (y, x) is active at $z + 1$, then $minL[y][x]$ is the minimum prefix sum of the strip of the cell to the left of $a[y][x]$ $(a[y][x - 1])$.

$$PmaxL(y, x, z) \Leftrightarrow x \leq z+1 \rightarrow maxL[y][x] = \texttt{MAX}_{\underline{\beta},\overline{\beta}\in\{y+x-z-1\ldots y\},\underline{\alpha},\overline{\alpha}\in\{1,2\ldots x-1\}} \sum_{\beta=\underline{\beta}}^{\overline{\beta}} \sum_{\alpha=\underline{\alpha}}^{\overline{\alpha}} a[\beta][\alpha])$$

If cell (y, x) is active at $z + 1$, then $maxL[y][x]$ is the maximum subarray of the strip of the cell left of $a[y][x]$ $(a[y][x - 1])$.

$$PsumU(y, x, z) \Leftrightarrow x \leq z \rightarrow sumU[y][x] = \sum_{\beta=y+x-z-1}^{y-1} \sum_{\alpha=1}^{x} a[\beta][\alpha]$$

If cell (y, x) is active at z, then $sumU[y][x]$ is the sum of all entries of the strip of the cell above $a[y][x]$ $(a[y - 1][x])$.

$$PmaxU(y, x, z) \Leftrightarrow x \leq z \rightarrow maxU[y][x] = \texttt{MAX}_{\underline{\beta},\overline{\beta}\in\{y+x-z-1\ldots y-1\},\underline{\alpha},\overline{\alpha}\in\{1,2\ldots x\}} \sum_{\beta=\underline{\beta}}^{\overline{\beta}} \sum_{\alpha=\underline{\alpha}}^{\overline{\alpha}} a[\beta][\alpha])$$

If cell (y, x) is active at z, then $maxU[y][x]$ is the maximum subarray of the strip of the cell above $a[y][x]$ ($a[y-1][x]$).

$$PU1(y, x, z) \Leftrightarrow Pr(y, x, z) \wedge Psum(y, x, z) \wedge Pmin(y, x, z) \wedge Pmax(y, x, z)$$

The invariant for the first update phase (lines 16-25) of Algorithm 1.

$$PU2L(y, x, z) \Leftrightarrow PrL(y, x, z) \wedge PminL(y, x, z) \wedge PsumL(y, x, z) \wedge PmaxL(y, x, z)$$

The invariant for the *left* type registers. This invariant is used for the second update phase (lines 28-33) of Algorithm 1.

$$PU2U(y, x, z) \Leftrightarrow PsumU(y, x, z) \wedge PmaxU(y, x, z)$$

The invariant for the *up* (or above) type registers. This invariant is used for the second update phase (lines 28-33) of Algorithm 1.

$$PU2(y, x, z) \Leftrightarrow PU2L(y, x, z) \wedge PU2U(y, x, z)$$

The invariant combining both *up* and *left* invariants for the second update phase (lines 28-33) of Algorithm 1.

$$PU(z) \Leftrightarrow \forall x, y \in (I \cup \{0\}) PU1(y, x, z) \wedge PU2(y, x, z)$$

For all x, y the previous invariants all hold at step z.

The correctness proof proceeds in a top-down manner. The lemmas used are given after the usage points.

Theorem 45 (Partial Correctness of Algorithm 1)**.** *Let C be Algorithm 1. Let I be the index set given by $\{1, 2, 3, \ldots N\}$ for some positive integer N. Let*

$$Q \Leftrightarrow max[N][N] = \texttt{MAX}_{\underline{\beta},\overline{\beta},\underline{\alpha},\overline{\alpha} \in I} \sum_{\beta=\underline{\beta}}^{\overline{\beta}} \sum_{\alpha=\underline{\alpha}}^{\overline{\alpha}} a[\beta][\alpha]$$

then $\{true\}\ C\ \{Q\}$.

