Journal of Software Engineering and Applications, 2014, 7, 555-561
Published Online June 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.77051

Scientific Research

A Logical Treatment of Non-Termination and Program Behaviour

Martin Ward[1*], Hussein Zedan[2]

[1]Software Technology Research Lab, De Montfort University, Leicester, UK
[2]Applied Science University, Al Eker, Bahrain
Email: [*]martin@gkc.org.uk, hussein.zedan@googlemail.com

Received 13 February 2014; revised 10 March 2014; accepted 18 March 2014

Abstract

Can the semantics of a program be represented as a single formula? We show that one formula is insufficient to handle assertions, refinement or slicing, while two formulae are sufficient: $\mathcal{A}(S)$, defining non-termination, and $\mathcal{B}(S)$, defining behaviour. Any two formulae A and B will define a corresponding program. Refinement is defined as implication between these formulae.

Keywords

Formal Methods, Refinement, Non-Termination, Non-Determinism, Weakest Precondition, Temporal Logic, Wide-Spectrum Language

1. Introduction

The idea of using a single formula to represent the behaviour of a program is a very attractive one: proving that two programs are equivalent then reduces to the task of proving that two formula are equivalent. For the latter task, mathematicians have developed many powerful techniques over the last few thousand years of the history of mathematics.

In this paper, we show that a single formula is insufficient to represent the semantics of a program in the desired way, but there are two formulae which are sufficient.

2. The WSL Language

The WSL transformation theory is based in infinitary logic: an extension of first order logic which allows infi-

[*]Corresponding author.

How to cite this paper: Ward, M. and Zedan, H. (2014) A Logical Treatment of Non-Termination and Program Behaviour. *Journal of Software Engineering and Applications*, **7**, 555-561. http://dx.doi.org/10.4236/jsea.2014.77051

nitely long formulae. The statements in the WSL kernel language are as follows: $\{\mathbf{P}\}$ is an assertion which terminates immediately if $\mathbf{P}$ is true and aborts if $\mathbf{P}$ is false, $[\mathbf{P}]$ is a guard which ensures that $\mathbf{P}$ is true without changing the value of any variable, $\mathbf{add}(x)$ adds the variables in x to the state space, if they are not already present, and assigns arbitrary values to the variables. $\mathbf{remove}(x)$ removes the variables in x from the state space, $\mathbf{S}_1;\mathbf{S}_2$ is sequential composition, $(\mathbf{S}_1 \sqcap \mathbf{S}_2)$ is nondeterministic choice, and $(\mu X_y^x.\mathbf{S})$ is a recursive subroutine which also adds the variables in x to the state space and removes the variables in y from the state space.

The semantics of a WSL program is defined as a function which maps each initial state to the set of possible final states. A state is either the special state $\perp$, which represents non-termination, or a proper state which is a function from a set of variables (the state space) to the set of values. The semantics of a program is always defined in the context of a particular initial state space and final state space.

For any list of variables x, we define $\tilde{x}$ to be the set of variables in x, and x' and x'' to be the corresponding sequences of primed and doubly primed variables. The formula $x \neq x''$ is true precisely when the value of any variable in x differs from the value of the corresponding variable in x''.

The interpretation function $\mathrm{Int}_M^{\mathrm{den}}(\mathbf{S},V)$ maps each statement $\mathbf{S}$ and initial state space V to the corresponding state transformation (under a model M for the logic in question). For a state space V, and set of values $\mathcal{H}$, let $D_{\mathcal{H}}(V)$ be the set of states on V and $\mathcal{H}$ (including $\perp$), and for any formula $\mathbf{P}$ let $\mathrm{int}_M(\mathbf{P},V)$ be the set of states which satisfies the formula. See [1] for details.

For initial state $\perp$ we define $\mathrm{Int}_M^{\mathrm{den}}(\mathbf{S},V)(\perp) = D_{\mathcal{H}}(W)$ for every statement $\mathbf{S}:V \to W$, *i.e.* if the program is started in the non-terminating state then it cannot be guaranteed to terminate. (Starting the program in a non-terminating state simply means that some previous program in the sequence has failed to terminate, so this program can never actually start. This restriction simply means that a later statement in a sequence cannot somehow "recover" from non-termination of an earlier statement in the sequence the program).

The semantics for the recursive program is simply the intersection of the semantics for each finite truncation. The result is the least defined statement which is a refinement of all the truncations.

