The Algorithmic Verification of Reactive Systems

Lecture 6:

Algorithms for Checking Temporal Logic Properties (Branching Time)

The Logic CTL

Recall that CTL is the branching-time temporal logic whose formulas are defined as follows.

A CTL state formula is either:

- $\mathbf{true}$, $\mathbf{false}$, p, or $\neg p$, for all $p \in P$;
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are CTL state formulas;
- $A\psi$ or $E\psi$, where ψ is a CTL path formula.

A CTL path formula is of the form

- $\bigcirc \varphi_1$, $\varphi_1 \, U \, \varphi_2$, or $\varphi_1 \, \tilde{U} \, \varphi_2$, where φ_1 and φ_2 are CTL state formulas.

Examples

$A \, \Box \, p$, $E \, \Box \, p$, $A \, \Box \, E \bigcirc p$, $E \, \Box \, E \, \Diamond \, p$,
$A(E(\Box \, p) \, \tilde{U} \, A(q \, U \, p))$.

Branching-Time Model Checking
The Problem

- One is given a finite transition system obtained from a concurrent program, i.e a finite Kripke structures $K = \langle P, W, R, w^0, L\rangle$, where
  - P is a set of atomic propositions,
  - W is a set of states,
  - $R \subseteq W \times W$ is a transition relation that must be total (i.e., for every $w \in W$ there exists $w' \in W$ such that $\langle w, w'\rangle \in R$),
  - w^0 is an initial state, and
  - $L : W \to 2^P$ maps each state to the set of atomic propositions true in that state.
- One is also given a CTL formula φ.
- The model-checking problem is to check that K satisfies (is a model of) φ.
- Note that in contrast to the linear-time case where there was an implicit "for all behaviors" quantification in the definition of the model-checking problem, there is no implicit quantification in the branching-time model-checking problem.

A First Approach

The problem can be solved by checking if the initial state w^0 of the Kripke structure can be labeled by φ while respecting the semantic rules of the logic. Formally, one checks whether there is a labeling $\tau : W \rightarrow cl(\varphi)$ such that the following rules are satisfied.

- $\varphi \in \tau(w^0)$
- For $p \in P$, if $p \in \tau(w)$ then $p \in L(w)$, and if $\neg p \in \tau(w)$ then $p \notin L(w)$
- if $\varphi_1 \wedge \varphi_2 \in \tau(w)$ then $\varphi_1 \in \tau(w)$ and $\varphi_2 \in \tau(w)$.
- if $\varphi_1 \vee \varphi_2 \in \tau(w)$ then $\varphi_1 \in \tau(w)$ or $\varphi_2 \in \tau(w)$.

- if $A \bigcirc \varphi_1 \in \tau(w)$ then for all w' such that $(w, w') \in R$, $\varphi_1 \in \tau(w')$.

- if $E \bigcirc \varphi_1 \in \tau(w)$ then for some w' such that $(w, w') \in R$, $\varphi_1 \in \tau(w')$.

- if $A(\varphi_1 \, U \, \varphi_2) \in \tau(w)$ then either $\varphi_2 \in \tau(w)$, or $\varphi_1 \in \tau(w)$ and for all w' such that $(w, w') \in R$ $A(\varphi_1 \, U \, \varphi_2) \in \tau(w')$.

- if $E(\varphi_1 \, U \, \varphi_2) \in \tau(w)$ then either $\varphi_2 \in \tau(w)$, or $\varphi_1 \in \tau(w)$ and for some w' such that $(w, w') \in R$ $E(\varphi_1 \, U \, \varphi_2) \in \tau(w')$.

- if $A(\varphi_1 \, \tilde{U} \, \varphi_2) \in \tau(w)$ then $\varphi_2 \in \tau(w)$, and either $\varphi_1 \in \tau(w)$ or for all w' such that $(w, w') \in R$ $A(\varphi_1 \, \tilde{U} \, \varphi_2) \in \tau(w')$

- if $E(\varphi_1 \, \tilde{U} \, \varphi_2) \in \tau(w)$ then $\varphi_2 \in \tau(w)$, and either $\varphi_1 \in \tau(w)$ or for some w' such that $(w, w') \in R$ $E(\varphi_1 \, \tilde{U} \, \varphi_2) \in \tau(w')$

Furthermore, to these rules we need to add the fact that the eventualities are satisfied.

