

Chapter 4

CTL

Course “Specification and Verification of Parallel Systems”

Tobias John

Autumn 2023

Plan for today

- repeat basic CTL MC algorithm
- introduce symbolic model checking
- efficient data structures for symbolic MC
  - ROBDDs





Section

Repetition: Basic Algorithm

Autumn 2023

Basic algorithm

- recursive algorithm on structure of Φ
- $\exists\Box$ and $\exists\ U$ need special treatment (fixpoint algorithms)

```
1  input:  finite transition system T and Φ
2  output: Sat(Φ)
3  ------------------------------------------------------------
4  switch Φ
5      ⊤              => return Q
6      p              => return {q ∈ Q | p ∈ L(q)}
7      ¬Φ             => return Q \ Sat(Φ)
8      Φ1 ∧ Φ2        => return Sat(Φ1) ∩ Sat(Φ2)
9      ∃◯Φ            => return {q ∈ Q | post(s) ∩ Sat(Φ) ≠ ∅}
10     ∃(Φ1 U Φ2)     => ``lfp for ∃ U''
11     ∃□Φ            => ``gfp for ∃□''
```

- Remark: $post(s) = \{s' \in Q \mid s \to s'\}$

Fixpoint algorithm for $\exists\ U$

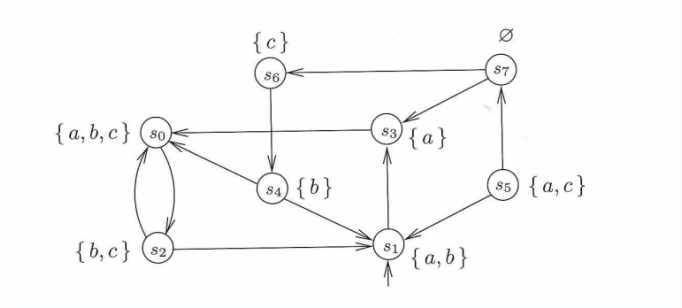
- Remember:
- $\mathrm{Sat}(\exists(\Phi_1\ U\ \Phi_2))$ is the *smallest* subset R of Q such that
  1. $\mathrm{Sat}(\Phi_2) \subseteq R$ and
  2. $q \in \mathrm{Sat}(\Phi_1)$ and $\exists s' \in R. s \to s'$ implies $s \in R$.

```
1  R := Sat(Φ2);
2  while {q ∈ Sat(Φ1) \ R | post(q) ∩ R ≠ ∅} ≠ ∅ do
3     let q ∈ {q ∈ Sat(Φ1) \ R | post(q) ∩ R ≠ ∅};
4     R := R ∪ {q};
5  od;
6  return R;
```

hi

Example: until operator

$$\Phi = \exists(\top\ U\ (a = c) \wedge (a \neq b))$$

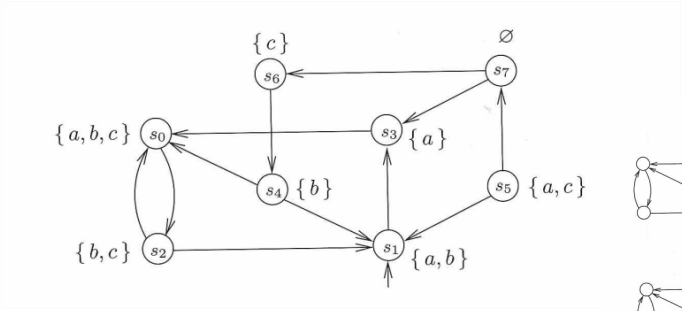


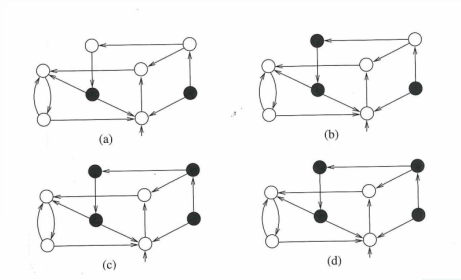

```
1  R := Sat(Φ2);
2  while {q ∈ Sat(Φ1) \ R | post(q) ∩ R ≠ ∅} ≠ ∅ do
3     let q ∈ {q ∈ Sat(Φ1) \ R | post(q) ∩ R ≠ ∅};
4     R := R ∪ {q};
5  od;
6  return R;
```



