Model Checking and Games

Binary Decision Diagrams

Rüdiger Ehlers, Clausthal University of Technology

Based on Material from PD Ralf Wimmer and Prof. Christoph Scholl, University of Freiburg

September 11, 2019

Structure of this course

Basics

1. Modelling reactive systems
2. Modelling specifications
3. Verifying specifications against models

Tools

Spin

Symbolic verification

1. Binary decision diagrams / Algorithms for BDDs
2. Satisfiability solving & Bounded Model checking

Tools

(dd/CUDD)
Alloy

Reactive synthesis

1. Introduction to reactive synthesis

Tools

Slugs

References

- R. Drechsler, B. Becker: Graphenbasierte Funktionsdarstellung, Teubner, 1998.
- P. Molitor, C. Scholl: Datenstrukturen und effiziente Algorithmen für die Logiksynthese kombinatorischer Schaltungen, Teubner, 1999.
- I. Wegener: Branching Programs and Binary Decision Diagrams, SIAM, 2000

Main problem with efficient verification so far

Problem

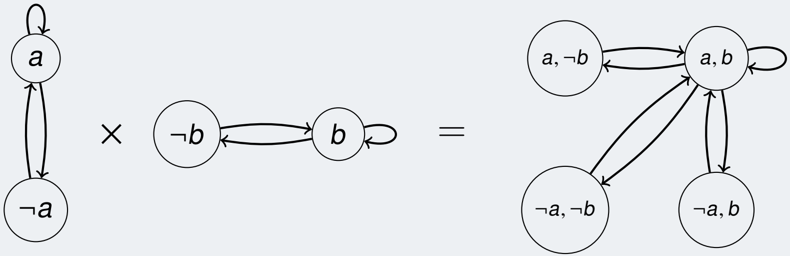
The state space explosion problem! For instance, the number of states in a Kripke structures grows exponentially with the number of processes.

Observation

But the state space is highly regular. Can we somehow take advantage of this?

On implicit state spaces

Example



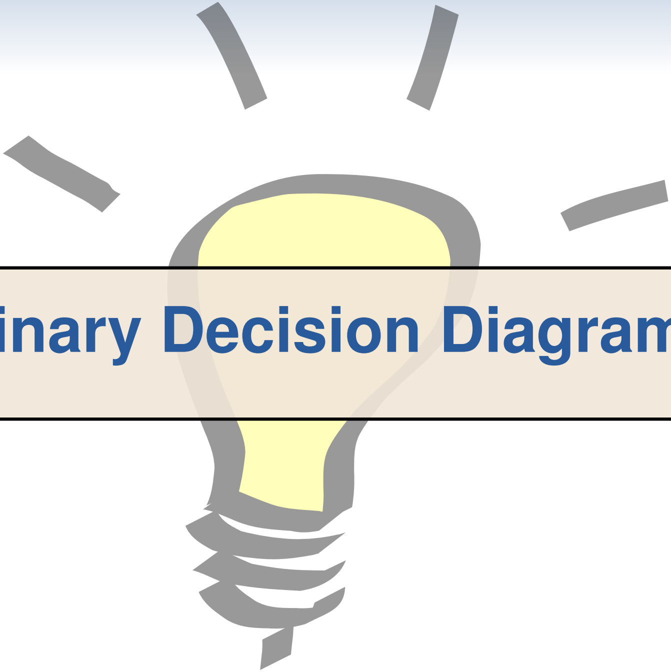
Alternative representation as Boolean function

$$(a' \lor a) \land (b' \lor b)$$

Symbolic model checking

Idea

- We represent the transition *implicitely* using Boolean functions
- We implement model checking using *operations over Boolean functions*
- If we have efficient algorithms to perform the necessary operations on Boolean functions, model checking can be made more
  - space efficient and
  - time efficient.



Binary Decision Diagrams

Binary Decision Diagrams

Definition (Syntax of BDDs)

A Binary Decision Diagram (BDD) over a set of variables $X_n = \{x_1, \ldots, x_n\}$ is a directed, acyclic graph $G = (V, E)$ having exactly one root v and the following properties:

- V consists of terminal and non-terminal (decision) nodes
- Each terminal node in V is labeled with a value from $\{0, 1\}$
- Each non-terminal node is labeled with a variable $x_i \in X_n$ and has exactly two outgoing edges, whose ends are denoted by $low(v)$ resp. $high(v)$ ($low(v), high(v) \in V$)

Binary Decision Diagrams

Definition (Semantics of BDDs)

The function represented by a BDD G $f_G : \mathbb{B}^n \rightarrow \mathbb{B}$ is inductively defined as follows:

- If G only consists of a terminal node labeled with 0, then G is a BDD for the constant 0-function (similarly for 1)
- If G has the root v, labeled with x_i, then G is a BDD of the function
$$F_G = \left(\neg x_i \wedge f_{low(v)}\right) \vee \left(x_i \wedge f_{high(v)}\right)$$
where $f_{low(v)}$ is the function represented by the BDD with root $low(v)$ (likewise for $f_{high(v)}$)

Binary Decision Diagrams

Theorem (Shannon theorem)

Let $F : \mathbb{B}^n \rightarrow \mathbb{B}$ be a Boolean function. For every $i \in \{1, \ldots, n\}$ the following holds:

$$F = \left(\neg x_i \wedge F_{\neg x_i}\right) \vee \left(x_i \wedge F_{x_i}\right)$$

with the negative cofactor

$$F_{\neg x_i}(x_1, \ldots, x_i, \ldots, x_n) = F(x_1, \ldots, 0, \ldots, x_n)$$

resp. with the positive cofactor

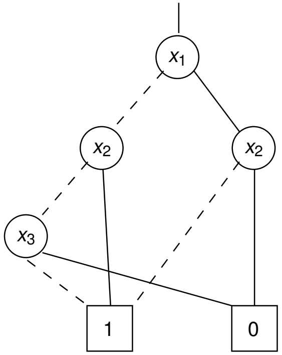
$$F_{x_i}(x_1, \ldots, x_i, \ldots, x_n) = F(x_1, \ldots, 1, \ldots, x_n)$$

Proof.

