Networks and Processes: Model Checking $CTL^*$ and $CTL$

Keijo Heljanko

January 22, 2004
Model Checking $CTL$

There is a straightforward model checker for $CTL$, whose running time is linear in both the size of the Kripke structure and the size of the formula.

We will now present a $CTL$ model checking algorithm due to Emerson, Clarke, and Sistla.
In our presentation we use $AU$ and $EU$ as single subformulas. Also, we will only present $AX$ and $\wedge$, as $EX$ and $\vee$ can be obtained from the using negation.

For convenience we assume that all the subformulas are numbered in an order, where for all subformulas $f$ it holds that $\text{left}(f)$ and $\text{right}(f)$ have smaller numbers than $f$.

The algorithm uses one bit-array of size $|S|$ called “label” for each subformula to store the truth values of different subformulas. This array for the top-level formula will also store the final model checking result.

One bit-array of size $|S|$ called “marked” is also used for internal bookkeeping.

This algorithm uses both successor and predecessor lists of a state in the Kripke structure.

For an on-the-fly implementation another (more complex) algorithm is needed.
Algorithm 1  Main loop of a CTL model checker.

procedure global_model_checker(f)
  for i := 1 to length(f) do
    foreach s ∈ S do
      reset_label(s, i); // Init formula $f_i$ to false
    od
    label_graph(i); // Evaluate the formula $f_i$ in all states
  od
end procedure
Algorithm 2  Choose processing subroutine based on formula type

procedure label_graph(i)
    ftype := formula_type($f_i$);
    if ftype = atomic proposition then
        atomic(i);
    elsif ftype = NOT then
        negation(i);
    elsif ftype = AND then
        conjunction(i);
    elsif ftype = AX then
        ax(i);
    elsif ftype = AU then
        au(i);
    elsif ftype = EU then
        eu(i);
    endif
end procedure
Algorithm 3  Process atomic proposition

procedure atomic(i)
    foreach $s \in S$ do
        if evaluate_proposition($s, i$) then
            add_label($s, i$);
        endif
    od
end procedure
Algorithm 4  Process negation

procedure negation(i)
    foreach s ∈ S do
        if ¬labeled(s, left(i)) then
            add_label(s, i);
        endif
    od
end procedure
Algorithm 5  Process conjunction

procedure conjunction(i)
    foreach s ∈ S do
        if labeled(s, left(i)) ∧ labeled(s, right(i)) then
            add_label(s, i);
        endif
    od
end procedure
Algorithm 6  *Process universal next-state formula*

procedure ax(i)
  foreach s ∈ S do
    add_label(s, i);
    foreach t ∈ successors(s) do
      if ¬labeled(t, left(i)) then
        reset_label(s, i);
        break;
      endif
    od
  od
end procedure
Algorithm 7  Process universal until formula

procedure au(i)
    foreach s ∈ S do
        reset_marked(s);
    od
    foreach s ∈ S do
        if ¬marked(s) then
            check_au(i, s);
        endif
    od
end procedure
Algorithm 8  Process universal until formula for state $s$

procedure check_au(i, s)
   if marked(s) then
      if labeled(s, i) then
         return true;
      else
         return false;
      endif
   endif
endif
set_marked(s);
if labeled(s, right(i)) then
    add_label(s, i);
    return true;
elsif ¬labeled(s, left(i)) then
    return false;
endif
foreach t ∈ successors(s) do
    if ¬check_au(i, t) then
        return false;
    endif
od
add_label(s, i);
return true;
end procedure
Algorithm 9  Process existential until formula
procedure eu(i)
    foreach $s \in S$ do
        reset_marked(s);
    od
    foreach $s \in S$ do
        if $\neg$marked(s) then
            check_eu(i, s);
        endif
    od
end procedure
Algorithm 10  Process existential until formula for state $s$

procedure check_eu(i, s)
    if labeled(s, right(i)) then
        add_label(s, i);
        label_predecessors(i, s);
    endif
end procedure
Algorithm 11  *Propagate label change to predecessor states*

**procedure** label_predecessors(i, s)
  
  `set_marked(s);`

  `foreach t ∈ predecessors(s) do // Note the use of predecessor relation!`

  `if ¬marked(t) ∧ labeled(t, left(i)) then`

  `add_label(t, i);`

  `label_predecessors(i, t);`

  `endif`

  `od`

**end procedure**
Model checking $CTL^*$

Model checking $CTL^*$ is quite straightforward once we have a global model checker for $LTL$. (An algorithm which evaluates an $LTL$ formula in all states of the system.)

Assume we have an $LTL$ model checker, which (in $CTL^*$ notation) returns the set of states $\{ s \in S \mid M, s \modelsAf f_1 \}$, where $f_1$ is an $LTL$ formula. (We can implement this procedure by e.g., calling a standard $LTL$ model checker $|S|$ times for the formula $f$ setting the initial state of the Kripke structure to each state of the Kripke structure at a time.)

We call this algorithm “$\text{GlobalCheckLTL}()$”.

We will now show that model checking $CTL^*$ can be made with an algorithm of essentially the same complexity as the complexity of “$\text{GlobalCheckLTL}()$” by using the following procedure.
The recursive evaluation procedure “GlobalCheckCTL*(f)” goes as follows:

1. Convert the CTL* formula f into negation normal form.
   (Push negations in as far as possible with DeMorgan rules).

2. If f is of the form Af1, where f1 is an LTL formula, return GlobalCheckLTL(f1).

3. If f is of the form Ef1, return (S \ GlobalCheckCTL*(A¬f1)).
4. Let $g_1, g_2, \ldots, g_n$ be the maximal subformulas of $f$, which are not $LTL$ formulas. For each $g_i$, create a new atomic proposition $h_i$, and calculate the valuation of it by calling $GlobalCheckCTL^*(g_i)$. Furthermore, replace each subformula $g_i$ in the formula $f$ by the corresponding atomic proposition $h_i$.