Proof. Let C_S and C_U be lines $1-12$ (setup phase) and $14-34$ (update phase) of Algorithm 1 respectively. By the sequencing rule, Lemma 47 and Lemma 48:

$$\{PU(step-1)\}\ C_U\ \{PU(step)\}$$

Hence as *step* does not appear in $PU(i)$ for any i and is not assigned a value in C_U,

$$\{PU(0)\}\ \texttt{FOR}\ step := 1 \ldots 2N-1\ \texttt{DO}\ C_U\ \{PU(2N-1)\}$$

by Proposition 43. Therefore by Lemma 46 and the sequencing rule (Axiom 6):

$$\{True\}\ C_S\ \texttt{FOR}\ step := 1 \ldots 2N-1\ \texttt{DO}\ C_U\ \{PU(2N-1)\}$$

But $PU(2N-1) \rightarrow Pmax(N, N, 2N-1) \rightarrow Q$, so by postcondition weakening (Axiom 4)

$$\{True\}\ C_S\ \texttt{FOR}\ step := 1 \ldots 2N-1\ \texttt{DO}\ C_U\ \{Q\}$$

□

Lemma 46. *Let C be the setup phase of Algorithm 1 (lines $1-12$). Let $PU(0)$ be the invariant condition given in Remark 44. Then*

$$\{true\}\ C\ \{PU(0)\}$$

Proof. Let C_s be given by

$r[i][j] := 0;$
$sum[i][j] := 0;$
$min[i][j] := 0;$
$max[i][j] := -\infty;$
$rL[i][j] := 0;$
$minL[i][j] := 0;$
$sumL[i][j] := 0;$
$maxL[i][j] := -\infty;$
$sumU[i][j] := 0;$
$maxU[i][j] := -\infty;$

Let

$$reg0 = \{r, sum, min, rL, minL, sumL, maxL, sumU, maxU\}$$

$$reg\infty = \{max, maxL, maxU\}$$

$$Q1(i,j) \Leftrightarrow \wedge_{R \in reg0} R[i][j] = 0$$

and

$$Q2(i,j) \Leftrightarrow \wedge_{R \in reg\infty} R[i][j] = -\infty$$

By repeated application of the assignment rule (Axiom 2):

$$\{true\}\ C_s\ \{Q_1(i,j) \wedge Q_2(i,j)\}$$

Hence by precondition strengthening (Axiom 7)

$$\{true \wedge (i,j) \in I^2\}\ C_s\ \{Q_1(i,j) \wedge Q_2(i,j)\}$$

So by Example 42 and Axiom 10

$$\{true\}\ ||_{(i,j)\in(I\cup\{0\})^2} C_s\ \{\forall (l,k) \in (I \cup \{0\})^2 Q1(l,k) \wedge Q2(l,k)\}$$

Because of the defaulting behavior discussed in Remark 44 we have:

$$\forall (l,k) \in (I \cup \{0\})^2 Q1(l,k) \wedge Q2(l,k) \rightarrow PU(0)$$

Hence by postcondition weakening (Axiom 4)

$$\{true\}\ ||_{(i,j)\in(I\cup\{0\})^2} C_s\ \{PU(0)\}$$

□

Lemma 47. *Let C be the first update phase of Algorithm 1 (lines $15-26$). Then*

$$\{PU(step-1)\}\ C\ \{\forall x,y \in (I \cup \{0\}) PU1(x,y,step)\}$$

Proof. Let C_u be given by

```
BEGIN
  r[i][j] := rL[i][j] + a[i][j];
  min[i][j] := minL[i][j];
   IF min[i][j] > sumL[i][j] DO min[i][j] := sumL[i][j];
  sum[i][j] := sumU[i][j] + r[i][j];
   IF sum[i][j] − min[i][j] > max[i][j] DO max[i][j] := sum[i][j] − min[i][j];
   IF maxU[i][j] > max[i][j] DO max[i][j] := maxU[i][j];
   IF maxL[i][j] > max[i][j] DO max[i][j] := maxL[i][j]
END
```

