Hoare logic and Model checking

Part II: Model checking

Lecture 10: Implementing model checking

Christopher Pulte cp526
University of Cambridge

CST Part II – 2022/23

In the last two lectures we saw LTL and CTL as examples of temporal logics that can specify the behaviour of temporal models. For using temporal logics in the verification of artefacts, we need model checkers to check a temporal model against a temporal logic specification.

In this lecture we will implement a naïve model checker for CTL: the world's worst model checker for CTL.

Model checking

What is model checking?

The model checking problem for CTL is to determine, for a given a temporal model M over some set of atomic propositions AP and CTL formula ψ over AP, whether M satisfies ψ:

$$M \models \psi$$

We need a function that computes this.

Definite temporal models

The temporal models we have defined in lecture 7 were not restricted in any way that guarantees computability. Here we will assume a **definite temporal model**, using a finite set of states and computable functions for the initial state predicate, the transition relation and the labelling of states.

Type of temporal models

```
type state = int

module States = Set.Make(Int)

type 'ap tmodel = {
  s : States.t;                 (* finite set *)
  s0 : state -> bool;           (* computable *)
  t : state -> state -> bool;   (* computable *)
  l : state -> 'ap -> bool;     (* computable *)
}
```

Specifying a CTL model checker

We will implement a naïve CTL model checker:

```
val mc : 'ap tmodel -> 'ap state_prop -> bool
```

which has the following specification:

$$\forall M, \psi.\ (\mathsf{mc}\ M\ \psi \Leftrightarrow M \vDash \psi)$$

Defining a CTL model checker. Updated

To check whether the model satisfies a property ψ, we have to check whether the initial states satisfy ψ. We check this using an auxiliary function *mca* that returns the states satisfying a given state property.

```
let mc (m : 'ap tmodel) (psi : 'ap state_prop) : bool =
  assert (left_total m);
  let v = mca m psi in
  States.for_all (fun s ->
      not (m.s0 s) || States.mem s v
    ) m.s
```

This *mca* function works by recursion on the formula, calling itself on the sub-formulas.

CTL model checker

(This is often phrased in terms of "labelling" of states.)

Strategy: For a given CTL state-property ψ: compute the states of the temporal model that satisfies ψ, by

- exploiting CTL formula equivalences to encode ψ as a formula $\hat{\psi}$ that uses only existential path quantification (using negation in the right places)
- (recursively) computing the states satisfying the **sub-formulas** of $\hat{\psi}$, and
- using this information to determine which states should be returned for $\hat{\psi}$.

CTL model checker: propositional fragment

mca, for a given temporal model and state property returns the set of states satisfying the state property.

```
let rec mca (m : 'ap tmodel) (psi : 'ap state_prop)
    : States.t =
  match psi with
  | True ->
     m.s
  | False ->
     States.empty
  | AP p ->
     States.filter (fun s -> m.l s p) m.s
  | Not psi' ->
     let v = mca m psi' in
     States.diff m.s v
  ...
```

CTL model checker: propositional fragment (continued)

```
let rec mca (m : 'ap tmodel) (psi : 'ap state_prop)
    : States.t =
  ...
  | And (psi1, psi2) ->
     let v1 = mca m psi1 in
     let v2 = mca m psi2 in
     States.inter v1 v2
  | Or (psi1, psi2) ->
     let v1 = mca m psi1 in
     let v2 = mca m psi2 in
     States.union v1 v2
  | Impl (psi1, psi2) ->
     mca m (Or (Not psi1, psi2))
  ...
```

CTL model checker: A

We use

- A X $\psi' = \neg$E X $(\neg\psi')$
- A G $\psi' = \neg$E F $(\neg\psi')$

```
let rec mca (m : 'ap tmodel) (psi : 'ap state_prop)
    : States.t =
  ...
  | A (X psi') ->
     mca m (Not (E (X (Not psi'))))
  | A (G psi') ->
     mca m (Not (E (F (Not psi'))))
  | A (F _) ->
     failwith "TODO: exercise"
  | A (U (psi1, psi2)) ->
     failwith "TODO: tricky exercise"
  ...
```

CTL model checker: EX

If we know in which states ψ' holds, then we know in which states X ψ' holds: their predecessors:

```
let rec mca (m : 'ap tmodel) (psi : 'ap state_prop)
    : States.t =
  ...
  | E (X psi') ->
     let v = mca m psi' in
     States.filter (fun s ->
         States.exists (fun s' ->
             m.t s s'
           ) v
       ) m.s
  ...
```

CTL model checker: EF

We use E F $\psi' =$ E $(\top$ U $\psi')$

```
let rec mca (m : 'ap tmodel) (psi : 'ap state_prop)
    : States.t =
  ...
  | E (F psi') ->
     mca m (E (U (True, psi')))
  ...
```

CTL model checker: EG and EU

Left to do are E G ψ' and E (ψ_1 U ψ_2), which talk about infinite paths. We will implement those using fixpoint operations on sets, where the finite size of the set of states guarantees termination.

