Discrete Event Systems

- Verification of Finite Automata –

Lothar Thiele

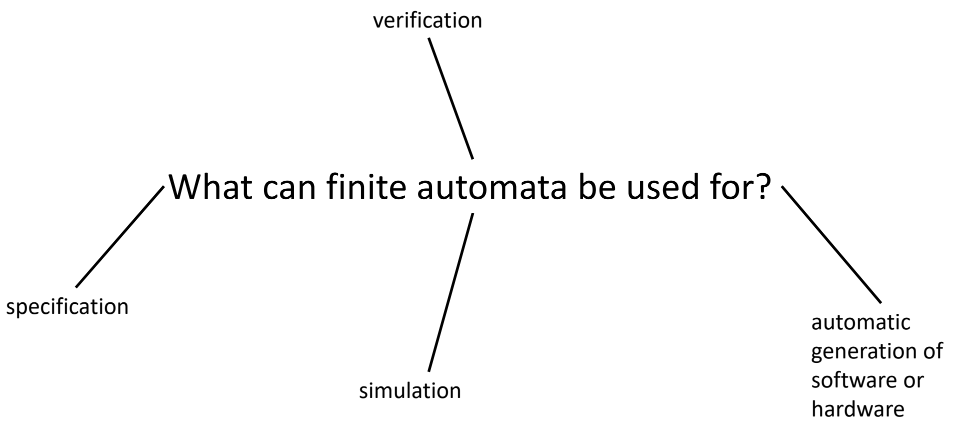
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks

Overview

- **Introduction**
- Binary Decision Diagrams
  - Representation of Boolean Functions
  - Comparing two circuits
  - Representation of Sets
- Finite Automata
  - Reachability of States
  - Comparing two finite automata
  - Proving properties of finite automata
    - Computation Tree Logic (CTL)
    - Evaluating formulas
    - Verification of finite automata

What can finite automata be used for?



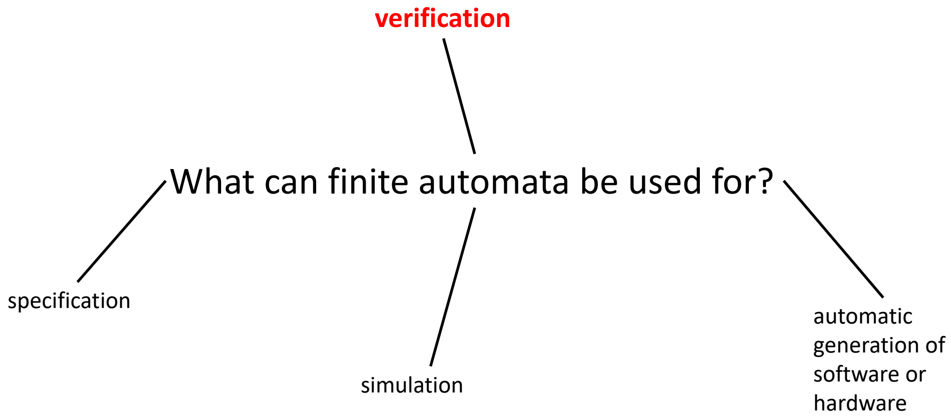
What can finite automata be used for?

verification

specification

simulation

automatic generation of software or hardware



verification

What can finite automata be used for?

specification

simulation

automatic generation of software or hardware

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:

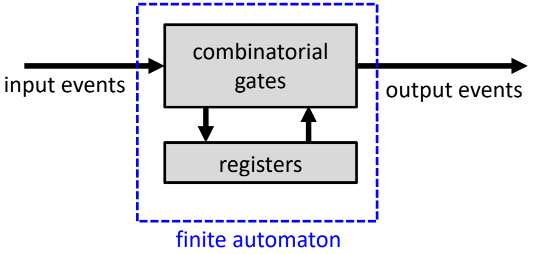
- Simulation (sometimes also called validation): Unless the simulation is exhaustive, i.e., all possible input sequences are tested, the result is not trustworthy. In general, simulation can only show the presence of errors but not the absence (correctness).
- Formal analysis (sometimes also called verification):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

→ this is what we will do

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?



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).

Recap Finite Automata (now with output)

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

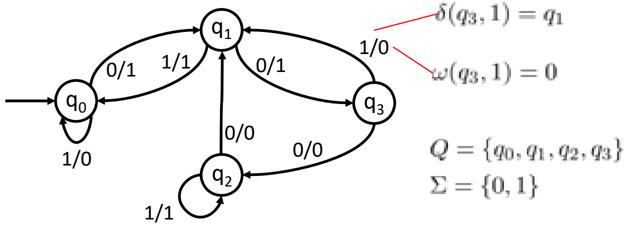
Q is a finite set called the states,

Σ is a finite set called the alphabet,

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

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

$q_0 \in Q$ is the start state.



$\delta(q_3, 1) = q_1$

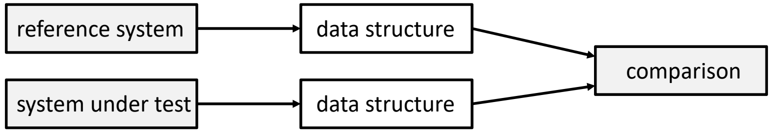
$\omega(q_3, 1) = 0$

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

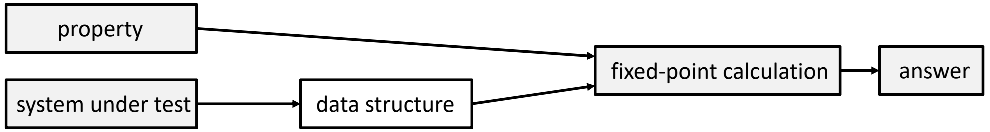
$\Sigma = \{0, 1\}$

Verification Scenarios

- Comparison of specification and implementation



- Proving properties



Overview

- Introduction
- **Binary Decision Diagrams**
  - **Representation of Boolean Functions**
  - Comparing two circuits
  - Representation of Sets
- Finite Automata
  - Reachability of States
  - Comparing two finite automata
  - Proving properties of finite automata
    - Computation Tree Logic (CTL)
    - Evaluating formulas
    - Verification of finite automata

Basic concept of verification using BDDs

- BDDs represent Boolean functions.

- Therefore, they can be used to describe sets of states and transformation relations.

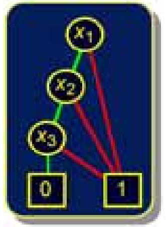
- Due to the unique representation of Boolean functions, *reduced ordered* BDDs (ROBDD) can be used to proof equivalence between Boolean functions or between sets of states.

- BDDs can easily and efficiently be manipulated.

Binary Decision Diagrams (BDD)

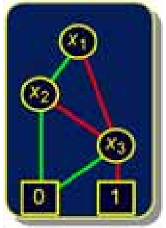
- Concept
  - Data structure that allows to represent Boolean functions.
  - The representation is unique for a given ordering of variables. If the ordering of variables is fixed, we call it an ordered BDD (OBDD).
- Structure
  - BDDs contain "decision nodes" which are labeled with variable names.
  - Edges are labeled with input values.
  - Leaves are labeled with output values.

