5.3 Symbolic Model Checking
**Last week:**

(Explicit) Model Checking

- **Benefit:**
  - Counterexample when a property is not fulfilled
- **Limitations:**
  - Not feasible for too large models (state explosion)
  - Not feasible for too complex formal properties

- It is always true..
- It is never true…

Temporal logic formula (e.g. CTL, LTL, …)
Symbolic Model Checking - Idea

- Don’t consider single states but state sets
- Use a compact representation of Kripke structures – BDDs (binary decision diagrams)
- Verification of systems with up to $10^{20}$ states
Symbolic Model Checking - Overview

- Uses very efficient data structure for representing Kripke structures
- Kripke structures represented as a BDD encoding a boolean formula
  (Don’t confuse these formulas with CTL-formulas!)

- Fast operations on BDDs for
  - Equivalence checking
  - Boolean operations

- MC algorithm works on this **symbolic** representation of the Kripke structure
Binary Decision Diagrams (BDDs)

\[(a \land b) \lor (c \land d)\]
BDD: Definition

- BDD: directed acyclic graph with a unique root node and two kinds of nodes
  - Terminal nodes, either mapped to 0 or 1
  - Non-terminal nodes, mapped to a variable v and has two successor nodes
Properties/Advantages of BDDs

1. Canonical representation of boolean functions, i.e. 2 boolean functions are equivalent iff their representations are equivalent (given the same variable order)

2. Efficient basic operations (intersection, union, comparison, complement, etc.)

3. Not restricted to a specific “family” of automata, applicable to all kinds of finite state systems
**BDDs for Kripke structures**

**Reminder:** A Kripke Structure is a 5-Tupel \((\mathcal{AP}, S, L, T, S_0)\):
- \(\mathcal{AP}\): Set of atomic propositions
- \(S\): finite set of states
- \(L: S \rightarrow \mathcal{P}(\mathcal{AP})\), labels
- \(T \subseteq S \times S\), left-total transition relation
- \(S_0\): set of initial states

**Basic Idea:**
- Encode each of the components (states, transitions, atomic propositions) as a *boolean function*.
- Represent boolean functions as BDDs.

**Boolean function:**
- Maps \(k\) bits to one bit
- e.g. \(k = 2\): \(f: \{0, 1\}^2 \rightarrow \{0, 1\}\), \(f(x_0, x_1) = \neg x_0 \land \neg x_1 \lor \neg x_0 \land x_1 \lor x_0 \land \neg x_1\)
- \(f(0,0) = 1, f(0,1) = 1, f(1,0) = 1, \text{ but } f(1,1) = 0\)
Encoding of states

1st step: Encode states as bit-strings e.g.
Encoding of states

1st step: Encode states as bit-strings e.g.

- $s_0$: 00
Encoding of states

1st step: Encode states as bit-strings e.g.

- s₀: 00

2nd step: Construct boolean function that is true (maps to 1) iff the bit-string passed as an argument represents one of the states, i.e.
**Encoding of states**

1st step: Encode states as bit-strings e.g.

- $s_0$: 00

2nd step: Construct boolean function that is true (maps to 1) iff the bit-string passed as an argument represents one of the states, i.e.

$$f_S(x_0, x_1) = (\neg x_0 \land \neg x_1)$$

The diagram shows the states and transitions:

- $s_0$: 00
- $s_1$: $p, q$
- $s_2$: $q$
Encoding of states

1st step: Encode states as bit-strings e.g.
- $s_0$: 00
- $s_1$: 01
- $s_2$: 10

2nd step: Construct boolean function that is true (maps to 1) iff the bit-string passed as an argument represents one of the states, i.e.

$$f_S(x_0, x_1) = \left( \neg x_0 \land \neg x_1 \right) \lor \left( \neg x_0 \land x_1 \right) \lor \left( x_0 \land \neg x_1 \right)$$

with:
- $S_0$ for state $s_0$
- $S_1$ for state $s_1$
- $S_2$ for state $s_2$
Encoding of initial states