CTL model checker: EG

For E G ψ':

1. compute the set v of states satisfying ψ'
2. define the output set to be $v' := v$
3. until there are no more changes: remove from v' elements that cannot transition into v'

```
| E (G psi') ->
   let v = mca m psi' in
   fixpoint (fun v' ->
       States.filter (fun s ->
           States.exists (fun s' ->
               m.t s s'
             ) v'
         ) v'
     ) v
 ...
```

CTL model checker: EU

For E (ψ_1 U ψ_2):

1. compute the sets v_1 and v_2 of states satisfying ψ_1 and ψ_2
2. define the output set to be $v' := v_2$
3. until there are no more changes: add states from v_1 that can transition into v'

```
 | E (U (psi1, psi2)) ->
   let v1 = mca m psi1 in
   let v2 = mca m psi2 in
   fixpoint (fun v' ->
       States.union v'
         (States.filter (fun s ->
              States.exists (fun s' ->
                  m.t s s'
                ) v'
            ) v1)
     ) v2
```

Actually implementing model checking. Updated

This is not very efficient!

In practice,

- the labelling (the vs) are memoised: in our code the vs are re-computed each time, in the case of nested CTL formulas
- "symbolic model checking" uses binary decision diagrams (IB Logic and proof) to represent sets of states, and performs operations on sets-as-BDDs, instead of explicitly manipulating the sets;
- the states can be computed lazily;
- "partial order reduction" tries to not enumerate redundant interleavings;
- ...
- 40+ years of tricks!

Counterexamples

Generating counterexamples

Adapted from “Tree-Like Counterexamples in Model Checking”.

If the specification is not satisfied, and is in ACTL, then we can do better than just say “no”: we can produce a counterexample.

The idea is that $M \not\models \psi^{\text{ACTL}}$ is equivalent to $M \models \neg\psi^{\text{ACTL}}$, where $\neg\psi^{\text{ACTL}}$ can be expressed in ECTL.

So $M \not\models \psi^{\text{ACTL}}$ implies the existence of a witness for the corresponding ECTL property.

We will now assume formulas in negation normal form: formulas without implication, and where the only use of negation is immediately preceding an atomic proposition.

Shape of ECTL witnesses

The shape of an ECTL witness for a set of atomic propositions AP and temporal model M:

$$
\begin{array}{l}
\mathsf{Witness}_{M} := \\
\mid \mathsf{WAP} \in M.S \to \mathsf{Witness}_{M} \\
\mid \mathsf{WNAP} \in M.S \to \mathsf{Witness}_{M} \\
\mid \mathsf{WAnd} \in \mathsf{Witness}_{M} \to \mathsf{Witness}_{M} \to \mathsf{Witness}_{M} \\
\mid \mathsf{WOrL} \in \mathsf{Witness}_{M} \to \mathsf{Witness}_{M} \\
\mid \mathsf{WOrR} \in \mathsf{Witness}_{M} \to \mathsf{Witness}_{M} \\
\mid \mathsf{WX} \in M.S \to M.S \to \mathsf{Witness}_{M} \to \mathsf{Witness}_{M} \\
\mid \mathsf{WF} \in \mathsf{list}\ M.S \to \mathsf{Witness}_{M} \to \mathsf{Witness}_{M} \\
\mid \mathsf{WG} \in \mathsf{list}\ (M.S \times \mathsf{Witness}_{M}) \to \mathsf{Witness}_{M} \\
\mid \mathsf{WU} \in \mathsf{list}\ (M.S \times \mathsf{Witness}_{M}) \to M.S \to \mathsf{Witness}_{M} \to \mathsf{Witness}_{M}
\end{array}
$$

There are (on purpose) no cases for A . . ..

Being an ECTL witness

We will define when a witness is a “valid witness” for an ECTL property:

$$(s \vDash_{M} \psi) \text{ wit-by } W$$

should hold whenever W is a valid witness for the fact that ψ holds in state s of temporal model M.

Being an ECTL witness: atomic propositions

A witness for an atomic proposition is just the fact that the atomic proposition holds according to $M.\ell$:

$(s \vDash_{M} p)$ wit-by $W \overset{def}{=}$
$\quad W = \mathrm{WAP}\ s \wedge M.\ell\ s\ p$

Similarly for negation of atomic propositions.

Being an ECTL witness: next

A witness for ‘next’ is a transition from the current state to a next state, and a witness that the sub-property holds in the next state:

$$
\begin{aligned}
&(s \vDash_M (\mathsf{E}\ \mathsf{X}\ \psi))\ \text{wit-by}\ W \overset{def}{=} \\
&\quad \exists s' \in M.S, W' \in \mathsf{Witness}_M. \\
&\qquad \begin{pmatrix} W = \mathsf{WX}\ s\ s'\ W' \wedge \\ s\ M.T\ s' \wedge \\ (s' \vDash_M \psi)\ \text{wit-by}\ W' \end{pmatrix}
\end{aligned}
$$

