The Eventual Predecessor Predicate Transformer

The immediate predecessor transformer $\rho \diamond \psi$ can be iterated to yield the eventual predecessor transformer:

$$\rho^* \diamond \psi = \psi \lor \rho \diamond \psi \lor \rho \diamond (\rho \diamond \psi) \lor \rho \diamond (\rho \diamond (\rho \diamond \psi)) \lor \cdots$$

Obviously, $\rho^* \diamond \psi$ characterizes all the states from which it is possible to reach a $\psi$-state by 0 or more $\rho$-steps.

A state $s$ is called feasible if it initiates a fair run.

Let $D$ be an FDS. We denote by $D_T$ the FDS obtained from $D$ by replacing the initial condition by the trivial assertion $T$ (true). The state-transition graph $G_{DT}$ represents all the possible $D$-states, including some which are not reachable by $D$. 
A Symbolic Algorithm for Model Checking Response

**Algorithm** set-feasible \((\mathcal{D})\) : assertion — Calculate the set of \(\mathcal{D}_T\)-states initiating a fair \(\mathcal{D}\)-run, using symbolic operations

\[
\text{new, old : assertion}
\]

1. \(\text{old} := 0\)
2. \(\text{new} := 1\)
3. while \((\text{new} \neq \text{old})\) do
   begin
   \hspace{1cm} \(\text{old} := \text{new}\)
   \hspace{1cm} \(\text{new} := \text{new} \land (\rho_D \Diamond \text{new})\)
   \hspace{1.5cm} —— Only retain states which have a successor within \text{new}\n   \hspace{1cm} \textbf{for each} \(J \in \mathcal{J}\) \textbf{do}
   \hspace{1.5cm} \(\text{new} := (\text{new} \land \rho_D)^* \Diamond (\text{new} \land J)\)
   \hspace{1.5cm} —— Only retain states with a \text{new}-path leading to a \(J\)-state
   \hspace{1cm} \textbf{for each} \((p, q) \in \mathcal{C}\) \textbf{do}
   \hspace{1.5cm} \(\text{new} := \left\{ \begin{array}{l}
\text{new} \land \neg p \\
\vee (\text{new} \land \rho_D)^* \Diamond (\text{new} \land q)
\end{array} \right.\)
   \hspace{1.5cm} —— Retain states violating \(p\) or having a \text{new}-path leading to a \(q\)-state
   end
4. \(\text{return}(\rho_D^* \Diamond \text{new})\)
Correctness of the Algorithm

**Claim 5.** Algorithm SET-FEASIBLE terminates, with state $s$ satisfying SET-FEASIBLE($D$) iff there exists a $G_{DT}$-path leading from $s$ to a fair subgraph of $G_{DT}$.

The proof is partitioned into three parts:

1. **The Algorithm terminates:** We define an ordering relation on assertions by letting

   $$p \leq q \iff \|p\| \leq \|q\|.$$

   Denote by $new^j_i$ the assertion which is the (symbolic) value of variable $new$ at the $j$th visit to line $i$ (before executing line $i$).

   Since all operations applied to variable $new$ are of the form $new \land E$ or a disjunction of such expressions, it is easy to see that lines 5, 7, and 9 only remove states from $new$. Therefore, we have that $new^{j+1}_3 \leq new^j_3 = old^{j+1}_3$ for all $j = 1, 2, \ldots$.

   Since $G_{DT}$ is finite, the algorithm must terminate.
Correctness of the Algorithm: Completeness

Next, we prove that Algorithm \textsc{set-feasible} is complete. Namely, if $S$ is a fair subgraph of $G_{D_T}$ and $s$ is a state leading to $S$, then $s \in \|\textsc{set-feasible}(D)\|$.

To do so, we show that $S \subseteq \|new_{10}\|$ from which the claim of completeness follows.

The above inclusion follows by induction on the number of steps performed by the algorithm, where the induction basis is provided by

$$S \subseteq G_{D_T} = \|1\| = \|new^1\|,$$

and the induction step is supported by the fact that, due to $S$ being a fair subgraph, $S \subseteq \|new\|$ implies the following:

$$S \subseteq \|\text{new} \land (\rho_D \lozenge \text{new})\|$$
$$S \subseteq \| (\text{new} \land \rho_D)^\ast \lozenge (\text{new} \land J)\| \quad \text{For every } J \in \mathcal{J}$$
$$S \subseteq \| \left( \lor (\text{new} \land \lnot p) \land (\text{new} \land q) \right)\| \quad \text{For every } (p, q) \in \mathcal{C}$$
Algorithm Correctness: Soundness

As finally, we show that the algorithm is sound. Namely, if \( s \in \text{SET-FEASIBLE}(D) \) then there exists \( S \), a fair subgraph of \( G_{DT} \), and a path leading from \( s \) to \( S \).

