Formal verification. Lecture 3
Marius Minea

Fixpoint representations

Def: \( x \in D \) is a fixpoint for \( f : D \to D \) if \( f(x) = x \).

Def: A lattice is a partially ordered set in which any finite subset has a least upper bound and a greatest lower bound.

Examples: powerset (set of subsets) \( P(S) \) of a set \( S \), with \( \subseteq \) as order

We work with functions \( \tau : P(S) \to P(S) \) over the lattice \( P(S) \).

We regard \( S' \subseteq S \) as a predicate over \( S \): \( S'(s) = \text{true} \) \( \Rightarrow \) \( s \in S' \).

In particular: \( \emptyset = \text{false} \), \( S = \text{true} \).

\( \Rightarrow \) \( \tau : P(S) \to P(S) \) is a predicate transformer.

Def: \( \tau \)

- is monotone if \( P' \subseteq Q \Rightarrow \tau(P') \subseteq \tau(Q) \)
- is union-continuous if for any sequence \( P_1 \subseteq P_2 \subseteq \ldots \) we have \( \tau(\bigcup_i P_i) = \bigcup_i \tau(P_i) \)
- is intersection-continuous if for any sequence \( P_1 \supseteq P_2 \supseteq \ldots \) we have \( \tau(\bigcap_i P_i) = \bigcap_i \tau(P_i) \)

Fixpoint theorems

A monotone predicate transformer over \( P(S) \) always has

- a minimal fixpoint, denoted \( \mu \tau(s) \)
- and a maximal fixpoint, denoted \( \nu \tau(s) \) [Tarski]

If \( S \) is finite and \( \tau \) is monotone, then \( \mu \tau \) exists and is continuous.

\( \Rightarrow \) \( \mu \tau(\text{false}) \subseteq \nu \tau(\text{false}) \) \( \Rightarrow \) \( \mu \tau(\text{true}) \supseteq \nu \tau(\text{true}) \) [Tarski]

If \( \tau \) is monotone and \( S \) is finite, then there exist \( i, j \geq 0 \) such that

\[ \forall k \geq i, s^k(\text{False}) = \tau^{k+1}(\text{false}) \] \( \Rightarrow \) \( \tau^{i+j}(\text{true}) = s^i(\text{false}) \) and \( \nu \tau(s) \).

If \( \tau \) is monotone and \( S \) is finite, then there exist \( i, j \geq 0 \) such that

\[ \mu \tau(s) = s^i(\text{false}) \] and \( \nu \tau(s) = s^i(\text{true}) \) for some \( i, j \geq 0 \).
Formal verification. Lecture 3

Marius Minea

Computing the minimal/maximal fixpoint

function Lfp(∀ x . Trans) : Pred
function Gfp(∀ x . Trans) : Pred

Q := False;
Q' := r(Q);
while (Q' ≠ Q) do
Q := Q';
Q' := r(Q);
return Q;

Fixpoint relations for CTL

We identify a CTL formula \( f \) with the set of states that satisfy it:
\[
\{ s \mid \exists M, s \models f \}
\]
• AF \( f \) = \( \mu Z. f \lor AX Z \)

• AG \( f \) = \( \nu Z. f \land AX Z \)

• E[1f U f] = \( \mu Z. f \lor (f \land \nu E X Z) \)

• A[1f U f] = \( \nu Z. f \land \nu f \lor (f \land AX Z) \)

• E[1f R f] = \( \mu Z. f \lor (f \land E X Z) \)

• A[1f R f] = \( \nu Z. f \land (f \land EX Z) \)

minimal fixpoint: liveness properties: \( F \)
maximal fixpoint: safety properties (invariants): \( G \)

Representations for Boolean functions

\( f : B^n \to B \) can encode both state sets and transition relations

• Usual representations (truth tables, Karnaugh diagrams, canonical sum of minterms) have exponential size

• Improvements: reduced sums of products, factorizations, etc.

• A canonical and compact representation of Boolean functions

• Efficient manipulation

• Significant impact on formal verification:

ACM Kanellakis Award for Theory & Practice, 1998
– Randal E. Bryant
– Edmund M. Clarke, E. Allen Emerson: model checking ('81)
– Ken McMillan: symbolic model checking ('92)

Binary Decision Diagrams (BDDs)

Symbolic model checking. Binary decision diagrams

Symbolic model checking. Binary decision diagrams

Symbolic model checking. Binary decision diagrams
Reduction rule 1: Merge terminal nodes

\[
\begin{array}{cccc}
2 & 2 & 2 & 2 \\
5 & 5 & 5 & 5 \\
0 & 0 & 1 & 0 \\
\end{array} \quad \Rightarrow \quad 
\begin{array}{cccc}
2 & 2 & 2 & 2 \\
0 & 0 & 1 & 0 \\
\end{array}
\]