Example: until operator

$$\Phi = \exists(\top\ U\ (a = c) \wedge (a \neq b))$$






```
1  R := Sat(Φ2);
2  while {q ∈ Sat(Φ1) \ R | post(q) ∩ R ≠ ∅} ≠ ∅ do
3     let q ∈ {q ∈ Sat(Φ1) \ R | post(q) ∩ R ≠ ∅};
4     R := R ∪ {q};
5  od;
6  return R;
```



Section

Explicit vs. Symbolic state

Autumn 2023

Symbolic state

- Observation: mainly operations on sets of states in CTL MC algorithm
- Idea:
  - do computations directly on sets
  - i.e. do not represent states individually
  - representation of set is way smaller that representing its states
- Advantage:
  - systems with way more states can be investigated



Symbolic MC: $\exists\Box\Phi$

- Remember:
- $\mathrm{Sat}(\exists\Box\Phi)$ is the *largest* subset R of Q such that
  1. $R \subseteq \mathrm{Sat}(\Phi)$ and
  2. $q \in R$ implies $\exists q' \in R.q \to q'$.

```
1  R := Sat(Φ);
2  while {q ∈ R | post(q) ∩ R = ∅} ≠ ∅ do
3      let q ∈ {q ∈ R | post(q) ∩ R = ∅};
4      R := R \ {q};
5  od;
6  return R;
```



Symbolic MC: $\exists\Box\Phi$

- Remember:
- $\mathrm{Sat}(\exists\Box\Phi)$ is the *largest* subset R of Q such that
  1. $R \subseteq \mathrm{Sat}(\Phi)$ and
  2. $q \in R$ implies $\exists q' \in R.q \to q'$.

```
1  R := Sat(Φ);
2  while {q ∈ R | post(q) ∩ R = ∅} ≠ ∅ do
3      let q ∈ {q ∈ R | post(q) ∩ R = ∅};
4      R := R \ {q};
5  od;
6  return R;
```

- symbolic notation:

```
1  R := Sat(Φ);
2  repeat
3      U := {q ∈ R | post(q) ∩ R = ∅};
4      R := R \ U;
5  until U = ∅;
6  return R;
```



Symbolic model checking: Requirement

- we need efficient representations / algorithms for the following:
  - sets of states
  - transition systems
  - operations on the sets of states (e.g. union, equivalence)
  - combining sets of states and transitions





Section

Switching Functions

Autumn 2023

Mathematical Structure: switching functions

“bit-vectors”

$$\text{Eval}(Var) = Var \to \{0,1\} \ .$$



Mathematical Structure: switching functions

"bit-vectors"

$$\text{Eval}(\mathit{Var}) = \mathit{Var} \to \{0,1\} \ .$$

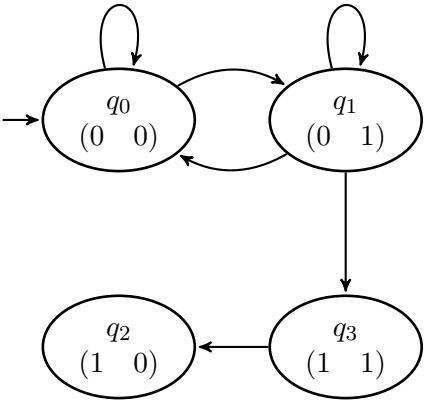
Switching function

$$f : \text{Eval}(\mathit{Var}) \to \{0,1\} = (\mathit{Var} \to \{0,1\}) \to \{0,1\}$$

- mapping from variable assignment to bit
- used to encode sets of states and transitions

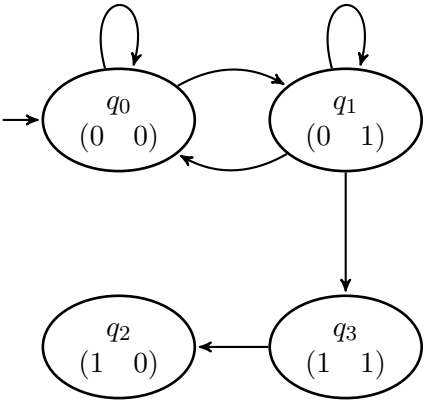


Encoding of state sets



- encode states with boolean variables $(x, y, \ldots)$
  i.e. $Q = \mathrm{Eval}(\vec{x})$
- $R = \{q_0, q_1\}$:

