Explicit-state Model Checking

Abhik Roychoudhury
Department of Computer Science
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,→,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.

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 is not discussed in this lecture.

CTL syntax

- f := p | f ∧ f | ¬f | AX f | EX f |
-   AG f | EG f | AF f | EF f |
-   A(f U f) | E(f U f) | A(f R f) | 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 | f ∧ f | ¬f | EX f | EG f | 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 ?

Class Practice

- $E(\varphi\ R\ \Psi) = \neg A(\neg \varphi\ U \neg \Psi)$
- 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 $A(\varphi\ U\ \Psi)$

Structure of MC algorithm

- To check $M = (S, S0, \rightarrow, L) \models f$
  - 1. Rewrite f to an equivalent CTL formula f1 where f1 contains only the operators $\neg$, $\wedge$, EX, EG, EU
  - 2. Compute(f1)
    - For all sub-formula g1 of f1 do{
    - if g1 = atomic prop then St_{g1} := …
    - else Compute(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 $M = (S, S0, \rightarrow, L)$
  - 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
    - $St_{EXg} = \{s \mid s \in S \wedge (s,t) \in \rightarrow \wedge\ t \in St_g\}$

Computing St_{f1}

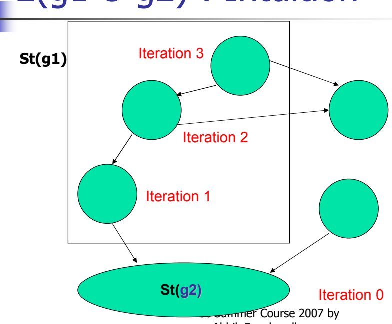
- There are two more cases
  - Case 5: f = E(g1 U g2)
  - Case 6: f = EG f1
- We now give search algorithms for these two cases.

- 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,Ψ1)$\cap$Find(M, Ψ2)
- 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(Ψ1UΨ2) then ***call EU algorithm and return results***;
- Else if φ is A(Ψ1UΨ2) 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]

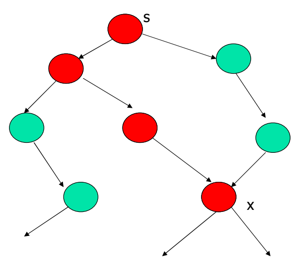
E(g1 U g2) : Intuition



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

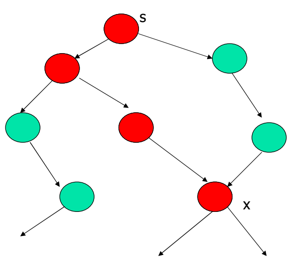
- We assume that St_{φ} has been computed for all sub-formula φ of E(g1 U g2)
  - Thus, St_{g1} and St_{g2} must have been computed.
- Need to find states from which a state in St_{g2} is forward reachable using only state in St_{g1}
- Accomplished by
  - Start from states in St_{g2}
  - Perform backwards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

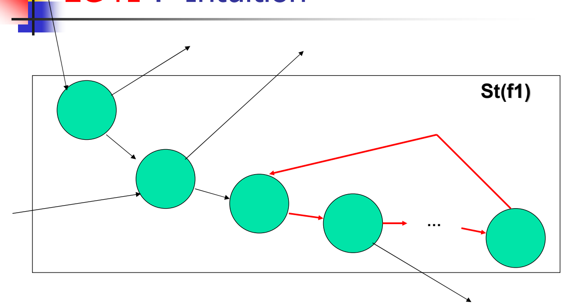
Checking EU

- 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 $\neq$ empty do
-     pick s $\in$ Temp; Temp := Temp – {s};
-     Backstep:= {s1 | s1 R s, and s1 $\in$ St(g1)};
-     Temp := Temp $\cup$ Backstep;
-     Result := Result $\cup$ Backstep;
- endwhile;
- return Result;

EG f1 : Intuition



Case 6: f = EG f1

- Inputs:
  - Kripke Structure M = (S, S0, →, 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

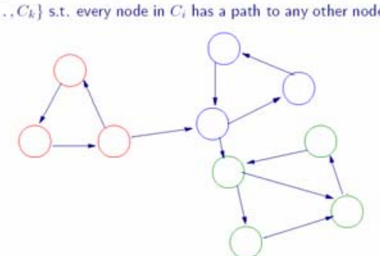
- Result := St(f1);
- **repeat**
- Temp := { s | s ∈ Result , and
- ∀s1. s → s1 ⇒ s1 ∉ Result } ;
- Result := Result – Temp;
- **until** Temp = empty;
- **return** Result;

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.

