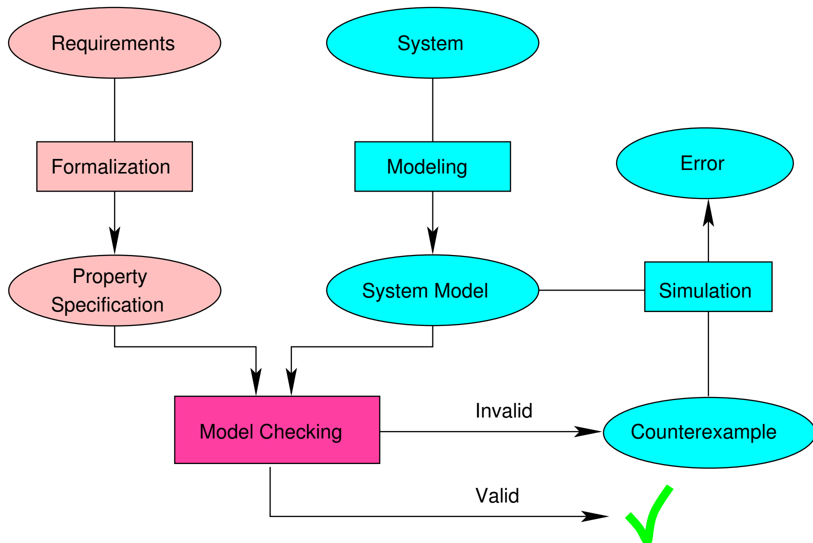
IA169 System Verification and Assurance

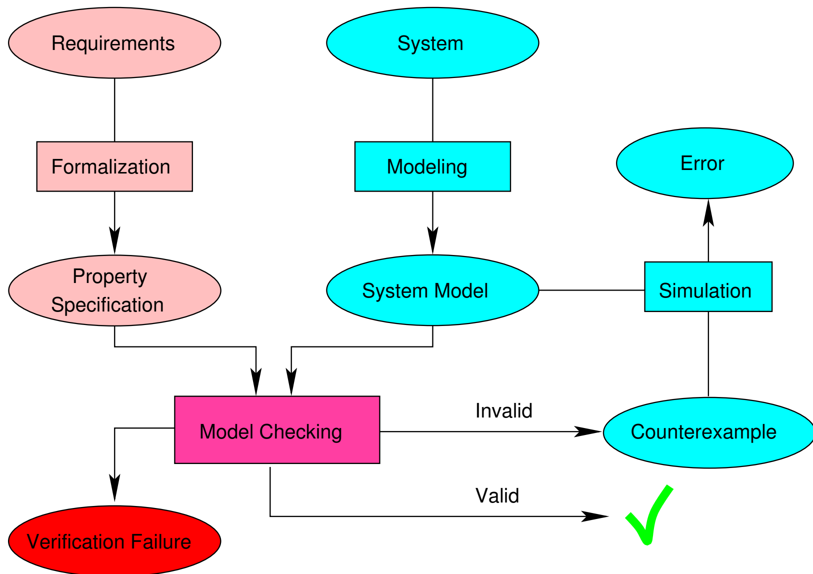
Symbolic Representations in CTL Model Checking

Jiří Barnat

State Space Explosion Problem and Model Checking



State Space Explosion Problem and Model Checking



Motivation

Observation

- Computation state is given by valuation of state variables.
- Every variable has a finite domain, its value may be stored using a fixed number of bits.
- Computation state represented as a bit vector $(a_1, \ldots, a_n)$ of fixed length n.

Set of States

- Algorithms for verification store set of states.
- Set of state can be viewed as a set of binary vectors.
- Set of binary vectors may be described with a **Boolean function**.

Boolean Functions

- These are formulae in propositional logic over a given set of Boolean variables.

Task

- Let system state be given by valuation of four bit variables $(a1, b1, a2, b2)$.
- A state is erroneous if the values of $a1$ and $b1$ and values of $a2$ and $b2$ agree.
- Describe a set of erroneous states with Boolean function.

