Explicit-state Model Checking

CS 4271
Abhik Roychoudhury
National University of Singapore

Background

- Kripke Structures as models
- Temporal Properties
  - LTL, CTL*, CTL
- This lecture
  - Explicit state model checking algorithm for Computation Tree Logic (CTL)

CTL Model Checking

- **Given**
  - Finite state Kripke Structure M = (S,S0,R,L)
  - CTL formula f
- **Check whether**
  - All initial states of M satisfy f, that is,
    - $S0 \subseteq \{ s \mid s \in S \wedge M,s \models f \}$
- Explicit-state MC in this lecture.
  - Property checking by finite graph search.

Checking M |= f

- Define $St_f = \{s \mid s \in S \text{ and } M,s \models f\}$
  - Start with computing St_p for each atomic prop. p
    - $St_p = \{ s \mid s \in S \text{ and } p \in L(s) \}$
  - Computation of St_f proceeds by a bottom-up parse of the formula f
    - Compute St_g for each sub-formula g of formula f
  - Check whether $S0 \in St_f$
    - Details of counter-example construction are not discussed in this lecture.

CTL syntax

- $f := p \mid f \wedge f \mid \neg f \mid AX\ f \mid EX\ f \mid$
- $AG\ f \mid EG\ f \mid AF\ f \mid EF\ f \mid$
- $A(f\ U\ f) \mid E(f\ U\ f) \mid A(f\ R\ f) \mid E(f\ R\ f)$
- The ten temporal operators can be expressed in terms of EX, EG, EU
  - We will justify this !
- So, our MC algorithm needs to consider only
  - $f := p \mid f \wedge f \mid \neg f \mid EX\ f \mid EG\ f \mid E(f\ U\ f)$

CTL operators

- $AX\ \varphi = \neg\neg AX\varphi = \neg EX\ \neg\varphi$
- $AG\varphi = \neg\neg AG\varphi = \neg EF\ \neg\varphi$
- $EF\varphi = E(\text{ true } U\ \varphi)$
- $AF\varphi = \neg EG\ \neg\varphi$
- $A(\varphi\ R\ \Psi) = \neg\neg A(\varphi\ R\ \Psi) = \neg E(\neg\varphi\ U \neg\Psi)$
- Can you derive the above equivalences ?

CTL operators

- $E(\varphi \; R \; \Psi) = \neg A(\neg \varphi \; U \; \neg \Psi)$
- What about $A(\varphi \; U \; \Psi)$??

- $\varphi \; R \; \Psi = (\Psi \; U \; (\varphi \wedge \Psi)) \vee G \Psi$
  - Prove this result
  - Use this result to define $E(\varphi \; R \; \Psi)$ and hence $A(\varphi \; U \; \Psi)$

Structure of MC algorithm

- To check M = (S,S0,R,L) |= f
  - 1. Rewrite f to an equivalent CTL formula f1 where f1 contains only the operators $\neg$, $\wedge$, EX, EG, EU
  - 2. Find(M,f1)
    - For all sub-formula g1 of f1 do{
    - if g1 = atomic prop then St_{g1} := …
    - else Find(M,g1)
    - }
    - Construct St_{f1} from St_{g1} computed above
    - Return St_{f1}
  - 3. If $S0 \subseteq St_{f1}$ then return "yes" else return "no"

Computing St_f

- Kripke Structure M = (S, S0, R, L)
  - Case 1: f = p
    - $St_p = \{s \mid s \in S \text{ and } p \in L(s)\}$
  - Case 2: $f = \neg g$
    - $St_{\neg g} = S - St_g$
  - Case 3: $f = g1 \wedge g2$
    - $St_{g1 \wedge g2} = St_{g1} \cap St_{g2}$
  - Case 4: f = EX g
    - $St_{EXg} = \{s \mid s \in S \wedge (s,t) \in R \wedge t \in St_g\}$

Computing St_{f1}

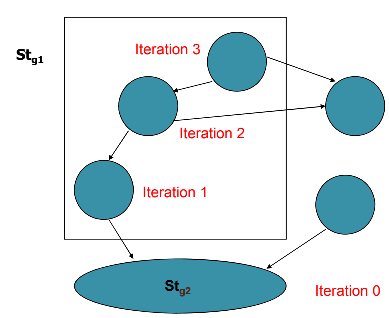
- There are two more cases
  - Case 5: f = E(g1 U g2)
  - Case 6: f = EG f1
- We now give search algorithms for these two cases.