Reduction rule 2: Merge isomorphic nodes

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
0 & 0 & 1 & 1 \\
\end{array} \quad \Rightarrow \quad 
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
0 & 0 & 1 & 1 \\
\end{array}
\]

Reduction rule 3: Eliminate redundant test

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
0 & 0 & 1 & 1 \\
\end{array} \quad \Rightarrow \quad 
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
0 & 1 & 1 & 1 \\
\end{array}
\]

The 3 rules can be applied whatever the variable ordering down the tree. In an ordered BDD (OBDD), one additional condition: On all paths from root to terminals, variables appear in same order (there exists a global ordering of variables) ⇒ canonical representation ⇒ equivalence or satisfiability checking in constant time

Note: A subgraph rooted as a BDD node is also a BDD ⇒ BDDs for several functions may share subgraphs in the same forest

Consider the function: \((a_1 \land b_1) \lor (a_2 \land b_2) \lor (a_3 \land b_3)\)

Linear growth: \(2(n + 1)\)  
Exponential growth: \(2^{n+1}\)

BDD algorithms: Apply

\[
\begin{align*}
\text{function } & \text{Apply}(f, g, \text{op}, \text{BDD}) : \text{Operator} \rightarrow \text{BDD} \\
& \text{if } \text{isLeaf}(f) \land \text{isLeaf}(g) \rightarrow \text{return} \text{op}(f, g); \\
& \text{elsif} (f, g, \text{op}, h) \text{ in applyCache return } h; \\
& \text{else} \\
& \quad x := \text{topVar}(f) // \text{variable at root of } f \\
& \quad y := \text{topVar}(g) \\
& \quad \text{if } (\text{ord}(x) = \text{ord}(y)) \rightarrow \text{if } x = y \text{ same variable} \\
& \quad \quad h := \text{findBDD}(x, \text{Apply}(f \upharpoonright x=0, g \upharpoonright x=0, \text{op})); \\
& \quad \quad \text{// findBDD creates a new BDD if not already existent} \\
& \quad \quad \text{elsif} (\text{ord}(x) < \text{ord}(y)) \rightarrow \text{if } x \text{ before } y \text{ in ordering} \\
& \quad \quad \quad h := \text{findBDD}(x, \text{Apply}(f \upharpoonright x=0, g \upharpoonright x=0, \text{op})); \\
& \quad \quad \quad \text{// findBDD creates a new BDD if not already existent} \\
& \quad \quad \text{else} h := \text{findBDD}(y, \text{Apply}(f \upharpoonright y=0, g \upharpoonright y=0, \text{op})); \\
& \quad \quad \text{insert } (f, g, \text{op}, h) \text{ in applyCache} \\
& \quad \quad \text{return } h
\end{align*}
\]
function Relprod(f, g : OBDD, E : varset) : OBDD
if f = false ∨ g = false return false
else if f = true ∧ g = true return true
else if (f, g, E, h) in relprod_cache return h
else
    x := tovar(f) // variable at root of f
    y := tovar(g)
    z := topmost(z, x) // first in variable order
    h₀ := RelProof(f |z=x, g |z=x, E)
    h₁ := RelProof(f |z=x, g |z=x, E)
    if x ∈ E : h := Relprod(h₀, h₁) // */ 32 : h = h₀ ∧ h₁ */
else h := bdd_if_then_else(x, h₀, h₁)
insert ((f, g, E, h) in relprod_cache
return h

BDD algorithms: relational product

Complexity of BDD algorithms

• Reduction (to canonical form) O(|G| · log(|G|))
• Apply (f₁ op f₂) O(|G₁| · |G₂|)
• Restrict (f |x=x₁) O(|G| · log(|G|))
• Compose (f₁ |x=x₁) O(|G₁|² · |G₂|)
• Satisfy-one (u ∨ v | f(u) = 1) O(n)
• Satisfy-count (u | f(u) = 1)) O(|G|)

Logarithmic factors can be eliminated (by more sophisticated algorithms or hashing)

Relational product may have exponential complexity

Implementation

• There are mature BDD libraries (packages) (CMU, Cal, CUDD)
• In a typical application, many BDDs have common subgraphs ⇒ pointers into a graph with unique root
• Memory management: reference counter and garbage collection
• Many optimizations and heuristics – memory layout and traversal for efficient caching – parallel and distributed algorithms, etc.

