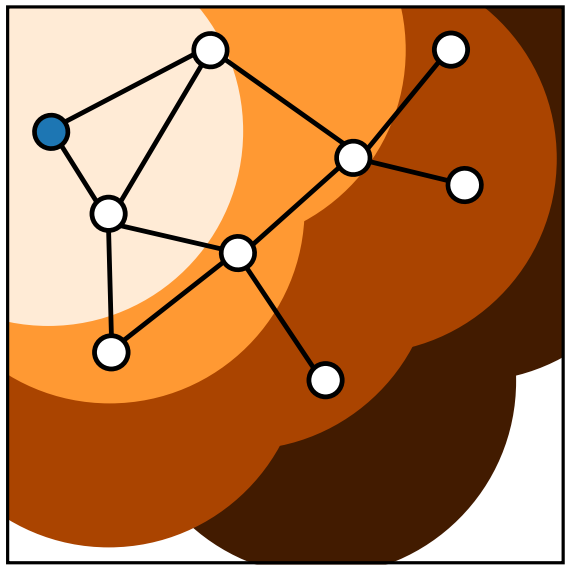
Discrete Event Systems
Verification of Finite Automata (Part 1)



Romain Jacob

www.romainjacob.net

ETH Zurich (D-ITET)

November 25, 2021

Most materials from Lothar Thiele

Who am I?

Montenvers, Chamonix, Oct. 2021

Post-doc since ~1.5y
with Prof. Vanbever

PhD from ETH
with Prof. Thiele

Used to TA this course
Master in Control of DES

Grew up in Bordeaux, France
Father of a 1-year old

What are finite automata useful for?

What are finite automata useful for?

specification

- Digital circuits
- Protocols (e.g. BGP)

simulation

- Anything specified with automata

synthesis of software or hardware

- UML
- Network configurations

What are finite automata useful for?

verification

specification

- Digital circuits
- Protocols (e.g. BGP)

simulation

- Anything specified with automata

synthesis of software or hardware

- UML
- Network configurations

Verification of Finite Automata

Questions:

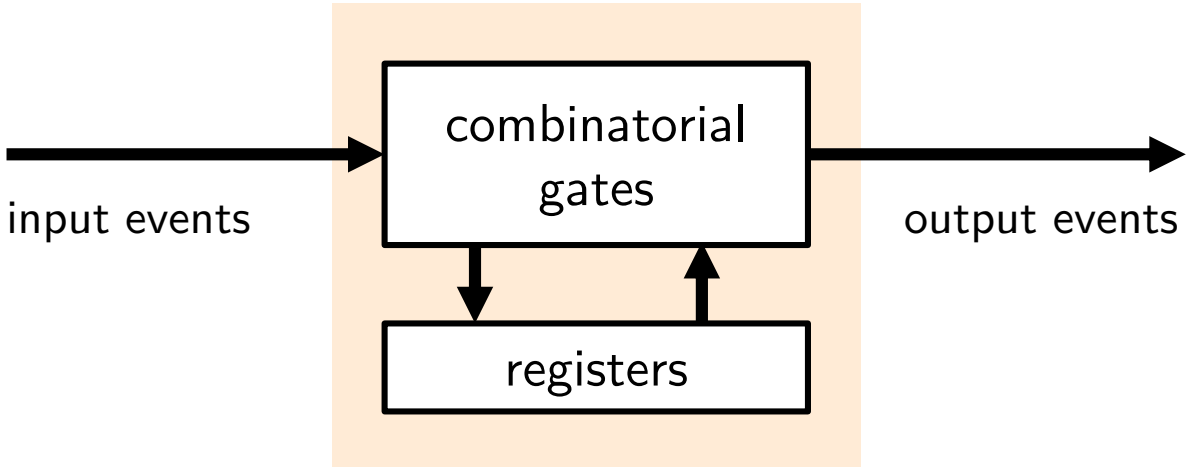
- Does the system specification model the desired behavior correctly?
- Do implementation and specification describe the same behavior?
- Can the system enter an undesired (or dangerous) state?

Possible solutions:

- Simulation (sometimes also called validation or testing): Unless the simulation is exhaustive, i.e., all possible input sequences are tested, the result is not trustworthy. In general, simulation can only show the presence of errors but not the absence (correctness).
- Formal analysis (sometimes also called verification): Formal (unambiguous) proof of correctness.

Verification of Finite Automata

- Due to the finite number of states, proving properties of a finite state machine can be done by enumeration.
- As computer systems have finite memory, properties of processors (and embedded systems in general) could be shown in principle.
- But is enumeration a reasonable approach in practice?



finite automaton

memory	number of states
8 Bit	256
32 Bit	4.10^9
1KBit	10^{300}
1MBit	$10^{300\ 000}$
1GBit	$10^{300\ 000\ 000}$

atoms in the universe is about 10^{82}

Verification of Finite Automata

- There have been major breakthroughs in recent years on the verification of finite automata with very large state spaces. Prominent methods are based on
  - symbolic model checking via binary decision diagrams (covered in this course) and
  - transformation to a Boolean Satisfiability (SAT) problem (not covered in this course).

- Symbolic model checking is a method of verifying temporal properties of finite (and sometimes infinite) state systems that relies on a symbolic representation of sets, typically as Binary Decision Diagrams (BDD's).

- Verification is used in industry for proving the correctness of complex digital circuits (control, arithmetic units, cache coherence), safety-critical software and embedded systems (traffic control, train systems, security protocols).

Finite Automata (with output)

A finite automaton (FA) with output is a 5-tuple $(Q, \Sigma, \delta, \omega, q_0)$ where

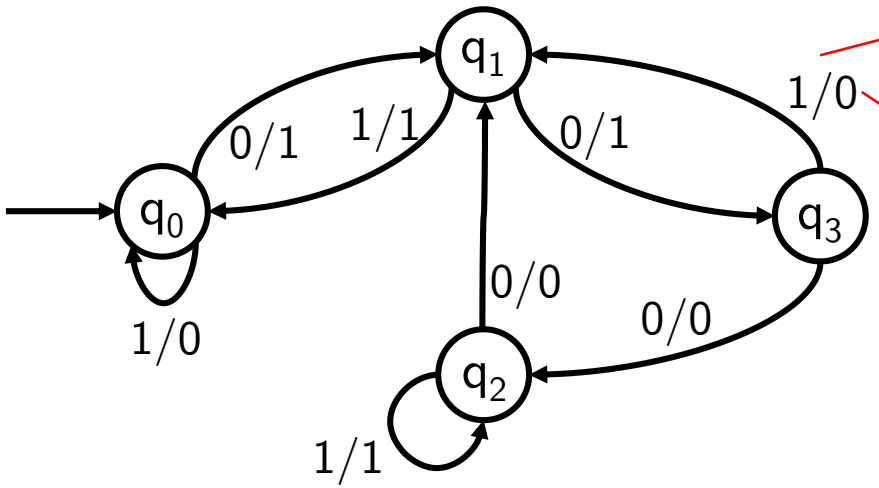
Q is a finite set called the states,

Σ is a finite set called the alphabet,

$\delta : Q \times \Sigma \to Q$ is the transformation function,

$\omega : Q \times \Sigma \to \Sigma$ is the output function, and

$q_0 \in Q$ is the start state.



$\delta(q_3, 1) = q_1$

$\omega(q_3, 1) = 0$

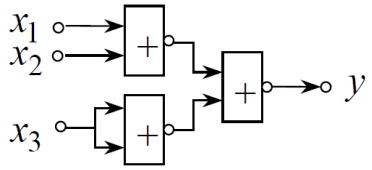
$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

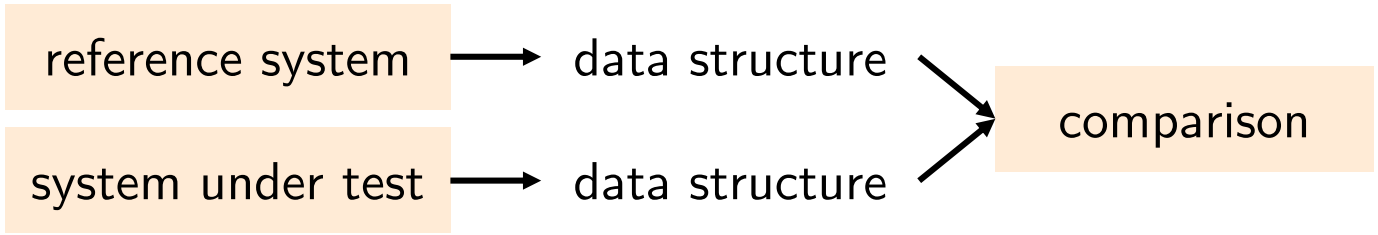
Verification Scenarios

Example

$y = (x_1 + x_2) \cdot x_3$

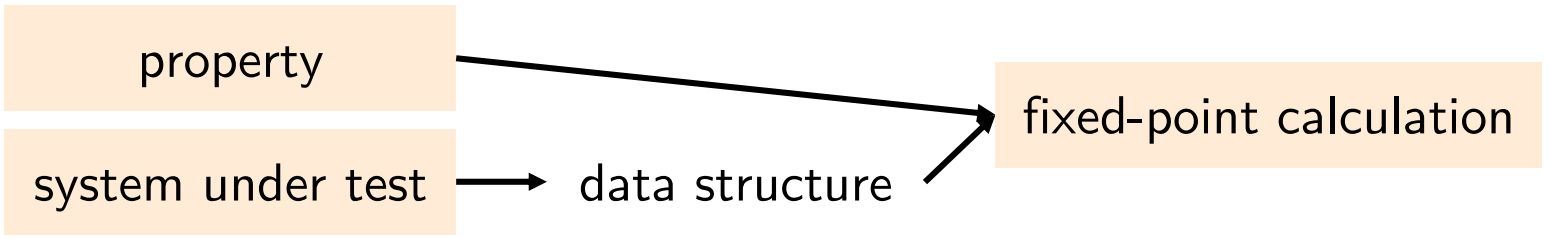


Comparison of specification and implementation



Proving properties

"The device can always be switched off."