- So, the overall algorithm is

Find(M,φ)

- Let M = (S, S0, R, L)
- If φ is true then return S
- Else if φ is false then return null-set;
- Else if φ is $\neg\Psi$ then return S – Find(M, Ψ)
- Else if φ is $\Psi1 \wedge \Psi2$ then return Find(M,$\Psi1$)$\cap$Find(M, $\Psi2$)
- Else if φ is AXΨ then return Find(M, $\neg$EX$\neg\Psi$)
- Else if φ is EXΨ then ***call EX algorithm and return results***;
- Else if φ is E($\Psi1$U$\Psi2$) then ***call EU algorithm and return results***;
- Else if φ is A($\Psi1$U$\Psi2$) then return ?? [do it yourself now]
- Else if φ is EGΨ then ***call EG algorithm and return results***;
- Else if …. [fill up the rest of the cases yourself]

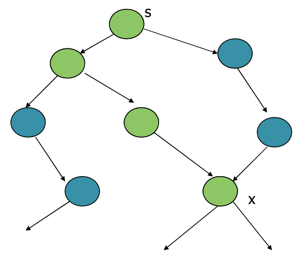
E(g1 U g2) : Intuition



Computing $St_{E(g1\ U\ g2)}$

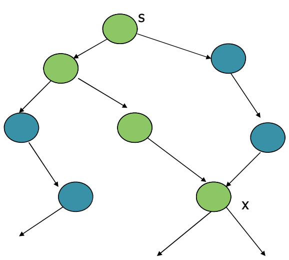
- We assume that St_X has been computed for all sub-formula X of E(g1 U g2)
  - Thus, St_{g1} and St_{g2} must have been computed.
- Need to find states from which a state in St_{g2} is forward reachable using only state in St_{g1}
- Accomplished by
  - Start from states in St_{g2}
  - Perform backwards reachability analysis using only states in St_{g1}. All these states are in $St_{E(g1\ U\ g2)}$

Forward reachability



x is forward reachable from s

Backward reachability



s is backward reachable from x

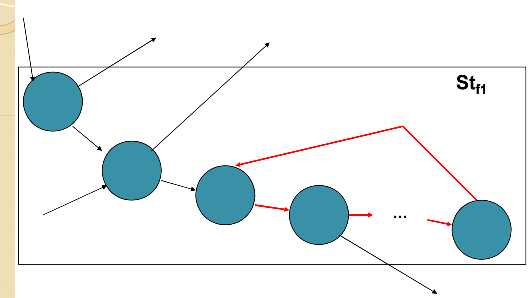
Checking EU

- Inputs:
  - Kripke Structure M = (S, S0, R, L).
  - CTL formula to be checked E(g1 U g2)
  - St_{g1}, set of states satisfying g1 in M.
  - St_{g2}, set of states satisfying g2 in M.
- Output:
  - Set of states satisfying E(g1 U g2) in M.
- Technique:
  - ***Traversing the states (and transitions) of M.***

E(g1 U g2) : Algorithm

- Result := St_{g2};
- Temp := St_{g2};
- while Temp ≠ empty do
- pick s ∈ Temp; Temp := Temp – {s};
- Backstep:= {s1 | s1 R s, and s1 ∈ St_{g1}};
- Temp := Temp ∪ Backstep;
- Result := Result ∪ Backstep;
- endwhile;
- return Result;