If the initial state space is empty, then there are two possible initial states: $\perp$ and the single proper state $\varnothing$. For $\perp$ the set of final states must be $D_{\mathcal{H}}(W)$, by definition. So the state transformation is entirely determined by its value on the initial state $\varnothing$. If the final state space is also empty, then there are exactly three distinct state transformations, corresponding to the three possible sets of states.[1] The three state transformations are f_1, f_2 and f_3 where:

$$f_1(\varnothing) = \{\ \} \qquad f_2(\varnothing) = \{\varnothing\} \qquad f_3(\varnothing) = \{\perp, \varnothing\}$$

These correspond to the three fundamental statements $\mathbf{null} =_{\mathrm{DF}} [\mathbf{false}]$, $\mathbf{skip} =_{\mathrm{DF}} \{\mathbf{true}\}$ and $\mathbf{abort} =_{\mathrm{DF}} \{\mathbf{false}\}$:

$$\mathrm{Int}_M^{\mathrm{den}}([\mathbf{false}],\varnothing) = f_1 \qquad \mathrm{Int}_M^{\mathrm{den}}(\{\mathbf{true}\},\varnothing) = f_2 \qquad \mathrm{Int}_M^{\mathrm{den}}(\{\mathbf{false}\},\varnothing) = f_3$$

Note that there are three different semantic function which use no variables, but only *two* semantically-different formulae with no free variables (namely $\mathbf{true}$ and $\mathbf{false}$). Under any interpretation of the logic, any formula with no free variables must be interpreted as either universally true or universally false. There is no way to map three different semantics onto two formulae: so this proves that a single formula is insufficient to represent the semantics of a program.

3. The Abort and Behaviour Predicates

Since it is not possible to represent the semantics of a program using one formula, we will now consider how we can represent the denotational semantics of a program using *two* formulae from infinitary first order logic. The formulae are defining in terms of the weakest precondition.

For any program $\mathbf{S}$ and postcondition (condition on the final state space) $\mathbf{R}$, the weakest precondition $\mathrm{WP}(\mathbf{S},\mathbf{R})$ is the weakest condition on the initial state space such that if the initial state satisfies $\mathrm{WP}(\mathbf{S},\mathbf{R})$ then $\mathbf{S}$ is guaranteed to terminate in a state which satisfies $\mathbf{R}$.

In [1], we show that $\mathrm{WP}(\mathbf{S},\mathbf{R})$ can be defined as a formula in infinitary logic and that refinement can be

[1]Recall that if $\perp$ is in the set of final states, then every other state has to be included: so $\{\perp\}$ is not a valid final set of states.

characterised by the weakest precondition as follows: for any to programs $\mathbf{S}_1$ and $\mathbf{S}_2$ with the same initial and final state spaces, $\mathbf{S}_2$ is a refinement of $\mathbf{S}_1$, written $\mathbf{S}_1 \le \mathbf{S}_2$, if and only if for all postconditions $\mathbf{R}$:

$$\mathrm{WP}(\mathbf{S}_1, \mathbf{R}) \Rightarrow \mathrm{WP}(\mathbf{S}_2, \mathbf{R})$$

In the same paper, we also prove that it is not necessary to determine the weakest preconditions for all post-conditions: two very simple postconditions are sufficient. These are the conditions **true** and $\boldsymbol{x} \neq \boldsymbol{x}''$ where $\boldsymbol{x}$ is a list of all the variables in W and $\mathbf{x}''$ is a list of new variables, not appearing in either program, which are the doubly-primed versions of the variables in W. Then $\mathbf{S}_1 \le \mathbf{S}_2$ if and only if:

$$\mathrm{WP}(\mathbf{S}_1, \mathbf{true}) \Rightarrow \mathrm{WP}(\mathbf{S}_2, \mathbf{true}) \quad \text{and} \quad \mathrm{WP}(\mathbf{S}_1, \boldsymbol{x} \neq \boldsymbol{x}'') \Rightarrow \mathrm{WP}(\mathbf{S}_2, \boldsymbol{x} \neq \boldsymbol{x}'')$$

For any statement $\mathbf{S}$, the formula $\mathrm{WP}(\mathbf{S}, \mathbf{true})$ describes precisely those initial states on which $\mathbf{S}$ is guaranteed to terminate. For each of these states, the formula $\neg\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'')$ describes the behaviour of $\mathbf{S}$ in the sense that, if s is an initial state for which $\mathbf{S}$ terminates, and s'' is an extension of s which adds the variables $\boldsymbol{x}''$ to the state with a given set of values, then s'' satisfies $\neg\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'')$ precisely when the values assigned to $\boldsymbol{x}''$ form a possible final state for $\mathbf{S}$ when they are assigned to the corresponding un-primed variables.