Efficient state representation	▪ Set of states as Boolean function ▪ Binary Decision Diagram representation
Computing reachability	▪ Leverage efficient state representation ▪ Explore successor sets of states
Proving properties	▪ Temporal logic (CTL) ▪ Encoding as reachability problem

This week

Efficient state representation

- Set of states as Boolean function
- Binary Decision Diagram representation

Computing reachability

- Leverage efficient state representation
- Explore successor sets of states

Proving properties

- Temporal logic (CTL)
- Encoding as reachability problem

Efficient state representation

- Set of states as Boolean function
- Binary Decision Diagram representation

Computing reachability

- Leverage efficient state representation
- Explore successor sets of states

Proving properties

- Temporal logic (CTL)
- Encoding as reachability problem

Basic concept of verification using BDDs

- BDDs represent Boolean functions.
- Therefore, they can be used to describe sets of states and transformation relations.
- Due to the unique representation of Boolean functions, reduced ordered BDDs (ROBDD) can be used to proof equivalence between Boolean functions or between sets of states.
- BDDs can easily and efficiently be manipulated.

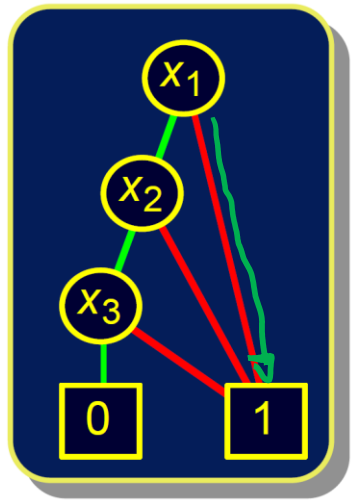
Binary Decision Diagrams (BDD)

- Concept
  - Data structure that allows to represent Boolean functions.
  - The representation is unique for a given ordering of variables. If the ordering of variables is fixed, we call it an ordered BDD (OBDD).

- Structure
  - BDDs contain "decision nodes" which are labeled with variable names.
  - Edges are labeled with input values.
  - Leaves are labeled with output values.

$f(1,0,1)$

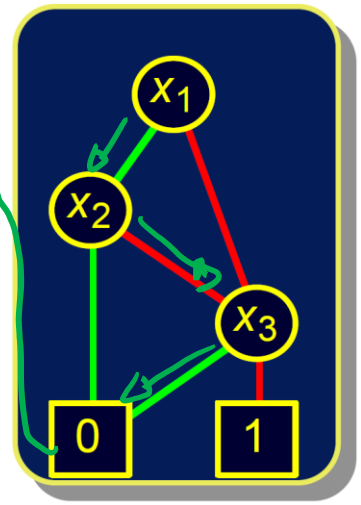
$f = x_1 + x_2 + x_3$



red: 1
green: 0

$g(0,1,0) = 0$ false

$g = (x_1 + x_2) \cdot x_3$



red: 1
green: 0

Decomposition

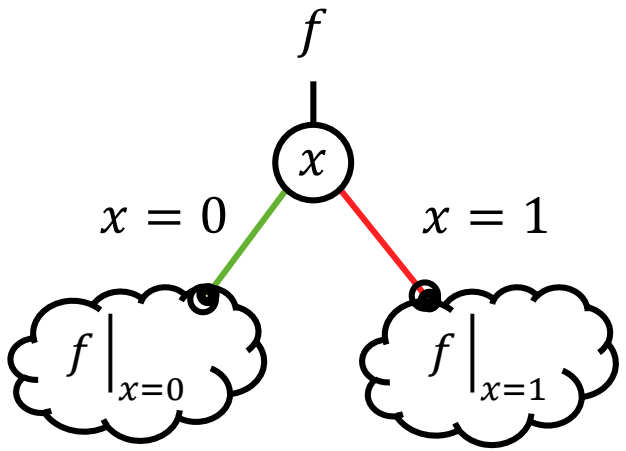
Logic	Boolean	Binary
OR	+	∨
AND	·	∧
NOT	$\overline{X}$	¬ or $\overline{X}$

BDDs are based on the Boole-Shannon-Decomposition:

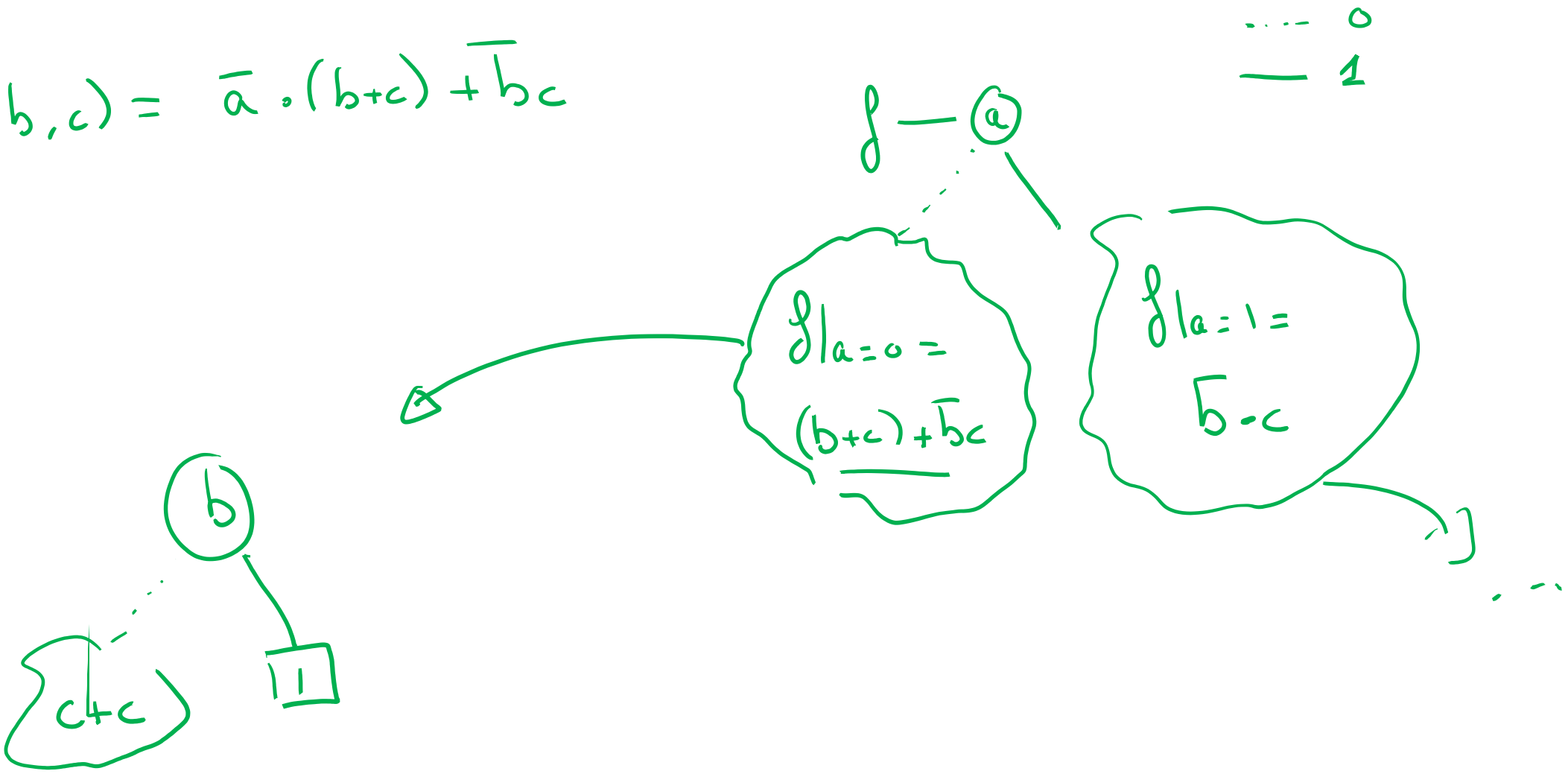
$$f = \bar{x} \cdot f\Big|_{x=0} + x \cdot f\Big|_{x=1}$$