Turning the Approach into an Algorithm

To turn the labeling rules above into an algorithm, we proceeed by labeling the structure with elements of the closure in an order compatible with the "subformula" order, starting with the innermost subformulas (i.e., the atomic or negated atomic propositions).

The general structure of the algorithm is thus the following.

1. Pick an formula $\varphi_i \in cl(\varphi)$ all subformulas of which have already been processed (initially this can only be an atomic or negated atomic proposition).
2. For each state $w \in W$, label w by φ_i if the labeling rules allow it (note that we label every time it is allowed, i.e. we label maximally).
3. Repeat until φ is processed.

We still need to describe how to do step 2.

- For formulas that are of the form $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $A \bigcirc \varphi_1$, or $E \bigcirc \varphi_1$, it is perfectly straightforward: one only needs to look at the current labeling.

- For formulas of the form $A(\varphi_1 \, U \, \varphi_2)$, $E(\varphi_1 \, U \, \varphi_2)$, $A(\varphi_1 \, \tilde{U} \, \varphi_2)$, or $E(\varphi_1 \, \tilde{U} \, \varphi_2)$, more needs to be done.

Coping with U and $\tilde{U}$ Formulas

For U and $\tilde{U}$ formulas, the labeling rules are recursive. Furthermore, for U formulas we have to guarantee that their second argument is indeed eventually satisfied.

Consider first the case of $A(\varphi_1 \, U \, \varphi_2)$ formulas. We use the following observations.

- A node w that is already labeled by φ_2 can be unconditionally labeled by $A(\varphi_1 \, U \, \varphi_2)$.
- If for all w' such that $(w, w') \in R$, w' is unconditionally labeled by $A(\varphi_1 \, U \, \varphi_2)$ and if w is labeled by φ_1, then w can be unconditionally labeled by $A(\varphi_1 \, U \, \varphi_2)$.

These observations can be directly translated into the following algorithm.

1. Label all nodes already labeled by φ_2 by $A(\varphi_1 \, U \, \varphi_2)$.

2. For each node $w \in W$, if all w' such that $(w, w') \in R$ are already labeled by $A(\varphi_1 \, U \, \varphi_2)$ and if w is labeled by φ_1, then label w by $A(\varphi_1 \, U \, \varphi_2)$.

3. Repeat step 2 until no more nodes need to be labeled.

For formulas of the form $A(\varphi_1 \tilde{U} \varphi_2)$, the situation is dual. Precisely, we start with an optimistic labeling (all states in which φ_2 appears) and then remove nodes that have been labeled unsoundly.

The rules are the following.

- A node w that is already labeled by φ_2 can be provisionally labeled by $A(\varphi_1 \tilde{U} \varphi_2)$.
- if a node w is not labeled φ_1 and some node w' such that $(w, w') \in R$ is not provisionally labeled by $A(\varphi_1 \tilde{U} \varphi_2)$, then remove $A(\varphi_1 \tilde{U} \varphi_2)$ from the label of w.

The algorithm is then the following.

1. Label all nodes already labeled by φ_2 by $A(\varphi_1 \tilde{U} \varphi_2)$.

2. For each node $w \in W$, if w is not labeled by φ_1 and if some w' such that $(w, w') \in R$ is not labeled by $A(\varphi_1 \tilde{U} \varphi_2)$, then remove $A(\varphi_1 \tilde{U} \varphi_2)$ from the label of w.

3. Repeat step 2 until no more nodes need to be unlabeled.

A very similar procedure can be used for formulas of the form $E(\varphi_1 \tilde{U} \varphi_2)$.

