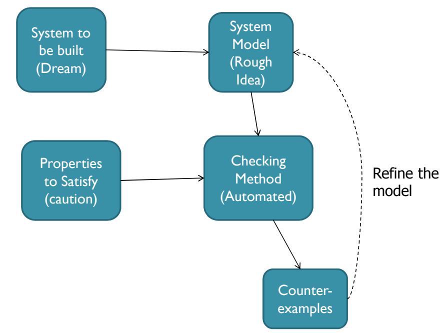
Symbolic Model Checking

CS 4271
Abhik Roychoudhury
National University of Singapore

Recap: the big picture



Efficient Model Checking

- Avoid graph construction and traversal.
  - Kripke Structure via propositional logic
    - Sets of states
    - Sets of transitions
  - Kripke Structure traversal via propositional logic formula manipulation.

So far …

- We have encoded transition systems as boolean functions. We can encode
  - Sets of states
  - Sets of transitions
- Now, to verify a temporal property φ, we will
  - Compute St_{φ} iteratively.
  - During the computation, we always maintain our current value of St_{φ} as a BDD
  - Compute St_{φ} iteratively by repeatedly applying boolean operations.
- Let us look at the case where φ = EG f.

Computing St_{EGf}

- Inputs:
  - Kripke Structure M = (S, S0, $\rightarrow$, L).
  - CTL formula to be checked EG f
  - St_f, set of states satisfying f in M.
- Output:
  - Set of states satisfying EG f in M

Computing St_{EGf}

- ***Technique***
  - Result := St_f ;
  - **repeat**
  - Temp := { s | s $\in$ Result , and
  - $\forall s'\ s \rightarrow s' \Rightarrow s' \notin$ Result } ;
  - Result := Result – Temp;
  - **until** Temp = empty;
  - **return** Result;

Just a little rewriting

- ***Technique***
  - Result := St_f ;
  - **repeat**
    - Result := Result $\cap$ {s | $\exists s \rightarrow t$, t $\in$ Result}
  - **until** no change;
  - **return** Result;

A little more rewriting

- ***Technique***
  - Result := S *(i.e., set of all states in M);*
  - **repeat**
    - Result := Result $\cap$
    - {s |s $\in St_f \wedge \exists s \rightarrow t$, t $\in$ Result}
  - **until** no change;
  - **return** Result;

A little more rewriting

- ***Technique***
  - Result := S *(i.e., set of all states in M);*
  - **repeat**
    - Result := {s |s $\in St_f \wedge \exists s \rightarrow t$, t $\in$ Result}
  - **until** no change;
  - **return** Result;

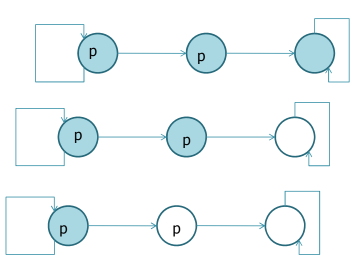
A little more rewriting

- ***Technique***
  - Result := S *(i.e., set of all states in M);*
  - **repeat**
    - Result := $St_f \cap$ {s | $\exists s \rightarrow t$, t $\in$ Result}
  - **until** no change;
  - **return** Result;

Equivalent to the algorithm in the previous slide.

Example of fixed point constr.

Set of states satisfying EGp



Symbolic fixed-pt constr.

- The prev slide shows computing of St_{EGp} via fixed point construction.
  - Computation terminates when St_{EGp} does not change from one iteration to the next.
- In symbolic model checking
  - St_{EGp} represented as a ROBDD
  - Computation terminates when ROBDD for St_{EGp} does not change from one iteration to the next.

Symbolic computation

- Now suppose we represent all sets of states and sets of transitions as boolean function and hence ROBDD.
  - All set operations are boolean operations
- For representing sets of states
  - Input variables are AP, set of atomic props.
  - St_f (one of the inputs) is presented as a ROBDD
  - St_{EGf} (the output) is computed as another ROBDD.

Symbolically computing St_{EGf}

- Input
  - Kripke Structure M
    - Transition relation $\rightarrow$ represented as ROBDD
  - St_f represented as a ROBDD with AP (set of atomic props.) as input variables
    - To emphasize this representation, call it bdd_f
- Output
  - St_{EGf} represented as a ROBDD with AP as input
    - To emphasize this representation, call it bdd_{EGf}

Symbolically computing St_{EGf}

- Technique
  - bdd_{EGf} (AP) := true;
  - Repeat{
  - bdd_{EGf} (AP) := bdd_f (AP) $\wedge$ $bdd_{\{s \mid \exists s \rightarrow t \text{ s.t. } t \in bdd_{EGf}\}}$
  - Until no change ;
  - Return bdd_{EGf} (AP)
- How to compute $bdd_{\{s \mid \exists s \rightarrow t \text{ s.t. } t \in bdd_{EGf}\}}$
- $\wedge$ will be achieved by the Apply algorithm of BDDs

Symbolically computing St_{EGf}

- St_{EGf} is a subset of states of the Kripke Structure M being verified.
  - Represented as ROBDD.
  - Iteratively updated via boolean operations
  - Need to be reduced after each iteration to ensure we always work with a ROBDD
  - Iteration stopped when fixed point reached
    - Easily detected since ROBDD is a normal form for a boolean function.

