

CS6910: Testing/Verification of Concurrent Programs

Symbolic Reachability Computation

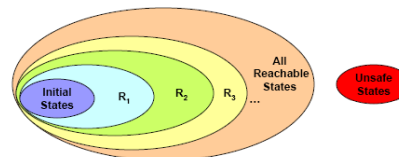
Implicit Representation

- Implicit representation of transition relation and reachable states
- For propositional or enumerated modules, **binary decision diagrams (BDDs)** serve as a compact representation
 - BDDs: representation for boolean functions due to Bryant
 - Application to model checking: tool SMV by McMillan
 - Popular in hardware applications

Symbolic Data Types

- Instead of enumerating states, compute with regions (state-sets) that are represented symbolically
 - Example: $20 \leq x \leq 99$
- Note: No enumeration, so number of states in a region is not an issue

Symbolic Search



- LIKE breadth-first search, guaranteed to terminate if the diameter of the graph is bounded: there is i such that every reachable state is reachable within i transitions from some initial state

Search Algorithm

Input: a transition graph G , and a region σ^r
Output: answer to reachability problem (G, σ^r)

```

Input G: symgraph;  $\sigma^r$ : symreg
Local  $\sigma^R$ : symreg;
 $\sigma^R := \text{InitReg}(G)$ ;
repeat
  if  $\sigma^R \cap \sigma^r \neq \text{EmptySet}$  then
    return Yes;
  if  $\text{PostReg}(\sigma^R, G) \subseteq \sigma^R$  then //or  $\text{PostReg}(\sigma^R, G) \cup \sigma^R = \sigma^R$ 
    return No;
   $\sigma^R := \sigma^R \cup \text{PostReg}(\sigma^R, G)$ 
forever
```

Symbolic Data Types

- Symbolic regions support following operations
 - \cup : $\text{symreg} \times \text{symreg} \rightarrow \text{symreg}$
 - \cap : $\text{symreg} \times \text{symreg} \rightarrow \text{symreg}$
 - $=$: $\text{symreg} \times \text{symreg} \rightarrow \text{bool}$
 - \subseteq : $\text{symreg} \times \text{symreg} \rightarrow \text{bool}$
 - EmptySet**: symreg
- Symbolic transition graph supports
 - InitReg**: $\text{symgraph} \rightarrow \text{symreg}$
 - PostReg**: $\text{symreg} \times \text{symgraph} \rightarrow \text{symreg}$

Symbolic Representation

- Symbolic representation of a transition graph
 - Type of variable x : T_x
 - Type of state over X : product type $T_X (\prod_{x \in X} T_x)$
 - Type of transition: $T_{X \cup X'}$
 - Transition relation is, thus, a symbolic region over primed and unprimed variables
 - initial region $\{\sigma^I\}_s$ of type $\text{symreg}[T_X]$
 - transition relation $\{\rightarrow\}_s$ of type $\text{symreg}[T_{X \cup X'}]$
- We need to decide on how to represent regions and compute **PostReg**

Computing Post

- Renaming
 - **Rename**(x, y, σ) returns the renamed region $\sigma[x := y]$
 - Extends to variable-sets **Rename**(X, Y, σ)
- Existential quantifier elimination
 - For σ of type $\text{symreg}[T_X]$, **Exists**(x, σ) returns the region $\{s \in \Sigma_{X \setminus \{x\}} \mid (\exists m. s[x := m] \in \sigma)\}$
 - Extends to variable sets **Exists**(X, σ)

Computing Post

- Computing **PostReg**(σ):
 - conjunct σ with $\{\rightarrow\}_s$ to obtain the set of transitions originating in σ
 - project the result onto the set X'
 - rename each primed variable x' to x
- **PostReg**($\sigma, \{G\}_s$) = **Rename**($X', X, \text{Exists}(X, \sigma \cap \{\rightarrow\}_s)$)

Summary

- To implement symbolic reachability, we need
 - implementation the data type **symreg** that supports $\cup, \cap, =, \subseteq, \text{Rename}$, and **Exists**
 - an efficient way to compute, from the module text, the symbolic representation of the initial region $\{\sigma^I\}_s$ and the transition region $\{G\}_s$

Symbolic Search Using Propositional Formulas

- Represent regions by boolean expression
 - Obtaining symbolic representations of σ^I and \rightarrow is easy (no blow-up)
 - Union, intersection easy
 - Renaming: textual substitution
 - Quantifier elimination in linear time:
 - **Exists**(x, p) = $(p[x := \text{true}] \vee p[x := \text{false}])$