Consider the $r[i][j]$ assignment of C_u. By Lemma 49 we have:

$$\{PrL(i,j,step-1)\}\ r[i][j] := rL[i][j] + a[i][j]\ \{Pr(i,j,step)\}$$

Let C_{min} be given by:

$$min[i][j] := minL[i][j];\ \texttt{IF}\ min[i][j] > sumL[i][j]\ \texttt{DO}\ min[i][j] := sumL[i][j]$$

By Lemma 50 and Corollary 15 we have:

$$\{PminL(i,j,step-1) \wedge PsumL(i,j,step-1)\}\ C_{min}\ \{Pmin(i,j,step)\}$$

By Lemma 51 we have:

$$\{Pr(i,j,step) \wedge PsumU(i,j,step-1)\}\ sum[i][j] := sumU[i][j]+r[i][j]\ \{Psum(i,j,step)\}$$

Hence by sequencing rule (Axiom 6) and the assumption $\wedge(i,j) \in I$ we can establish the precondition

$$Pmax(i,j,step-1) \wedge PmaxL(i,j,step-1) \wedge PmaxU(i,j,step-1) \wedge Psum(i,j,step) \wedge Pmin(i,j,step)$$

before the $max[i][j]$ assignment conditionals. Therefore by Lemma 52, Proposition 14 and precondition strengthening (Axiom 7):

$$\{PU(step-1) \wedge j \leq step \wedge i,j \in I\}\ C_u\ \{PU1(i,j,step)\}$$

Also, as the invariants say nothing if $x > z$ (or $j > step$) so we have:

$$PU(step-1) \wedge j > step \wedge i,j \in I \rightarrow PU1(i,j,step)$$

Therefore by the weak conditional rule Axiom 4:

$$\{PU(step-1) \wedge i,j \in I\}\ \ \texttt{IF}\ j <= step\ \texttt{DO}\ C_u\ \{PU1(i,j,step)\}$$

Hence by Axiom 10 and Example 42:

$$\{PU(step-1)\}\ C\ \{\forall x,y \in (I \cup \{0\}) PU1(x,y,step)\}$$

The case of $x = 0 \vee y = 0$ is invariant in C and therefore is justified in $PU1(step)$. □

Lemma 48. *Let C be the second update phase of Algorithm 1 (lines $27-34$). Then*

$$\{\forall x, y \in (I \cup \{0\})PU1(x, y, step)\}\ C\ \{PU(step)\}$$

Proof. Let C_u be given by

$rL[i][j] := r[i][j-1]$
$minL[i][j] := min[i][j-1];$
$sumL[i][j] := sum[i][j-1];$
$maxL[i][j] := max[i][j-1];$
$sumU[i][j] := sum[i-1][j];$
$maxU[i][j] := max[i-1][j]$

The only assignment statements are made to registers of the form RL and RU. Hence

$$\forall x, y \in (I \cup \{0\})PU1(x, y, step)$$

is invariant in C_u. As each assignment in C_u is to a different array we may verify each of the corresponding invariants in $PU2L(i, j, step)$ and $PU2L(i, j, step)$ individually.

By Lemmas 53-58 and the assignment rule (Axiom 2) we find that the verification relies on the preconditions:

$$\begin{aligned} &Pr(i, j-1, step)\\ &Pmin(i, j-1, step)\\ &Psum(i, j-1, step)\\ &Pmax(i, j-1, step)\\ &Psum(i-1, j, step)\\ &Pmax(i-1, j, step) \end{aligned}$$

The precondition

$$(\forall x, y \in (I \cup \{0\})PU1(x, y, step)) \wedge i, j \in I$$

implies all the listed preconditions and is invariant in C_u. Hence by precondition strengthening (Axiom 7) and Axiom 12:

$$\{(\forall x, y \in (I \cup \{0\})PU1(x, y, step)) \wedge i, j \in I\}\ C_u\ \{PU1(i, j, step) \wedge PU2(i, j, step)\}$$