A Boolean function has two co-factors for each variable, one for each valuation

- $f|_{x=0}$: remaining function for $x = 0$
- $f|_{x=1}$: remaining function for $x = 1$

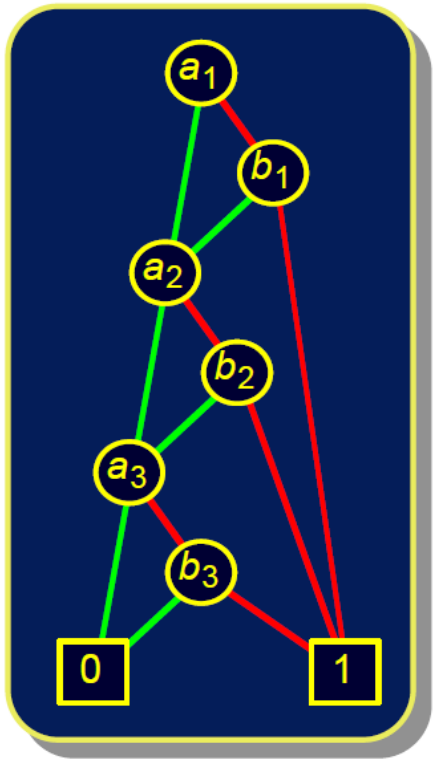


$f(a,b,c) = \bar{a} \cdot (b+c) + \bar{b}c$

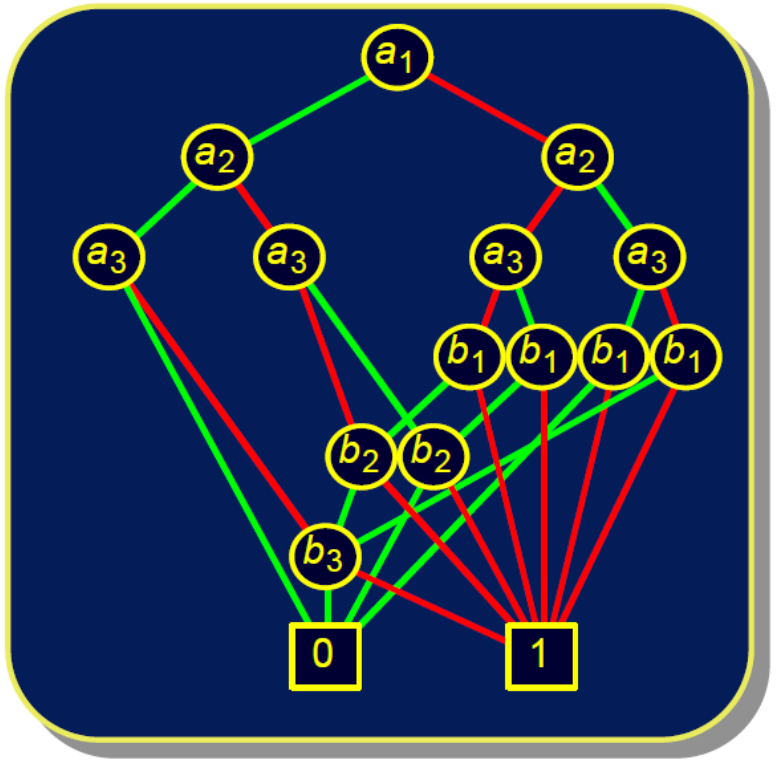


Variable Order

- If we fix the ordering of variables,
  BDDs are called OBBDs (Ordered Binary Decision Diagrams).
- The ordering is essential for the size of a BDD.



$$(a_1 \cdot b_1) + (a_1 \cdot b_2) + (a_3 \cdot b_3)$$

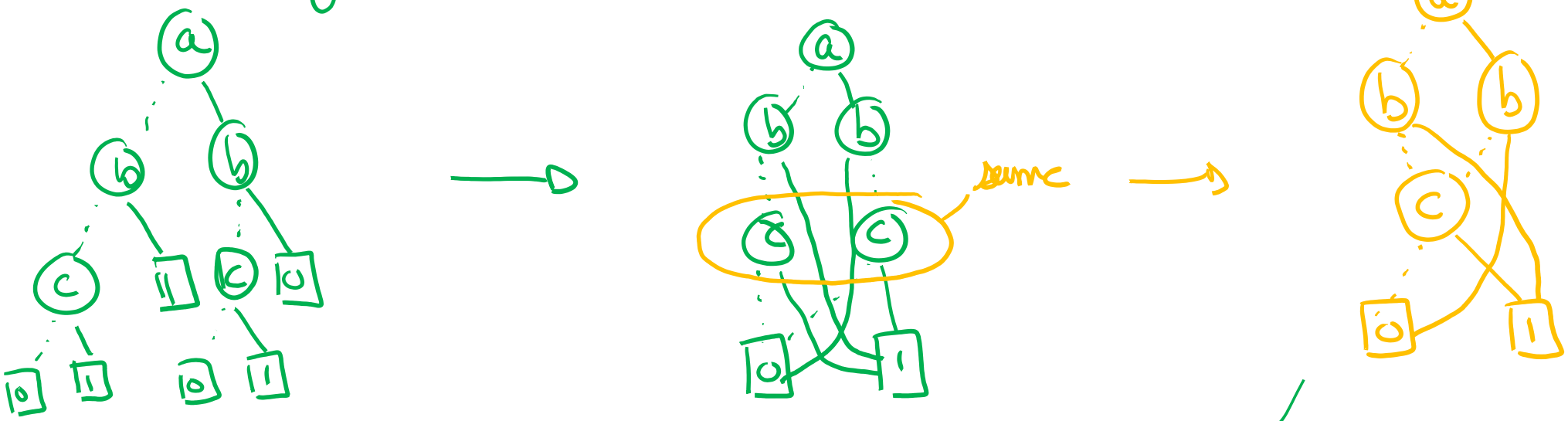


Calculating with BDDs

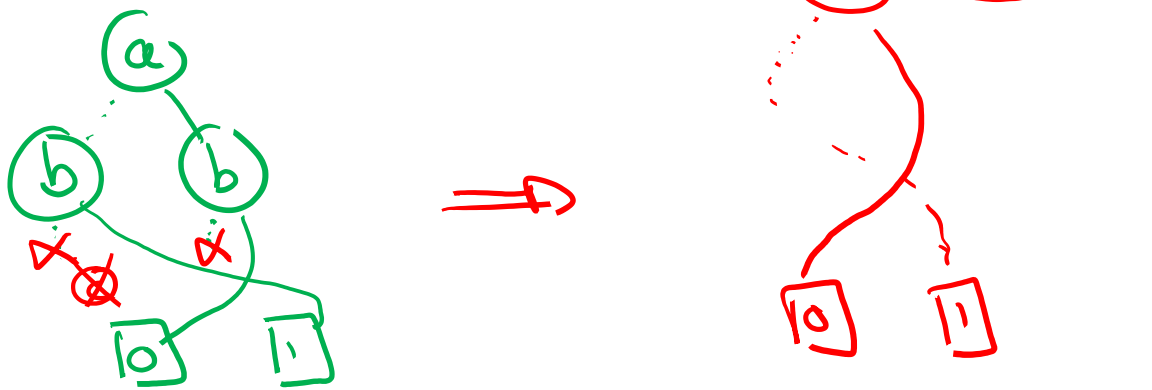
- RESTRICT: Given BDD for f, determine BDD for $f|_{x=k}$.
  - Delete all edges that represent $x = \bar{k}$;
  - For every pair of edges $(a - x\,,\ \ x - b)$ include a new edge $(a - b)$ and remove the old ones;
  - Remove all nodes that represent x.

- SIMPLIFY: Given BDD for f, determine simplified BDD for f.
  - Eliminate redundant nodes.
    - Merge equivalent leaves ($\boxed{0}$ and $\boxed{1}$)
    - Merge isomorphic nodes, i.e., nodes that represent the same Boolean function.
  - A BDD that can not be further simplified is called a reduced BDD.
    A reduced OBDD (also denoted as ROBDD) is a unique representation of a given Boolean function.