Labeling Rules and Fixpoints

The labeling algorithms we have just given for U and $\tilde{U}$ can be seen as the computation of *fixpoints.*

- Consider the set of subsets of W (2^W) ordered with the subset relation.
- Consider then a function $\mathcal{T} : 2^W \rightarrow 2^W$. A *fixpoint* of $\mathcal{T}$ is a set X such that $X = \mathcal{T}(X)$.
- The *least fixpoint* $\mu X.\mathcal{T}(X)$ of $\mathcal{T}$ is the smallest set such that $X = \mathcal{T}(X)$.
- The *greatest fixpoint* $\nu X.\mathcal{T}(X)$ of $\mathcal{T}$ is the largest set such that $X = \mathcal{T}(X)$.

Theorem. If $\mathcal{T}$ is monotonic, i.e. if when $X_1 \leq X_2$ then $\mathcal{T}(X_1) \leq \mathcal{T}(X_2)$, then least and greatest fixpoints exist. Furthermore, under continuity conditions (satisfied when W is finite), the fixpoints can be computed as follows (these computations terminate since W is finite).

$$\begin{array}{rcl} \mu X \mathcal{T}.(X) & = & \bigcup_i \mathcal{T}^i(\emptyset) \\ \nu X \mathcal{T}.(X) & = & \bigcap_i \mathcal{T}^i(W) \end{array}$$

Using these notions, we can write:

$$\begin{array}{rcl} A(\varphi_1 \, U \, \varphi_2) & = & \mu X(\varphi_2 \vee (\varphi_1 \wedge A \bigcirc X)) \\ A(\varphi_1 \, \tilde{U} \, \varphi_2) & = & \nu X(\varphi_2 \wedge (\varphi_1 \vee A \bigcirc X)) \end{array}$$

The algorithms we have given perform a computation corresponding exactly to the evaluation of these fixpoints.

Complexity Issues

The algorithms given for labeling with U and $\tilde{U}$ formulas have a complexity that is $O(|W|^2)$. This can be reduced to $O(|W|)$ by avoiding duplicate work.

Note that to achieve this complexity, we have to mark in one pass over the structure *all* nodes in which the formula is true.

For AU formulas, one uses a depth-first search and marks processed states (using a table *marked*).

The procedure then applies the recursive marking procedure *aulabel* to all nodes that have not yet been processed while handling a previous node:

for all $w \in W$
    begin
    if $\neg$marked(w) **then** aulabel(w, f)
    end

The two arguments of the procedure *aulabel* are a node w and a formula f of the form $A(\varphi_1 \, U \, \varphi_2)$.

procedure aulabel(w, f): **boolean**
begin
if marked(w) **then**
if $f \in \tau(w)$ **then** **return**(T)
else **return**(F);
marked(w) $:= T$;
if $\varphi_2 \in \tau(w)$ **then** **begin**
$\tau(w) := \tau(w) \cup \{f\}$;
return(T)
end;
if $\varphi_1 \notin \tau(w)$ **then** **return**(F);
for all $w'.(w, w') \in R$
begin
if $\neg$aulabel(w', f) **then** **return**(F)
end;
$\tau(w) := \tau(w) \cup \{f\}$;
return(T)
end

For $E\tilde{U}$ formulas, the algorithm is similar and is given below

```
for all w ∈ W
    begin
    if ¬marked(w) then eũlabel(w, f)
    end


procedure eũlabel(w, f): boolean
  begin
    if marked(w) then
                  if f ∈ τ(w) then return(T)
                               else return(F);
    marked(w) := T;
    τ(w) := τ(w) ∪ {f};
    if φ2 ∉ τ(w) then begin
                      τ(w) := τ(w) \ {f};
                      return(F)
                      end;
    if φ1 ∈ τ(w) then return(T);
    for all w'.(w, w') ∈ R
        begin
        if eũlabel(w', f) then return(T)
        end;
        τ(w) := τ(w) \ {f};
        return(F)
  end
```

Note that when these algorithms find a cycle that goes back to a node whose processing has started but is not finished, this node is assumed to be unlabeled by f in the case of an AU formula, and is assumed to be labeled by f in the case of an $E\tilde{U}$ formulas.

