Grant Agreement: 287829

Comprehensive Modelling for Advanced Systems of Systems

COMPASS

CML Definition 3 – Hoare Logic

Deliverable Number: D23.4d

Version: 0.2

Date: 30 September 2013

Public Document

http://www.compass-research.eu

Contributors:

Samuel Canham, York
Jim Woodcock, York

Editors:

Samuel Canham, York
Jim Woodcock, York

Reviewers:

Document Reviewers

Document History

Ver	Date	Author	Description
0.1	06-09-2013	Samuel Canham	Initial document version
0.2	07-09-2013	Jim Woodcock	Minor revisions

Contents

Hoare Logic

1 Introduction

A proof system is a collection of axioms and inference rules in a formal language from which proofs can be constructed for a class of assertions in that language. Axioms are statements that are assumed to be true and should be tautologies of the logic. An axiom schema is a set of axioms parametrised by a predicate, used to construct specific axioms referring to arbitrary assertions. Inference rules, or deduction rules, consist of a premise, which must be satisfied to allow the rule to be applied, and a conclusion, which can advance a proof when the rule is applied.

A Hoare logic is a proof system used to construct proofs for Hoare triples. A Hoare triple is an assertion that states that one process satisfies, or refines, another more abstract specification. The specification is represented as a precondition and postcondition pair, and the triple is written as $\{P\}\ C\ \{Q\}$. This asserts that if the command C is executed in a state satisfying an assumption P and terminates, then it will terminate in a state satisfying the commitment Q. This allows the construction of correctness proofs for commands.

This section describes the foundations of a Hoare logic for ***CML***. The axiom schema and inference rules for that Hoare logic are described in the following two sections. The axioms of the Hoare logic will be used to construct Hoare triples for the simple non-composite operators. Given an arbitrary postcondition, they describe the weakest precondition that will establish it. The precondition can then be strengthened as necessary without affecting the truth of the triple. The inference rules will, given Hoare triples referring to the constituent parts, allow deduction of a Hoare triple referring to a composite operator applied to those parts. Its postcondition will be the strongest possible information deducible from the premise, and can be weakened as necessary.

Definition 1.1 *(Hoare triple)*

$\{P\}\ C\ \{Q\} \triangleq \mathbf{RT}(P \vdash Q) \sqsubseteq C$

In the definition of a Hoare triple, since the ***RT*** healthiness conditions are applied to $P \vdash Q$ before comparison with C, P and Q can each be replaced with more helpful forms that are equivalent under the healthiness conditions. It is therefore helpful to require that Q is free only in v, v', $wait'$ and tt', while P is free only in v and tt'. Additionally, P must be prefix closed;

$P = P \wedge \forall\, tt_0 \leq tt' \bullet P[tt_0/tt']$.

Finally, Q must conceal intermediate states;

$Q = (\exists\, v' \bullet Q) \triangleleft wait' \triangleright Q.$

Lemma 1.1 *(Hoare triple expansion)*

If A and P are healthy preconditions and B and Q are healthy postconditions then

$\{P\}\ \mathbf{RT}(A \vdash B)\ \{Q\} = \forall\, v, tt' \bullet P \Rightarrow A \wedge \forall\, v', wait' \bullet B \Rightarrow Q$

Proof 1

$\{P\}\ \mathbf{RT}(A \vdash B)\ \{Q\}$
$= \{$ *Definition of Hoare triple* $\}$
$\mathbf{RT}(P \vdash Q) \sqsubseteq \mathbf{RT}(A \vdash B)$
$= \{$ *Definition of refinement* $\}$
$[\mathbf{RT}(A \vdash B) \Rightarrow \mathbf{RT}(P \vdash Q)]$
$= \{$ *Definitions of* **RT1** *and* **RT3**, *predicate calculus*$\}$
$[\neg\, rt \leq rt' \vee ((A \vdash B) \Rightarrow (P \vdash Q)) \vee (wait \wedge SKIP)]$
$= \{$ *Definition of designs, predicate calculus* $\}$
$[P \Rightarrow (A \wedge (B \Rightarrow Q))]$
$= \{$ *Definition of universal quantification, predicate calculus* $\}$
$\forall\, v, tt' \bullet (P \Rightarrow (A \wedge \forall\, v', wait' \bullet (B \Rightarrow Q)))$

□

It would be desirable to use this result to deduce the most general precondition which establishes the postcondition, but this would require that the precondition were healthy. Applying the healthiness conditions results in the following corollary.