Some Possible Solutions

Boolean Functions

- These are formulae in propositional logic over a given set of Boolean variables.

Task

- Let system state be given by valuation of four bit variables $(a1, b1, a2, b2)$.
- A state is erroneous if the values of $a1$ and $b1$ and values of $a2$ and $b2$ agree.
- Describe a set of erroneous states with Boolean function.

Some Possible Solutions

- $(a1 \wedge b1 \wedge a2 \wedge b2) \vee (a1 \wedge b1 \wedge \neg a2 \wedge \neg b2) \vee$ $(\neg a1 \wedge \neg b1 \wedge \neg a2 \wedge \neg b2) \vee (\neg a1 \wedge \neg b1 \wedge a2 \wedge b2)$
- $a1 \Leftrightarrow b1 \wedge a2 \Leftrightarrow b2$

Representation of Boolean Functions

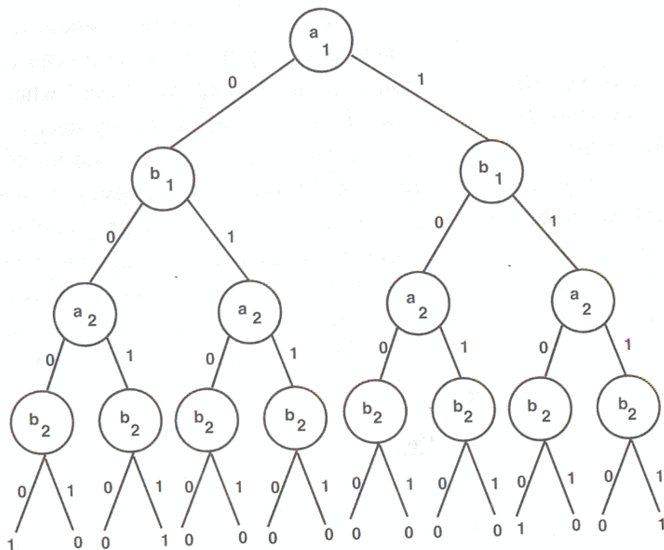
Binary Decision Trees (BDTs)

- Directed tree with a single root state.
- Every inner node is denoted with a Boolean variable (v) and lead to exactly two successors referred to as to ($low(v)$, $high(v)$).
- Every leaf is assigned a binary value, i.e. 0 or 1.

Coding of Boolean Functions with BDTs

- Every combination of values of input variables corresponds to exactly one path from the root of BDT to a leaf.
- Values stored at leaves give the the value of the function for the corresponding input values.

Binary Decision Tree $\psi = (a_1 \Leftrightarrow b_1) \wedge (a_2 \Leftrightarrow b_2)$



Disadvantage of BDTs

- BDTs are uselessly space demanding (contain redundant information).

Task

- Identify isomorphic sub-trees of the BDT from the previous slide.

Binary Decision Diagrams (BDD)

- Acyclic directed graph, of which vertices have output degree either zero (leaf) or two (inner vertex).
- Vertices of BDD have otherwise the same properties as BDT nodes.

Computing (minimal) BDD

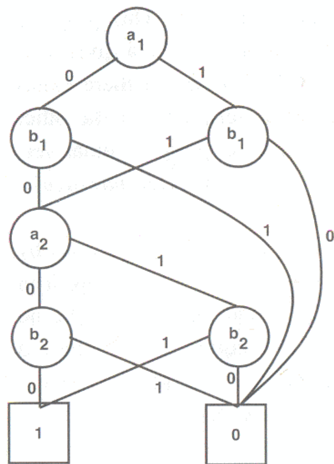
Initialisation

- For a given Boolean function take arbitrary BDD or BDT.
- Eliminate unreachable vertices
- Eliminate duplicate
  1) Remove all but one leaves with the same value.
  2) All edges incident with eliminated leaves reconnect to the the remaining leaf with the same value.