$x_1 \vee x_2 \vee x_3$



red: 1
green: 0

$(x_1 \vee x_2) \wedge x_3$



red: 1
green: 0

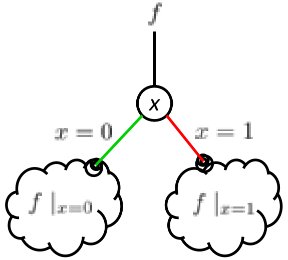
Decomposition

- BDDs are based on the Boole-Shannon-Decomposition:

$$\boxed{f = \overline{x} \cdot f\,|_{x=0} + x \cdot f\,|_{x=1}}$$

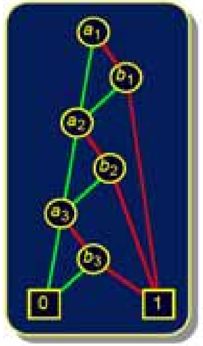
(we sometimes use + for logical or and · for logical and)

- 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

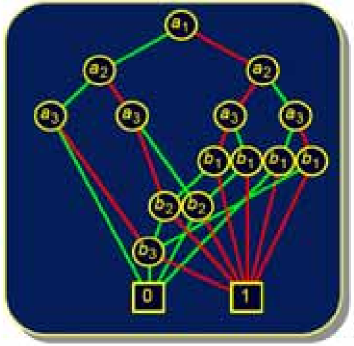


Variable Order

- 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.



$(a_1 \wedge b_1) \vee$
$(a_2 \wedge b_2) \vee$
$(a_3 \wedge b_3)$



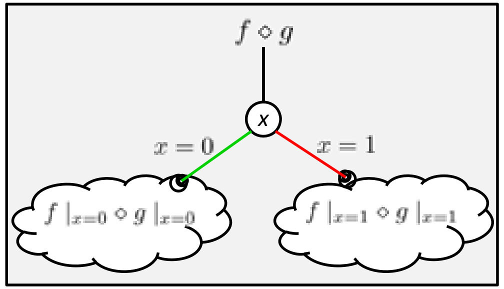
Calculating with BDDs

- ***RESTRICT***: Given BDD for f, determine BDD for $f\,|_{x=k}$.
  - Delete all edges that represent $x = \overline{k}$.
  - Remove all nodes that represent *x*; for every pair of edges *(a, x), (x, b)* include a new edge *(a, b)* and remove the old ones.
- ***SIMPLIFY***: Given BDD for f, determine simplified BDD for f.
  - Merge equivalent leaves
  - Merge isomorphic nodes, i.e., nodes that represent the same Boolean function.
  - Eliminate redundant nodes.
  - 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.

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:

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


$f \diamond g$
x
$x = 0$
$x = 1$
$f\,|_{x=0} \diamond g\,|_{x=0}$
$f\,|_{x=1} \diamond g\,|_{x=1}$

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

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

Calculating with BDDs

- Digital circuits are first converted to a Boolean expression, e.g., first sort the gates topologically and then construct the expression from the end.

- Quantifiers are constructed by ***APPLY*** and ***RESTRICT***:

$$\begin{array}{lcl} (\exists x : f) & \Leftrightarrow & (f\,|_{x=0} + f\,|_{x=1}) \\ (\forall x : f) & \Leftrightarrow & (f\,|_{x=0} \cdot f\,|_{x=1}) \end{array}$$

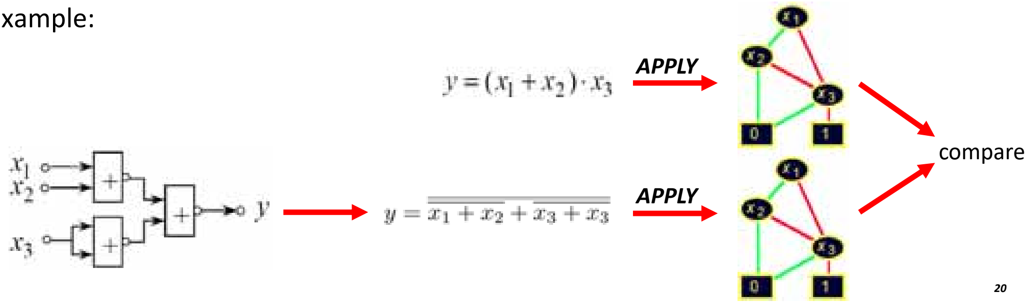
$$\begin{array}{lcl} (\exists x_1, x_2 : f) & \Leftrightarrow & (\exists x_1\ (\exists x_2 : f)) \\ (\forall x_1, x_2 : f) & \Leftrightarrow & (\forall x_1\ (\forall x_2 : f)) \end{array}$$

Overview

- Introduction
- **Binary Decision Diagrams**
  - Representation of Boolean Functions
  - **Comparing two circuits**
  - Representation of Sets
- Finite Automata
  - Reachability of States
  - Comparing two finite automata
  - Proving properties of finite automata
    - Computation Tree Logic (CTL)
    - Evaluating formulas
    - Verification of finite automata

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:



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

$y = \overline{\overline{x_1 + x_2} + \overline{x_3 + x_3}}$

APPLY

APPLY

compare

Overview

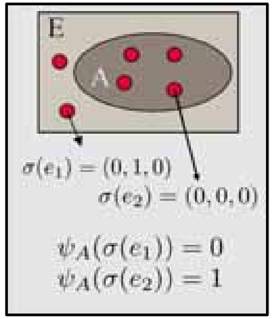
- Introduction
- **Binary Decision Diagrams**
  - Representation of Boolean Functions
  - Comparing two circuits
  - **Representation of Sets**
- Finite Automata
  - Reachability of States
  - Comparing two finite automata
  - Proving properties of finite automata
    - Computation Tree Logic (CTL)
    - Evaluating formulas
    - Verification of finite automata

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: $a \in A \quad \Leftrightarrow \quad \psi_A(\sigma(a))$
  - Stepwise construction of the BDD corresponding to some subset:

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



$\sigma(e_1) = (0,1,0)$

$\sigma(e_2) = (0,0,0)$

$\psi_A(\sigma(e_1)) = 0$

$\psi_A(\sigma(e_2)) = 1$

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

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

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

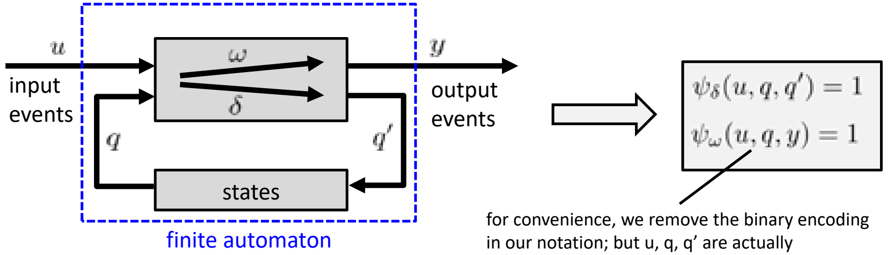
Sets and Relations using BDDs

- 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 :

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

characteristic function of a relation *R*

- Example finite automaton:



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

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

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

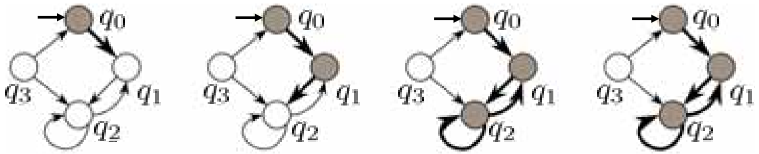
Overview

- Introduction
- Binary Decision Diagrams
  - Representation of Boolean Functions
  - Comparing two circuits
  - Representation of Sets
- **Finite Automata**
  - **Reachability of States**
  - Comparing two finite automata
  - Proving properties of finite automata
    - Computation Tree Logic (CTL)
    - Evaluating formulas
    - Verification of finite automata

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:

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



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 :