$f = \bar{a} \cdot (b+c) + \bar{b}c$



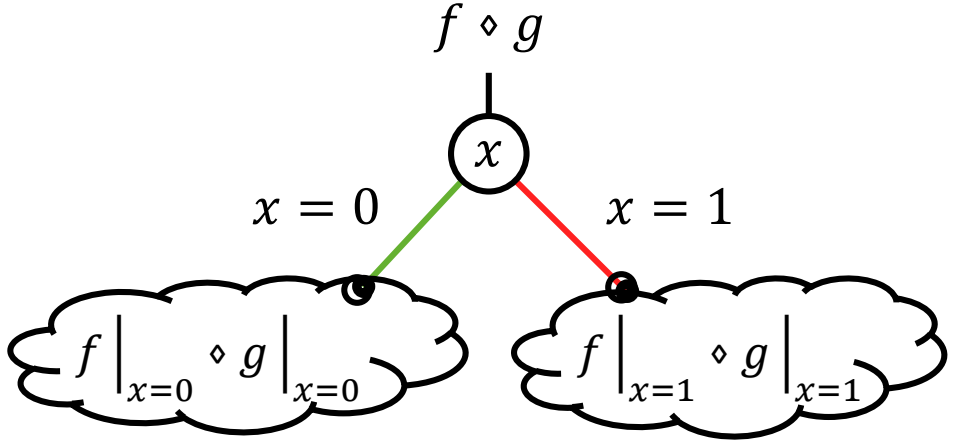
Restrict: $f|_{b=1}$



Calculating with BDDs

- APPLY: Given BDDs for f and g, determine a BDD for $f \diamond g$ for some operation $\diamond$.
  - Combine the two BDDs recursively based on the following relation:

$$f \diamond g = \overline{x} \cdot (f\,|_{x=0} \diamond g\,|_{x=0}) + x \cdot (f\,|_{x=1} \diamond g\,|_{x=1})$$



- Boolean functions can be converted to BDDs step by step using APPLY.

$$y = (x_1 \rightarrow x_2) \oplus x_3 \longrightarrow \begin{array}{l} y_1 = (x_1 \rightarrow x_2) \\ y = y_1 \oplus x_3 \end{array}$$

Calculating with BDDs

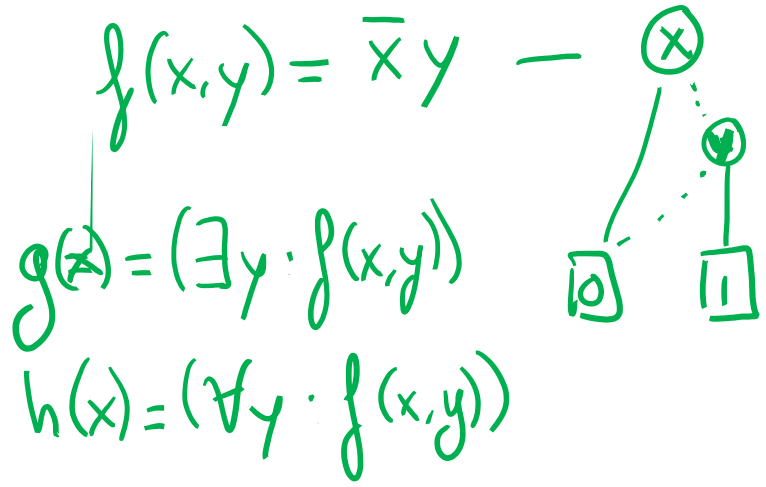
- Quantifiers are constructed by APPLY and RESTRICT:

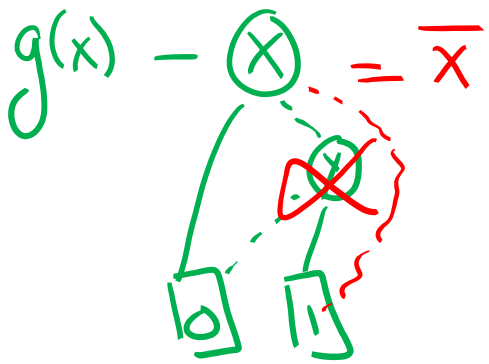
$$(\exists x : f) \quad \Leftrightarrow \quad (f\,|_{x=0} + f\,|_{x=1})$$

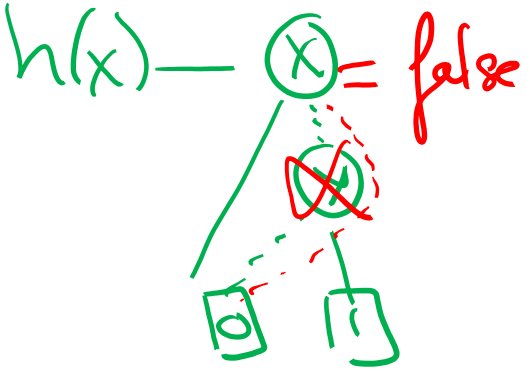
$$(\forall x : f) \quad \Leftrightarrow \quad (f\,|_{x=0} \cdot f\,|_{x=1})$$

$$(\exists x_1, x_2 : f) \quad \Leftrightarrow \quad (\exists x_1\,(\exists x_2 : f))$$

$$(\forall x_1, x_2 : f) \quad \Leftrightarrow \quad (\forall x_1\,(\forall x_2 : f))$$

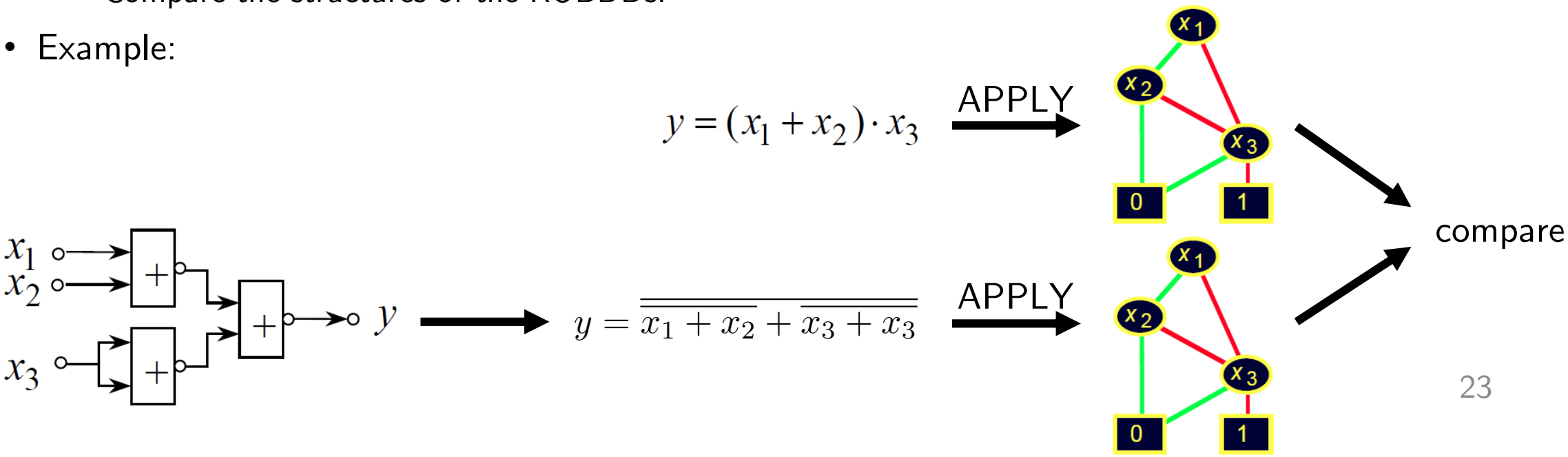






Comparison using BDDs

- Boolean (combinatorial) circuits: Compare specification and implementation, or compare two implementations.
- Method:
  - Representation of the two systems in ROBDDs, e.g., by applying the APPLY operator repeatedly.
  - Compare the structures of the ROBDDs.
- Example:

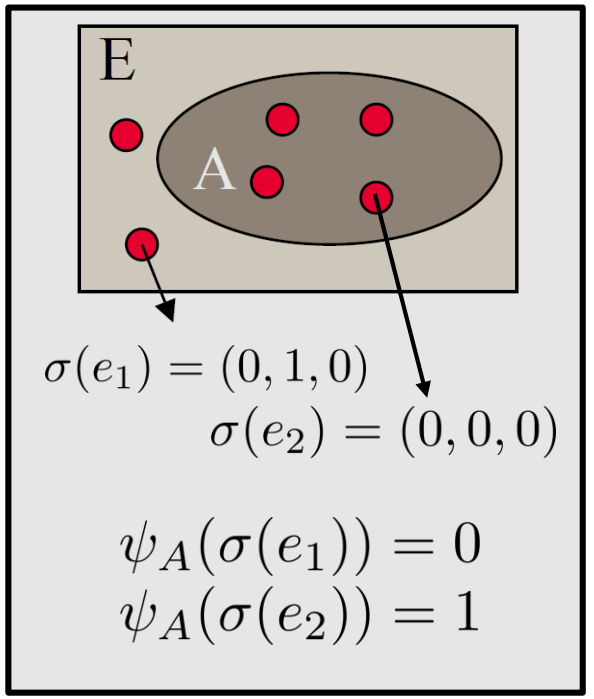


Sets and Relations

characteristic function of subset A

- Representation of a subset $A \subseteq E$:
  - Binary encoding $\sigma(e)$ of all elements $e \in E$
  - Subset A is represented by $a \in A \Leftrightarrow \psi_A(\sigma(a))$
  - Stepwise construction of the BDD corresponding to some subsets.