Simple calculation. □

Binary Decision Diagrams

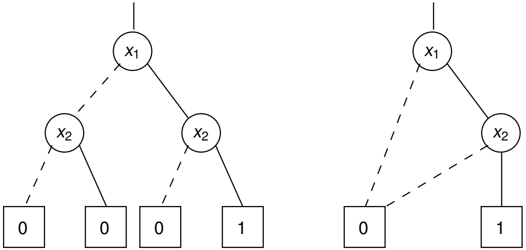
Example: $F(x_1, x_2, x_3) = \neg x_1 \neg x_2 \neg x_3 \vee \neg x_1 x_2 \vee x_1 \neg x_2$



Drawback of BDDs

"General" BDDs according to the previous definition do not provide a canonical representations of Boolean functions.

- Example: $F(x_1, x_2) = x_1 \wedge x_2$



- It is not clear how Boolean operations such as $\wedge$, $\vee$, $\neg$, etc. can be realized efficiently on general BDDs

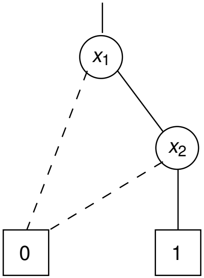
$\Rightarrow$ Limitations regarding the BDD structure necessary

$\Rightarrow$ Reduced Ordered Binary Decision Diagrams (ROBDDs)

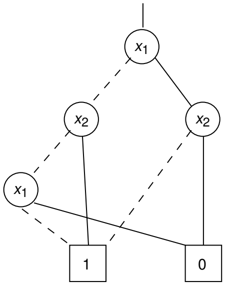
Preliminary Work

Definition (Free BDDs)

A BDD G is free iff each variable along every path from the root to a terminal node occurs at most once.



free



not free

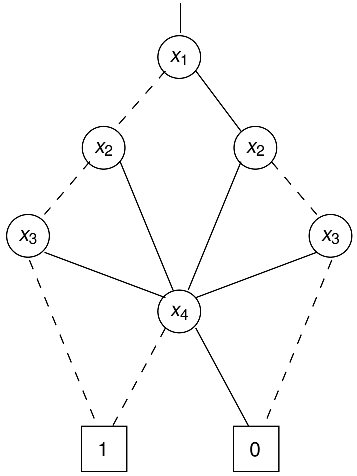
Preliminary Work

Definition (ordered BDDs, OBDDs)

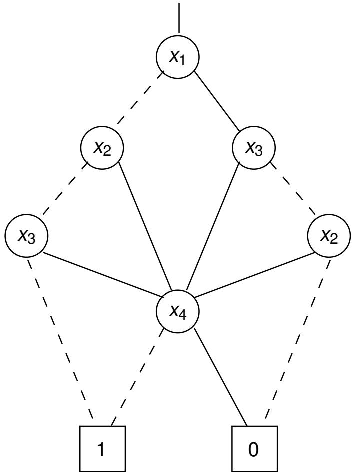
A BDD G over the set of variables $X_n = \{x_1, \ldots, x_n\}$ is ordered iff it is free and the variables on every path from the root to a terminal node occur in the same order.
This sequence is called variable order, and can also be formally described through a permutation of the form

$$\pi : \{1, \ldots, n\} \rightarrow \{x_1, \ldots, x_n\}$$

Preliminary work



ordered with $\pi = (x_1, x_2, x_3, x_4)$



not ordered

Preliminary work

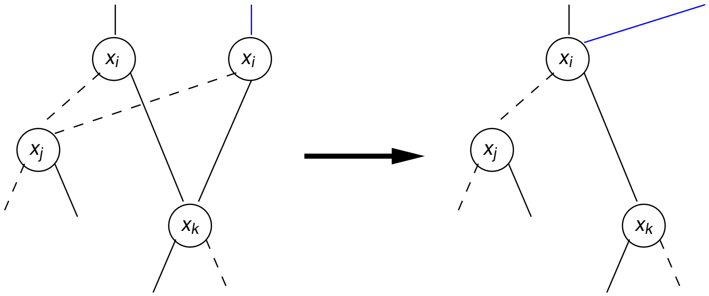
- This is still not enough . . . there can be redundancy in the OBDD:

Preliminary work

Definition ("Isomorphism" reduction)

Let $G = (V, E)$ be a BDD. Furthermore let $v, w \in V$ be two non-terminal nodes with $v \neq w$, which are labeled with the same variable from $X_n = \{x_1, \ldots, x_n\}$, and for which the expression $\mathit{low}(v) = \mathit{low}(w)$ and $\mathit{high}(v) = \mathit{high}(w)$ holds.
Then redirect all the edges from w to v, and remove w as well as both outgoing edges.