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

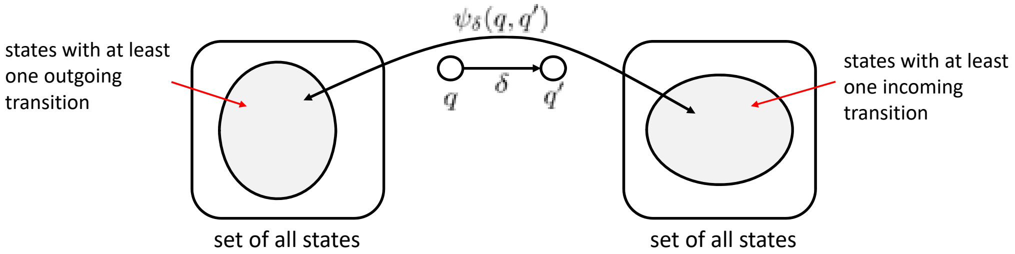
As we neglect the input in the above formulation, we use $\psi_\delta(q, q') = (\exists u : \psi_\delta(u, q, q'))$

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 :

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

As we neglect the input in the above formulation, we use $\psi_\delta(q,q') = (\exists u : \psi_\delta(u,q,q'))$



states with at least one outgoing transition

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

q δ q'

states with at least one incoming transition

set of all states

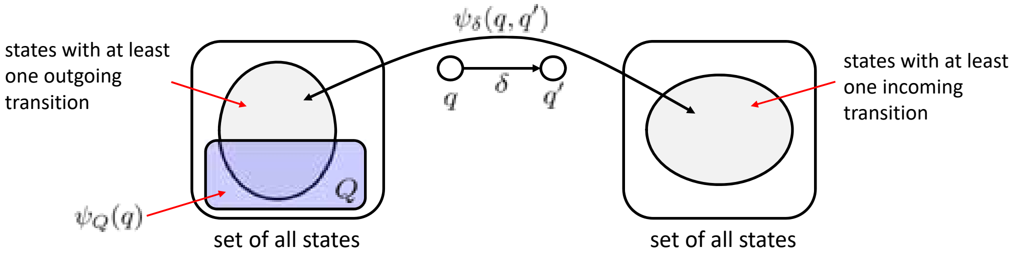
set of all states

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 :

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

As we neglect the input in the above formulation, we use $\psi_\delta(q,q') = (\exists u : \psi_\delta(u,q,q'))$



states with at least one outgoing transition

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

q δ q'

states with at least one incoming transition

Q

$\psi_Q(q)$

set of all states

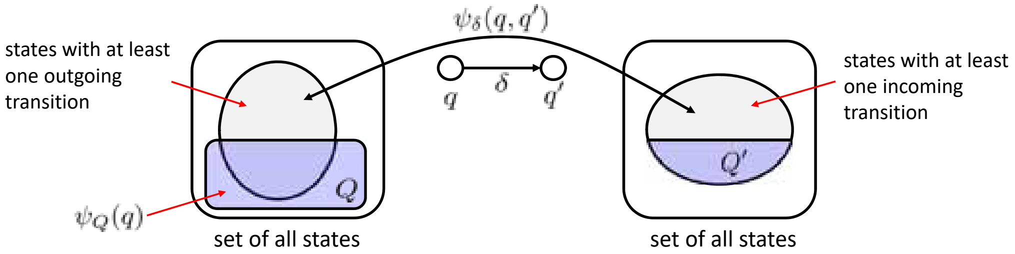
set of all states

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 :

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

As we neglect the input in the above formulation, we use $\psi_\delta(q,q') = (\exists u : \psi_\delta(u,q,q'))$



states with at least one outgoing transition

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

q δ q'

states with at least one incoming transition

Q

Q'

$\psi_Q(q)$

set of all states

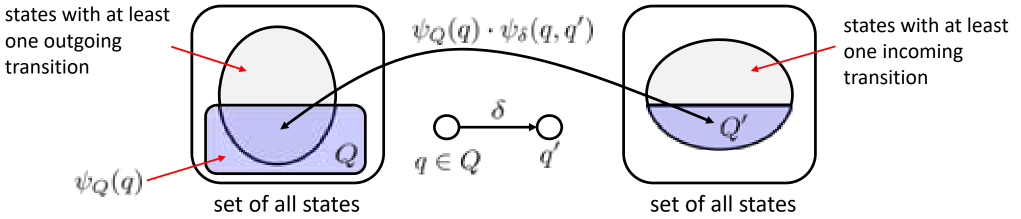
set of all states

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 :

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

As we neglect the input in the above formulation, we use $\psi_\delta(q,q') = (\exists u : \psi_\delta(u,q,q'))$



states with at least one outgoing transition

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

δ

$q \in Q$ q'

states with at least one incoming transition

Q

Q'

$\psi_Q(q)$

set of all states

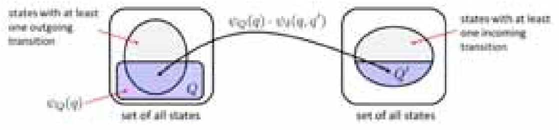
set of all states

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 :

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

As we neglect the input in the above formulation, we use $\psi_\delta(q, q') = (\exists u : \psi_\delta(u, q, q'))$


states with at least one outgoing transition
$\psi_Q(q) \cdot \psi_\delta(q,q')$
states with at least one incoming transition
$\psi_Q(q)$
Q
Q'
set of all states
set of all states

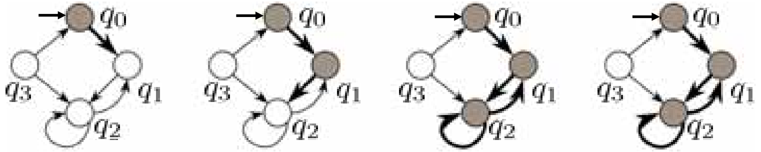
- Computation:

$$\left.\begin{aligned} h(q, q') &= \psi_Q(q) \cdot \psi_\delta(q, q') \\ \psi_{Q'}(q') &= (\exists q : h(q, q')) \end{aligned}\right\} \text{computation using ROBDDs}$$

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 set of sets. 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.

$$Q_0 = \{q_0\}$$

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

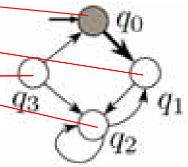
$$\psi_{Q_{i+1}}(q') = \psi_{Q_i}(q') + (\exists q \ : \ \psi_{Q_i}(q) \cdot \psi_\delta(q, q'))$$

  - 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

- Encode states $(x_1, x_0) = \sigma(q)$:

	x_1	x_0
q_0	0	0
q_1	0	1
q_2	1	0
q_3	1	1



- Encode transition relation $\psi_\delta(q, q')$ (Table just contains the entries where $\psi_\delta(q, q') = 1$):

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

As a Boolean function: $\psi_\delta(q, q') = \overline{x_0'} \cdot (x_0 \cdot (x_1 + x_1') + x_1 \cdot x_1') + \overline{x_0} \cdot x_0' \cdot \overline{x_1'}$

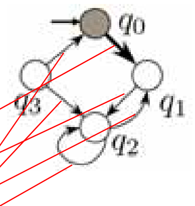
Reachability of States - Example

- Encode states $(x_1, x_0) = \sigma(q)$:

	x_1	x_0
q_0	0	0
q_1	0	1
q_2	1	0
q_3	1	1

- Encode transition relation $\psi_\delta(q, q')$ (Table just contains the entries where $\psi_\delta(q, q') = 1$):