Lemma 1.2 *(Corollary) If Q is a healthy postcondition and*

$$P = A \wedge \forall\, v', wait', tt_0 \leq tt' \bullet (B \Rightarrow Q)[tt_0/tt']$$

then P is a healthy precondition and $\{P\}\ \mathbf{RT}(A \vdash B)\ \{Q\}$ can be deduced.

It is at first surprising that preconditions can refer to the variable tt', which is essentially a variable of the final program state. This is justifiable for two reasons. Firstly, the trace is accumulative. This means that it is impossible for a process to violate its precondition by engaging in particular events that change the value of tt' and hide the fact that this has occurred. Secondly, events can only occur in the trace if all processes that can observe those events, including the environment, agree. From the point of view of a particular process, all other processes can be viewed as part of the environment. A precondition regarding tt' can then be viewed as a contract between the process and its environment by which the process guarantees it will terminate and satisfy its postcondition provided the environment does not force it to behave in a certain way. Preconditions and postconditions can then be viewed as rely and guarantee conditions. This idea will be illustrated with some examples.

2 Constant operators

This section describes axiom schema for the constant processes in CML, such as *SKIP* and *STOP*. They can be calculated using the rule given in the previous section. In each case, we calculate the weakest precondition for the process to establish an arbitrary healthy postcondition. This weakest precondition will be healthy; it will be prefix closed and depend only on v and tt'.

The weakest precondition to establish a particular postcondition only needs to hold for values of tt' which can actually be observed of the process when it does not diverge. For example, the only trace that can be observed of the assignment process is the empty trace, so the process is not obliged to establish anything if an event has been observed or time has passed resulting in a non-empty trace. Consequently, each precondition will consist of a number of implications. Each of these will assert that the postcondition is established by the process in a particular state whenever the trace corresponding to that state is observed.

If an assignment $(v := e)$ is started in a stable state with a predecessor that is not waiting then the only observable trace is the empty trace. There is only one corresponding observation that can be made of it: $v' = e \wedge \neg\, wait'$. The terminating process *SKIP* will have exactly the same weakest precondition with e equal to the initial state v, since it is a special case of assignment.

Lemma 2.1 *(Axiom schema for assignment)*

$$\frac{true}{\{tt' = \langle\rangle \Rightarrow Q[e, false/v', wait']\}\ (v := e)\ \{Q\}}$$

Lemma 2.2 *(Axiom schema for termination)*

$$\frac{true}{\{tt' = \langle\rangle \Rightarrow Q[v, false/v', wait']\}\ SKIP\ \{Q\}}$$

The observable traces of the deadlocked process *STOP* are precisely those that satisfy $events(tt') = \langle\rangle$. The only observation that can be made of it is that it is always waiting, so the state will always be concealed.

Lemma 2.3 *(Axiom schema for deadlock)*

$$\frac{true}{\{events(tt') = \langle\rangle \Rightarrow Q[true/wait']\}\ STOP\ \{Q\}}$$

The prefixed termination $a \rightarrow SKIP$ has two distinct observable states, depending on whether or not the event a has been observed. In either case, a will not have been refused in any trace observed of the process. If the event has not been observed, the trace will additionally satisfy $events(tt') = \langle\rangle$ and the process will be waiting. If, however, the event has been observed, then it will be the last element of the trace, and then the process will not be waiting and will have the same final state as its initial state.

