Advanced modelling techniques

Formal verification, temporal logics, model-checking

Laure Petrucci

Université Paris 13

Objectives of the module

- introduce formal models for critical systems specification
  - automata
  - Petri nets
  - their extensions
- use model-checking to verify their properties
  - reachability
  - deadlocks
  - properties expressed in LTL and CTL logics

Outline

1. Automata
   - Introductory notions
     - Automata
     - Execution and execution tree
     - Atomic properties
   - Formal definitions
     - Automata
     - Behaviour
   - Extensions of automata
     - Automata with variables
     - Synchronised product of automata
     - Synchronisation by message passing
2. Temporal logic
   - Language
   - LTL
     - Formal syntax and semantics
     - Illustration
     - Examples of LTL formulae
   - CTL
     - Formal syntax and semantics
     - Illustration
     - Examples of CTL formulae
3. Model-checking
   - CTL model-checking
   - LTL model-checking
4. Symbolic model-checking
   - Computation of state sets
   - Binary Decision Diagrams
   - Automata representation
5. Reachability Properties
   - Reachability in temporal logic
   - Computation of the reachability

Outline

1 Automata
- Introductory notions
  - Automata
  - Execution and execution tree
  - Atomic properties
- Formal definitions
  - Automata
  - Behaviour
- Extensions of automata
  - Automata with variables
  - Synchronised product of automata
  - Synchronisation by message passing

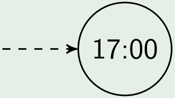
Automata

Intuitively, an automaton is a machine evolving from one state to another under the action of transitions.

Automata

Intuitively, an automaton is a machine evolving from one state to another under the action of transitions.

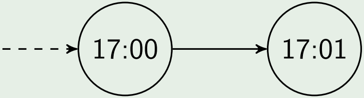
Example: Digital clock



Automata

Intuitively, an automaton is a machine evolving from one state to another under the action of transitions.

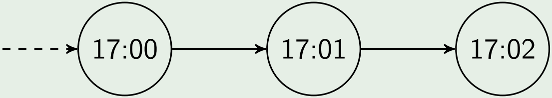
Example: Digital clock



Automata

Intuitively, an automaton is a machine evolving from one state to another under the action of transitions.

Example: Digital clock



Automata

Intuitively, an automaton is a machine evolving from one state to another under the action of transitions.

Example: Digital clock

Automata

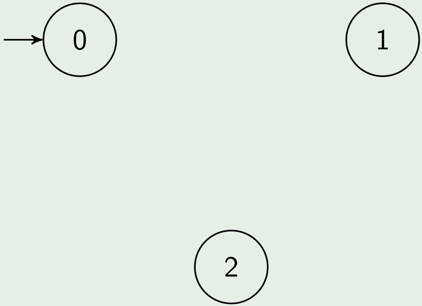
Example: Modulo 3 counter

- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement

Automata

Example: Modulo 3 counter

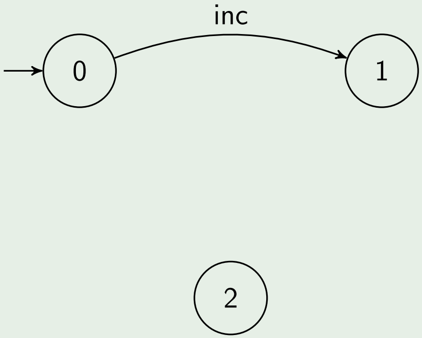
- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement



Automata

Example: Modulo 3 counter

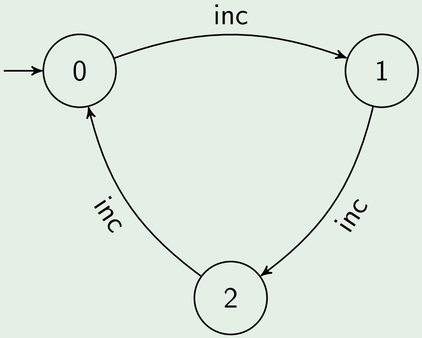
- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement



Automata

Example: Modulo 3 counter

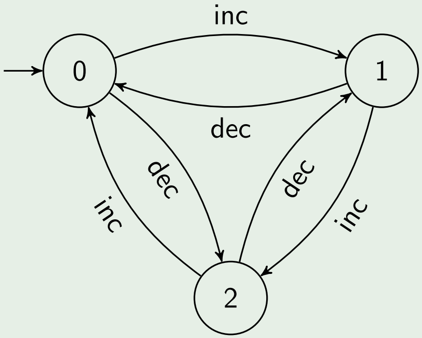
- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement



Automata

Example: Modulo 3 counter

- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement



Automata

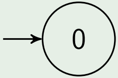
Example: Digicode

- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again

Automata

Example: Digicode

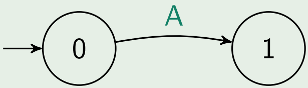
- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again



Automata

Example: Digicode

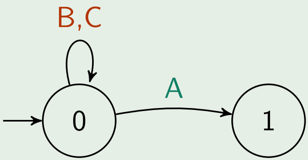
- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again



Automata

Example: Digicode

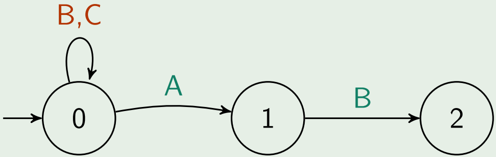
- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again



Automata

Example: Digicode

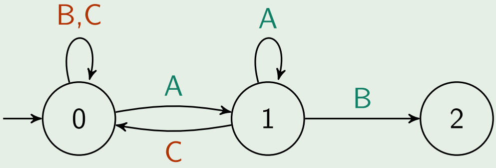
- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again



Automata

Example: Digicode

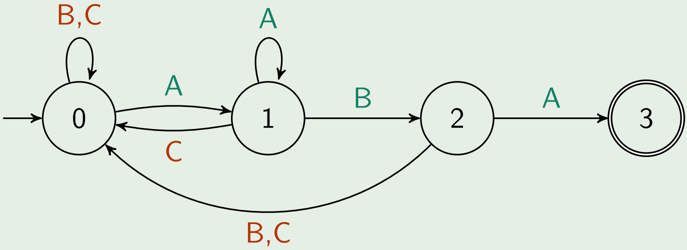
- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again



Automata

Example: Digicode

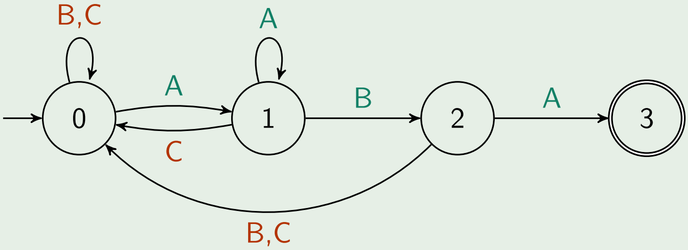
- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again



Automata

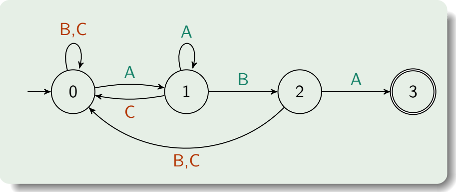
Example: Digicode

- 3 keys A, B, C
- code to open door ABA
- if the wrong key is pressed the whole operation has to start again



Remark: The numbers in the states are the number of correct keys that have already been pressed.

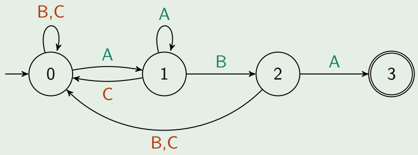
Executions of a model



Execution

An execution is a sequence of states describing a possible evolution of the system.

Executions of a model

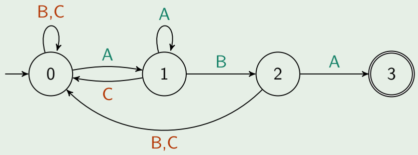


Execution

An execution is a sequence of states describing a possible evolution of the system.

- 0123
- 001201
- 001123

Executions of a model



Execution

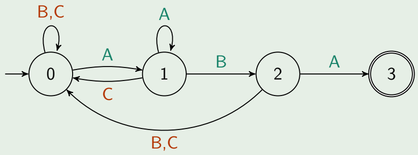
An execution is a sequence of states describing a possible evolution of the system.

- 0123
- 001201
- 001123

Questions

- Which executions lead to opening the door?
- Is there a possible infinite execution?

Executions of a model



Execution

An execution is a sequence of states describing a possible evolution of the system.

- 0123
- 001201
- 001123

Questions

- Which executions lead to opening the door?
- Is there a possible infinite execution?

- All those that end in state 3
- For example 00000000...

Execution tree

A tree to represent all possible executions

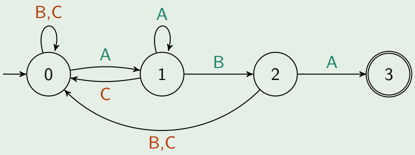
- root: initial state of the automaton
- children of a node: its immediate successors (states accessible from the node in one step)

Execution tree

A tree to represent all possible executions

- root: initial state of the automaton
- children of a node: its immediate successors (states accessible from the node in one step)

The digicode example

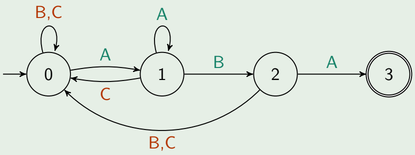


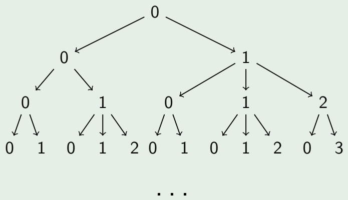
Execution tree

A tree to represent all possible executions

- root: initial state of the automaton
- children of a node: its immediate successors (states accessible from the node in one step)

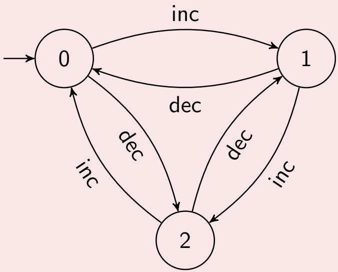
The digicode example





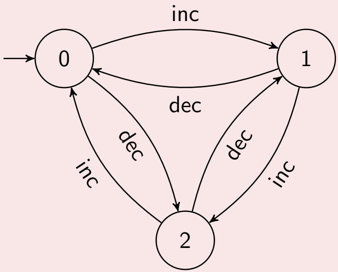
Exercise

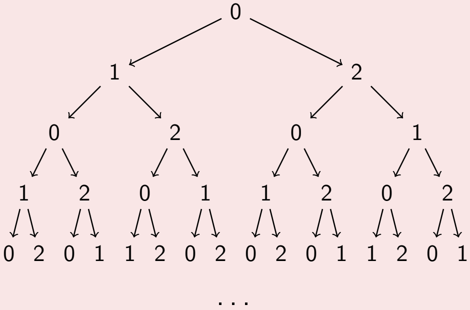
Execution tree for the modulo 3 counter



Exercise

Execution tree for the modulo 3 counter





Atomic properties

- Atomic properties are elementary properties known to be true or false
- some atomic properties can be associated with each state
- used to define more complex properties

Atomic properties

- Atomic properties are elementary properties known to be true or false
- some atomic properties can be associated with each state
- used to define more complex properties

Digicode atomic properties

- P_A: A has just been pressed
- P_B: B has just been pressed
- P_C: C has just been pressed

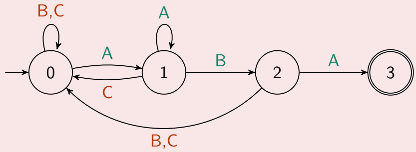
Atomic properties

- Atomic properties are elementary properties known to be true or false
- some atomic properties can be associated with each state
- used to define more complex properties

Digicode atomic properties

- P_A: A has just been pressed
- P_B: B has just been pressed
- P_C: C has just been pressed

Associate properties with states



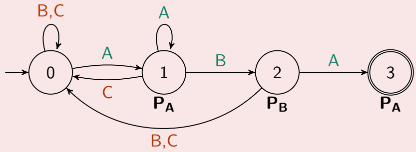
Atomic properties

- Atomic properties are elementary properties known to be true or false
- some atomic properties can be associated with each state
- used to define more complex properties

Digicode atomic properties

- P_A: A has just been pressed
- P_B: B has just been pressed
- P_C: C has just been pressed

Associate properties with states



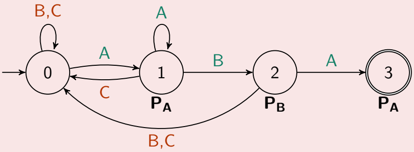
Atomic properties

- Atomic properties are elementary properties known to be true or false
- some atomic properties can be associated with each state
- used to define more complex properties

Digicode atomic properties

- P_A: A has just been pressed
- P_B: B has just been pressed
- P_C: C has just been pressed

Associate properties with states



Prove the correct code was entered when the door opens

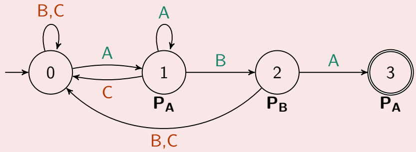
Atomic properties

- Atomic properties are elementary properties known to be true or false
- some atomic properties can be associated with each state
- used to define more complex properties

Digicode atomic properties

- P_A: A has just been pressed
- P_B: B has just been pressed
- P_C: C has just been pressed

Associate properties with states



Prove the correct code was entered when the door opens

The door is open only in state 3. Its only predecessor is 2 and transition A is used from state 2 to state 3. So A is the last key pressed.
The only predecessor of 2 is 1, and transition B was used.
State 1 has two possible predecessors: 0 and 1, and both used A.
Therefore, the code entered ends with ABA.

Formal definition of automata

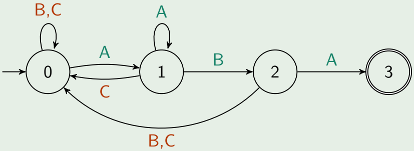
Automaton

Let *Prop* be a set of atomic propositions. An automaton is a tuple $\mathcal{A} = \langle Q, E, T, q_0, l, F \rangle$ such that:

- Q is a finite set of states
- E is a finite set of transition labels
- $T \subseteq Q \times E \times Q$ is a set of transitions
- q_0 is the initial state
- $l : Q \longrightarrow 2^{Prop}$ associates with each state a finite set of atomic propositions
- F is a set of final states

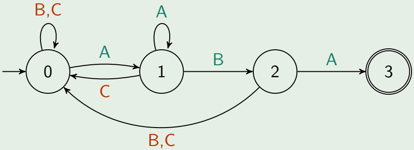
Example

The digicode example



Example

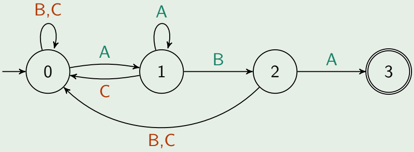
The digicode example



- $Q = \{0, 1, 2, 3\}$

Example

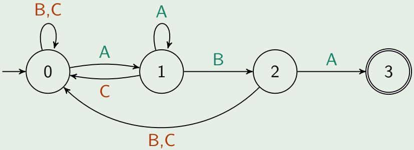
The digicode example



- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$

Example

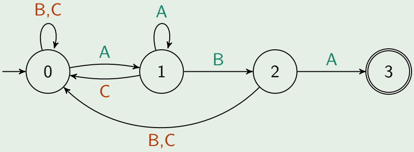
The digicode example



- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0)\}$

Example

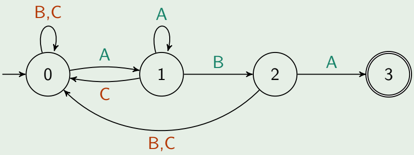
The digicode example



- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0)\}$
- $q_0 = 0$

Example

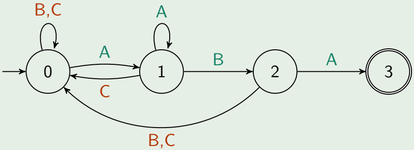
The digicode example



- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0)\}$
- $q_0 = 0$
- $Prop = \{P_A, P_B, P_C\}$

Example

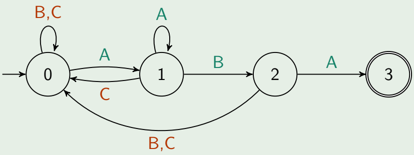
The digicode example



- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0)\}$
- $q_0 = 0$
- $Prop = \{P_A, P_B, P_C\}$
- $l(0) = \emptyset$, $l(1) = \{P_A\}$, $l(2) = \{P_B\}$, $l(3) = \{P_A\}$

Example

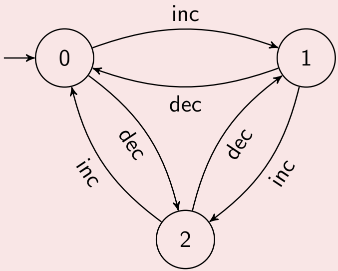
The digicode example



- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \{(0, A, 1), (0, B, 0), (0, C, 0), (1, A, 1), (1, B, 2), (1, C, 0), (2, A, 3), (2, B, 0), (2, C, 0)\}$
- $q_0 = 0$
- $Prop = \{P_A, P_B, P_C\}$
- $l(0) = \emptyset,\ l(1) = \{P_A\}, l(2) = \{P_B\},\ l(3) = \{P_A\}$
- $F = \{3\}$

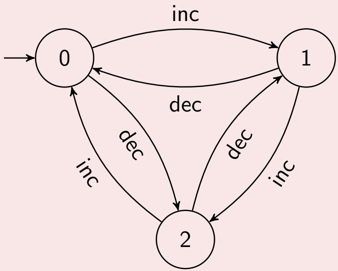
Exercise

Formal representation of the modulo 3 counter (no property)



Exercise

Formal representation of the modulo 3 counter (no property)



- $Q = \{0, 1, 2\}$
- $E = \{inc, dec\}$
- $T = \{(0, inc, 1), (0, dec, 2),$
  $(1, inc, 2), (1, dec, 0),$
  $(2, inc, 0), (2, dec, 1),$
- $q_0 = 0$
- $Prop = \emptyset$
- $l(0) = l(1) = l(2) = \emptyset$
- $F = \emptyset$

Behaviour

Runs (or paths)

- A run (or path) of an automaton $\mathcal{A}$ is a sequence σ of successive transitions (q_i, e_i, q'_i) of $\mathcal{A}$, i.e. such that $\forall i, q_{i+1} = q'_i$.
  $\sigma = q_1 \xrightarrow{e_1} q_2 \xrightarrow{e_2} q_3 \xrightarrow{e_3} q_4 \ldots$
- The length of a run σ is its number of transitions $|\sigma| \in \mathbb{N} \cup \{\omega\}$ where ω denotes infinity.
- The i^{th} state of σ is the state q_{i+1} reached after i transitions.

Behaviour

Runs (or paths)

- A run (or path) of an automaton $\mathcal{A}$ is a sequence σ of successive transitions (q_i, e_i, q'_i) of $\mathcal{A}$, i.e. such that $\forall i, q_{i+1} = q'_i$.
  $\sigma = q_1 \xrightarrow{e_1} q_2 \xrightarrow{e_2} q_3 \xrightarrow{e_3} q_4 \ldots$
- The length of a run σ is its number of transitions $|\sigma| \in \mathbb{N} \cup \{\omega\}$ where ω denotes infinity.
- The i^{th} state of σ is the state q_{i+1} reached after i transitions.

Executions

- A partial execution of $\mathcal{A}$ is a run starting from the initial state q_0.
- A complete execution of $\mathcal{A}$ is an execution that is maximal. It is either infinite or ends in a state where no transition is possible. This state might be final (in F), or a deadlock.
- A state is reachable if there exists an execution in which it appears.
- The complete executions define the behaviour of the automaton.

Exercise

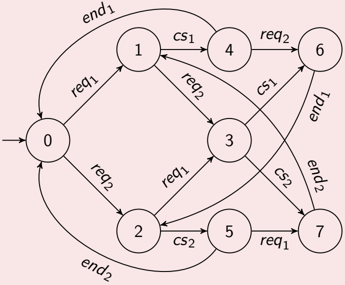
Mutual exclusion between two processes

- two processes execute and need access to the same resource
- each process can request access to a critical section of its code
- they must not execute this part at the same time
- when they have finished they signal they exit their critical section and loop back to their initial state

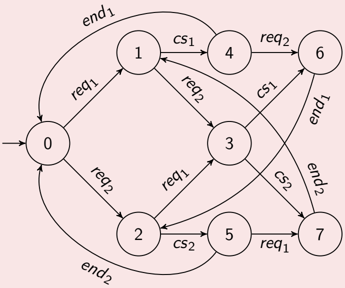
Questions

1. Model this problem with an automaton
2. Associate atomic properties with each state
3. Is the mutual exclusion requirement satisfied?
4. Is the system fair?
5. What would happen if you wanted to add a third process?