Repeat Until Fixpoint

- Eliminate duplicate inner vertices.
  - If there are two inner vertices u, v with the same label such that $low(v) = low(u)$ a $high(v) = high(u)$, then remove u and reconnect edges originally leading to u to v.
- Eliminate useless tests
  - Eliminate inner vertex v if $low(v) = high(v)$. Reconnect edges originally leading to v to $low(v)$.

BDD for $\psi = (a_1 \Leftrightarrow b_1) \wedge (a_2 \Leftrightarrow b_2)$



Observation

- Every vertex v of BDD encodes some Boolean function $F_v(x_1, \ldots, x_n)$.

Computing $F_v(x_1, \ldots, x_n)$ for values $h_1, \ldots, h_n$.

- If v is a leaf then
  - $F_v(h_1, \ldots, h_n) = 1$, if v is labelled with value 1.
  - $F_v(h_1, \ldots, h_n) = 0$, if v is labelled with value 0.
- If v is an inner vertex then
  - $F_v(h_1, \ldots, h_n) = F_{low(v)}(h_1, \ldots, h_n)$, if $h_i == 0$.
  - $F_v(h_1, \ldots, h_n) = F_{high(v)}(h_1, \ldots, h_n)$, if $h_i == 1$.

Ordering Variables in BDD — OBDD

Observation

- Some intermediate representation computed during minimisation of a BDD are also valid BDDs.
- A given Boolean function may be represented with multiple different BDDs.

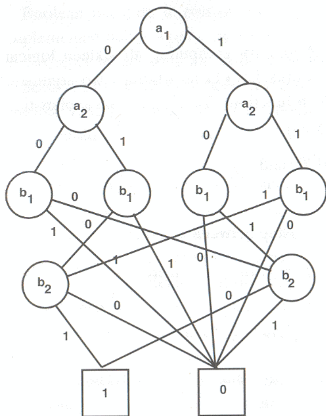
Canonical Form for BDD

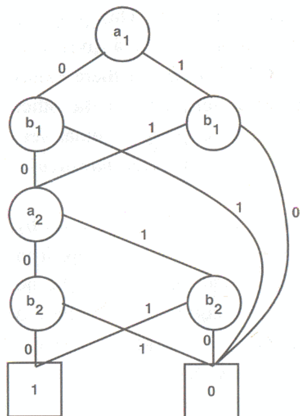
- Minimal BDD computed from a BDD, or BDT with a fixed ordering on variables in inner vertices **is unique**.
- BDD with a fixed variable ordering is referred to as to Ordered BDD (OBDD).

Computing Canonical Form

- Apply algorithm for minimal BDD.
- If performed in a bottom-up manner, obtained in linear time w.r.t. the size of initial BDT or BDD.

OBDDs for Different Variable Ordering





Restriction Operator for OBDDs

Observation

- Every OBDD represents some Boolean function.
- Boolean functions can be combined/composed using unary and binary logic operators such as $\neg, \wedge, \vee, \implies, XOR, \ldots$.
- OBDDs can be composed similarly.

Application of Logic Operators on OBDD

- Let O and O' be OBDDs corresponding to functions f and f', respectively.
- We will refer to function $Apply(O, O', \star)$, as to function that computes OBDD that represents result of application of logic operator $\star$ to functions f and f'.

Operation of Restriction

- $F_{x_i \leftarrow b}(x_1, \ldots, x_n) = F(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$
- Produces Boolean function with all but one free variables.

Realisation for OBDD

- If root r is denoted with the restricted variable x_i, the resulting OBDD will have new root
  - $low(r)$ if $b = 0$
  - $high(r)$ if $b = 1$
- Any edge leading to a inner vertex t that is denoted with the restricted variable x_i is reconnected to
  - $low(t)$ if $b = 0$
  - $high(t)$ if $b = 1$
- OBDD is minimised (contains unreachable nodes).