Symbolically computing St_{EGf}

- Start with St_{EGf} = set of all states
  - Set of all states trivially represented as a single node BDD, no state space explosion
- In each iteration update
  - $St_{EGf} = bdd_f \wedge EX\ St_{EGf}$
  - bdd_f is the BDD representation of St_f
  - How to compute the BDD for EX St_{EGf}, i.e.

Symbolically computing St_{EGf}

- How to compute $bdd_{\{s \mid \exists s \rightarrow t \text{ s.t. } t \in bdd_{EGf}\}}$
  - bdd_{EGf} represents the current approximation of St_{EGf}
    - Call this current approximation as Υ
  - We need to compute the set of states satisfying EX Υ, and represent it as ROBDD
    - This should be achieved without converting BDDs to sets of states and then back to BDDs

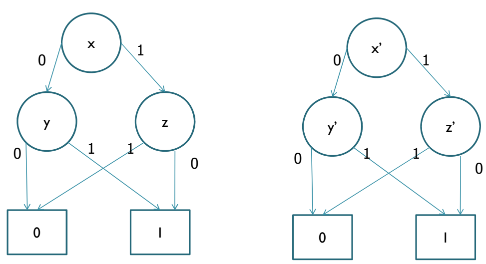
The EX operator

- Given the BDD for a set of states $\Upsilon \subseteq S$, compute the BDD for the set of states satisfying (EX Υ)
- Sets of states represented as boolean functions
  - Input variables are AP (set of atomic props.)
  - So BDD representation of $\Upsilon \subseteq S$ captures a boolean function $F_\Upsilon(AP)$
- Transition relation also represented as boolean func.
  - Input variables are AP, AP'
  - BDD representation of transition relation $\rightarrow$ captures a boolean func. $F_\rightarrow(AP, AP')$

The EX operator

- BDD for the set of states satisfying (EX Υ) then captures the boolean function
  - $F_{EX}(AP) = \exists AP' \; (\; F_\Upsilon(AP') \; \wedge \; F_\rightarrow(AP, AP') \;)$
- If we have the BDD for $F_\Upsilon(AP')$ and $F_\rightarrow(AP, AP')$
  - $\mathbf{bdd_{EX}(AP) \; = \exists AP' \; (\; bdd_\Upsilon(AP') \; \wedge \; bdd_\rightarrow (AP, AP') \;)}$
    - $\wedge$ is computed by Apply
    - How to compute $bdd_\Upsilon(AP')$ from $bdd_\Upsilon(AP)$?
    - How to compute existential quantification ?

Primed BDD – trivial !



QBF – Easy !

- Quantified Boolean Formula
  - $\exists x$ F is same as $F|_x \vee F|_{\neg x}$
  - $\forall x$ F is same as $F|_x \wedge F|_{\neg x}$
  - $\exists x,y$ F is same as $F|_{x,y} \vee F|_{x,\neg y} \vee F|_{\neg x,y} \vee F|_{\neg x,\neg y}$
    - Similarly define $\forall x,y$ F
  -
  - Can use the above def to define $\exists \overline{X}$ F
    -
    - Similarly define $\forall \overline{X}$ F

Computing BDD for St_{EGf}

- **Input**: M=(S,S0,$\rightarrow$,L) and bdd_f capturing St_f
- **Output:** bdd_{EGf} capturing St_{EGf}
- **Technique**
  - bdd_{EGf} := single-node BDD for "true";
  - Repeat{
    - bdd1(AP) := $\exists$AP' ($bdd_\rightarrow$ (AP,AP') $\wedge$ bdd_{EGf} (AP'))
    - bdd_{EGf} (AP):= bdd_f(AP) $\wedge$ bdd1(AP);
  - } until no change in bdd_{EGf}
  - Return bdd_{EGf}

Transition graph is not traversed explicitly.

Computation proceeds by BDD manipulation.