Lemma 2.4 *(Axiom schema for prefixed termination)*

$$\frac{true}{\left\{ \begin{array}{l} \left(\begin{array}{l} a \notin refusals(tt') \wedge \\ events(tt') = \langle\rangle \end{array} \right) \Rightarrow Q[true/wait'] \wedge \\ \left(\begin{array}{l} a \notin refusals(tt') \wedge \\ tt' = idleprefix(tt') \frown \langle a \rangle \end{array} \right) \Rightarrow Q[v, false/v', wait'] \end{array} \right\} \; a \rightarrow SKIP \; \{Q\}}$$

The divergent process *CHAOS* has no non-divergent behaviour, and so can never establish any postcondition.

Lemma 2.5 *(Axiom schema for divergence)*

$$\frac{true}{\{false\} \; CHAOS \; \{Q\}}$$

3 Composite operators

We now wish to calculate a set of deduction rules for the composite operators. The premise for each rule will be that Hoare triples have been observed for each of the component operators. The conclusion will be the exact information that can be inferred about the composite operator from only the information in the premise, and its structure. This can be obtained by replacing each process with the process formed by applying the healthiness conditions to the design formed from the precondition and postcondition of the premise Hoare triple.

Lemma 3.1 *(Inference rules for composite operators)*

$$\frac{\forall\, i \leq n \bullet \{P_i\} \; C_i \; \{Q_i\}}{\left\{ \neg \; X \left(\begin{array}{l} \mathbf{RT}(P_1 \vdash Q_1), \\ ..., \\ \mathbf{RT}(P_n \vdash Q_n) \end{array} \right)_f^f \right\} \; X(C_1, ..., C_n) \; \left\{ X \left(\begin{array}{l} \mathbf{RT}(P_1 \vdash Q_1), \\ ..., \\ \mathbf{RT}(P_n \vdash Q_n) \end{array} \right)_f^t \right\}}$$

where X is an operator taking n input operators.

The simplest application of this rule is in the case of disjunction. Taking $n = 2$ and $X = C_1 \vee C_2$, we replace C_1 by its generalisation $RT(P_1 \vdash Q_1)$, and similarly for C_2. The precondition of the resulting construct is $P_1 \wedge P_2$, and the postcondition is $Q_1 \vee Q_2$. These form the expected Hoare triple for disjunction.

3.1 Inference rules for Sequential Composition and related operators

The following four operators are derived from sequential composition and has an inference rule with a similar form. In each case, there are three possible states in which conditions must hold. The precondition of each Hoare triple has a component establishing the assumption of the first part of the composition and a component asserting that in any intermediate state the precondition of the second is established. Each postcondition states that either the whole process has terminated in some way, or that the first component has terminated but the condition to trigger the second component has not yet occurred.

Sequential Composition

This pattern can be seen in simple sequential composition. In order for the sequential composition to establish a postcondition, it must be that the precondition of the first process is established, and that it is impossible for its postcondition to fail to establish the precondition of the second process if it terminates. The postcondition established by the sequential composition will either be the postcondition of the first process if it hasn't terminated, or the composition of the two postconditions, if the first terminates.

Lemma 3.2 *(Inference rule for Sequential Composition)*

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 \wedge \neg\,(Q_1[false/wait'];\ \neg\ P_2)\}\ C_1;\ C_2\ \left\{ \begin{array}{l} Q_1[true/wait'] \wedge wait' \vee \\ Q_1[false/wait'];\ Q_2 \end{array} \right\}}$$

Prefixing

A prefixed process will fail to establish a postcondition if the process would fail when started after the prefixing event has occurred. The prefixed process will either establish the composition of the event and the process postcondition if the event occurs, or it will wait for the event if it has not occurred.