To be more precise, we will prove the following theorem:

Theorem 3.1. If f is the interpretation of $\mathbf{S}$, then for every initial state s and final state $t \in f(s)$, the corresponding extended initial state s_t is in $\neg\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'')$, and conversely, every state in $\neg\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'')$ is of the form s_t for some initial state s and $t \in f(s)$.

Proof: Suppose f is the interpretation of $\mathbf{S}$ as a state transformation from V to W, and let s be any initial state. Let t be any proper state in $f(s)$ (*i.e.* any element of $f(s)$ apart from $\perp$). Let f'' be the extension of f which adds W'' to the initial and final state spaces and preserves the values of these variables. Then f'' is the interpretation of $\mathbf{S}$ as a state transformation from $V \cup W''$ to $W \cup W''$ and for every variable $x'' \in W''$, the initial and final values of x'' on f'' are identical. Let s_t be the state s extended to state space $V \cup W''$ which gives the variables in W'' the same values that the corresponding unprimed variables have in t. So, for every $x'' \in W''$, $s_t(x'') = t(x)$, and $s_t(x) = s(x)$ for $x \notin W''$. Let t_t be the corresponding extension to t. Then, by the definition of f'', $t_t \in f''(s_t)$.

Claim: s_t is in the interpretation of $\neg\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'')$. To prove the claim, assume for contradiction that s_t is in $\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'')$. Then $\mathbf{S}$ is guaranteed to terminate in a state which satisfies $\boldsymbol{x} \neq \boldsymbol{x}''$, in other words, every state in $f''(s_t)$ satisfies $\boldsymbol{x} \neq \boldsymbol{x}''$. But state t_t is in $f''(s_t)$ and within t_t, for each $x'' \in W''$, $t_t(x'') = s_t(x'') = t(x) = t_t(x)$. So t_t does *not* satisfy $\boldsymbol{x} \neq \boldsymbol{x}''$, which is a contradiction.

Conversely, any state which satisfies $\neg\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'')$ is of the form s_t for some s and t, since $\mathbf{S}$ cannot change the value of any variable in W''. So, let s_t be any initial state which satisfies $\neg\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'')$, where s is the restriction of s_t to V and t'' is the restriction of s_t to W'' and where t is the state on W which corresponds to t''. We claim that $t \in f(s)$. Assume for contradiction that $t \notin f(s)$, then $\mathbf{S}$ is guaranteed to terminate on s (since otherwise $f(s)$ contains every state) and t is not a possible final state for s. So every final state in $f(s)$ differs from t. As before, let t_t be the extension of t over W'' such that $t_t(x'') = t(x)$ for all $x \in W$. Then $t_t(x'') = t_t(x)$ for all $x \in W$, and by definition of f'', every final state in $f''(s_t)$ differs from t_t. So, f'' terminates on s_t and every final state in $f''(s_t)$ satisfies $\boldsymbol{x} \neq \boldsymbol{x}''$. So $s_t \in \mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'')$ which is a contradiction.

As an example, for the program $\mathbf{S} = (x := 1 \sqcap x := 2)$ If the final state space is $\{x, y\}$, then: $\neg\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'') \Leftrightarrow (x = 1 \vee x'' = 2) \wedge y'' = y$.

In [1] we also prove the *Representation Theorem*:

Theorem 3.2. *For any statement* $\mathbf{S} : V \to W$, *let* $\tilde{\boldsymbol{y}} = V \setminus W$. *Then*:

$$\mathbf{S} \approx \neg\mathrm{WP}(\mathbf{S}, \mathbf{false});\ \boldsymbol{x} := \boldsymbol{x}'.(\neg\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}') \wedge \mathrm{WP}(\mathbf{S}, \mathbf{true}));\ \mathbf{remove}(\boldsymbol{y})$$

The representation theorem seems to imply that a third formula, namely $\mathrm{WP}(\mathbf{S}, \mathbf{false})$, is needed to fully characterise the behaviour of a program. However, this formula can be derived from the behaviour formula:

Theorem 3.3. *For any statement* $\mathbf{S}$, $\mathrm{WP}(\mathbf{S}, \mathbf{false}) \Leftrightarrow \forall \boldsymbol{x}''.\mathrm{WP}(\mathbf{S}, \boldsymbol{x} \neq \boldsymbol{x}'')$.