5. return $GlobalCheckLTL(f)$.
   (After the step 4. above, $f$ is guaranteed to be an $LTL$ formula.)

Now $M, s^0 \models f$ iff $s^0 \in GlobalCheckCTL^*(f)$. 
Note that the procedure above only does some minor bookkeeping, and that the most expensive part are the calls to the subroutine $GlobalCheckLTL(f)$. However, this subroutine is called at most $|f|$ times. Thus the $GlobalCheckCTL^*(f)$ is (roughly) at most $|f|$ times slower than $GlobalCheckLTL(f)$.

To implement the global $LTL$ model checking procedure “$GlobalCheckLTL(f_1)$”, one can for example call a standard (local) $LTL$ model checking procedure $|S|$ times, each time with a new initial state, will calculate the required set of states.

However, doing so will not be very efficient, as a lot of redundant is done across the different calls. (The global model checking algorithm is now quadratic in the number of states in the Kripke structure, instead of being linear.)

By using an MSCC based emptiness checking algorithm (e.g., modified Tarjan’s MSCC algorithm) this quadratic overhead can be brought back to a linear one.
Note, however, that using $CTL^*$ instead of $LTL$ has also disadvantages. For example, so-called “partial order reduction methods” for $CTL^*$ are less effective than for $LTL$. Also, the use of abstraction methods is more cumbersome for $CTL^*$.

Thus even though model checking as such is not harder for $CTL^*$, practical model checking use might still prefer $LTL$ over it. This is one of the reasons why many model checking tools do not support $CTL^*$. 
Bisimulation

Bisimulation is the equivalence which is characterized by the logic $CTL^*$. 

**Definition 1** Let $M = (S, s^0, R, L)$ and $M' = (S', s'^0, R', L')$ be two Kripke structures with the same set of atomic propositions $AP$.

A relation $B \subseteq S \times S'$ is a bisimulation relation iff for all $s \in S, s' \in S'$, if $B(s, s')$ then the following conditions hold:

- $L(s) = L'(s')$
- For every $s_1 \in S$ such that $R(s, s_1)$ there is $s'_1 \in S'$ such that $R'(s', s'_1)$ and $B(s_1, s'_1)$
- For every $s'_1 \in S'$ such that $R(s', s'_1)$ there is $s_1 \in S$ such that $R(s, s_1)$ and $B(s_1, s'_1)$

Two Kripke structures $M$ and $M'$ are bisimulation equivalent (denoted $M \equiv M'$) iff there exists a bisimulation $B$, such that $B(s^0, s'^0)$. 
The following theorem states that bisimulation preserves $CTL^*$:

**Theorem 2**  If $M \equiv M'$ then for every $CTL^*$ formula $f$, it holds that $M \models f$ iff $M' \models f$. 
Thus, if one can prove that some manipulation of a model results in a (hopefully much smaller) Kripke structure $M'$, which is still bisimulation equivalent to the original Kripke structure $M$, we can do the following.

Model check any $CTL^*$ formula $f$ on $M'$ instead of $M$. (To save memory and space.)

Unfortunately, most of the manipulations on model level do not preserve bisimulation, as bisimulation requires that the (among other things) branching structure of the program is preserved.

Also, in most cases the algorithms for minimizing a Kripke structure w.r.t. bisimulation are slower than model checking algorithms.
Bisimulation is the best we can do, as given by the following theorem.

**Theorem 3** If $M \not\equiv M'$ then there exist a $CTL$ formula $f$, such that $M \models f$ and $M' \not\models f$.

Thus, in fact, by preserving $CTL$ one also preserves all of $CTL^*$.

If one considers the logic $CTL^*-X$ ($CTL^*$ with the next-time operator removed), there also exists an equivalence characterizing it, usually called *stuttering bisimulation*. This equivalence is used by partial order reductions preserving all of $CTL^*-X$. 
Simulation

For $\mathit{LTL}$ a more useful notion to be used is that of a simulation.

**Definition 4** Let $M = (S, s^0, R, L)$ and $M' = (S', s^0', R', L')$ be two Kripke structures with the same set of atomic propositions $\mathit{AP}$.

A relation $H \subseteq S \times S'$ is a simulation relation iff for all $s \in S, s' \in S'$, if $H(s, s')$ ($s'$ simulates $s$) then the following conditions hold:

- $L(s) = L'(s')$

- For every $s_1 \in S$ such that $R(s, s_1)$ there is $s'_1 \in S'$ such that $R'(s', s'_1)$ and $H(s_1, s'_1)$

We say that $M'$ simulates $M$ (denoted by $M \preceq M'$), if there exists a simulation relation $H$, such that $H(s^0, s^0')$. 
The logic $ACTL^*$ consist of those formulas of $CTL^*$, which in negation normal form (i.e., with negations pushed in as far as possible with DeMorgan rules) do not contain the existential $E$ path quantifier. Most notably all $LTL$ formulas are also $ACTL^*$ formulas (once you add the implicit $A$ in front of an $LTL$ formula).

We have the following result.

**Theorem 5** Suppose $M \preceq M'$. Then for every $ACTL^*$ formula $f$ it holds that $M' \models f$ implies $M \models f$.

Thus, if one can prove that $M \preceq M'$ then all $ACTL^*$ formulas can be checked on the reduced structure $M'$. In particular, if one can prove that an $LTL$ property $f$ holds in $M'$, then it also holds in $M$. 