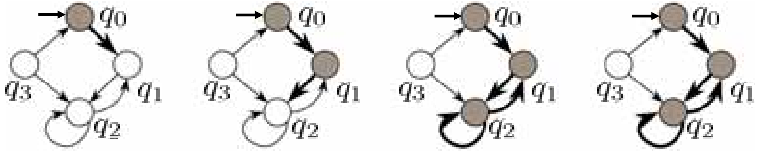
x_1	x_0	x_1'	x_0'
0	0	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	1	0
1	1	0	0



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

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



$$\psi_{Q_0}(q') = \overline{x_1'} \cdot \overline{x_0'}$$

$$\psi_{Q_1}(q') = \overline{x_1'} \cdot \overline{x_0'} + \overline{x_1'} \cdot x_0' = \overline{x_1'}$$

$$\psi_{Q_2}(q') = \overline{x_1'} + (x_1' \cdot \overline{x_0'} + \overline{x_1'} \cdot x_0') = \overline{x_1'} + \overline{x_0'}$$

$$\psi_{Q_3}(q') = (\overline{x_1'} + \overline{x_0'}) + (\overline{x_1'} + \overline{x_0'}) = \overline{x_1'} + \overline{x_0'}$$

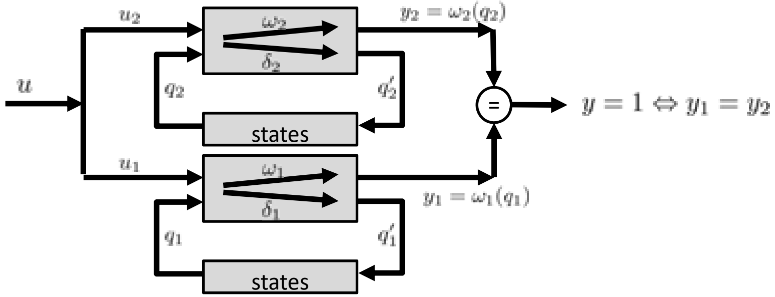
	x_1	x_0
q_0	0	0
q_1	0	1
q_2	1	0
q_3	1	1

Overview

- Introduction
- Binary Decision Diagrams
  - Representation of Boolean Functions
  - Comparing two circuits
  - Representation of Sets
- **Finite Automata**
  - Reachability of States
  - **Comparing two finite automata**
  - Proving properties of finite automata
    - Computation Tree Logic (CTL)
    - Evaluating formulas
    - Verification of finite automata

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 $\omega : Q \rightarrow \Sigma$ and $y = \omega(q)$.



- Strategy:
  - Compute the set of jointly reachable states.
  - Compare the output values of the two finite automata.

Comparison of Finite Automata

- Computation of the joint transition function:

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

- Computation of the reachable states (method according to previous slides):

$$\psi_Q(q_1,q_2)$$

- Computation of the 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:

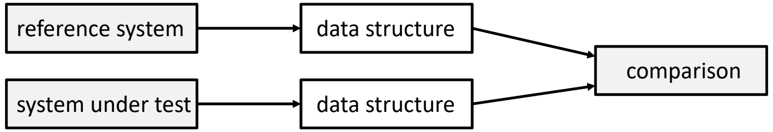
$$\exists y_1,y_2 \,:\, \psi_Y(y_1,y_2) \cdot (y_1 \neq y_2)$$

Overview

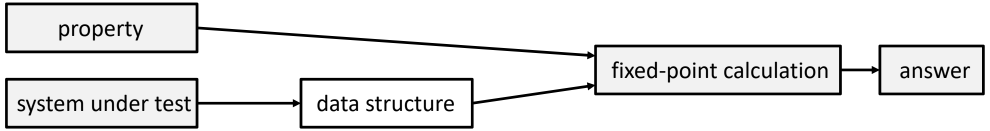
- Introduction
- Binary Decision Diagrams
  - Representation of Boolean Functions
  - Comparing two circuits
  - Representation of Sets
- **Finite Automata**
  - Reachability of States
  - Comparing two finite automata
  - **Proving properties of finite automata**
    - **Computation Tree Logic (CTL)**
    - Evaluating formulas
    - Verification of finite automata

Verification Scenarios

- Comparison of specification and implementation



- Proving properties



Verification of Finite Automata - CTL

- Verify properties of a finite automaton, for example
  - Can we always reset the automaton?
  - Is every request followed by an acknowledgement?
  - Are both outputs always equivalent?

- Specification of the query in a formula of temporal logic. We use a simple form that is denoted as Computation Tree Logic (CTL).

The printer is busy.
The light is on.

- Let us start with a minimal set of operators.
  - Any atomic proposition is a CTL formula.
  - Suppose that ϕ_1, ϕ_2 are CTL formula. Then the following are as well:

$$\neg\phi_1, \quad \phi_1 + \phi_2, \quad \mathrm{EX}\phi_1, \quad \mathrm{EG}\phi_1, \quad \phi_1\mathrm{EU}\phi_2$$

Verification of Finite Automata - CTL

- What is the meaning of the quantifiers?
    - Eϕ: “There exists at least one path from the current state where ϕ holds”
    - Xϕ: “ϕ has to hold at the next state”
    - Gϕ: “ϕ has to hold on the entire subsequent path”
    - ϕ_1Uϕ_2: “ϕ_1 has to hold *at least* until at some state ϕ_2 holds”

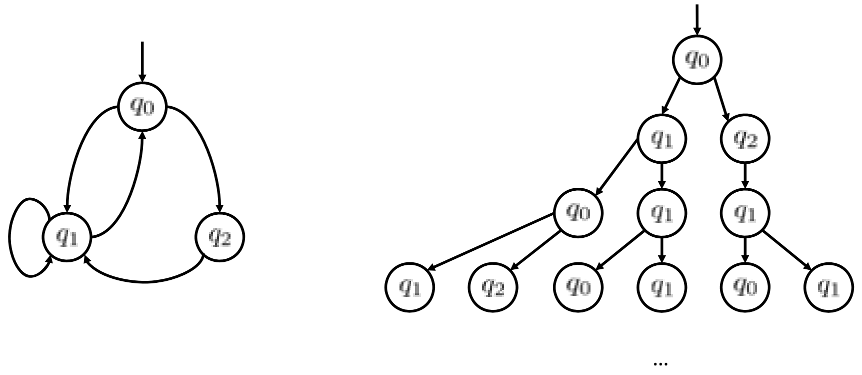
- There are more quantifiers, but they can be replaced by the above ones:
    - Fϕ: “ϕ eventually has to hold (somewhere on the subsequent path)
    - Aϕ: “ϕ has to hold on all paths starting from the current state“

- Some rules: $\mathrm{AF}\phi \equiv \neg\mathrm{EG}(\neg\phi)$ $\quad \mathrm{AX}\phi \equiv \neg\mathrm{EX}(\neg\phi)$ $\quad \mathrm{AG}\phi \equiv \neg\mathrm{EF}(\neg\phi)$

  $\mathrm{EF}\phi \equiv \mathrm{true\,EU}\phi$ $\quad \phi_1\mathrm{AU}\phi_2 = \neg([(\neg\phi_1)\mathrm{EU}(\neg(\phi_1+\phi_2))] + \mathrm{EG}(\neg\phi_2))$

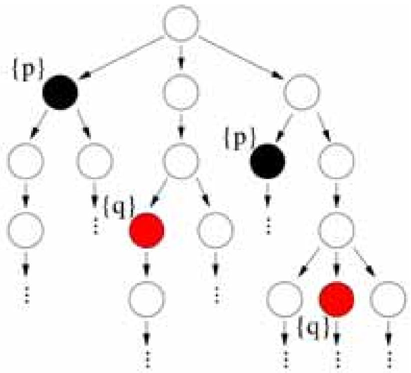
Verification of Finite Automata - CTL

- Visualization of a transition system and its computation tree:

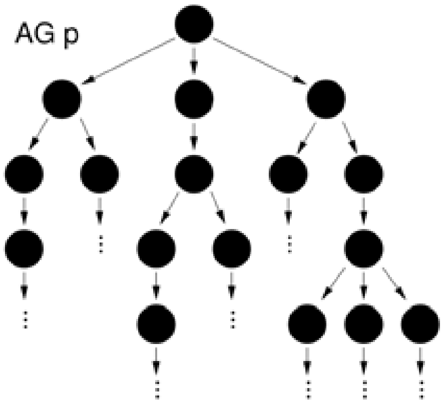


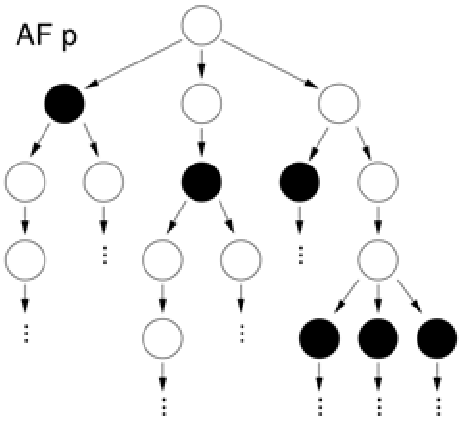
Verification of Finite Automata - CTL

- We use the depicted computation tree as a running example.
- Moreover, we suppose that the black states satisfy *p* and the red states satisfy *q*. Then, the topmost state satisfies the given formula in the examples.

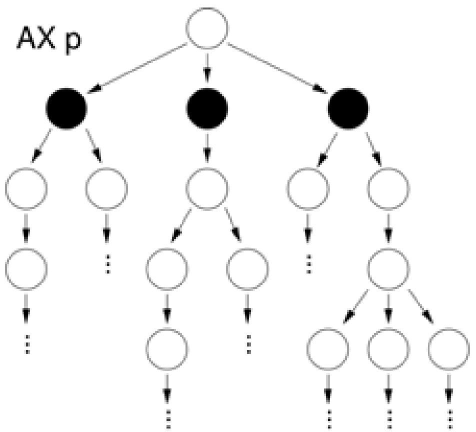


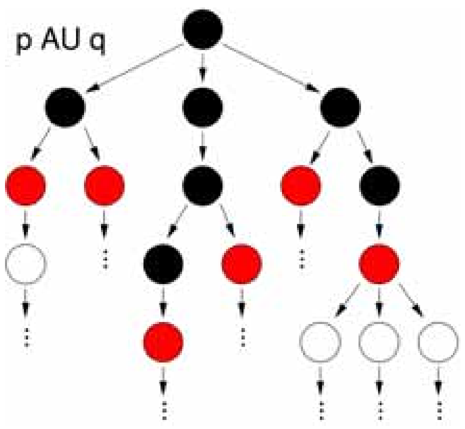
Verification of Finite Automata - CTL



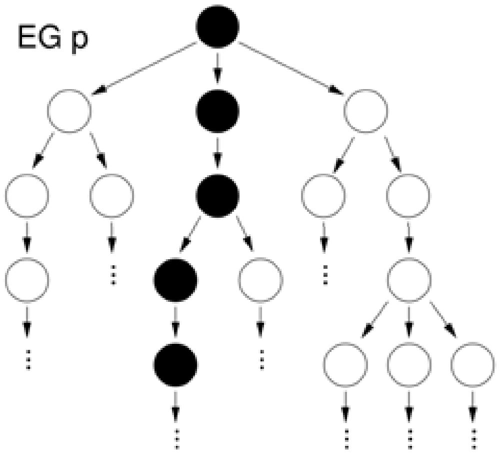


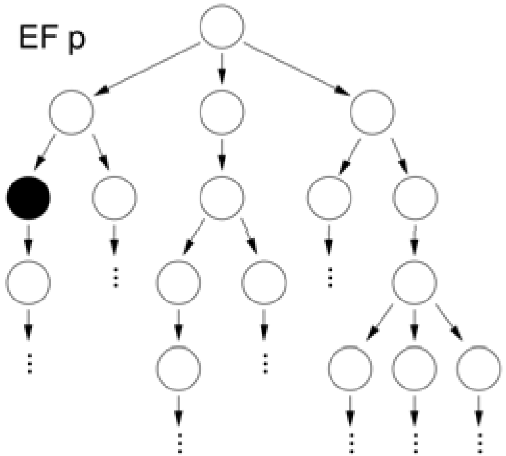
Verification of Finite Automata - CTL



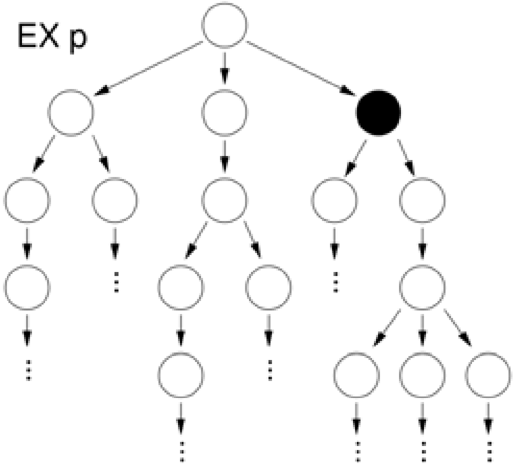


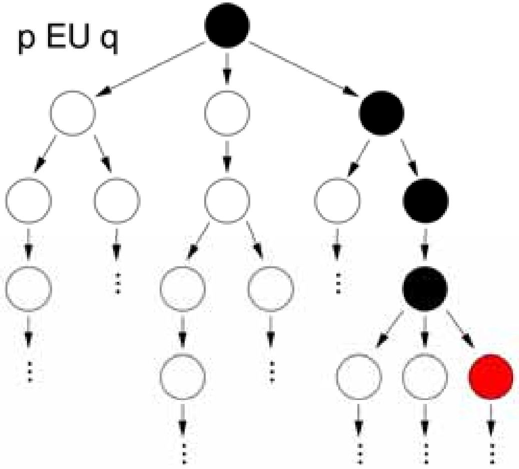
Verification of Finite Automata - CTL





Verification of Finite Automata - CTL





Verification of Finite Automata - CTL

Let "P" mean "I like chocolate" and Q mean "It's warm outside.“

- “AG P”:
- “EF P”:
- “AF EG P”:
- “EG AF P”:

- “P AU Q”:

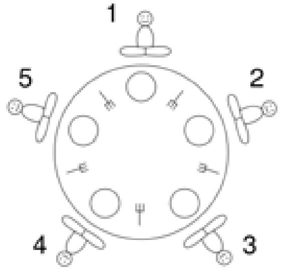
Verification of Finite Automata - CTL

Let "P" mean "I like chocolate" and Q mean "It's warm outside.“

- “AG P”: I will like chocolate from now on, no matter what happens.
- “EF P”: It's possible I may like chocolate some day, at least for one day.
- “AF EG P”: It's always possible (AF) that I will suddenly start liking chocolate for the rest of time.
- “EG AF P”: This is a critical time in my life. Depending on what happens next (E), it's possible that for the rest of time (G), there will always be some time in the future (AF) when I will like chocolate. However, if the wrong thing happens next, then all bets are off and there's no guarantee about whether I'll ever like chocolate.
- “P AU Q”: No matter what happens, I will like chocolate from now on. But when it gets warm outside, I don’t know whether I still like it.