Symbolic Search Using Propositional Formulas

- Inclusion (\subseteq) or equivalence ($=$)
 - checking validity of propositional formulas
 - hard: coNP-complete

Symbolic Reachability for Pete

- Initial region $q1^I$: $pc1 = out$
- Symbolic rep of transition relation $q1^T$
 - $(pc1=out \wedge pc1' = req \wedge x1' = x2) \vee$
 - $(pc1=req \wedge (pc2=out \vee x1'=x2) \wedge pc1'=in \wedge x1'=x1) \vee$
 - $(pc1=in \wedge pc1'=out \wedge x1' = x1) \vee$
 - $(pc1=pc1 \wedge x1' = x1)$
- $PostReg(q1^I)$
 - $Rename(X',X,Exists(X, q1^I \cap q1^T))$
 - Simplifies to $pc1=out \vee pc1 = req$

Can We Improve Formula Representation?

- Propositional formulas not very satisfactory
 - Each step of the computation may be expensive (equality test)
 - More importantly, size of $PostReg^{<=i}(q^I)$ may grow with i , with no good heuristics for simplification
- Desirable properties of representation:
 - All operations should be efficient
 - Should maintain compactness whenever possible
 - Representing q^I and q^T from module text should be efficient

Representations of Boolean Functions?

- We often reduce problems to the manipulation of Boolean Functions
- Truth Tables
- Disjunctive Normal Form
 - Sum-of-Products
- Conjunctive Normal Form
 - Product-of-Sums

Truth Table

a	b	c	f(a,b,c)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

DNF and CNF

- $f = abc + ab'c + a'bc$
- $f = (a + b + c) (a + b + c') (a + b' + c) (a' + b + c) (a' + b' + c)$
- $f = (a+b) c$

Review

- How about DNF?
 - $F = ab$
 - $F = abc + abc'$
 - $F = ab' + bc' + ca'$
 - $F = a'b + b'c + c'a$

Ordered Binary Decision Graphs

- Representation for predicates over a set X of boolean variables
- The variables in X are totally ordered
- Paths in the graph encode assignments to variables in X
 - Terminal vertices classify paths into accepting and rejecting

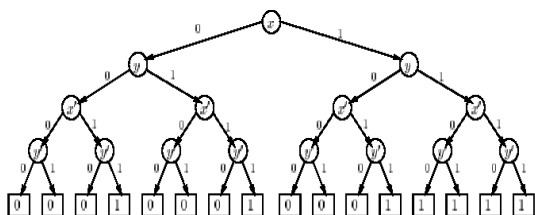
Definition of BDG

- A finite set V of vertices
 - Internal vertices V^I and terminal vertices V^T
 - Root: a root vertex v^i in V
- Labeling: a labeling function that labels each internal vertex with a variable in X , and each terminal vertex with a constant 0 or 1
- Left edges: a function $left : V^I \rightarrow V$
 - If $left(v)$ is an internal vertex then $label(v) < label(left(v))$
 - Right edges similar

Sample BDG

Function: $(x \wedge y) \vee (x' \wedge y')$

- Ordering: $x < y < x' < y'$
- On every path from root to a terminal vertex, a variable appears 0 or 1 times



Boolean Function of a BDG

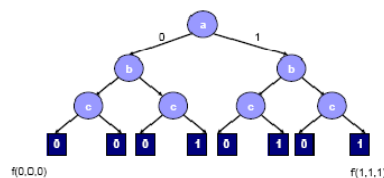
- With vertex v , associate a function $r(v)$
 - if v is terminal then $r(v)$ equals its label (0 or 1)
 - if v is internal then $r(v)$ equals $(\neg label(v) \wedge r(left(v))) \vee (label(v) \wedge r(right(v)))$
 - BDG B represents the function $r(B) = r(v^i)$
- To check whether a state s satisfies $r(B)$ simply follow path according to values assigned by s