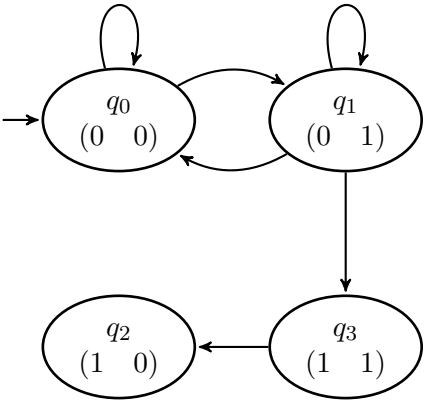
Encoding of state sets



- encode states with boolean variables $(x, y, \ldots)$
  i.e. $Q = \mathrm{Eval}(\vec{x})$
- $R = \{q_0, q_1\}$: $\neg x$
- $R = \{q_1, q_2\}$:

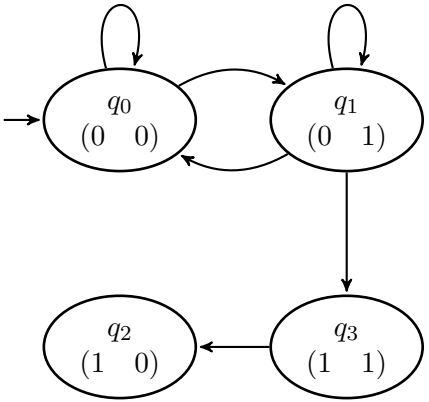


Encoding of state sets



- encode states with boolean variables $(x, y, \ldots)$
  i.e. $Q = \mathrm{Eval}(\vec{x})$
- $R = \{q_0, q_1\}$: $\neg x$
- $R = \{q_1, q_2\}$: $(x = 0) \leftrightarrow (y = 1)$

Encoding of state sets



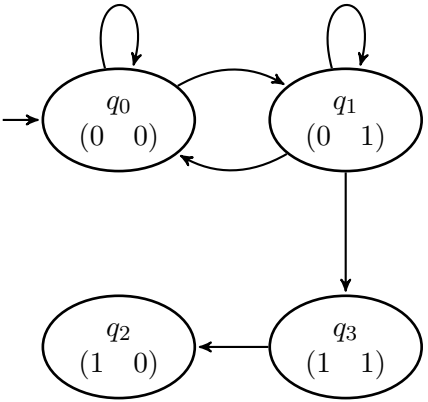
- encode states with boolean variables $(x, y, \ldots)$
  i.e. $Q = \mathrm{Eval}(\vec{x})$
- $R = \{q_0, q_1\}$: $\neg x$
- $R = \{q_1, q_2\}$: $(x = 0) \leftrightarrow (y = 1)$

Characteristic function for $R \subseteq Q = \mathrm{Eval}(\vec{x})$

$\chi_R : \mathrm{Eval}(\vec{x}) \to \{0, 1\}$ with $\chi_R(q) =$



Encoding of state sets

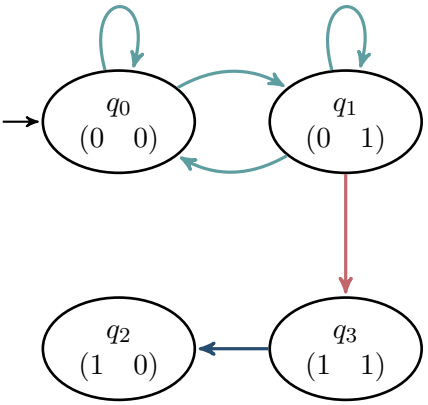


- encode states with boolean variables $(x, y, \ldots)$
  i.e. $Q = \mathrm{Eval}(\vec{x})$
- $R = \{q_0, q_1\}$: $\neg x$
- $R = \{q_1, q_2\}$: $(x = 0) \leftrightarrow (y = 1)$

