Initiation à la Vérification

Binary Decision Diagrams

Stefan Schwoon

MPRI, 2012/13
January 23, 2013
Set representations

The solution of the model-checking problem for CTL and *finite-state systems* can be expressed by operations on sets:

- states satisfying (sub)formulae: $[[\psi]]$
- subformulae combined by set operations: $\cap, \cup, \ldots$

E.g., $[[\textbf{EX }\psi]]$ can be obtained by the operation

$$\text{pre}(S) := \{ s \mid \exists t : s \rightarrow t \land t \in S \}$$

$\textbf{EG}$ and $\textbf{EU}$ require fixed-point iterations on set equations

Likewise, computing the reachable states is expressible with set operations:

Start by setting $X := I$, the set of initial states;

Iterate $X := X \cup \{ t \mid \exists s : s \rightarrow t \land s \in S \}$ until fixpoint.
Set representations

How can such sets be represented:

**explicit list**: \( S = \{s_1, s_2, s_4, \ldots \} \)

**symbolic representation**: compact notation or data structure

Idea: Find a data structure that

- can compress the representation of large state sets
- permits efficient operations that match the set operations needed in CTL model checking
Caveat

Due to the pigeon-hole principle, no lossless compression method can compress *all* sets (or work efficiently for all).

The idea that we study (*binary decision diagrams*) usually works well for systems whose states can be represented as Boolean vectors, with logical operations between them. We assume:

$$S = \{0, 1\}^m \quad \text{for some } m \geq 1$$

Remark: In general, the elements of *any* finite set can be represented by Boolean vectors if *m* is chosen large enough. However, this may not be adequate in all situations.
Some pointers:

H.R. Andersen, *An Introduction to Binary Decision Diagrams*, Lecture notes, Department of Information Technology, IT University of Copenhagen


Tools:

CUDD library, including DDcal ("BDD calculator")

URL: [http://vlsi.colorado.edu/~fabio/](http://vlsi.colorado.edu/~fabio/)

SMV (BDD-based model checker):
[http://www.cs.cmu.edu/~modelcheck/smv.html](http://www.cs.cmu.edu/~modelcheck/smv.html)
Example 1: Petri-net

Consider the following Petri net:

A state can be written as \((p_1, p_2, \ldots, p_6)\), where \(p_i, \ 1 \leq i \leq 6\) indicates whether there is a token on \(P_i\).

Initial state \((1, 0, 1, 0, 1, 0)\);
other reachable states are, e.g., \((0, 1, 1, 0, 1, 0)\) or \((1, 0, 0, 1, 0, 1)\).
Example 2: Circuit

Half-adder:

The circuit has got two inputs \((x_1, x_2)\) and two outputs \((\text{carry}, \text{sum})\). Their admissible combinations can be denoted by Boolean 4-tuples, e.g. \((1, 0, 0, 1)\) \((x_1 = 1, x_2 = 0, \text{carry} = 0, \text{sum} = 1)\) is a possible combination.
The admissible combinations in Example 2 correspond to the following formula of propositional logic:

\[ F \equiv (\text{carry} \leftrightarrow (x_1 \land x_2)) \land (\text{sum} \leftrightarrow (x_1 \lor x_2) \land \neg \text{carry}) \]

In the following, we shall treat

sets of states (i.e. sets of Boolean vectors)

and formulae of propositional logic

simply as different representations of the same objects.
Binary decision graphs

Let $V$ be a set of variables (atomic propositions) and $< \, \text{a total order on} \, V$, e.g.

$$x_1 < x_2 < \text{carry} < \text{sum}$$

A binary decision graph (w.r.t. $<$) is a directed, connected, acyclic graph with the following properties:

- there is exactly one root, i.e. a node without incoming arcs;
- there are at most two leaves, labelled by 0 or 1;
- all non-leaves are labelled with variables from $V$;
- every non-leaf has two outgoing arcs labelled by 0 and 1;
- if there is an edge from an $x$-labelled node to a $y$-labelled node, then $x < y$. 
Example 2: Binary decision graph (here: a full tree)

Paths ending in 1 correspond to vectors whose entry in the truth table is 1.
A binary decision diagram (BDD) is a binary decision graph with two additional properties:

- no two subgraphs are isomorphic;
- there are no redundant nodes, where both outgoing edges lead to the same target node.