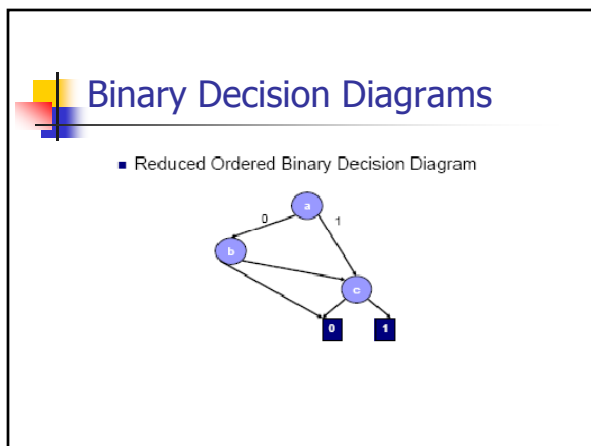
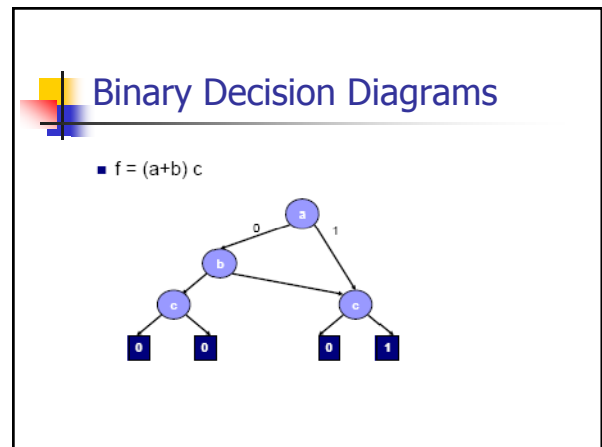
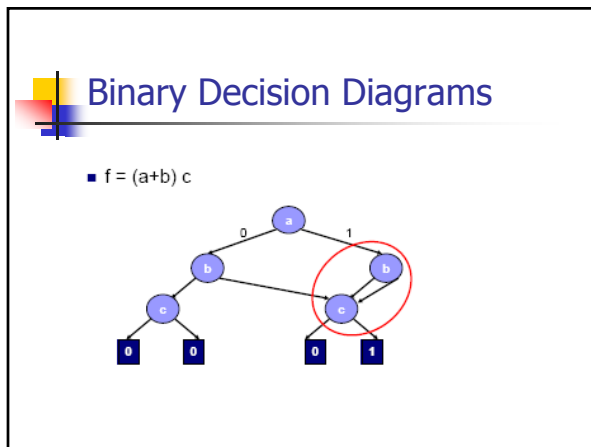
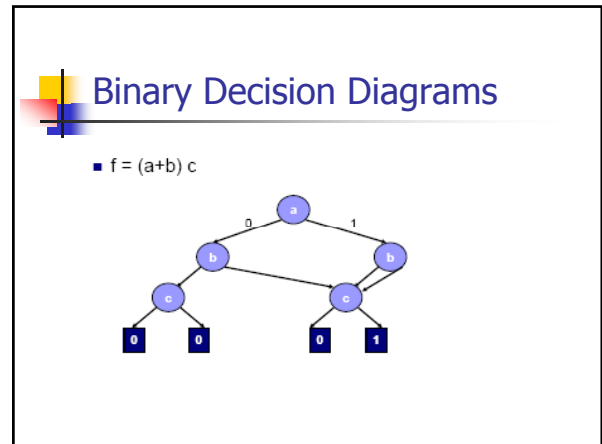
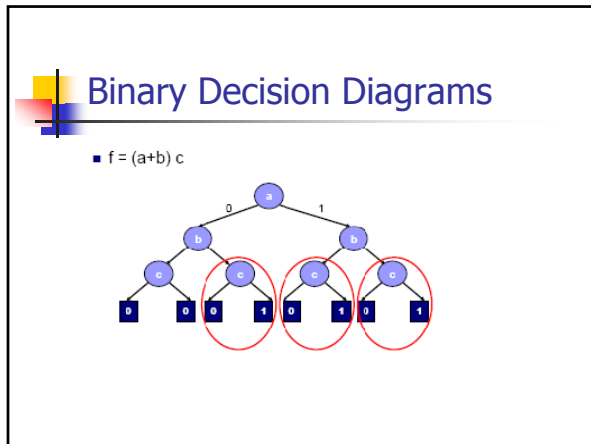
Isomorphism and Equivalence

- Two BDGs B and C are isomorphic if the corresponding labeled graphs are isomorphic
 - Deciding isomorphism is easy
- Two BDGs B and C are equivalent if the boolean expressions $r(B)$ and $r(C)$ are equivalent.
 - Equivalence does not imply isomorphism
 - Deciding equivalence is difficult (so this is not what we want for symbolic reachability)

Binary Decision Tree

$f = (a+b) \cdot c$





Reduction Rules

- If the two successors of a node v are identical, then remove v .

- If two nodes represent the same function, remove one and share other.



ROBDD

- **Reduced**
 - No more reduction rules can be applied
 - Represent each sub-function only once
 - Remove nodes where both successors are identical
- **Ordered**
 - Same variable ordering on all paths
 - Size of BDD depends on variable ordering