Being an ECTL witness: future

A witness for the 'future' temporal operator is a finite path that leads to a state for which we have a witness that it satisfies the sub-property:

$$
\begin{aligned}
&(s \vDash_M \mathsf{E}\ \mathsf{F}\ \psi)\ \text{wit-by}\ W \overset{\mathit{def}}{=} \\
&\quad \exists s' \in M.S, \pi \in \text{list}\ M.S, W' \in \text{Witness}_M. \\
&\qquad \left(\begin{array}{l}
W = \text{WF}\ s\ \pi\ W' \wedge \\
\text{IsFinitePath}\ M\ \pi \wedge \\
\text{nth}\ \pi\ 0 = \text{some}\ s \wedge \\
\text{last}\ \pi = \text{some}\ s' \wedge \\
(s' \vDash_M \psi)\ \text{wit-by}\ W'
\end{array}\right)
\end{aligned}
$$

Being an ECTL witness: generally

A witness for the 'generally' temporal operator is a lasso-shaped path, together with witnesses that each state along the path satisfies the sub-property:

$$
\begin{array}{l}
(s \vDash_{M} \mathsf{E}\ \mathsf{G}\ \psi)\ \text{wit-by}\ W \overset{def}{=} \\
\quad \exists SWs \in \mathsf{list}\ (M{.}S \times \mathsf{Witness}_{M}). \\
\qquad \left(\begin{array}{l}
W = \mathsf{WG}\ SWs \wedge \\
\mathit{let}\ \pi = \mathit{firsts}\ SWs\ \mathit{in} \\
\mathsf{IsFinitePath}\ M\ \pi \wedge \\
\mathsf{nth}\ \pi\ 0 = s \\
(\exists i \in \mathbb{N}. (\mathsf{last}\ \pi)\ M{.}T\ (\mathsf{nth}\ \pi\ i)) \wedge \\
\left(\begin{array}{l}
\forall j \in \mathbb{N}, s' \in M{.}S, W' \in \mathsf{Witness}_{M}. \\
\quad \left(\begin{array}{l}
\mathsf{nth}\ SWs\ j = \mathsf{some}\ \langle s', W' \rangle \Rightarrow \\
\quad (s' \vDash_{M} \psi)\ \text{wit-by}\ W'
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array}
$$

Being an ECTL witness: until

$$
\begin{array}{l}
(s \models_M \mathsf{E}\ (\psi_1\ \mathsf{U}\ \psi_2))\ \text{wit-by}\ W \overset{def}{=} \\
\quad \exists SWs \in \text{list}\ (M.S \times \text{Witness}_M), s' \in M.S, W' \in \text{Witness}_M. \\
\qquad \left(\begin{array}{l}
W = \text{WU}\ SWs\ s'\ W' \land \\
\text{IsFinitePath}\ M\ (\text{firsts}\ SWs +\!\!+ [s']) \land \\
\text{nth}\ SWs\ 0 = (s, \_) \land \\
\left(\begin{array}{l}
\forall i \in \mathbb{N}, s'' \in M.S, W'' \in \text{Witness}_M. \\
\qquad \left(\begin{array}{l}
\text{nth}\ SWs\ i = \text{some}\ \langle s'', W'' \rangle \Rightarrow \\
\qquad (s'' \models_M \psi_1)\ \text{wit-by}\ W''
\end{array}\right)
\end{array}\right) \land \\
((s' \models_M \psi_2)\ \text{wit-by}\ W')
\end{array}\right)
\end{array}
$$

Being an ECTL witness: conjunction

$$
\begin{array}{l}
(s \vDash_M \psi_1 \wedge \psi_2) \text{ wit-by } W \overset{def}{=} \\
\quad \exists W_1 \in \mathrm{Witness}_M, W_2 \in \mathrm{Witness}_M. \\
\quad \left(\begin{array}{l} W = \mathrm{WAnd}\ W_1\ W_2 \wedge \\ (s \vDash_M \psi_1) \text{ wit-by } W_1 \wedge (s \vDash_M \psi_2) \text{ wit-by } W_2 \end{array} \right)
\end{array}
$$

Being an ECTL witness: disjunction

$$\begin{array}{l}
(s \vDash_M \psi_1 \vee \psi_2) \text{ wit-by } W \stackrel{\textit{def}}{=} \\
\quad \exists W' \in \mathrm{Witness}_M . \\
\qquad \left(\begin{array}{l} \left(\; W = \mathrm{WOrL}\ W' \wedge (s \vDash_M \psi_1) \text{ wit-by } W' \; \right) \vee \\ \quad \left(\; W = \mathrm{WOrR}\ W' \wedge (s \vDash_M \psi_2) \text{ wit-by } W' \; \right) \end{array} \right)
\end{array}$$

Satisfiability and existence of witnesses. Fixed

Here we have required finite temporal models, and so witnesses are finite. (Otherwise, we would need to deal with infinite witnesses.)

Now, if we have $M \not\models \psi$ for some ACTL formula ψ, there exists a witness W for the fact that the ECTL formula corresponding to $\neg\psi$ holds — and we could effectively find it by tweaking our model checking algorithm (details elided).

Witnesses beyond ECTL

Can we have witnesses for more than just ECTL?

Yes. For example, one of the nice things about LTL is that counterexamples are just paths.

Summary

We saw a model checking algorithm for CTL, and sketched how it could be modified to generate counterexamples for ACTL formulas.