 Optionally, we omit the 0-node and the edges leading there.

Remarks: On the following slides, the blue edges are meant to be labelled by 1, the red edges by 0.
Example 2: Eliminate isomorphic subgraphs (1/3)

Alle 0- und 1-Knoten werden zusammengefasst.
Example 2: Eliminate isomorphic subgraphs (2/3)

Merged the isomorphic \textit{sum}-nodes (and the leaves).
Example 2: Eliminate isomorphic subgraphs (3/3)

Two carry nodes can be merged, but no others → done.
Example 2: Remove redundant nodes (1/2)

Both edges of the right $sum$-node point to 0.
Example 2: Remove redundant nodes (2/2)

No more redundant nodes $\rightarrow$ we are done.
Example 2: Omit 0-node

Optionally, we can remove the 0-node and edges leading to it, which makes the representation clearer (but still unambiguous).
In the following, we shall investigate operations on BDDs that are needed for CTL model checking.

- Construction of a BDD (from a PL formula)
- Equivalence check
- Intersection, complement, union
- Relations, computing predecessors
In the following, we will consider formulae of propositional logic (PL), extended with the constants $0$ and $1$, where:

0 is an unsatisfiable formula;

1 is a tautology.
Let $F$ and $G$ be formulae of PL (possibly with constants), and let $x$ be an atomic proposition.

$F[x/G]$ denotes the PL formula obtained by replacing each occurrence of $x$ in $F$ by $G$.

In particular, we will consider formulae of the form $F[x/0]$ and $F[x/1]$.

Example: Let $F = x \land y$. Then $F[x/1] \equiv 1 \land y \equiv y$ and $F[x/0] \equiv 0 \land y \equiv 0$. 
If-then-else

Let us introduce a new, ternary PL operator. We shall call it \( \text{ite} \) (if-then-else).

Note: \( \text{ite} \) does not extend the expressiveness of PL, it is simply a convenient shorthand notation.

Let \( F, G, H \) be PL formulae. We define

\[
\text{ite}(F, G, H) := (F \land G) \lor (\neg F \land H).
\]

The set of \textbf{INF formulae} (if-then-else normal form) is inductively defined as follows:

- \( 0 \) and \( 1 \) are INF formulae;

- if \( x \) is an atomic proposition and \( G, H \) are INF formulae, then \( \text{ite}(x, G, H) \) is an INF formula.
Shannon partitioning

Let $F$ be a PL formula and $x$ an atomic proposition. We have:

$$F \equiv \text{ite}(x, F[x/1], F[x/0])$$

Proof: In the following, $G$ denotes the right-hand side of the equivalence above. Let $\nu$ be a valuation s.t. $\nu \models F$. Either $\nu(x) = 1$, then $\nu$ is also a model of $F[x/1]$ and of $x$ and therefore also of $G$. The case $\nu(x) = 0$ is analogous. For the other direction, suppose $\nu \models G$. Then either $\nu(x) = 1$ and the “rest” of $\nu$ is a model of $F[x/1]$. Then, however, $\nu$ will be a model for any formula in which some of the ones in $F[x/1]$ are replaced by $x$, in particular also for $F$. The case $\nu(x) = 0$ is again analogous.

Remark: $G$ is called the Shannon partitioning of $F$.

Corollary: Every PL formula is equivalent to an INF formula.
(Proof: apply the above partitioning multiple times.)
Construction of BDDs

We can now solve our first BDD-related problem: Given a PL formula $F$ and some ordering of variables $<$, construct a BDD w.r.t. $<$ that represents $F$.

