Appendix A - EMC Model Checker Skeleton

This section presents a classical CTL Model Checking algorithm by Emerson, Clarke and Sistla. It has good time complexity, but needs both the successor and predecessor relations. It doesn't contain a counterexamples and witnesses facility. See my thesis for an algorithm which doesn't have these drawbacks.

```
proc global_model_checker(f) ≡
  init_formulas(f);                     Create formula evaluation order.
  for i := 1 to length(f) do            Evaluate formulas bottom-up
      foreach s ∈ S do
        reset_label(s, i);              Initialize subformula i to False
      od;
      label_graph(i);                   Evaluate i, access results by labeled(s, i)
  od;
.
```

Figure 1: Main loop of a global model checker.

```
proc label_graph(i) ≡
  ftype := formula_type(sub_f[i]);
  if ftype = atomic_proposition
    then
        atomic(i);                          Process atomic subformula
  elsif ftype = NOT
      then
          negation(i);                      Process negation
  elsif ftype = AND
      then
          conjunction(i);                   Process conjunction
  elsif ftype = AX
      then
          ax(i);                            Process formula AX(p)
  elsif ftype = AU
      then
          au(i);                            Process formula A[p U q]
  elsif ftype = EU
      then
          eu(i);                            Process formula E[p U q]
  fi;
.
```

Figure 2: Choosing processing subroutine for different formula types.

```
proc atomic(i) ≡
  foreach s ∈ S do
    if evaluate_proposition(s, i);          Proposition True in state s?
      then
          add_label(s, i);    Change formula labeling to True in s
    fi;
  od;
.
```

Figure 3: Process atomic proposition.

```
proc negation(i) ≡
  foreach s ∈ S do
    if ¬labeled(s, left(i));                    Left formula False?
      then
          add_label(s, i);    Change formula labeling to True in s
    fi;
  od;
.
```

Figure 4: Process negation.

```
proc conjunction(i) ≡
  foreach s ∈ S do
    if labeled(s, left(i)) ∧ labeled(s, right(i))        Both True?
      then
          add_label(s, i);                     Yes. → Formula True.
    fi;
  od;
.
```

Figure 5: Process conjunction.

```
proc ax(i) ≡
  foreach s ∈ S do
    add_label(s, i);            We suppose formula i is True in s.
    foreach t ∈ successors(s) do
      if ¬labeled(t, left(i))
        then
            reset_label(s, i);       Contradiction! → i isn't True.
            break;
      fi;
    od;
  od;
.
```

Figure 6: Process universal next-state formula.

```
proc au(i) ≡
  foreach s ∈ S do
    reset_marked(s);
  od;
  foreach s ∈ S do
    if ¬marked(s)                          Formula i not evaluated in s?
      then
        check_au(i, s);                    Check formula i in s
    fi;
  od;
.
```

Figure 7: Process universal until formula.

```
proc check_au(i, s) ≡
  if marked(s)                  Formula i evaluated in s or loop found?
    then
        if labeled(s, i)   Labeled gets the right value in both cases!
          then
            return True;
          else
            return False;
        fi;
  fi;
  set_marked(s);             Mark state s reached and formula evaluated
  if labeled(s, right(i))
    then
        add_label(s, i);                   right(i) True → i True
        return True;
  elsif ¬labeled(s, left(i))
      then
          return False;         left(i) ∧ right(i) False → i False
  fi;
  foreach t ∈ successors(s) do   left(i) True, check all successor for i
    if ¬check_au(i, t) then return False; fi
  od;
  add_label(s, i);                         All successors OK, i True
  return True;
.
```

Figure 8: Process universal until formula for state s.

```
proc eu(i) ≡
  foreach s ∈ S do
    reset_marked(s);
  od;
  foreach s ∈ S do
    if ¬marked(s)                                Formula i not evaluated in s?
      then
          check_eu(i, s);                        Check formula i in s
    fi;
  od;
.
```

Figure 9: Process existential until formula.

```
proc check_eu(i, s) ≡
  if labeled(s, right(i))
    then
        add_label(s, i);                         Formula is True directly
        label_predecessors(i, s);                Label states which can reach s
  fi;
.
```

Figure 10: Process existential until formula for state s.

```
proc label_predecessors(i, s) ≡
  set_marked(s);                                 Never visit same state twice
  foreach t ∈ predecessors(s) do
    if ¬marked(t) ∧ labeled(t, left(i))          i is True in predecessor ?
      then
          add_label(t, i);                       Yes. → Label predecessor
          label_predecessors(i, t);              Recursively depth-first search
    fi;
  od;
.
```

Figure 11: Propagate label change to predecessor states.