Shannon expansion

- Any binary logic operator can be applied on OBDDs using **Shannon expansion**:

$$F = (\neg x \wedge F_{x \leftarrow 0}) \vee (x \wedge F_{x \leftarrow 1})$$

- If $F = f \star f'$, for any binary logic operation $\star$, then

$$f \star f' = (\neg x \wedge (f_{x \leftarrow 0} \star f'_{x \leftarrow 0})) \vee (x \wedge (f_{x \leftarrow 1} \star f'_{x \leftarrow 1}))$$

Algorithm for Application of Binary Operators on OBDDs

$Apply(O, O', \star)$

- Let v, v' be root nodes of O, O', denoted with x, x', respectively.
- If v and v' are leaves denoted with values h and h', respectively, then return a leave denoted with $h \star h'$.
- Otherwise, if

$x = x'$ then return a new node w denoted with variable x, where

- $low(w) = Apply(low(v), low(v'), \star)$
- $high(w) = Apply(high(v), high(v'), \star)$

$x < x'$ then return a new node w denoted with variable x, where

- $low(w) = Apply(low(v), O', \star)$
- $high(w) = Apply(high(v), O', \star)$

$x' < x$ then return a new node w denoted with variable x', where

- $low(w) = Apply(O, low(v'), \star)$
- $high(w) = Apply(O, high(v'), \star)$

Negation Operation and Emptiness Check

Observation

- Let *OBDD X* encodes function F_X, then *OBDD Y* encoding negation function $\neg F_X$ is created as a copy of *OBDD X* in which values of leaves are switched.

Emptiness Check

- OBDDs have canonical form.
- Canonical OBDD representing an empty set is made of a single leaf denoted with 0.

Test for a Presence of Set Member (complicated way)

- Create an OBDD describing the tested member.
- Apply operation $\wedge$ on tested and newly created OBDDs.
- Employ emptiness check on the resulting OBDD.

Symbolic Representation of Kripke Structure

Encoding of Transitions of Kripke Structure

Observation

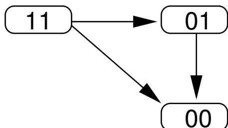
- A state of Kripke structure $M = (S, T, I)$ is given by n binary variables $a_1, \ldots, a_n$.
- Every set of states of Kripke structure can be encoded by an OBDD with n variables.
- Similarly, transition relation $T \subseteq S \times S$ can be encoded by Boolean function with $2n$ variables.

Simplification of OBDD

- Edges leading to zero leaf can be omitted.
- Non-existence of an edge indicates an edge to zero leaf.

Task

- $M = (\{00, 01, 11\}\}, \{(11, 00), (11, 01), (01, 00)\}, I)$



- T can be encoded as $F(a, b, a', b')$
- $F(a, b, a', b') =$
$(a \land b \land \neg a' \land b') \lor (a \land b \land \neg a' \land \neg b') \lor (\neg a \land b \land \neg a' \land \neg b')$
- Assume variable ordering $a < b < a' < b'$ and draw OBDD for F.

Successors of States

Observation

- Assume $M = (S, T, I)$ and $OBDD_T(a, b, a', b')$.
- Let X be a set of states given with $OBDD_X(a, b)$.
- Using $OBDD_T$ and $OBDD_X$, $OBDD_{X'}(a', b')$ representing set of successors of states in X can be computed, i.e.

$$X' = \{v \in S \mid u \in X \wedge (u, v) \in T\}.$$

Successors of States – Algorithm Idea

Computing $OBDD_{X'}$ (intuitively)

- $OBDD_{X'} = Apply(OBDD_T, OBDD_X, \wedge)$
- Modify $OBDD'_X$ so that every path of it contains vertex labelled with a'.
- In $OBDD_{X'}$ erase all vertices labelled with a and b.
- Iterate over all a' vertices, consider them as root and compute respective minimal OBDDs.
- The computed set of OBDDs combine with operation $\vee$.
- Minimise the resulting OBDD.
- Rename primed variables to unprimed.

Task

- Compute OBDD representing successors of states $\{00, 11\}$.

Predecessors of States

Computing Predecessors (intuitively).