Preliminary work

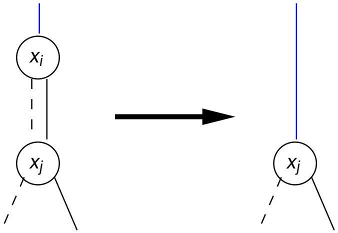


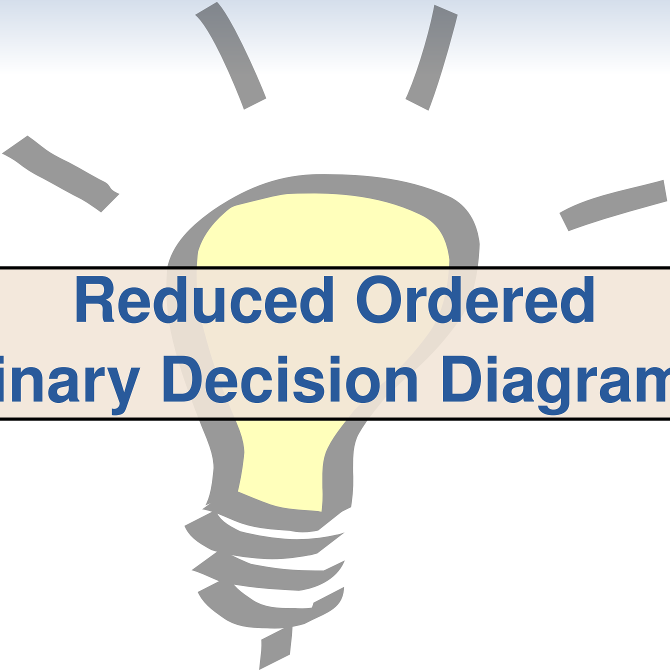
“Isomorphism” reduction

Preliminary work

Definition (“Shannon” reduction)

Let $G = (V, E)$ be a BDD $v \in V$ be a non-terminal node having $low(v) = high(v)$.
Then redirect all the edges pointing v to $low(v)$, and remove v as well as its outgoing edges.





Reduced Ordered Binary Decision Diagrams

Reduced Ordered Binary Decision Diagrams

Definition (ROBDDs)

A BDD G is a Reduced Ordered Binary Decision Diagram iff it is ordered and reduced.

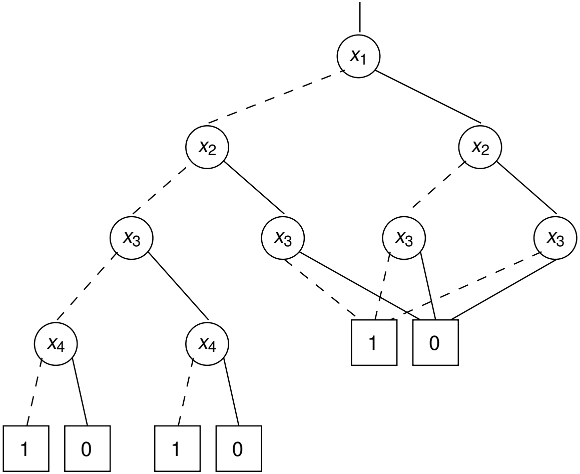
In this context, reduced means that none of the reduction rules above (“isomorphism” and “Shannon”) can be applied (anymore).

Reduced Ordered Binary Decision Diagrams

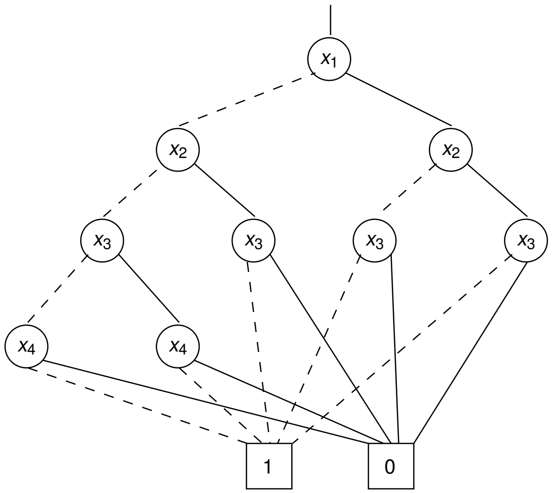
Notice

- OBDDs can be reduced to ROBDDs in layers starting from the terminal nodes

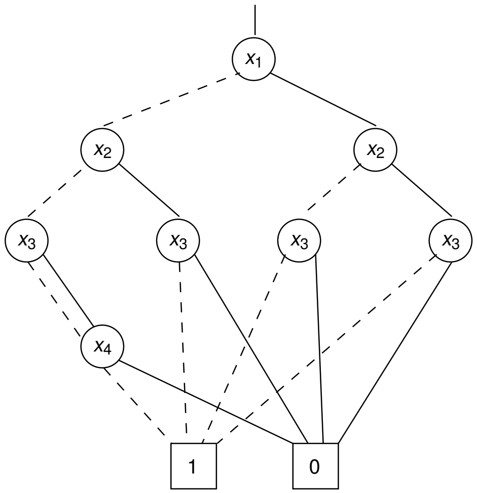
Reduced Ordered Binary Decision Diagrams



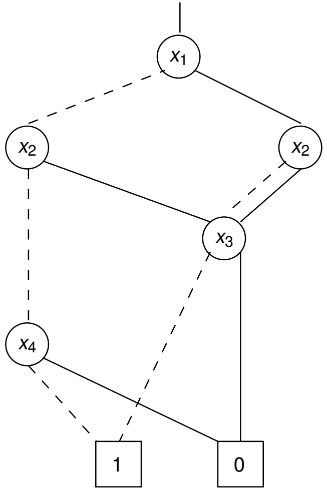
Reduced Ordered Binary Decision Diagrams