Recap: Strongly Connected Components

Strongly connected components of G are maximal subgraphs $\{C_1, \ldots, C_k\}$ s.t. every node in C_i has a path to any other node in C_i.



Efficiently computing EG f1

- Input: M = (S,S0,→,L) , St_{f1}
- Output: St_{EGf1}
- Technique:
- Compute M'=(S',S0',→',L') from M by keeping only nodes in St_{f1}
- Temp := St_{EGf1} := All nodes in nontrivial SCCs of M'
- while Temp ≠ empty do
- pick s ∈ Temp; Temp := Temp – {s};
- St_{EGf1} := St_{EGf1} ∪ { t | t →' s ∧ t ∉ St_{EGf1} };
- Temp := Temp ∪ { t | t →' s ∧ t ∉ St_{EGf1} };
- endwhile

Complexity of $M \models \varphi$

- In terms of
  - $|\varphi|$: size of formula (number of operators)
  - $|S|$: Total number of states in M
  - $|R|$: Total number of edges in M
- At each level of nesting of the formula φ, we employ the EG, EU algorithms shown before.
- The EG algorithm (efficient version) is $O(|S| + |R|)$. At most $|S|$ iterations of the loop.
- Similarly for EU, EX etc.
- Complexity of model checking is then $O(|\varphi| * (|S| + |R|))$.

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(AFφ)
- ***Algorithm***
- St(AFφ) := St(φ)
- repeat{
- Addition := {s | s $\notin$ St(AFφ) and
- $\forall$s $\rightarrow$ t t $\in$ St(AFφ) }
- St(AFφ) := St(AFφ) U Addition
- } until St(AF φ) not changed from last iteration;
- return St(AFφ)

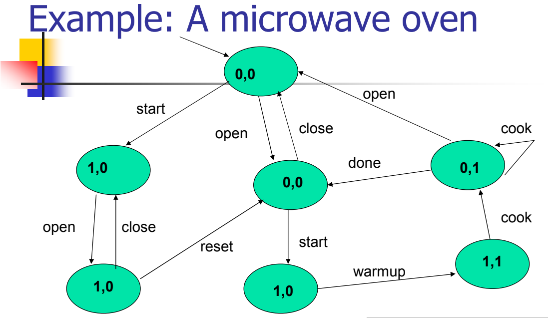
EF

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

AG

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

Example: A microwave oven



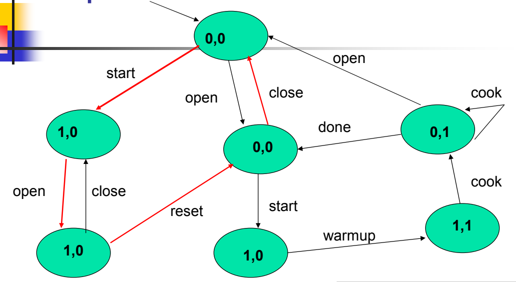
Assignment of **start**, **heat** are shown

AG (start $\Rightarrow$ AF heat)