Lemma 3.3 *(Inference rule for Prefixing)*

$$\frac{\{P\}\ C\ \{Q\}}{\left\{ \begin{array}{l} \left(\begin{array}{l} a \notin refusals(tt') \wedge \\ \langle a \rangle \leq events(tt') \end{array} \right) \Rightarrow \\ P[idlesuffix(tt')/tt'] \end{array} \right\}\ a \rightarrow C\ \left\{ \begin{array}{l} \left(\begin{array}{l} events(tt') = \langle\rangle \wedge \\ a \notin refusals(tt') \wedge \\ wait' \end{array} \right) \vee \\ \left(\begin{array}{l} a \notin refusals(idleprefix(tt')) \wedge \\ \langle a \rangle \leq events(tt') \wedge \\ Q[idlesuffix(tt')/tt'] \end{array} \right) \end{array} \right\}}$$

Timeout

The timeout rule demonstrates all of the potential components of a sequential composition rule. The fragment $\#(idleprefix(tt')) < n \Rightarrow P_1$ requires that, unless n time units have passed without event, the precondition of the first component will hold, and the fragment $\neg\ ((Q_1 \wedge events(tt') = \langle\rangle \wedge \#tt' = n)\,;\ (wait \wedge \neg\ P_2))$ asserts that every state in which the first component is timed out after n time units without engaging in a visible event or terminating, the precondition of the second component must hold. In the postcondition, the fragment $(Q_1 \wedge \#(idleprefix(tt')) < n)$ asserts that unless n time units have passed without a visible event, the postcondition of the first component will hold with the second fragment $((Q_1 \wedge events(tt') = \langle\rangle \wedge \#tt' = n)\,;\ (wait \wedge Q_2))$ asserts that the first component held for the first n time units then was successfully timed out, after which the postcondition of the second component was established.

Lemma 3.4 *(Inference rule for Timeout)*

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\left\{\begin{array}{l} \#(idleprefix(tt')) < n \Rightarrow P_1 \wedge \\ \neg \left(\begin{pmatrix} Q_1 \wedge \\ events(tt') = \langle\rangle \wedge \\ \#tt' = n \end{pmatrix}; \begin{pmatrix} (wait \wedge \neg\ P_2) \end{pmatrix}\right) \end{array}\right\}\ C_1 \rhd^n C_2\ \left\{\begin{array}{l} (Q_1 \wedge \#(idleprefix(tt')) < n) \\ \vee \left(\begin{pmatrix} Q_1 \wedge \\ events(tt') = \langle\rangle \\ \wedge \#tt' = n \end{pmatrix}; \begin{pmatrix} (wait \wedge Q_2) \end{pmatrix}\right) \end{array}\right\}$$

Interrupt

The first component of the precondition of the interrupt operator describes the case when the the interrupt has not occurred. In this case the initial events of the second process will not have been offered in the trace. The process will share its precondition with the first process. The second component states that the first process cannot behave in any way that would violate the precondition of the second process if it could still be interrupted. The only way it could violate the following precondition would be to terminate.The postcondition established will either be the postcondition of the first process if the initials of the second are not offered, or the composition of that postcondition together with the postcondition of the second.

Lemma 3.5 *(Inference rule for Interrupt)*

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\left\{\begin{array}{l} notoffered(initials(C_2), tt') \Rightarrow P_1 \wedge \\ \neg \left(\begin{pmatrix} Q_1 \wedge wait' \wedge \\ notoffered(initials(C_2), tt') \end{pmatrix}; \begin{pmatrix} (\neg\ P_2 \wedge idleprefix(tt') = \langle\rangle) \end{pmatrix}\right) \end{array}\right\} C_1 \vartriangle C_2 \left\{\begin{array}{l} (Q_1 \wedge notoffered(initials(C_2), tt')) \\ \vee \begin{pmatrix} Q_1 \wedge wait' \wedge \\ notoffered(initials(C_2), tt') \end{pmatrix}; \\ (Q_2 \wedge idleprefix(tt') = \langle\rangle) \end{array}\right\}$$

3.2 Inference rules for Choice and related operators

Again, each of the three choice rules has a very similar form. Each precondition says that the assumptions of any choice that could occur must be satisfied. The postconditions state that the commitment of whichever choice resolves will apply.

Conditional

The most simple case is the conditional, since the resolution of its choice depends entirely on the condition, which can be evaluated in the initial state. The precondition and the postcondition are each conditionals composed of the preconditions and postconditions respectively of the two processes.