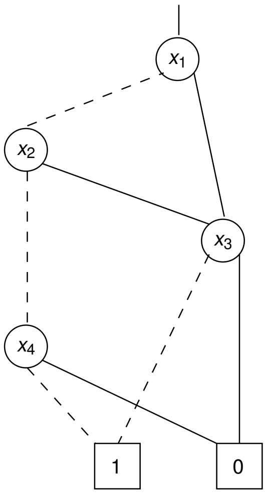
Reduced Ordered Binary Decision Diagrams



Reduced Ordered Binary Decision Diagrams



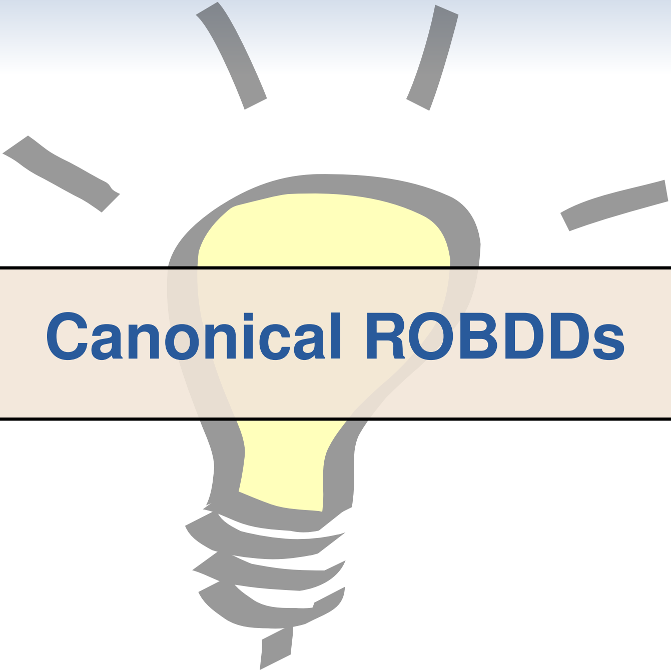
Reduced Ordered Binary Decision Diagrams



Reduced Ordered Binary Decision Diagrams

Remark:

- Current implementations of ROBDD packages do not have any reduction operation, rather they guarantee for each point in time that two identical “subfunctions” are *not* represented by different nodes (and that no node with two identical successors is inserted). This holds for the whole “system” (made of several ROBDDs).



Canonical ROBDDs

Canonical ROBDDs

Definition (Isomorphism)

Two BDDs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ over the set of variables $X_n = \{x_1, \ldots, x_n\}$ are isomorphic to each other iff there exists a bijective mapping $\alpha : V_1 \rightarrow V_2$ with the following properties:

- $\mathit{low}(\alpha(v)) = \alpha(\mathit{low}(v))$ for each $v \in V_1$
- $\mathit{high}(\alpha(v)) = \alpha(\mathit{high}(v))$ for each $v \in V_1$
- For each $v \in V_1$ the labeling of v is identical to that of $\alpha(v)$

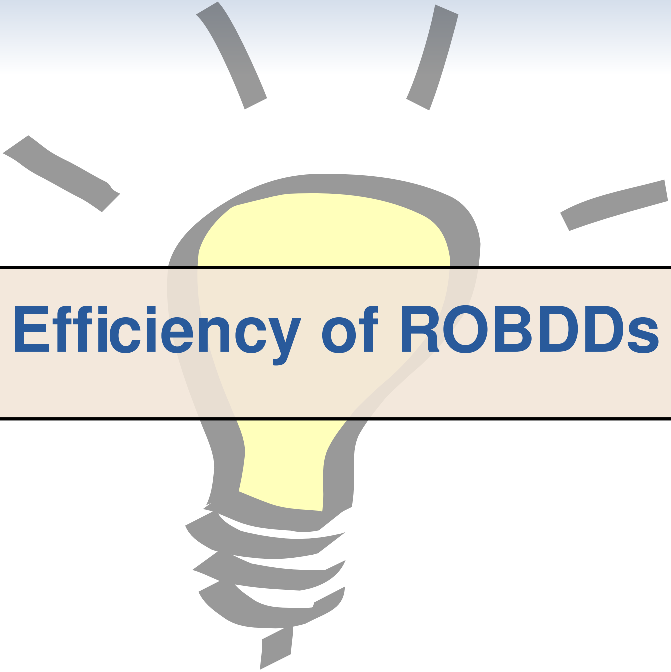
Canonical ROBDDs

Theorem

Let π be a fixed variable order. Then for each Boolean function $F : \mathbb{B}^n \to \mathbb{B}$ there exists (up to isomorphism) exactly one ROBDD for F over the set of variables $X_n = \{x_1, \ldots, x_n\}$ with π as variable order.

Proof.

The theorem can be proven by induction over the number n of variables on which F depends. $\square$



Efficiency of ROBDDs

Efficiency of ROBDDs

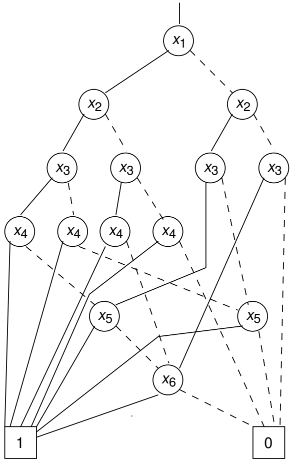
Definition (Size of a BDDs)

Let G be a BDD. Then the size of G is given by the number of non-terminal nodes.