If $F$ does not contain any atomic propositions at all, then either $F \equiv 0$ or $F \equiv 1$, and the corresponding BDD is simply the corresponding leaf node.

Otherwise, let $x$ be the smallest variable (w.r.t. $<$) occurring in $F$. Construct BDDs $B_0$ and $B_1$ for $F[x/1]$ and $F[x/0]$, respectively (these formulae have one variable less than $F$).

Because of the Shannon partitioning, $F$ is representable by a binary decision graph whose root is labelled by $x$ and whose subtrees are $B_0$ and $B_1$. To obtain a BDD, we check whether $B_0$ and $B_1$ are isomorphic; if yes, then $F$ is represented by $B_0$. Otherwise we merge all isomorphic subtrees in $B_0$ and $B_1$. 
BDDs are unique

Given a PL formula \( F \) and a variable ordering \(<\), there is (up to isomorphism) exactly one BDD that respects \(<\) and represents \( F \).

**Proof:** (sketch) by induction on the number of variables, start with 0 (constant functions), then use Shannon partitioning.

Remark: Different orderings still lead to different BDDs. (possibly with vastly different sizes!)
Example: Variable orderings

Recall Example 1 (the Petri net), and let us construct a BDD representing the reachable markings:

Remark: $P_1$ is marked iff $P_2$ is not, etc.
The corresponding BDD for the ordering $p_1 < p_2 < p_3 < p_4 < p_5 < p_6$: 

![BDD Diagram](image-url)
Remarks:

If we increase the number of components from 3 to \( n \) (for some \( n \geq 0 \)), the size of the corresponding BDD will be linear in \( n \).

In other words, a BDD of size \( n \) can represent \( 2^n \) (or even more) valuations.

However, the size of a BDD strongly depends on the ordering!
Example: Repeat the previous construction for the ordering

\[ p_1 < p_3 < p_5 < p_2 < p_4 < p_6. \]
To implement CTL model checking, we need a test for equivalence between BDDs (e.g., to check the termination of a fixed-point computation).

**Problem:** Given BDDs $B$ and $C$ (w.r.t. the same ordering) do $B$ and $C$ represent equivalent formulae?

**Solution:** Test whether $B$ and $C$ are isomorphic.

**Special cases:**

- Unsatisfiability test: Check if the BDD consists just of the 0 leaf.
- Tautology test: Check if the BDD consists just of the 1 leaf.
Suppose we want to write an application in which we need to manipulate multiple BDDs.

Efficient BDDs implementations exploit the uniqueness property by storing all BDD nodes in a hash table. (Recall that each node is in fact the root of some BDD.)

Initially, the hash table has only two unique entries, the leaves 0 and 1.

Every other node is uniquely identified by the triple \((x, B_0, B_1)\), where \(x\) is the atomic proposition labelling that node and \(B_0, B_1\) are the subtrees of that node, represented by pointers to their respective roots.
Usually, one implements a function \( mk(x, B_0, B_1) \) that checks whether the hash table already contains such a node; if yes, then the pointer to that node is returned, otherwise a new node is created.

Each BDD is then simply represented by a pointer to its root.

A multitude of BDDs is then stored as a “forest” (a DAG with multiple roots).

Problem: garbage collection (by reference counting)
Equivalence test II

Let us reconsider the equivalence-checking problem. (Given two BDDs $B$ and $C$, do $B$ and $C$ represent equivalent formulae?)

If $B$ and $C$ are stored in hash tables (as described previously), then $B$ and $C$ are representable by pointers to their roots.

Due to the uniqueness property, one then simply has to check whether the pointers are the same (a constant-time procedure).
Let $F$ be a PL formula and $B$ a BDD representing $F$.

**Problem:** Compute a BDD for $\neg F$.

**Solution:** Exchange the two leaves of $B$.

(Caution: This is not quite so simple with the hash-table implementation.)
Let $F, G$ be PL formulae and $B, C$ the corresponding BDDs (with the same ordering).