Indeed, for an AU formula, a cycle in which φ_2 does not occur has been found and the formula is false. For an $E\tilde{U}$ formula, a cycle in which φ_2 is always true has been found and the formula is thus true.

A similar approach cannot work for EU (or $A\tilde{U}$ formulas). Indeed, finding a cycle on which φ_2 is not true tells us nothing about and EU formula since only the existence of a satisfying path is required. The same applies dually to $A\tilde{U}$ formulas.

A different approach is thus needed for EU and $A\tilde{U}$ formulas. It consists of proceeding backwards through the graph from nodes whose marking is immediate.

For EU formulas, the algorithm is as below.

for all $w \in W$
    begin
    if $\varphi_2 \in \tau(w) \wedge f \notin \tau(w)$ **then** eulabel(w, f)
    end

procedure eulabel(w, f)
   begin
      $\tau(w) := \tau(w) \cup \{f\}$;
      for all $w'.(w', w) \in R$
          begin
          if $\varphi_1 \in \tau(w') \wedge f \notin \tau(w')$ **then**
                                        eulabel(w', f)
          end
      end

For $A\tilde{U}$ formulas, the algorithm is similar, except that we unlabel nodes.

```
for all w ∈ W
    begin
    τ(w) := τ(w) ∪ {f}
    end;

for all w ∈ W
    begin
    if φ2 ∉ τ(w) ∧ f ∈ τ(w) then aũlabel(w, f)
    end


procedure aũlabel(w, f)
  begin
     τ(w) := τ(w) \ {f};
     for all w'.(w', w) ∈ R
         begin
         if φ1 ∉ τ(w') ∧ f ∈ τ(w') then
                                     aũlabel(w', f)
         end
     end
```

Complexity Results

Since the model-checking algorithm uses time $O(|W|)$ for each element of the closure of the formula, its overall complexity is

$$O(|W| \times |\varphi|).$$

To summarize the situation is the following.

Complexity of LTL Model Checking:
Linear in the size of the transition system (NLOGSPACE),
Exponential in the size of the FCS (PSPACE),
Linear in the size of the CTL property.

Note that the only difference with respect to LTL model checking is the complexity in the size of the formula, which usually is not the crucial factor.

Model Checking for CTL: The Automata Theoretic Approach

- To check that a Kripke structure $K = \langle P, W, R, w^0, L\rangle$ satisfies a CTL formula φ, first construct the alternating automaton $\mathcal{A}_{\mathcal{D},\varphi}$ where $\mathcal{D}$ is the set of degrees occurring in K.
- The problem is then to determine if the infinite tree generated by K is accepted by $\mathcal{A}_{\mathcal{D},\varphi}$.
- To solve this, one computes the product $K \times \mathcal{A}_{\mathcal{D},\varphi}$, which represents the possible computations of $\mathcal{A}_{\mathcal{D},\varphi}$ on K.
- One then checks whether an accepting computation can be extracted from $K \times \mathcal{A}_{\mathcal{D},\varphi}$.
- Note that $K \times \mathcal{A}_{\mathcal{D},\varphi}$ is an automaton without input (the input is given by K). It can be viewed as a Boolean graph and checking whether it embodies an accepting computation amounts to a rather simple algorithmic problem on Boolean graphs.

Computing the Product Automaton

- Given
  - Let $\mathcal{A}_{\mathcal{D},\varphi} = \langle 2^P, \mathcal{D}, S_\varphi, \delta_\varphi, S_0, F_\varphi \rangle$ be the alternating automaton obtained from φ for the set of degrees $\mathcal{D}$ and
  - $\langle W, R, w^0, L \rangle$ a Kripke structure with degrees in $\mathcal{D}$,
- the product automaton of $\mathcal{A}_{\mathcal{D},\varphi}$ and K is $\mathcal{A}_{K,\varphi} =$ $\langle \{a\}, \mathcal{D}, W \times S_\varphi, \delta, \langle w^0, S_0 \rangle, F_\varphi \times W \rangle$ with,
  - if $\delta_\varphi(s, L(w), d(w)) = \theta$, . then $\delta(\langle w, s \rangle, a, d(w)) = \theta'$, where $\theta' = \theta[(c, s') \leftarrow (c, \langle w_c, s' \rangle)]$

Note that the input alphabet in the product is just the dummy single letter alphabet $\{a\}$.