Therefore by Axiom 10 and Example 42:

$$\{\forall x, y \in (I \cup \{0\})PU1(x, y, step)\}\ C\ \{PU(step)\}$$

The case of $x = 0 \vee y = 0$ is invariant in C_u and therefore is justified in $PU(step)$. □

Lemma 49. $Pr(i, j, step)[rL[i][j] + a[i][j]/r[i][j]] \Leftrightarrow PrL(i, j, step - 1)$

Proof.

$$
\begin{aligned}
& Pr(i,j,step)[rL[i][j] + a[i][j]/r[i][j]] \\
\Leftrightarrow & j \leq step \rightarrow rL[i][j] + a[i][j] = \sum_{\alpha=1}^{j} a[i][\alpha] \\
\Leftrightarrow & j \leq (step-1)+1 \rightarrow rL[i][j] = \sum_{\alpha=1}^{j} a[i][\alpha] - a[i][j] \\
\Leftrightarrow & j \leq (step-1)+1 \rightarrow rL[i][j] = \sum_{\alpha=1}^{j-1} a[i][\alpha] \\
\Leftrightarrow & PrL(i,j,step-1)
\end{aligned}
$$

□

Lemma 50.

$PminL(i,j,step{-}1) \wedge PsumL(i,j,step{-}1) \rightarrow Pmin(i,j,step)[\ \texttt{MIN}\ (minL[i][j], sumL[i][j])/min[i][j]]$

Proof. Assume

$$minL[i][j] > \ \texttt{MIN}_{\ \overline{\alpha} \in \{1,2,\ldots j-1\}} \sum_{\beta=i+j-step}^{i} \sum_{\alpha=1}^{\overline{\alpha}}$$

Then

$$sumL[i][j] = \ \texttt{MIN}_{\ \overline{\alpha} \in \{1,2,\ldots j-1\}} \sum_{\beta=i+j-step}^{i} \sum_{\alpha=1}^{\overline{\alpha}}$$

as it must have the elements of the last column.

In either case:

$$
\begin{aligned}
\Rightarrow j \leq step \rightarrow \ \texttt{MIN}\ (minL[i][j], sumL[i][j]) = \ & \texttt{MIN}_{\ \overline{\alpha} \in \{1,2,\ldots j-1\}} \sum_{\beta=i+j-step}^{i} \sum_{\alpha=1}^{\overline{\alpha}} a[\beta][\alpha] \\
\Rightarrow Pmin(i,j,step)[\ & \texttt{MIN}\ (minL[i][j], sumL[i][j])/min[i][j]]
\end{aligned}
$$

□

Lemma 51. $PsumU(i,j,step{-}1) \wedge Pr(i,j,step) \rightarrow Psum(i,j,step)[sumU[i][j]{+}r[i][j]/sum[i][j]]$

Proof. If $j = step$ then $sumU[i][j]$ defaults to its original value ($sumU[i][j] = 0$) and

$$
\begin{aligned}
& \Rightarrow r[i][j] = \sum_{\beta=i}^{i} \sum_{\alpha=1}^{j} a[\beta][\alpha] \\
& \Rightarrow sumU[i][j] + r[i][j] = \sum_{\beta=i+step-step}^{i} \sum_{\alpha=1}^{j} a[\beta][\alpha]
\end{aligned}
$$

If $j <= step - 1$ then:

$$sumU[i][j] + r[i][j] = \sum_{\beta=i+j-step}^{i} \sum_{\alpha=1}^{j} a[\beta][\alpha]$$

In either case:

$$\Rightarrow j \leq step \rightarrow sumU[i][j] + r[i][j] = \sum_{\beta=i+j-step}^{i} \sum_{\alpha=1}^{j} a[\beta][\alpha]$$
$$\Rightarrow Psum(i, j, step)[sumU[i][j] + r[i][j]/sum[i][j]]$$