Exercise

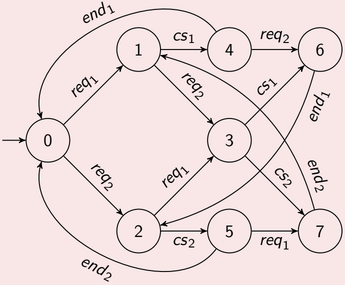


Exercise



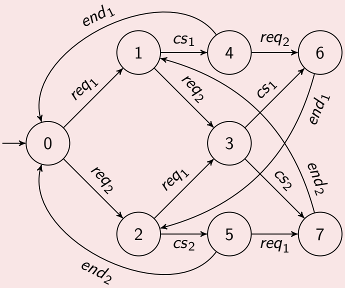
2. P_i: Process i is requesting access,
   C_i: Process i is in its critical section,
   R_i: Process i is at rest.
   P_1: states 1, 3, 7; P_2: states 2, 3, 6;
   C_1: states 4, 6; C_2: states 5, 7;
   R_1: states 0, 2, 5; R_2: states 0, 1, 4

Exercise



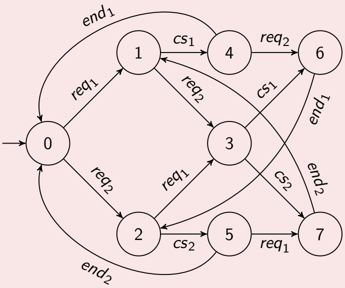
2. P_i: Process i is requesting access,
   C_i: Process i is in its critical section,
   R_i: Process i is at rest.
   P_1: states 1, 3, 7; P_2: states 2, 3, 6;
   C_1: states 4, 6; C_2: states 5, 7;
   R_1: states 0, 2, 5; R_2: states 0, 1, 4
3. Yes: no state has both C_1 and C_2

Exercise



2. P_i: Process i is requesting access,
   C_i: Process i is in its critical section,
   R_i: Process i is at rest.
   P_1: states 1, 3, 7; P_2: states 2, 3, 6;
   C_1: states 4, 6; C_2: states 5, 7;
   R_1: states 0, 2, 5; R_2: states 0, 1, 4
3. Yes: no state has both C_1 and C_2
4. No: run 0137137... never allows process 1 to enter its critical section

Exercise



2. P_i: Process i is requesting access,
   C_i: Process i is in its critical section,
   R_i: Process i is at rest.
   P_1: states 1, 3, 7; P_2: states 2, 3, 6;
   C_1: states 4, 6; C_2: states 5, 7;
   R_1: states 0, 2, 5; R_2: states 0, 1, 4
3. Yes: no state has both C_1 and C_2
4. No: run 0137137... never allows process 1 to enter its critical section
5. The number of states would blow up

Extension with variables

Why and how to use variables?

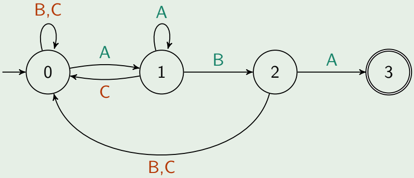
- more compact models, improving readability
- guards and assignments on transitions

Extension with variables

Why and how to use variables?

- more compact models, improving readability
- guards and assignments on transitions

Example: The digicode limited to 3 errors



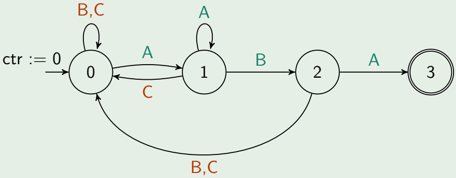
Extension with variables

Why and how to use variables?

- more compact models, improving readability
- guards and assignments on transitions

Example: The digicode limited to 3 errors

var ctr: int;



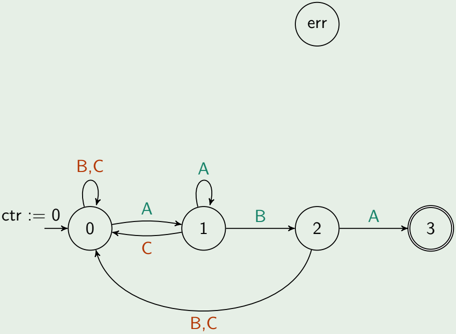
Extension with variables

Why and how to use variables?

- more compact models, improving readability
- guards and assignments on transitions

Example: The digicode limited to 3 errors

var ctr: int;



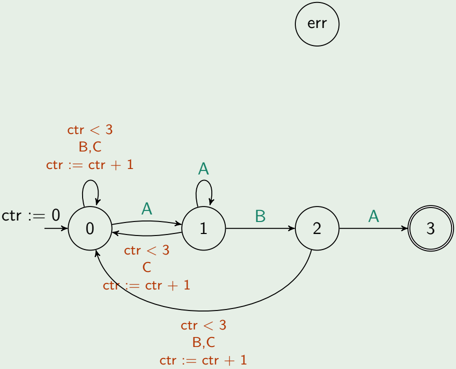
Extension with variables

Why and how to use variables?

- more compact models, improving readability
- guards and assignments on transitions

Example: The digicode limited to 3 errors

var ctr: int;



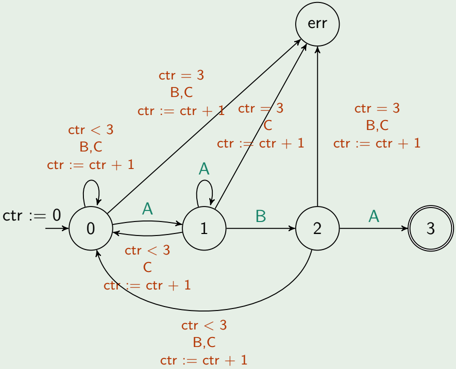
Extension with variables

Why and how to use variables?

- more compact models, improving readability
- guards and assignments on transitions

Example: The digicode limited to 3 errors

var ctr: int;



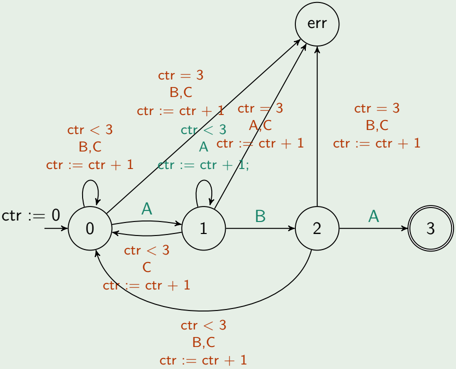
Extension with variables

Why and how to use variables?

- more compact models, improving readability
- guards and assignments on transitions

Example: The digicode limited to 3 errors

var ctr: int;

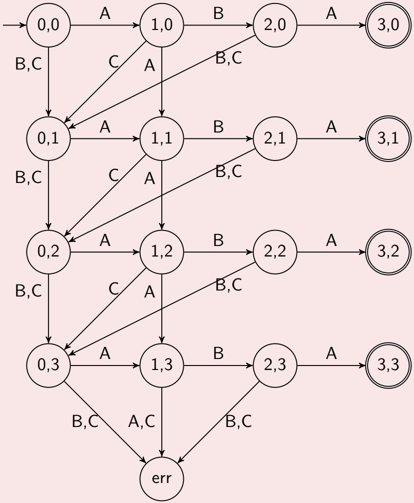


Extension with variables

Exercise: The digicode with 3 errors without variables

Extension with variables

Exercise: The digicode with 3 errors without variables



Synchronised product

Why?

- each component of the system is designed as an automaton
- composition of automata

Synchronised product

Why?

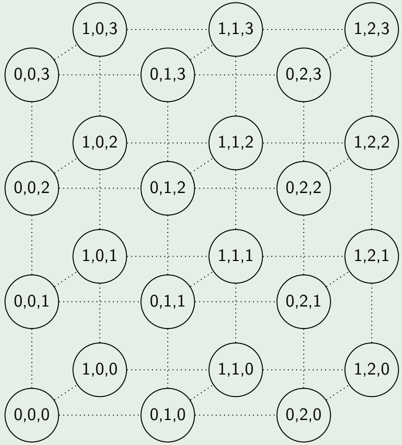
- each component of the system is designed as an automaton
- composition of automata

How?

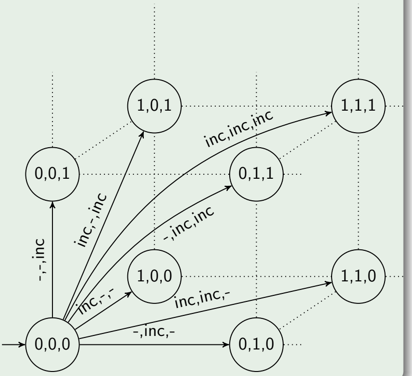
- independent actions lead to a cartesian product of states
- synchronised actions occur simultaneously

Synchronised product

3 counters, modulo 2, 3, 4: states

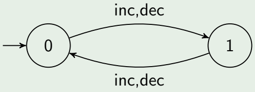


3 counters: some transitions

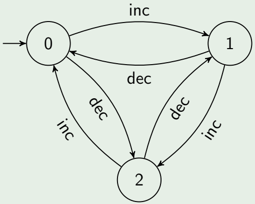


Example: Synchronised counters

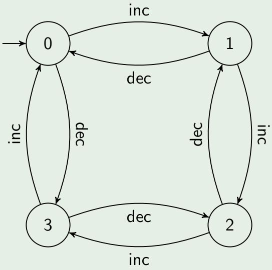
Modulo 2 counter



Modulo 3 counter

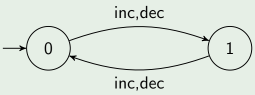


Modulo 4 counter

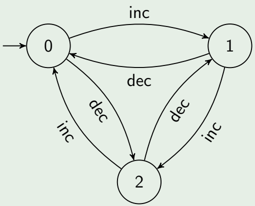


Example: Synchronised counters

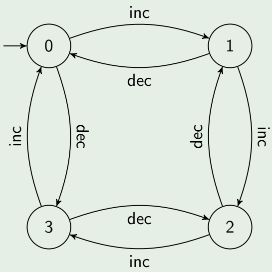
Modulo 2 counter



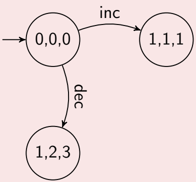
Modulo 3 counter



Modulo 4 counter

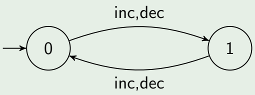


Synchronised actions: all counters increment or decrement simultaneously

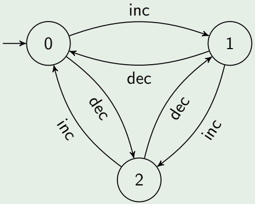


Example: Synchronised counters

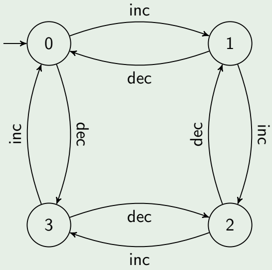
Modulo 2 counter



Modulo 3 counter



Modulo 4 counter



Synchronised actions: all counters increment or decrement simultaneously

Formal definition of the cartesian product

Let $(\mathcal{A}_i)_{1 \leq i \leq n}$ be a family of automata $\mathcal{A}_i = \langle Q_i, E_i, T_i, q_{0i}, l_i, F_i \rangle$.

Cartesian product of automata

The cartesian product $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ of the automata in the family is the automaton $\mathcal{A} = \langle Q, E, T, q_0, l, F \rangle$ such that :

- $Q = Q_1 \times \cdots \times Q_n$
- $E = \prod_{1 \leq i \leq n}(E_i \cup \{-\})$ (where $-$ represents an empty action)
- $$\begin{aligned} T = \; & \{((q_1, \ldots, q_n), (e_1, \ldots, e_n), (q'_1, \ldots, q'_n)) \mid \\ & \forall 1 \leq i \leq n, (e_i = - \wedge q'_i = q_i) \vee (e_i \neq - \wedge (q_i, e_i, q'_i) \in T_i)\} \end{aligned}$$
- $q_0 = (q_{01}, \ldots, q_{0n})$
- $\forall (q_1, \ldots, q_n) \in Q : l((q_1, \ldots, q_n)) = \bigcup_{1 \leq i \leq n} l_i(q_i)$
- $F = \{(q_1, \ldots, q_n) \in Q \mid \exists 1 \leq i \leq n, q_i \in F_i\}$

Formal definition of the synchronised product

Let $(\mathcal{A}_i)_{1 \leq i \leq n}$ be a family of automata $\mathcal{A}_i = \langle Q_i, E_i, T_i, q_{0i}, l_i, F_i \rangle$.

Synchronisation set

The synchronisation set, denoted *Sync* describes all permitted simultaneous actions:

$$Sync \subseteq \prod_{1 \leq i \leq n} (E_i \cup \{-\})$$

Formal definition of the synchronised product

Let $(\mathcal{A}_i)_{1 \leq i \leq n}$ be a family of automata $\mathcal{A}_i = \langle Q_i, E_i, T_i, q_{0_i}, l_i, F_i \rangle$.

Synchronisation set

The synchronisation set, denoted *Sync* describes all permitted simultaneous actions:

$$Sync \subseteq \prod_{1 \leq i \leq n} (E_i \cup \{-\})$$

Synchronised product of automata

The synchronised product of $(\mathcal{A}_i)_{1 \leq i \leq n}$ over a set *Sync* is the cartesian product restricted to $E = Sync$.

Synchronisation by message passing

Message passing: a special case of synchronised product

!m send a message m

?m receive a message m

- reception and sending occur simultaneously
- they concern the same message

Synchronisation by message passing

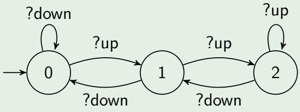
Example: a small lift

Model of a lift in a 3 floors building, composed of:

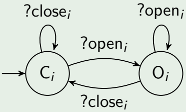
- **the cabin** which goes up and down according to the current floor and the lift controller commands
- **3 doors** (one per floor) which open and close according to the controller's commands
- **a controller** which operates the lift

Example: the lift

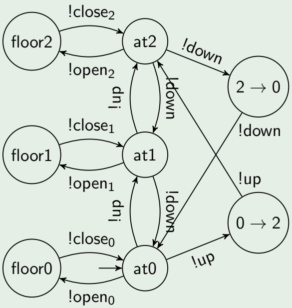
Cabin



i^{th} door

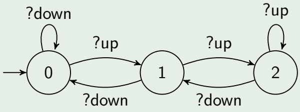


Controller

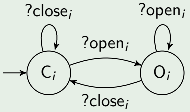


Example: the lift

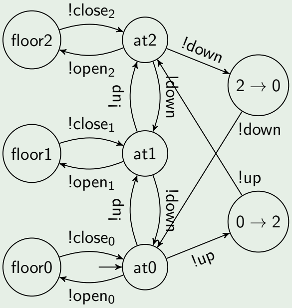
Cabin



i^{th} door



Controller



Properties

- A door on a floor cannot open while the cabin is on a different floor
- The cabin cannot move while one of the doors is open

Exercise

Mutual exclusion problem

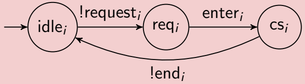
1. Model the mutual exclusion problem with message passing:
   - one automaton per participating process (2 processes)
   - a controller
2. How do you add a new process? Give the model for 3 processes, and explain how to generalise it to *n* processes

Exercise

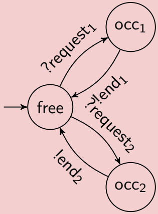
Mutual exclusion problem

1. Model the mutual exclusion problem with message passing:
   - one automaton per participating process (2 processes)
   - a controller
2. How do you add a new process? Give the model for 3 processes, and explain how to generalise it to *n* processes

process i, $i \in \{1, 2\}$



controller

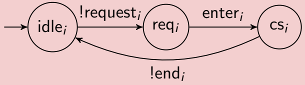


Exercise

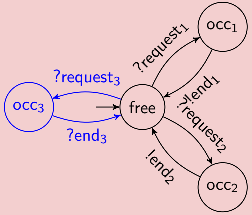
Mutual exclusion problem

1. Model the mutual exclusion problem with message passing:
   - one automaton per participating process (2 processes)
   - a controller
2. How do you add a new process? Give the model for 3 processes, and explain how to generalise it to *n* processes

process i, $i \in \{1, 2, 3\}$



controller

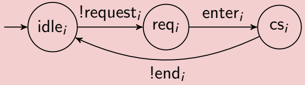


Exercise

Mutual exclusion problem

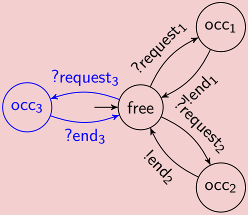
1. Model the mutual exclusion problem with message passing:
   - one automaton per participating process (2 processes)
   - a controller
2. How do you add a new process? Give the model for 3 processes, and explain how to generalise it to n processes

process i, $i \in \{1, 2, 3\}$



- n process automata
- controller: n states *occ*

controller



Outline

2 Temporal logic

- Language
- LTL
  - Formal syntax and semantics
  - Illustration
  - Examples of LTL formulae
- CTL
  - Formal syntax and semantics
  - Illustration
  - Examples of CTL formulae

Introduction to temporal logics

- express dynamic behaviour of the system
- use formal syntax and semantics to avoid any ambiguity
- capture statements and reasoning that involve the notion of order in time

Introduction to temporal logics

- express dynamic behaviour of the system
- use formal syntax and semantics to avoid any ambiguity
- capture statements and reasoning that involve the notion of order in time

The lift example

- any request must ultimately be satisfied
- the lift never traverses a floor for which a request is pending without satisfying the request

Introduction to temporal logics

- express dynamic behaviour of the system
- use formal syntax and semantics to avoid any ambiguity
- capture statements and reasoning that involve the notion of order in time

The lift example

- any request must ultimately be satisfied
  False: The lift can continuously go up and down without opening doors (run $(\text{at0},C_0,C_1,C_2,0)\xrightarrow{up}(\text{at1},C_0,C_1,C_2,1)\xrightarrow{up}(\text{at2},C_0,C_1,C_2,2)\xrightarrow{down}(\text{at1},C_0,C_1,C_2,1)\ldots$)
- the lift never traverses a floor for which a request is pending without satisfying the request

Introduction to temporal logics

- express dynamic behaviour of the system
- use formal syntax and semantics to avoid any ambiguity
- capture statements and reasoning that involve the notion of order in time

The lift example

- any request must ultimately be satisfied
  False: The lift can continuously go up and down without opening doors (run $(\mathrm{at0},C_0,C_1,C_2,0) \xrightarrow{up} (\mathrm{at1},C_0,C_1,C_2,1) \xrightarrow{up} (\mathrm{at2},C_0,C_1,C_2,2) \xrightarrow{down} (\mathrm{at1},C_0,C_1,C_2,1) \ldots$)
- the lift never traverses a floor for which a request is pending without satisfying the request
  False: consequence of the previous property

The language CTL*

- atomic propositions
- boolean combinators:
  - `true`, `false`
  - $\neg$ (negation)
  - $\wedge$ (and), $\vee$ (or)
  - $\implies$ (logical implication), $\iff$ (if and only if)
- temporal combinators:
  - X (neXt), F (Future), G (Globally)
  - U (Until), W (Weak until)
- quantifiers: A (Always), E (Exists)

The language CTL*

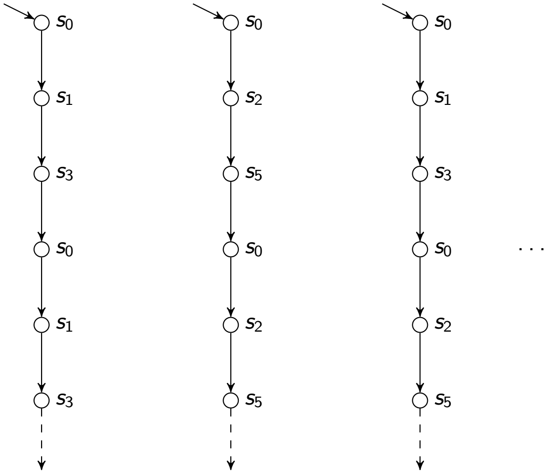
- atomic propositions
- boolean combinators:
  - `true`, `false`
  - $\neg$ (negation)
  - $\wedge$ (and), $\vee$ (or)
  - $\implies$ (logical implication), $\iff$ (if and only if)
- temporal combinators:
  - X (neXt), F (Future), G (Globally)
  - U (Until), W (Weak until)
- quantifiers: A (Always), E (Exists)

Main temporal logics

LTL Linear-time Temporal Logic

CTL Computation Tree Logic

LTL: Linear-time Temporal Logic



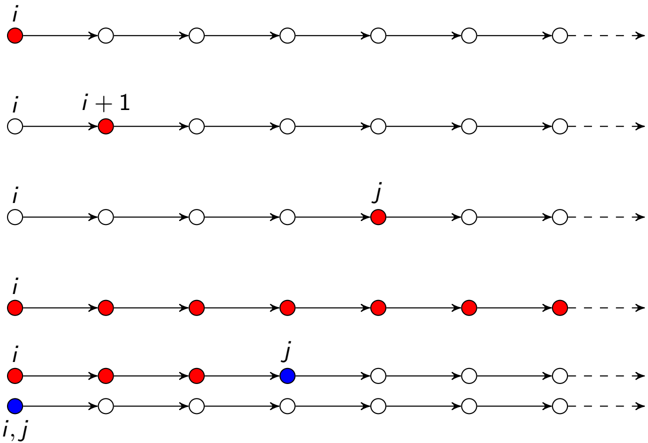
Semantics of LTL

Let σ be a run and $p \in Prop$ an atomic proposition.
$\sigma, i \models \phi$ denotes that at time i of its execution, σ satisfies formula ϕ.