1st step: Use encoding of states also for initial states:

- $s_0: 00$

2nd step: Construct boolean function that is true (maps to 1) iff the bit-string passed as an argument represents one of the initial states, i.e.

$$f_{S_0}(x_0, x_1) = \overline{x_0} \land \overline{x_1}$$
Encoding of transitions

1st step: For each transition \((s_i, s_j)\) concatenate the bit-string of \(s_i\) with the bit-string of \(s_j\), i.e.
Encoding of transitions

1st step: For each transition \((s_i, s_j)\) concatenate the bit-string of \(s_i\) with the bit-string of \(s_j\), i.e.

- \((s_0, s_1)\): 0001
**Encoding of transitions**

1st step: For each transition \((s_i, s_j)\) concatenate the bit-string of \(s_i\) with with the bit-string of \(s_j\), i.e.

- \((s_0, s_1)\): 0001

2nd step: Construct boolean function that is true iff the bit-string passed as an argument represents one of the transitions, i.e.
Encoding of transitions

**1st step:** For each transition \((s_i, s_j)\) concatenate the bit-string of \(s_i\) with the bit-string of \(s_j\), i.e.

- \((s_0, s_1): 0001\)

**2nd step:** Construct boolean function that is true iff the bit-string passed as an argument represents one of the transitions, i.e.

\[
f_T(x_0, x_1, x'_0, x'_1) = (\neg x_0 \land \neg x_1 \land \neg x'_0 \land x'_1)
\]
1st step: For each transition \((s_i, s_j)\) concatenate the bit-string of \(s_i\) with the bit-string of \(s_j\), i.e.

- \((s_0, s_1)\): 0001
- \((s_1, s_0)\): 0100
- \((s_1, s_2)\): 0110
- \((s_2, s_2)\): 1010

2nd step: Construct boolean function that is true iff the bit-string passed as an argument represents one of the transitions, i.e.

\[
f_T (x_0, x_1, x'_0, x'_1) = \left(\neg x_0 \land \neg x_1 \land \neg x'_0 \land x'_1\right) \lor \left(\neg x_0 \land x_1 \land \neg x'_0 \land \neg x'_1\right) \lor \left(\neg x_0 \land x_1 \land x'_0 \land \neg x'_1\right) \lor \left(x_0 \land \neg x_1 \land x'_0 \land \neg x'_1\right)
\]
1st step: For each atomic proposition $r$ collect bit encodings of states in which $r$ holds:
1st step: For each atomic proposition r collect bit encodings of states in which r holds:

- p: 00, 01 (representing s₀, s₁)
Encoding of atomic propositions and labeling function $L$

1st step: For each atomic proposition $r$ collect bit encodings of states in which $r$ holds:

- $p$: 00, 01 (representing $s_0$, $s_1$)

2nd step: Construct a boolean function $f_r$ that is true iff the bit-string passed as an argument represents one of the states in which $r$ holds, i.e.
Encoding of atomic propositions and labeling function $L$

1st step: For each atomic proposition $r$ collect bit encodings of states in which $r$ holds:

- $p$: 00, 01 (representing $s_0$, $s_1$)

2nd step: Construct a boolean function $f_r$ that is true iff the bit-string passed as an argument represents one of the states in which $r$ holds, i.e.

$$f_p(x_0, x_1) = (\neg x_0 \land \neg x_1) \lor (\neg x_0 \land x_1)$$
Encoding of atomic propositions and labeling function $L$

1st step: For each atomic proposition $r$ collect bit encodings of states in which $r$ holds:

- $p$: 00, 01 (representing $s_0$, $s_1$)
- $q$: 00, 10 (representing $s_0$, $s_2$)

2nd step: Construct a boolean function $f_r$ that is true iff the bit-string passed as an argument represents one of the states in which $r$ holds, i.e.

$$
\begin{align*}
    f_p(x_0, x_1) &= (\neg x_0 \land \neg x_1) \lor (\neg x_0 \land x_1) \\
    f_q(x_0, x_1) &= (\neg x_0 \land \neg x_1) \lor (x_0 \land \neg x_1)
\end{align*}
$$
Reduced Ordered BDDs