We define two formulae: $\mathcal{A}(\mathbf{S})$ which captures the termination properties of $\mathbf{S}$ (the *abort* states) and $\mathcal{B}(\mathbf{S})$ which captures the behaviour of $\mathbf{S}$:

$$\mathcal{A}(\mathbf{S}) =_{\mathrm{DF}} \neg \mathrm{WP}(\mathbf{S}, \mathbf{true}) \quad \text{and} \quad \mathcal{B}(\mathbf{S}) =_{\mathrm{DF}} \neg \mathrm{WP}(\mathbf{S}, x \neq x')$$

$\mathcal{A}(\mathbf{S})$ is true on precisely those initial states for which $\mathbf{S}$ can abort (not terminate), while $\mathcal{B}(\mathbf{S})$ is true on initial states for which the values of x'' are the values of x in one of the possible final states. Note that an initial state for which $\mathbf{S}$ could abort will include all possible values in the set of final states, so we would expect that $\mathcal{B}(\mathbf{S}) \Rightarrow \mathcal{A}(\mathbf{S})$ for all statements $\mathbf{S}$.

If $\mathbf{S}$ is guaranteed to terminate and satisfy $\mathbf{R}_1$ and if $\mathbf{R}_1 \Rightarrow \mathbf{R}_2$, then $\mathbf{S}$ is also guaranteed to terminate and satisfy $\mathbf{R}_2$. So the weakest precondition is monotonic in the postcondition, and, since $x \neq x'' \Rightarrow \mathbf{true}$, we have $\mathcal{B}(\mathbf{S}) \Rightarrow \mathcal{A}(\mathbf{S})$, and therefore:

$$\mathcal{A}(\mathbf{S}) \Leftrightarrow \mathcal{A}(\mathbf{S}) \wedge \mathcal{B}(\mathbf{S}) \quad \text{and} \quad \mathcal{B}(\mathbf{S}) \Leftarrow \mathcal{A}(\mathbf{S}) \vee \mathcal{B}(\mathbf{S})$$

With these definitions we can prove that $\mathcal{A}$ and $\mathcal{B}$ fully characterise the refinement property:

Theorem 3.4. *For any statements* $\mathbf{S}_1$ *and* $\mathbf{S}_2$*, with the same initial and final state spaces,* $\mathbf{S}_2$ *is a refinement of* $\mathbf{S}_1$*, written* $\mathbf{S}_1 \leq \mathbf{S}_2$*, if and only if*:

$$\mathcal{A}(\mathbf{S}_1) \Leftarrow \mathcal{A}(\mathbf{S}_2) \quad \text{and} \quad \mathcal{B}(\mathbf{S}_1) \Leftarrow \mathcal{B}(\mathbf{S}_2)$$

Note that the implications are in the opposite direction to the weakest precondition implications since $\mathcal{A}$ and $\mathcal{B}$ are both the negation of a weakest precondition.

With these definitions we can rewrite Theorem 3.2 as:

$$\mathbf{S} \approx \left[\exists x''.\mathcal{B}(\mathbf{S})\right]; \left\{\neg \mathcal{A}(\mathbf{S})\right\}; x := x'.\left(\mathcal{B}(\mathbf{S})\left[x'/x''\right]\right)$$

or, equivalently:

$$\mathbf{S} \approx \left[\exists x''.\mathcal{B}(\mathbf{S})\right]; \left\{\neg \mathcal{A}(\mathbf{S})\right\}; x := x''.\mathcal{B}(\mathbf{S})$$

The three statements may be interpreted informally as stating:

1. If there is no defined behaviour (terminating or otherwise) then the statement is null;
2. Otherwise, if $\mathcal{A}(\mathbf{S})$ is true, then the statement aborts;
3. Otherwise, new values x'' are assigned to the variables x such that $\mathcal{B}(\mathbf{S})$ is satisfied.