$$
\begin{aligned}
c \in A \cap B \quad &\Leftrightarrow \quad \psi_A(\sigma(c)) \cdot \psi_B(\sigma(c)) \\
c \in A \cup B \quad &\Leftrightarrow \quad \psi_A(\sigma(c)) + \psi_B(\sigma(c)) \\
c \in A \setminus B \quad &\Leftrightarrow \quad \psi_A(\sigma(c)) \cdot \overline{\psi_B(\sigma(c))} \\
c \in E \setminus A \quad &\Leftrightarrow \quad \overline{\psi_A(\sigma(c))}
\end{aligned}
$$



  - Example:

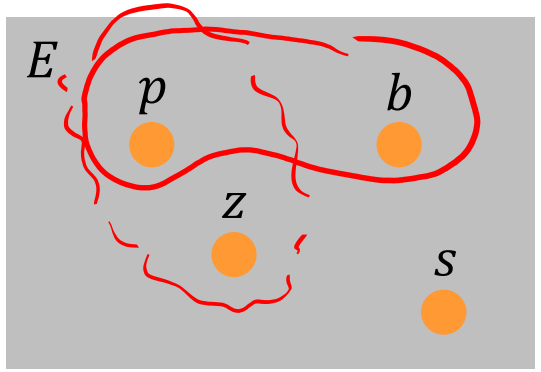
$$\forall e \in E: \ \sigma(e) = (x_1, x_0)$$

$$\sigma(e_0) = (0,0) \quad \sigma(e_1) = (0,1) \quad \sigma(e_2) = (1,0) \quad \sigma(e_3) = (1,1)$$

$$\psi_A = x_0 \oplus x_1 \quad \Leftrightarrow \quad A = \{e_1, e_2\}$$

Sets and Relations

$\sigma(e)$	x_1	x_0
Zürich	0	0
Sydney	0	1
Beijing	1	0
Paris	1	1



$\Psi_A(x_1, x_0) = v_1$

$\Psi_B(x_1, x_0) = \bar{x}_1\bar{x}_0 + x_1x_0$

$= \overline{x_0 \oplus x_1}$

$C = A \cap B$

$\Rightarrow \Psi_C = \Psi_A \cdot \Psi_B$

$= x_1 \cdot x_0$

Selecting a "good" encoding is both important and difficult

It's one challenge of ML:
How to efficiently
encode the inputs?

For a state space encoded with N bits

1 bit represents up to 2^{N-1} states

In previous example

Subset A of all capitals is represented by $\psi_A = x_1$

- No need to iterate through all capitals to verify that some property holds (e.g. "All capitals have a parliament.")
- We can use the (compact) representation of the set.

But...

Selecting a good encoding —Representing state efficiently is difficult in practice.

▶ Subject of several PhD theses

Efficient state representation

- Set of states as Boolean function
- Binary Decision Diagram representation

Computing reachability

- Leverage efficient state representation
- Explore successor sets of states

Proving properties

- Temporal logic (CTL)
- Encoding as reachability problem

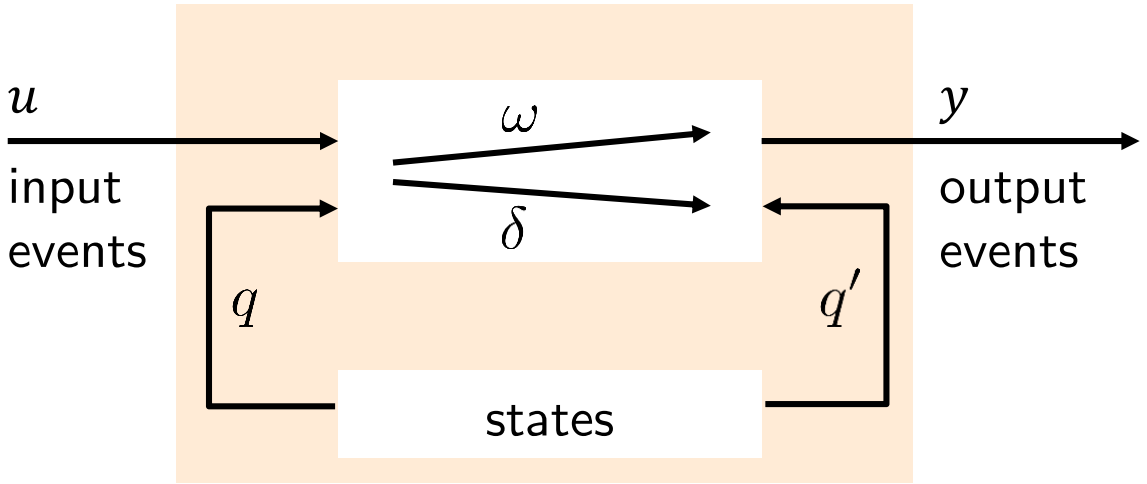
Sets and Relations using BDDs

- Representation of a relation $R \subseteq A \times B$
  - Binary encoding $\sigma(a)$, $\sigma(b)$ of all elements $a \in A$, $b \in B$
  - Representation of R

$$(a, b) \in R \Leftrightarrow \psi_R(\sigma(a), \sigma(b))$$

characteristic function of the relation R

- Example finite automaton:



finite automaton

$$\psi_\delta(u, q, q') = 1$$

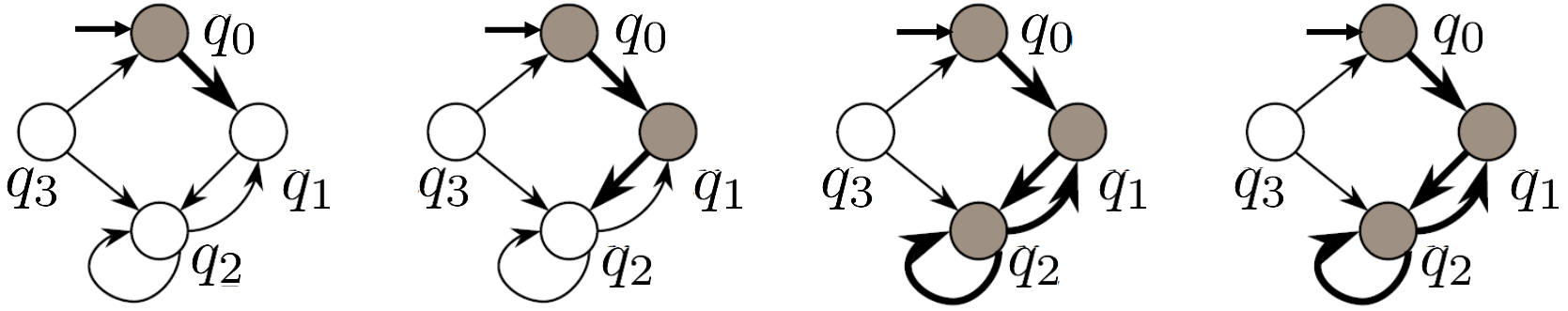
$$\psi_\omega(u, q, y) = 1$$

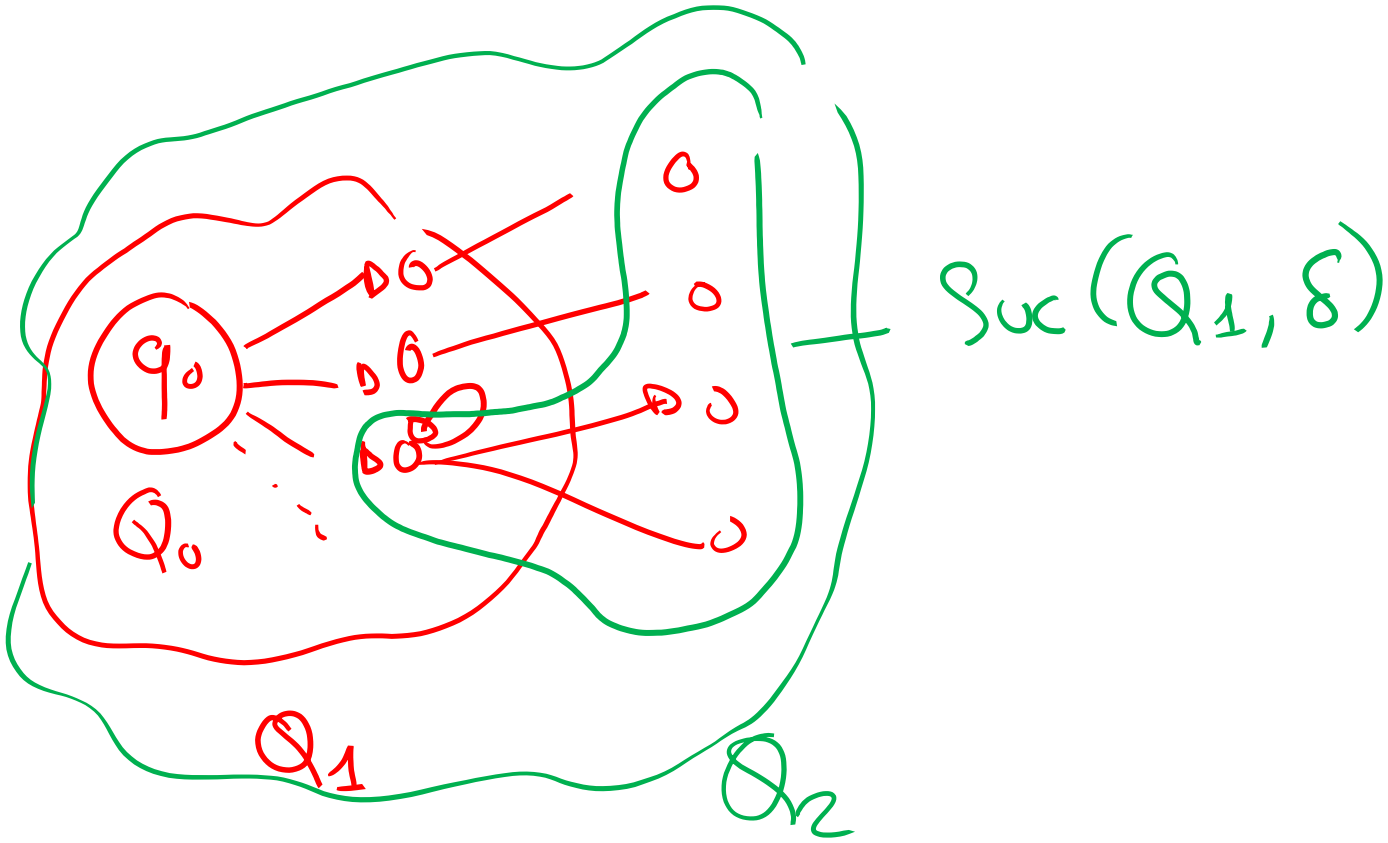
we remove the binary encoding for convenience in our notation; but u, q, q' are actually represented as binary vectors

Reachability of States

- Problem: Is a state $q \in Q$ reachable by a sequence of state transitions?
- Method:
  - Represent set of states and the transformation relation as ROBDDs.
  - Use these representations to transform from one set of states to another. Set Q_i corresponds to the set of states reachable after i transitions.
  - Iterate the transformation until a fixed-point is reached, i.e., until the set of states does not change anymore (steady-state).
- Example:

$Q_0 = \{q_0\}$ $Q_1 = Q_0 \cup \{q_1\}$ $Q_2 = Q_1 \cup \{q_1, q_2\}$ $Q_3 = Q_2 \cup \{q_1, q_2\}$




q_0
Q_0
Q_1
Q_2
$Suc(Q_1, \delta)$

But drawing state-diagrams is not feasible in general.

But drawing state-diagrams is not feasible in general.

1. Work with sets of states
2. Use characteristic functions to represent sets of states
3. Use ROBDDs to encode characteristic functions

Reachability of States

- Transformation of sets of states:
  - Determine the set of all direct successor states of a given set of states Q by means of the transformation function δ:

$$Q' = Suc(Q, \delta) = \{q' \mid \exists q : \psi_Q(q) \cdot \psi_\delta(q, q')\}$$

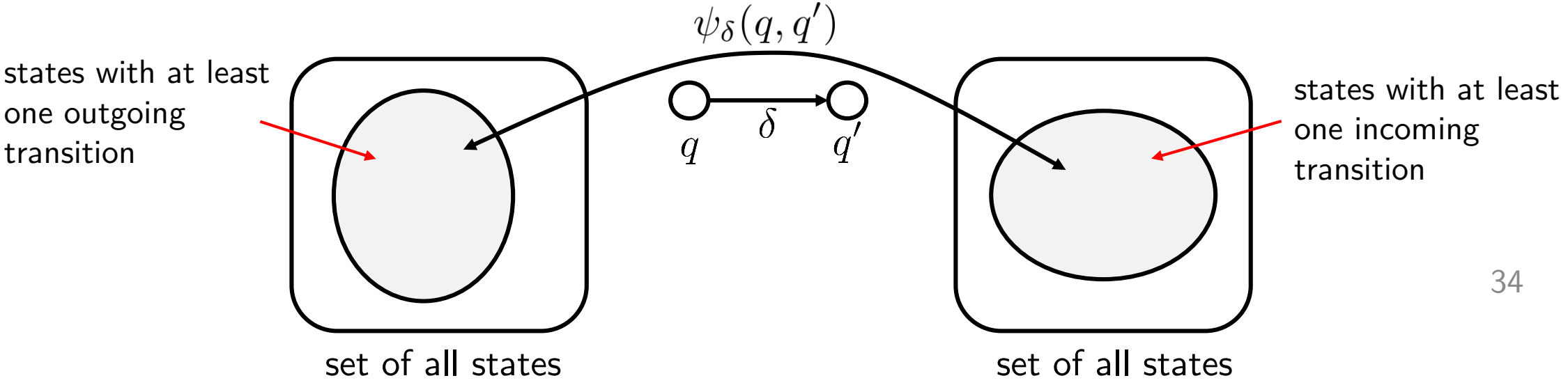
$$\psi_\delta(q, q') = (\exists u : \psi_\delta(u, q, q'))$$

To simplify notation

Reachability of States

- Transformation of sets of states:
  - Determine the set of all direct successor states of a given set of states Q by means of the transformation function δ:

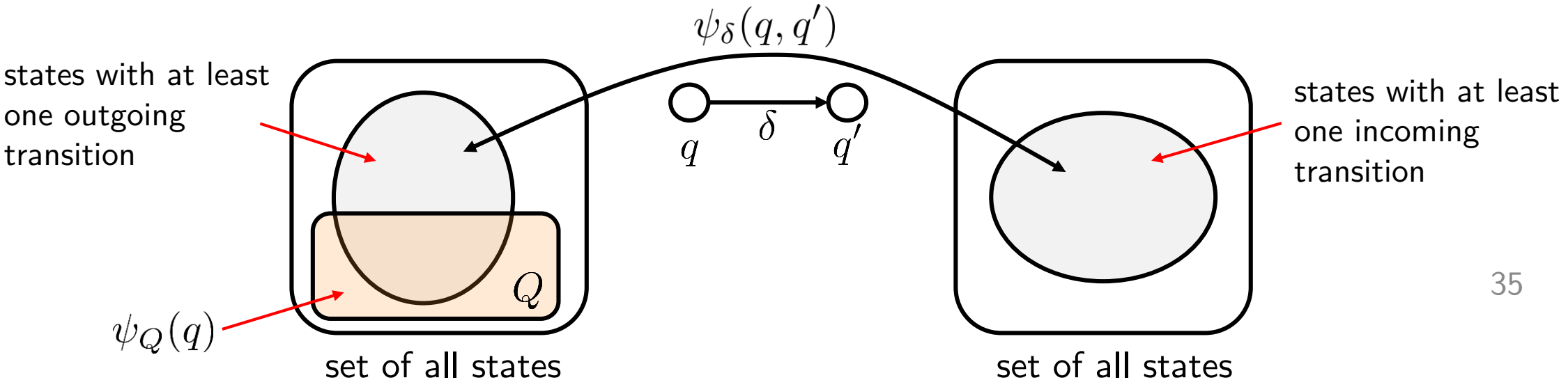
$$Q' = Suc(Q, \delta) = \{q' \mid \exists q : \psi_Q(q) \cdot \psi_\delta(q, q')\}$$



Reachability of States

- Transformation of sets of states:
  - Determine the set of all direct successor states of a given set of states Q by means of the transformation function δ:

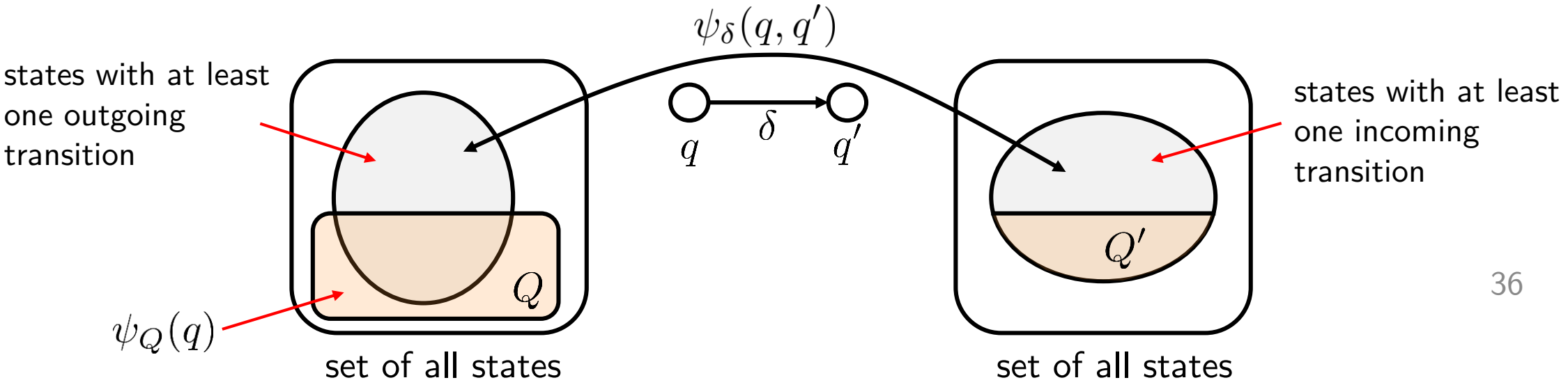
$$Q' = Suc(Q, \delta) = \{q' \mid \exists q : \psi_Q(q) \cdot \psi_\delta(q, q')\}$$