Characteristic function for $R \subseteq Q = \mathrm{Eval}(\vec{x})$

$$\chi_R : \mathrm{Eval}(\vec{x}) \to \{0,1\} \quad \text{with} \quad \chi_R(q) = \begin{cases} 1 & \text{if} \quad q \in R \\ 0 & \text{else} \end{cases}$$

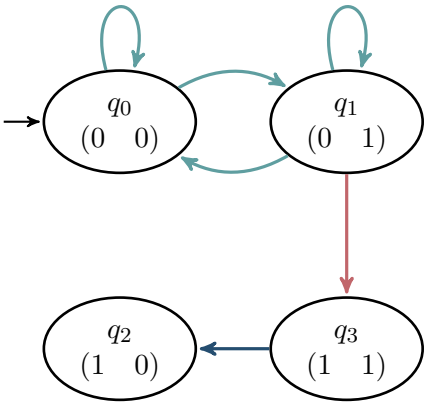
Encoding of transition functions



- remember: $Q = \mathrm{Eval}(\vec{x})$
- switching function for $\rightarrow$ is $\Delta : \mathrm{Eval}(\vec{x}, \vec{x}') \rightarrow \{0, 1\}$

$\Delta(x, y, x', y') =$

Encoding of transition functions

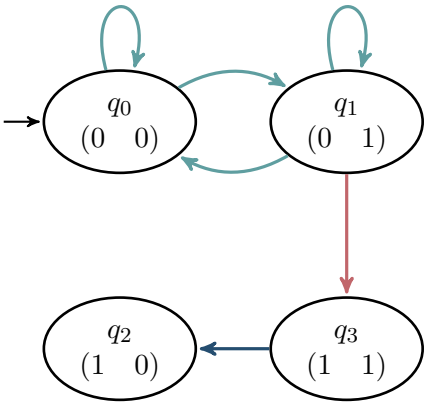


- remember: $Q = \mathrm{Eval}(\vec{x})$
- switching function for $\rightarrow$ is $\Delta : \mathrm{Eval}(\vec{x}, \vec{x}') \rightarrow \{0, 1\}$

$$\Delta(x, y, x', y') = (\neg x \wedge y \wedge x' \wedge y') \vee (x \wedge y \wedge x' \wedge \neg y') \vee$$



Encoding of transition functions



- remember: $Q = \mathrm{Eval}(\vec{x})$
- switching function for $\rightarrow$ is $\Delta : \mathrm{Eval}(\vec{x}, \vec{x}') \rightarrow \{0, 1\}$

$$\Delta(x, y, x', y') = (\neg x \wedge y \wedge x' \wedge y') \vee (x \wedge y \wedge x' \wedge \neg y') \vee (\neg x \wedge \neg x')$$

Encoding the transition relation $\rightarrow$

$$\rightarrow \subseteq Q \times Q$$

- make a "copy" of $\vec{x}$: different variables $\vec{x}'$
- renaming operation on switching functions $[y \leftarrow x]$

Encode $\rightarrow$ as Δ

$$\Delta : \mathrm{Eval}(\vec{x}, \vec{x}') \rightarrow \{0, 1\},$$

$$\Delta(q_1, q_2[\vec{x}' \leftarrow \vec{x}]) = \begin{cases} 1 & \text{if } q_1 \rightarrow q_2 \\ 0 & \text{else} \end{cases}$$



Operations on switching functions (disjunction, conjunction)

- f_1 is a switching function for $\{x_1, \ldots, x_n, \ldots, x_m\}$
- f_2 is a switching function for $\{x_n, \ldots, x_m, \ldots, x_k\}$



Operations on switching functions (disjunction, conjunction)

- f_1 is a switching function for $\{x_1, \ldots, x_n, \ldots, x_m\}$
- f_2 is a switching function for $\{x_n, \ldots, x_m, \ldots, x_k\}$

Disjunction

$$(f_1 \vee f_2)([x_1 = a_1, \ldots, x_k = a_k]) = \max(f_1([x_1 = a_1, \ldots, x_m = a_m]), f_2([x_n = a_n, \ldots, x_k = a_k]))$$

Conjunction



Operations on switching functions (disjunction, conjunction)

- f_1 is a switching function for $\{x_1, \ldots, x_n, \ldots, x_m\}$
- f_2 is a switching function for $\{x_n, \ldots, x_m, \ldots, x_k\}$