EG f1 : Intuition



Case 6: f = EG f1

- Inputs:
  - Kripke Structure M = (S, S0, R, L).
  - CTL formula to be checked EG f1
  - St_{f1}, set of states satisfying f1 in M.
- Output:
  - Set of states satisfying EG f1 in M.
- Technique:
  - ***Traversing the states (and transitions) of M.***

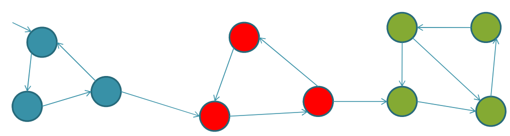
EG f1 : Algorithm

- Result := St_{f1};
- **repeat**
- Temp := { s | s $\in$ Result , and
- $\forall$s1. s R s1 $\Rightarrow$ s1 $\notin$ Result } ;
- Result := Result – Temp;
- **until** Temp = empty;
- **return** Result;

How to make it more efficient

- We initialize $St_{EGf1} = St_{f1}$
  - For each state in St_{f1}, we check the out-edges. Many of the destination states are not in St_{f1}, so cannot satisfy EGf1
- It suffices to consider a reduced Kripke Structure M' constructed from M such that
  - All states of M which satisfy f1 are retained.
  - All other states and transitions are deleted.
- For any s, M,s |= EG f1 if and only if
  - s is a state in M'
  - s reaches a state s' in M' where s' loops back to itself.

Strongly Connected Components



Strongly connected components of a graph G are maximal sub-graphs $\{C_1,\ldots,C_k\}$ of G, such that every node in C_i has a path to any other node in C_i

Efficiently computing EG f1

- Input: M = (S,S0,R,L) , St_{f1}
- Output: St_{EGf1}
- Technique:
- Compute M'=(S',S0',R',L') from M by keeping only nodes in St_{f1}
- Temp := St_{EGf1} := All nodes in nontrivial SCCs of M'
- while Temp $\neq$ empty do
- pick s $\in$ Temp; Temp := Temp – {s};
- St_{EGf1} := St_{EGf1} $\cup$ { t | t R' s $\wedge$ t $\notin$ St_{EGf1} };
- Temp := Temp $\cup$ { t | t R' s $\wedge$ t $\notin$ St_{EGf1} };
- endwhile

Complexity of Model Checking

- In terms of
  - $|\varphi|$ size of formula
  - |S| number of states in M
  - |R| number of transitions
- At each level of nesting of φ
  - Employ the EG, EU,EX algorithms
  - Efficient EG algorithm is O(|S| + |R|)
  - Similarly for EU, EX algorithms
- Complexity is $O(|\varphi| * (|S| + |R|))$

Exercise

- The previous slides give iterative algorithms for computing St_{EGf} and $St_{E(f\ U\ g)}$
- These algorithms can be used to indirectly compute
  - St_{EFf}, St_{AFf}, St_{AGf}
- Construct iterative algorithms for directly computing $St_{EF\ f}$, $St_{AF\ f}$, St_{AGf} without exploiting the translation of CTL formulae.

AF

- ***Input*** $M = (S, S0, \rightarrow, L)$
- St_{φ}
- ***Output*** $St_{AF\varphi}$
- ***Algorithm***
- $St_{AF\varphi}$:= St_{φ}
- repeat{
- Addition := {s | s $\notin$ St(AFφ) and
- $\forall s \rightarrow t\ \ t \in St_{AF\varphi}$ }
- $St_{AF\varphi}$:= $St_{AF\varphi}$ U Addition
- } until $St_{AF\ \varphi}$ not changed from last iteration;
- return $St_{AF\varphi}$