Lemma 3.6 *(Inference rule for Conditional)*

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 \lhd b \rhd P_2\}\ C_1 \lhd b \rhd C_2\ \{Q_1 \lhd b \rhd Q_2\}}$$

Internal Choice

An internal choice can fail to establish its postcondition if either of its branches can, so it will require both of their preconditions to hold. It could establish either of their postconditions as its own.

Lemma 3.7 *(Inference rule for Internal Choice)*

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 \wedge P_2\}\ C_1 \sqcap C_2\ \{Q_1 \vee Q_2\}}$$

External Choice

External choice can fail to establish a postcondition only if one of its branches can. It will otherwise establish one of their postconditions. Additionally, it will establish a condition consistent with both branches of the choice up to the point where the choice occurred. This condition is established by insisting that the postcondition of each branch must have been feasible at the point before the first event, if any, was observed. Since each branch can terminate independently and operates on its own private state before the choice is resolved, the values of $wait'$ and v' must both be quantified. However, if the process has terminated and no events were observed, the choice must have been resolved by the termination, so at least one of the branches must have been prepared to terminate at this point. This is enforced by insisting that at least one of the quantified values of $wait'$ is false in this case.

Lemma 3.8 *(Inference rule for External Choice)*

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 \wedge P_2\}\ C_1 \Box C_2\ \left\{ \begin{array}{c} (Q_1 \vee Q_2) \wedge \exists\, v_0, v_1, wait_0, wait_1 \bullet \\ (events(tt') = \langle\rangle \wedge \neg\, wait') \Leftrightarrow (\neg\, wait_0 \vee \neg\, wait_1) \wedge \\ \left(\begin{array}{c} Q_1[v_0, wait_0/v', wait'] \wedge \\ Q_2[v_1, wait_1/v', wait'] \end{array} \right)[idleprefix(tt')/tt'] \end{array} \right\}}$$

3.3 Inference rule for Parallel Composition

A parallel composition will not diverge provided that no trace can be extracted from the observed trace which violates the precondition of either constituent process. It will partition the trace into two components and establish the postconditions of each constituent process on one of those components. Additionally, since $wait'$ is true if either $wait_1$ or $wait_2$, the values of $wait'$ for the constituent processes, are true, the process only terminates if both constituents do.

Lemma 3.9 *(Inference rule for Parallel Composition)*

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\left\{ \begin{array}{l} \forall\, tt_1, tt_2 \bullet tt' \in (tt_1\ ||_A\ tt_2) \Rightarrow \\ P_1[tt_1/tt'] \wedge P_2[tt_2/tt'] \end{array} \right\}\ C_1\ ||_A\ C_2\ \left\{ \begin{array}{c} \exists\, tt_1, tt_2, wait_1, wait_2 \bullet \\ wait' = wait_1 \vee wait_2 \wedge \\ tt' \in (tt_1\ ||_A\ tt_2) \Rightarrow \\ \left(\begin{array}{c} Q_1[tt_1, wait_1/tt', wait'] \wedge \\ Q_2[tt_2, wait_2/tt', wait'] \end{array} \right) \end{array} \right\}}$$

3.4 Inference rule for Hiding

This operator will diverge if there is a way to hide events in A from the observable state while making them urgent which violates the precondition of the process without the hiding. Otherwise, it will establish the state that would have been observed without the hiding, with all events in A hidden and urgent.

Lemma 3.10 *(Inference rule for Hiding)*

$$\frac{\{P\}\ C\ \{Q\}}{\left\{\begin{array}{l}\forall\, u \bullet \\ \left(\begin{array}{l} A \subseteq \bigcap ref\,sduring(u) \wedge \\ u \backslash A = tt' \end{array}\right) \Rightarrow \\ P[u/tt'] \end{array}\right\}\ C \backslash A\ \left\{\begin{array}{l}\exists\, u \bullet \\ \left(\begin{array}{l} A \subseteq \bigcap ref\,sduring(u) \wedge \\ u \backslash A = tt' \end{array}\right) \Rightarrow \\ Q[u/tt'] \end{array}\right\}}$$

4 Recursion

Recursion is the most complex operator in the language and will require special treatment. The rule presented below will hold only for operators in which the recursion is stable at the point of observation. By this, we mean that the recursion has occurred a finite number of times. Consequently, non-terminating loops can be modelled provided they do not occur unboundedly often in any time interval. This condition is similar to Zeno-freedom, and would be guaranteed if each time the recursion triggered an event was generated.