Disjunction

$$(f_1 \vee f_2)([x_1 = a_1, \ldots, x_k = a_k]) = \max(f_1([x_1 = a_1, \ldots, x_m = a_m]), f_2([x_n = a_n, \ldots, x_k = a_k]))$$

Conjunction

$$(f_1 \wedge f_2)([x_1 = a_1, \ldots, x_k = a_k]) = \min(f_1([x_1 = a_1, \ldots, x_m = a_m]), f_2([x_n = a_n, \ldots, x_k = a_k]))$$



Operations on switching functions (existential quantification)

Definition (Cofactors)

Assume a variable set $Var = \{x, y_1, \ldots, y_m\}$ and let $f : \mathrm{Eval}(Var) \to \{0, 1\}$ be a switching function over it. The *positive cofactor* of f for variable x, written $f\,|_{x=1}$ is the switching function given by

$$f\,|_{x=1}\,(a_1, \ldots, a_m) = f(1, a_1, \ldots, a_m)$$

The *negative cofactor* of f for x, written $f\,|_{x=0}$ is defined analogously.



Operations on switching functions (existential quantification)

Definition (Cofactors)

Assume a variable set $Var = \{x, y_1, \ldots, y_m\}$ and let $f : \mathrm{Eval}(Var) \to \{0, 1\}$ be a switching function over it. The *positive cofactor* of f for variable x, written $f\,|_{x=1}$ is the switching function given by

$$f\,|_{x=1}\,(a_1, \ldots, a_m) = f(1, a_1, \ldots, a_m)$$

The *negative cofactor* of f for x, written $f\,|_{x=0}$ is defined analogously.

Definition (Existential quantification)

Let f be a switching function for Var and $x \in Var$.
$\exists x.f \quad =$



Operations on switching functions (existential quantification)

Definition (Cofactors)

Assume a variable set $Var = \{x, y_1, \ldots, y_m\}$ and let $f : \mathrm{Eval}(Var) \to \{0, 1\}$ be a switching function over it. The *positive cofactor* of f for variable x, written $f\,|_{x=1}$ is the switching function given by

$$f\,|_{x=1}\,(a_1, \ldots, a_m) = f(1, a_1, \ldots, a_m)$$

The *negative cofactor* of f for x, written $f\,|_{x=0}$ is defined analogously.

Definition (Existential quantification)

Let f be a switching function for Var and $x \in Var$.
$\exists x.f \quad = \quad f\,|_{x=1} \vee f\,|_{x=0}$

- both definitions can be extended to tuples of variables $\vec{x} = (x_1, \ldots, x_k)$



MC with switching functions: next

- Remember:
  $\mathrm{Sat}(\exists \bigcirc \Phi) = \{q \in Q \mid \exists q'.q \to q' \wedge q' \in \mathrm{Sat}(\Phi)\}$
- Algorithm:
  $\mathrm{Sat}(\exists \bigcirc \Phi) = \{q \in Q \mid post(s) \cap \mathrm{Sat}(\Phi) \neq \emptyset\}$
- Using switching functions:



MC with switching functions: next

- Remember:
  $\mathrm{Sat}(\exists \bigcirc \Phi) = \{q \in Q \mid \exists q'.q \to q' \wedge q' \in \mathrm{Sat}(\Phi)\}$
- Algorithm:
  $\mathrm{Sat}(\exists \bigcirc \Phi) = \{q \in Q \mid post(s) \cap \mathrm{Sat}(\Phi) \neq \emptyset\}$
- Using switching functions:
  $\chi_{\mathrm{Sat}(\exists \bigcirc \Phi)}(\vec{x}) = \exists \vec{x}'.(\Delta(\vec{x}, \vec{x}') \wedge \chi_{\mathrm{Sat}(\Phi)}(\vec{x}'))$



MC with switching functions: always

- Remember:
- $\mathrm{Sat}(\exists \Box \Phi)$ is the *largest* subset R of Q such that
  1. $R \subseteq \mathrm{Sat}(\Phi)$ and
  2. $q \in R$ implies $\exists q' \in R. q \to q'$.

```
1  R := Sat(Φ);
2  repeat
3      U := {q ∈ R | post(q) ∩ R = ∅};
4      R := R \ U;
5  until U = ∅;
6  return R;
```



