Algorithms for Model Checking (2IW55)

Lecture 2

Symbolic Model Checking for CTL

(“Model Checking”, Chapter 2, 6.1, 6.2. Also read Chapter 5.)

Tim Willemse
(timw@win.tue.nl)
http://www.win.tue.nl/~timw
MF 7.073

Technische Universiteit Eindhoven University of Technology

Outline

Fixed Points

Model checking complexity:

- In general, there are infinitely many states and transitions.
- Many of the states behave very similarly (e.g. the start value of some variables may not matter)
- We're interested in an algorithm that can benefit from this.

Fixed Points

Consider a Kripke Structure $M = \langle S, R, L \rangle$

In what follows, we (temporarily) ignore the difference between syntax and semantics

- Identify sets of states and predicates on states
- So, two notations are often mixed:
  - subsets: $X \subseteq S$ or $X \in \mathcal{P}(S)$, versus
  - predicates: $X \in 2^S$ or $X : S \to \{0, 1\}$
    $s \in X \Leftrightarrow X(s) = 1$ and $s \notin X \Leftrightarrow X(s) = 0$
- In general: we identify CTL formulae with the set of states where they hold: f versus $\{s \mid s \models f\}$
- We freely mix $\vee, \wedge$ and $\cup, \cap$: compare $\emptyset \cup \mathrm{E\ G}\ f$ and $\mathrm{false} \vee \mathrm{E\ G}\ f$

Predicate Transformers and Monotonicity

Consider a Kripke Structure $M = \langle S, R, L \rangle$

- The set $(\mathcal{P}(S), \subseteq)$ is a complete lattice.
- A predicate transformer is a function on predicates. For example, the relations *Pre* and *Post* that lift the transition relation R to sets of states:

$$\begin{array}{ll} Pre_R(X) & = \{s \in S \mid \exists t \in X.\ s\ R\ t\} \\ Post_R(X) & = \{t \in S \mid \exists s \in X.\ s\ R\ t\} \end{array}$$

- Let $\tau : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ be an arbitrary predicate transformer.
- τ is monotonic iff $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$.
- We write $\tau^i(X)$ for applying τ i times to X:

$$\begin{cases} \tau^0(X) & = X \\ \tau^{i+1}(X) & = \tau(\tau^i(X)) \end{cases}$$

TU/e Technische Universiteit Eindhoven University of Technology

Fixed Points

Let $\tau : \mathcal{P}(S) \to \mathcal{P}(S)$.

- A fixed point of τ is a set Z such that $\tau(Z) = Z$
- The least fixed point of τ, denoted $\mu X.\tau(X)$ is a set Z such that:
  - $Z = \tau(Z)$ (i.e. Z is a fixed point)
  - for all X, if $\tau(X) = X$, then $Z \subseteq X$
- The greatest fixed point of τ, denoted $\nu X.\tau(X)$ is a set Z such that:
  - $Z = \tau(Z)$ (i.e. Z is a fixed point)
  - for all X, if $\tau(X) = X$, then $X \subseteq Z$

A theorem by Tarski: a monotonic operator on $\mathcal{P}(S)$ always has least and greatest fixed points:

- $\mu Z.\tau(Z) = \bigcap\{X \mid \tau(X) \subseteq X\}$
- $\nu Z.\tau(Z) = \bigcup\{X \mid X \subseteq \tau(X)\}$

Fixed Points

Assume now that:

- S (hence also $\mathcal{P}(S)$) is finite, and
- $\tau : \mathcal{P}(S) \to \mathcal{P}(S)$ is monotonic

Then:

1. $\forall i . \tau^i(\emptyset) \subseteq \tau^{i+1}(\emptyset)$ .... (induction on i and monotonicity)
2. There exists an i such that $\tau^i(\emptyset) = \tau^{i+1}(\emptyset)$ ..... (sets become bigger and S is finite)
3. If $\tau^i(\emptyset) = \tau^{i+1}(\emptyset)$, then $\tau^i(\emptyset)$ is a fixed point of τ .... (by definition)
4. If X is a fixed point of τ, then $\forall i . \tau^i(\emptyset) \subseteq X$ ...... (induction on i and monotonicity)

So an approximant τ^i can be found such that $\tau^i(\emptyset) = \tau^{i+1}(\emptyset)$, and this set is the least fixed point of τ.