For null-free programs, $\mathrm{WP}(\mathbf{S}, \mathbf{false}) \Leftrightarrow \mathbf{false}$, this is Dijkstra's "Law of the Excluded Miracle" [2], and so for these programs the initial guard is always equivalent to **skip**, and $\exists x''.\mathcal{B}(\mathbf{S})$ is true.

Theorem 3.5. *Let* A *and* B *be any two formula, such that* (1) $A \Rightarrow B$ *and* (2) *No variable in* x'' *appears free in* A*. Let*

$$\mathbf{S} = \left[\exists x''.B\right]; \left\{\neg A\right\}; x := x''.B$$

Then $\mathcal{A}(\mathbf{S}) = A$ and $\mathcal{B}(\mathbf{S}) = B$.

We have proved that:

1) Given any statement $\mathbf{S}$, we can define the corresponding $\mathcal{A}(\mathbf{S})$ and $\mathcal{B}(\mathbf{S})$ by using weakest preconditions;

2) Given any two formulae, A and B, where $A \Rightarrow B$ and no variable in x'' appears in B, we can define a statement $\mathbf{S}$ such that $A \Leftrightarrow \mathcal{A}(\mathbf{S})$ and $B \Leftrightarrow \mathcal{B}(\mathbf{S})$.

These results prove that the two formulae $\mathcal{A}(\mathbf{S})$ and $\mathcal{B}(\mathbf{S})$ completely capture the semantic behaviour of statement $\mathbf{S}$.

Given *any* two formulae A and B we can define A' as the formula $\forall x''(A \wedge B)$. Then $A' \Rightarrow B$ is true and none of the variables in x'' are free in A'. So A' and B satisfy the requirements for Theorem 3.5. Note that if A and B already satisfy the requirements, then $A' \Leftrightarrow A$.

These comments motivate the definition of a function $\mathcal{S}$ which maps any two formulae A and B to a WSL statement:

$$\mathcal{S}(A, B) =_{\mathrm{DF}} \left[x''.B\right]; \left\{\neg \forall x''.(A \wedge B)\right\}; x := x''.B$$

By Theorem 3.5, $\mathcal{A}(\mathcal{S}(A, B)) = \forall x''.(A \wedge B)$ and $\mathcal{B}(\mathcal{S}(A, B)) = B$.

In the case where $A \Rightarrow B$ and none of the variables in x'' appears free in A, we have: $\mathcal{A}(\mathcal{S}(A, B)) = A$ and $\mathcal{B}(\mathcal{S}(A, B)) = A$.

Four fundamental programs are **abort**, **add**(x), **skip** and **null** for which the corresponding formulae are given in Table 1. From this table we see that:

$$\textbf{abort} \le \textbf{add}(x) \le \textbf{skip} \le \textbf{null}$$

when x is non-empty, then all three refinements are *strict* refinements. Note that **abort** and **add**(x) have the same behaviour but different termination conditions, while **add**(x) and **skip** have the same termination conditions but different behaviour. (When x is empty **add**$(\langle\ \rangle)$ is equivalent to **skip** and the formula $x \neq x'$ is equivalent to **true**.)

Examples

Some example programs to illustrate the formulae:

$$\mathbf{S}_1 = \{y > 0\};\ (x := 1 \sqcap x := 2)$$

$$\mathbf{S}_2 = \{y \ge 0\};\ x := 1$$

$$\mathbf{S}_3 = \textbf{if}\ y \ge 0 \to x := 1 \Box\, y \le 0 \to x := 3\ \textbf{fi}$$

$$\mathbf{S}_4 = [y \ge 0];\ x := 1$$

Here, $\mathbf{S}_2$ is both more defined (terminating on more initial states) and more deterministic than $\mathbf{S}_1$, and so $\mathbf{S}_2$ is a refinement of $\mathbf{S}_1$. A refinement of a program can define any behaviour for initial states on which the original program aborts, so $\mathbf{S}_3$ is also a refinement of $\mathbf{S}_1$. Finally, $\mathbf{S}_4$ is more deterministic than $\mathbf{S}_3$ in that it restricts the set of possible final states (for initial states with $y < 0$ the set of final states is empty).

These facts are reflected in the formulae in Table 2.