$\sigma, i \models p$	iff	$p \in l(\sigma(i))$
$\sigma, i \models \neg\phi$	iff	$\sigma, i \not\models \phi$
$\sigma, i \models \phi \wedge \psi$	iff	$\sigma, i \models \phi$ and $\sigma, i \models \psi$
$\sigma, i \models \mathrm{X}\phi$	iff	$i < \lvert\sigma\rvert$ and $\sigma, i+1 \models \phi$
$\sigma, i \models \mathrm{F}\phi$	iff	$\exists j, i \leq j \leq \lvert\sigma\rvert : \sigma, j \models \phi$
$\sigma, i \models \mathrm{G}\phi$	iff	$\forall j, i \leq j \leq \lvert\sigma\rvert : \sigma, j \models \phi$
$\sigma, i \models \phi \mathrm{U} \psi$	iff	$\exists j, i \leq j \leq \lvert\sigma\rvert : \sigma, j \models \psi$ and $\forall k, i \leq k < j : \sigma, k \models \phi$

Illustration of the LTL semantics

- p
- $\mathrm{X}\phi$
- $\mathrm{F}\phi$
- $\mathrm{G}\phi$
- $\phi_1 \mathrm{U} \phi_2$



Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

Modulo 3 counter

1. $XXX0$
2. $F(1 \vee 2)$
3. $F1$

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

Modulo 3 counter

1. XXX0
   The third state reached is 0
2. $F(1 \vee 2)$
3. F1

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

Modulo 3 counter

1. XXX0
   The third state reached is 0
   All runs starting with 0120 or 0210
2. $F(1 \vee 2)$
3. F1

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

Modulo 3 counter

1. XXX0
   The third state reached is 0
   All runs starting with 0120 or 0210
2. $F(1 \vee 2)$
   In the future state 1 or state 2 will be reached
3. F1

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

Modulo 3 counter

1. XXX0
   The third state reached is 0
   All runs starting with 0120 or 0210
2. $F(1 \vee 2)$
   In the future state 1 or state 2 will be reached
   All runs
3. F1

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

Modulo 3 counter

1. XXX0
   The third state reached is 0
   All runs starting with 0120 or 0210
2. $F(1 \vee 2)$
   In the future state 1 or state 2 will be reached
   All runs
3. F1
   In the future state 1 will be reached

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

Modulo 3 counter

1. XXX0
   The third state reached is 0
   All runs starting with 0120 or 0210
2. $F(1 \vee 2)$
   In the future state 1 or state 2 will be reached
   All runs
3. F1
   In the future state 1 will be reached
   All runs containing 1, i.e. all runs except 020202...

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

The digicode

1. F3
2. G¬3

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

The digicode

1. F3
   The door can open
2. G¬3

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

The digicode

1. F3
   The door can open
   All runs ending in state 3
2. G¬3

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

The digicode

1. F3

   The door can open

   All runs ending in state 3

2. G¬3

   The door never opens

Examples of LTL formulae

- What do the following formulae mean?
- Which runs satisfy the LTL property?

The digicode

1. F3
   The door can open
   All runs ending in state 3
2. G¬3
   The door never opens
   All runs not ending in state 3

Exercises

Express $\vee$, $\implies$, $\iff$, W with $\neg$, $\wedge$, X, F, G, U

(W is similar to U but ψ may never happen)

Exercises

Express $\vee$, $\implies$, $\iff$, W with $\neg$, $\wedge$, X, F, G, U

(W is similar to U but ψ may never happen)

- $\phi \vee \psi \equiv \neg(\phi \wedge \psi)$
- $\phi \implies \psi \equiv \neg\phi \vee \psi$
- $\phi \iff \psi \equiv (\neg\phi \vee \psi) \wedge (\phi \vee \neg\psi)$
- $\phi \mathsf{W} \psi \equiv (\phi \mathsf{U} \psi) \vee \mathsf{G}\phi$

Exercises

Prove that:

1. $\mathsf{F}\phi \equiv \mathtt{true}\mathsf{U}\phi$

2. $\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$

Exercises

Prove that:

1. $\mathsf{F}\phi \equiv \mathtt{true}\mathsf{U}\phi$

$$\begin{aligned}\mathtt{true}\mathsf{U}\phi &\equiv \exists j, i \leq j \leq |\sigma| : \sigma, j \models \phi \wedge \forall k, i \leq k < j : \sigma, k \models \mathtt{true} \\ &\equiv \exists j, i \leq j \leq |\sigma| : \sigma, j \models \phi \\ &\equiv \mathsf{F}\phi\end{aligned}$$

2. $\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$

Exercises

Prove that:

1. $\mathsf{F}\phi \equiv \mathtt{true}\mathsf{U}\phi$

$$\begin{aligned} \mathtt{true}\mathsf{U}\phi &\equiv \exists j, i \leq j \leq |\sigma| : \sigma, j \models \phi \wedge \forall k, i \leq k < j : \sigma, k \models \mathtt{true} \\ &\equiv \exists j, i \leq j \leq |\sigma| : \sigma, j \models \phi \\ &\equiv \mathsf{F}\phi \end{aligned}$$

2. $\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$

$$\begin{aligned} \neg\mathsf{F}\neg\phi &\equiv \neg(\exists j, i \leq j \leq |\sigma| : \sigma, j \models \neg\phi) \\ &\equiv \forall j, i \leq j \leq |\sigma| : \sigma, j \not\models \neg\phi) \\ &\equiv \forall j, i \leq j \leq |\sigma| : \sigma, j \models \phi) \\ &\equiv \mathsf{G}\phi \end{aligned}$$

Exercises

Digicode

1. Write a LTL formula satisfied by all runs where keys A and B have successively been pressed
2. Write a LTL formula that characterises the infinite loop on state 0
3. Same question using atomic propositions P_A, P_B, P_C

Exercises

Digicode

1. Write a LTL formula satisfied by all runs where keys A and B have successively been pressed
   $\mathsf{F}(P_A \implies \mathsf{X}P_B)$
2. Write a LTL formula that characterises the infinite loop on state 0
3. Same question using atomic propositions P_A, P_B, P_C

Exercises

Digicode

1. Write a LTL formula satisfied by all runs where keys A and B have successively been pressed
   $\mathsf{F}(P_A \implies \mathsf{X}P_B)$
2. Write a LTL formula that characterises the infinite loop on state 0
   $\mathsf{G}0$
3. Same question using atomic propositions P_A, P_B, P_C

Exercises

Digicode

1. Write a LTL formula satisfied by all runs where keys A and B have successively been pressed
   $\mathsf{F}(P_A \implies \mathsf{X}P_B)$
2. Write a LTL formula that characterises the infinite loop on state 0
   $\mathsf{G}0$
3. Same question using atomic propositions P_A, P_B, P_C
   $\mathsf{G}\neg P_A$

Exercises

Mutual exclusion between two processes (synchronised product)

Write an LTL formula satisfied by all runs where:

1. The two processes are not simultaneously in their critical section

2. Whenever process 1 requests to enter its critical section, it will eventually succeed

Exercises

Mutual exclusion between two processes (synchronised product)

Write an LTL formula satisfied by all runs where:

1. The two processes are not simultaneously in their critical section
   $\mathrm{G}\neg(cs_1 \wedge cs_2)$
2. Whenever process 1 requests to enter its critical section, it will eventually succeed

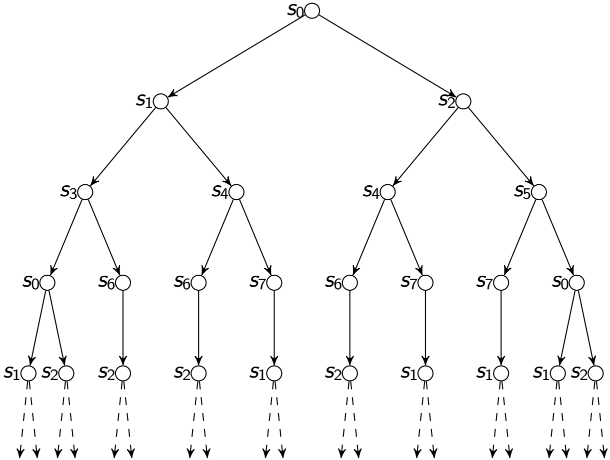
Exercises

Mutual exclusion between two processes (synchronised product)

Write an LTL formula satisfied by all runs where:

1. The two processes are not simultaneously in their critical section
   $\mathsf{G}\neg(cs_1 \wedge cs_2)$
2. Whenever process 1 requests to enter its critical section, it will eventually succeed
   $\mathsf{G}(req_1 \implies \mathsf{F}cs_1)$

CTL: Computation Tree Logic



Semantics of CTL

Same as LTL plus:

$$\boxed{\begin{array}{lll} \sigma, i \models \mathrm{E}\phi & \text{iff} & \exists \sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \text{ and } \sigma', i \models \phi \\ \sigma, i \models \mathrm{A}\phi & \text{iff} & \forall \sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \text{ we have } \sigma', i \models \phi \end{array}}$$

In CTL, each use of a temporal operator (X, F, G, U) is in the immediate scope of a quantifier (E, A)

This restriction does not apply in CTL*

Illustration of the CTL semantics (1/8)

EXϕ

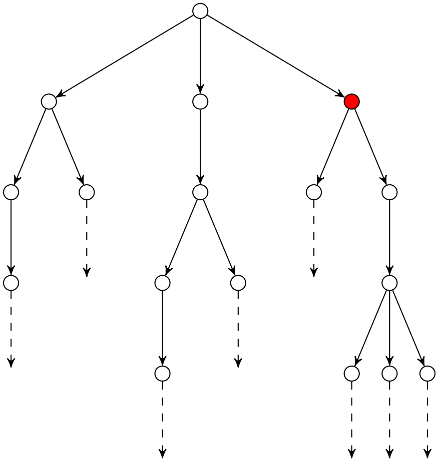


Illustration of the CTL semantics (2/8)

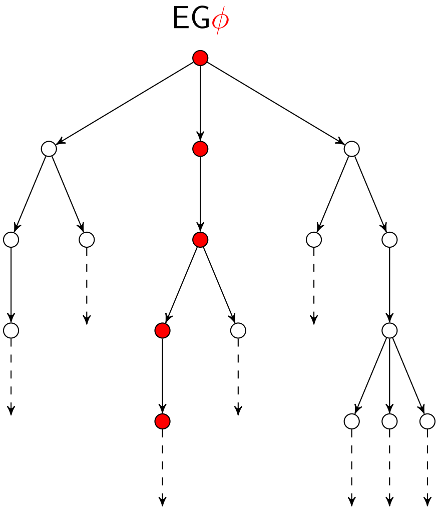


Illustration of the CTL semantics (3/8)

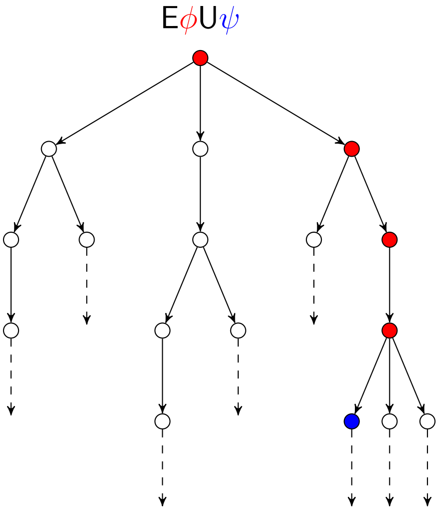


Illustration of the CTL semantics (4/8)

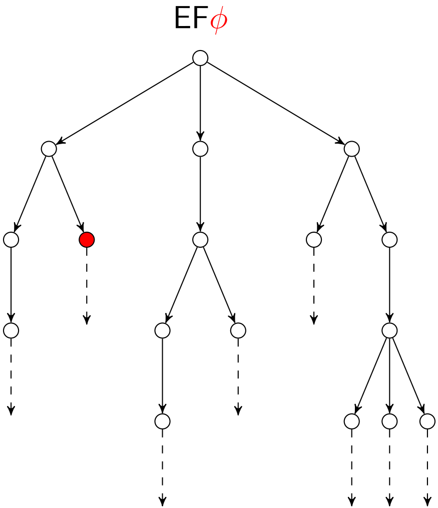


Illustration of the CTL semantics (5/8)

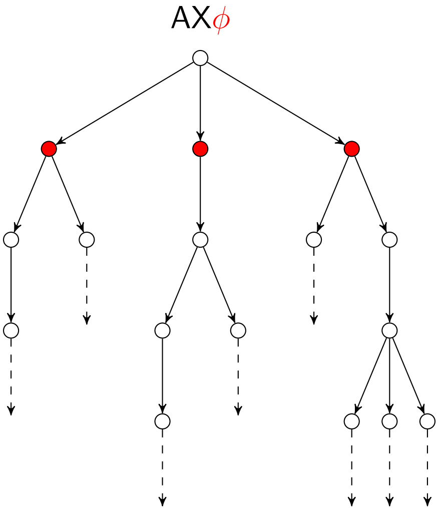


Illustration of the CTL semantics (6/8)

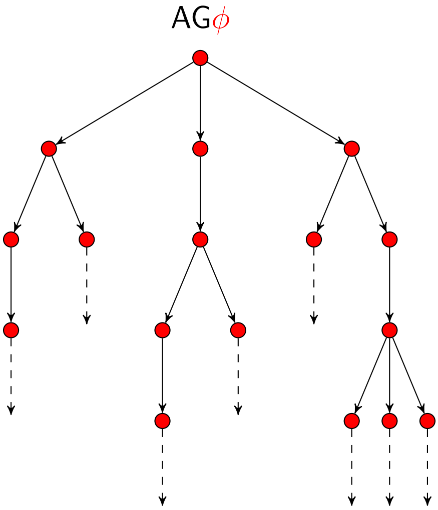


Illustration of the CTL semantics (7/8)

AFϕ

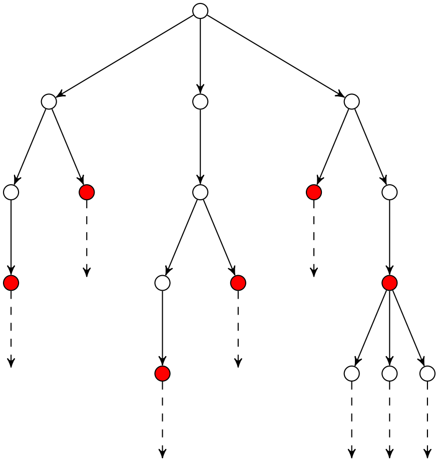
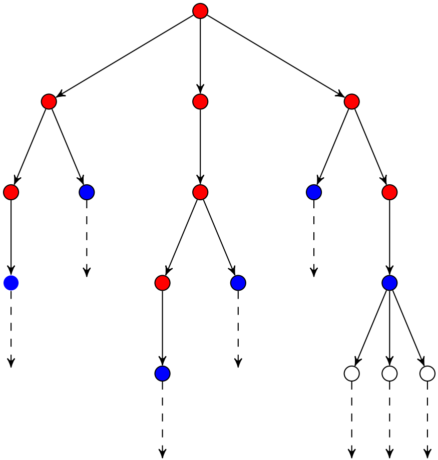


Illustration of the CTL semantics (8/8)

AϕUψ



Examples of CTL formulae

Explain the following CTL formulae, and if they are true or false:

Mutual exclusion between 2 processes (synchronised product)

1. $\mathrm{AG}\neg(cs_1 \wedge cs_2)$
2. $\mathrm{AG}(req_1 \implies \mathrm{AF}cs_1)$
3. $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

Examples of CTL formulae

Explain the following CTL formulae, and if they are true or false:

Mutual exclusion between 2 processes (synchronised product)

1. $\mathrm{AG}\neg(cs_1 \wedge cs_2)$
   Whatever happens, the two processes cannot be simultaneously in their critical section
   `true`
2. $\mathrm{AG}(req_1 \implies \mathrm{AF}cs_1)$
3. $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

Examples of CTL formulae

Explain the following CTL formulae, and if they are true or false:

Mutual exclusion between 2 processes (synchronised product)

1. $\mathrm{AG}\neg(cs_1 \wedge cs_2)$
   Whatever happens, the two processes cannot be simultaneously in their critical section
   `true`
2. $\mathrm{AG}(req_1 \implies \mathrm{AF}cs_1)$
   It is always the case that when process 1 requests access to its critical section, it will eventually be granted
   `false`
3. $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

Examples of CTL formulae

Explain the following CTL formulae, and if they are true or false:

Mutual exclusion between 2 processes (synchronised product)

1. $\mathrm{AG}\neg(cs_1 \wedge cs_2)$
   Whatever happens, the two processes cannot be simultaneously in their critical section
   `true`
2. $\mathrm{AG}(req_1 \implies \mathrm{AF}cs_1)$
   It is always the case that when process 1 requests access to its critical section, it will eventually be granted
   `false`
3. $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$
   Whatever the state of the system, it is possible to have both processes idle in the future.
   `true`

Exercises

Prove that:

1. $\mathsf{EF}\phi \equiv \mathsf{E}\,\mathtt{true}\,\mathsf{U}\phi$

2. $\mathsf{AX}\phi \equiv \neg\mathsf{EX}\neg\phi$

3. $\mathsf{AG}\phi \equiv \neg(\mathsf{E}\,\mathtt{true}\,\mathsf{U}\neg\phi)$

4. $\mathsf{AF}\phi \equiv \neg\mathsf{EG}\neg\phi$

Exercises

Prove that:

1. $\mathsf{EF}\phi \equiv \mathsf{E}\mathtt{true}\mathsf{U}\phi$
   We already proved that $\mathsf{F}\phi \equiv \mathtt{true}\mathsf{U}\phi$. Hence: $\mathsf{EF}\phi \equiv \mathsf{E}(\mathtt{true}\mathsf{U}\phi)$
2. $\mathsf{AX}\phi \equiv \neg\mathsf{EX}\neg\phi$
3. $\mathsf{AG}\phi \equiv \neg(\mathsf{E}\mathtt{true}\mathsf{U}\neg\phi)$
4. $\mathsf{AF}\phi \equiv \neg\mathsf{EG}\neg\phi$

Exercises

Prove that:

1. $\mathsf{EF}\phi \equiv \mathsf{E}\mathtt{true}\mathsf{U}\phi$

   We already proved that $\mathsf{F}\phi \equiv \mathtt{true}\mathsf{U}\phi$. Hence: $\mathsf{EF}\phi \equiv \mathsf{E}(\mathtt{true}\mathsf{U}\phi)$

2. $\mathsf{AX}\phi \equiv \neg\mathsf{EX}\neg\phi$

$$\begin{aligned}\neg\mathsf{EX}\neg\phi &\equiv \neg(\exists\sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \wedge \sigma', i \models \mathsf{X}\neg\phi) \\ &\equiv \forall\sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \text{ we have } \sigma', i \not\models \mathsf{X}\neg\phi \\ &\equiv \forall\sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \text{ we have } \sigma', i \models \mathsf{X}\phi \\ &\equiv \mathsf{AX}\phi\end{aligned}$$

3. $\mathsf{AG}\phi \equiv \neg(\mathsf{E}\mathtt{true}\mathsf{U}\neg\phi)$

4. $\mathsf{AF}\phi \equiv \neg\mathsf{EG}\neg\phi$

Exercises

Prove that:

1. $\mathsf{EF}\phi \equiv \mathsf{E}\mathtt{true}\mathsf{U}\phi$
   We already proved that $\mathsf{F}\phi \equiv \mathtt{true}\mathsf{U}\phi$. Hence: $\mathsf{EF}\phi \equiv \mathsf{E}(\mathtt{true}\mathsf{U}\phi)$
2. $\mathsf{AX}\phi \equiv \neg\mathsf{EX}\neg\phi$
   $$\begin{aligned} \neg\mathsf{EX}\neg\phi &\equiv \neg(\exists\sigma' : \sigma(0)\dots\sigma(i) = \sigma'(0)\dots\sigma'(i) \wedge \sigma', i \models \mathsf{X}\neg\phi) \\ &\equiv \forall\sigma' : \sigma(0)\dots\sigma(i) = \sigma'(0)\dots\sigma'(i) \text{ we have } \sigma', i \not\models \mathsf{X}\neg\phi \\ &\equiv \forall\sigma' : \sigma(0)\dots\sigma(i) = \sigma'(0)\dots\sigma'(i) \text{ we have } \sigma', i \models \mathsf{X}\phi \\ &\equiv \mathsf{AX}\phi \end{aligned}$$
3. $\mathsf{AG}\phi \equiv \neg(\mathsf{E}\mathtt{true}\mathsf{U}\neg\phi)$
   We know that $\mathsf{EF}\phi \equiv \mathsf{E}\mathtt{true}\mathsf{U}\phi$ and $\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$. Hence:
   $$\begin{aligned} \neg(\mathsf{E}\mathtt{true}\mathsf{U}\neg\phi) &\equiv \neg\mathsf{EF}\neg\phi \\ &\equiv \mathsf{A}\neg\mathsf{F}\neg\phi \\ &\equiv \mathsf{AG}\phi \end{aligned}$$
4. $\mathsf{AF}\phi \equiv \neg\mathsf{EG}\neg\phi$

Exercises

Prove that:

1. $\mathsf{EF}\phi \equiv \mathsf{E}\mathtt{true}\mathsf{U}\phi$

   We already proved that $\mathsf{F}\phi \equiv \mathtt{true}\mathsf{U}\phi$. Hence: $\mathsf{EF}\phi \equiv \mathsf{E}(\mathtt{true}\mathsf{U}\phi)$

2. $\mathsf{AX}\phi \equiv \neg\mathsf{EX}\neg\phi$

$$\begin{aligned}
\neg\mathsf{EX}\neg\phi &\equiv \neg(\exists\sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \wedge \sigma', i \models \mathsf{X}\neg\phi) \\
&\equiv \forall\sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \text{ we have } \sigma', i \not\models \mathsf{X}\neg\phi \\
&\equiv \forall\sigma' : \sigma(0)\ldots\sigma(i) = \sigma'(0)\ldots\sigma'(i) \text{ we have } \sigma', i \models \mathsf{X}\phi \\
&\equiv \mathsf{AX}\phi
\end{aligned}$$

3. $\mathsf{AG}\phi \equiv \neg(\mathsf{E}\mathtt{true}\mathsf{U}\neg\phi)$

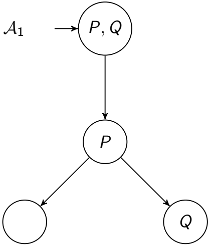
   We know that $\mathsf{EF}\phi \equiv \mathsf{E}\mathtt{true}\mathsf{U}\phi$ and $\mathsf{G}\phi \equiv \neg\mathsf{F}\neg\phi$. Hence:

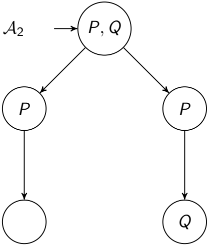
$$\begin{aligned}
\neg(\mathsf{E}\mathtt{true}\mathsf{U}\neg\phi) &\equiv \neg\mathsf{EF}\neg\phi \\
&\equiv \mathsf{A}\neg\mathsf{F}\neg\phi \\
&\equiv \mathsf{AG}\phi
\end{aligned}$$

4. $\mathsf{AF}\phi \equiv \neg\mathsf{EG}\neg\phi$

$$\begin{aligned}
\neg\mathsf{EG}\neg\phi &\equiv \mathsf{A}\neg\mathsf{G}\neg\phi \\
&\equiv \mathsf{AF}\phi
\end{aligned}$$

LTL and CTL do not recognise the same behaviours





LTL

Runs for both automata:

- $\{P, Q\}\ \{P\}\ \{-\}$
- $\{P, Q\}\ \{P\}\ \{Q\}$

$\forall \phi : \mathcal{A}_1 \models \phi \iff \mathcal{A}_2 \models \phi$

CTL

$\mathcal{A}_1 \models \mathrm{AX}(\mathrm{EX} Q \wedge \mathrm{EX} \neg Q)$
$\mathcal{A}_2 \not\models \mathrm{AX}(\mathrm{EX} Q \wedge \mathrm{EX} \neg Q)$

Outline

3. Model-checking
   - CTL model-checking
   - LTL model-checking

CTL model-checking algorithm

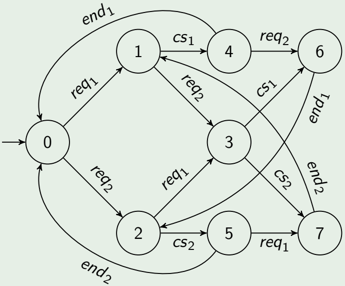
- algorithm marking states where a formula is satisfied
- memorises the already computed results
- reuses the computed results of sub-formulae to compute new formulae

CTL model-checking algorithm

Procedure marking(ϕ)

case $\phi = p$ **do**
  forall $q \in Q$ **do**
    if $p \in l(q)$ **then**
      `q.`ϕ`:=true`
    else
      `q.`ϕ`:=false`

$\phi = req_1$

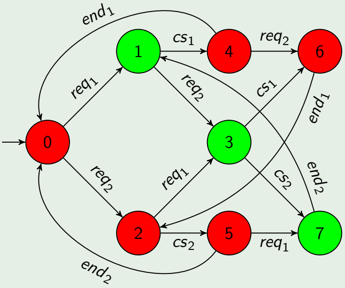


CTL model-checking algorithm

Procedure marking(ϕ)

case $\phi = p$ **do**
  forall $q \in Q$ **do**
    if $p \in l(q)$ **then**
      q.ϕ:=true
    else
      q.ϕ:=false

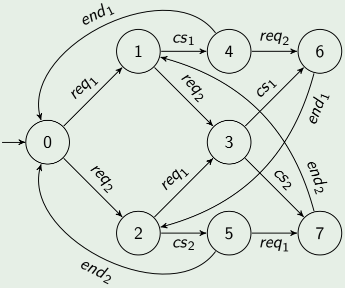
$\phi = req_1$



Case 2: $\phi = \neg\psi$

marking(ψ);
forall $q \in Q$ **do**
 q.ϕ:=¬q.ψ

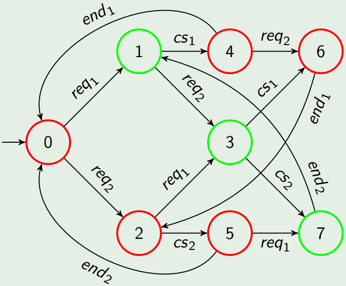
$\phi = \neg req_1$



Case 2: $\phi = \neg\psi$

marking(ψ);
forall $q \in Q$ **do**
  q.ϕ:=$\neg$q.ψ

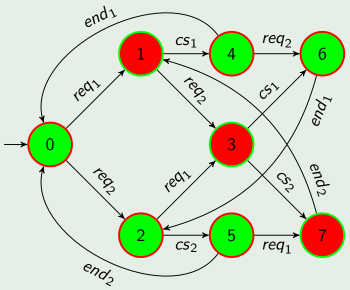
$\phi = \neg req_1$



Case 2: $\phi = \neg\psi$

marking(ψ);
forall $q \in Q$ **do**
  q.ϕ:=$\neg$q.ψ

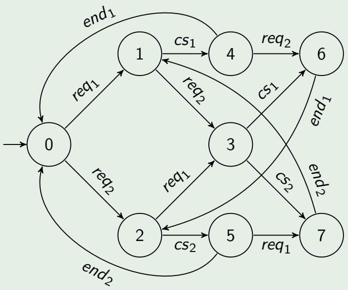
$\phi = \neg req_1$



Case 3: $\phi = \psi_1 \wedge \psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
  q.ϕ:=q.ψ_1∧q.ψ_2

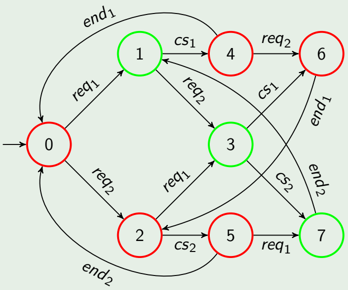
$\phi = req_1 \wedge req_2$



Case 3: $\phi = \psi_1 \land \psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
 q.ϕ:=q.$\psi_1 \land$q.ψ_2

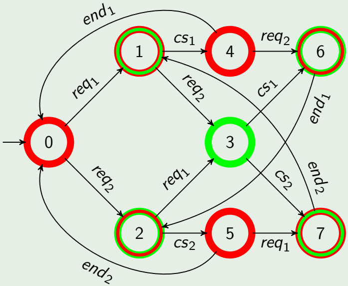
$\phi = req_1 \land req_2$



Case 3: $\phi = \psi_1 \wedge \psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
  q.ϕ:=q.$\psi_1 \wedge$q.ψ_2

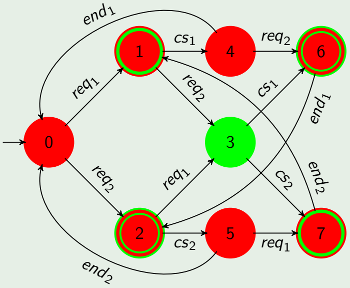
$\phi = req_1 \wedge req_2$



Case 3: $\phi = \psi_1 \wedge \psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
q.ϕ:=q.$\psi_1 \wedge$q.ψ_2

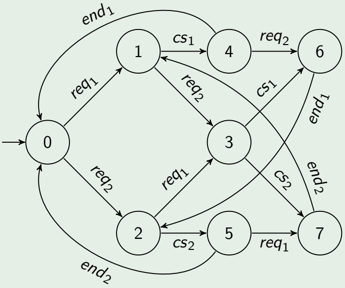
$\phi = req_1 \wedge req_2$



Case 4: $\phi = \mathrm{EX}\psi$

marking(ψ);
forall $q \in Q$ **do**
    q.ϕ:=`false`
forall $(q, \_, q') \in T$ **do**
    if q'.ψ=`true` **then**
        q.ϕ:=`true`

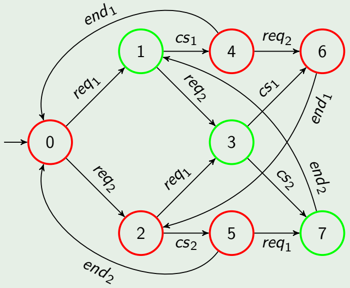
$\phi = \mathrm{EX}\mathit{req}_1$



Case 4: $\phi = \mathrm{EX}\psi$

marking(ψ);
forall $q \in Q$ **do**
  q.ϕ:=`false`
forall $(q, \_, q') \in T$ **do**
  if q'.ψ=`true` **then**
    q.ϕ:=`true`

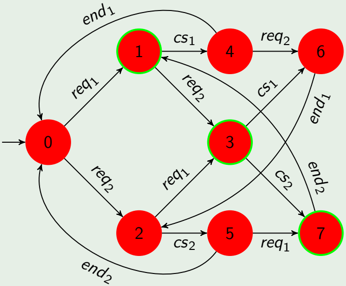
$\phi = \mathrm{EX}\mathit{req}_1$



Case 4: $\phi = \mathrm{EX}\psi$

marking(ψ);
forall $q \in Q$ **do**
  q.ϕ:=`false`
forall $(q, \_, q') \in T$ **do**
  if q'.ψ=`true` **then**
    q.ϕ:=`true`

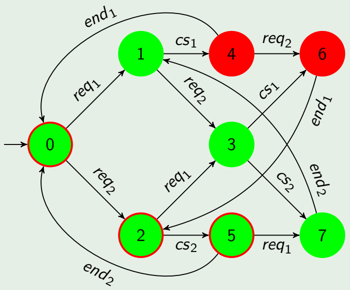
$\phi = \mathrm{EX}\mathit{req}_1$



Case 4: $\phi = \text{EX}\psi$

marking(ψ);
forall $q \in Q$ **do**
  q.ϕ:=`false`
forall $(q, \_, q') \in T$ **do**
  if q'.ψ=`true` **then**
    q.ϕ:=`true`

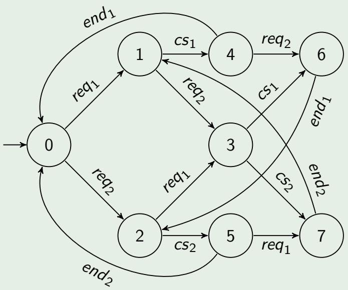
$\phi = \text{EX}\mathit{req}_1$



Case 5: $\phi = \mathsf{E}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
    q.ϕ:=`false`;
    q.seenbefore:=`false`
L:=$\emptyset$;
forall $q \in Q$ **do**
    if q.ψ_2=`true` **then**
        L:=L$\cup$\{q\}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L$\setminus$\{q\};
    q.ϕ:=`true`;
    forall $(q', \_, q) \in T$ **do**
        if q'.seenbefore=`false` **then**
            q'.seenbefore:=`true`;
            if q'.ψ_1=`true` **then**
                L:=L$\cup$\{q'\}

$\phi = \mathsf{E}req_1\mathsf{U}cs_1$

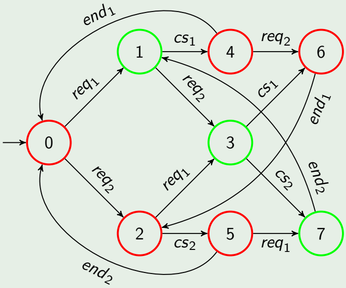


seenbefore	L

Case 5: $\phi = \mathsf{E}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
    q.ϕ:=`false`;
    q.seenbefore:=`false`
L:=$\emptyset$;
forall $q \in Q$ **do**
    if q.ψ_2=`true` **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L$\setminus${q};
    q.ϕ:=`true`;
    forall $(q', \_, q) \in T$ **do**
        if q'.seenbefore=`false` **then**
            q'.seenbefore:=`true`;
            if q'.ψ_1=`true` **then**
                L:=L$\cup${q'}

$\phi = \mathsf{E}req_1\mathsf{U}cs_1$

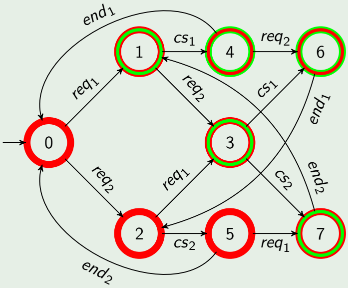


seenbefore	L

Case 5: $\phi = \mathsf{E}\psi_1 \mathsf{U} \psi_2$

```
marking(ψ1);
marking(ψ2);
forall q ∈ Q do
    q.φ:=false;
    q.seenbefore:=false
L:=∅;
forall q ∈ Q do
    if q.ψ2=true then
        L:=L∪{q}
while L≠ ∅ do
    pick q from L; L:=L\{q};
    q.φ:=true;
    forall (q', _, q) ∈ T do
        if q'.seenbefore=false then
            q'.seenbefore:=true;
            if q'.ψ1=true then
                L:=L∪{q'}
```

$\phi = \mathsf{E}\,req_1 \mathsf{U}\, cs_1$

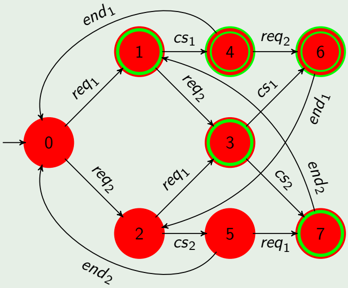


seenbefore	L

Case 5: $\phi = \mathsf{E}\psi_1 \mathsf{U} \psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
    q.ϕ:=`false`;
    q.seenbefore:=`false`
L:=$\emptyset$;
forall $q \in Q$ **do**
    if q.ψ_2=`true` **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L$\setminus${q};
    q.ϕ:=`true`;
    forall $(q', \_, q) \in T$ **do**
        if q'.seenbefore=`false` **then**
            q'.seenbefore:=`true`;
            if q'.ψ_1=`true` **then**
                L:=L$\cup${q'}

$\phi = \mathsf{E} req_1 \mathsf{U} cs_1$



seenbefore	L
$\emptyset$	

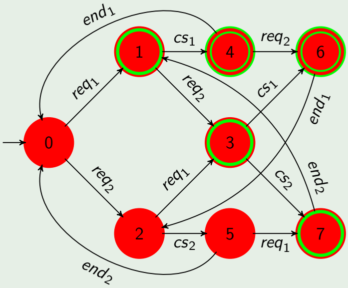
Case 5: $\phi = \mathsf{E}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
  q.ϕ:=`false`;
  q.seenbefore:=`false`

L:=$\emptyset$;
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup$\{q\}

while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus$\{q\};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    if q'.seenbefore=`false` **then**
      q'.seenbefore:=`true`;
      if q'.ψ_1=`true` **then**
        L:=L$\cup$\{q'\}

$\phi = \mathsf{E}req_1\mathsf{U}cs_1$

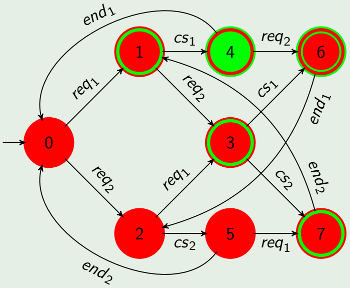


seenbefore	L
$\emptyset$	$\{4,6\}$

Case 5: $\phi = \mathsf{E}\psi_1 \mathsf{U} \psi_2$

```
marking(ψ1);
marking(ψ2);
forall q ∈ Q do
    q.φ:=false;
    q.seenbefore:=false
L:=∅;
forall q ∈ Q do
    if q.ψ2=true then
        L:=L∪{q}
while L≠ ∅ do
    pick q from L; L:=L\{q};
    q.φ:=true;
    forall (q′, _, q) ∈ T do
        if q'.seenbefore=false then
            q'.seenbefore:=true;
            if q'.ψ1=true then
                L:=L∪{q'}
```

$\phi = \mathsf{E}req_1 \mathsf{U} cs_1$

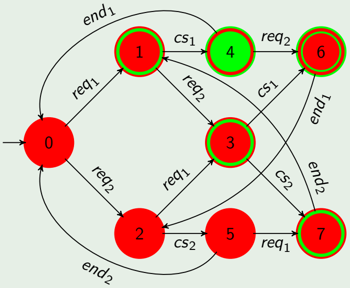


seenbefore	L
$\emptyset$	$\{6\}$

Case 5: $\phi = \mathrm{E}\psi_1 \mathrm{U} \psi_2$

```
marking(ψ1);
marking(ψ2);
forall q ∈ Q do
    q.φ:=false;
    q.seenbefore:=false
L:=∅;
forall q ∈ Q do
    if q.ψ2=true then
        L:=L∪{q}
while L≠ ∅ do
    pick q from L; L:=L\{q};
    q.φ:=true;
    forall (q', _, q) ∈ T do
        if q'.seenbefore=false then
            q'.seenbefore:=true;
            if q'.ψ1=true then
                L:=L∪{q'}
```

$\phi = \mathrm{E}req_1 \mathrm{U} cs_1$

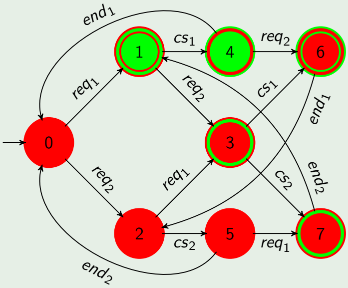


seenbefore	L
{1}	{1,6}

Case 5: $\phi = \mathsf{E}\psi_1\mathsf{U}\psi_2$

```
marking(ψ1);
marking(ψ2);
forall q ∈ Q do
    q.φ:=false;
    q.seenbefore:=false
L:=∅;
forall q ∈ Q do
    if q.ψ2=true then
        L:=L∪{q}
while L≠∅ do
    pick q from L; L:=L\{q};
    q.φ:=true;
    forall (q', _, q) ∈ T do
        if q'.seenbefore=false then
            q'.seenbefore:=true;
            if q'.ψ1=true then
                L:=L∪{q'}
```