Similarly, the smallest i such that $\tau^i(S) = \tau^{i+1}(S)$ yields the greatest fixed point.

Fixed Points

Algorithms for computing the least fixed point and the greatest fixed point based on the observations on the previous slide.

function $\mathsf{lfp}(\tau{:}\mathcal{P}(S){\to}\mathcal{P}(S)) : \mathcal{P}(S)$
  $Q := \emptyset$;
  $Q' := \tau(Q)$;
  while $Q \neq Q'$ **do**
    $Q := Q'$;
    $Q' := \tau(Q')$;
  end while
  return Q;
end function

function $\mathsf{gfp}(\tau{:}\mathcal{P}(S){\to}\mathcal{P}(S)) : \mathcal{P}(S)$
  $Q := S$;
  $Q' := \tau(Q)$;
  while $Q \neq Q'$ **do**
    $Q := Q'$;
    $Q' := \tau(Q')$;
  end while
  return Q;
end function

Outline

Fixed Point Algorithm for CTL

Recall that CTL has the following ten temporal operators:

- A X and E X : for all/some next state
- A F and E F : inevitably and potentially
- A G and E G : invariantly and potentially always
- A [U] and E [U]: for all/some paths, until
- A [R] and E [R]: for all/some paths, releases

Besides atomic propositions (AP), the constant true and the Boolean connectives ($\neg, \vee$), the following temporal operators are sufficient: E X , E G , E [U].

Hence: only algorithms for computing formulae of the above form are needed.

Fixed Point Algorithm for CTL

CTL operators can be seen as fixed point operators. Fix a Kripke Structure $M = \langle S, R, L \rangle$. Identify a CTL formula f with predicate $\{s \mid s \models f\}$.

- $\text{A X } f = \neg \text{E X } \neg f$ and $\text{E X } f = Pre_R(f)$
- $\text{A F } f = \mu Z.f \cup \text{A X } Z$ and $\text{E F } f = \mu Z.f \cup \text{E X } Z$
- $\text{A G } f = \nu Z.f \cap \text{A X } Z$ and $\text{E G } f = \nu Z.f \cap \text{E X } Z$
- $\text{E } [f \text{ U } g] = \mu Z.g \cup (f \cap \text{E X } Z)$

Intuition:

- least and greatest fixed points deal differently with loops:
  - Greatest fixed point: recursion includes loops, so possibly infinitely many "steps"
  - Least fixed point: finite recursion through loops, so only finitely many "steps"
- Eventualities .... least fixed points
  (a witness of the eventuality is needed in finitely many steps)
- Globally .... greatest fixed points
  (an infinite path without error is OK)

Technische Universiteit Eindhoven University of Technology

Fixed Point Algorithm for CTL

Proof obligations for E G :

1. The transformer $Z \mapsto f \wedge \text{E X } Z$ is monotonic, so its fixed point can be computed by iteration, see lfp and gfp
   (If $Z_1 \subseteq Z_2$ then $f \wedge \text{E X } Z_1 \subseteq f \wedge \text{E X } Z_2$).
2. E G f is a fixed point of $Z \mapsto f \wedge \text{E X } Z$
   ($\text{E G } f = f \wedge \text{E X E G } f$)
3. E G f is the largest such fixed point
   (for all Z: if $Z = f \wedge \text{E X } Z$, then $Z \subseteq \text{E G } f$)

- For 1,2,3: prove $X \subseteq Y$ by $\forall s . s \in X \Rightarrow s \in Y$.
- For 2: prove $\subseteq$ and $\supseteq$.
- For 2,3: use the semantics of CTL-formulae

Proof obligations for E [U] are similar (see for yourself)

Proofs for (2):
E G f = $f \wedge$ E X E G f follows from:

- "$\subseteq$". Let $s \in$ E G f. Then $\exists \pi \in \text{path}(s) : \forall i : \pi^i \models f$. Let π be such a path starting in s. We have

$$\pi^0 \models f \quad \text{(a)}$$
$$\forall j : \pi^{j+1} \models f \quad \text{(b)}$$

  Since f is a state formula (we are in the CTL fragment), from (a) we conclude $s \models f$. By definition, from (b) we conclude $\pi^1 \models$ G f. Hence $\pi^1(0) \models$ E G f. Since $sR\pi^1(0)$, we have $s \models$ E X E G f. Thus $s \in f \wedge$ E X E G f.

- "$\supseteq$". Let $s \in f \wedge$ E X E G f. Then $s \in f$ and sRt for some $t \in$ E G f. But then also $s \in$ E G f.

Fixed Point Algorithm for CTL

Proofs sketches for (3):
Let $\tau(Z) = f \wedge \text{E X } Z$. We show that $\text{E G } f = \bigcap_i \tau^i(S)$ by proving both inclusions.

- "$\subseteq$". By means of an induction on i. Clearly $\text{E G } f \subseteq S = \tau^0(S)$. Assume $\text{E G } f \subseteq \tau^i(S)$. Then by monotonicity, we have $\tau(\text{E G } f) \subseteq \tau^{i+1}(S)$. Since $\text{E G } f$ is a fixpoint of τ, we have $\tau(\text{E G } f) = \text{E G } f$. So $\text{E G } f \subseteq \tau^{i+1}(S)$.

- "$\supseteq$". Let $s \in \bigcap_i \tau^i(S)$. Observe that since S is finite and τ is monotonic, $\tau^k(S) = \tau^{k+1}(S)$ for some k, and $\tau^l(S) \supseteq \tau^k(S)$ for all $l \leq k$. Then, for every $s \in \tau^k(S)$, we have (by definition of τ) $s \in f$ and $s \in \text{E X } \tau^k(S)$. So, for all $t \in \tau^k(S)$, we have tRu for some $u \in \tau^k(S)$. Therefore, there must be some infinite path starting in s, satisfying f. Thus, $s \in \text{E G } f$.

CTL model checking with Fixed Points

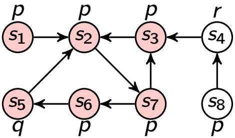
Function check(f) takes a formula f and returns the set of states where f holds: $\{s \mid s \models f\}$ (given a fixed Kripke Structure $M = \langle S, R, L \rangle$).

check(p)	$\{s \mid p \in L(s)\}$
check($\neg f$)	$S \setminus$ check(f)
check($f \vee g$)	check(f)$\cup$ check(g)
check(E X f)	Pre_R(check(f))
check(E [f U g])	lfp$\big(Z \mapsto$ check(g) $\cup$ (check(f) $\cap$ $Pre_R(Z)$)$\big)$
check(E G f)	gfp$\big(Z \mapsto$ check(f) $\cap$ $Pre_R(Z)\big)$

Recall: $Pre_R(Z) = \{s \in S \mid \exists t \in Z . s\ R\ t\}$

Example

- To check: E [p U q]
- Compute: $\mu Z.q \vee (p \wedge \text{E X } Z)$ (with lfp)



$$
\begin{aligned}
Z_0 &= \text{false} = \emptyset \\
Z_1 &= q \vee (p \wedge \text{E X } Z_0) = \{s_5\} \\
Z_2 &= q \vee (p \wedge \text{E X } Z_1) = \{s_5, s_6\} \\
Z_3 &= q \vee (p \wedge \text{E X } Z_2) = \{s_5, s_6, s_7\} \\
Z_4 &= q \vee (p \wedge \text{E X } Z_3) = \{s_2, s_5, s_6, s_7\} \\
Z_5 &= q \vee (p \wedge \text{E X } Z_4) = \{s_1, s_2, s_3, s_5, s_6, s_7\} \\
Z_6 &= q \vee (p \wedge \text{E X } Z_5) = \{s_1, s_2, s_3, s_5, s_6, s_7\}
\end{aligned}
$$

$Z_5 = Z_6$, so this is the least fixed point.