□

Lemma 52.

$$Pmax(i, j, step{-}1) \wedge PmaxL(i, j, step{-}1) \wedge PmaxU(i, j, step{-}1) \wedge Psum(i, j, step) \wedge Pmin(i, j, step)$$
$$\rightarrow Pmax(i, j, step)[\ \texttt{MAX}\ (max[i][j], maxU[i][j], maxL[i][j], sum[i][j] - min[i][j])/max[i][j]]$$

Proof. Assume the maximum (of the `MAX` argument in $Pmax(i, j, step)$) contains $a[i][j]$ and $a[i + j - step][j]$. By $Psum(i, j, step) \wedge Pmin(i, j, step)$, the maximum is given by $sum[i][j] - min[i][j]$ as $sum[i][j]$ is the sum of the strip $i + j - step \ldots i$ up to j and $min[i][j]$ is the minimum prefix sum along the same strip (A formal justification of these argument can be found in Bae[4]).

If the maximum contains $a[i][j]$ but not $a[i + j - step][j]$, then as it is rectangular it contains no element of a from row $i + j - step$. Therefore, by $Pmax(i, j, step - 1)$, it is given by $max[i][j]$.

Assume the maximum does not contain $a[i][j]$ but contains an element $a[x][j]$ for some x. Then, as it is rectangular, it contains no element of a from row i. Therefore, by $PmaxU(i, j, step - 1)$, it is given by $maxU[i][j]$.

Conversely, suppose the maximum does not contain $a[i][j]$ or an element $a[x][j]$ for any x. As it is rectangular, it contains no element of a from the column j. Therefore, by $PmaxL(i, j, step)$, it is given by $maxL[i][j]$.

In any case

$$Pmax(i, j, step{-}1) \wedge PmaxL(i, j, step{-}1) \wedge PmaxU(i, j, step{-}1) \wedge Psum(i, j, step) \wedge Pmin(i, j, step)$$
$$\rightarrow Pmax(i, j, step)[\ \texttt{MAX}\ (max[i][j], maxU[i][j], maxL[i][j], sum[i][j] - min[i][j])/max[i][j]]$$

□

Lemma 53. $PrL(i, j, step)[r[i][j - 1]/rL[i][j]] \Leftrightarrow Pr(i, j - 1, step)$

Proof.

$$\begin{aligned}
&PrL(i,j,step)[r[i][j-1]/rL[i][j]]\\
\Leftrightarrow &j \le step+1 \rightarrow r[i][j-1] = \sum_{\alpha=1}^{j-1} a[\beta][\alpha]\\
\Leftrightarrow &(j-1) \le step \rightarrow r[i][j-1] = \sum_{\alpha=1}^{j-1} a[\beta][\alpha]\\
\Leftrightarrow &Pr(i,j-1,step)
\end{aligned}$$

□

Lemma 54. $PminL(i,j,step)[min[i][j-1]/minL[i][j]] \Leftrightarrow Pmin(i,j-1,step)$

Proof.

$$\begin{aligned}
&PminL(i,j,step)[min[i][j-1]/minL[i][j]]\\
\Leftrightarrow &j \le step+1 \rightarrow min[i][j-1] = \texttt{MIN}_{\overline{\alpha}\in\{1,2,\ldots j-2\}} \sum_{\beta=i+j-step-1}^{i} \sum_{\alpha=1}^{\overline{\alpha}} a[\beta][\alpha]\\
\Leftrightarrow &(j-1) \le step \rightarrow min[i][j-1] = \texttt{MIN}_{\overline{\alpha}\in\{1,2,\ldots (j-1)-1\}} \sum_{\beta=i+(j-1)-step}^{i} \sum_{\alpha=1}^{\overline{\alpha}} a[\beta][\alpha]\\
\Leftrightarrow &Pmin(i,j-1,step)
\end{aligned}$$

