On Correctness of Imperative Programs

—

Precondition Calculation (DRAFT)*

R.D. Arthan

2 October 2011

1 Introduction

This note is the second in a planned series concerned with specification via pre- and post-conditions in a partial correctness setting. It applies the general notion of specification and refinement given in the first note in the series [1] to give a notion of correctness for a simple imperative programming language, where programs and program fragments may be equipped with Floyd-Hoare style specification annotations.

In the sequel, we present a simple formal model in ProofPower-Z of a refinement notation comprising a miniature, but complete, imperative programming language annotated with formal specifications; the semantics of that programming language and the notion of correctness relative to the specification annotations is defined. A semantic model of a verification condition generator (or pre-condition calculator) is given which can be proved to be sound with respect both to the programming language semantics and to the intensional semantics of the specification annotations.

The Z specification uses infix notation for the following relation and function symbols (in addition to those defined in the first note):

$$\vdash \textit{relation} \quad \_ \models \_,\ \_ \perp \_$$

$$\vdash \textit{function 60 rightassoc} \quad \_ \rhd_* \_,\ \_ \trianglerighteq_* \_$$

The specification contains a number of conjectures. All of these have been proved with ProofPower and the resulting theory listing is included as an appendix to this document. An index to the Z specification is given at the end of the document.

Please note this is work-in-progress and the current version of this note lacks any serious attempt to cite the literature or compare the results with other work. Cousot's article on

program verification in the Handbook of Theoretical Computer Science gives an appropriate survey but tends to veer towards the theoretical. In particular, completeness results are not particularly relevant in practical applications — they just give you warm feelings that if you give sufficiently strong specifications, you will get weakest, rather than just sufficient pre-conditions. The relational semantics given here goes back at least as far as early work of Hoare.

2 States and State Transformers

The internal structure of states is not relevant to our purposes. For a conventional imperative programming language, the states will be assignments of values to program variables. We just introduce a given set to represent the states:

$$[\boldsymbol{STATE}]$$

The commands in our programming language will denote *state transformers*. A state transformer is just a relation on states:

$$\boldsymbol{STATE\_TRANSFORMER} \mathrel{\widehat=} STATE \leftrightarrow STATE$$

We think of a state transformer t as *responding* to a state, the *before-state*, s in its domain by non-deterministically selecting some *response* or *after-state*, which is a state s' such that $(s, s') \in t$.

3 Predicates and Specifications

We instantiate the notions of predicates, pre- post-conditions and specifications from the first document in the series:

$$\boldsymbol{P\_PRED} \mathrel{\widehat=} PRED[STATE]$$

$$\boldsymbol{P\_PRE\_COND} \mathrel{\widehat=} PRE\_COND[STATE]$$

$$\boldsymbol{P\_POST\_COND} \mathrel{\widehat=} POST\_COND[STATE, STATE]$$

$$\boldsymbol{P\_SPEC} \mathrel{\widehat=} SPEC[STATE, STATE]$$

4 Programs

Our notion of program has five syntactic categories:

Atom	Some set of primitive operations on the state.
Seq	Sequential composition
If	If-then-else
While	While-loop
Spec	Programs with specification annotations

The following free type gives the abstract syntax of programs, in which we mingle semantic and syntactic concepts to simplify later work. We also use a tree structure rather than a linear list for sequential composition, since that is semantically harmless, and, again, helps to keep things simple later on.

$$
\begin{array}{ll}
\boldsymbol{PROG} ::= & \\
 & \boldsymbol{Atom}\ (STATE\_TRANSFORMER) \\
| & \boldsymbol{Seq}\ (PROG \times PROG) \\
| & \boldsymbol{If}\ (P\_PRED \times PROG \times PROG) \\
| & \boldsymbol{While}\ (P\_PRED \times PROG) \\
| & \boldsymbol{Spec}\ (P\_SPEC \times PROG)
\end{array}
$$

In a typical imperative language, the atoms might be the denotations of assignment statements and procedure calls. In the Compliance Notation, the denotation of a procedure call is effectively represented by an instance of the formal specification appearing in the procedure header.

The following function gives the semantics of this notion of a program. The semantic value of a program is a state transformer. In the semantics, the specification annotations are just ignored — it is the actual code that determines the semantics, not our aspirations for it.

$$
\boldsymbol{semantics} : PROG \rightarrow STATE\_TRANSFORMER
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\forall t : STATE\_TRANSFORMER;\ p_1,\ p_2 : PROG;\ c : P\_PRED;\ s : P\_SPEC \bullet} \\
 & semantics\ (Atom\ t) = t \\
\land & semantics\ (Seq(p_1,\ p_2)) = semantics\ p_1 \mathbin{;} semantics\ p_2 \\
\land & semantics\ (If(c,\ p_1,\ p_2)) = (c \lhd semantics\ p_1) \cup (c \mathbin{⩤} semantics\ p_2) \\
\land & semantics\ (While(c,\ p_1)) = (c \lhd semantics\ p_1)^* \mathbin{⩥} c \\
\land & semantics\ (Spec(s,\ p_1)) = semantics\ p_1
\end{array}
$$

It is in the above that the convenience of dealing with partial correctness begins to become apparent. The semantic equation for a while-loop says that the body of the loop is to be executed repeatedly in states satisfying the predicate c until a state which does not satisfy c is reached. If this fails to terminate the result is just the empty relation: we are under no obligation to assign any more complex notion of meaning to the non-terminating execution.

The partial semantics also embraces in an abstract way the possibility of the program failing gracefully. Throughout the sequel, when we talk about non-termination, we include the possibility that via some exception-raising mechanism that is outside the scope of the present model, execution of a command may result in some kind of abnormal termination which is handled properly in the physical environment in which the program is executed.

As a simple check on our definitions and to give a first exercise in reasoning by induction over the syntax of programs, let us state as a conjecture the claim that the function on programs which simply strips out all the specifications is semantics-preserving:

$$
\begin{array}{l}
\mathit{strip\_specs\_cnj}\ ?\vdash \\
\quad \forall \quad \mathit{strip\_specs}\colon \mathit{PROG} \to \mathit{PROG} \\
\quad \mid \quad \forall t : \mathit{STATE\_TRANSFORMER};\ p_1,\ p_2 : \mathit{PROG};\ c : \mathit{P\_PRED};\ s : \mathit{SPEC} \bullet \\
\qquad \mathit{strip\_specs}\ (\mathit{Atom}\ t) = (\mathit{Atom}\ t) \\
\quad \wedge \quad \mathit{strip\_specs}\ (\mathit{Seq}(p_1,\ p_2)) = \mathit{Seq}(\mathit{strip\_specs}\ p_1,\ \mathit{strip\_specs}\ p_2) \\
\quad \wedge \quad \mathit{strip\_specs}\ (\mathit{If}(c,\ p_1,\ p_2)) = \mathit{If}(c,\ \mathit{strip\_specs}\ p_1,\ \mathit{strip\_specs}\ p_2) \\
\quad \wedge \quad \mathit{strip\_specs}\ (\mathit{While}(c,\ p_1)) = \mathit{While}(c,\ \mathit{strip\_specs}\ p_1) \\
\quad \wedge \quad \mathit{strip\_specs}\ (\mathit{Spec}(s,\ p_1)) = \mathit{strip\_specs}\ p_1 \\
\quad \bullet\ \forall p : \mathit{PROG} \bullet\ \mathit{semantics}(\mathit{strip\_specs}\ p) = \mathit{semantics}\ p
\end{array}
$$