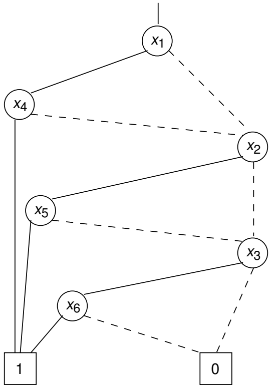
Remark

- The size of a ROBDD depends strongly on the variable order π one chooses
- Example
  - $f^{(n)}(x_1, \ldots, x_{2n}) = x_1\, x_{n+1} \vee x_2\, x_{n+2} \vee \ldots \vee x_n\, x_{2n}$
  - Exponential size when $\pi = (x_1, x_2, \ldots, x_{2n})$
  - Linear size when $\pi = (x_1, x_{n+1}, x_2, x_{n+2}, \ldots, x_n, x_{2n})$

Efficiency of ROBDDs

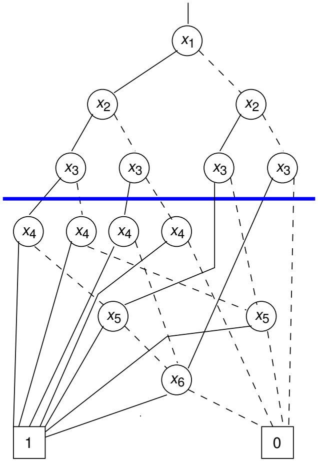


$$\pi = (x_1, x_2, \ldots, x_{2n})$$



$$\pi = (x_1, x_{n+1}, \ldots, x_n, x_{2n})$$

Efficiency of ROBDDs



$$\pi = (x_1, x_2, \ldots, x_{2n})$$

Every cofactor regarding the first n variables consitutes a different Boolean function

- $\Rightarrow$ 2^n different cofactors
- $\Rightarrow$ Below the cut at least 2^n nodes

Efficiency of ROBDDs

Theorem (Bollig, Savicky, Wegener, 1994)

Given a ROBDD with variable order π, the problem of finding a new variable order π' with minimal ROBDD-size is NP-Complete.

Heuristics:

- Define an initial variable order
- Perform dynamic reordering within an existing variable order to reduce the size of the ROBDD

Efficiency of ROBDDs

For many practically relevant functions ROBDDs provide compact representations. However the following theorem holds:

Theorem (Shannon)

"Almost every" Boolean function $F : \mathbb{B}^n \to \mathbb{B}$ requires more than $(2^n/n)$ 2-input gates for an optimal implementation

Holds also for ROBDDs, because they can be seen as multiplexer circuits

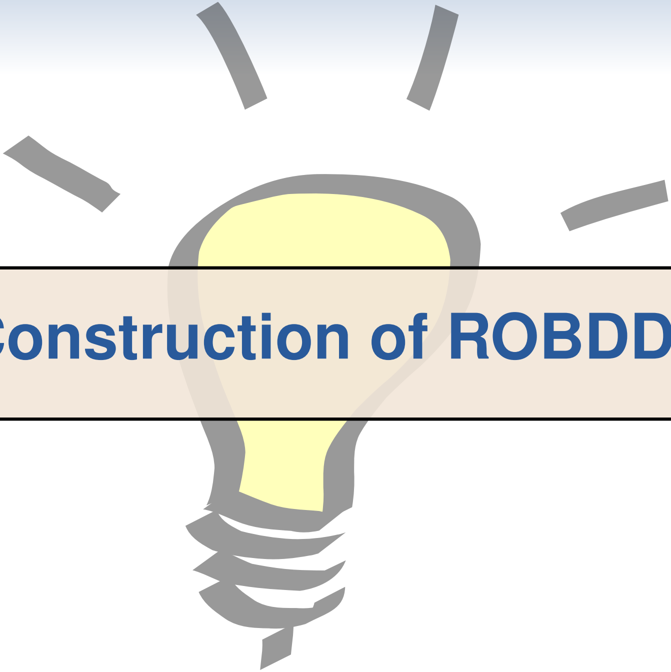
Efficiency of ROBDDs

As the following Lemma demonstrates, ROBDDs do not provide compact representations for each relevant function:

Lemma (Bryant, 1986)

Independently from the variable order, multiplication is representable with ROBDDs only with exponential complexity in the bit-width.

Construction of ROBDDs



Construction of ROBDDs

Preliminary remark

- For the conversion of models (resp. the functions representing them) into ROBDDs we have to implement binary Boolean operators like $\land$, $\lor$ and $\neg$

$\Rightarrow$ Implementation with the help of a ternary "If-Then-Else"-Operator ITE, since all the binary operation can be reduced to a call to ITE.

ITE-Operator

Definition ("If-Then-Else"-Operator, ITE)

Let $F, G, H : \mathbb{B}^n \rightarrow \mathbb{B}$ be three Boolean functions. Then the ternary ITE-Operator is defined as follows:

$$ITE(F, G, H) = (F \wedge G) \vee (\neg F \wedge H)$$

Every binary operation can be realized by using the ITE-Operator.

- $AND(F, G) = ITE(F, G, 0)$
- $OR(F, G) = ITE(F, 1, G)$
- $NOT(F) = ITE(F, 0, 1)$
- The equations for $NAND$, NOR, ... can be obtained in the same way.