□

Lemma 55. $PsumL(i,j,step)[sum[i][j-1]/sumL[i][j]] \Leftrightarrow Psum(i,j-1,step)$

Proof.

$$\begin{aligned}
&PsumL(i,j,step)[sum[i][j-1]/sumL[i][j]]\\
\Leftrightarrow &j \le step+1 \rightarrow sum[i][j-1] = \sum_{\beta=i+j-step-1}^{i} \sum_{\alpha=1}^{j-1} a[\beta][\alpha]\\
\Leftrightarrow &j-1 \le step \rightarrow sum[i][j-1] = \sum_{\beta=i+(j-1)-step}^{i} \sum_{\alpha=1}^{j-1} a[\beta][\alpha]\\
\Leftrightarrow &Psum(i,j-1,step)
\end{aligned}$$

□

Lemma 56. $PmaxL(i,j,step)[max[i][j-1]/maxL[i][j]] \Leftrightarrow Pmax(i,j-1,step)$

Proof.

$$PmaxL(i,j,step)[max[i][j-1]/maxL[i][j]]$$

$$\Leftrightarrow j \leq step+1 \rightarrow max[i][j-1] = \texttt{MAX}_{\underline{\beta},\overline{\beta}\in\{i+j-step-1\ldots i\},\underline{\alpha},\overline{\alpha}\in\{1,2\ldots j-1\}} \sum_{\beta=\underline{\beta}}^{\overline{\beta}} \sum_{\alpha=\underline{\alpha}}^{\overline{\alpha}} a[\beta][\alpha])$$

$$\Leftrightarrow (j-1) \leq step \rightarrow max[i][j-1] = \texttt{MAX}_{\underline{\beta},\overline{\beta}\in\{i+(j-1)-step\ldots i\},\underline{\alpha},\overline{\alpha}\in\{1,2\ldots (j-1)\}} \sum_{\beta=\underline{\beta}}^{\overline{\beta}} \sum_{\alpha=\underline{\alpha}}^{\overline{\alpha}} a[\beta][\alpha])$$

$$\Leftrightarrow Pmax(i,j-1,step)$$

□

Lemma 57. $PsumU(i,j,step)[sum[i-1][j]/sumU[i][j]] \Leftrightarrow Psum(i-1,j,step)$

Proof.

$$\begin{aligned}
&PsumU(i,j,step)[sum[i-1][j]/sumU[i][j]] \\
\Leftrightarrow &j \leq step \rightarrow sum[i-1][j] = \sum_{\beta=i+j-step-1}^{i-1} \sum_{\alpha=1}^{j} a[\beta][\alpha] \\
\Leftrightarrow &j \leq step \rightarrow sum[i-1][j] = \sum_{\beta=(i-1)+j-step}^{i-1} \sum_{\alpha=1}^{j} a[\beta][\alpha] \\
\Leftrightarrow &Psum(i-1,j,step)
\end{aligned}$$

□

Lemma 58. $PmaxU(i,j,step)[max[i-1][j]/maxU[i][j]] \Leftrightarrow Pmax(i-1,j,step)$

Proof.

$$PmaxU(i,j,step)[max[i-1][j]/maxU[i][j]]$$

$$\Leftrightarrow j \leq step \rightarrow max[i-1][j] = \texttt{MAX}_{\underline{\beta},\overline{\beta}\in\{i+j-step-1\ldots i-1\},\underline{\alpha},\overline{\alpha}\in\{1,2\ldots j\}} \sum_{\beta=\underline{\beta}}^{\overline{\beta}} \sum_{\alpha=\underline{\alpha}}^{\overline{\alpha}} a[\beta][\alpha])$$

$$\Leftrightarrow j \leq step \rightarrow max[i-1][j] = \texttt{MAX}_{\underline{\beta},\overline{\beta}\in\{(i-1)+j-step\ldots i-1\},\underline{\alpha},\overline{\alpha}\in\{1,2\ldots j\}} \sum_{\beta=\underline{\beta}}^{\overline{\beta}} \sum_{\alpha=\underline{\alpha}}^{\overline{\alpha}} a[\beta][\alpha])$$

$$\Leftrightarrow Pmax(i-1,j,step)$$

□

5 Conclusion

In this chapter we summarise the research undertaken (Section 5.1), clarify which parts are original (Section 5.2) and describe potential future work (Section 5.3).