MC with switching functions: always

- Remember:
- $\mathrm{Sat}(\exists \Box \Phi)$ is the *largest* subset R of Q such that
  1. $R \subseteq \mathrm{Sat}(\Phi)$ and
  2. $q \in R$ implies $\exists q' \in R. q \to q'$.

```
1  R := Sat(Φ);
2  repeat
3      U := {q ∈ R | post(q) ∩ R = ∅};
4      R := R \ U;
5  until U = ∅;
6  return R;
```

- equivalent (symbolic) algorithm:

```
1  R_0 := Sat(Φ);
2  j := 0;
3  repeat
4     R_{j+1} := R_j ∩ {q ∈ Q | post(q) ∩ R_j ≠ ∅};
5     j := j + 1;
6  until R_j = R_{j-1};
7  return R_j.
```



MC with switching functions: always

- Remember:
- $\mathrm{Sat}(\exists\Box\Phi)$ is the *largest* subset R of Q such that
  1. $R \subseteq \mathrm{Sat}(\Phi)$ and
  2. $q \in R$ implies $\exists q' \in R.q \to q'$.

```
1  R_0 := Sat(Φ);
2  j := 0;
3  repeat
4     R_{j+1} := R_j ∩ {q ∈ Q | post(q) ∩ R_j ≠ ∅};
5     j := j + 1;
6  until R_j = R_{j-1};
7  return R_j.
```

MC with switching functions: always

- Remember:
- $\mathrm{Sat}(\exists\Box\Phi)$ is the *largest* subset R of Q such that
  1. $R \subseteq \mathrm{Sat}(\Phi)$ and
  2. $q \in R$ implies $\exists q' \in R.q \to q'$.

```
1  R_0 := Sat(Φ);
2  j := 0;
3  repeat
4     R_{j+1} := R_j ∩ {q ∈ Q | post(q) ∩ R_j ≠ ∅};
5     j := j + 1;
6  until R_j = R_{j-1};
7  return R_j.
```

- Let $A = \mathrm{Sat}(\Phi)$

```
1  f_0(x⃗) := χ_A(x⃗);
2  j := 0;
3  repeat
4     f_{j+1}(x⃗) := f_j(x⃗) ∧ ∃x⃗'.(Δ(x⃗, x⃗') ∧ f_j(x⃗'));
5     j := j + 1;
6  until f_j(x⃗) = f_{j-1}(x⃗);
7  return f_j(x⃗).
```



Requirements for data structure

- Goal: implementation for switching functions
- Should support (efficient)
  - representation of the function
  - boolean operations
  - equivalence check (for termination of fixpoint algorithms)
- Candidates?
  - truth tables?



Requirements for data structure

- Goal: implementation for switching functions
- Should support (efficient)
  - representation of the function
  - boolean operations
  - equivalence check (for termination of fixpoint algorithms)
- Candidates?
  - truth tables? $\rightarrow$ always exponential in number of variables
  - CNFs or DNFs?



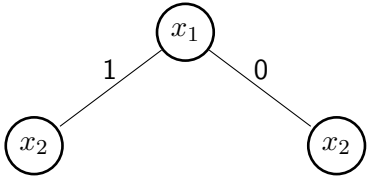
Requirements for data structure

- Goal: implementation for switching functions
- Should support (efficient)
  - representation of the function
  - boolean operations
  - equivalence check (for termination of fixpoint algorithms)
- Candidates?
  - truth tables? $\rightarrow$ always exponential in number of variables
  - CNFs or DNFs? $\rightarrow$ checking equivalence is hard (coNP-complete)
- Solution: binary decision trees
  - in particular: OBDDs (also known as ROBDDs)