ITE-Operator

Theorem

Let $F, G, H : \mathbb{B}^n \to \mathbb{B}$ be three Boolean functions over the set of variables $X_n = \{x_1, \ldots, x_n\}$. For each $i \in \{1, \ldots, n\}$ holds:

$$ITE(F, G, H) = \neg x_i \; ITE(F_{\neg x_i}, G_{\neg x_i}, H_{\neg x_i}) \vee x_i \; ITE(F_{x_i}, G_{x_i}, H_{x_i})$$

Proof.

$$\begin{aligned} ITE(F, G, H) &= F\,G \vee \neg F\,H \\ &= \neg x_i\,(F\,G \vee \neg F\,H)_{\neg x_i} \vee x_i\,(F\,G \vee \neg F\,H)_{x_i} \\ &= \neg x_i\,(F_{\neg x_i}\,G_{\neg x_i} \vee \neg F_{\neg x_i}\,H_{\neg x_i}) \vee x_i\,(F_{x_i}\,G_{x_i} \vee \neg F_{x_i}\,H_{x_i}) \\ &= \neg x_i\,ITE(F_{\neg x_i}, G_{\neg x_i}, H_{\neg x_i}) \vee x_i\,ITE(F_{x_i}, G_{x_i}, H_{x_i}) \end{aligned}$$

□

Algorithm for the ITE-Operator

For the application of the ITE-Operator to ROBDDs, a recursive method can be derived from the previous theorem, which provides the foundation for the symbolic simulation covered in the following

- Given
  - ROBDDs F, G and H over the set of variables $X_n = \{x_1, \ldots, x_n\}$ with an identical variable order π
- Find
  - ROBDD for $ITE(F, G, H)$

Algorithm for the ITE-Operator

Procedure

- Compute recursively the ROBDDs of $ITE(F_{\neg x_i}, G_{\neg x_i}, H_{\neg x_i})$ and $ITE(F_{x_i}, G_{x_i}, H_{x_i})$ for the “top variable” $x_i \in X_n$
- Use the Shannon reduction as soon as possible

$$ITE(F_{\neg x_i}, G_{\neg x_i}, H_{\neg x_i}) = ITE(F_{x_i}, G_{x_i}, H_{x_i})$$
$$\Downarrow$$
$$ITE(F, G, H) = ITE(F_{\neg x_i}, G_{\neg x_i}, H_{\neg x_i}) = ITE(F_{x_i}, G_{x_i}, H_{x_i})$$

- In case no such node already exists create a new node v labeled with x_i, whose *low* (resp. *high*) edge points to $ITE(F_{\neg x_i}, G_{\neg x_i}, H_{\neg x_i})$ (resp. $ITE(F_{x_i}, G_{x_i}, H_{x_i})$). Otherwise perform isomorphism reduction!

Algorithm for the ITE-Operator

Continue the procedure until some base case is encountered

- $$\begin{aligned} ITE(1, F, G) &= ITE(0, G, F) \\ &= ITE(F, 1, 0) \\ &= ITE(G, F, F) = F \end{aligned}$$

  (Covers also cases with “pure” constant parameters)

Algorithm for the ITE-Operator

ITE-Algorithm

```
**robdd*** ITE(F, G, H)
{
    (robdd* result, bool terminal) = TERMINAL_CASE(F,G,H);
    if (terminal) { return result; }

    variable x = TOP_VARIABLE(F,G,H);
    robdd* low = ITE($F_{\neg x}$,$G_{\neg x}$,$H_{\neg x}$);
    robdd* high = ITE($F_x$,$G_x$,$H_x$);
    if (low == high) { return low; }

    robdd* r = FIND_OR_ADD_UNIQUE_TABLE(x,low,high);
    return r;
}
```

Running time of this algorithm??

Algorithm for the ITE-Operator

Improved ITE-Algorithm

```
robdd* ITE(F, G, H)
{
   (robdd* result, bool terminal) = TERMINAL_CASE(F,G,H);
   if (terminal) { return result; }

   (robdd* result, bool found) = CHECK_COMPUTED_TABLE(F,G,H);
   if (found) { return result; }

   variable x = TOP_VARIABLE(F,G,H);
   robdd* low = ITE($F_{\neg x}$,$G_{\neg x}$,$H_{\neg x}$);
   robdd* high = ITE($F_x$,$G_x$,$H_x$);
   if (low == high) { return low; }

   robdd* r = FIND_OR_ADD_UNIQUE_TABLE(x,low,high);

   INSERT_COMPUTED_TABLE((F,G,H),r);

   return r;
}
```

$\Rightarrow$ “Computed Table” records the ITE-operations already done

Runtime of the ITE-Operators

Theorem

The runtime of $ITE(F, G, H)$ lies in $O(|F| \cdot |G| \cdot |H|)$, where $| \cdot |$ indicates the size of a ROBDD.

Proof.

For each triple (F, G, H), due to the use of the "Computed Table" $ITE(F, G, H)$ is computed at most once. The number of possible triples (F, G, H) is limited by $(|F| \cdot |G| \cdot |H|)$, since the recursive calls which are triggered by $ITE(F, G, H)$ can have only nodes as arguments, which are also contained in F, G, and H. □

Which optimistic assumption does the proof make?

Runtime of the ITE-Operators

Lemma

The logical And of two ROBDDs F and G can be computed with a runtime of $O(|F| \cdot |G|)$.

Proof.

Follows from the previous theorem with
$AND(F, G) = ITE(F, G, 0)$. □

Runtime of the ITE-Operators

Lemma

The logical Or of two ROBDDs F and G can be computed with a runtime of $O(|F| \cdot |G|)$.

