Model checking

- For a Kripke structure $M = (S, I, R, L)$ over AP and a (state based) temporal logic formula $\varphi$
  find the set of all states in $S$ that satisfy $\varphi$:
  
  $$X = \{ s \in S : M, s \models \varphi \}$$
Explicit vs. symbolic model checking

- **Explicit model checking**
  - M is *explicitly* represented in memory as a labeled, directed graph

- **Symbolic model checking**
  - Based on manipulation with *Boolean formulas*
  - The algorithm operates on entire sets of states rather than on individual states
  - Reduction of time and memory consumption
Did you know...?

- Explicit model checking
  - M is explicitly represented in memory directed graph

- Symbolic model checking
  - Based on manipulation with Boolean functions
  - The algorithm operates on entire system, not on individual states
  - Reduction of time and memory costs

George Boole (1815 –1864)
*English mathematician, philosopher and logician*
Today

- Ordered Binary Decision Diagrams (OBDDs)
  - We will later present a symbolic CTL model checking algorithm, based on manipulation with OBDDs
Outline

- Representing Boolean functions using OBDDs
  - Size of the OBDDs depends on the variable ordering
  - Heuristics for good variable ordering
- Logical operations on OBDDs
- Representing Kripke structures using OBDDs
Ordered Binary Decision Diagrams

• Canonical form representation for Boolean formulas
  ▪ Often substantially more compact than traditional normal forms (conjunctive NF, disjunctive NF)
  ▪ Variety of applications
    • symbolic simulation
    • verification of combinational logic
    • verification of finite-state concurrent systems

• We first introduce binary decision trees
  ▪ ... and then generalize binary decision trees to obtain (ordered) binary decision diagrams
Binary Decision Trees (BDTs)

- Rooted, directed trees
- Two types of vertices
  - Nonterminal
    - Each nonterminal vertex $v$
      - is labeled by a variable $\text{var}(v)$
      - has two successors:
        - $\text{low}(v)$ ... variable $v$ is assigned 0
        - $\text{high}(v)$ ... variable $v$ is assigned 1
  - Terminal
    - Each terminal vertex $v$ is labeled by $\text{value}(v)$ which is either 0 or 1
Binary Decision Trees (BDTs)

\[ \text{var}(u) = a_1 \]

\[ \text{low}(u) = v \]

\[ \text{high}(u) = w \]

Assignment \( t \): \( \text{value}(t) = 1 \)
Q: What function does this represent?
Binary Decision Trees (BDTs)
Binary Decision Trees (BDTs)

- Every binary decision tree represents a Boolean formula (Boolean function \( f : \{0,1\}^n \rightarrow \{0,1\} \))
- Our example: two bit comparator
  \[
  f(a_1, a_2, b_1, b_2) = (a_1 \leftrightarrow b_1) \land (a_2 \leftrightarrow b_2)
  \]

- To decide whether a particular truth assignment makes the formula true or false
  - Traverse the tree from the root to a terminal vertex \( t \)
  - On the path, in a nonterminal vertex \( v \):
    - If the variable \( \text{var}(v) \) is 0, then the next vertex on the path from the root to the terminal vertex will be \( \text{low}(v) \)
    - If the variable \( \text{var}(v) \) is 1, then the next vertex on the path from the root to the terminal vertex will be \( \text{high}(v) \)
  - \( \text{value}(t) \) is the value of the function / formula for this assignment
Binary Decision Trees (BDTs)

a1 := 1, a2 := 0
b1 := 1, b2 := 1
Binary Decision Trees (BDTs)

a1:= 1, a2:= 0

b1:= 1, b2:= 1
Binary Decision Trees (BDTs)

- $a_1 := 1$, $a_2 := 0$
- $b_1 := 1$, $b_2 := 1$
Binary Decision Trees (BDTs)

- $a_1 := 1, a_2 := 0$
- $b_1 := 1, b_2 := 1$
Binary Decision Trees (BDTs)