Verification of Finite Automata - CTL

- Example Dining Philosophers: Five philosophers are sitting around a table, taking turns at thinking and eating.



- We shall express a couple of properties in CTL. Let us assume the following atomic propositions:
  - e_i : philosopher *i* is currently eating

Verification of Finite Automata - CTL

- “Philosophers 1 and 4 will never eat at the same time.”
- “Always every philosopher will get infinitely many turns to eat.”
- “Philosopher 2 will be the first to eat.”

Verification of Finite Automata - CTL

- “Philosophers 1 and 4 will never eat at the same time.”

$$\mathrm{AG}\neg(e_1 \cdot e_4)$$

- “Always every philosopher will get infinitely many turns to eat.”
- “Philosopher 2 will be the first to eat.”

Verification of Finite Automata - CTL

- “Philosophers 1 and 4 will never eat at the same time.”

$$\mathrm{AG}\neg(e_1 \cdot e_4)$$

- “Always every philosopher will get infinitely many turns to eat.”

$$\mathrm{AG}(\mathrm{AF}e_1 \cdot \mathrm{AF}e_2 \cdot \mathrm{AF}e_3 \cdot \mathrm{AF}e_4 \cdot \mathrm{AF}e_5)$$

- “Philosopher 2 will be the first to eat.”

Verification of Finite Automata - CTL

- “Philosophers 1 and 4 will never eat at the same time.”

$$\mathrm{AG}\neg(e_1 \cdot e_4)$$

- “Always every philosopher will get infinitely many turns to eat.”

$$\mathrm{AG}(\mathrm{AF}e_1 \cdot \mathrm{AF}e_2 \cdot \mathrm{AF}e_3 \cdot \mathrm{AF}e_4 \cdot \mathrm{AF}e_5)$$

- “Philosopher 2 will be the first to eat.”

$$\neg(e_1 + e_3 + e_4 + e_5)\,\mathrm{AU}\,e_2$$

Overview

- Introduction
- Binary Decision Diagrams
  - Representation of Boolean Functions
  - Comparing two circuits
  - Representation of Sets
- **Finite Automata**
  - Reachability of States
  - Comparing two finite automata
  - **Proving properties of finite automata**
    - Computation Tree Logic (CTL)
    - **Evaluating formulas**
    - Verification of finite automata

Verification of Finite Automata

- In order to compute CTL formula, we first define $[\![\phi]\!]$ as the set of all initial states of the finite automaton for which CTL formula ϕ is true. Then we can say that a finite automaton with initial state q_0 satisfies ϕ iff

$$q_0 \in [\![\phi]\!]$$

- Now, we can use our “trick”: computing with sets of states!
  - $\psi_{[\![\phi]\!]}(q)$ is true if the state q is in the set $[\![\phi]\!]$, i.e., it is an initial state for which the CTL formula is true.
  - Therefore, we can also say

$$q_0 \in [\![\phi]\!] \quad \equiv \quad \psi_{[\![\phi]\!]}(q_0)$$

characteristic function of the set $[\![\phi]\!]$

- When we compute the CTL-formulas, we start from the innermost terms.
- We suppose that every state has at least one successor state (could be itself).

Verification of Finite Automata

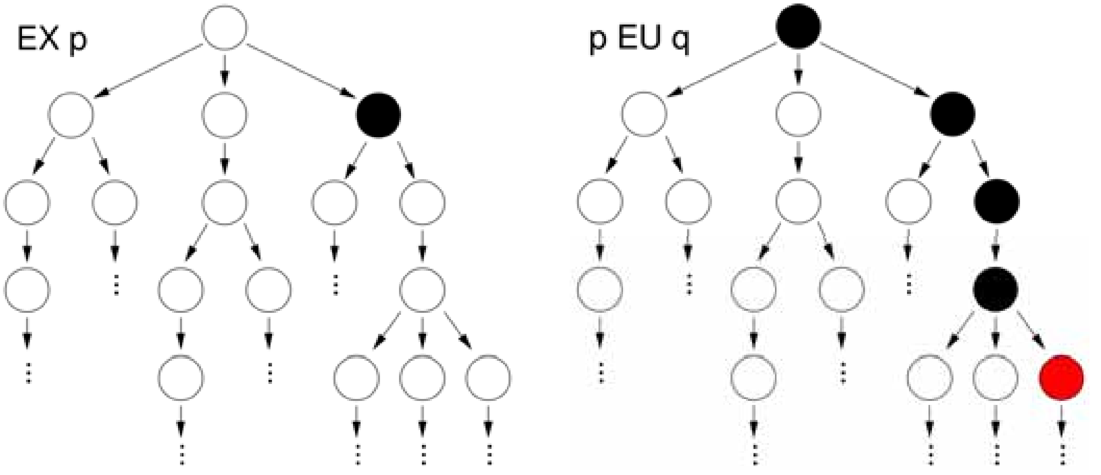
- We now show how to compute some operators in CTL. All others can be determined using the equivalence relations between operators that we listed earlier.
  - $\mathrm{EX}\phi$: Let us first define the set of predecessor states of *Q*, i.e., the set of states that lead in one transition to a state in *Q*:

$$Q' = \mathrm{Pre}(Q, \delta) = \{q' \mid \exists q \text{ with } \psi_\delta(q', q) \cdot \psi_Q(q)\}$$

Suppose that *Q* is the set of initial states for which the formula ϕ is true. Then we can write

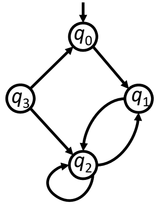
$$\begin{tikzcd}
\text{sets } \{ & \boxed{Q = \llbracket \phi \rrbracket} \arrow[r] \arrow[d, leftrightarrow] & \boxed{Q' = \llbracket \mathrm{EX}\phi \rrbracket = \mathrm{Pre}(\llbracket \phi \rrbracket, \delta)} \arrow[d, leftrightarrow] \\
\text{characteristic functions } \{ & \boxed{\psi_Q(q)} \arrow[r] & \boxed{\psi_{Q'}(q') = (\exists q \ : \ \psi_Q(q) \cdot \psi_\delta(q', q))}
\end{tikzcd}$$

Verification of Finite Automata - CTL



Verification of Finite Automata

- Example for $\mathrm{EX}\phi$: Compute EX q_2



$$\llbracket q_2 \rrbracket = \{q_2\}$$

$$Q' = \llbracket \mathrm{EX} q_2 \rrbracket = \mathrm{Pre}(\{q_2\}, \delta) = \{q_1, q_2, q_3\}$$

$$\{q' \mid \exists q \text{ with } \psi_Q(q) \cdot \psi_\delta(q', q)\} = \{q_1, q_2, q_3\}$$

As $q_0 \notin \llbracket \mathrm{EX} q_2 \rrbracket = \{q_1, q_2, q_3\}$, the CTL formula EX q_2 is not true.