$\phi = \mathsf{E}req_1\mathsf{U}cs_1$

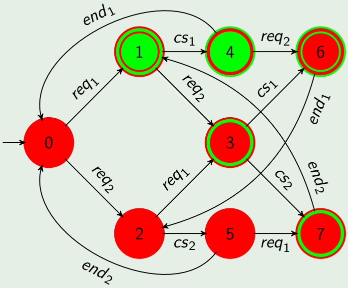


seenbefore	L
{1}	{6}

Case 5: $\phi = \mathsf{E}\psi_1 \mathsf{U} \psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
    q.ϕ:=false;
    q.seenbefore:=false
L:=$\emptyset$;
forall $q \in Q$ **do**
    if q.ψ_2=true **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L$\setminus${q};
    q.ϕ:=true;
    forall $(q', \_, q) \in T$ **do**
        if q'.seenbefore=false **then**
            q'.seenbefore:=true;
            if q'.ψ_1=true **then**
                L:=L$\cup${q'}

$\phi = \mathsf{E}req_1 \mathsf{U} cs_1$

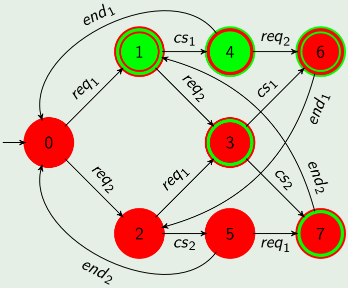


seenbefore	L
{0,1}	{6}

Case 5: $\phi = \mathrm{E}\psi_1 \mathrm{U} \psi_2$

```
marking(ψ1);
marking(ψ2);
forall q ∈ Q do
    q.ϕ:=false;
    q.seenbefore:=false
L:=∅;
forall q ∈ Q do
    if q.ψ2=true then
        L:=L∪{q}
while L≠ ∅ do
    pick q from L; L:=L\{q};
    q.ϕ:=true;
    forall (q', _, q) ∈ T do
        if q'.seenbefore=false then
            q'.seenbefore:=true;
            if q'.ψ1=true then
                L:=L∪{q'}
```

$\phi = \mathrm{E}req_1 \mathrm{U} cs_1$

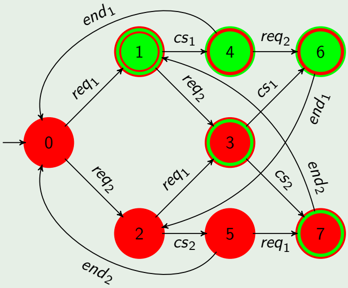


seenbefore	L
{0,1,7}	{6,7}

Case 5: $\phi = \mathsf{E}\psi_1 \mathsf{U} \psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
  q.ϕ:=`false`;
  q.seenbefore:=`false`
L:=$\emptyset$;
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup\{$q$\}$
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus\{$q$\}$;
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    if q'.seenbefore=`false` **then**
      q'.seenbefore:=`true`;
      if q'.ψ_1=`true` **then**
        L:=L$\cup\{$q'$\}$

$\phi = \mathsf{E}req_1 \mathsf{U} cs_1$

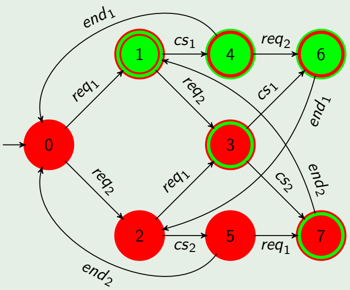


seenbefore	L
{0,1,7}	{7}

Case 5: $\phi = \mathsf{E}\psi_1 \mathsf{U} \psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
  q.ϕ:=`false`;
  q.seenbefore:=`false`
L:=$\emptyset$;
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup$\{q\}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L\\{q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    if q'.seenbefore=`false` **then**
      q'.seenbefore:=`true`;
      if q'.ψ_1=`true` **then**
        L:=L$\cup$\{q'\}

$\phi = \mathsf{E}req_1 \mathsf{U} cs_1$

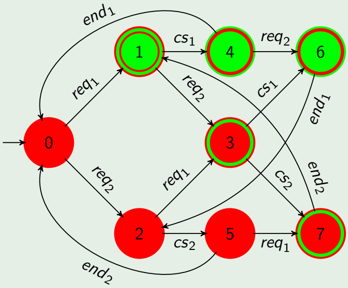


seenbefore	L
\{0,1,3,7\}	\{3,7\}

Case 5: $\phi = \mathrm{E}\psi_1 \mathrm{U} \psi_2$

```
marking(ψ1);
marking(ψ2);
forall q ∈ Q do
    q.ϕ:=false;
    q.seenbefore:=false
L:=∅;
forall q ∈ Q do
    if q.ψ2=true then
        L:=L∪{q}
while L≠ ∅ do
    pick q from L; L:=L\{q};
    q.ϕ:=true;
    forall (q′, _, q) ∈ T do
        if q'.seenbefore=false then
            q'.seenbefore:=true;
            if q'.ψ1=true then
                L:=L∪{q'}
```

$\phi = \mathrm{E}req_1 \mathrm{U} cs_1$

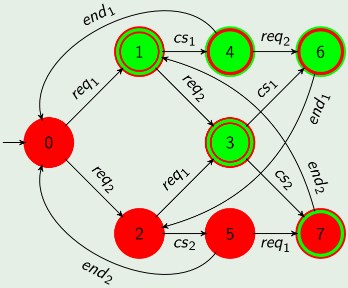


seenbefore	L
{0,1,3,4,7}	{3,7}

Case 5: $\phi = \mathsf{E}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
  q.ϕ:=`false`;
  q.seenbefore:=`false`
L:=$\emptyset$;
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup${q}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus${q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    if q'.seenbefore=`false` **then**
      q'.seenbefore:=`true`;
      if q'.ψ_1=`true` **then**
        L:=L$\cup${q'}

$\phi = \mathsf{E}req_1\mathsf{U}cs_1$

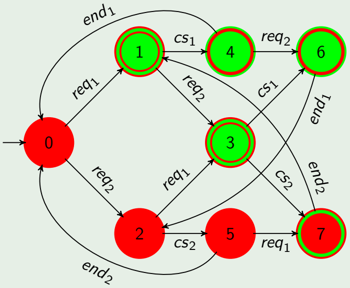


seenbefore	L
{0,1,3,4,7}	{7}

Case 5: $\phi = \mathsf{E}\psi_1 \mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
    q.ϕ:=`false`;
    q.seenbefore:=`false`
L:=$\emptyset$;
forall $q \in Q$ **do**
    if q.ψ_2=`true` **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L\{q};
    q.ϕ:=`true`;
    forall $(q', \_, q) \in T$ **do**
        if q'.seenbefore=`false` **then**
            q'.seenbefore:=`true`;
            if q'.ψ_1=`true` **then**
                L:=L$\cup${q'}

$\phi = \mathsf{E}req_1 \mathsf{U} cs_1$

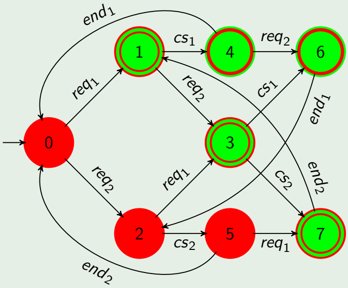


seenbefore	L
{0,1,2,3,4,7}	{7}

Case 5: $\phi = \mathsf{E}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
  q.ϕ:=`false`;
  q.seenbefore:=`false`
L:=$\emptyset$;
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup$\{q\}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L\\{q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    if q'.seenbefore=`false` **then**
      q'.seenbefore:=`true`;
      if q'.ψ_1=`true` **then**
        L:=L$\cup$\{q'\}

$\phi = \mathsf{E}\,req_1\,\mathsf{U}\,cs_1$

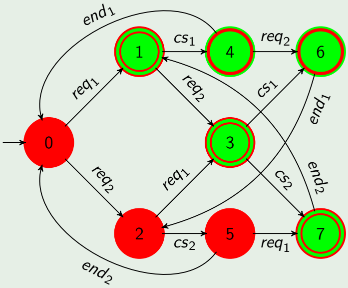


seenbefore	L
\{0,1,2,3,4,7\}	$\emptyset$

Case 5: $\phi = \mathrm{E}\psi_1 \mathrm{U} \psi_2$

marking(ψ_1);
marking(ψ_2);
forall $q \in Q$ **do**
  q.ϕ:=`false`;
  q.seenbefore:=`false`
L:=$\emptyset$;
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup$\{q\}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus$\{q\};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    if q'.seenbefore=`false` **then**
      q'.seenbefore:=`true`;
      if q'.ψ_1=`true` **then**
        L:=L$\cup$\{q'\}

$\phi = \mathrm{E}\mathit{req}_1 \mathrm{U} \mathit{cs}_1$

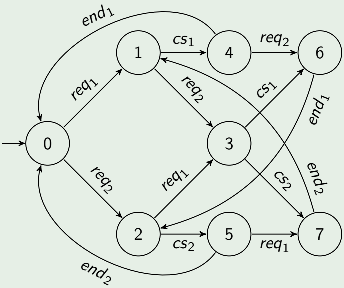


seenbefore	L
{0,1,2,3,4,5,7}	$\emptyset$

Case 6: $\phi = \mathsf{A}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
    q.nb:=degree(q);
    q.ϕ:=`false`
forall $q \in Q$ **do**
    if q.ψ_2=`true` **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L$\setminus${q};
    q.ϕ:=`true`;
    forall $(q', \_, q) \in T$ **do**
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.ψ_1=`true`
        and q'.ϕ=`false` **then**
            L:=L$\cup$ {q'}

$\phi = Areq_1Ucs_1$

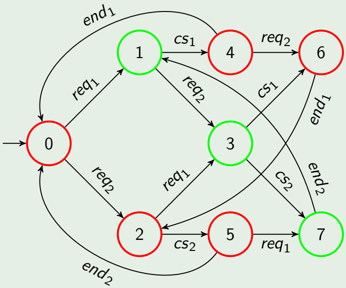


nb 0 1 2 3 4 5 6 7	L

Case 6: $\phi = A\psi_1 U \psi_2$

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
  q.nb:=degree(q);
  q.ϕ:=`false`
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup${q}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus${q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=`true`
    and q'.ϕ=`false` **then**
      L:=L$\cup$ {q'}

$\phi = A req_1 U cs_1$

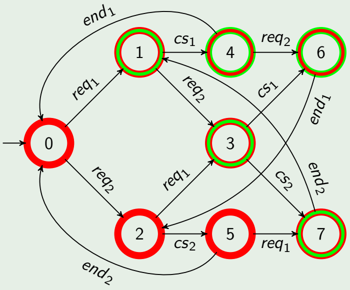


nb 0 1 2 3 4 5 6 7	L

Case 6: $\phi = A\psi_1 U \psi_2$

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
    q.nb:=degree(q);
    q.ϕ:=`false`
forall $q \in Q$ **do**
    if q.ψ_2=`true` **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L$\setminus${q};
    q.ϕ:=`true`;
    forall $(q', \_, q) \in T$ **do**
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.ψ_1=`true` and q'.ϕ=`false` **then**
            L:=L$\cup$ {q'}

$\phi = Areq_1 U cs_1$

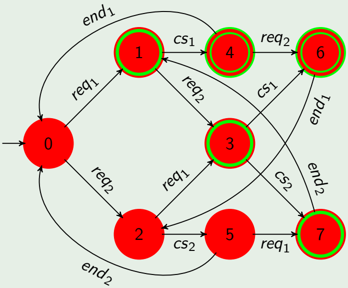


nb 0 1 2 3 4 5 6 7	L

Case 6: $\phi = \mathsf{A}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
  q.nb:=degree(q);
  q.ϕ:=`false`
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup${q}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus${q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=`true`
    and q'.ϕ=`false` **then**
      L:=L$\cup$ {q'}

$\phi = \mathsf{A}req_1\mathsf{U}cs_1$

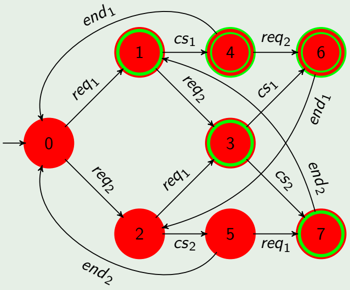


nb 0 1 2 3 4 5 6 7	L
2 2 2 2 2 2 1 1	$\emptyset$

Case 6: $\phi = A\psi_1 U\psi_2$

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
    q.nb:=degree(q);
    q.ϕ:=`false`
forall $q \in Q$ **do**
    if q.ψ_2=`true` **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L$\setminus${q};
    q.ϕ:=`true`;
    forall $(q', \_, q) \in T$ **do**
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.ψ_1=`true` and q'.ϕ=`false` **then**
            L:=L$\cup$ {q'}

$\phi = Areq_1 U cs_1$

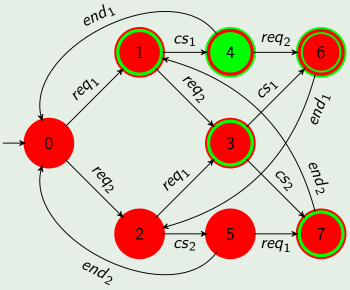


nb								L
0	1	2	3	4	5	6	7	
2	2	2	2	2	2	1	1	{4,6}

Case 6: $\phi = \mathsf{A}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
  q.nb:=degree(q);
  q.ϕ:=`false`
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup$\{q\}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus$\{q\};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=`true`
    and q'.ϕ=`false` **then**
      L:=L$\cup$ \{q'\}

$\phi = \mathsf{A}req_1\mathsf{U}cs_1$

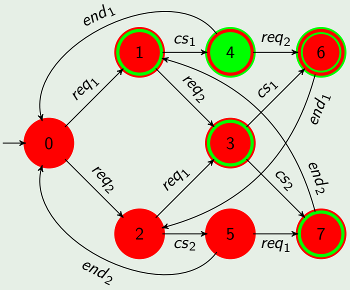


nb								L
0	1	2	3	4	5	6	7	
2	2	2	2	2	2	1	1	\{6\}

Case 6: $\phi = \mathsf{A}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
    q.nb:=degree(q);
    q.ϕ:=`false`
forall $q \in Q$ **do**
    if q.ψ_2=`true` **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L$\setminus${q};
    q.ϕ:=`true`;
    forall $(q', \_, q) \in T$ **do**
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.ψ_1=`true`
        and q'.ϕ=`false` **then**
            L:=L$\cup$ {q'}

$\phi = \mathsf{A}\mathit{req}_1\mathsf{U}\mathit{cs}_1$

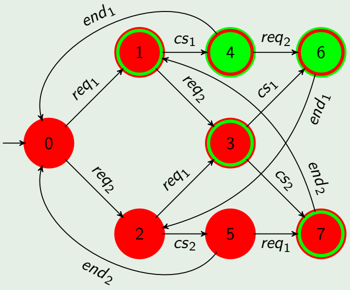


nb								L
0	1	2	3	4	5	6	7	
2	1	2	2	2	2	1	1	{6}

Case 6: $\phi = \mathsf{A}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
    q.nb:=degree(q);
    q.ϕ:=`false`
forall $q \in Q$ **do**
    if q.ψ_2=`true` **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L$\setminus${q};
    q.ϕ:=`true`;
    forall $(q', \_, q) \in T$ **do**
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.ψ_1=`true`
        and q'.ϕ=`false` **then**
            L:=L$\cup$ {q'}

$\phi = \mathsf{A}req_1\mathsf{U}cs_1$

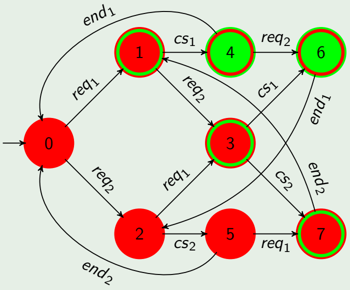


nb								L
0	1	2	3	4	5	6	7	
2	1	2	2	2	2	1	1	$\emptyset$

Case 6: $\phi = \mathsf{A}\psi_1\mathsf{U}\psi_2$

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
  q.nb:=degree(q);
  q.ϕ:=`false`
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup${q}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus${q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=`true`
    and q'.ϕ=`false` **then**
      L:=L$\cup$ {q'}

$\phi = \mathsf{A}\,req_1\,\mathsf{U}\,cs_1$

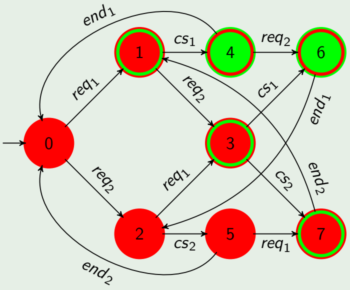


nb								L
0	1	2	3	4	5	6	7	
2	1	2	1	2	2	1	1	$\emptyset$

Case 6: $\phi = A\psi_1 U \psi_2$

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
  q.nb:=degree(q);
  q.ϕ:=`false`
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup${q}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\backslash${q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=`true`
    and q'.ϕ=`false` **then**
      L:=L$\cup$ {q'}

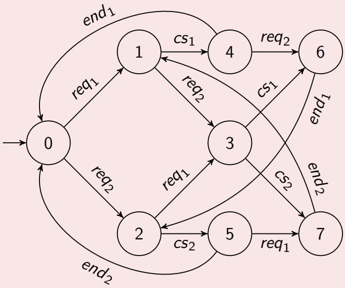
$\phi = Areq_1 U cs_1$



nb 0 1 2 3 4 5 6 7	L
2 1 2 1 1 2 1 1	$\emptyset$

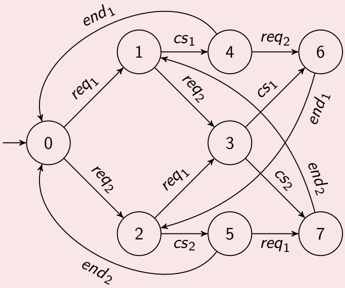
Exercises

Check $AG(EF(idle_1 \wedge idle_2))$



Exercises

Check $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

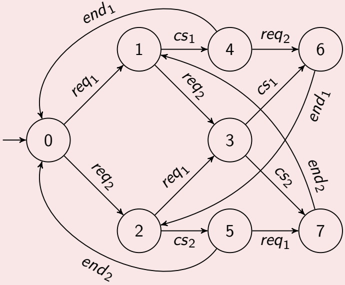


$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

- $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))))$

Exercises

Check $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

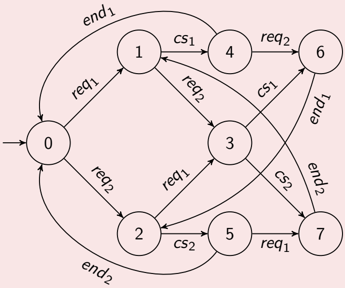


$$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$$

- $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5

Exercises

Check $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

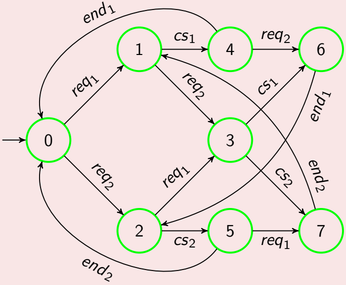


$$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$$

- $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0

Exercises

Check $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

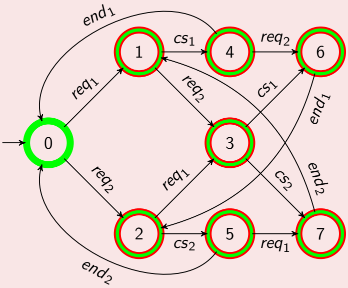


$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

- $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))$

Exercises

Check AG(EF($idle_1 \wedge idle_2$))

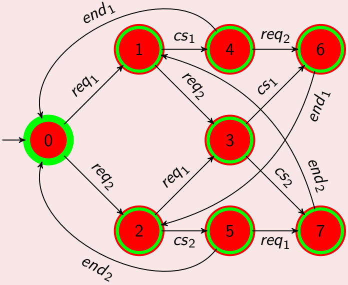


$AG(EF(idle_1 \wedge idle_2))$

- $\equiv \neg(E\mathtt{true}U\neg(EF(idle_1 \wedge idle_2)))$
  $\equiv \neg(E\mathtt{true}U\neg(E(\mathtt{true}U(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = E(\mathtt{true}U(idle_1 \wedge idle_2))$