**Problem:** Compute a BDD for $F \land G$ from $B$ and $C$.

We have the following equivalence:

$$F \land G \equiv \text{ite}(x, (F \land G)[x/1], (F \land G)[x/0]) \equiv \text{ite}(x, F[x/1] \land G[x/1], F[x/0] \land G[x/0])$$

If $x$ is the smallest variable occurring in either $F$ or $G$, then $F[x/1], F[x/0], G[x/1], G[x/0]$ are either the children of the roots of $B$ and $C$ (or the roots themselves).
We construct a BDD for conjunction according to the following, recursive strategy:

If $B$ and $C$ are equal, then return $B$.

If either $B$ or $C$ are the 0 leaf, then return 0.

If either $B$ or $C$ are the 1 leaf, then return the other BDD.

Otherwise, compare the two variables labelling the roots of $B$ and $C$, and let $x$ be the smaller among the two (or the one labelling both).

If the root of $B$ is labelled by $x$, then let $B_1, B_0$ be the subtrees of $B$; otherwise, let $B_1, B_0 := B$. We define $C_1, C_0$ analogously.

Apply the strategy recursively to the pairs $B_1, C_1$ and $B_0, C_0$, yielding BDDs $E$ and $F$. If $E = F$, return $E$, otherwise $mk(x, E, F)$.
Logical operations III: Union/Disjunction

Let $F, G$ be PL formulae and $B, C$ the corresponding BDDs (with the same ordering).

**Problem:** Compute a BDD for $F \lor G$ from $B$ and $C$.

**Solution:** Analogous to conjunction, with the rules for 1 and 0 leaves adapted accordingly.

**Complexity:** With dynamic programming: $O(|B| \cdot |C|)$ (every pair of nodes at most once).
Computing predecessors

In the following, we derive a strategy for computing the set

\[ \text{pre}(M) = \{ s \mid \exists s' : (s, s') \in \rightarrow \land s' \in M \}. \]

Note that the relation \( \rightarrow \) is a subset of \( S \times S \) whereas \( M \subset S \).

We represent \( M \) by a BDD with variables \( y_1, \ldots, y_m \).

\( \rightarrow \) will be represented by a BDD with variables \( x_1, \ldots, x_m \) and \( y_1, \ldots, y_m \) (states “before” and “after”).
Remark: Every BDD for $M$ is at the same time a BDD for $S \times M$!

Thus, we can rewrite $\text{pre}(M)$ as follows:

$$\{ s | \exists s': (s, s') \in \rightarrow \cap (S \times M) \}$$

Then, $\text{pre}$ reduces to the operations intersection and existential abstraction.
Existential abstraction

Existential abstraction w.r.t. an atomic proposition $x$ is defined as follows:

$$\exists x : F \equiv F[x/0] \lor F[x/1]$$

I.e., $\exists x : F$ is true for those valuations that can be extended with a value for $x$ in such a way that they become models for $F$.

**Example:** Let $F \equiv (x_1 \land x_2) \lor x_3$. Then

$$\exists x_1 : F \equiv F[x_1/0] \lor F[x_1/1] \equiv (x_3) \lor (x_2 \lor x_3) \equiv x_2 \lor x_3$$

By extension, we can consider existential abstraction over sets of atomic propositions (abstract from each of them in turn).
Let us consider the following Petri net with just one transition:
The BDD $F_{t_1}$ describes the effect of $t_1$, where $p_1, p_2, p_3$ describe the state before and $p'_1, p'_2, p'_3$ the state after firing $t_1$. 
Example: Existential abstraction

(a) $F_{t_1}[p_2'/1]$;  (b) $F_{t_1}[p_2'/0]$;  (c) $\exists p_2': F_{t_1}$;  (d) $\exists p_1', p_2', p_3': F_{t_1}$