Clearly, $\mathcal{A}(\mathbf{S}_2) \Rightarrow \mathcal{A}(\mathbf{S}_1)$ and $\mathcal{B}(\mathbf{S}_2) \Rightarrow \mathcal{B}(\mathbf{S}_1)$ which shows that $\mathbf{S}_1 \le \mathbf{S}_2$. Also $\mathcal{A}(\mathbf{S}_3) \Rightarrow \mathcal{A}(\mathbf{S}_1)$ and $\mathcal{B}(\mathbf{S}_3) \Rightarrow \mathcal{B}(\mathbf{S}_1)$ which shows that $\mathbf{S}_1 \le \mathbf{S}_3$. However, it is not the case that $\mathbf{S}_2 \le \mathbf{S}_3$ since when $y = 0$ initially, $\mathbf{S}_2$ must assign x the value 1, while $\mathbf{S}_3$ can non-deterministically choose to assign x the value 1 or 3. Conversely, $\mathbf{S}_3$ is not refined by $\mathbf{S}_2$ because $\mathbf{S}_3$ is defined on initial states for which $\mathbf{S}_2$ is not defined: namely, those initial states in which $y < 0$.

Finally, $\mathbf{S}_4$ is a (strict) refinement of all the other programs, and none of the other programs is a refinement of $\mathbf{S}_4$ because $\mathbf{S}_4$ is null on initial states with $y < 0$.

Given that $\mathcal{A}$ and $\mathcal{B}$ capture the semantics of a program, it should be possible to compute the formulae for a compound statement from the formulae for the components. For the primitive statements in the first level of WSL, the formulae are given in Table 3.

Table 1. $\mathcal{A}$ and $\mathcal{B}$ for some fundamental programs.

	$\mathcal{A}$	$\mathcal{B}$
abort	**true**	**true**
add(x)	**false**	**true**
skip	**false**	$x' = x$
null	**false**	**false**

Table 2. $\mathcal{A}$ and $\mathcal{B}$ for the example programs.

	$\mathcal{A}$	$\mathcal{B}$
$\mathbf{S}_1$	$y \le 0$	$y \le 0 \vee (x' = 1 \vee x' = 2 \vee y' = y)$
$\mathbf{S}_2$	$y < 0$	$y < 0 \vee x' = 1$
$\mathbf{S}_3$	**false**	$(y \ge 0 \wedge x' = 1 \vee y \le 0 \wedge x' = 3) \wedge y' = y$
$\mathbf{S}_4$	**false**	$y \ge 0 \wedge x' = 1 \wedge y' = y$

Table 3. $\mathcal{A}$ and $\mathcal{B}$ for the atomic statements.

	$\mathcal{A}$	$\mathcal{B}$
$\{\mathbf{P}\}$	$\mathbf{P}$	$\neg\mathbf{P} \vee x'' = x$
$[\mathbf{P}]$	**false**	$\mathbf{P} \wedge x'' = x$
$x := e$	**false**	$x'' = e$
$x := x'.\mathbf{Q}$	$\neg\exists x'.\mathbf{Q}$	$\neg\exists x'.\mathbf{Q} \vee \mathbf{Q}[x''/x']$

4. Computing $\mathcal{A}$ and $\mathcal{B}$ for Compound Statements

Given the two formulae $\mathcal{A}(\mathbf{S})$ and $\mathcal{B}(\mathbf{S})$, which are in effect, the weakest preconditions on $\mathbf{S}$ for two particular postconditions, we can determine the weakest precondition for any given postcondition. This means that we can compute $\mathcal{A}$ and $\mathcal{B}$ for any compound statement, given the corresponding formulae for the component statements.

For nondeterministic choice:

$$\begin{aligned}\mathcal{A}(\mathbf{S}_1 \sqcap \mathbf{S}_2) &\Leftrightarrow \neg\mathrm{WP}(\mathbf{S}_1 \sqcap \mathbf{S}_2, \mathbf{true})\\ &\Leftrightarrow \neg\mathrm{WP}(\mathbf{S}_1, \mathbf{true}) \wedge \mathrm{WP}(\mathbf{S}_2, \mathbf{true})\\ &\Leftrightarrow \neg\mathrm{WP}(\mathbf{S}_1, \mathbf{true}) \vee \neg\mathrm{WP}(\mathbf{S}_2, \mathbf{true})\\ &\Leftrightarrow \mathcal{A}(\mathbf{S}_1) \vee \mathcal{A}(\mathbf{S}_2)\end{aligned}$$

and similarly:

$$\mathcal{B}(\mathbf{S}_1 \sqcap \mathbf{S}_2) \Leftrightarrow \mathcal{B}(\mathbf{S}_1) \vee \mathcal{B}(\mathbf{S}_2)$$