Symbolic model checking. Binary decision diagrams

Formal verification. Lecture 3
Marius Minea

Symbolic model checking. Binary decision diagrams

Dynamic variable reordering

• Variable ordering is critical for BDD size
• Functions exist with exponential size BDDs regardless of ordering (e.g., middle bit of a multiplier [Bryant’91])
• Shape and size of BDDs evolves during computation ⇒ dynamic variable reordering is important
  – transparent for verification algorithms constructed on top
  – reordering adjacent levels does not change pointers into BDD

BDD variants and applications

• choice of other decompositions for Boolean functions:
  – OBDD: Boole-Shannon decomposition f = f₀ ∧ f₁ ∧ ... ∧ fₙ = f₀ ∨ f₁ ∨ ...
  – f₀ = f₀ ∨ f₁ ∧ fₙ
  – f₁ = f₁ ∨ f₀ ∧ fₙ
  – fₙ = fₙ ∨ f₀ ∧ f₁
  – Reed-Muller decomposition
  – positive Davio decomposition
• Multiterminal BDDs: allow arbitrary terminal nodes (typically integers)
• BDDs for arithmetic representations: f = x₀ + 2 * x₁ + 4 * x₂ + ...

Applications

• Mainly: CAD (equivalence checking) and formal verification
• Compact representations for data with some regularities/repetitions, but difficult to express analytically:
  – coding theory, large data structures, indexing, computational biology

Symbolic model checking with BDDs

System represented as binary encoding for states and atomic propositions ⇒ use BDDs for state sets, transition relation

Check(x) = \{ s ∈ S | p ∈ L(x)\}  bdd if_then_else(p, 1, 0)
Check(-f) = \{ \check{f} \}  bdd not
Check(f ∨ g) = Check(f) ∩ Check(g)  bdd and
CheckEX(f) = CheckEX(Check(f))  
CheckEX(f)(δ) = δ → [f(δ) ∨ R(δ, δ')]  RelProof(f, R, δ')
Check(E[f U g]) = CheckEU(Check(f), Check(g))
E[[f U g]] = μδ. f₂ ∨ (f₁ ∧ EX δ)  algorithm Lfp
EG[f] = CheckEG(Check(f))
EG[f] = μδ. f ∨ δ  algorithm Gfp

Formal verification. Lecture 3
Marius Minea

Formal verification. Lecture 3
Marius Minea

Formal verification. Lecture 3
Marius Minea

Formal verification. Lecture 3
Marius Minea
Monolithic transition relation – grows – can become major obstacle in building system model to fit in memory

- *disjunctive* partitioning (asynchronous systems)

  \[ R(E, v_f') = R_1(E, v_f') \lor \cdots \lor R_k(E, v_f') \]

  because of distributivity:
  \[ \exists f'[f(E') \land R(E, v_f')] = \exists f'[f(E') \land \cdots \land f(E') \land R(E, v_f')] \]

- *conjointive* partitioning (for synchronous systems)

  \[ \exists f'[f(E') \land R(E, v_f')] = \exists f'[f(E') \land \cdots \land f(E') \land R(E, v_f')] \]

  (does not distribute over \( \land \), but may exploit locality)

  \[ \exists f'[f(E') \land R(E, v_f')] = \exists f'[f(E') \land \cdots \land f(E') \land R(E, v_f')] \]

  (perform conjunction and quantification successively for each component)

Recall: fairness constraint is: \( F = \{ P_1, P_2, \ldots, P_k \} \), with \( P_i \subseteq S \)

\[ \text{EG} \, f \, \text{is true in the maximal set } Z \text{ such that:} \]

- all states of \( Z \) satisfy \( f \)

- \( \forall P_i \in F, s \in Z \) there is a path from \( s \) to a state of \( Z \cap P_k \)

  (passing only through states that satisfy \( f \))

\[ \Rightarrow \text{can be expressed as fixpoint and thus computed symbolically} \]

\[ \text{EG} \, \text{fair} \, f = \forall Z. \, f \land \bigwedge_{i=1}^{k} \text{EX} (f \cup (Z \cap P_i)) \]

Likewise for the other fundamental operators:

\[ \text{EX} \, \text{fair} \, f = \text{EX} (f \land \text{fair}) \]

\[ \text{EU} \, \text{fair} \, (f, g) = \text{EU} (f, g \land \text{fair}) \]

### Witness for EF \( f \)

- minimal fixpoint: \( \text{EF} \, f = \nu Z. \, f \lor \text{EX} \, Z \)

- compute and retain successive approximations \( f = Q_0 \subseteq Q_1 \subseteq \cdots \subseteq Q_k \)

  - \( Q_i \): set of states from which \( f \) can be reached in at most \( k \) steps

  - find intersection \( Q_i \cap S_0 \neq \emptyset \)

    (first traversal: backwards, symbolic)

    - choose \( s_2 \in S_0 \cap Q_k \)

      - compute set \( \text{Succ}(s_2) \) of successors for \( s_2 \)

      - must have nonempty intersection \( Q_{i-1} \) (from \( s_2 \) \( f \) is reachable in at most \( k \) steps, so there is a successor reaching it in \( k - 1 \) steps)

    - choose \( s_{k-1} \in \text{Succ}(s_k) \cap Q_{k-1} \), etc. until \( Q_0 = f \)

      (second traversal, forward, through individual states)

    - we have found path \( s_2 \rightarrow \ldots \rightarrow s_k \) reaching \( f \)