Verification of Finite Automata

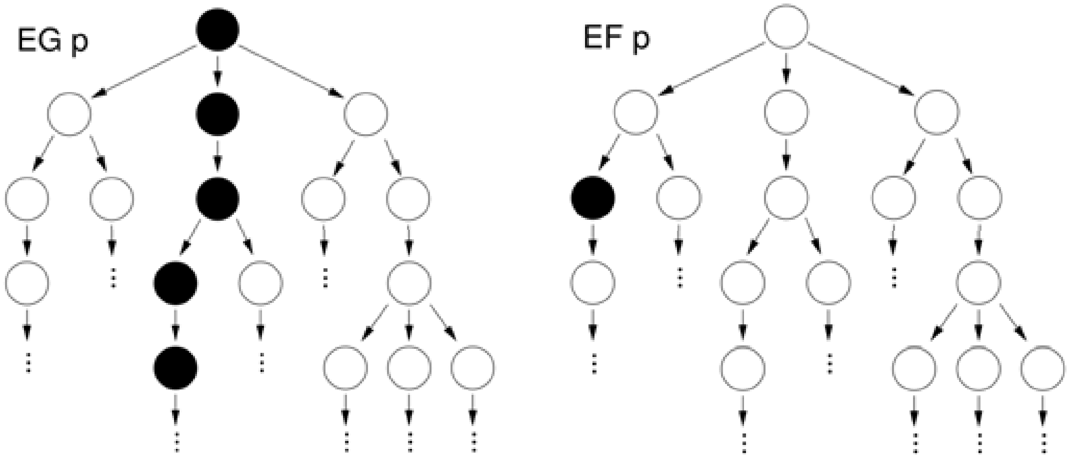
- $\mathrm{EF}\phi$: The idea here is to start with the set of initial states for which the formula ϕ is true. Then we add to this set the set of predecessor states. For the resulting set of states we do the same, …., until we reach a fixed-point. The corresponding operations can be done using BDDs (as described before).

$$Q_0 = \llbracket \phi \rrbracket$$

$$Q_i = Q_{i-1} \cup \mathrm{Pre}(Q_{i-1}, \delta) \qquad \text{for all } i > 1 \text{ until a fixed-point } Q' \text{ is reached}$$

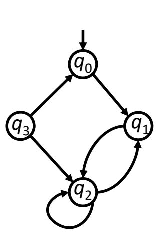
$$\llbracket \mathrm{EF}\phi \rrbracket = Q'$$

Verification of Finite Automata - CTL



Verification of Finite Automata

- Example for $\mathrm{EF}\phi$: Compute EF q_2

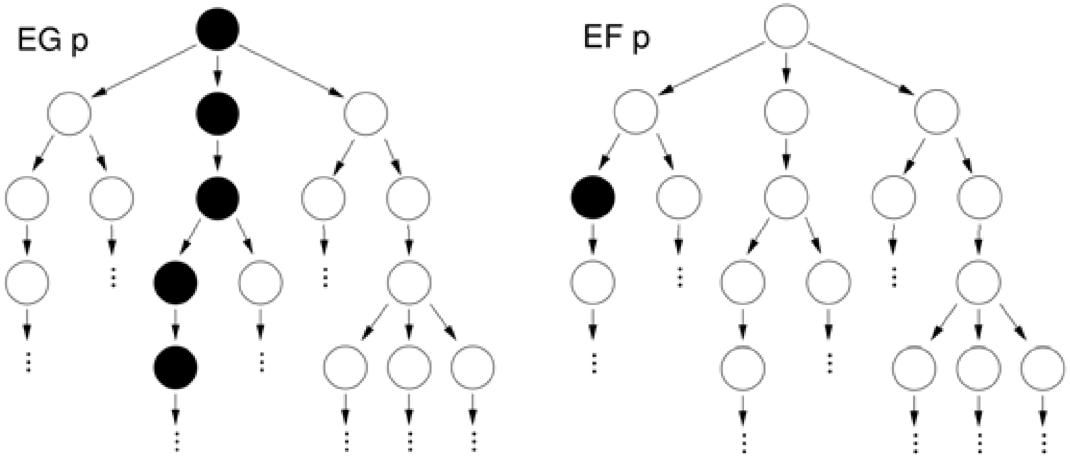


$$\{q' \mid \exists q \text{ with } \psi_Q(q) \cdot \psi_\delta(q', q)\} = \{q_1, q_2, q_3\}$$

$$Q_0 = [\![q_2]\!] = \{q_2\}$$
$$Q_1 = \{q_2\} \cup \mathrm{Pre}(\{q_2\}, \delta) = \{q_1, q_2, q_3\}$$
$$Q_2 = \{q_1, q_2, q_3\} \cup \mathrm{Pre}(\{q_1, q_2, q_3\}, \delta) = \{q_0, q_1, q_2, q_3\}$$
$$Q_3 = \{q_0, q_1, q_2, q_3\} \cup \mathrm{Pre}(\{q_0, q_1, q_2, q_3\}, \delta) = \{q_0, q_1, q_2, q_3\}$$
$$[\![\mathrm{EF}q_2]\!] = Q_3 = \{q_0, q_1, q_2, q_3\}$$

As $q_0 \in [\![\mathrm{EF}q_2]\!] = \{q_0, q_1, q_2, q_3\}$, the CTL formula EF q_2 is true.

Verification of Finite Automata - CTL



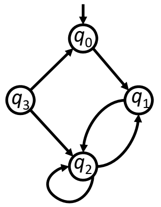
Verification of Finite Automata

- $\mathrm{EG}\phi$: The idea here is to start with the set of initial states for which the formula ϕ is true. Then we cut this set with the set of predecessor states. For the resulting set of states we do the same, …., until we reach a fixed-point. The corresponding operations can be done using BDDs (as described before).

$$Q_0 = [\![\phi]\!]$$
$$Q_i = Q_{i-1} \cap \mathrm{Pre}(Q_{i-1}, \delta) \qquad \text{for all } i > 1 \text{ until a fixed-point is reached}$$

Verification of Finite Automata

- Example for $\mathrm{EG}\phi$: Compute EG q_2

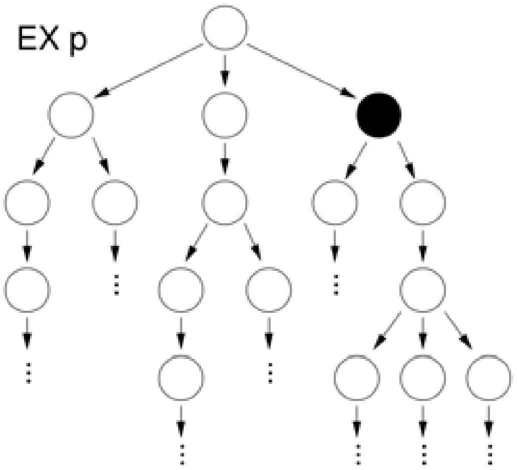


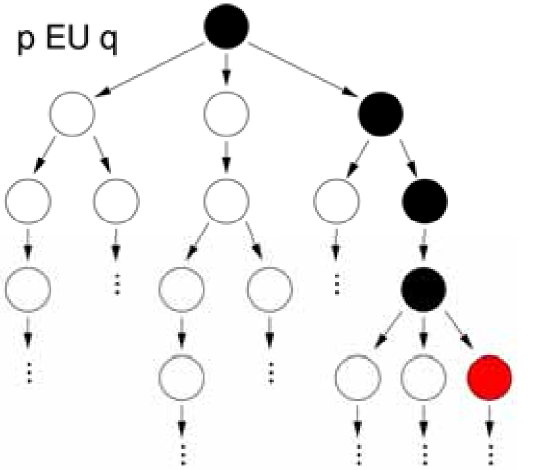
$\{q' \mid \exists q \text{ with } \psi_Q(q) \cdot \psi_\delta(q', q)\} = \{q_1, q_2, q_3\}$

$$Q_0 = [\![q_2]\!] = \{q_2\}$$
$$Q_1 = \{q_2\} \cap \mathrm{Pre}(\{q_2\}, \delta) = \{q_2\}$$
$$Q_2 = \{q_2\} \cap \mathrm{Pre}(\{q_2\}, \delta) = \{q_2\}$$
$$[\![\mathrm{EG}q_2]\!] = Q_2 = \{q_2\}$$

As $q_0 \notin [\![\mathrm{EG}q_2]\!] = \{q_2\}$, the CTL formula EG q_2 is not true.