5 Program Correctness

For a program to be correct every part of it that has a specification must certainly satisfy that specification, which in our setting means that the semantic value of the program must be a refinement of the given specification annotation. We write $p \models s$ to mean that program p satisfies specification s.

$$
\begin{array}{l}
\_ \models \_ : \mathit{PROG} \leftrightarrow \mathit{SPEC} \\
\hline
\forall\ \mathit{prog} : \mathit{PROG};\ \mathit{prec} : \mathit{P\_PRE\_COND};\ \mathit{postc} : \mathit{P\_POST\_COND} \bullet \\
\quad \mathit{prog} \models (\mathit{prec},\ \mathit{postc}) \Leftrightarrow (\mathit{prec},\ \mathit{postc}) \sqsubseteq (\mathit{prec},\ \mathit{semantics}\ \mathit{prog})
\end{array}
$$

However, a good intuitive notion of correctness also requires the pre-condition of each specification in the program to be satisfied whenever the relevant part of the program is executed. For example, every part of a program that has a specification might satisfy its specification, but, the program might still include a reachable specification with an empty post-condition: clearly, this part of the program is "correct, but for the wrong reasons".

For the want of a better name, if p is a program and c is a set of states, we will say that p is *upright* on c iff. no pre-condition in p will be violated when p is executed in a starting state in c. We write $p \perp c$ when this holds.

$$
\begin{array}{l}
\_ \perp \_ : \mathit{PROG} \leftrightarrow \mathit{PRED} \\
\hline
\forall t : \mathit{STATE\_TRANSFORMER};\ \mathit{postc} : \mathit{P\_POST\_COND};\ p_1,\ p_2 : \mathit{PROG}; \\
\quad c_1,\ c_2,\ \mathit{prec} : \mathit{P\_PRE\_COND} \bullet \\
\qquad ((\mathit{Atom}\ t) \perp c_1) \\
\wedge \quad ((\mathit{Seq}(p_1,\ p_2) \perp c_1) \Leftrightarrow p_1 \perp c_1 \wedge p_2 \perp \mathit{semantics}\ p_1\ (\!|c_1|\!)) \\
\wedge \quad ((\mathit{If}(c_2,\ p_1,\ p_2) \perp c_1) \Leftrightarrow p_1 \perp c_1 \cap c_2 \wedge p_2 \perp c_1 \setminus c_2) \\
\wedge \quad ((\mathit{While}(c_2,\ p_1) \perp c_1) \Leftrightarrow p_1 \perp c_1 \cap c_2 \wedge p_1 \perp \mathit{semantics}\ p_1\ (\!|c_1 \cap c_2|\!) \cap c_2) \\
\wedge \quad ((\mathit{Spec}((\mathit{prec},\ \mathit{postc}),\ p_1) \perp c_1) \Leftrightarrow c_1 \subseteq \mathit{prec} \wedge p_1 \perp c_1)
\end{array}
$$

Uprightness enjoys the following two useful properties:

$$upright\_mono\_cnj \mathrel{?\vdash} \forall\, prog : PROG;\ c_1,\ c_2 : P\_PRED\bullet\ prog \perp c_1 \land c_2 \subseteq c_1 \Rightarrow prog \perp c_2$$

$$upright\_cup\_cnj \mathrel{?\vdash} \forall\, prog : PROG;\ c_1,\ c_2 : P\_PRED\bullet\ prog \perp c_1 \land prog \perp c_2 \Rightarrow prog \perp c_1 \cup c_2$$

6 Pre-condition Calculation

The simplest view of a pre-condition calculator would be a predicate transformer: a function that takes a program and a post-condition given as a predicate on the final state as its argument and returns a predicate that, we hope, gives a pre-condition which will guarantee achievement of the post-condition.

When one looks into the details of defining a useful pre-condition calculator one finds that it is really has to be a post-condition transformer: in order to give an algorithm that works by recursion over the structure of a program, one must work with post-conditions, i.e., relations on states, rather than just states. The domain of all these relations corresponds to an appropriate initial or intermediate program state throughout the calculation. The calculation essentially works backwards through the program reversing the effects of program execution.

$$\boldsymbol{PREC\_CALC} \mathrel{\widehat=} PROG \times P\_POST\_COND \to P\_POST\_COND$$

In the simple view as a predicate transformer, a pre-condition calculator would be sound if the pre-condition it returns for any program and post-condition is such that execution of the program subject to the pre-condition refines the specification statement formed from the returned pre-condition and the given post-condition. In the post-condition transformer view that we are taking, our soundness criterion is formalised in the following definition which will ensure that pre-condition calculation interacts nicely with sequential composition.

$$\boldsymbol{sound\_prec\_calc} : \mathbb{P} PREC\_CALC$$

$$\begin{array}{ll} & \forall pc : PREC\_CALC\bullet \\ & \quad pc \in sound\_prec\_calc \\ \Leftrightarrow & \quad (\forall prog : PROG;\ postc,\ postc' : P\_POST\_COND \\ & \quad \mid \quad postc = pc(prog,\ postc') \\ & \quad \bullet \quad (dom\ postc,\ postc') \sqsubseteq (dom\ postc,\ postc \fatsemi semantics\ prog)) \end{array}$$

When one has completed the pre-condition calculation both the domain and the range of the resulting relation now correspond to the initial program state as well, and one may extract the desired post-condition by intersecting with the identity relation. Using this idea, we can formulate, the (not very deep) conjecture that if the program has a specification at the top

level and if the calculated pre-condition contains the pre-condition of the specification, then the program satisfies the specification:

$$
\begin{array}{l}
prec\_calc\_sat\_cnj \ ?\vdash \\
\quad \forall\ pc : PREC\_CALC;\ c\_prec,\ s\_prec\colon P\_PRE\_COND; \\
\quad\ \ s\_postc : P\_POST\_COND;\ p : PROG \\
\quad \mid \quad pc \in sound\_prec\_calc \\
\quad \land \quad c\_prec = dom\ (pc(Spec((s\_prec,\ s\_postc),\ p),\ s\_postc) \cap (id\ STATE)) \\
\quad \land \quad s\_prec \subseteq c\_prec \\
\quad \bullet \quad p \models (s\_prec,\ s\_postc)
\end{array}
$$

We can easily exhibit a sound but not at all useful, pre-condition calculator, which simply offers the empty set as a pre-condition which will achieve any desired post-condition:

$$
\begin{array}{l}
trivial\_prec\_calc\_sound\_cnj \ ?\vdash \\
\quad \forall\ pc : PREC\_CALC \\
\quad \mid \quad \forall prog : PROG;\ postc : P\_POST\_COND \bullet\ pc\ (prog,\ postc) = \varnothing \\
\quad \bullet \quad pc \in sound\_prec\_calc
\end{array}
$$

We will now give a model of a more useful pre-condition calculator. There are some preliminaries to take care of: the treatment of if-then-else is made more readable using the following two variants on the theme of range restriction and range anti-restriction:

$$
\begin{array}{l}
[X, Y] \\
\hline
\_ \rhd_* \_ : (X \leftrightarrow Y) \times \mathbb{P} Y \to (X \leftrightarrow Y); \\
\_ \rhd\!\!\!\rhd_* \_ : (X \leftrightarrow Y) \times \mathbb{P} Y \to (X \leftrightarrow Y) \\
\hline
\forall\ R : X \leftrightarrow Y;\ T : \mathbb{P}\ Y \bullet \\
\quad R \rhd_* T = R \sim (\!|\ Y \setminus T\ |\!) \lhd R \\
\land \quad R \rhd\!\!\!\rhd_* T = R \sim (\!|\ T\ |\!) \lhd R
\end{array}
$$