Binary decision tree

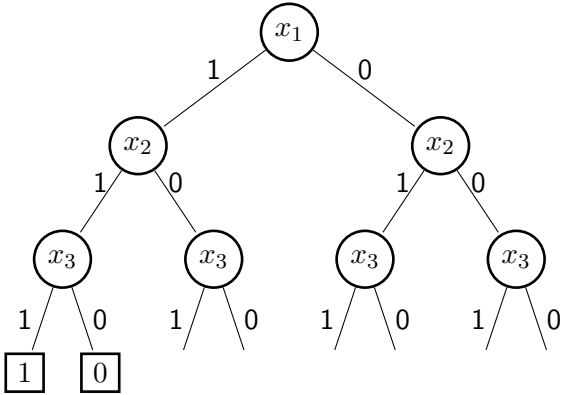
- nodes contain variables
- branching contains value of variable
- leaves contain 1 or 0
- consider $f = x_1 \wedge (\neg x_2 \vee x_3)$





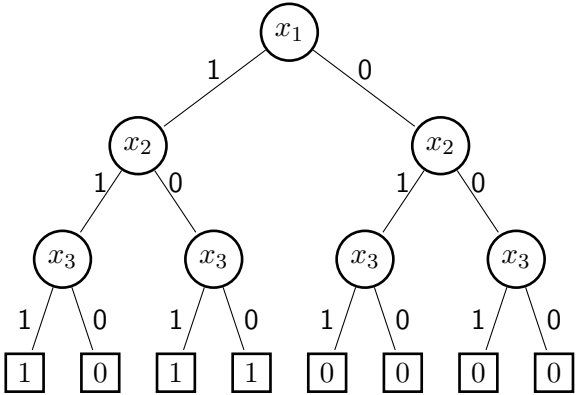
Binary decision tree

- nodes contain variables
- branching contains value of variable
- leaves contain 1 or 0
- consider $f = x_1 \wedge (\neg x_2 \vee x_3)$



Binary decision tree

- nodes contain variables
- branching contains value of variable
- leaves contain 1 or 0
- consider $f = x_1 \wedge (\neg x_2 \vee x_3)$



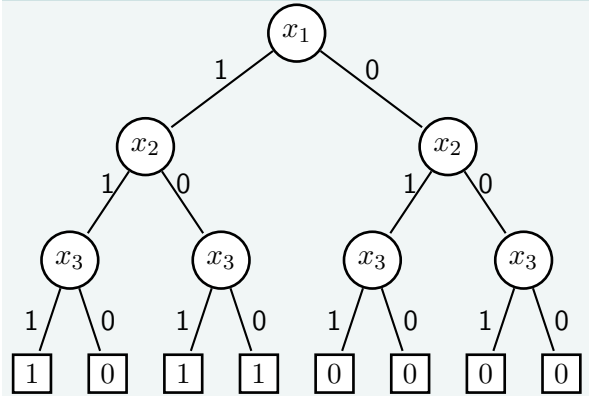
OBDD: ordered binary decision diagram

- similar to binary decision trees but often smaller
- fix ordering of variables (i.e., same ordering in different sub-trees)
  - important to make operations efficient
- allow to reduce size
  - remove unnecessary nodes
- nice effect: represent all functions (efficiently) in one diagram (multiple different roots)



OBDD: Example

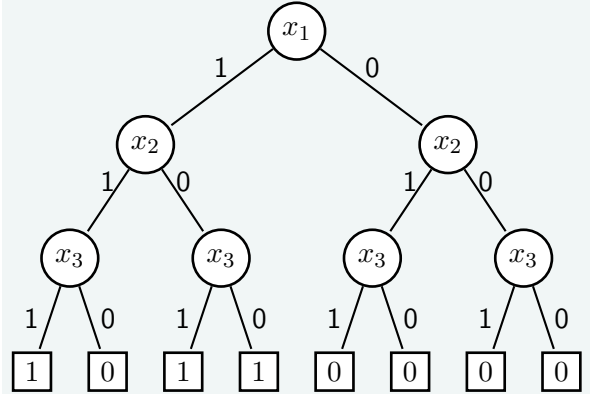
Decision tree



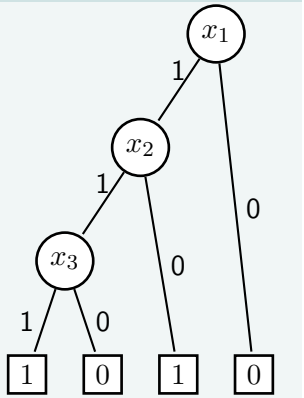
an equivalent OBDD

OBDD: Example

Decision tree



an equivalent OBDD



OBDDs: reductions

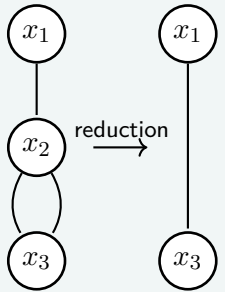
- usually: consider OBDDs that can not be reduced further
- each switching function has one *unique* such OBDD
- example reduction rules include (but are not limited to):