When the algorithm terminates, we know that

1. Every \( s \in \|new_{10}\| \) has a successor \( s' \in \|new_{10}\| \).
2. Every \( s \in \|new_{10}\| \) initiates a \( \|new_{10}\| \)-path leading to a \( J \)-state, for every \( J \in J \).
3. Every \( s \in \|new_{10}\| \) initiates a \( \|new_{10}\| \)-path leading to a \( q \)-state or satisfies \( \neg p \), for every \( (p, q) \in C \).

Assume that \( s \in \|\text{SET-FEASIBLE}(D)\| \). Line 10 implies that \( s \) is connected by a path \( \pi \) to a \( \|new_{10}\| \)-state. Repeat the following successive extensions of \( \pi \) ad-infinitum, denoting the last state of \( \pi \) by \( s_{\ell} \):

1. Extend \( \pi \) by a \( \|new_{10}\| \)-successor of \( s_{\ell} \), guaranteed by P1.
2. For every \( J \in J \), extend \( \pi \) by a \( \|new_{10}\| \)-path leading to a \( J \)-state, guaranteed by P2.
3. For every \( (p, q) \in C \), if there exists a \( \|new_{10}\| \)-path \( \pi' \) connecting \( s_{\ell} \) to a \( q \)-state, then extend \( \pi \) by \( \pi' \). Otherwise, do not extend \( \pi \). When done, go back to 1..

Can show that \( S = \text{Inf}(\pi) \) is an \( s \)-reachable fair subgraph.
Relation to Previous Work

- Model checking of LTL with full fairness was proposed first in [LP85] and independently in [EL85]. The algorithms were applied to explicit state elaboration of the state-space, and relied on the construction of an LTL tableau and its composition with the system. Can be interpreted also as algorithms for checking the emptiness of a Street Automaton [LP85], [VW86].
- [LP85] also contained fix-point expressions for the calculation of $E_f G r$ under weak fairness. These were later implemented in most symbolic model checkers, e.g., [BCMDH92].
- Efficient symbolic model checking of LTL has been proposed in [CGH94], based on the construction of additional modules, serving as LTL testers. Only weak fairness was considered. Our approach improves on [CGH94] in the direct treatment of compassion and not relying on a reduction into CTL.
- All previous treatments of compassion suggested adding it as an antecedent to the LTL property we wish to verify.
Model Checking Response Properties

We denote by $\mathcal{D}_{\neg q}$ the FDS obtained from FDS $\mathcal{D}$ by replacing the transition relation $\rho_{\mathcal{D}}$ by the transition relation

$$\rho_{\neg q} : \quad \neg q \land \rho_{\mathcal{D}} \land \neg q'$$

this transition relation connects state $s$ with state $\tilde{s}$ iff $\tilde{s}$ is a $\mathcal{D}$-successor of $s$, and neither state satisfies $q$.

Algorithm SMC-RESP ($\mathcal{D}, p, q$) : assertion — Check that FDS $\mathcal{D}$ satisfies $p \rightsquigarrow q$, using symbolic operations

$\text{cycles, pending : assertion}$

1. $\text{cycles} := \text{SET-FEASIBLE}(\mathcal{D}_{\neg q})$
   — — Compute all states initiating a fair $\neg q$-run.

2. $\text{pending} := p \land \text{cycles}$
   — — All $p$-states initiating a fair $\neg q$-run.

3. $\text{return } \Theta_{\mathcal{D}} \land (\rho^*_{\mathcal{D}} \Diamond \text{pending})$
   — — All initial states leading to $p$-states initiating a fair $\neg q$-run.

Claim 6. Algorithm SMC-RESP returns a vacuous (unsatisfiable, $= 0$) assertion iff $\mathcal{D}$ satisfies $p \rightsquigarrow q$. 
Model Checking Accessibility

Accessibility for process $P_1$ of MUX-SEM can be specified by the response property

$$T_1 \leadsto C_1$$

Invoking $\text{SET-FEASIBLE}(\text{MUX-SEM}_{\neg C_1})$, we get:

\[
\begin{align*}
\text{next}^1_3 & : 1 \\
\text{next}^2_3 & : \neg C_1 = N_1 \lor T_1 \\
\text{next}^3_3 & : N_1 \lor (T_1 \land y = 0) \\
\text{next}^4_3 = \text{next}^1_{10} & : N_1 \lor (T_1 \land y = 0 \land \neg C_2)
\end{align*}
\]

Computing $\text{pending}$, we get $\text{pending} = T_1 \land y = 0 \land \neg C_2$.