The useful pre-condition calculator will use a heuristic to propose a specification for the body of a while-loop. The soundness of the pre-condition calculator is independent of the heuristic — it has to be, because as the following loose specification shows, one possibility is that the specification is just lifted from the program without further analysis.

$$
\begin{array}{l}
\textbf{\textit{guess\_spec}} : PROG \to P\_SPEC \\
\hline
\forall s : P\_SPEC;\ p : PROG \bullet guess\_spec(Spec(s,\ p)) = s
\end{array}
$$

With the preliminaries in place, we can now define the useful pre-condition calculator, which we think of as pulling a post-condition backwards through a program transforming it as we go. In the definition, the various syntactic categories are dealt with as follows:

- A post-condition is pulled back through an atomic statement, by calculating the set of pairs (s, s') such that the response of the atom on s' is a response permitted by the post-condition on s.

- A post-condition is pulled back through the sequential composition of p_1 and p_2 in the obvious way: pull it back through p_2 and then pull the result back through p_1.

- A post-condition is pulled back through an if-then-else statement by pulling it back through the then- and else-parts of the statement. The overall result is then the union of these intermediate results after discarding all transitions which do not unambiguously belong to the if-part or the else-part.

- A post-condition is pulled back through a while loop by applying the heuristic to guess a specification for the body of the loop. The overall result is formed as a union of two parts.

  The first part of the union corresponds to states where the body of the loop is never execute and is just the appropriate restriction of the original post-condition.

  The second part corresponds to states where the body of the loop is executed at least once and is given as a set comprehension below. The set comprehension is empty unless three conditions are satisfied: *(i)* the condition of the while-loop must denote a set of states that are included in the guessed pre-condition; *(ii)* the pre-condition of the body must denote a set of states that satisfy the pre-condition resulting from pulling the guessed post-condition back through the body; and *(iii)* for each state satisfying the guessed pre-condition, the set of all states allowed by the guessed post-condition in response to this state which do not satisfy the loop condition must be contained in every possible response of the original post-condition.

- A post-condition is pulled back through a specification statement in much the same way as it is pulled back through an atomic statement treating the post-condition of the specification statement in the same way as the state transformer of the atom. The result is then filtered to remove all state transitions which do not unambiguously satisfy both the pre-condition of the specification statement and the pre-condition calculated from the body of the specification statement.

$$
\begin{array}{|l}
\textbf{\textit{prec\_calc}} : \mathit{PREC\_CALC} \\
\hline
\forall t : \mathit{STATE\_TRANSFORMER};\ \mathit{postc},\ \mathit{postc}_1 : \mathit{P\_POST\_COND}; \\
\ p_1,\ p_2 : \mathit{PROG};\ c : \mathit{P\_PRED}; \\
\ \mathit{prec}_1 : \mathit{P\_PRE\_COND};\ \mathit{body\_prec} : \mathit{P\_PRED};\ \mathit{body\_postc} : \mathit{P\_POST\_COND} \bullet \\
\\
\quad \mathit{prec\_calc}\ (\mathit{Atom}\ t,\ \mathit{postc}) = \{s,\ s' : \mathit{STATE} \mid t(\!|\{s'\}|\!) \subseteq \mathit{postc}(\!|\{s\}|\!)\} \\
\\
\land \quad \mathit{prec\_calc}\ (\mathit{Seq}(p_1,\ p_2),\ \mathit{postc}) = \mathit{prec\_calc}(p_1,\ \mathit{prec\_calc}(p_2,\ \mathit{postc})) \\
\\
\land \quad \mathit{prec\_calc}\ (\mathit{If}(c,\ p_1,\ p_2),\ \mathit{postc}) = \\
\qquad (\mathit{prec\_calc}(p_1,\ \mathit{postc}) \rhd_* c) \cup (\mathit{prec\_calc}(p_2,\ \mathit{postc}) \rhd\!\!=_* c) \\
\\
\land \quad ((\mathit{body\_prec},\ \mathit{body\_postc}) = \mathit{guess\_spec}\ p_1 \\
\quad \Rightarrow \quad \mathit{prec\_calc}\ (\mathit{While}(c,\ p_1),\ \mathit{postc}) = \\
\qquad\qquad \mathit{postc} \rhd\!\!= c \cup \\
\qquad\qquad \{\ ss' : \mathit{dom}\ \mathit{postc} \times c \\
\qquad\qquad \mid c \subseteq \mathit{body\_prec} \\
\qquad\qquad \land\ \mathit{body\_prec} \subseteq \mathit{dom}\ (\mathit{prec\_calc}(p_1,\ \mathit{body\_postc}) \cap (\mathit{id}\ \mathit{STATE})) \\
\qquad\qquad \land\ \mathit{dom}\ \mathit{postc} \times (\mathit{body\_postc}(\!|\mathit{body\_prec}|\!) \setminus c) \subseteq \mathit{postc}\ \}) \\
\\
\land \quad \mathit{prec\_calc}\ (\mathit{Spec}((\mathit{prec}_1,\ \mathit{postc}_1),\ p_1),\ \mathit{postc}) = \\
\qquad \{\ s : \mathit{STATE};\ s' : \mathit{STATE} \mid \mathit{postc}_1(\!|\{s'\}|\!) \subseteq \mathit{postc}(\!|\{s\}|\!)\ \} \rhd \\
\qquad (\mathit{prec}_1 \cap \mathit{dom}(\mathit{prec\_calc}(p_1,\ \mathit{postc}_1) \cap (\mathit{id}\ \mathit{STATE}))) \\
\end{array}
$$

Before stating some conjectures about the properties of the useful pre-condition calculator, some remarks about the definition and how it is actually realised in a practical system are in order.

A practical implementation can represent the post-condition being transformed as (the conceptual conjunction of) a finite set of syntactic predicates $\mathcal{P}_i(\vec{x_0}, \vec{x})$, where $\vec{x}$ represents some list of program variables and $\vec{x_0}$ represents a list of program variables decorated to distinguish them as initial variables (i.e., they refer to the before-state of the code being analysed). The pre-condition calculator will operate by syntactic transformations on these predicates which hold the initial variables fixed but may make substitutions to $\vec{x}$. At the beginning of the calculation, $\vec{x}$ refers to the final state of the program, and as the calculation works backwards through the code, the execution state referred to by $\vec{x}$ moves backwards in step.

The most primitive state-changing operation will be the atomic statements that represent program language assignments. Given an assignment, $v := e$, the requirements of the above formal definition are precisely met by substituting e for v in the $\mathcal{P}_i(\vec{x_0}, \vec{x})$. Here we are tacitly assuming that program variables and expressions have some well-defined representation as logical variables and expressions in the logical system in use.

Procedure calls are the other common form of atomic statements and as already discussed these can be treated much as specification statements (with empty bodies).

If the programming language has them, then other forms of atomic statements can be dealt with in an *ad hoc* way as their semantics dictates. For example, many programming languages have a null statement form, which corresponds to the identity operation on the set of predicates. An atomic statement that aborted execution could be dealt with by delivering an empty set of predicates, or equivalently, the single predicate *true*, (see example pre-condition calculations at the end of this section).