Similarly, in θ' the directions c are no longer necessary.

Weak Alternating Automata

For CTL formulas, the automaton $K \times \mathcal{A}_{\mathcal{D},\varphi}$ is weak:

- The set of sets Q can be partitioned into disjoint subsets such that
  - either $S_i \subseteq F$ or $S_i \cap F = \emptyset$,
  - there is an order on the S_i such that for $q \in S_i$, if s' appears in $\delta(q, \sigma, k)$ then $s' \in S_j$ such that $S_j \leq S_i$.
- This partition is the one that gathers in the same component all states (w, s) in which where the second component s is the same subformula of φ.
- The order between the components is then any completion of the subformula order.

Checking Acceptance of an Inputless Weak Alternation Automata

It is easy to check whether an Inputless weak alternating automaton has an accepting computation (the problem of ensuring that conjunctively related paths in the run tree have the same labeling disappears).

- Label the bottom S_i with $\mathbf{true}$ or $\mathbf{false}$ depending on whether it is accepting or not.
- Propagate the labeling up through the transition function simplifying on the way.
- When no further simplification of the labeling of states in a component S_i is possible, the remaining states are labeled according to whether the component is accepting or not.
- The automaton is nonempty iff $\langle w^0, \varphi \rangle$ is labeled $\mathbf{true}$.

If carefully implemented, this procedure can be performed in linear time.

On-the-Fly (Local) Model Checking for CTL

- The model-checking algorithms we have seen (automata-based or not) assume that the Kripke structure has been entirely constructed before the algorithm is applied.
- In practice this is not very efficient since
  - The Kripke structure might not be entirely needed and
  - Its transition relation and its inverse transition relation are needed and thus need to be stored.
- The Alternative is to explore the product of the Kripke structure and the alternating automaton starting in the initial state and building it as needed.
- This can be done with the following algorithm.

- The algorithm is a search through the states of the product $\mathcal{A}_{K,\varphi}$ that starts in the initial state (w^0, φ).

- The search aims at filling in a table *label* that indicates for each state whether it evaluates to $\mathbf{true}$ or $\mathbf{false}$, or whether it is yet unlabeled ($\perp$, the initial value).

- The algorithm distinguishes the case in which it is jumping from one component of the Weak Alternating Automaton to another from the case in which it is staying within the same component (checking the same formula).

- When operating within a component, it only labels the node through which it has entered this component. This avoids the type of problem mentioned above with respect to checking EU or $A\tilde{U}$ formulas forwards. The consequence of this is a non-optimal complexity.

- The algorithm uses the recursive procedure explore(w, s, w_o). The last argument is used to determine whether the call moves within a component or not. The initial call is to explore(w^0, φ, w^0).

- In this procedure, the function

  - The array *marked* is used to note which states have been visited during a specific visit to a given component.
  - *accepting* indicates whether a state is accepting or not;
  - the function *simplify* simplifies a Boolean expression given values for its atoms.

- **Challenge problem** The given algorithm has suboptimal complexity. Modify it so that in runs in time $O(|K| \times |cl(\varphi)|)$.

```
procedure explore(w, s, w_o): boolean
  begin
     if label(w, s) ≠ ⊥ then return
                              (label(w, s));
     if marked(w, s) = w_o then return
                                 (accepting(w, s));
     marked(w) := w_o;
     if no (w', s') occurs in δ(w, s) then
                    begin
                       label(w, s) := simplify(δ(w, s));
                       return(label(w, s))
                    end;
     for all (w', s') occurring in δ(w, s)
        begin
        if s' = s then
              rval(w', s') := explore(w', s', w_o)
                   else
              rval(w', s') := explore(w', s', w');
        end
        /* use returned values rval(w', s')
           to simplify */
        if w = w_o then
                  label(w, s) := simplify(δ(w, s));
        return(simplify(δ(w, s))
        end
```