For deterministic choice:

$$\begin{aligned}&\mathcal{A}(\textbf{if } \mathbf{B} \textbf{ then } \mathbf{S}_1 \textbf{ else } \mathbf{S}_2 \textbf{ fi})\\ &\Leftrightarrow \neg\big((\mathbf{B} \Rightarrow \mathrm{WP}(\mathbf{S}_1, \mathbf{true})) \wedge (\neg\mathbf{B} \Rightarrow \mathrm{WP}(\mathbf{S}_2, \mathbf{true}))\big)\\ &\Leftrightarrow \big(\mathbf{B} \wedge \neg\mathrm{WP}(\mathbf{S}_1, \mathbf{true})\big) \vee \big(\neg\mathbf{B} \wedge \neg\mathrm{WP}(\mathbf{S}_2, \mathbf{true})\big)\\ &\Leftrightarrow \big(\mathbf{B} \wedge \mathcal{A}(\mathbf{S}_1)\big) \vee \big(\neg\mathbf{B} \wedge \mathcal{A}(\mathbf{S}_2)\big)\end{aligned}$$

and similarly:

$$\begin{aligned}&\mathcal{B}(\textbf{if } \mathbf{B} \textbf{ then } \mathbf{S}_1 \textbf{ else } \mathbf{S}_2 \textbf{ fi})\\ &\Leftrightarrow \big(\mathbf{B} \wedge \mathcal{B}(\mathbf{S}_1)\big) \vee \big(\neg\mathbf{B} \wedge \mathcal{B}(\mathbf{S}_2)\big)\end{aligned}$$

For sequential composition:

$$\begin{aligned}\mathcal{A}(\mathbf{S}_1;\mathbf{S}_2) &\Leftrightarrow \neg\mathrm{WP}(\mathbf{S}_1, \mathrm{WP}(\mathbf{S}_2, \mathbf{true}))\\ &\Leftrightarrow \neg\mathrm{WP}(\mathbf{S}_1, \neg\mathcal{A}(\mathbf{S}_2))\\ &\Leftrightarrow \neg\big(\neg\exists x''.\mathcal{B}(\mathbf{S}_1) \vee \neg\mathcal{A}(\mathbf{S}_1) \wedge \forall x''.(\mathcal{B}(\mathbf{S}_1) \Rightarrow \neg\mathcal{A}(\mathbf{S}_2)[x''/x])\big)\\ &\Leftrightarrow \exists x''.\mathcal{B}(\mathbf{S}_1) \wedge \mathcal{A}(\mathbf{S}_1) \vee \exists x''.(\mathcal{B}(\mathbf{S}_1) \wedge \mathcal{A}(\mathbf{S}_2)[x''/x])\end{aligned}$$

In other words, for $\mathbf{S}_1;\mathbf{S}_2$ to abort on an initial state, $\mathbf{S}_1$ must not be null and either $\mathbf{S}_1$ aborts or $\mathbf{S}_1$ terminates in a state in which $\mathbf{S}_2$ aborts.

To compute $\mathcal{B}(\mathbf{S}_1;\mathbf{S}_2)$ we need to compute $\mathrm{WP}(\mathbf{S}_1, \mathrm{WP}(\mathbf{S}_2, x \neq x''))$, but this contains a postcondition which includes variables in x''. We can solve this problem by computing the formula $\mathrm{WP}(\mathbf{S}_1, \mathrm{WP}(\mathbf{S}_2, x \neq x'))$ and then renaming x' to x'' in the result. Note that the postcondition $\mathrm{WP}(\mathbf{S}_2, x \neq x')$ is simply $\neg\mathcal{B}(\mathbf{S}_2)[x'/x'']$.