Outline

Example (GCD)

Consider the following program:

```
repeat
   if x > y-> x := x - y;
   []x < y-> y := y - x;
   fi
until false
```

This program uses:

- variables: $\{x, y\}$, with an (implicit) domain of variables: $\mathbb{N}$
- States of this program are functions of type: $\{x, y\} \to \mathbb{N}$
- An example state could be: $\{x \mapsto 5, y \mapsto 15\}$
- An execution is a sequence of transitions: e.g.

$$\{x \mapsto 5, y \mapsto 15\} \to \{x \mapsto 5, y \mapsto 10\} \to \{x \mapsto 5, y \mapsto 5\} \to \{x \mapsto 5, y \mapsto 5\} \to \ldots$$

Example (SWAP)

Consider the following program fragment:

$$\begin{array}{ll} z := x; & \%\ \text{l1} \\ x := y; & \%\ \text{l2} \\ y := z; & \%\ \text{l3} \end{array}$$

- Besides variables $x, y, z : \mathbb{N}$, this program has a program counter, whose values are labels (line numbers)
- Let $pc : \{l_1, l_2, l_3\}$. Now, a state is a function that gives a value to $\{x, y, z, pc\}$
- A possible execution is the following sequence:

$$\begin{array}{ll} & \{x \mapsto 5, y \mapsto 15, z \mapsto 500, pc \mapsto l_1\} \\ \rightarrow & \{x \mapsto 5, y \mapsto 15, z \mapsto 5, pc \mapsto l_2\} \\ \rightarrow & \{x \mapsto 15, y \mapsto 15, z \mapsto 5, pc \mapsto l_3\} \\ \rightarrow & \{x \mapsto 15, y \mapsto 5, z \mapsto 5, pc \mapsto l_4\} \end{array}$$

Idea: the set of states can be represented very concisely by a number of formulae

- for GCD:
  - initial set of states: $x < 100 \wedge y < 100$
  - next state predicate:

$$(x > y \wedge x' = x - y \wedge y' = y) \vee (x < y \wedge y' = y - x \wedge x' = x)$$

- for SWAP:
  - initial states: $x = 5 \wedge y = 15$
  - next state predicate:

$$(pc = l_1 \wedge pc' = l_2 \wedge z' = x \wedge \ldots) \vee \ldots$$

The system specification is represented by propositional logic formula

- Let V be a set of variables $v_0, v_1, \ldots, v_n$
- Let D be the domain of these variables
- The states of the Kripke Structure will be functions $v : V \to D$
- A formula $S_0(V)$ represents the initial states
- Let V' be a copy of the variables in V: $v'_0, v'_1, \ldots, v'_n$
- A formula $\mathcal{R}(V, V')$ represents the transition relation.
  - V denotes the value of the variables before the transition
  - V' denotes the value of the variables after the transition.

Example

- $V = \{TL_1, TL_2\}$,
- $D = \{r(ed), y(ellow), g(reen)\}$
- $\mathcal{S}_0(TL_1, TL_2) := \; TL_1 = r \wedge TL_2 = r$
- $\mathcal{R}(TL_1, TL_2, TL'_1, TL'_2) := \; R_1 \vee R_2 \vee R_3 \vee R_4 \vee R_5 \vee R_6$, where:
  - $R_1 := \; TL_1 = r \wedge TL'_1 = g \wedge TL'_2 = TL_2$
  - $R_2 := \; TL_1 = g \wedge TL'_1 = y \wedge TL'_2 = TL_2$
  - $R_3 := \; TL_1 = y \wedge TL'_1 = r \wedge TL'_2 = TL_2$
  - $R_4 := \; TL_2 = r \wedge TL'_2 = g \wedge TL'_1 = TL_1$
  - $R_5 := \; TL_2 = g \wedge TL'_2 = y \wedge TL'_1 = TL_1$
  - $R_6 := \; TL_2 = y \wedge TL'_2 = r \wedge TL'_1 = TL_1$