In the UTP treatment of recursion, this occurs precisely when the recursion does not diverge. This is shown to be the case when there is a unique fixed point to the recursion, which is both the strongest and weakest fixed point. In [HH98], the following technique is given to find a condition $B(F)$ which guarantees that the recursion $\mu\, F$ terminates. It works by calculating a sequence of predicates called an approximation chain, E_n. The nth term in this sequence describes the case when less than n loop iterations have occurred. Consequently, $E_0 = \mathit{false}$ and $\forall\, n \bullet E_n \Rightarrow E_{n+1}$.

For an arbitrary predicate X, a series of approximations to $F(X)$ is given by $F(X) \wedge E_n$, in which less than n loops of F have occurred. In order to determine a unique fixed point, F must be E-constructive; each approximation of $F(X)$ must be calculable in terms of lower order approximations. This is established by the condition

$$F(X) \wedge E_{n+1} = F(C \wedge E_n) \wedge E_{n+1}$$

B is then the disjunction of the E_n, $\bigvee_i E_i$, and gives the weakest non-divergence condition, $\sqcap\{b \mid b \Rightarrow \mu\, F = \nu F\}$.

We use this technique to calculate $B(F)$ for use in the precondition of the Hoare triple. Since stable observations can not be made in diverging states, we will require that the precondition P of the Hoare triple satisfies $P \Rightarrow B(F)$.

Since F is assumed to be ***RT***-healthy, there exists F_1 and F_2 such that

$$F(\mathbf{RT}(P \vdash Q)) = \mathbf{RT}(F_1(P, Q) \vdash F_2(P, Q))$$

If our fixed point is a strongest fixed point, we know further that $F(\textbf{\textit{RT}}(P \vdash Q)) \Rightarrow \textbf{\textit{RT}}(P \vdash Q)$, which can be expanded to show that

$$P \Rightarrow (B \wedge F_1(P, Q) \wedge (F_2(P, Q) \Rightarrow Q))$$

Combining this with the result above, and taking fixed points, gives the following result.

Lemma 4.1 *(Axiom schema for Recursion)*

$$\frac{\mathit{true}}{\{\mu\, X \bullet B(F) \wedge F_1(X, Q) \wedge \forall\, v', \mathit{wait}' \bullet F_2(X, Q) \Rightarrow Q\}\ \ \mu\, X \bullet F(X)\ \{Q\}}$$

Unlike the previous triple in this section, this operator does not calculate weakest preconditions. The term involving F_1 is required in each iteration of the loop to prevent divergence, but will also try to establish additional properties of the F_2 term in intermediate states.

While loop example

As an example, consider the while loop. The while loop $b * C$ is equal to $\mu\, X \bullet C;\ X \lhd b \rhd \mathit{SKIP}$, where b is a condition on v and tt' only.

Let $b = v \neq 0$ and $C = (v := v - 1)$. This describes the while loop which repeatedly reduces a variable by 1 until it reaches 0, and will enter a non-terminating loop if it is started with a negative value. First, the function F must be calculated:

$$F(\textbf{\textit{RT}}(P \vdash Q)) = \textbf{\textit{RT}}\left(v \neq 0 \Rightarrow P[v-1/v] \vdash \left(\begin{array}{l}(v' = 0 \wedge tt' = \langle\rangle \wedge \neg\ \mathit{wait}') \\ \lhd v = 0 \rhd Q[v-1/v]\end{array}\right)\right)$$

and so

$$F_1(P, Q) = v \neq 0 \Rightarrow P[v-1/v]$$

and

$$F_2(P, Q) = v' = 0 \wedge tt' = \langle\rangle \wedge \neg\ \mathit{wait}' \lhd v = 0 \rhd Q[v-1/v]$$

Since there are no stable waiting states in this process, an observation can only be made on termination. The process will terminate within n steps provided the initial value of v was less than n. Consequently, an approximation chain is given by $E_i = 0 \leq v < i$, and hence $B(F) = 0 \leq v$. The nth approximation to the process is $0 \leq v < n \wedge (v := 0)$.