Exercises

Check $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

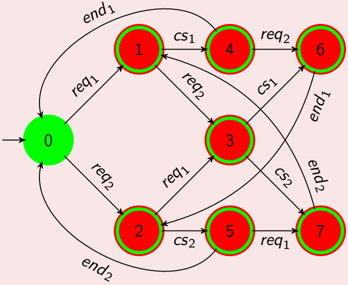


$$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$$

- $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))$

Exercises

Check AG(EF($idle_1 \wedge idle_2$))

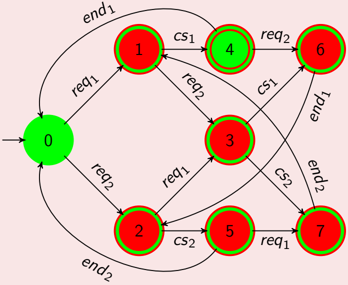


$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

- $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\,\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\,\neg(\mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))$

Exercises

Check AG(EF($idle_1 \wedge idle_2$))

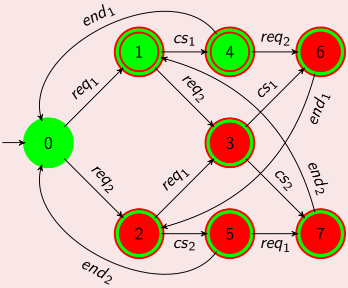


AG(EF($idle_1 \wedge idle_2$))

- $\equiv \neg(\mathtt{EtrueU}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathtt{EtrueU}\neg(\mathrm{E}(\mathtt{trueU}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{trueU}(idle_1 \wedge idle_2))$

Exercises

Check $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

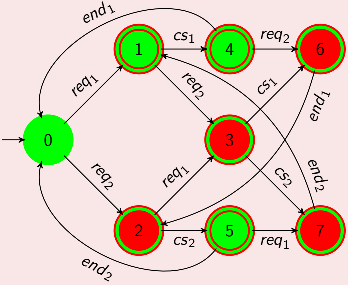


$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

- $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))$

Exercises

Check AG(EF($idle_1 \land idle_2$))

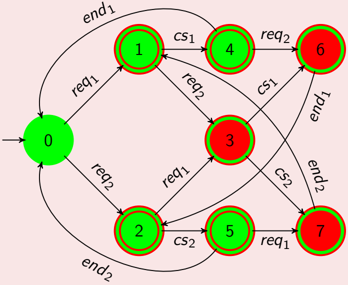


AG(EF($idle_1 \land idle_2$))

- $\equiv \neg(\mathtt{E}\,\mathtt{true}\,\mathtt{U} \neg(\mathrm{EF}(idle_1 \land idle_2)))$
  $\equiv \neg(\mathtt{E}\,\mathtt{true}\,\mathtt{U} \neg(\mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \land idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \land idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \land idle_2))$

Exercises

Check AG(EF($idle_1 \wedge idle_2$))

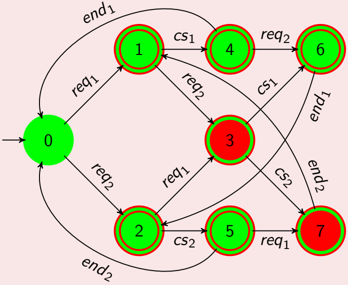


$AG(EF(idle_1 \wedge idle_2))$

- $\equiv \neg(\mathtt{E}\,\mathtt{true}\,\mathtt{U} \neg(EF(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathtt{E}\,\mathtt{true}\,\mathtt{U} \neg(E(\mathtt{true}\,\mathtt{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = E(\mathtt{true}\,\mathtt{U}(idle_1 \wedge idle_2))$

Exercises

Check $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

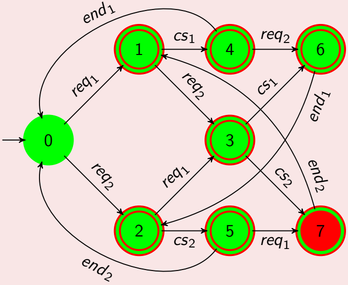


$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

- $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))$

Exercises

Check $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

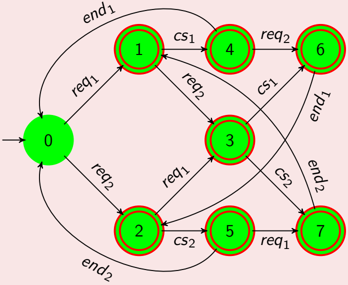


$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

- $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))$

Exercises

Check AG(EF($idle_1 \wedge idle_2$))

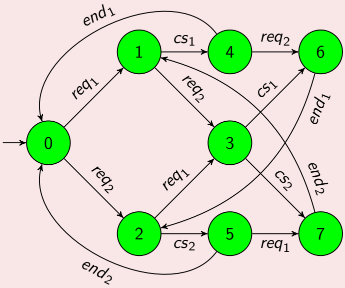


$\text{AG}(\text{EF}(idle_1 \wedge idle_2))$

- $\equiv \neg(\mathtt{E}\mathtt{true}\mathtt{U}\neg(\text{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathtt{E}\mathtt{true}\mathtt{U}\neg(\text{E}(\mathtt{true}\text{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \text{E}(\mathtt{true}\text{U}(idle_1 \wedge idle_2))$

Exercises

Check AG(EF($idle_1 \wedge idle_2$))

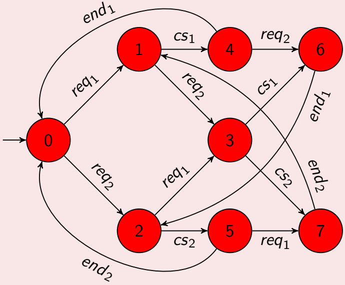


$$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$$

- $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))$
- case 2, mark $\phi_2 = \neg\phi_1$

Exercises

Check AG(EF($idle_1 \wedge idle_2$))

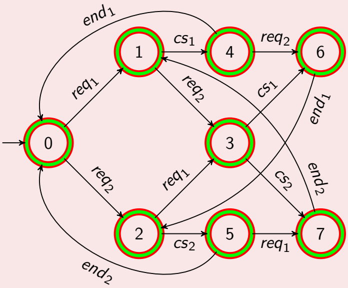


AG(EF($idle_1 \wedge idle_2$))

- $\equiv \neg(\mathtt{E}\,\mathtt{true}\,\mathtt{U}\,\neg(\mathtt{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathtt{E}\,\mathtt{true}\,\mathtt{U}\,\neg(\mathtt{E}(\mathtt{true}\,\mathtt{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathtt{E}(\mathtt{true}\,\mathtt{U}(idle_1 \wedge idle_2))$
- case 2, mark $\phi_2 = \neg\phi_1$

Exercises

Check $AG(EF(idle_1 \wedge idle_2))$

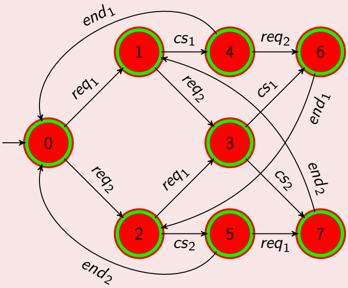


$AG(EF(idle_1 \wedge idle_2))$

- $\equiv \neg(\mathtt{E}\mathtt{true}\mathtt{U}\neg(EF(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathtt{E}\mathtt{true}\mathtt{U}\neg(E(\mathtt{true}U(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = E(\mathtt{true}U(idle_1 \wedge idle_2))$
- case 2, mark $\phi_2 = \neg\phi_1$
- case 5, mark $\phi_3 = E(\mathtt{true}U\phi_2)$

Exercises

Check AG(EF($idle_1 \wedge idle_2$))

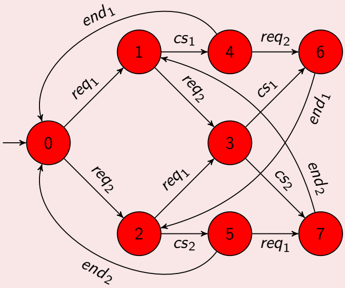


$$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$$

- $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\,\mathtt{true}\,\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\,\mathrm{U}(idle_1 \wedge idle_2))$
- case 2, mark $\phi_2 = \neg\phi_1$
- case 5, mark $\phi_3 = \mathrm{E}(\mathtt{true}\,\mathrm{U}\phi_2)$

Exercises

Check $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

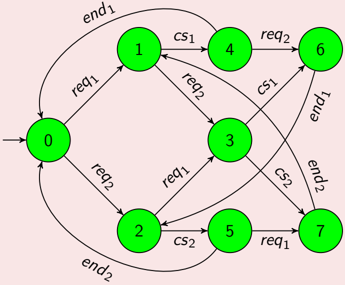


$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$

- $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))$
- case 2, mark $\phi_2 = \neg\phi_1$
- case 5, mark $\phi_3 = \mathrm{E}(\mathtt{true}\mathrm{U}\phi_2)$
- case 2, mark $\phi_4 = \neg\phi_3$

Exercises

Check $\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$



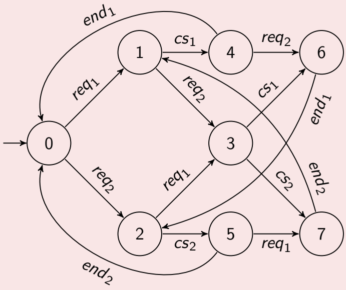
$$\mathrm{AG}(\mathrm{EF}(idle_1 \wedge idle_2))$$

- $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{EF}(idle_1 \wedge idle_2)))$
  $\equiv \neg(\mathrm{E}\mathtt{true}\mathrm{U}\neg(\mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))))$
- mark $idle_1$: states 0, 2, 5
- mark $idle_2$: states 0, 1, 4
- case 3, mark $idle_1 \wedge idle_2$: state 0
- case 5, mark $\phi_1 = \mathrm{E}(\mathtt{true}\mathrm{U}(idle_1 \wedge idle_2))$
- case 2, mark $\phi_2 = \neg\phi_1$
- case 5, mark $\phi_3 = \mathrm{E}(\mathtt{true}\mathrm{U}\phi_2)$
- case 2, mark $\phi_4 = \neg\phi_3$

Exercise

```
marking(ψ1);
marking(ψ2);
L:=∅;
forall q ∈ Q do
    q.nb:=degree(q);
    q.ϕ:=false
forall q ∈ Q do
    if q.ψ2=true then
        L:=L∪{q}
while L≠ ∅ do
    pick q from L; L:=L\{q};
    q.ϕ:=true;
    forall (q′, _, q) ∈ T do
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.ψ1=true
        and q'.ϕ=false then
            L:=L∪ {q'}
```

$\phi = A req_1 U cs_2$

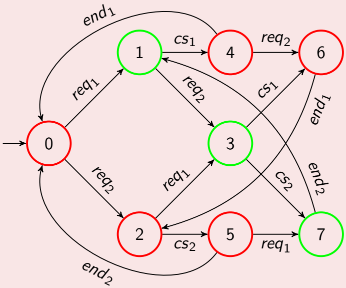


nb 0 1 2 3 4 5 6 7	L

Exercise

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
    q.nb:=degree(q);
    q.ϕ:=`false`
forall $q \in Q$ **do**
    if q.ψ_2=`true` **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L\{q};
    q.ϕ:=`true`;
    forall $(q', \_, q) \in T$ **do**
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.ψ_1=`true`
        and q'.ϕ=`false` **then**
            L:=L$\cup$ {q'}

$\phi = A req_1 U cs_2$

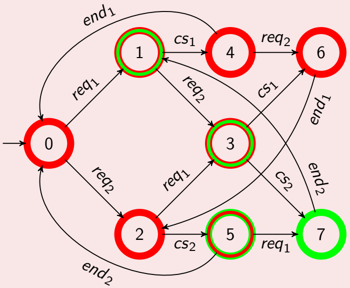


nb 0 1 2 3 4 5 6 7	L

Exercise

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
    q.nb:=degree(q);
    q.ϕ:=false
forall $q \in Q$ **do**
    if q.ψ_2=true **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L\{q};
    q.ϕ:=true;
    forall $(q', \_, q) \in T$ **do**
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.ψ_1=true
        and q'.ϕ=false **then**
            L:=L$\cup$ {q'}

$\phi = Areq_1Ucs_2$

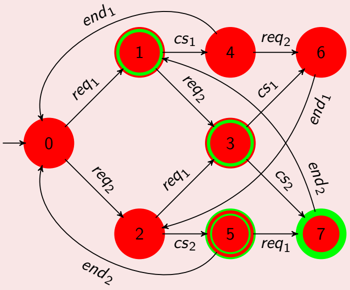


nb 0 1 2 3 4 5 6 7	L

Exercise

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
  q.nb:=degree(q);
  q.ϕ:=`false`
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup${q}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus${q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=`true` and q'.ϕ=`false` **then**
      L:=L$\cup$ {q'}

$\phi = A req_1 U cs_2$

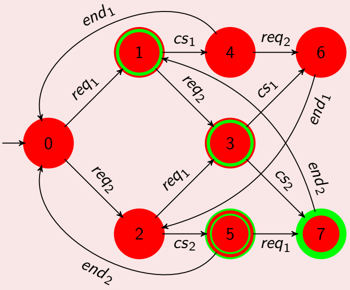


nb 0 1 2 3 4 5 6 7	L
2 2 2 2 2 2 1 1	$\emptyset$

Exercise

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
  q.nb:=degree(q);
  q.ϕ:=`false`
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup${q}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus${q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=`true`
    and q'.ϕ=`false` **then**
      L:=L$\cup$ {q'}

$\phi = \mathsf{A}req_1 \mathsf{U} cs_2$

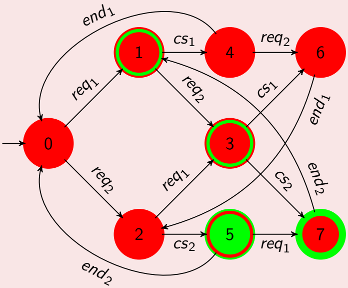


nb 0 1 2 3 4 5 6 7	L
2 2 2 2 2 2 1 1	{5,7}

Exercise

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
  q.nb:=degree(q);
  q.ϕ:=`false`
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup${q}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus${q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=`true`
    and q'.ϕ=`false` **then**
      L:=L$\cup$ {q'}

$\phi = A req_1 U cs_2$

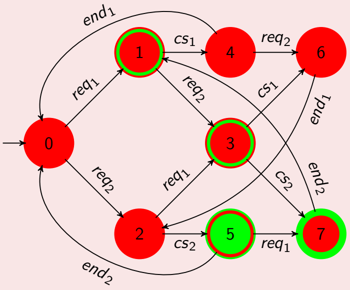


nb 0 1 2 3 4 5 6 7	L
2 2 2 2 2 2 1 1	{7}

Exercise

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
    q.nb:=degree(q);
    q.ϕ:=false
forall $q \in Q$ **do**
    if q.ψ_2=true **then**
        L:=L$\cup${q}
while $L \neq \emptyset$ **do**
    pick q from L; L:=L$\setminus${q};
    q.ϕ:=true;
    forall $(q', \_, q) \in T$ **do**
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.ψ_1=true
        and q'.ϕ=false **then**
            L:=L$\cup$ {q'}

$\phi = A req_1 U cs_2$

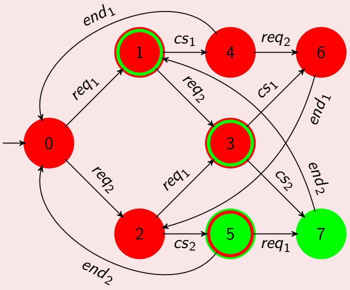


nb	L
0 1 2 3 4 5 6 7	
2 2 1 2 2 2 1 1	{7}

Exercise

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
  q.nb:=degree(q);
  q.ϕ:=`false`
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup${q}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus${q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=`true` and q'.ϕ=`false` **then**
      L:=L$\cup$ {q'}

$\phi = Areq_1Ucs_2$

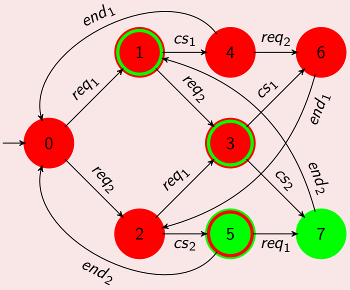


nb 0 1 2 3 4 5 6 7	L
2 2 1 2 2 2 1 1	$\emptyset$

Exercise

marking(ψ_1);
marking(ψ_2);
L:=$\emptyset$;
forall $q \in Q$ **do**
  q.nb:=degree(q);
  q.ϕ:=`false`
forall $q \in Q$ **do**
  if q.ψ_2=`true` **then**
    L:=L$\cup${q}
while $L \neq \emptyset$ **do**
  pick q from L; L:=L$\setminus${q};
  q.ϕ:=`true`;
  forall $(q', \_, q) \in T$ **do**
    q'.nb:=q'.nb - 1;
    if q'.nb=0 and q'.ψ_1=`true`
    and q'.ϕ=`false` **then**
      L:=L$\cup$ {q'}

$\phi = Areq_1 U cs_2$

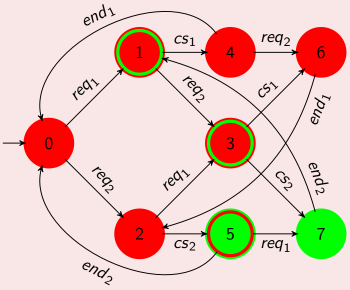


nb 0 1 2 3 4 5 6 7	L
2 2 1 2 2 1 1 1	$\emptyset$

Exercise

```
marking(ψ1);
marking(ψ2);
L:=∅;
forall q ∈ Q do
    q.nb:=degree(q);
    q.ϕ:=false
forall q ∈ Q do
    if q.ψ2=true then
        L:=L∪{q}
while L≠ ∅ do
    pick q from L; L:=L\{q};
    q.ϕ:=true;
    forall (q′, _, q) ∈ T do
        q'.nb:=q'.nb - 1;
        if q'.nb=0 and q'.ψ1=true
        and q'.ϕ=false then
            L:=L∪ {q'}
```

$\phi = \mathsf{A} req_1 \mathsf{U} cs_2$



nb 0 1 2 3 4 5 6 7	L
2 2 1 1 2 1 1 1	∅

LTL model-checking

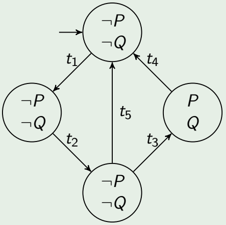
Algorithm working on path formulae

Principle for checking if $\mathcal{A} \models \phi$

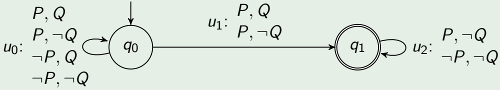
1. construct automaton $\mathcal{B}_{\neg\phi}$ recognising all executions not satisfying ϕ
2. construct the synchronised product $\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$
3. if its recognised language is empty, then $\mathcal{A} \models \phi$

Example

$\mathcal{A}$

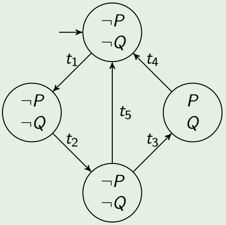


$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}(P \implies \mathsf{XF}Q)$

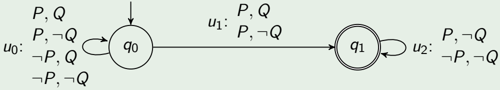


Example

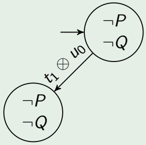
$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}(P \implies \mathsf{XF}Q)$

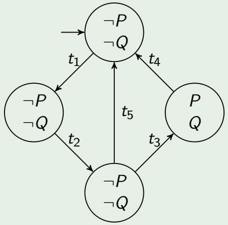


$\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$

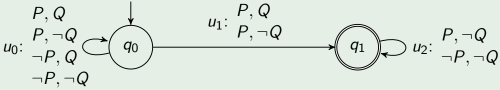


Example

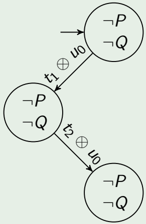
$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}(P \implies \mathsf{X}\mathsf{F}Q)$

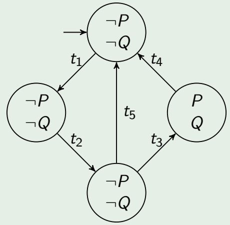


$\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$

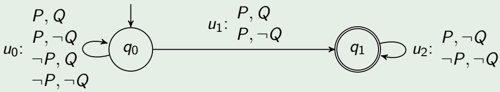


Example

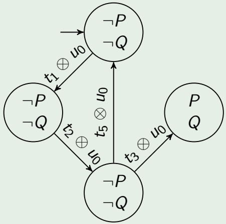
$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}(P \implies \mathsf{X}\mathsf{F}Q)$