Reachability of States

- Transformation of sets of states:
  - Determine the set of all direct successor states of a given set of states Q by means of the transformation function δ:

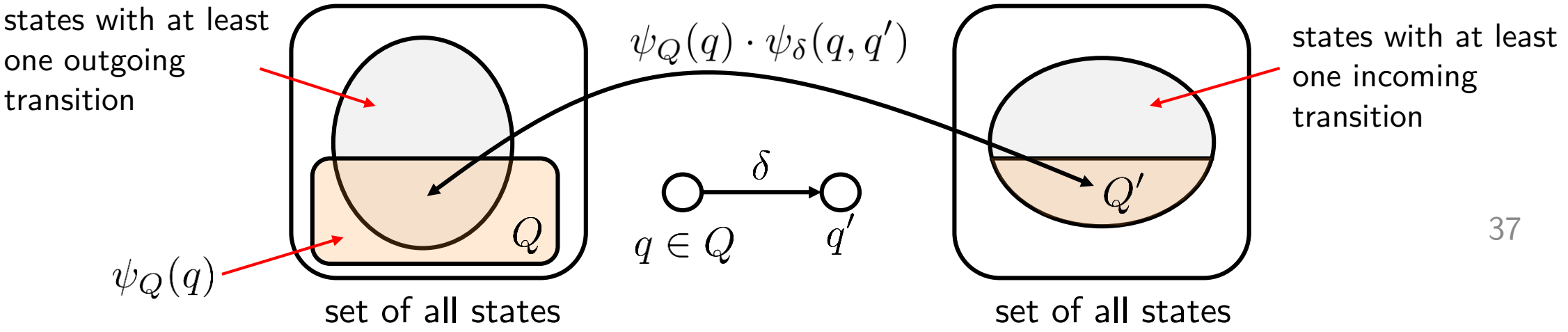
$$Q' = Suc(Q, \delta) = \{q' \mid \exists q : \psi_Q(q) \cdot \psi_\delta(q, q')\}$$



Reachability of States

- Transformation of sets of states:
  - Determine the set of all direct successor states of a given set of states Q by means of the transformation function δ:

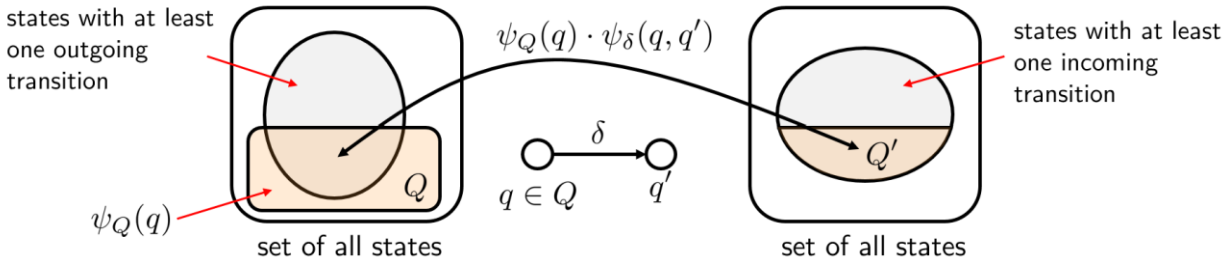
$$Q' = Suc(Q, \delta) = \{q' \mid \exists q : \psi_Q(q) \cdot \psi_\delta(q, q')\}$$



Reachability of States

- Transformation of sets of states:
  - Determine the set of all direct successor states of a given set of states Q by means of the transformation function δ:

$$Q' = Suc(Q, \delta) = \{q' \mid \exists q : \psi_Q(q) \cdot \psi_\delta(q, q')\}$$



$$h(q, q') = \psi_Q(q) \cdot \psi_\delta(q, q')$$

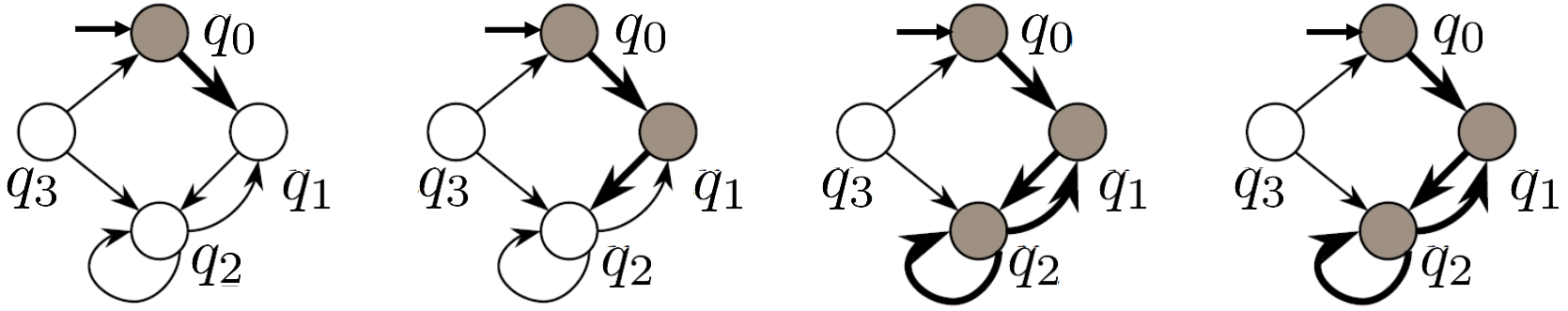
$$\psi_{Q'}(q') = (\exists q : h(q, q'))$$

Efficient to compute with ROBDDs

Reachability of States

- Problem: Is a state $q \in Q$ reachable by a sequence of state transitions?
- Method:
  - Represent set of states and the transformation relation as ROBDDs.
  - Use these representations to transform from one set of states to another. Set Q_i corresponds to the set of states reachable after i transitions.
  - Iterate the transformation until a fixed-point is reached, i.e., until the set of states does not change anymore (steady-state).
- Example:

$Q_0 = \{q_0\}$ $Q_1 = Q_0 \cup \{q_1\}$ $Q_2 = Q_1 \cup \{q_1, q_2\}$ $Q_3 = Q_2 \cup \{q_1, q_2\}$



Reachability of States

- Fixed-point iteration
  - Start with the initial state, then determine the set of states that can be reached in one or more steps.

$$Q_0 = \{q_0\}$$

$$Q_{i+1} = Q_i \cup Suc(Q_i, \delta) \qquad \text{until } Q_{i+1} = Q_i$$

$$\psi_{Q_{i+1}}(q') = \psi_{Q_i}(q') + (\exists q : \psi_{Q_i}(q) \cdot \psi_\delta(q, q'))$$

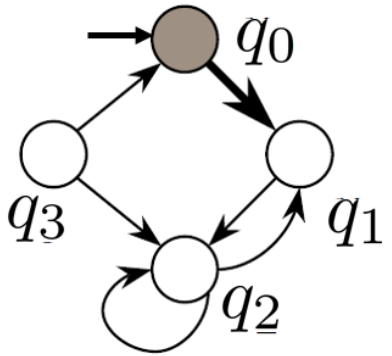
  - Due to the finite number of states, the fixed-point exists and is reached in a finite number of steps (at most the diameter of the state diagram).
  - Determine whether the fixed-point is reached or not can be done by comparing the ROBDDs of the current set of reachable states.

Reachability of States - Example

State encoding

$$(x_1, x_0) = \sigma(q)$$

$\sigma(q)$	x_1	x_0
q_0	0	0
q_1	0	1
q_2	1	0
q_3	1	1

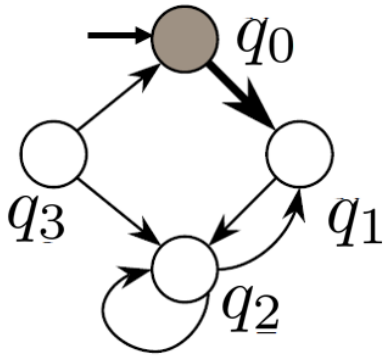


Reachability of States - Example

State encoding

$(x_1, x_0) = \sigma(q)$

$\sigma(q)$	x_1	x_0
q_0	0	0
q_1	0	1
q_2	1	0
q_3	1	1



Transition relation encoding

$\psi_\delta(q, q')$

entries where $\psi_\delta(q, q') = 1$ only

x_1	x_0	x_1'	x_0'
0	0	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	1	0
1	1	0	0

e.g.