$$\begin{aligned}
\mathcal{B}(\mathbf{S}_1;\mathbf{S}_2) &\Leftrightarrow \neg \mathrm{WP}(\mathbf{S}_1, \mathrm{WP}(\mathbf{S}_2, x \neq x'))[x''/x'] \\
&\Leftrightarrow \neg \mathrm{WP}(\mathbf{S}_1, \neg\mathcal{B}(\mathbf{S}_2)[x'/x''])[x''/x'] \\
&\Leftrightarrow \neg\left(\neg\exists x''.\mathcal{B}(\mathbf{S}_1) \vee \neg\mathcal{A}(\mathbf{S}_1) \wedge \forall x''.\left(\mathcal{B}(\mathbf{S}_1) \Rightarrow \neg\mathcal{B}(\mathbf{S}_2)[x'/x''][x''/x]\right)\right)[x''/x'] \\
&\Leftrightarrow \exists x''.\mathcal{B}(\mathbf{S}_1) \wedge \left(\mathcal{A}(\mathbf{S}_1) \vee \exists x''.\left(\mathcal{B}(\mathbf{S}_1) \wedge \mathcal{B}(\mathbf{S}_2)[x'/x''][x''/x]\right)\right)[x''/x'] \\
&\Leftrightarrow \exists x''.\mathcal{B}(\mathbf{S}_1) \wedge \left(\mathcal{A}(\mathbf{S}_1) \vee \exists x'.\left(\mathcal{B}(\mathbf{S}_1)[x'/x''] \wedge \mathcal{B}(\mathbf{S}_2)[x'/x]\right)\right)
\end{aligned}$$

In other words, for x'' to be a possible final state for $\mathbf{S}_1;\mathbf{S}_2$ on initial state x, either $\mathbf{S}_1$ aborts or there exists some intermediate state x' which is a possible final state for $\mathbf{S}_1$ and for which $\mathbf{S}_2$ gives x'' as a possible final state when started in this initial state.

Recursion $\left(\mu X_y^x.\mathbf{S}\right)$ is defined in terms of the set of all finite truncations. Define:

$$\left(\mu X_y^x.\mathbf{S}\right)^0 =_{\mathrm{DF}} \mathbf{abort}; \mathbf{add}(x); \mathbf{remove}(y)$$

$$\left(\mu X_y^x.\mathbf{S}\right)^{n+1} =_{\mathrm{DF}} \mathbf{S}\left[\left(\mu X_y^x.\mathbf{S}\right)^n / X\right] \quad \text{forall } n < \omega$$

Then for any postcondition $\mathbf{R}$:

$$\mathrm{WP}\left(\left(\mu X_y^x.\mathbf{S}\right), \mathbf{R}\right) =_{\mathrm{DF}} \bigvee_{n<\omega}\left(\left(\mu X_y^x.\mathbf{S}\right)^n, \mathbf{R}\right)$$

So:

$$\mathcal{A}\left(\left(\mu X_y^x.\mathbf{S}\right)\right) \equiv \bigwedge_{n<\omega} \mathcal{A}\left(\left(\mu X_y^x.\mathbf{S}\right)^n\right) \quad \text{and} \quad \mathcal{B}\left(\left(\mu X_y^x.\mathbf{S}\right)\right) \equiv \bigwedge_{n<\omega} \mathcal{B}\left(\left(\mu X_y^x.\mathbf{S}\right)^n\right)$$

5. Temporal Logic

Temporal logic [3] is a class of logical theories for reasoning about propositions qualified in terms of time and can therefore be used to reason about finite and infinite sequences of states. These sequences can define an operational semantics for programs which maps each initial state to the set of possible histories: where a history is a possible sequence of states in the execution of a program. Since a formula can describe an infinite sequence of states: and therefore a non-terminating program, there would appear to be no need for the special state $\perp$ to indicate non-termination, and therefore it would appear possible to represent the operational semantics of a program using a single formula in temporal logic.

This turns out not to be the case: a single formula is not sufficient, but two formulae (the temporal equivalent of the abort and behaviour predicates defined here) are sufficient. Lack of space precludes a full discussion of these questions in this paper.

References

[1] Ward, M. (2004) Pigs from Sausages? Reengineering from Assembler to C via Fermat Transformations. *Science of Computer Programming, Special Issue on Program Transformation*, **52**, 213-255. http://www.gkc.org.uk/martin/papers/migration-t.pdf

[2] Dijkstra, E.W. (1976) A Discipline of Programming. Prentice-Hall, Englewood Cliffs.

[3] Moszkowski, B. (1994) Some Very Compositional Temporal Properties. In: Olderog, E.-R., Ed., *Programming Concepts, Methods and Calculi*, IFIP Transactions, North-Holland Publishing Co., Amsterdam, 307-326.