Sequential composition can be handled exactly as in the formal definition: the set of predicates calculated for the second statement is just passed in as the target post-condition for the first statement.

If-then-else statements cause sets of predicates to be combined. Each $\mathcal{P}_i(\vec{x_0}, \vec{x})$ resulting from analysing the then-part of the conditional with condition c would contribute $c \Rightarrow \mathcal{P}_i(\vec{x_0}, \vec{x})$ to the result. Similarly, each $\mathcal{P}_j(\vec{x_0}, \vec{x})$ resulting from the analysis of the else-part would contribute $\neg c \Rightarrow \mathcal{P}_j(\vec{x_0}, \vec{x})$.

While-loops are handled by logical transformations that mimic the various parts of the set comprehension in the formal definition above. There are various possible approaches, some of which necessitate a more complex representation of the post-condition involving quantifiers, rather than a flat conjunction of quantifier-free formulae. The Compliance Notation avoids this complexity by generating what are called side conditions, universally closed conjectures that have to be proved to justify the correctness of the main calculation. From a user's perspective the end result of the whole process is just a set of verification conditions (VCs) that have to be proved and these side conditions just get added to the final set of VCs. For example, in a loop with condition c, if the post-condition *postc* in the formal definition above is represented by the set of syntactic predicates $\mathcal{A}_j$, a side condition of the form $\mathcal{P}_i(\vec{x_0}, \vec{x}) \wedge c \Rightarrow \mathcal{A}_j$ is generated for each $\mathcal{P}_i(\vec{x_0}, \vec{x})$ in the representation of *body_postc*. This corresponds to the requirements of the last conjunct in the set comprehension above.

Like while-loops, specification statements require a more complex representation using quantifiers if full generality is to be achieved. Again, the Compliance Notation adopts the simpler approach of generating side conditions, if necessary. For example, side conditions of the form $\mathcal{P}_i(\vec{x_0}, \vec{x}) \wedge c \Rightarrow \mathcal{A}_j$ will be generated for each $\mathcal{P}_i(\vec{x_0}, \vec{x})$ in the representation of what is called $postc_1$ above and for each $\mathcal{A}_j$ in the representation of *postc*. This corresponds to the predicate of the set comprehension above.

We now return to the formal work. We conjecture that the useful pre-condition calculator is sound:

$$prec\_calc\_sound\_cnj \ ?\vdash \ prec\_calc \in sound\_prec\_calc$$

The following conjecture gives a useful property of our useful pre-condition calculator, which turns out to be a simple consequence of its soundness.

$$\begin{array}{l} prec\_calc\_dom\_cnj \ ?\vdash \\ \quad \forall\ prog{:}\ PROG;\ postc\ :\ P\_POST\_COND \\ \quad \bullet \quad dom\ (prec\_calc(prog,\ postc) \cap (id\ STATE)) \cap dom\ (semantics\ prog) \\ \qquad\qquad \subseteq dom\ postc \end{array}$$

We also conjecture that a program is upright in every state in the pre-condition produced by the above pre-condition calculator, i.e., no execution of the program can cause the pre-condition of any specification in the program to be violated in those states. Taken together

with the soundness conjecture, this shows that a VC generator based on the pre-condition calculator does indeed guarantee program correctness as discussed in section 5 above.

$$
\begin{array}{l}
prec\_calc\_upright\_cnj\ ?\vdash \\
\quad \forall\ prog{:}\ PROG;\ postc,\ postc' : P\_POST\_COND \\
\quad |\quad postc = prec\_calc(prog,\ postc') \\
\quad \bullet\quad prog \perp ran\ postc
\end{array}
$$

Finally, we give some evidence that the useful pre-condition calculator really is useful by exhibiting some simple programs for which it returns something more interesting than an empty relation. The first block of examples covers various forms of atom.

$$
\begin{array}{l}
prec\_calc\_atom\_egs\_cnj\ ?\vdash \\
\quad \forall\ null,\ chaos,\ stop : PROG;\ postc : P\_POST\_COND \\
\quad |\quad null = Atom\ (id\ STATE) \\
\quad \wedge\quad chaos = Atom\ (STATE \times STATE) \\
\quad \wedge\quad stop = Atom\ \varnothing \\
\quad \bullet\quad prec\_calc(null,\ postc) = postc \\
\quad \wedge\quad prec\_calc(chaos,\ postc) = \{s,\ s'{:}\ STATE \mid postc(\!|\{s\}|\!) = STATE\} \\
\quad \wedge\quad prec\_calc(stop,\ postc) = STATE \times STATE
\end{array}
$$

The second block gives at least one example of each of the compound syntactic categories.

$$
\begin{array}{l}
prec\_calc\_compound\_egs\_cnj\ ?\vdash \\
\quad \forall\ null,\ chaos,\ stop,\ p,\ spec\_null : PROG;\ postc : P\_POST\_COND;\ c : P\_PRED \\
\quad |\quad null = Atom\ (id\ STATE) \\
\quad \wedge\quad chaos = Atom\ (STATE \times STATE) \\
\quad \wedge\quad stop = Atom\ \varnothing \\
\quad \wedge\quad spec\_null = Spec((STATE,\ id\ STATE),\ null) \\
\quad \bullet\quad prec\_calc(If(c,\ null,\ stop),\ postc) = postc \rhd_* c \cup (STATE \times STATE) \rhd\!\!\!=_* c \\
\quad \wedge\quad prec\_calc(Seq(p,\ null),\ postc) = prec\_calc(p,\ postc) \\
\quad \wedge\quad prec\_calc(Seq(null,\ p),\ postc) = prec\_calc(p,\ postc) \\
\quad \wedge\quad prec\_calc(While(STATE,\ spec\_null),\ postc) = dom\ postc \times STATE \\
\quad \wedge\quad prec\_calc\ (spec\_null,\ postc) = postc
\end{array}
$$

There is some value in the above conjectures: it was only when I tried to prove them that I realised that I had mistakenly written:

$$\{s'\} \lhd t \subseteq \{s\} \lhd postc$$

instead of

$$t(\!|\{s'\}|\!) \subseteq postc(\!|\{s\}|\!)$$

in the equation for the semantic category Atom. The pre-condition calculator is sound and guarantees uprightness with this mistake, but very far from useful (since the mistaken predicate will require $s' = s$ is $\{s'\} \lhd t$ is not empty).

References

[1] LEMMA1/ZED/WRK069. *On Refinement Calculus and Partial Correctness.* R.D. Arthan, Lemma 1 Ltd., `rda@lemma-one.com`.