Comparison

- *Explicit-state*
- St_{EGf} := S
- do{
- S1 := {s| $\exists$ s $\rightarrow$ s1 s.t.
- s1 $\in$ St_{EGf}}
- St_{EGf} := $St_f \cap$ S1 ;
- }until no change in St_{EGf}
- Return St_{EGf}

- *Symbolic*
- bdd_{EGf} (AP) := true
- do{
- bdd1(AP) := $\exists$AP' ($bdd_\rightarrow$ (AP,AP')
- $\wedge bdd_{EGf}$ (AP'))
- bdd_{EGf} (AP):= bdd_f(AP)$\wedge$
- bdd1(AP);
- } until no change in bdd_{EGf}
- Return bdd_{EGf}

Symbolic FP computation via BDDs

- Since the symbolic FPs compute boolean functions, they are compactly represented as BDDs.
- The logical operations can be done over BDDs using **Apply** algorithm.
- Fixed point detection can proceed due to canonical nature of ROBDD for fixed variable order.
- To compute the set of states satisfying a CTL property as a fixed point construction, the property needs to have an iterative definition.

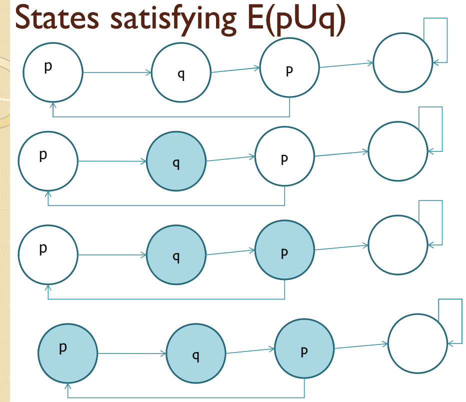
Fixed point characterization

- EG φ = $\varphi \wedge$ EX EG φ
- $E(\varphi \text{ U } \Psi) = \Psi \vee (\varphi \wedge \text{EX } E(\varphi \text{ U } \Psi))$
- Similar characterizations exist for AG, EF, AF …

Computing $St_{E(f \text{ U } g)}$

- Input: M = (S, S0, $\to$,L)
- St_f , St_g
- Output: $St_{E(f \text{ U } g)}$
- Technique:
  - $St_{E(f \text{ U} g)}$:= empty-set;
  - do{
  - St_{EX} := {s | $\exists$ s' such that s $\to$ s' and s' $\in St_{E(f \text{ U } g)}$ }
  - $St_{E(f \text{ U } g)} := St_g \cup (St_f \cap St_{EX})$;
  - } until no change in $St_{E(f \text{ U } g)}$;
  - Return $St_{E(f \text{ U } g)}$

States satisfying E(pUq)



Computing BDD for $St_{E(f \text{ U } g)}$

- **Input**: M=(S,S0,$\to$,L), bdd_f/bdd_g capturing St_f/St_g
- **Output:** $bdd_{E(fUg)}$ capturing $St_{E(fUg)}$
- **Technique**
  - $bdd_{E(f \text{ U } g)}$:= single-node BDD for "false";
  - Repeat{
    - bdd1(AP) := $\exists$AP' ($bdd_{\to}$ (AP,AP') $\wedge bdd_{E(fUg)}$ (AP'))
    - $bdd_{E(f \text{ ug})}$ (AP):= bdd_g(AP) $\vee$ (bdd_f(AP) $\wedge$ bdd1(AP)) ;
  - } until no change in $bdd_{E(f \text{ U } g)}$
  - Return $bdd_{E(f \text{ U } g)}$

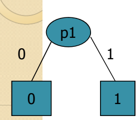
Transition graph is not traversed explicitly.

Computation proceeds by BDD manipulation.

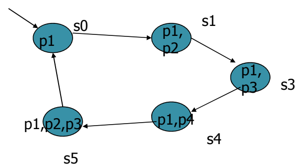
Symbolic MC for CTL

- We have given techniques for computing BDD for set of states satisfying E(f U g) and EGf.
- How to compute BDD for set of states satisfying
  - Atomic props.
  - $\neg$ f
  - f $\wedge$g
  - EX f
- Case 1 :Atomic prop. p1 $\in$ AP={p1,p2,…,pm}
  - Captured by F_{p1}(p1,…,pm) = p1

Symbolic MC for CTL



BDD for set of states satisfying p1



Explicit representation for set of states satisfying p1

{s0,s1,s2,s3,s4,s5}

So, how come the BDD appears to be independent of the model being verified?
-- The property did not involve lookup of the transition relation

Symbolic MC for CTL

- Case 2: Given bdd_f (BDD for St_f) how to compute $bdd_{\neg f}$ (BDD for $St_{\neg f}$)
  - Swap the leaves of bdd_f
- Case 3 : Given bdd_f , bdd_g (BDD for St_f , St_g) how to compute $bdd_{f\wedge g}$ (BDD for $St_{f\wedge g}$)
  - Algorithm Apply
- Case 4: Given bdd_f, how to compute bdd_{EXf}
  - $bdd_{EXf}(Vars) = \exists Vars'\ (bdd_{\rightarrow}(Vars,Vars') \wedge bdd_f(Vars'))$
  - Vars is the set of variables used to define sets of states
    - For our purposes Vars = AP (set of atomic propositions)
  - Vars' is another set of variables used to describe the destination states in a transition.