Proof.

Follows from the previous theorem with $OR(F, G) = ITE(F, 1, G)$. $\square$

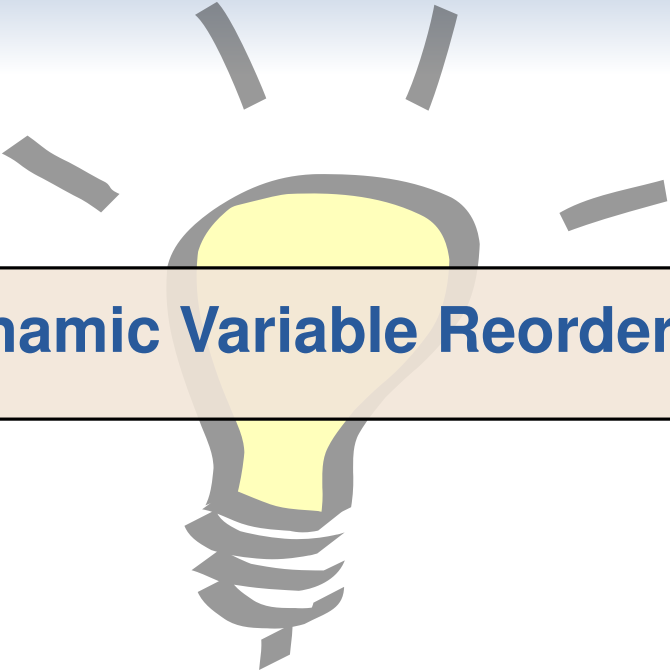
Runtime of the ITE-Operators

Lemma

With the help of complementary edges the logical negation of a ROBDD F can be computed in constant time. Otherwise the runtime lies in $O(|F|)$.

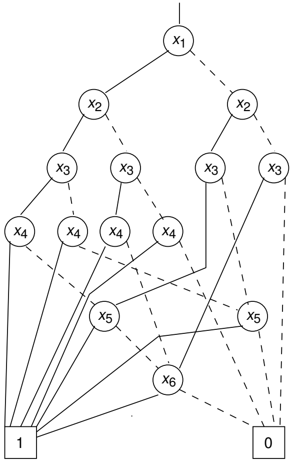
Proof.

The first part is trivial, the second part follows from the previous theorem with $NOT(F) = ITE(F, 0, 1)$. □

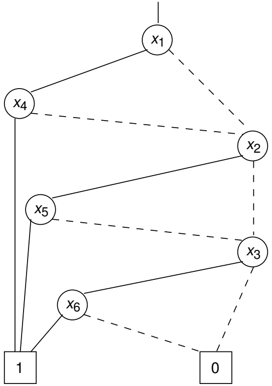


Dynamic Variable Reordering

Another look at an example from earlier



$\pi = (x_1, x_2, \ldots, x_{2n})$



$\pi = (x_1, x_{n+1}, \ldots, x_n, x_{2n})$

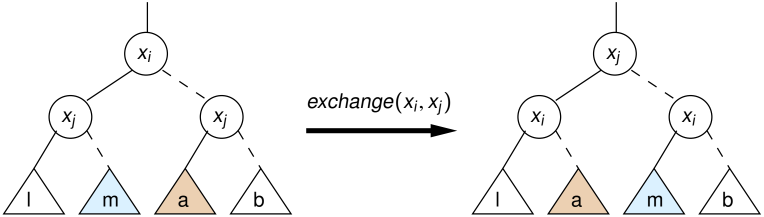
Dynamic modification of the variable ordering

Procedure

- Begin the ROBDD construction with an initial variable order
- If during the construction its size exceeds a given threshold (that is in turn continuously increased during the symbolic simulation), then the current ROBDD operation is interrupted, and the variable order is changed in order to reduce the ROBDD size.

Sifting

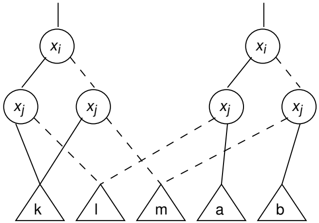
- The most famous technique for the dynamic variable ordering is called Sifting (Rudell, 1993)
- Sifting is based on swapping pairs of variables which are next to each other in the current variable order



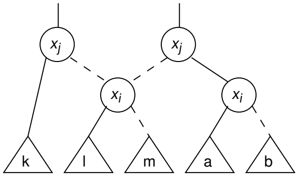
- Variable swapping is a local operation and involves only the nodes labeled with the variables being swapped.

Sifting

Swapping variables can modify the ROBDD size



Starting Basis



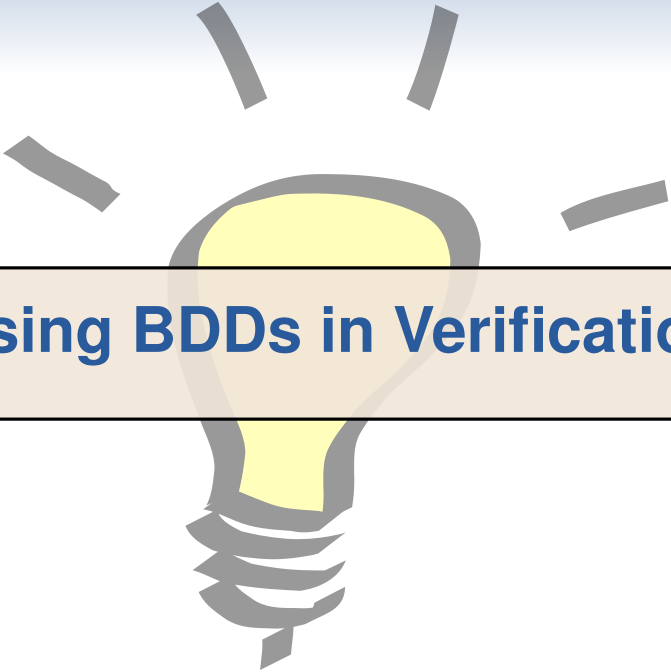
exchange(x_i, x_j)

