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 <= x <= 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 o'
- Output answer to reachability problem (G, o')
- Input G: symgraph; o': symreg
- Local o": symreg;
- o": = InitReg(G);
- repeat
  - if o" ≠ EmptySet then
  - return Yes;
  - if PostReg(o", G)∩o" then // o" = o" ∪ PostReg(o", G)
  - return No;
- end
- forever

Symbolic Data Types
- Symbolic regions support following operations
  - ∪: symreg × symreg → symreg
  - ∩: symreg × symreg → symreg
  - =: symreg × symreg → bool
  - EmptySet: symreg
- Symbolic transition graph supports
  - InitReg : symgraph → symreg
  - PostReg : symreg × symgraph → symreg
Symbolic Representation

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

Computing Post

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

Symbolic Search Using Propositional Formulas

- Represent regions by boolean expression
  - Obtaining symbolic representations of \( \sigma \) and \( \rightarrow \) is easy (no blow-up)
  - Union, intersection easy
  - Renaming: textual substitution
  - Quantifier elimination in linear time:
    - \( Exists(x, p) = (p[x := true] \lor p[x := false]) \)

Summary

- To implement symbolic reachability, we need
  - implementation the data type symreg that supports \( \cup, \cap, =, \subseteq \), Rename, and Exists
  - an efficient way to compute, from the module text, the symbolic representation of the initial region \( \{ \sigma \} \), and the transition relation \( \{ \rightarrow \} \)

Symbolic Search Using Propositional Formulas

- Inclusion \( \subseteq \) or equivalence \( = \)
  - checking validity of propositional formulas
  - hard: coNP-complete
Symbolic Reachability for Pete

- Initial region q1I: pc1 = out
- Symbolic rep of transition relation q1T:
  
  \[(pc1 = \text{out} \land pc1' = \text{req} \land x1' = x2) \lor
  (pc1 = \text{req} \land pc2 = \text{out} \land x1' = x2) \lor
  (pc1' = \text{in} \land pc1 = \text{out} \land x1' = x1) \lor
  (pc1' = pc1 \land x1' = x1)\]

- PostReg(q1I):
  
  Rename(X', X, Exists(X, q1I \cap q1T))

  Simplifies to pc1 = out \lor 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(qi) 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 qi and q\textsuperscript{\textprime}i 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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>f(a,b,c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

DNF and CNF

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

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 \(v^\circ\) 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 \(\text{left} : V^i \rightarrow V\)
    - If \(\text{left}(v)\) is an internal vertex then \(\text{label}(v) < \text{label(\text{left}(v)})\)
    - Right edges similar

Sample BDG

Function: \((x \land y) \lor (x' \land y')\)
- Ordering: \(x \land x' < x' < y < y'\)
- 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 \((\neg \text{label}(v) \land r(\text{left}(v))) \lor (\text{label}(v) \land r(\text{right}(v)))\)
- BDG \(B\) represents the function \(r(B) = r(v^\circ)\)
- 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

- \(f = (x \lor y) \land (z \lor w)\)
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