A THE Z THEORY preccalc

A.1 Parents

refcalc

A.2 Global Variables

$\boldsymbol{STATE}$	$\mathbb{P}\ STATE$
$\boldsymbol{STATE\_TRANSFORMER}$	$\mathbb{P}\ (STATE \leftrightarrow STATE)$
$\boldsymbol{P\_PRED}$	$\mathbb{P}\ (\mathbb{P}\ STATE)$
$\boldsymbol{P\_PRE\_COND}$	$\mathbb{P}\ (\mathbb{P}\ STATE)$
$\boldsymbol{P\_POST\_COND}$	$\mathbb{P}\ (STATE \leftrightarrow STATE)$
$\boldsymbol{P\_SPEC}$	$\mathbb{P}\ STATE \leftrightarrow STATE \leftrightarrow STATE$
$\boldsymbol{PROG}$	$\mathbb{P}\ PROG$
$\boldsymbol{Atom}$	$(STATE \leftrightarrow STATE) \leftrightarrow PROG$
$\boldsymbol{Seq}$	$PROG \times PROG \leftrightarrow PROG$
$\boldsymbol{If}$	$\mathbb{P}\ STATE \times PROG \times PROG \leftrightarrow PROG$
$\boldsymbol{While}$	$\mathbb{P}\ STATE \times PROG \leftrightarrow PROG$
$\boldsymbol{Spec}$	$(\mathbb{P}\ STATE \times (STATE \leftrightarrow STATE)) \times PROG \leftrightarrow PROG$
$\boldsymbol{semantics}$	$PROG \leftrightarrow STATE \leftrightarrow STATE$
$(\_ \models \_)$	$PROG \leftrightarrow \mathbb{P}\ STATE \times (STATE \leftrightarrow STATE)$
$(\_ \perp \_)$	$PROG \leftrightarrow \mathbb{P}\ STATE$
$\boldsymbol{PREC\_CALC}$	$\mathbb{P}\ (PROG \times (STATE \leftrightarrow STATE) \leftrightarrow STATE \leftrightarrow STATE)$
$\boldsymbol{sound\_prec\_calc}$	$\mathbb{P}\ (PROG \times (STATE \leftrightarrow STATE) \leftrightarrow STATE \leftrightarrow STATE)$
$(\_ \trianglerighteq_* \_)[\boldsymbol{X}, \boldsymbol{Y}]$	$(X \leftrightarrow Y) \times \mathbb{P}\ Y \leftrightarrow X \leftrightarrow Y$
$(\_ \triangleright_* \_)[\boldsymbol{X}, \boldsymbol{Y}]$	$(X \leftrightarrow Y) \times \mathbb{P}\ Y \leftrightarrow X \leftrightarrow Y$
$\boldsymbol{guess\_spec}$	$PROG \leftrightarrow \mathbb{P}\ STATE \times (STATE \leftrightarrow STATE)$
$\boldsymbol{prec\_calc}$	$PROG \times (STATE \leftrightarrow STATE) \leftrightarrow STATE \leftrightarrow STATE$

A.3 Fixity

$\boldsymbol{fun}$ **60** $\boldsymbol{rightassoc}$

$(\_ \trianglerighteq_* \_)$ $(\_ \triangleright_* \_)$

$\boldsymbol{rel}$ $(\_ \perp \_)$

A.4 Axioms

Atom
Seq
If
While
Spec $\vdash$ $(Atom \in STATE\_TRANSFORMER \rightarrowtail PROG$
$\wedge\ Seq \in PROG \times PROG \rightarrowtail PROG$
$\wedge\ If \in P\_PRED \times PROG \times PROG \rightarrowtail PROG$
$\wedge\ While \in P\_PRED \times PROG \rightarrowtail PROG$
$\wedge\ Spec \in P\_SPEC \times PROG \rightarrowtail PROG)$
$\wedge\ disjoint\ \langle ran\ Atom,$
$ran\ Seq,$
$ran\ If,$
$ran\ While,$
$ran\ Spec\rangle$
$\wedge\ (\forall\ W : \mathbb{P}\ PROG$
$\mid Atom\ (\!|\ STATE\_TRANSFORMER\ |\!)$
$\cup\ (Seq\ (\!|\ W \times W\ |\!)$
$\cup\ (If\ (\!|\ P\_PRED \times W \times W\ |\!)$
$\cup\ (While\ (\!|\ P\_PRED \times W\ |\!)$
$\cup\ Spec\ (\!|\ P\_SPEC \times W\ |\!))))$
$\subseteq W$
$\bullet\ PROG \subseteq W)$

semantics $\vdash semantics \in PROG \rightarrow STATE\_TRANSFORMER$
$\wedge\ (\forall\ t : STATE\_TRANSFORMER;$
$p_1, p_2 : PROG;$
$c : P\_PRED;$
$s : P\_SPEC$
$\bullet\ semantics\ (Atom\ t) = t$
$\wedge\ semantics\ (Seq\ (p_1, p_2))$
$= semantics\ p_1 \mathbin{;} semantics\ p_2$
$\wedge\ semantics\ (If\ (c, p_1, p_2))$
$= c \lhd semantics\ p_1 \cup c \ntriangleleft semantics\ p_2$
$\wedge\ semantics\ (While\ (c, p_1))$
$= (c \lhd semantics\ p_1)^* \ntriangleright c$
$\wedge\ semantics\ (Spec\ (s, p_1)) = semantics\ p_1)$

$\_ \models \_$ $\vdash (\_ \models \_) \in PROG \leftrightarrow SPEC$
$\wedge\ (\forall\ prog : PROG;$
$prec : P\_PRE\_COND;$
$postc : P\_POST\_COND$
$\bullet\ prog \models (prec, postc)$
$\Leftrightarrow (prec, postc) \sqsubseteq (prec, semantics\ prog))$

$\_ \perp \_$ $\vdash (\_ \perp \_) \in PROG \leftrightarrow PRED$
$\wedge\ (\forall\ t : STATE\_TRANSFORMER;$
$postc : P\_POST\_COND;$
$p_1, p_2 : PROG;$
$c_1, c_2, prec : P\_PRE\_COND$
$\bullet\ Atom\ t \perp c_1$
$\wedge\ (Seq\ (p_1, p_2) \perp c_1$
$\Leftrightarrow p_1 \perp c_1 \wedge p_2 \perp semantics\ p_1\ (\!|\ c_1\ |\!))$
$\wedge\ (If\ (c_2, p_1, p_2) \perp c_1$

$\Leftrightarrow p_1 \perp c_1 \cap c_2 \wedge p_2 \perp c_1 \setminus c_2)$
$\wedge$ (*While* $(c_2, p_1) \perp c_1$
$\Leftrightarrow p_1 \perp c_1 \cap c_2$
$\wedge\ p_1 \perp$ *semantics* $p_1 \llparenthesis c_1 \cap c_2 \rrparenthesis \cap c_2$)
$\wedge$ (*Spec* $((prec, postc), p_1) \perp c_1$
$\Leftrightarrow c_1 \subseteq prec \wedge p_1 \perp c_1$))

sound_prec_calc

$\vdash$ *sound_prec_calc* $\in \mathbb{P}$ *PREC_CALC*
$\wedge$ ($\forall$ *pc* : *PREC_CALC*
• *pc* $\in$ *sound_prec_calc*
$\Leftrightarrow$ ($\forall$ *prog* : *PROG*; *postc*, *postc'* : *P_POST_COND*
| *postc* = *pc* (*prog*, *postc'*)
• (*dom postc*, *postc'*)
$\sqsubseteq$ (*dom postc*, *postc* $\fatsemi$ *semantics prog*)))