$\Rightarrow$ Saving: 2 Nodes

Sifting

- Sifting can be used to optimizing the variable order regarding the ROBDD size
  1. Define for each variable (in the example x_4) its optimal position within the variable order keeping the position of the other variables
  2. Move the variables to its “locally optimal” position, where the ROBDD size was the smallest
- In the worst case $O(n^2)$ level exchanges with n variables
- In case the “move” of a variable is aborted when the resulting ROBDD size exceeds a threshold $c \cdot |G|$ (given G as the original ROBDD), the runtime lies in $O(n^2 \cdot |G|)$
- In general, the presented technique does not lead to a globally optimal variable order (but nevertheless works well in practice).

x_1	x_1	x_1	x_1	x_1	x_1	x_1	x_4	x_1	x_1
x_2	x_2	x_2	x_2	x_2	x_2	x_4	x_1	x_4	x_2
x_3	x_3	x_3	x_3	x_3	x_4	x_2	x_2	x_2	x_4
x_4	x_5	x_5	x_5	x_4	x_3	x_3	x_3	x_3	x_3
x_5	x_4	x_6	x_4	x_5	x_5	x_5	x_5	x_5	x_5
x_6	x_6	x_4	x_6	x_6	x_6	x_6	x_6	x_6	x_6



Using BDDs in Verification

Encoding a Kripke structure into a Boolean formula

Main idea

Let a Kripke structure $\mathcal{K} = (S, \mathrm{AP}, T, S_0, L)$ be given with *finite* state set S. We can encode $\mathcal{K}$ in Boolean function (and BDD) form as follows:

- We use a set of variables $\mathcal{V}$ large enough so that we can encode all states in S as a *valuation* to all variables $\mathcal{V}$. For every state $s \in S$, let us write the encoding as $[s]_{\mathcal{V}}$.
- Allocate a second set of variables $\mathcal{V}'$ of the same size, and for every state $s \in S$ let $[s]_{\mathcal{V}'}$. be the same encoding into the second set of variables.
- We can now represent T as the following Boolean formula:

$$B_T = \bigvee_{(s,s')\in T} [s]_{\mathcal{V}} \wedge [s']_{\mathcal{V}'}$$

Motivation

Why do we do this?

- The BDDs obtained for large Kripke structures are often small (for *some* variable ordering)
- Parallel composition is now quite easy!

Asynchronous composition

Let $B_{T,1}$ and $B_{T,2}$ be the Boolean function encodings of two transition systems with overlapping state sets $\mathcal{V}^1/\mathcal{V}'^1$ and $\mathcal{V}^2/\mathcal{V}'^2$. We can encode the transition function of the asynchronous product as follows:

$$\begin{aligned} B_{T,1+2} = B_{T,1} \wedge & \bigwedge_{v \in \mathcal{V}^2 \setminus \mathcal{V}^1} (v = v') \\ \vee\, B_{T,2} \wedge & \bigwedge_{v \in \mathcal{V}^1 \setminus \mathcal{V}^2} (v = v') \end{aligned}$$

Where v' is the variable of the second variable set corresponding to v.

Binary decision diagrams for CTL model checking

Modelling a Kripke structure as a BDD

We can represent all parts of a Kripke structure $\mathcal{K} = (S, \text{AP}, T, S_0, L)$ as BDDs over variables $\mathcal{V}$ and $\mathcal{V}'$:

- the transition relation as a BDD over $\mathcal{V}$ and $\mathcal{V}'$
- the initial states as a BDD over $\mathcal{V}$
- the labelling as a family of BDDs $\{B_p\}_{p \in AP}$ over $\mathcal{V}$.

Binary decision diagrams for CTL model checking

Evaluating a μ-calculus formula

If we have an encoding of the $\Box$ and $\Diamond$ operators, we can evaluate μ-calculus formulas such as:

$$\mu X.p \cup \Box X$$

When evaluating the formula,

- the state sets are always represented as BDDs that map those variable valuations to **tt** for which the respective state is contained in the state set computed, and
- Disjunction implements the $\cup$ operator, conjunction implements the $\cap$ operator

Binary decision diagrams for CTL model checking

Evaluating the $\diamond$ operator

Key to the symbolic approach to CTL model checking is an efficient implementation of the $\diamond$ and $\diamond$ operators. Using purely Boolean terms, for instance, $\diamond X$ can be encoded as:

$$\exists \mathcal{V}'.(B_T \wedge X[\mathcal{V}'/\mathcal{V}])$$

Note

Two new operations are used here:

- Variable substitution (the [.../...] part)
- Existential quantification

Both can be implemented efficienty using BDDs.

Summary

Summary / List of Concepts

- Binary decision diagrams
- Reduced ordered binary decision diagrams
- Canonicity of ROBDDs
- Efficiency of ROBDDs
- Variable reordering
- Use of ROBBDs in (CTL) model checking