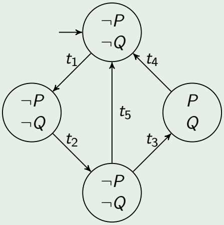


$\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$

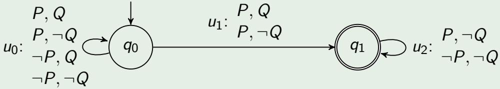


Example

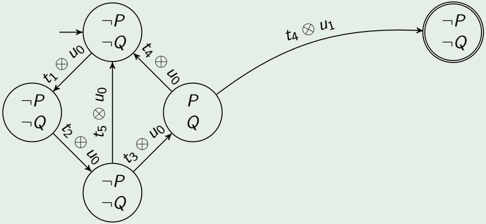
$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}(P \implies \mathsf{XF}Q)$

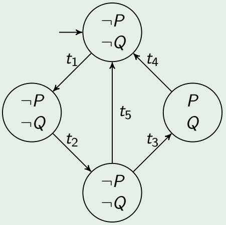


$\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$

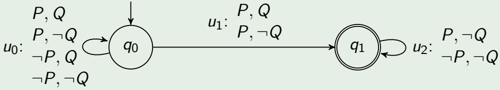


Example

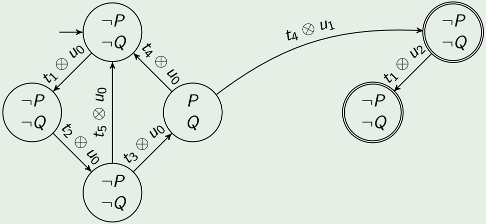
$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}(P \implies \mathsf{XF}Q)$

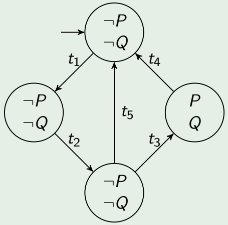


$\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$

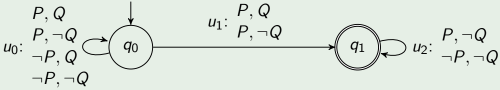


Example

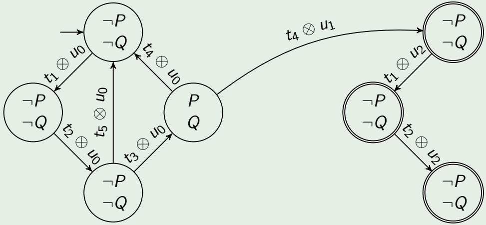
$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}(P \implies \mathsf{X}\mathsf{F}Q)$

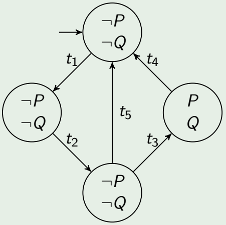


$\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$

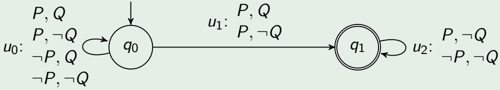


Example

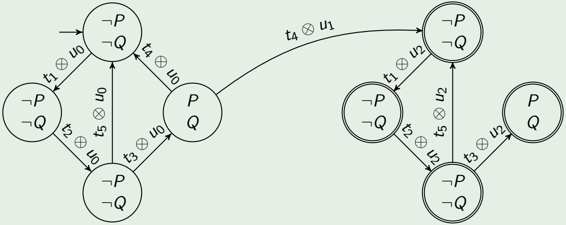
$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}(P \implies \mathsf{XF}Q)$

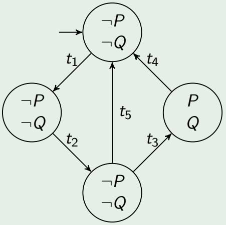


$\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$

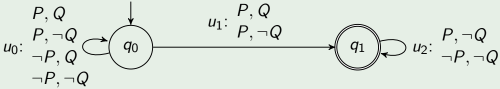


Example

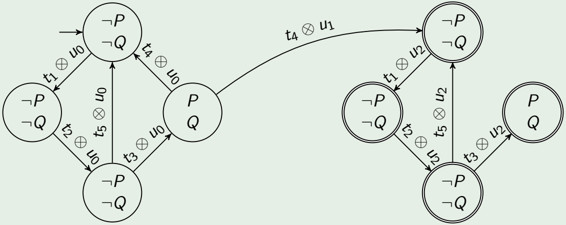
$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = G(P \implies XFQ)$

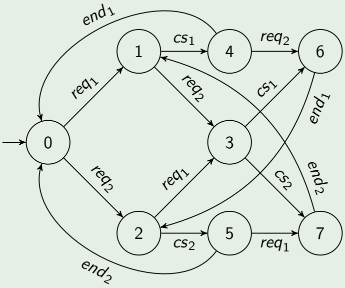


$\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$ $\mathcal{A} \not\models \phi$



Exercise

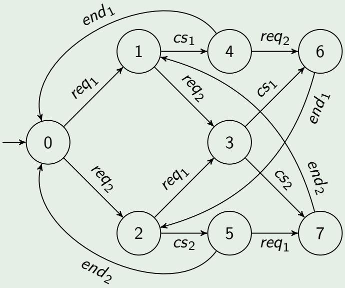
$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}\neg(cs_1 \wedge cs_2)$

Exercise

$\mathcal{A}$

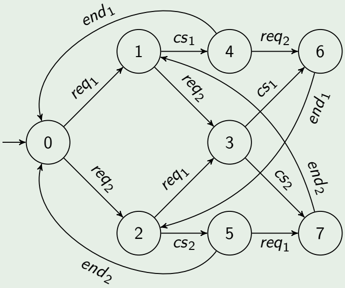


$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}\neg(cs_1 \wedge cs_2)$

$$
\begin{aligned}
\neg\phi &\equiv \neg\mathsf{G}\neg(cs_1 \wedge cs_2) \\
&\equiv \neg(\neg\mathsf{F}\neg(cs_1 \wedge cs_2)) \\
&\equiv \mathsf{F}(cs_1 \wedge cs_2) \\
&\equiv \mathtt{true}\mathsf{U}(cs_1 \wedge cs_2)
\end{aligned}
$$

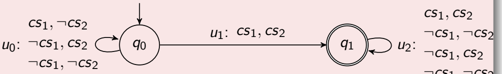
Exercise

$\mathcal{A}$



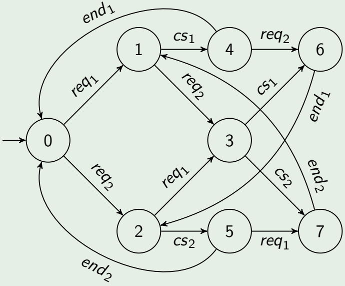
$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}\neg(cs_1 \wedge cs_2)$

$$\begin{aligned} \neg\phi &\equiv \neg\mathsf{G}\neg(cs_1 \wedge cs_2) \\ &\equiv \neg(\neg\mathsf{F}\neg\neg(cs_1 \wedge cs_2)) \\ &\equiv \mathsf{F}(cs_1 \wedge cs_2) \\ &\equiv \mathtt{true}\,\mathsf{U}(cs_1 \wedge cs_2) \end{aligned}$$



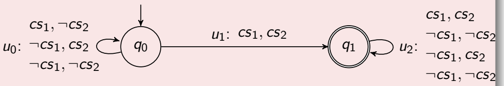
Exercise

$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}\neg(cs_1 \wedge cs_2)$

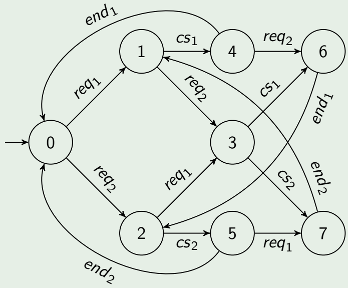
$$\begin{aligned}
\neg\phi &\equiv \neg\mathsf{G}\neg(cs_1 \wedge cs_2) \\
&\equiv \neg(\neg\mathsf{F}\neg(cs_1 \wedge cs_2)) \\
&\equiv \mathsf{F}(cs_1 \wedge cs_2) \\
&\equiv \mathtt{true}\mathsf{U}(cs_1 \wedge cs_2)
\end{aligned}$$



$\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$

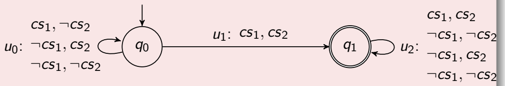
Exercise

$\mathcal{A}$



$\mathcal{B}_{\neg\phi}$ for $\phi = \mathsf{G}\neg(cs_1 \wedge cs_2)$

$$
\begin{aligned}
\neg\phi &\equiv \neg\mathsf{G}\neg(cs_1 \wedge cs_2)\\
&\equiv \neg(\neg\mathsf{F}\neg(cs_1 \wedge cs_2))\\
&\equiv \mathsf{F}(cs_1 \wedge cs_2)\\
&\equiv \mathtt{true}\,\mathsf{U}(cs_1 \wedge cs_2)
\end{aligned}
$$



$\mathcal{A} \otimes \mathcal{B}_{\neg\phi}$ $\qquad \mathcal{A} \models \phi$

All transitions of $\mathcal{A}$ synchronise with u_0.
So there is no accepting state and the formula is true.

Outline

4 Symbolic model-checking

- Computation of state sets
- Binary Decision Diagrams
- Automata representation

Motivation for symbolic approaches

- state space explosion problem
  - main obstacle with model-checking algorithms
  - because of the necessity to construct the state space
- represent symbolically states and transitions
- it aims at representing concisely large sets of states

Symbolic computation of state sets

Let $\mathcal{A} = \langle Q, E, T, q_0, l, F \rangle$ be an automaton, and $S \subseteq Q$ a set of its states. Let ϕ be a CTL formula.

Notations

- $Pre(S) = \{q \in Q \mid (q, \_, q') \in T \wedge q' \in S\}$ is the set of immediate predecessors of states in S
- $Sat(\phi) = \{q \in Q \mid q \models \phi\}$ is the set of states of the automaton satisfying formula ϕ
- $Pre^*(S)$ is the set of predecessors of states in S, whatever the number of steps

Computing $Sat(\phi)$

$$
\begin{aligned}
Sat(\neg\phi) &= Q \setminus Sat(\phi) \\
Sat(\psi_1 \wedge \psi_2) &= Sat(\psi_1) \cap Sat(\psi_2) \\
Sat(\mathrm{EX}\phi) &= Pre(Sat(\phi)) \\
Sat(\mathrm{AX}\phi) &= Q \setminus Pre(Q \setminus Sat(\phi)) \\
Sat(\mathrm{EF}\phi) &= Pre^*(Sat(\phi))
\end{aligned}
$$

Symbolic features

- symbolic representations of the state sets
- functions to manipulate these symbolic representations

Symbolic features

- symbolic representations of the state sets
- functions to manipulate these symbolic representations

Example

- suppose the automaton has 2 integer variables $a, b \in \{0, \dots, 255\}$
- each state is a triple (q, v_a, v_b) where v_a and v_b are values for a and b
- the set of reachable states can contain $|Q| \times 256 \times 256$ states (huge!)
- a possible symbolic representation could be $(q_2, 3, \_)$ for all states in q_2 with $a = 3$ and any value for b

Requirements for symbolic model-checking

1. symbolic representation of $Sat(p)$ for each proposition $p \in Prop$
2. algorithm to compute a symbolic representation of $Pre(S)$ from a symbolic representation of S
3. algorithms to compute the complement, union and intersection of symbolic representations of the sets
4. algorithm to compare symbolic representations of sets

Binary Decision Diagrams

- data structure commonly used for the symbolic representation of state sets
- Efficiency: cheap basic operations, compact data structure
- Simplicity: data structure and associated algorithms simple to describe and implement
- Easy adaptation: appropriate for problems dealing with loosely correlated data
- Generality: not tied to a particular family of automata

BDD structure

n boolean variables $x_1, \ldots, x_n$

- suppose $n = 4$. $\langle b_1, b_2, b_3, b_4 \rangle$ associates values with $x_1, \ldots, x_4$

BDD structure

n boolean variables $x_1, \dots, x_n$

- suppose $n = 4$. $\langle b_1, b_2, b_3, b_4 \rangle$ associates values with $x_1, \dots, x_4$
- Let us represent $S = \{\langle b_1, b_2, b_3, b_4 \rangle \mid (b_1 \vee b_3) \wedge (b_2 \implies b_4)\}$

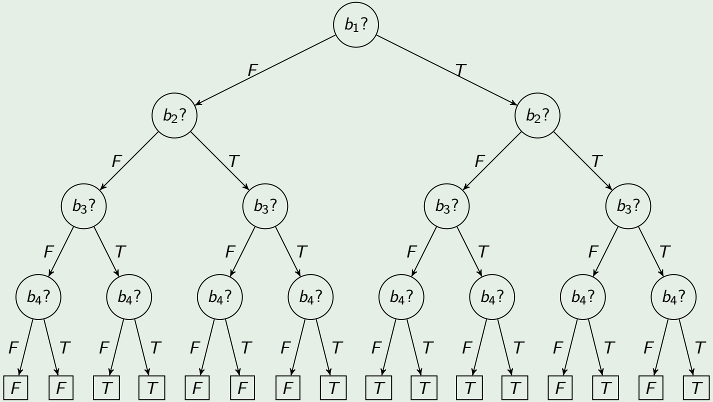
BDD structure

n boolean variables $x_1, \dots, x_n$

- suppose $n = 4$. $\langle b_1, b_2, b_3, b_4 \rangle$ associates values with $x_1, \dots, x_4$
- Let us represent $S = \{\langle b_1, b_2, b_3, b_4 \rangle \mid (b_1 \vee b_3) \wedge (b_2 \implies b_4)\}$
- Possible representations:
  - $|S| = 9$: $$S = \{ \langle F, F, T, F \rangle, \langle F, F, T, T \rangle, \langle F, T, T, T \rangle, \langle T, F, F, F \rangle, \langle T, F, F, T \rangle, \langle T, F, T, F \rangle, \langle T, F, T, T \rangle, \langle T, T, F, T \rangle, \langle T, T, T, T \rangle \}$$
  - the formula itself: $(b_1 \vee b_3) \wedge (b_2 \implies b_4)$
  - the formula in disjunctive normal form: $(b_1 \wedge \neg b_2) \vee (b_1 \wedge b_4) \vee (b_3 \wedge \neg b_2) \vee (b_3 \wedge b_4)$
  - a decision tree

Representation with a decision tree

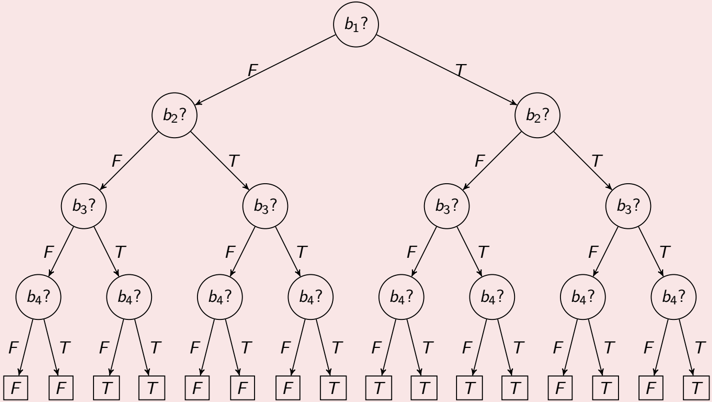
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

- identical subtrees are shared $\rightsquigarrow$ directed acyclic graph (*dag*)
- internal superfluous nodes are deleted (where no choice is possible)

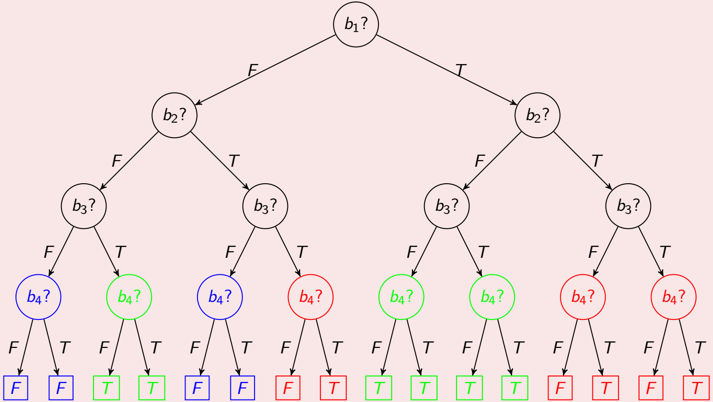
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

- identical subtrees are shared $\rightsquigarrow$ directed acyclic graph (*dag*)
- internal superfluous nodes are deleted (where no choice is possible)

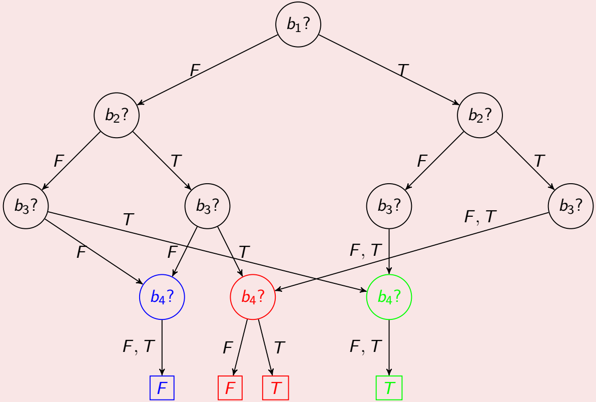
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

- identical subtrees are shared $\rightsquigarrow$ directed acyclic graph (*dag*)
- internal superfluous nodes are deleted (where no choice is possible)

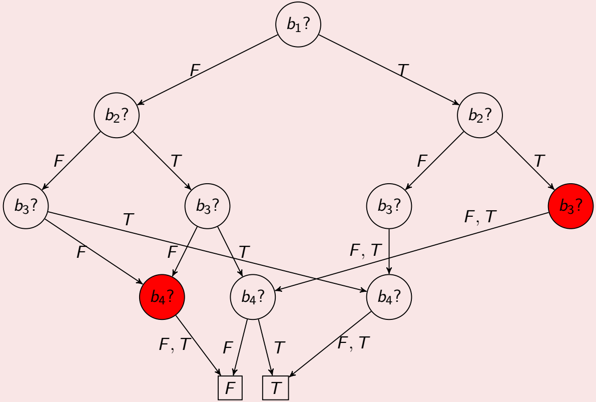
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

- identical subtrees are shared $\rightsquigarrow$ directed acyclic graph (*dag*)
- internal superfluous nodes are deleted (where no choice is possible)

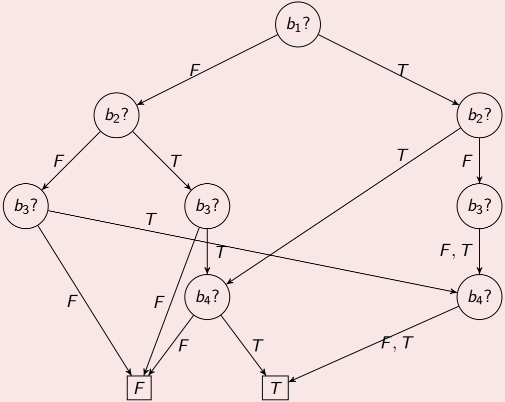
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

- identical subtrees are shared $\rightsquigarrow$ directed acyclic graph (*dag*)
- internal superfluous nodes are deleted (where no choice is possible)

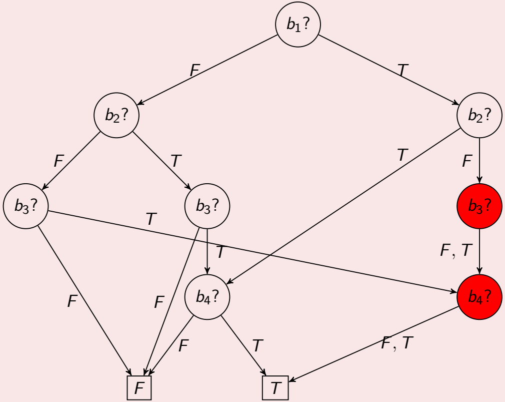
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

- identical subtrees are shared $\rightsquigarrow$ directed acyclic graph (*dag*)
- internal superfluous nodes are deleted (where no choice is possible)

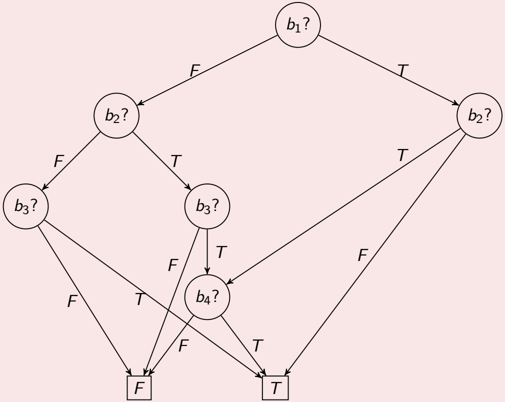
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

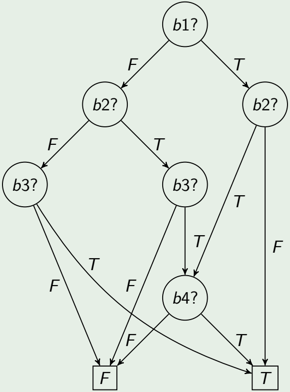
- identical subtrees are shared $\rightsquigarrow$ directed acyclic graph (*dag*)
- internal superfluous nodes are deleted (where no choice is possible)

$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



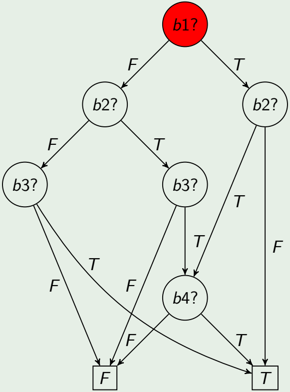
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F\rangle$, $\langle F, F, T, F\rangle$ in S?