\- $\rhd\!\!\!*$ -

\- $\rhd_*$ -

$\vdash [X,$
$Y](((\_ \rhd_* \_)[X, Y] \in (X \leftrightarrow Y) \times \mathbb{P}\ Y \rightarrow X \leftrightarrow Y$
$\wedge\ (\_ \rhd\!\!\!*\ \_)[X, Y] \in (X \leftrightarrow Y) \times \mathbb{P}\ Y \rightarrow X \leftrightarrow Y)$
$\wedge$ ($\forall$ $R : X \leftrightarrow Y$; $T : \mathbb{P}\ Y$
• $(\_ \rhd_* \_)[X, Y]\ (R, T) = (R \sim) \llparenthesis Y \setminus T \rrparenthesis \lhd R$
$\wedge\ (\_ \rhd\!\!\!*\ \_)[X, Y]\ (R, T) = (R \sim) \llparenthesis T \rrparenthesis \lhd R$))

guess_spec

$\vdash$ *guess_spec* $\in$ *PROG* $\rightarrow$ *P_SPEC*
$\wedge$ ($\forall$ s : *P_SPEC*; p : *PROG*
• *guess_spec* (*Spec* $(s, p)) = s$)

prec_calc

$\vdash$ *prec_calc* $\in$ *PREC_CALC*
$\wedge$ ($\forall$ t : *STATE_TRANSFORMER*;
postc, $postc_1$: *P_POST_COND*;
p_1, p_2 : *PROG*;
c : *P_PRED*;
$prec_1$: *P_PRE_COND*;
body_prec : *P_PRED*;
body_postc : *P_POST_COND*
• *prec_calc* (*Atom* t, *postc*)
$= \{s, s' : STATE$
$\mid t \llparenthesis \{s'\} \rrparenthesis \subseteq postc \llparenthesis \{s\} \rrparenthesis\}$
$\wedge$ *prec_calc* (*Seq* (p_1, p_2), *postc*)
$=$ *prec_calc* $(p_1$, *prec_calc* $(p_2$, *postc*))
$\wedge$ *prec_calc* (*If* (c, p_1, p_2), *postc*)
$=$ *prec_calc* $(p_1$, *postc*) $\rhd_*$ c
$\cup$ *prec_calc* $(p_2$, *postc*) $\rhd\!\!\!*$ c
$\wedge$ ((*body_prec*, *body_postc*) $=$ *guess_spec* p_1
$\Rightarrow$ *prec_calc* (*While* (c, p_1), *postc*)
$=$ *postc* $\rhd\!\!\!*$ c
$\cup \{ss' : dom\ postc \times c$
$\mid c \subseteq body\_prec$
$\wedge\ body\_prec$
$\subseteq dom$
(*prec_calc* $(p_1$, *body_postc*)
$\cap\ id\ STATE$)
$\wedge\ dom\ postc$
$\times\ body\_postc \llparenthesis body\_prec \rrparenthesis \setminus c$

$$\subseteq\ postc\})$$
$$\wedge\ prec\_calc$$
$$(Spec\ ((prec_1,\ postc_1),\ p_1),\ postc)$$
$$= \{s : STATE;\ s' : STATE$$
$$\mid postc_1\ (\!|\ \{s'\}\ |\!) \subseteq postc\ (\!|\ \{s\}\ |\!)\}$$
$$\rhd\ (prec_1$$
$$\cap\ dom$$
$$(prec\_calc\ (p_1,\ postc_1)$$
$$\cap\ id\ STATE)))$$

A.5 Definitions

STATE $\vdash STATE = \mathbb{U}$

STATE_TRANSFORMER
$\vdash STATE\_TRANSFORMER = STATE \leftrightarrow STATE$

P_PRED $\vdash P\_PRED = PRED[STATE]$

P_PRE_COND $\vdash P\_PRE\_COND = PRE\_COND[STATE]$

P_POST_COND $\vdash P\_POST\_COND = POST\_COND[STATE,\ STATE]$

P_SPEC $\vdash P\_SPEC = SPEC[STATE,\ STATE]$

PROG $\vdash PROG = \mathbb{U}$

PREC_CALC $\vdash PREC\_CALC = PROG \times P\_POST\_COND \rightarrow P\_POST\_COND$

A.6 Conjectures

strip_specs_cnj
$$\forall\ strip\_specs : PROG \rightarrow PROG$$
$$\mid \forall\ t : STATE\_TRANSFORMER;$$
$$p_1,\ p_2 : PROG;$$
$$c : P\_PRED;$$
$$s : SPEC$$
$$\bullet\ strip\_specs\ (Atom\ t) = Atom\ t$$
$$\wedge\ strip\_specs\ (Seq\ (p_1,\ p_2))$$
$$= Seq\ (strip\_specs\ p_1,\ strip\_specs\ p_2)$$
$$\wedge\ strip\_specs\ (If\ (c,\ p_1,\ p_2))$$
$$= If\ (c,\ strip\_specs\ p_1,\ strip\_specs\ p_2)$$
$$\wedge\ strip\_specs\ (While\ (c,\ p_1))$$
$$= While\ (c,\ strip\_specs\ p_1)$$
$$\wedge\ strip\_specs\ (Spec\ (s,\ p_1)) = strip\_specs\ p_1$$
$$\bullet \forall\ p : PROG$$
$$\bullet\ semantics\ (strip\_specs\ p) = semantics\ p$$

upright_mono_cnj
$$\forall\ prog : PROG;\ c_1,\ c_2 : P\_PRED$$
$$\bullet\ prog \perp c_1 \wedge c_2 \subseteq c_1 \Rightarrow prog \perp c_2$$

upright_cup_cnj
$$\forall\ prog : PROG;\ c_1,\ c_2 : P\_PRED$$
$$\bullet\ prog \perp c_1 \wedge prog \perp c_2 \Rightarrow prog \perp c_1 \cup c_2$$

prec_calc_sat_cnj
$$\forall\ pc : PREC\_CALC;$$
$$c\_prec,\ s\_prec : P\_PRE\_COND;$$
$$s\_postc : P\_POST\_COND;$$

$p : PROG$
$\mid pc \in sound\_prec\_calc$
$\wedge\ c\_prec$
$= dom$
$(pc\ (Spec\ ((s\_prec,\ s\_postc),\ p),\ s\_postc)$
$\cap\ id\ STATE)$
$\wedge\ s\_prec \subseteq c\_prec$
$\bullet\ p \models (s\_prec,\ s\_postc)$

trivial_prec_calc_sound_cnj

$\forall\ pc : PREC\_CALC$
$\mid \forall\ prog : PROG;\ postc : P\_POST\_COND$
$\bullet\ pc\ (prog,\ postc) = \varnothing$
$\bullet\ pc \in sound\_prec\_calc$

prec_calc_sound_cnj

$prec\_calc \in sound\_prec\_calc$

prec_calc_dom_cnj

$\forall\ prog : PROG;\ postc : P\_POST\_COND$
$\bullet\ dom\ (prec\_calc\ (prog,\ postc) \cap id\ STATE)$
$\cap\ dom\ (semantics\ prog)$
$\subseteq dom\ postc$

prec_calc_upright_cnj

$\forall\ prog : PROG;\ postc,\ postc' : P\_POST\_COND$
$\mid postc = prec\_calc\ (prog,\ postc')$
$\bullet\ prog \perp ran\ postc$

prec_calc_atom_egs_cnj

$\forall\ null,\ chaos,\ stop : PROG;\ postc : P\_POST\_COND$
$\mid null = Atom\ (id\ STATE)$
$\wedge\ chaos = Atom\ (STATE \times STATE)$
$\wedge\ stop = Atom\ \varnothing$
$\bullet\ prec\_calc\ (null,\ postc) = postc$
$\wedge\ prec\_calc\ (chaos,\ postc)$
$= \{s,\ s' : STATE$
$\mid postc\ (\!|\ \{s\}\ |\!) = STATE\}$
$\wedge\ prec\_calc\ (stop,\ postc) = STATE \times STATE$