- Not very concise representation for Boolean functions
  - Essentially the same size as truth tables
- Usually a lot of redundancy in such trees
  - Two BDTs \( T_1, T_2 \) are isomorphic iff there exists one-to-one and onto function \( h \) s.t.
    - \( h \) maps terminals of \( T_1 \) to terminals of \( T_2 \)
    - \( h \) maps nonterminals of \( T_1 \) to nonterminals of \( T_2 \)
    - for every terminal vertex \( v \), \( \text{value}(v) = \text{value}(h(v)) \)
    - for every nonterminal vertex \( v \)
      - \( \text{var}(v) = \text{var}(h(v)) \)
      - \( h(\text{low}(v)) = \text{low}(h(v)) \)
      - \( h(\text{high}(v)) = \text{high}(h(v)) \)
  - In our example: 8 subtrees with roots labeled by \( b_2 \), but only 3 are distinct (i.e. not isomorphic)
    - merging the isomorphic subtrees, we obtain a more concise representation – a binary decision diagram
BDT → BDD

Jan Kofroň, František Plášil, Lecture 6
BDT $\rightarrow$ BDD

Jan Kofroň, František Plášil, Lecture 6
BDT $\rightarrow$ BDD
BDT $\rightarrow$ BDD

[Diagram of a binary decision tree (BDD) with nodes labeled a1, a2, and b1, b2, illustrating the conversion process.]
Jan Kofroň, František Plášil, Lecture 6
Binary Decision Diagrams (BDDs)

- Rooted, directed acyclic graphs
- Two types of vertices
  - Nonterminal
    - Each nonterminal vertex $v$
      - is labeled by a variable $\text{var}(v)$
      - has two successors:
        - $\text{low}(v)$ ... variable $v$ is assigned 0
        - $\text{high}(v)$ ... variable $v$ is assigned 1
  - Terminal
    - Each terminal vertex $v$ is labeled by $\text{value}(v)$ which is either 0 or 1
Every vertex $v$ in a BDD determines a Boolean function $f_v(x_1, ..., x_n)$

- If $v$ is a terminal vertex
  - $f_v(x_1, ..., x_n) = value(t)$

- If $v$ is a nonterminal vertex with $\text{var}(v) = x_i$
  - $f_v(x_1, ..., x_n) = (\neg x_i \land f_{\text{low}(v)}(x_1, ..., x_n)) \lor (x_i \land f_{\text{high}(v)}(x_1, ..., x_n))$

A BDD with root $r$ represents the Boolean function $f_r(x_1, ..., x_n)$
It is desirable to have a **canonical representation** for Boolean functions

- Two Boolean functions are logically equivalent if and only if they have isomorphic canonical representations
  - \( \rightarrow \) simplifies
    - checking equivalence of two formulas
    - checking satisfiability of a formula

Two BDDs \( B_1, B_2 \) are isomorphic iff there exists one-to-one and onto function \( h \) s.t.

- \( h \) maps terminals of \( B_1 \) to terminals of \( B_2 \)
- \( h \) maps nonterminals of \( B_1 \) to nonterminals of \( B_2 \)
- for every terminal vertex \( v \), \( \text{value}(v) = \text{value}(h(v)) \)
- for every nonterminal vertex \( v \)
  - \( \text{var}(v) = \text{var}(h(v)) \)
  - \( h(\text{low}(v)) = \text{low}(h(v)) \)
  - \( h(\text{high}(v)) = \text{high}(h(v)) \)
Order Binary Decision Diagrams (OBDDs)

• By placing two restrictions on BDDs, we obtain a canonical representation of Boolean functions –
  Ordered Binary Decision Diagrams (OBDDs)
  1. The same order of variables \( \rightarrow \) imposing a total ordering on the variables
  2. No isomorphic subtrees or redundant vertices \( \rightarrow \) applying 3 transformation rules

• Remove duplicate terminals
  ▪ Eliminate all but one terminal vertex with a given label and redirect all arcs to the eliminated vertices to the remaining one

• Remove duplicate nonterminals
  ▪ If two nonterminals \( u \) and \( v \) have \( \text{var}(u) = \text{var}(v), \text{low}(u) = \text{low}(v) \) and \( \text{high}(u) = \text{high}(v) \), then eliminate \( u \) or \( v \) and redirect all incoming arcs to the other vertex

• Remove redundant tests
  ▪ If nonterminal \( v \) has \( \text{low}(v) = \text{high}(v) \), then eliminate \( v \) and redirect all incoming arcs to \( \text{low}(v) \)
Remove Duplicate Terminals
Remove Duplicate Terminals
Remove Redundant Tests