We can now calculate the Hoare triple precondition P. This will be done by induction on v. Let $P_n = (v = n) \wedge P$. We will first show that

$$P_n = \left(\begin{array}{c} v = n \wedge (tt' = \langle\rangle \Rightarrow Q[0, 0, \mathit{false}/v, v', \mathit{wait}']) \wedge \\ \forall\, v', \mathit{wait}', n < v \bullet Q[n/v] \Rightarrow Q[n+1/v] \end{array}\right)$$

In the base case, $v = 0$ and $P_0 = (v = 0) \wedge (tt' = \langle\rangle \Rightarrow Q[0, \mathit{false}/v', \mathit{wait}'])$ which has the correct form.

Suppose that P_n is known and $v = n + 1$. Then
$P_{n+1} = P_n[v - 1/v] \wedge \forall\, v', wait' \bullet Q[v - 1/v] \Rightarrow Q$. By the induction hypothesis,

$$P_{n+1} = \left(\begin{array}{c} v - 1 = n \wedge (tt' = \langle\rangle \Rightarrow Q[0, 0, false/v, v', wait']) \wedge \\ \forall\, v', wait', n < v - 1 \bullet Q[n/v] \Rightarrow Q[n + 1/v] \wedge \\ \forall\, v', wait' \bullet Q[v - 1/v] \Rightarrow Q \end{array} \right)$$

.

The final result for P is then

$$P = \left(\begin{array}{c} v \geq 0 \wedge (tt' = \langle\rangle \Rightarrow Q[0, 0, false/v, v', wait']) \wedge \\ \forall\, v', wait', n < v \bullet Q[n/v] \Rightarrow Q[n + 1/v] \end{array} \right)$$

While this precondition is sufficient to establish the condition Q, it is not the weakest such precondition. By calculation, it can be seen that

$$(v \neq 0 * v := v - 1) = ((v := 0) \lhd v >= 0 \rhd CHAOS)$$

This evaluates to $\boldsymbol{RT}(v \geq 0 \vdash v' = 0 \wedge tt' = \langle\rangle \wedge \neg\ wait')$, and so the weakest precondition to establish Q is

$$v \geq 0 \wedge (tt' = \langle\rangle \Rightarrow Q[0, false/v', wait'])$$

This condition is guaranteed by P, which explicitly requires $Q[0, 0, false/v, v', wait']$ then builds up the value of v in the term inductively the required number of times in $\forall\, v', wait', n < v \bullet Q[n/v] \Rightarrow Q[n + 1/v]$. The additional restriction is generated by the coupling in the precondition of information regarding non-divergence in each iteration and information regarding establishing the postcondition. The two preconditions will agree when the postcondition is downward closed in v – if it holds for any particular value of v, it must also hold for all smaller non-negative values.

For example, consider the postcondition $wait' \vee (v < 5 + v')$. This is downward closed, and in both cases gives rise to the triple

$$\{0 \leq v < 5\}\ (v \neq 0 * v := v - 1)\ \{wait' \vee (v < 5 + v')\}$$

However, they disagree on the postcondition $\exists\, k \bullet v + v' = k^2$, which will be the case only when the initial value of v is not equal to 0.

$$\{0 \leq v \leq 1\}\ (v \neq 0 * v := v - 1)\ \{\exists\, k \bullet v + v' = k^2\}$$

$$\{\exists\, k \bullet v = k^2\}\ ((v := 0) \lhd v >= 0 \rhd CHAOS)\{\exists\, k \bullet v + v' = k^2\}$$

Bibliography

[HH98] C. A. R. Hoare and Jifeng He. *Unifying Theories of Programming*. Prentice-Hall, 1998.