Computing Relational Products

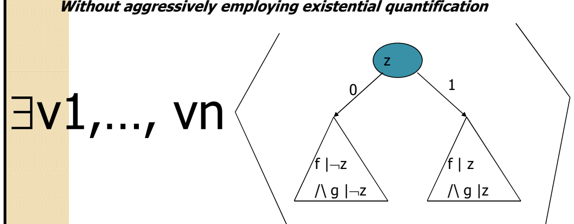
- $\exists Vars'\ (bdd_{\rightarrow}(Vars,Vars') \wedge bdd_f(Vars'))$
  - This is simply a boolean function.
  - Given the BDD for transition relation $\rightarrow$ and the BDD for St_f, we can compute the above BDD using Apply algorithm.
  - Just compute the conjunction $bdd_{\rightarrow}(Vars,Vars') \wedge bdd_f(Vars')$ and then employ existential quantification
  - But the conjunction itself can blow up in the numbers of variables, leading to a possibly large BDD
  - Furthermore, many of these variables will be quantified away anyway, so
- Can we employ the quantification aggressively ?

Computing Relational Products

- $\exists Vars'\ (bdd_{\rightarrow}(Vars,Vars') \wedge bdd_f(Vars'))$
  - Break the conjunction computation into sub-problems.
  - Employ existential quantification on the sub-problem results.
  - Quantification not postponed until the end
    - Substantial space savings in practice.
    - Worst-case space complexity remains same.

Showing the RelProd algorithm

Without aggressively employing existential quantification

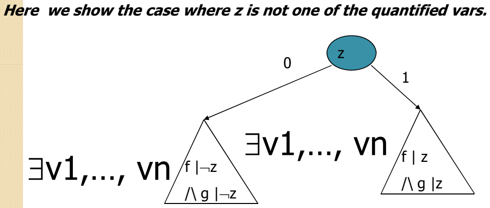


z is the top variable in the variable order from among the vars. in f, g.

Showing the RelProd algorithm

Aggressively employing existential quantification.

Here we show the case where z is not one of the quantified vars.



z is the top variable in the variable order from among the vars. in f, g.

The RelProd Algorithm

- Computes $\exists V(F \wedge G)$
  - V is subset of vars in F, G
- Computes by divide-and-conquer
  - Results cache remembers past answers
- Let z be the "top of the tops" in F, G
  - If $z \in V$
    - z does not appear in the returned bdd
    - Existential quantification at the earliest
  - If $z \notin V$
    - Test on z is introduced, and bdd returned.

The RelProd Algorithm

```
function RelProd( F, G: BDD,  V: variables):BDD
{
   if  (F ==0 ∨ G == 0) return 0;
   else if (F ==1 ∧ G == 1)  return 1;
   else if (F,G,V,bdd) ∈ Results  return bdd;
   else{
          let x, y be the top var. of F, G respectively;
          z = x > y? x: y;
          bdd1 =  RelProd( F|¬z, G|¬z, V);
          bdd2 = RelProd(F|z, G|z, V);
          if  (z ∈ V){
                      bdd = bdd1 ∨bdd2;
          } else{  bdd = (¬z∧  bdd1 )  ∨ (z ∧ bdd2);
          }
          Results = Results ∪  (F,G,V,bdd);
          return bdd;
     }
}
```


Summary

- Kripke Structure represented as BDD
  - BDD directly generated from SMV code.
- Sets of states satisfying any CTL formula always represented as BDDs.
- Fixed point computation over sets becomes fixed point computation over boolean functions i.e. ROBDD
- Application of these predicate transformers involve set operations
  - Achieved by logical operations
  - Algorithm Apply discussed earlier can implement all the 2^(2^2) = 16 possible binary logical ops.

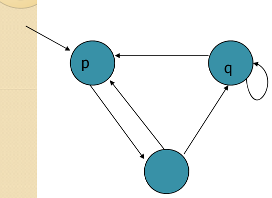
Exercises - I

- Try to write symbolic versions of the customized algorithms for AG, EF, AF that we worked out in an earlier lecture.
- We had re-written the explicit-state algo. for checking EGf before constructing the symbolic version.
  - Try to construct a symbolic version of the original algo. (reproduced in the next slide).

Exercises - I

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

Exercises - II



Employ our symbolic model checking method using ROBDDs to find the states satisfying

-- AG(p ∨ (¬q))

-- E(q U p)