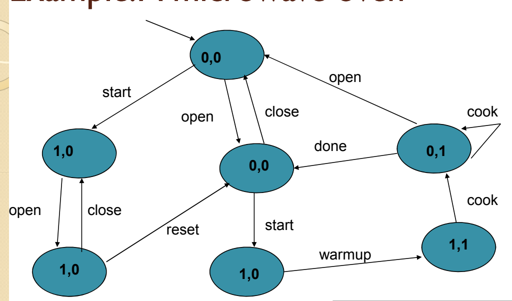
EF

- ***Input*** $M = (S, S0, \rightarrow, L)$
- St_{φ}
- ***Output*** $St_{EF\varphi}$
- ***Algorithm***
- $St_{EF\varphi}$:= St_{φ}
- repeat{
- Addition := {s | s $\notin$ $St_{EF\varphi}$ and
- $\exists\ s \rightarrow t\ \ t \in St_{EF\varphi}$ };
- $St_{EF\varphi}$:= $St_{EF\varphi}$ U Addition;
- } until $St_{EF\ \varphi}$ not changed from last iteration;
- return $St_{EF\varphi}$

AG

- ***Input*** $M = (S, S0, \rightarrow, L)$
- St_{φ}
- ***Output*** $St_{AG\varphi}$
- ***Algorithm***
- $St_{AG\varphi}$:= St_{φ};
- repeat{
- Deletion := {s | s $\in$ $St_{AG\varphi}$ and
- $\exists\ s \rightarrow t\ \ t \notin St_{AG\varphi}$ } ;
- $St_{AG\varphi}$:= $St_{AG\varphi}$ - Deletion;
- } until $St_{AG\varphi}$ not changed from last iteration;
- return $St_{AG\varphi}$

Example: A microwave oven



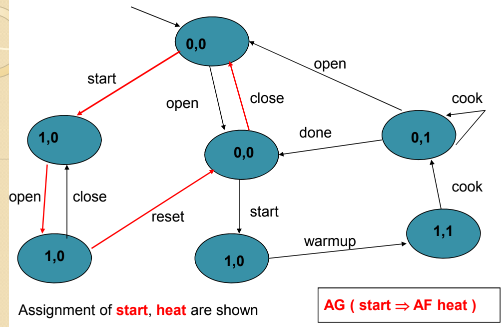
Assignment of **start**, **heat** are shown

AG (start ⇒ AF heat)

Example

- AG (start ⇒ AF heat)
  - For any reachable state, if start holds, then along all outgoing paths, heat eventually holds.
  - Can be Violated if:
    - $\exists$ a reachable state s where start holds
    - $\exists$ an acyclic path from s to s' in which heat does not hold in any state
    - And there is a cycle containing s' such that heat does not hold in all states of the cycle.

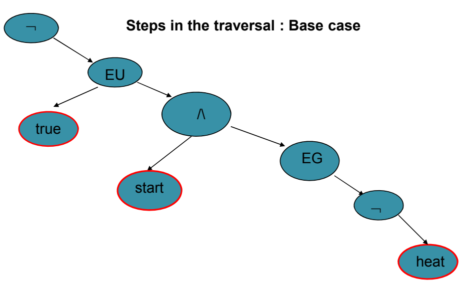
Example: A microwave oven



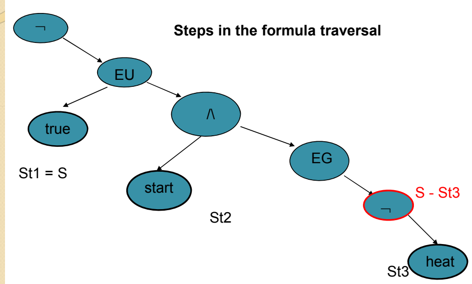
Example

- M = the model of microwave oven given earlier
- φ = AG (start $\Rightarrow$ AF heat)
- = $\neg$ EF $\neg$($\neg$start $\vee$ AF heat)
- = $\neg$ EF (start $\wedge$ $\neg$ AF heat)
- = $\neg$ EF (start $\wedge$ EG $\neg$ heat)
- = $\neg$ E(true U (start $\wedge$ EG $\neg$ heat))
- Now, how to compute the set of states in M satisfying this formula ?
  - Bottom up traversal of the formula