Jan Kofroň, František Plášil, Lecture 6
Remove Redundant Tests
Remove Redundant Tests
Remove Redundant Tests

Jan Kofroň, František Plášil, Lecture 6
Remove Redundant Tests

Jan Kofroň, František Plášil, Lecture 6
Remove Redundant Tests
Remove Duplicate Nonterminals
Remove Duplicate Nonterminals
Ordered Binary Decision Diagrams (OBDDs)

- Transformation procedure
  - Start with a BDD satisfying the ordering property
  - Apply the transformation rules until the size of the diagram can no longer be reduced
- This can be done in a bottom-up manner by a procedure called Reduce (in time which is linear in the size of the original BDD)
- OBDD as a canonical form
  - Checking equivalence = checking isomorphism
  - Checking satisfiability = checking equivalence to the trivial OBDD (only one terminal labeled by 0)
Ordered Binary Decision Diagrams (OBDDs)

- The size of an OBDD can depend critically on the variable ordering

\[ a_1 < b_1 < a_2 < b_2 \]

\[ a_1 < a_2 < b_1 < b_2 \]
Ordered Binary Decision Diagrams (OBDDs)

- For n-bit comparator
  - $a_1 < b_1 < ... < a_n < b_n$
    - 3n + 2 vertices in the OBDD
  - $a_1 < ... < a_n < b_1 < ... < b_n$
    - $3 \times 2^n - 1$ vertices in the OBDD

- In general
  - Finding an optimal ordering for variables is infeasible
    - Even checking that a particular ordering is optimal is NP-complete
  - There are many functions that have exponential size OBDDs for any variable ordering

- **However:** In practice, using OBDDs to encode Boolean functions, sets, Kripke structures, etc. in many cases saves time and memory
Heuristics for good variable ordering

- Combinational circuit
  - Related variables should be “close together” in the ordering
  - Variables in a sub-circuit
    - determining the sub-circuit output
  - Depth-first traversal

- Dynamic reordering
Logical operations with OBDDs

- \( f(x_1, \ldots, x_n) \) – a Boolean function
- **Restriction** of some argument \( x_i \) of \( f \) to a constant value \( b \) (0 or 1)
  - \( f|_{x_i \leftarrow b}(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n) \)
  - Implementation: depth-first traversal of the OBDD

\[
\begin{align*}
\text{Reduce} & \\
\text{b = 0} & \\
x & y
\end{align*}
\]
Logical operations with OBDDs

- **Shannon expansion**
  - \( f = (-x \land f|_{x=0}) \lor (x \land f|_{x=1}) \)
  - Application: efficient implementation of logical operations on Boolean functions represented using OBDDs
Let \(*\) be an arbitrary two-argument logical operation
- imagine logical AND for instance

\(f, f'\) – Boolean functions

\(v, v'\) – roots of the OBDDs representing \(f, f'\)
- Both OBDDs respect the same variable ordering

If \(v\) is a nonterminal vertex, \(x = var(v)\)

If \(v'\) is a nonterminal vertex, \(x' = var(v')\)
Logical operations with OBDDs

- If \( v, v' \) are terminal vertices
  
  \[ f \cdot f' = \text{value}(v) \cdot \text{value}(v') \]
  
  for instance: \( \text{value}(v) \land \text{value}(v') \)

- If \( v, v' \) are nonterminal vertices and \( x = x' \)
  
  \[ f \cdot f' = (\neg x \land (f|_{x\leftarrow 0} \cdot f'|_{x\leftarrow 0})) \lor (x \land (f|_{x\leftarrow 1} \cdot f'|_{x\leftarrow 1})) \]
  
  The subproblems are solved recursively

  The root of the resulting OBDD will be a new node \( w \) with

  \( \text{var}(w) = x \), \( \text{low}(w) \) will be the OBDD for \( f|_{x\leftarrow 0} \cdot f'|_{x\leftarrow 0} \) and

  \( \text{high}(w) \) will be the OBDD for \( f|_{x\leftarrow 1} \cdot f'|_{x\leftarrow 1} \)
Logical operations with OBDDs