prec_calc_compound_egs_cnj

$\forall\ null,\ chaos,\ stop,\ p,\ spec\_null : PROG;$
$postc : P\_POST\_COND;$
$c : P\_PRED$
$\mid null = Atom\ (id\ STATE)$
$\wedge\ chaos = Atom\ (STATE \times STATE)$
$\wedge\ stop = Atom\ \varnothing$
$\wedge\ spec\_null = Spec\ ((STATE,\ id\ STATE),\ null)$
$\bullet\ prec\_calc\ (If\ (c,\ null,\ stop),\ postc)$
$= postc \rhd_* c \cup (STATE \times STATE) \lhd_* c$
$\wedge\ prec\_calc\ (Seq\ (p,\ null),\ postc)$
$= prec\_calc\ (p,\ postc)$
$\wedge\ prec\_calc\ (Seq\ (null,\ p),\ postc)$
$= prec\_calc\ (p,\ postc)$
$\wedge\ prec\_calc\ (While\ (STATE,\ spec\_null),\ postc)$
$= dom\ postc \times STATE$
$\wedge\ prec\_calc\ (spec\_null,\ postc) = postc$

A.7 Theorems

refinement_u_thm

$\vdash STATE\_TRANSFORMER = \mathbb{U}$
$\quad \land STATE = \mathbb{U}$
$\quad \land P\_PRED = \mathbb{U}$
$\quad \land P\_PRE\_COND = \mathbb{U}$
$\quad \land P\_POST\_COND = \mathbb{U}$
$\quad \land PROG = \mathbb{U}$
$\quad \land P\_SPEC = \mathbb{U}$
$\quad \land (\_ \leftrightarrow \_) = \mathbb{U}$

strong_rres_thm

$\vdash \forall R : \mathbb{U};\ T : \mathbb{U}$
$\quad \bullet R \rhd_* T$
$\quad\quad = \{x : \mathbb{U};\ y : \mathbb{U}$
$\quad\quad\quad \mid (x, y) \in R \land (\forall z : \mathbb{U} \mid (x, z) \in R \bullet z \in T)\}$

strong_rantires_thm

$\vdash \forall R : \mathbb{U};\ T : \mathbb{U}$
$\quad \bullet R \ntriangleright_* T$
$\quad\quad = \{x : \mathbb{U};\ y : \mathbb{U}$
$\quad\quad\quad \mid (x, y) \in R \land (\forall z : \mathbb{U} \mid (x, z) \in R \bullet z \notin T)\}$

sound_prec_calc_thm

$\vdash sound\_prec\_calc$
$\quad = \{pc : PREC\_CALC$
$\quad\quad \mid \forall prog : \mathbb{U};\ postc, postc' : \mathbb{U}$
$\quad\quad\quad \mid postc = pc\ (prog, postc')$
$\quad\quad\quad \bullet (dom\ postc, postc')$
$\quad\quad\quad\quad \sqsubseteq (dom\ postc, postc \mathbin{⨾} semantics\ prog)\}$

semantics_def

$\vdash semantics \in (\_ \rightarrow \_)$
$\quad \land (\forall t : \mathbb{U};\ p_1, p_2 : \mathbb{U};\ c : \mathbb{U};\ s : \mathbb{U}$
$\quad\quad \bullet semantics\ (Atom\ t) = t$
$\quad\quad\quad \land semantics\ (Seq\ (p_1, p_2))$
$\quad\quad\quad\quad = semantics\ p_1 \mathbin{⨾} semantics\ p_2$
$\quad\quad\quad \land semantics\ (If\ (c, p_1, p_2))$
$\quad\quad\quad\quad = c \lhd semantics\ p_1 \cup c \ntriangleleft semantics\ p_2$
$\quad\quad\quad \land semantics\ (While\ (c, p_1))$
$\quad\quad\quad\quad = (c \lhd semantics\ p_1)^* \ntriangleright c$
$\quad\quad\quad \land semantics\ (Spec\ (s, p_1)) = semantics\ p_1)$

sat_def

$\vdash \forall prog : \mathbb{U};\ prec : \mathbb{U};\ postc : \mathbb{U}$
$\quad \bullet prog \models (prec, postc)$
$\quad\quad \Leftrightarrow (prec, postc) \sqsubseteq (prec, semantics\ prog)$

upright_def

$\vdash \forall t : \mathbb{U};\ postc : \mathbb{U};\ p_1, p_2 : \mathbb{U};\ c_1, c_2, prec : \mathbb{U}$
$\quad \bullet Atom\ t \perp c_1$
$\quad\quad \land (Seq\ (p_1, p_2) \perp c_1$
$\quad\quad\quad \Leftrightarrow p_1 \perp c_1 \land p_2 \perp semantics\ p_1 (\!|\ c_1\ |\!))$
$\quad\quad \land (If\ (c_2, p_1, p_2) \perp c_1$
$\quad\quad\quad \Leftrightarrow p_1 \perp c_1 \cap c_2 \land p_2 \perp c_1 \setminus c_2)$
$\quad\quad \land (While\ (c_2, p_1) \perp c_1$
$\quad\quad\quad \Leftrightarrow p_1 \perp c_1 \cap c_2$
$\quad\quad\quad\quad \land p_1 \perp semantics\ p_1 (\!|\ c_1 \cap c_2\ |\!) \cap c_2)$
$\quad\quad \land (Spec\ ((prec, postc), p_1) \perp c_1$

$$\Leftrightarrow c_1 \subseteq prec \land p_1 \perp c_1)$$

prec_calc_def

$$\vdash prec\_calc \in PREC\_CALC$$
$$\land (\forall\ t : \mathbb{U};$$
$$postc,\ postc_1 : \mathbb{U};$$
$$p_1,\ p_2 : \mathbb{U};$$
$$c : \mathbb{U};$$
$$prec_1 : \mathbb{U};$$
$$body\_prec : \mathbb{U};$$
$$body\_postc : \mathbb{U}$$
$$\bullet\ prec\_calc\ (Atom\ t,\ postc)$$
$$= \{s,\ s' : \mathbb{U}$$
$$\mid t\ (\!|\ \{s'\}\ |\!) \subseteq postc\ (\!|\ \{s\}\ |\!)\}$$
$$\land\ prec\_calc\ (Seq\ (p_1,\ p_2),\ postc)$$
$$= prec\_calc\ (p_1,\ prec\_calc\ (p_2,\ postc))$$
$$\land\ prec\_calc\ (If\ (c,\ p_1,\ p_2),\ postc)$$
$$= prec\_calc\ (p_1,\ postc) \rhd_* c$$
$$\cup\ prec\_calc\ (p_2,\ postc) \trianglerighteq_* c$$
$$\land\ ((body\_prec,\ body\_postc) = guess\_spec\ p_1$$
$$\Rightarrow prec\_calc\ (While\ (c,\ p_1),\ postc)$$
$$= postc \trianglerighteq c$$
$$\cup\ \{ss' : dom\ postc \times c$$
$$\mid c \subseteq body\_prec$$
$$\land\ body\_prec$$
$$\subseteq dom$$
$$(prec\_calc\ (p_1,\ body\_postc)$$
$$\cap\ (id\ \_))$$
$$\land\ dom\ postc$$
$$\times\ body\_postc\ (\!|\ body\_prec\ |\!) \setminus c$$
$$\subseteq postc\})$$
$$\land\ prec\_calc$$
$$(Spec\ ((prec_1,\ postc_1),\ p_1),\ postc)$$
$$= \{s : \mathbb{U};\ s' : \mathbb{U}$$
$$\mid postc_1\ (\!|\ \{s'\}\ |\!) \subseteq postc\ (\!|\ \{s\}\ |\!)\}$$
$$\rhd\ (prec_1$$
$$\cap\ dom$$
$$(prec\_calc\ (p_1,\ postc_1)$$
$$\cap\ (id\ \_))))$$