$q_0 \rightarrow q_1$

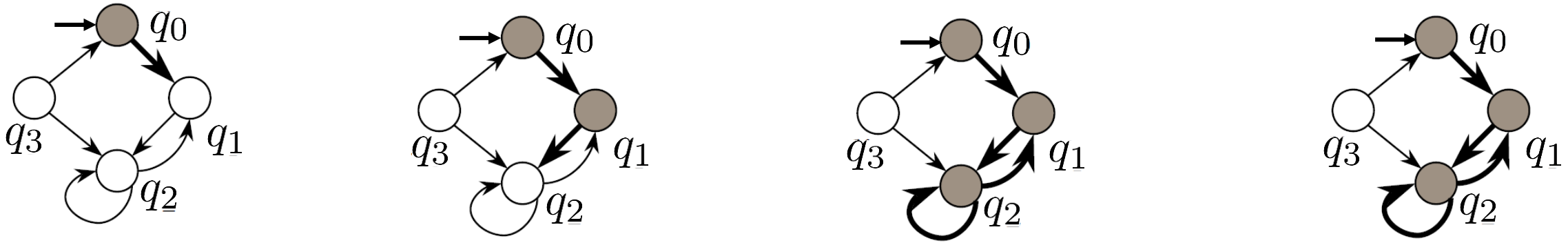
$q_2 \rightarrow q_2$

As a Boolean function

$\psi_\delta(q, q') = \overline{x_0}' \cdot (x_0 \cdot (x_1 + x_1') + x_1 \cdot x_1') + \overline{x_0} \cdot x_0' \cdot \overline{x_1}'$

Reachability of States - Example

$Q_0 = \{q_0\}$ $Q_1 = Q_0 \cup \{q_1\}$ $Q_2 = Q_1 \cup \{q_1, q_2\}$ $Q_3 = Q_2 \cup \{q_1, q_2\}$



$$\psi_{Q_0}(q') = \overline{x'_1} \cdot \overline{x'_0}$$

$$\psi_{Q_1}(q') = \overline{x'_1} \cdot \overline{x'_0} + \overline{x'_1} \cdot x'_0 = \overline{x'_1}$$

$\sigma(q)$	x_1	x_0
q_0	0	0
q_1	0	1
q_2	1	0
q_3	1	1

$$\psi_{Q_2}(q') = \overline{x'_1} + (x'_1 \cdot \overline{x'_0} + \overline{x'_1} \cdot x'_0) = \overline{x'_1} + \overline{x'_0}$$

$$\psi_{Q_3}(q') = (\overline{x'_1} + \overline{x'_0}) + (\overline{x'_1} + \overline{x'_0}) = \overline{x'_1} + \overline{x'_0}$$

It's always a reachability problem

Or rather

The goal is to transform the problem at hand
to encode it as a reachability problem.

Because these can be solved very efficiently

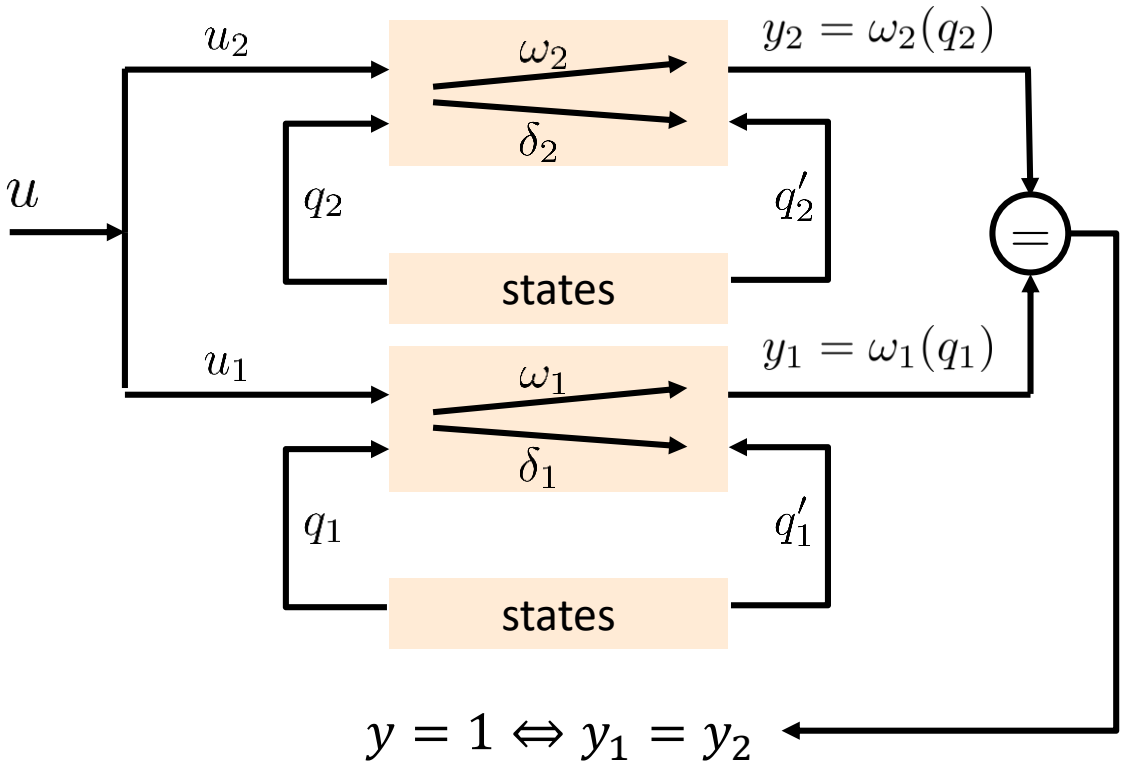
1. Work with sets of states
2. Use characteristic functions to represent sets of states
3. Use ROBDDs to encode characteristic functions

Comparison of Finite Automata

For simplicity, we only consider Moore automata, i.e., the output depends on the current state only. The output function is $\omega : Q \to \Sigma$ and $y = \omega(q)$.

Strategy

1. Compute the set of jointly reachable states.
2. Compare the output values of the two finite automata.



Comparison of Finite Automata

- Compute the joint transition function

$$\psi_\delta(q_1, q_2, q_1', q_2') = (\exists\, u : \psi_{\delta_1}(u, q_1, q_1') \cdot \psi_{\delta_2}(u, q_2, q_2'))$$

- Compute the set of reachable states

$$\psi_Q(q_1, q_2)$$

- Compute the set of reachable output values

$$\psi_Y(y_1, y_2) = (\exists\, q_1, q_2 : \psi_Q(q_1, q_2) \cdot \psi_{\omega_1}(q_1, y_1) \cdot \psi_{\omega_2}(q_2, y_2))$$

- The automata are not equivalent iff the following term is true

$$\exists\, y_1, y_2 : \psi_Y(y_1, y_2) \cdot (y_1 \neq y_2)$$

Your turn to practice!
after the break

1. Familiarise yourself with the equivalence
   "set of states" ≡ "characteristic functions"

2. Express system properties using
   characteristic functions

3. Draw and simplify BDDs to compare
   a specification and an implementation

Efficient state representation

- Set of states as Boolean function
- Binary Decision Diagram representation

Computing reachability

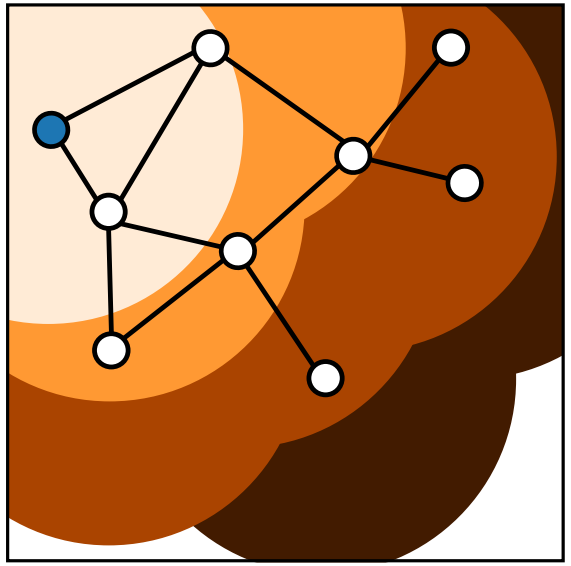
- Leverage efficient state representation
- Explore successor sets of states

Next week

Proving properties

- Temporal logic (CTL)
- Encoding as reachability problem

See you next week!
in Discrete Event Systems



Romain Jacob

www.romainjacob.net

ETH Zurich (D-ITET)

November 25, 2021

Most materials from Lothar Thiele