5.1 Summary of work and objectives

This project had two primary goals:

- To formalise and modify an existing framework to verify concurrent programs using Hoare logic.
- To formally verify Algorithm 1.

The first objective was achieved incrementally throughout each chapter. Chapter 2 provides the standard sequential basis and formulation of Hoare logic. The framework relies heavily on local variables (Definition 19) and shared variables (Definition 23) to reason about interference. These definitions and a complete discussion of them is provided in Chapter 3. The semantics defined in Definition 38 and Definition 40 were also key aspects to the framework (Section 4.1). The final and most important part of the framework is the mesh proof rule given in Axiom 10. This proof rule was presented in Section 4.2.

We have provided a full proof (Theorem 45) of the partial correctness of Algorithm 1 in Section 4.3. This proof was formal with regard to semantics and did not make restrictive assumptions about the order of execution. In some ways it can be thought of as a slight extension of some of the results from Bae[3] and Weddell[25]. The second objective has therefore been met.

5.2 Original Aspects

Although many of the definitions and theories in this thesis were independently formulated they are by no means original. Where the concepts can be credited to a particular author we have made an effort to refer to a publication indicating this. However, in many cases the concepts are standard and no specific publication exists.

Each proof presented (unless otherwise stated) is the authors original work (although in some of the trivial cases similar proofs will exist).

Specifically, there are two main original results in the thesis. The first is Axiom 10 in which we give a potential proof rule for mesh algorithms. The proof of Theorem 45 (the

partial correctness of Algorithm 1) and the associated lemmas using Hoare logic framework is the second.

5.3 Future Work and Reservations

There are many aspects to the research for this project that could be developed further. There are also reservations we have about the usefulness of the research conducted for this project. We describe some of each below:

The partial correctness of Algorithm 1 was one application of the framework we developed. It would be desirable to know if other mesh algorithms and even other classes of parallel algorithms can be verified using the same framework.

In Section 3.4 we present a combination of ideas but do not thoroughly explore or formalise them (besides Example 35). This is a potential route for further research.

No discussion of soundness or completeness is given for the framework we provided. To give the framework more credibility soundness must be established (this should be trivial based on the lack of shared variables in the semantics). To establish its usefulness, the degree to which it is complete should be investigated.

How much confidence does a very long proof of partial correctness give us? We essentially have a new problem of verifying a proof instead of a computer program. De Millo[9] argues that such proof verification is essentially a social process. If that is indeed the case, then a proof of partial correctness only provides us with greater confidence and does not truly prove correctness.

In Hoare's 1996 paper[13] (*Unification of theories: A challenge for computing science*), Hoare neatly summarises another key issue facing formal verification:

> *"Researchers into formal methods (and I was the most mistaken among them) predicted that the programming world would embrace with gratitude every assistance promised by formalisation to solve the problems of reliability that arise when programs get large and more safety-critical. Programs have now got very large and very critical – well beyond the scale which can be comfortably tackled by formal methods. There have been many problems and failures, but these have nearly always been attributable to inadequate analysis of requirements or inadequate management control. It has turned out that the world just does not suffer significantly from the kind of problem that our research was originally intended to solve."*

Bibliography

[1] K. R. Apt. Ten years of hoare's logic: A survey - part ii: Nondeterminism. *Theoretical Computer Science*, 28(1-2):83 – 109, 1984.

[2] K.R. Apt. Ten years of hoare's logic: a survey. i. *ACM Transactions on Programming Languages and Systems*, 3(4):431 – 83, 1981.