Notes:

- this corresponds to a Kripke Structure modelling an unsafe traffic light system at a junction
- a specification for n traffic lights gives $O(3^n)$ states $\Rightarrow$ State space explosion

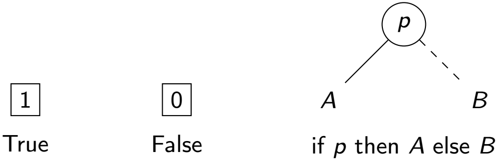
Symbolic Model Checking

We wish to avoid representing the state space and its subsets explicitly. To efficiently implement symbolic model checking, we need:

- A concise representation of sets of states
- Quick operations for:
  - Boolean operators $\wedge, \vee, \neg$
  - Existential quantification (for the relational composition)
  - Equivalence test

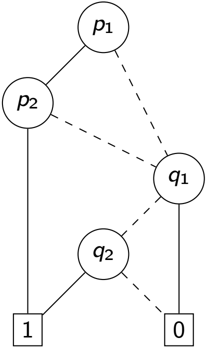
Solution: *Ordered Binary Decision Diagrams (OBDD)*

- Symbolic model checking is restricted to finite Kripke Structures
- All finite data can be encoded in “bits”
- Boolean functions can be represented concisely as (Ordered) Binary Decision Diagrams
- Binary Decision Diagrams are directed acyclic graphs, with the following ingredients:



Symbolic Model Checking

BDD representation of $(p_1 \wedge p_2) \vee (\neg q_1 \wedge q_2)$:



- In ordered BDDs, tests along a path occur in a fixed order (e.g. $p_1 < p_2 < q_1 < q_2$).
- Theorem[Bryant'86]: OBDDs are a unique representation for Boolean Functions.
- Claim: many practical formulae have a concise OBDD representation due to maximal sharing
- Disclaimer 1: some small formulae have only exponentially large BDDs. (multiplier)
- Disclaimer 2: the size of an OBDD can crucially depend on the ordering of the variables

More on OBDDs:

- OBDDs are implemented as maximally shared pointer structures in memory.
- The order of variables is fixed (some implementations feature dynamic reordering)
- Equivalence test can be performed in constant time, in particular, also checking for satisfiability and tautology.
- Boolean operations can be performed efficiently. Let B_1 and B_2 be OBDDs with m and n nodes, respectively, then:
  - OBDDs for $B_1 \wedge B_2$ and $B_1 \vee B_2$ can be computed in $\mathcal{O}(m \cdot n)$ time.
  - OBDDs for $\neg B_1$ can be computed in $\mathcal{O}(m)$ time.
  - the OBDD of $\exists x.B_1$ can be computed in $\mathcal{O}(m^2)$ time.
- Note: still a formula of size $\mathcal{O}(n)$ may have a BDD of size $\mathcal{O}(2^n)$.

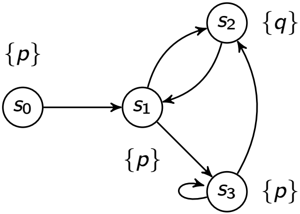
Attend *Automated Reasoning* (2IW15) for more information on OBDDs (Semester A.2).

Symbolic Model Checking

- The implementation of a symbolic model checking relies on a representation of all sets in check, lfp and gfp by OBDDs.
- Hence, in summary, symbolic model checking:
  - Recursively processes subformulae
  - Represent the set of states satisfying a subformula by OBDDs
  - Treats temporal operators by fixed point computations
  - Relies on efficient implementation of equivalence test, and $\wedge, \vee, \neg$ and $\exists$ connectives on OBDDs.

Exercise

Consider the following Kripke Structure:



Consider the following formulae, where p and q are atomic propositions:

(**A**) $\mathbf{A}(\mathbf{F}(q))$
(**B**) $\mathbf{A}[q\ \mathbf{R}\ p]$

Determine the set of states where (**A**) and (**B**) hold using the symbolic model checking algorithm for CTL . You may use explicit set notation to represents states.

Technische Universiteit Eindhoven University of Technology