Verification of Finite Automata - CTL





Verification of Finite Automata

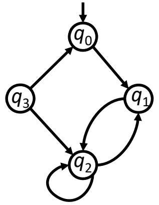
- $\phi_1 \mathrm{EU} \phi_2$: The idea here is to start with the set of initial states for which the formula ϕ_2 is true. Then we add to this set the set of predecessor states for which the formula ϕ_1 is true. For the resulting set of states we do the same, …., until we reach a fixed-point. The corresponding operations can be done using BDDs (as described before).

$$Q_0 = [\![\phi_2]\!]$$
$$Q_i = Q_{i-1} \cup (\mathrm{Pre}(Q_{i-1}, \delta) \cap [\![\phi_1]\!]) \qquad \text{for all } i > 1 \text{ until a fixed-point is reached}$$

Like $\mathrm{EF}\phi_2$, the only difference is that on our path backwards, we always make sure that also ϕ_1 holds.

Verification of Finite Automata

- Example for $\phi_1 \mathrm{EU} \phi_2$: Compute q_0 EU q_1



$\{q' \mid \exists q \text{ with } \psi_Q(q) \cdot \psi_\delta(q', q)\} = \{q_0, q_2\}$

$$Q_0 = [\![q_1]\!] = \{q_1\}$$
$$Q_1 = \{q_1\} \cup (\mathrm{Pre}(\{q_1\}, \delta) \cap \{q_0\}) = \{q_0, q_1\}$$
$$Q_2 = \{q_0, q_1\} \cup (\mathrm{Pre}(\{q_0, q_1\}, \delta) \cap \{q_0\}) = \{q_0, q_1\}$$
$$[\![q_0 \mathrm{EU} q_1]\!] = Q_2 = \{q_0, q_1\}$$

$\{q_0, q_2, q_3\}$

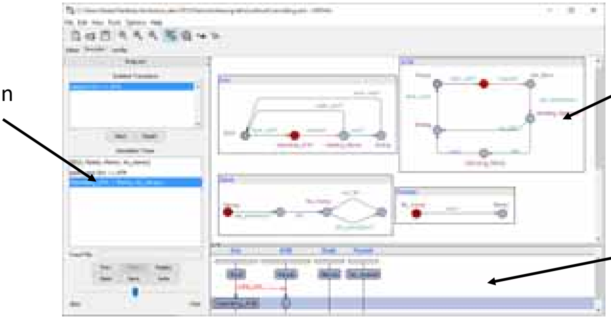
As $q_0 \in [\![q_0 \mathrm{EU} q_1]\!] = \{q_0, q_1\}$, the CTL formula q_0 EG q_1 is true.

Overview

- Introduction
- Binary Decision Diagrams
    - Representation of Boolean Functions
    - Comparing two circuits
    - Representation of Sets
- **Finite Automata**
    - Reachability of States
    - Comparing two finite automata
    - **Proving properties of finite automata**
        - Computation Tree Logic (CTL)
        - Evaluating formulas
        - **Verification of finite automata**

Verification of Finite Automata - Example

- Modeling and verification of a simple ATM-Money-Withdrawal protocol.
- We use the tool Uppaal
    - freely available
    - much more general modeling and verification possibilities than what we use here
    - can be used to verify timed behavior of discrete event systems



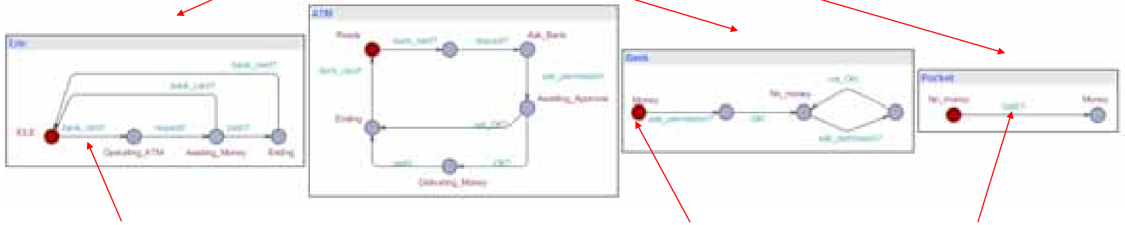
simulation trace

communicating finite automata

sequence diagram

ATM without Cancel

communicating finite automata

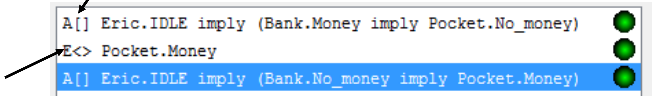


send event “bank_card”

initial state

enabled by event “cash”

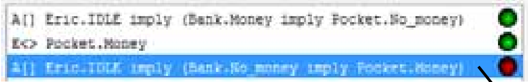
AG



```
A[] Eric.IDLE imply (Bank.Money imply Pocket.No_money)
E<> Pocket.Money
A[] Eric.IDLE imply (Bank.No_money imply Pocket.Money)
```

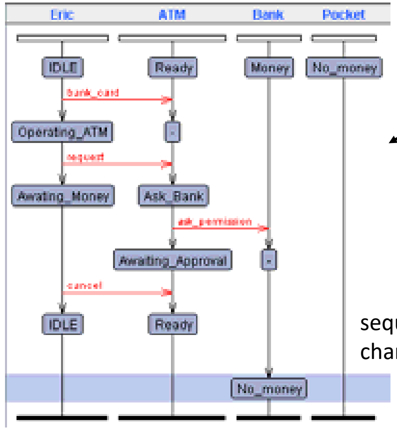
EF

ATM with Cancel

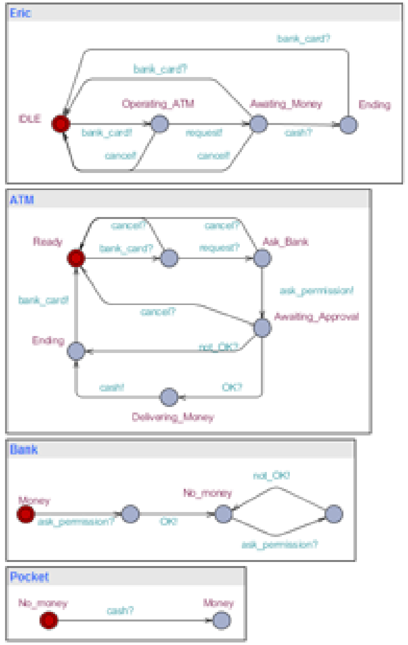


```
A[] Eric.IDLE imply (Bank.Money imply Pocket.No_money)
E<> Pocket.Money
A[] Eric.IDLE imply (Bank.No_money imply Pocket.Money)
```

counter example



sequence chart


Eric
bank_card?
bank_card?
Operating_ATM
Awaiting_Money
Ending
IDLE
bank_card!
request!
cash?
cancel!
cancel!
ATM
cancel?
cancel?
Ready
bank_card?
request?
Ask_Bank
bank_card!
ask_permission!
cancel?
Awaiting_Approval
Ending
not_OK?
cash!
OK?
Delivering_Money
Bank
not_OK!
No_money
Money
ask_permission?
OK!
ask_permission?
Pocket
No_money
cash?
Money