prog_induction_thm

$$\vdash \ulcorner \forall\ p$$
$$\bullet\ (\forall\ t\bullet\ p\ \llbracket_{\mathrm{Z}} Atom\ t \urcorner)$$
$$\land\ (\forall\ p1\ p2\bullet\ p\ p1 \land p\ p2 \Rightarrow p\ \llbracket_{\mathrm{Z}} Seq\ (p1,\ p2) \urcorner)$$
$$\land\ (\forall\ c\ p1\ p2\bullet\ p\ p1 \land p\ p2 \Rightarrow p\ \llbracket_{\mathrm{Z}} If\ (c,\ p1,\ p2) \urcorner)$$
$$\land\ (\forall\ c\ p1\bullet\ p\ p1 \Rightarrow p\ \llbracket_{\mathrm{Z}} While\ (c,\ p1) \urcorner)$$
$$\land\ (\forall\ prec\ postc\ p1$$
$$\bullet\ p\ p1 \Rightarrow p\ \llbracket_{\mathrm{Z}} Spec\ ((prec,\ postc),\ p1) \urcorner)$$
$$\Rightarrow (\forall\ prg\bullet\ p\ prg) \urcorner$$

strip_specs_thm

$$\vdash \forall\ strip\_specs : PROG \to PROG$$
$$\mid \forall\ t : STATE\_TRANSFORMER;$$
$$p_1,\ p_2 : PROG;$$

$c : P\_PRED;$
$s : SPEC$
$\bullet\ strip\_specs\ (Atom\ t) = Atom\ t$
$\wedge\ strip\_specs\ (Seq\ (p_1,\ p_2))$
$= Seq\ (strip\_specs\ p_1,\ strip\_specs\ p_2)$
$\wedge\ strip\_specs\ (If\ (c,\ p_1,\ p_2))$
$= If\ (c,\ strip\_specs\ p_1,\ strip\_specs\ p_2)$
$\wedge\ strip\_specs\ (While\ (c,\ p_1))$
$= While\ (c,\ strip\_specs\ p_1)$
$\wedge\ strip\_specs\ (Spec\ (s,\ p_1)) = strip\_specs\ p_1$
$\bullet\ \forall\ p : PROG$
$\bullet\ semantics\ (strip\_specs\ p) = semantics\ p$

chaos_thm $\vdash \forall\ postc : POST\_COND;\ s : SPEC \bullet (\varnothing,\ postc) \sqsubseteq s$

upright_mono_thm
$\vdash \forall\ prog : PROG;\ c_1,\ c_2 : P\_PRED$
$\bullet\ prog \perp c_1 \wedge c_2 \subseteq c_1 \Rightarrow prog \perp c_2$

upright_cup_thm
$\vdash \forall\ prog : PROG;\ c_1,\ c_2 : P\_PRED$
$\bullet\ prog \perp c_1 \wedge prog \perp c_2 \Rightarrow prog \perp c_1 \cup c_2$

prec_calc_sat_thm
$\vdash \forall\ pc : PREC\_CALC;$
$c\_prec,\ s\_prec : P\_PRE\_COND;$
$s\_postc : P\_POST\_COND;$
$p : PROG$
$|\ pc \in sound\_prec\_calc$
$\wedge\ c\_prec$
$= dom$
$(pc\ (Spec\ ((s\_prec,\ s\_postc),\ p),\ s\_postc)$
$\cap\ id\ STATE)$
$\wedge\ s\_prec \subseteq c\_prec$
$\bullet\ p \models (s\_prec,\ s\_postc)$

while_lemma $\vdash \forall\ R : \mathbb{U};\ c : \mathbb{U};\ x,\ y : \mathbb{U}$
$|\ (x,\ y) \in (c \lhd R)^{*} \rhd c$
$\bullet\ y \notin c$
$\wedge\ (y = x$
$\vee\ x \in c \wedge (\exists\ z : \mathbb{U} \bullet z \in c \wedge (z,\ y) \in R))$

trivial_sound_prec_calc_thm
$\vdash \forall\ pc : PREC\_CALC$
$|\ \forall\ prog : PROG;\ postc : P\_POST\_COND$
$\bullet\ pc\ (prog,\ postc) = \varnothing$
$\bullet\ pc \in sound\_prec\_calc$

prec_calc_sound_thm
$\vdash prec\_calc \in sound\_prec\_calc$

prec_calc_dom_thm
$\vdash \forall\ prog : PROG;\ postc : P\_POST\_COND$
$\bullet\ dom\ (prec\_calc\ (prog,\ postc) \cap id\ STATE)$
$\cap\ dom\ (semantics\ prog)$
$\subseteq dom\ postc$

prec_calc_upright_thm
$\vdash \forall\ prog : PROG;\ postc,\ postc' : P\_POST\_COND$
$|\ postc = prec\_calc\ (prog,\ postc')$

$\bullet\ prog \perp ran\ postc$

prec_calc_atom_egs_thm

$\vdash \forall\ null,\ chaos,\ stop : PROG;\ postc : P\_POST\_COND$
$\mid null = Atom\ (id\ STATE)$
$\wedge\ chaos = Atom\ (STATE \times STATE)$
$\wedge\ stop = Atom\ \varnothing$
$\bullet\ prec\_calc\ (null,\ postc) = postc$
$\wedge\ prec\_calc\ (chaos,\ postc)$
$= \{s,\ s' : STATE$
$\mid postc\ (\!|\ \{s\}\ |\!) = STATE\}$
$\wedge\ prec\_calc\ (stop,\ postc) = STATE \times STATE$

prec_calc_compound_egs_thm

$\vdash \forall\ null,\ chaos,\ stop,\ p,\ spec\_null : PROG;$
$postc : P\_POST\_COND;$
$c : P\_PRED$
$\mid null = Atom\ (id\ STATE)$
$\wedge\ chaos = Atom\ (STATE \times STATE)$
$\wedge\ stop = Atom\ \varnothing$
$\wedge\ spec\_null = Spec\ ((STATE,\ id\ STATE),\ null)$
$\bullet\ prec\_calc\ (If\ (c,\ null,\ stop),\ postc)$
$= postc \rhd_* c \cup (STATE \times STATE) \rhd_* c$
$\wedge\ prec\_calc\ (Seq\ (p,\ null),\ postc)$
$= prec\_calc\ (p,\ postc)$
$\wedge\ prec\_calc\ (Seq\ (null,\ p),\ postc)$
$= prec\_calc\ (p,\ postc)$
$\wedge\ prec\_calc\ (While\ (STATE,\ spec\_null),\ postc)$
$= dom\ postc \times STATE$
$\wedge\ prec\_calc\ (spec\_null,\ postc) = postc$

B INDEX