- If $v$ is a nonterminal vertex and
  - Either $v'$ is a nonterminal vertex and $x < x'$
  - Or $v'$ is a terminal vertex
  \[ \Rightarrow f' \text{ does not depend on } x \]
  \[ f'|_{x \leftarrow 0} = f'|_{x \leftarrow 1} = f' \]
  \[ \Rightarrow \text{Shannon expansion simplifies to} \]
  \[ f \ast f' = (\neg x \land (f|_{x \leftarrow 0} \ast f')) \lor (x \land (f|_{x \leftarrow 1} \ast f')) \]
  - The subproblems are solved recursively
  - The root of the resulting OBDD will be a new node $w$ with $\text{var}(w) = x$, $\text{low}(w)$ will be the OBDD for $f|_{x \leftarrow 0} \ast f'$ and $\text{high}(w)$ will be the OBDD for $f|_{x \leftarrow 1} \ast f'$
Logical operations with OBDDs

- If \( v' \) is a nonterminal vertex and
  - Either \( v \) is a nonterminal vertex and \( x' < x \)
  - Or \( v \) is a terminal vertex

  \( \Rightarrow f \) does not depend on \( x' \)
  - \( f|_{x \leftarrow 0} = f|_{x \leftarrow 1} = f \)

\( \Rightarrow \) Shannon expansion simplifies to

- \( f \ast f' = (\neg x' \land (f \ast f'|_{x \leftarrow 0})) \lor (x' \land (f \ast f'|_{x \leftarrow 1})) \)
  - The subproblems are solved recursively
  - The root of the resulting OBDD will be a new node \( w \) with \( \text{var}(w) = x, \text{low}(w) \) will be the OBDD for \( f \ast f'|_{x \leftarrow 0} \) and \( \text{high}(w) \) will be the OBDD for \( f \ast f'|_{x \leftarrow 1} \)
Logical operations with OBDDs

- To prevent the algorithm from being exponential, use dynamic programming
  ➔ polynomial algorithm

- Each subproblem corresponds to a pair of OBDDs that are subgraphs of OBDDs for $f, f'$
  - Each subgraph is uniquely determined by its root
  - The number of subgraphs in the OBDD for $f$ is bounded by the size of the OBDD for $f$ (similar bound for $f'$)
  ➔ the number of subproblems is bounded by the product of the size of the OBDDs for $f$ and $f'$

- Result Cache
  - A hash table used to record previously computed subproblems
Representing relations using OBDDs

• If $Q$ is an $n$-ary relation over $\{0,1\}$
  ▪ $Q$ can be represented by the OBDD for its characteristic function:
    $f_Q(x_1, ..., x_n) = 1$ iff $Q(x_1, ..., x_n)$

• Let $Q$ be an $n$-ary relation over a finite domain $D$
  ▪ Without loss of generality we assume $D$ has $2^m$ elements for some $m > 0$
  ▪ We encode elements of $D$ using a bijection
    $\phi: \{0,1\}^m \rightarrow D$
  ▪ We construct a Boolean relation $Q_b$ of arity $m \times n$:
    $Q_b(<x_1>, ..., <x_n>) = Q(\phi(<x_1>), ..., \phi(<x_n>))$
    • $<x_i>$ is a vector of $m$ Boolean variables that encodes the variable $x_i$, which takes values in $D$
  ▪ $Q$ can now be represented as the OBDD determined by the characteristic function $f_{Q_b}$ of $Q_b$
Representing Kripke structures using OBDDs

- \( M = (S, R, L) \)

- **Encoding \( S \)**
  - We assume there are exactly \( 2^m \) states
  - \( \phi: \{0,1\}^m \rightarrow S \)

- **Encoding \( R \)**
  - The OBDD for characteristic function \( f_{R_b} \) of \( R_b(<x>, <x'>) \)

- **Encoding \( L \)**
  - Typically, \( L \) is defined as mapping from states to subsets of atomic propositions
  - It is more convenient to consider it as mapping from atomic propositions to subsets of states
  - An atomic proposition \( p \) is mapped to the set of states that satisfy it: \( L_p = \{ s \mid p \in L(s) \} \)
  - \( L_p \) is represented using the encoding \( \phi \)
Representing Kripke structures using OBDDs

\[ R: (\neg x \land x') \lor (x \land x') \lor (x \land \neg x') \]

\[ L: a \rightarrow \{s_1, s_2\}, \quad b \rightarrow \{s_1\} \]

\{((0,0), (0,1), (1,0))\}