Intersecting with the reachable states, we get 0 (false).

We conclude that MUX-SEM has the property of accessibility.
The TLV System

Recall the schematic presentation of the SMC-INV algorithm:

Algorithm $\text{SMC-INV} (\mathcal{D}, p) : \text{assertion} —$ Check that FDS $\mathcal{D}$ satisfies $\text{Inv}(p)$, using symbolic operations

1. $old := 0$
2. $new := \neg p$
3. while ($new \neq old$) do
   begin
   4. $old := new$
   5. $new := new \lor (\rho_D \lozenge new)$
   end
4. return $\Theta_D \land new$

Programming it in TLV-BASIC

Func smc-inv(p);
   Local old := 1;
   Local new := 0;
   While (! (old = new))
      Let old := new;
      Let new := old | pred(total, old);
      If (new & _i)
         Let old := new;
      End -- If
   End -- end while
   Return new & _i;
End -- Func smc-inv(p);
A Response MC Algorithm which Provides Counter-Examples

Algorithm SMC-RESP \((D, p, q)\) — Model Check \(p \rightsquigarrow q\) providing counter-examples

\[
\text{cycles, rpend} : \text{assertion}
\]

\[
cycles := \text{SET-FEASIBLE}(D_q) \quad \text{— — All states initiating a fair } \neg q-\text{run}
\]

\[
rpend := p \land cycles \land (\Theta_D \diamond \rho_D^*) \quad \text{— — All reachable pending states}
\]

\[
\text{if } rpend = 0 \text{ then [print "Property is Valid"; return]}
\]

print "Property is Invalid. Counter-Example Follows"

\[
R := cycles \land \rho_D \land cycles' \quad \text{— — Restrict to transitions within } cycles
\]

\[
(position, psize) := (1, 0)
\]

\[
gpath(\Theta_D, rpend, \rho_D, \text{prefix, psize}) \quad \text{— — A path from } \Theta_D \text{ to } rpend
\]

\[
s := \text{prefix[psize]} \quad \text{— — The closest reachable pending state}
\]

\[
\text{while } (s \diamond R^*) \land \neg (R^* \diamond s) \neq 0 \text{ do}
\]

\[
s := \text{sat}((s \diamond R^*) \land \neg (R^* \diamond s)) \quad \text{— — Search for a terminal MSCS}
\]

\[
gpath(\text{prefix[psize]}, s, R, \text{prefix, psize}) \quad \text{— — Extend path to } s
\]

print "Prefix of Counter-Example:"

\[
\text{array-print(prefix, psize - 1, position)} \quad \text{— — Print ctr-example prefix}
\]

\[
(ps size, \text{period[1]}, \text{period[2]}) := (2, s, \text{sat}(s \diamond R)) \quad \text{— — Init. } period
\]

\[
\text{for each } J \in J \text{ do}
\]

\[
gpath(period[psize], J, R, \text{period, psize}) \quad \text{— — Visit next justice set}
\]

\[
\text{for each } (p, q) \in C \text{ do}
\]

\[
\text{if } (period[psize] \diamond R^*) \land q \neq 0 \text{ then}
\]

\[
gpath(period[psize], q, R, \text{period, psize}) \quad \text{— — Visit next compassion}
\]

\[
gpath(period[psize], s, R, \text{period, psize}) \quad \text{— — Close cycle}
\]

print "Repeating Period"

\[
\text{array-print(period, psize - 1, position)} \quad \text{— — Print ctr-example period}
\]
Tlv-Basic Implementation of gpath

Proc ngpath(source, destination, R, &arr, &asize);
  Local new := destination;
  Local old := 0;  Local pos := 1;  Let bpath[1] := new;
  While (!(old = new))
    Let old := new;
    If (null(new & source))
      Let new := old | pred(R,old);
      If (!(old = new))
        Let pos := pos + 1;
        Let bpath[pos] := new & !old;
      End -- If (!(old = new))
    End -- If (null(new & source))
  End -- While (!(old = new))
  If (new & source)
    If (asize = 0)
      Let asize := asize + 1;
      Let arr[asize] := sat(new & source);
    End -- If (asize = 0)
    While (pos)
      Let pos := pos - 1;
      If (pos)
        Let arr[asize+1] := sat(succ(R,arr[asize]) & bpath[pos]);
        Let asize := asize + 1;
      End -- If (pos)
    End -- While (pos)
  End -- If (new & source)
End -- Proc ngpath(source, destination, R, &arr);
destination = bpath[1]

bpath[2]

bpath[3]

\ldots

bpath[asize]