Example

- AG (start $\Rightarrow$ 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

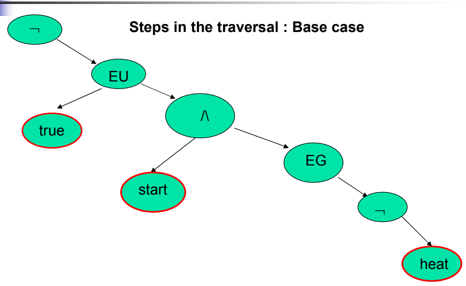


Assignment of **start**, **heat** are shown

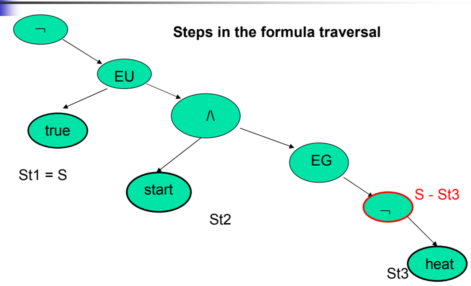
Example

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

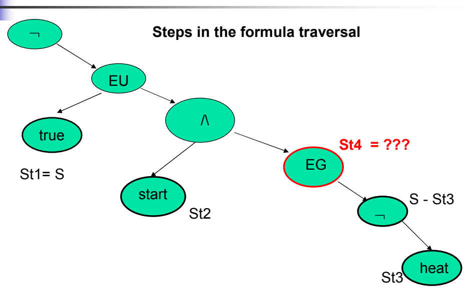
Bottom-up traversal



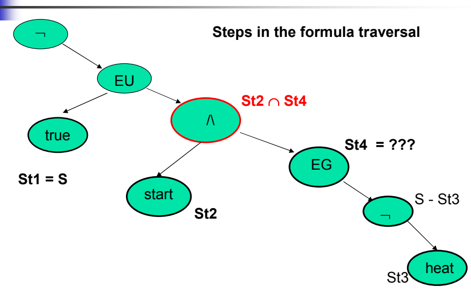
Bottom-up traversal



Bottom-up traversal

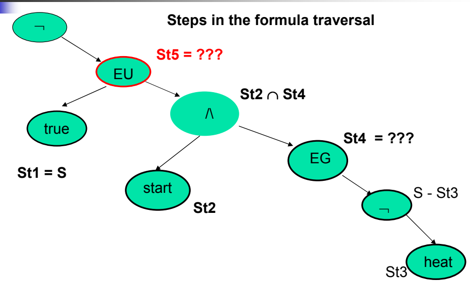


Bottom-up traversal



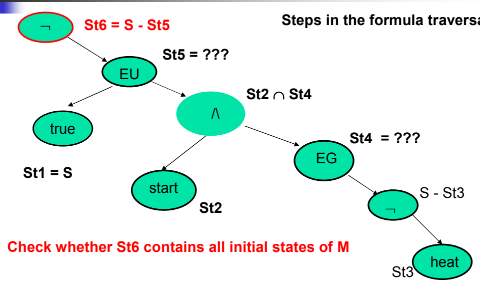
Bottom-up traversal

Steps in the formula traversal



Bottom-up traversal

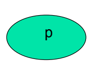
Steps in the formula traversal



Check whether St6 contains all initial states of M

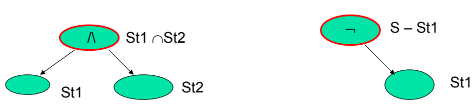
Bottom-up formula traversal

Atomic propositions



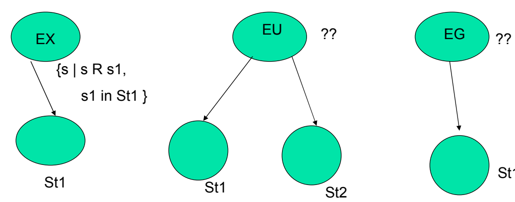
The set of states satisfying p in M = (S,S0,R,L), is found from L, the labeling function of M

Boolean operators: ∧, ¬



Questions Remaining

Temporal operators: EX, EU, EG



Answer:

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

Class Practice

- Two IISc students are taking the SW-Verification 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}
}                                                 }
```


Class Practice

- How would you construct the Kripke structure – you need to abstract variable n
- Now state the mutual exclusion property in CTL and model check it using the algorithm we discussed in class.

Class Practice

- We described CTL model checking by developing algorithms for
- EX, EG and EU. Show that it is also possible to express all the ten CTL
- operators using EX, EU and AU. Then, construct a bottom-up checking
- algorithm for AU, that is, given the set of states
- satisfying f and g, an algorithm for constructing the set of states satisfying A(f U g).