- Modify all vertices of $OBDD_X$ to be labelled with primed variables.
- $OBDD_{X'} = Apply(OBDD_T, OBDD_X, \wedge)$
- Modify $OBDD'_X$ so that every path contains vertex labelled with a'.
- Those a' that cannot reach leaf labelled with 1 replace with a new zero leaf.
- Other a' vertices replace with the other leaf.
- Remove all primed nodes and old leaves, and minimise OBDD.

Task

- Compute OBDD representing predecessor of state $\{00\}$.

Symbolic Approach to Model Checking CTL

Reminder

Observation

- If validity of formulae φ and ψ is known for all states of Kripke structure, validity of formulae $\neg\varphi$, $\varphi \vee \psi$, $EX\,\varphi$, etc., can be easily deduced.

Algorithm Idea for Model Checking CTL

- Let $M = (S, T, I)$ be a Kripke structure and φ a CTL formula.
- Labelling function $label : S \rightarrow 2^{\varphi}$ is computed, stating which sub-formulae of φ are valid in which states of M.
- Obviously, $s_0 \models \varphi \iff \varphi \in label(s_0)$.
- Function *label* is computed gradually for every sub-formula of φ starting with the simplest sub-formulae (atomic propositions) and terminating after computing the validity of φ.

Symbolic Approach

Idea

- Set of states in which particular sub-formulae hold can be efficiently represented with OBDDs.
- Computation of *label* function for more complex sub-formulae employs manipulation with respective OBDDs.

Realisation

- Set of states are represented with OBDDs.
- Initial OBDDs are defined by functions to evaluate atomic propositions.
- OBDDs for more complex sub-formulae are composed from OBDDs of the simpler sub-formulae.
- Test for membership of initial state of Kripke structure in the set of states satisfying the verified formula.

Atomic Propositions and Logic Operators

Recall Syntax of CTL

- $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\,\varphi \mid E[\varphi\, U\, \varphi] \mid EG\,\varphi$

Computing Set of States Satisfying CTL Formula

- Notation
  - $F(\psi)$ denotes (a function describing) set of states satisfying ψ.
  - $Succ(X)$ denotes immediate successors of states in the set X.
  - $Pred(X)$ denotes immediate predecessors of states in the set X.
- Boolean Functions for Atomic Proposition
  - Atomic propositions describe properties of state variables.
  - Atomic Propositions can be encoded as Boolean functions.
- Computing Boolean Operators $\neg$ and $\vee$
  - $F(\neg\psi_1) = \neg(F\psi_1)$
  - $F(\varphi \vee \psi) = F(\varphi) \vee F(\psi)$

Temporal operators $EX(\varphi)$, $E[\varphi\, U\, \psi]$ and $EG(\varphi)$

Operator $EX(\varphi)$

- $F(EX(\varphi)) = Pred(F(\varphi))$

Operator $E(\varphi\, U\, \psi)$

- $F(E(\varphi\, U\, \psi)) = X$,
  where X is the least fix-point of recursive rule

$$X = F(\psi) \cup (F(\varphi) \cap Pred(X))$$

Operator $EG\,(\varphi)$

- $F(EG\,\varphi) = X$,
  where X is the greatest fix-point of recursive rule

$$X = F(\varphi) \cap EX(X)$$

Computing Fix-Points of Function f

The Least Fix-Point

proc $LFP(f)$
  $X = \emptyset$
  $Xold = \emptyset$
  do
    $Xold = X$
    $X := f(X)$
  while $(X \neq Xold)$
end

The Greatest Fix-Point

proc $GFP(f)$
  $X = S$
  $Xold = S$
  do
    $Xold = X$
    $X := f(X)$
  while $(X \neq Xold)$
end

Model Checking – Summary

Model Checking – Summary

Enumerative × Symbolic Approach

- Enumerative – focused on "control-flow"
- Symbolic – focused on "data-flow"

Pros w.r.t. Testing

- No source-code necessary (can be applied on models).
- Suitable for testing of parallel programs.

Pros w.r.t. Static Analysis

- Complete for systems with a finite state space.
- Verification of temporal properties.

Cons

- State space explosion problem.

Self-study

Self-study

- Explore Z3 tutorial (rise4fun.com).