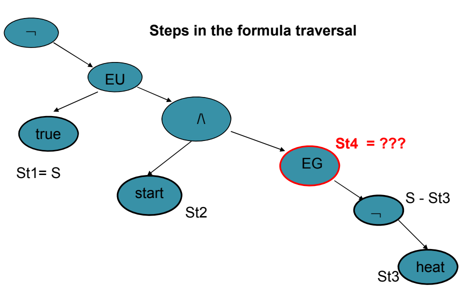
Bottom-up traversal



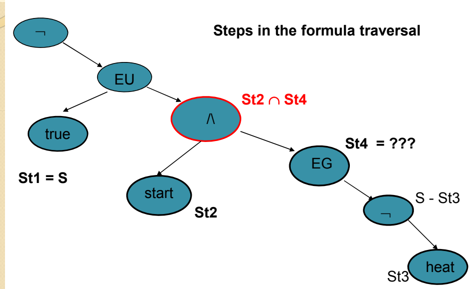
Bottom-up traversal



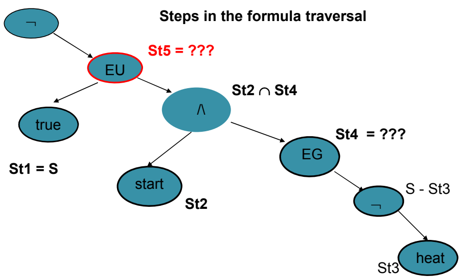
Bottom-up traversal



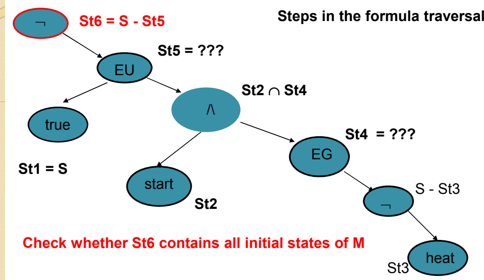
Bottom-up traversal



Bottom-up traversal

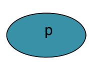


Bottom-up traversal



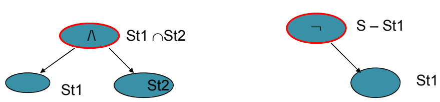
Bottom-up formula traversal

Atomic propositions



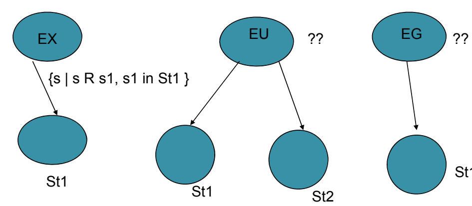
The set of states satisfying p in M = (S,S0,R,L), is found from L, the labeling function of M

Boolean operators: ∧, ¬



Questions Remaining

Temporal operators: EX, EU, EG



Answer:

Use the model checking algorithms for EU and EG that we discussed.

Summary

- In this class:
  - Explicit –state Model Checking Algorithm for Computation Tree Logic (CTL).
- In Future:
  - Symbolic model checking for space efficiency.

Exercise

- Two students are taking the CS4271 exam. We must ensure that they cannot leave the exam hall at the same time. To prevent this, each student reads a shared token n before leaving the hall. The shared token is an arbitrary natural number. The global state of the system is given by **<s1, s2, n>** where s1 and s2 are the local states of students 1 and 2 respectively; s1 ∈ {in, out}, s2 ∈{in, out}. The pseudo-code executed by the two students is given below. The two student processes are executed asynchronously. Every time one process is scheduled, it atomically executes one iteration of its loop. Initially s1 =in and s2 = in.

```
do forever{                                      do forever{
   if (s1 = in & n is odd) {s := out}             if (s2 = in & n is even) {s2:= out}
   else if (s1=out)  {s1:=in;n:=3*n+1}           else if (s2=out & n is even)
{s2:=in;n:=n/2}
   else {do nothing}                                else {do nothing}
}                                                }
```

- Earlier we had drawn the Kripke Structure for this example by maintaining approx info about n (even/odd). Now state the mutual exclusion property in CTL and model check it using the algorithm we discussed in class.