[3] S. Bae. Sequential and parallel algorithms for the generalized maximum subarray problem. *PhD Thesis (University of Canterbury)*, 2007.

[4] S. Bae and T. Takaoka. Improved algorithms for the k-maximum subarray problem for small k. *Computing and Combinatorics*, pages 621–631, 2005.

[5] S. E. Bae and T. Takaoka. Parallel approaches to the maximum subarray problem. *Proc. of the seventh Japan-Korea Workshop on Algorithms and Computation (2003)*, 7:94 – 104, 2003.

[6] J.A. Bergstra and J.V. Tucker. Axiomatic semantics of programs based on hoare's logic. *Acta Informatica*, 21(3):293 – 320, 1984.

[7] A. Blass and Y. Gurevich. The underlying logic of hoare logic. *Current Trends in Theoretical Computer Science*, pages 409–436, 2001.

[8] S.A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, 1978.

[9] R.A. De Millo, R.J. Lipton, and A.J. Perlis. Social processes and proofs of theorems and programs. *Communications of the ACM*, 22(5):271–280, 1979.

[10] R.W. Floyd. Assigning meanings to programs. *Mathematical aspects of computer science*, 19(19-32):1, 1967.

[11] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[12] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[13] T. Hoare. Unification of theories: A challenge for computing science. *Recent Trends in Data Type Specification*, pages 49–57, 1996.

[14] M. Huisman and B. Jacobs. Java program verification via a hoare logic with abrupt termination. *Fundamental Approaches to Software Engineering*, pages 284–303, 2000.

[15] T. Kleymann. Hoare logic and auxiliary variables. *Formal Aspects of Computing*, 11(5):541–566, 1999.

[16] L. Lamport and F.B. Schneider. The hoare logic of csp, and all that. *ACM Transactions on Programming Languages and Systems*, 6(2):281 – 96, 1984.

[17] G. Leavens. An overview of larch/c++: Behavioral specifications for c++ modules. *Object-Oriented Behavioral Specifications*, pages 121–142, 1996.

[18] R. Mallon. The semantics, formal correctness and implementation of history variables in an imperative programming. *Masters Thesis (University of Canterbury)*, 2006.

[19] R. Milner. *A calculus of communicating systems*. Springer-Verlag New York, Inc., 1982.

[20] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[21] VA Nepomniaschy, IS Anureev, IV Dubranovsky, and AV Promsky. Towards c# program verification: a three-level approach. *Preprint*, 128, 2005.

[22] S. Owicki and D. Gries. Verifying properties of parallel programs: An axiomatic approach. *Communications of the ACM*, 19(5):279 – 285, 1976.

[23] E. Stark. A proof technique for rely/guarantee properties. In *Foundations of software technology and theoretical computer science*, pages 369–391. Springer, 1985.

[24] T. Takaoka. Parallel program verification with directed graphs. In *Proceedings of the 1994 ACM symposium on Applied computing*, pages 462–466. ACM, 1994.

[25] S. J. Weddell, T. Takaoka, T. Read, and R. Candy. Maximum subarray algorithms for use in optical and radio astronomy. *Proc. SPIE*, 8500(24), 2012. doi: 10.1117/12.928318.

A Notation

A.1 Logic Symbols

In order of precedence:

- $\neg$: logic 'not'.
- $\wedge$: logic 'and'.
- $\vee$: logic 'or'.
- $\rightarrow$: logic implies.
- $\exists$: existential quantifier.
- $\forall$: universal quantifier.
- $\Leftrightarrow$: logic equivalence.

A.2 Abbreviations

- $P[X/Y]$: The condition P with all instances of Y replaced by X.
- $\wedge_{i \in I} P_i$: The conjunction of all P_i for $i \in I$.
- $\texttt{MIN}_{\,i \in I}\, E$: The minimum value of the expression E for any $i \in I$.
- $\texttt{MAX}_{\,i \in I}\, E$: The maximum value of the expression E for any $i \in I$.
- I^2: The direct product of the set I with itself.