The Eventual Predecessor Predicate Transformer

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

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

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

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

Let $\mathcal{D}$ be an FDS. We denote by $\mathcal{D}_T$ the FDS obtained from $\mathcal{D}$ by replacing the initial condition by the trivial assertion T (true). The state-transition graph $G_{\mathcal{D}_T}$ represents all the possible $\mathcal{D}$-states, including some which are not reachable by $\mathcal{D}$.

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

new, old : **assertion**

1. $old := 0$
2. $new := 1$
3. **while** $(new \neq old)$ **do**

   begin

4. $old := new$
5. $new := new \land (\rho_{\mathcal{D}} \diamond new)$

   — — Only retain states which have a successor within new

6. **for each** $J \in \mathcal{J}$ **do**
7. $new := (new \land \rho_{\mathcal{D}})^* \diamond (new \land J)$

   — — Only retain states with a new-path leading to a J-state

8. **for each** $(p, q) \in \mathcal{C}$ **do**
9. $new := \left(\begin{array}{ll} & new \land \neg p \\ \lor & (new \land \rho_{\mathcal{D}})^* \diamond (new \land q) \end{array} \right)$

   — — Retain states violating p or having a new-path leading to a q-stat

   end

10. **return**$(\rho_{\mathcal{D}}^* \diamond new)$

Correctness of the Algorithm

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

The proof is partitioned into three parts:

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

$$p \preceq q \quad \Longleftrightarrow \quad \|p\| \subseteq \|q\|.$$

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

Since all operations applied to variable new are of the form $new \wedge E$ 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 $new_3^{j+1} \preceq new_3^j = old_3^{j+1}$ for all $j = 1, 2, \ldots$.

Since $G_{\mathcal{D}_T}$ is finite, the algorithm must terminate.

Course G22.3033.007 Lecture 6 R. Dewar and A. Pnueli

Correctness of the Algorithm: Completeness

Next, we prove that Algorithm SET-FEASIBLE is complete. Namely, if S is a fair subgraph of $G_{\mathcal{D}_T}$ and s is a state leading to S, then $s \in \|\text{SET-FEASIBLE}(\mathcal{D})\|$.

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

$$S \subseteq G_{\mathcal{D}_T} = \|1\| = \|new_3^1\|,$$

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

$$S \subseteq \|new \wedge (\rho_{\mathcal{D}} \diamond new)\|$$

$$S \subseteq \|(new \wedge \rho_{\mathcal{D}})^* \diamond (new \wedge J)\| \qquad \text{For every } J \in \mathcal{J}$$

$$S \subseteq \left\| \begin{pmatrix} & new \wedge \neg p \\ \vee & (new \wedge \rho_{\mathcal{D}})^* \diamond (new \wedge q) \end{pmatrix} \right\| \qquad \text{For every } (p,q) \in \mathcal{C}$$

Algorithm Correctness: Soundness

-.3cm Finally, we show that the algorithm is sound. Namely, if $s \in \|\text{SET-FEASIBLE}(\mathcal{D})\|$ then there exists S, a fair subgraph of $G_{\mathcal{D}_T}$, and a path leading from s to S.

When the algorithm terminates, we know that

P1. Every $s \in \|new_{10}\|$ has a successor $s' \in \|new_{10}\|$.
P2. Every $s \in \|new_{10}\|$ initiates a $\|new_{10}\|$-path leading to a J-state, for every $J \in \mathcal{J}$.
P3. Every $s \in \|new_{10}\|$ initiates a $\|new_{10}\|$-path leading to a q-state or satisfies $\neg p$, for every $(p, q) \in \mathcal{C}$.

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

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

Can show that $S = \mathit{Inf}(\pi)$ 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 $\mathbf{E}_f\mathbf{G}\, r$ 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

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

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

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

$cycles, pending$: **assertion**

1. $cycles :=$ SET-FEASIBLE$(\mathcal{D}_{\neg q})$
   — — Compute all states initiating a fair $\neg q$-run.
2. $pending := p \ \wedge \ cycles$
   — — All p-states initiating a fair $\neg q$-run.
3. **return** $\Theta_{\mathcal{D}} \ \wedge \ (\rho^*_{\mathcal{D}} \diamond \ pending)$
   — — 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 $p \leadsto q$.*

Model Checking Accessibility

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

$$T_1 \quad \leadsto \quad C_1$$

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

$$\begin{array}{ll}
next_3^1 : & 1 \\
next_3^2 : & \neg C_1 \quad = \quad N_1 \ \vee \ T_1 \\
next_3^3 : & N_1 \ \vee \ (T_1 \ \wedge \ y = 0) \\
next_3^4 = next_{10} : & N_1 \ \vee \ (T_1 \ \wedge \ y = 0 \ \wedge \ \neg C_2)
\end{array}$$

Computing $pending$, we get $pending = T_1 \ \wedge \ y = 0 \ \wedge \ \neg C_2$.

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

new, old : **assertion**

1. $old := 0$
2. $new := \neg p$
3. **while** $(new \neq old)$ **do**
   begin
4. $old := new$
5. $new := new \vee (\rho_{\mathcal{D}} \diamond new)$
   end
6. **return** $\Theta_{\mathcal{D}} \wedge new$

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 $(\mathcal{D}, p, q)$ — Model Check $p \rightsquigarrow q$ providing counter-examples

$cycles, rpend$: **assertion**

$cycles :=$ SET-FEASIBLE$(\mathcal{D}_{\neg q})$ — — All states initiating a fair $\neg q$-run

$rpend := p \wedge cycles \wedge (\Theta_{\mathcal{D}} \diamond \rho^*_{\mathcal{D}})$ — — All reachable pending states

if $rpend = 0$ **then [print "Property is Valid"; return]**

print "Property is Invalid. Counter-Example Follows"

$R := cycles \wedge \rho_{\mathcal{D}} \wedge cycles'$ — — Restrict to transitions within $cycles$

$(position, psize) := (1, 0)$

$gpath(\Theta_{\mathcal{D}}, rpend, \rho_{\mathcal{D}}, prefix, psize)$ — — A path from $\Theta_{\mathcal{D}}$ to $rpend$

$s := prefix[psize]$ — — The closest reachable pending state

while $(s \diamond R^*) \wedge \neg(R^* \diamond s) \neq 0$ **do**

  $s := sat((s \diamond R^*) \wedge \neg(R^* \diamond s))$ — — Search for a terminal MSCS

$gpath(prefix[psize], s, R, prefix, psize)$ — — Extend path to s

print "Prefix of Counter-Example:"

$array\_print(prefix, psize - 1, position)$ — — Print ctr-example prefix

$(psize, period[1], period[2]) := (2, s, sat(s \diamond R))$ — — Init. $period$

for each $J \in \mathcal{J}$ **do**

  $gpath(period[psize], J, R, period, psize)$ — — Visit next justice set

for each $(p, q) \in \mathcal{C}$ **do**

  if $(period[psize] \diamond R^*) \wedge q \neq 0$ **then**

  $gpath(period[psize], q, R, period, psize)$ — — Visit next compassion

$gpath(period[psize], s, R, period, psize)$ — — Close cycle

print "Repeating Period"

$array\_print(period, psize - 1, position)$ — — 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);
```

Course G22.3033.007 Lecture 6 R. Dewar and A. Pnueli

Illustrate