- Canonical representation
- Variable ordering the same on all paths from root to leaves
- No isomorphic subtrees, no redundant nodes
Reduced Ordered BDDs - Reduction

\[(a \land b) \lor (c \land d)\]

Order: a, b, c, d
Reduced Ordered BDDs - Reduction

\((a \land b) \lor (c \land d)\)
Order: a, b, c, d

1. Merge 0 and 1 nodes.
Reduced Ordered BDDs - Reduction

\[(a \land b) \lor (c \land d)\]
Order: a, b, c, d

1. Merge 0 and 1 nodes.
2. Merge d node (due to same output)
Reduced Ordered BDDs - Reduction

(a \land b) \lor (c \land d)

Order: a, b, c, d

1. Merge 0 and 1 nodes.
2. Merge d nodes (due to same output)
3. Merge c nodes (due to same output)
Reduced Ordered BDDs - Reduction

(a \land b) \lor (c \land d)
Order: a, b, c, d

1. Merge 0 and 1 nodes.
2. Merge d nodes (due to same output)
3. Merge c nodes (due to same output)
4. Delete b node (no real alternative)
Order of predicates

- Size of ROBDD depends on order of the predicates

Order: $a, b, c, d$

Order: $a, b, c, d$

$(a \land b) \lor (c \land d)$
ROBDDs

- ROBDD (Reduced Ordered BDD): BDD
  - With a fixed variable order, i.e. variables occur in the same order on all paths
  - Reduction of the binary diagram until no more reduction rules can be applied
ROBDD reduction rules

- Reduction rules
  1. Delete duplicate leafs
  2. Find and merge identical sub trees
  3. Remove redundant inner nodes. A node is redundant if there is no real decision possible, i.e. both outgoing edges have the same target node.

Symbolic Model Checking
Properties/Advantages of ROBDDs

1. Canonical representation of boolean functions, i.e. 2 boolean functions are equivalent iff their representations are equivalent (given the same variable order)

2. Efficient basic operations (intersection, union, comparison, complement, etc.)

3. Not restricted to a specific “family” of automata, applicable to all kinds of finite state systems
Symbolic Model Checking Algorithm

- Returns a BDD which represents the set of all states that satisfy $\varphi$

function check(CTL formula $\varphi$) : BDD

switch
  $\varphi$ is an atomic proposition $p$: return $f_p$
  $\varphi = \neg g$: return ApplyNot(check(g))
  $\varphi = g \lor h$: return ApplyOr(check(g), check(h))
  $\varphi = \text{EX } g$: return CheckEX(check(g))
  $\varphi = E[gUh]$: return CheckEU(check(g), check(h))
  $\varphi = EG g$: return CheckEG(check(g))
end switch
end check

- Computation for CheckEU and CheckEG based on fixpoint computation (lecture Model Checking)
Symbolic Model Checking
Algorithm: Visualization

- Example: $\text{EX} (p \lor \text{EG} \neg q)$
- Function check() realizes recursion into the tree until leaves are reached
- From the leaves on the BBD $f_q$ of a sub-formula $q$ is propagated up the tree for further processing
- $\text{CheckEX}()$ and $\text{CheckEG}()$ use $f_T$ (BDD of transition relation) to reach successor states of a state $s$

```
Symbolic Model Checking
Algorithm: Visualization
```

```
Example: EX (p \lor EG \neg q)

- Function check() realizes recursion into the tree until leaves are reached
- From the leaves on the BBD $f_q$ of a sub-formula $q$ is propagated up the tree for further processing
- $\text{CheckEX}()$ and $\text{CheckEG}()$ use $f_T$ (BDD of transition relation) to reach successor states of a state $s$
```
References


- **Model Checking.** E.Clarke, O.Grumberg, D.Peled. MIT Press

- **Model Checking.** Vorlesung bei H.Wehrheim