OBDDs: reductions

- usually: consider OBDDs that can not be reduced further
- each switching function has one *unique* such OBDD
- example reduction rules include (but are not limited to):

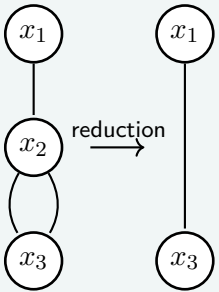
Reduction 1



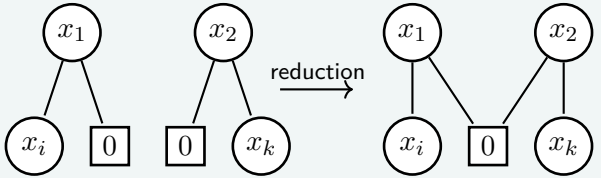
OBDDs: reductions

- usually: consider OBDDs that can not be reduced further
- each switching function has one *unique* such OBDD
- example reduction rules include (but are not limited to):

Reduction 1

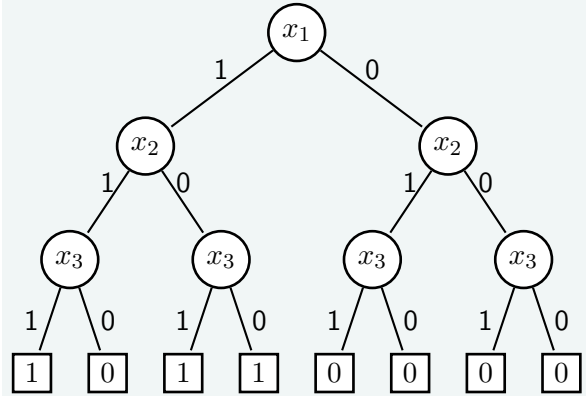


Reduction 2



Reduced OBDD: Example

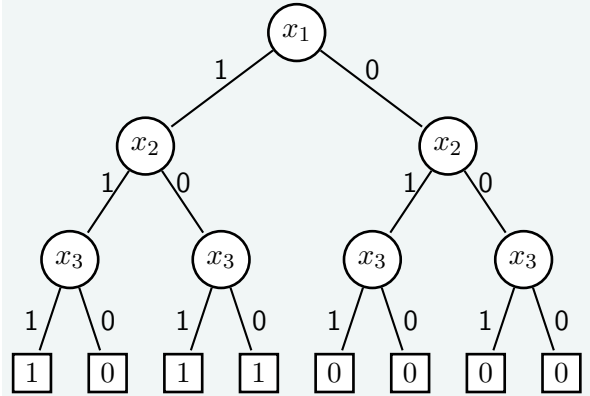
Decision tree



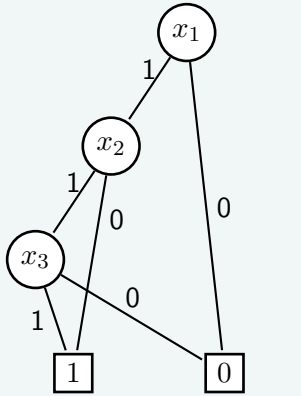
reduced OBDD

Reduced OBDD: Example

Decision tree



reduced OBDD



Summary: OBDDs

- related to binary decision trees, but reduce size by simplifying redundant parts
- compact representation of switching functions (although exponential in worst case)
- efficient implementations for operations exist:
  - conjunction, disjunction, quantification: linear time
  - equivalence check: constant time
- good data structure for symbolic CTL MC



Summary

- CTL MC algorithm: recursion on structure of formula
- symbolic model checking to handle huge systems
- efficient data structures for symbolic MC:
  - mathematical model: switching functions
  - implementation: OBDDs
- next week:
  - student talk
  - a bit CTL* MC



References I

Bibliography