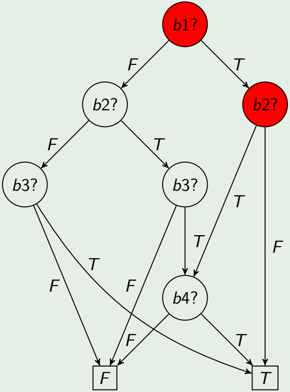
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F \rangle$, $\langle F, F, T, F \rangle$ in S?



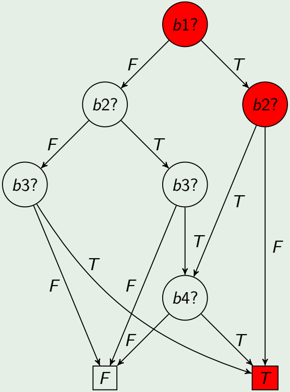
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F\rangle$, $\langle F, F, T, F\rangle$ in S?



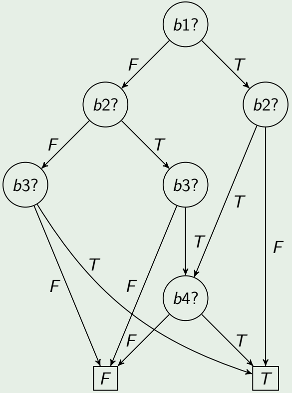
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F\rangle$, $\langle F, F, T, F\rangle$ in S?



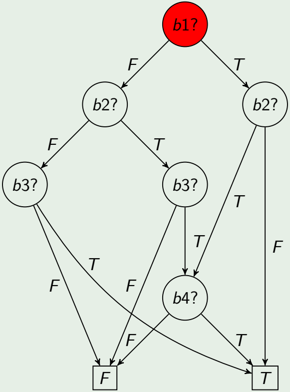
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F\rangle$, $\langle F, F, T, F\rangle$ in S?



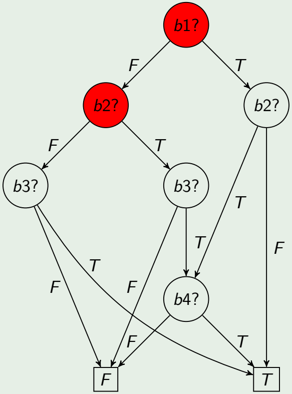
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F \rangle$, $\langle F, F, T, F \rangle$ in S?



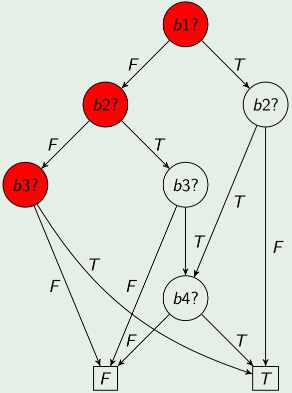
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F \rangle$, $\langle F, F, T, F \rangle$ in S?



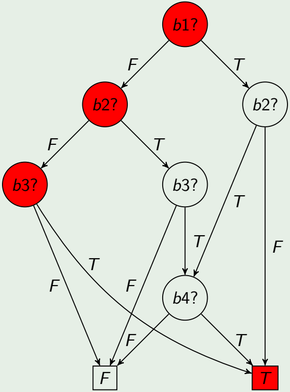
Testing whether a tuple belongs to the set

Are $\langle T, F, T, F\rangle$, $\langle F, F, T, F\rangle$ in S?



Testing whether a tuple belongs to the set

Are $\langle T, F, T, F \rangle$, $\langle F, F, T, F \rangle$ in S?

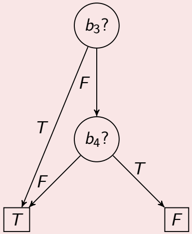


Exercise

BDD for $\neg((b_1 \wedge (b_2 \vee b_4) \wedge b_5) \vee \neg b_3) \vee (b_4 \implies (b_3 \wedge b_5))$

Exercise

BDD for $\neg((b_1 \wedge (b_2 \vee b_4) \wedge b_5) \vee \neg b_3) \vee (b_4 \implies (b_3 \wedge b_5))$

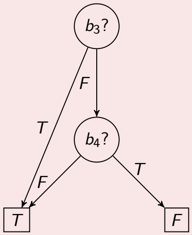


Exercise

BDD for $\neg((b_1 \wedge (b_2 \vee b_4) \wedge b_5) \vee \neg b_3) \vee (b_4 \implies (b_3 \wedge b_5))$ with ordering b_3, b_4, b_5, b_1, b_2

Exercise

BDD for $\neg((b_1 \wedge (b_2 \vee b_4) \wedge b_5) \vee \neg b_3) \vee (b_4 \implies (b_3 \wedge b_5))$ with ordering $b_3,\ b_4,\ b_5,\ b_1,\ b_2$



Advantages of BDDs

- small representations
- existence of a canonical BDD structure :
  - unicity for a fixed order of the variables
  - test the equivalence of two symbolic representations
  - test the emptyness
  - simple operations: complement, union, intersection, projection

Advantages of BDDs

- small representations
- existence of a canonical BDD structure :
  - unicity for a fixed order of the variables
  - test the equivalence of two symbolic representations

Identical canonical BDDs

  - test the emptyness

  - simple operations: complement, union, intersection, projection

Advantages of BDDs

- small representations
- existence of a canonical BDD structure :
  - unicity for a fixed order of the variables
  - test the equivalence of two symbolic representations

Identical canonical BDDs

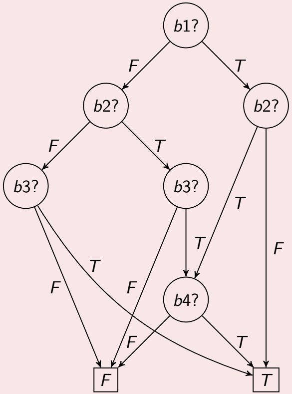
  - test the emptyness

Reduced to the F leaf

  - simple operations: complement, union, intersection, projection

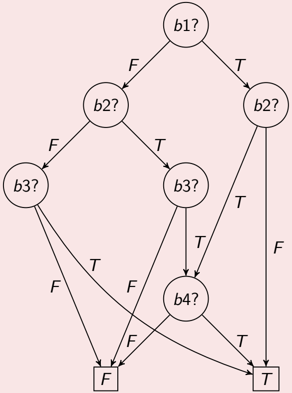
Exercise

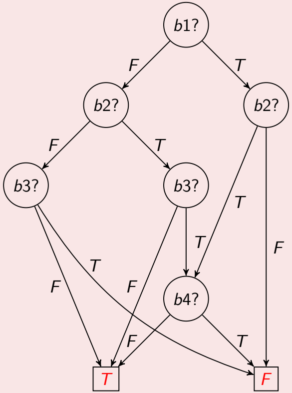
Complement



Exercise

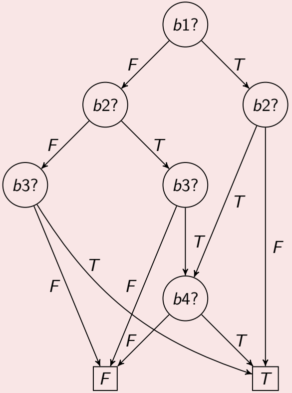
Complement

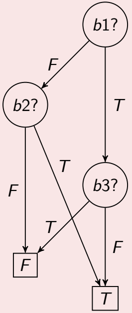




Exercise

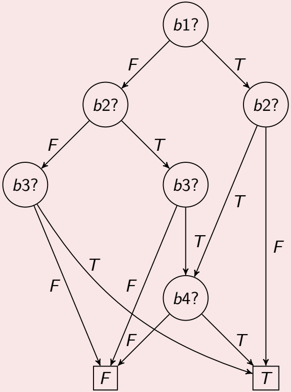
Union

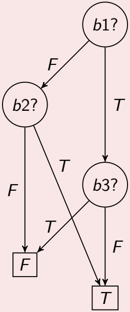


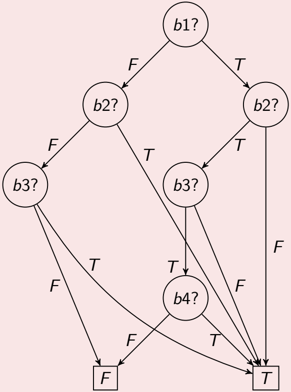


Exercise

Union

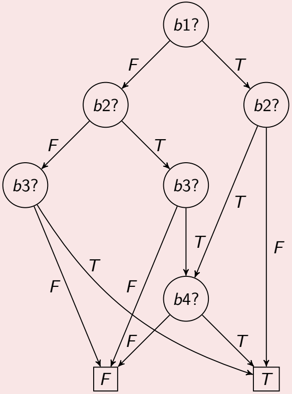


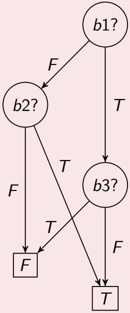




Exercise

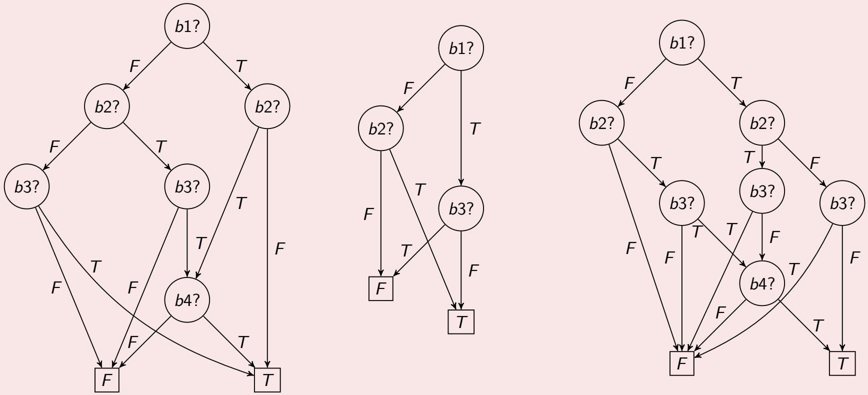
Intersection





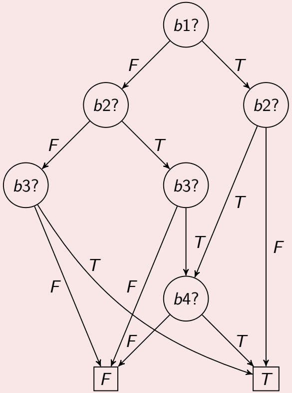
Exercise

Intersection



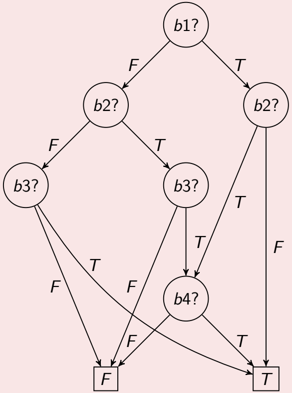
Exercise

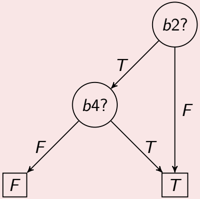
Projection $S[b_3 := T]$



Exercise

Projection $S[b_3 := T]$





Representing automata by BDDs

Encoding of states

- boolean encoding of states
- boolean encoding of each individual variable

Let us consider an automaton with:

- $Q = \{q_0, \dots, q_6\}$
- an integer variable *digit* $\in \{0, \dots, 9\}$
- a boolean variable *ready*

It can be encoded with 8 bits. For example, $\langle q_3, 8, F \rangle$ is represented by:

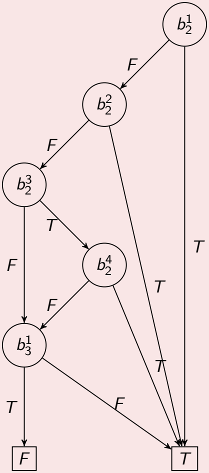
$$(\overbrace{\underset{b_1^1}{F}, \underset{b_1^2}{T}, \underset{b_1^3}{T}}^{q_3}, \overbrace{\underset{b_2^1}{T}, \underset{b_2^2}{F}, \underset{b_2^3}{F}, \underset{b_2^4}{F}}^{8}, \underset{b_3^1}{F})$$

Representing a set of states

$Sat(ready \implies (digit > 2))$

Representing a set of states

$Sat(ready \implies (digit > 2))$



Representing a transition

Transition seen as a pair of states

$\langle q_3, 8, F\rangle \longrightarrow \langle q_5, 0, F\rangle$ is represented by:

$$(\overbrace{\underset{b_1^1}{F}, \underset{b_1^2}{T}, \underset{b_1^3}{T}}^{q_3}, \overbrace{\underset{b_2^1}{T}, \underset{b_2^2}{F}, \underset{b_2^3}{F}, \underset{b_2^4}{F}}^{8}, \underset{b_3^1}{F}, \overbrace{\underset{b'^1_1}{T}, \underset{b'^2_1}{F}, \underset{b'^3_1}{T}}^{q_5}, \overbrace{\underset{b'^1_2}{F}, \underset{b'^2_2}{F}, \underset{b'^3_2}{F}, \underset{b'^4_2}{F}}^{0}, \underset{b'^1_3}{F})$$

Outline

5 Reachability Properties
- Reachability in temporal logic
- Computation of the reachability graph

Reachability properties

How to characterise reachability properties?

A reachability property states that some particular situation can be reached.

It may:

- be simple
- be conditional: restrict the form of paths reaching the state
- apply to any reachable state

Often, the negation of reachability is the interesting property.

Reachability properties

Examples

- we can obtain $n < 0$
- we can enter the critical section
- we cannot have $n < 0$
- we cannot reach the *crash* state
- we can enter the critical section without traversing $n = 0$
- we can always return to the initial state
- we can return to the initial state

Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section
- we cannot have $n < 0$
- we cannot reach the *crash* state
- we can enter the critical section without traversing $n = 0$
- we can always return to the initial state
- we can return to the initial state

Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$
- we cannot reach the *crash* state
- we can enter the critical section without traversing $n = 0$
- we can always return to the initial state
- we can return to the initial state

Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$ (negation)
- we cannot reach the *crash* state
- we can enter the critical section without traversing $n = 0$
- we can always return to the initial state
- we can return to the initial state

Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$ (negation)
- we cannot reach the *crash* state (negation)
- we can enter the critical section without traversing $n = 0$
- we can always return to the initial state
- we can return to the initial state

Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$ (negation)
- we cannot reach the *crash* state (negation)
- we can enter the critical section without traversing $n = 0$ (conditional)
- we can always return to the initial state
- we can return to the initial state

Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$ (negation)
- we cannot reach the *crash* state (negation)
- we can enter the critical section without traversing $n = 0$ (conditional)
- we can always return to the initial state (any reachable state)
- we can return to the initial state

Reachability properties

Examples

- we can obtain $n < 0$ (simple)
- we can enter the critical section (simple)
- we cannot have $n < 0$ (negation)
- we cannot reach the *crash* state (negation)
- we can enter the critical section without traversing $n = 0$ (conditional)
- we can always return to the initial state (any reachable state)
- we can return to the initial state (simple)

Reachability in temporal logic

Form of formulae in CTL

- use the EF combinator: $\mathrm{EF}\phi$
- ϕ is a propositional formula without temporal combinators
- E_U_ for conditional reachability
- nesting AG and EF when considering any reachable state

Reachability in temporal logic

Examples

- we can obtain $n < 0$:
- we can enter the critical section:
- we cannot have $n < 0$:
- we cannot reach the *crash* state:
- we can enter the critical section without traversing $n = 0$:
- we can always return to the initial state:
- we can return to the initial state:

Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\mathsf{EF}(n < 0)$
- we can enter the critical section:
- we cannot have $n < 0$:
- we cannot reach the *crash* state:
- we can enter the critical section without traversing $n = 0$:
- we can always return to the initial state:
- we can return to the initial state:

Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\mathrm{EF}(n < 0)$
- we can enter the critical section: $\mathrm{EF}\, cs$
- we cannot have $n < 0$:
- we cannot reach the *crash* state:
- we can enter the critical section without traversing $n = 0$:
- we can always return to the initial state:
- we can return to the initial state:

Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\mathrm{EF}(n < 0)$
- we can enter the critical section: $\mathrm{EF}cs$
- we cannot have $n < 0$: $\neg\mathrm{EF}(n < 0) \equiv \mathrm{AG}(n \geq 0)$
- we cannot reach the *crash* state:
- we can enter the critical section without traversing $n = 0$:
- we can always return to the initial state:
- we can return to the initial state:

Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\mathrm{EF}(n < 0)$
- we can enter the critical section: $\mathrm{EF}\,cs$
- we cannot have $n < 0$: $\neg\mathrm{EF}(n < 0) \equiv \mathrm{AG}(n \geq 0)$
- we cannot reach the *crash* state: $\neg\mathrm{EF}\,crash \equiv \mathrm{AG}\neg crash$
- we can enter the critical section without traversing $n = 0$:
- we can always return to the initial state:
- we can return to the initial state:

Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\mathrm{EF}(n < 0)$
- we can enter the critical section: $\mathrm{EF}\,cs$
- we cannot have $n < 0$: $\neg\mathrm{EF}(n < 0) \equiv \mathrm{AG}(n \geq 0)$
- we cannot reach the *crash* state: $\neg\mathrm{EF}\,crash \equiv \mathrm{AG}\neg crash$
- we can enter the critical section without traversing $n = 0$: $\mathrm{E}(n \neq 0)\mathrm{U}\,cs$
- we can always return to the initial state:
- we can return to the initial state:

Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\mathrm{EF}(n < 0)$
- we can enter the critical section: $\mathrm{EF}\mathit{cs}$
- we cannot have $n < 0$: $\neg\mathrm{EF}(n < 0) \equiv \mathrm{AG}(n \geq 0)$
- we cannot reach the *crash* state: $\neg\mathrm{EF}\mathit{crash} \equiv \mathrm{AG}\neg\mathit{crash}$
- we can enter the critical section without traversing $n = 0$: $\mathrm{E}(n \neq 0)\mathrm{U}\mathit{cs}$
- we can always return to the initial state: $\mathrm{AGEF}\mathit{init}$
- we can return to the initial state:

Reachability in temporal logic

Examples

- we can obtain $n < 0$: $\mathrm{EF}(n < 0)$
- we can enter the critical section: $\mathrm{EF}\mathit{cs}$
- we cannot have $n < 0$: $\neg\mathrm{EF}(n < 0) \equiv \mathrm{AG}(n \geq 0)$
- we cannot reach the *crash* state: $\neg\mathrm{EF}\mathit{crash} \equiv \mathrm{AG}\neg\mathit{crash}$
- we can enter the critical section without traversing $n = 0$: $\mathrm{E}(n \neq 0)\mathrm{U}\mathit{cs}$
- we can always return to the initial state: $\mathrm{AGEF}\mathit{init}$
- we can return to the initial state: $\mathrm{EF}\mathit{init}$

Computation of the reachability graph

Forward chaining

- start from the initial state
- add its successors
- continue until saturation

Drawback: potential explosion of the set being constructed

Computation of the reachability graph

Backward chaining

Construct the set of states which can lead to some target states

- start from target states
- add their immediate predecessors
- continue until saturation
- test whether some initial state is in the computed set

Drawbacks:

- identify target states
- computing predecessors can be more difficult than computing successors (e.g. for automata with variables)
- target states may be unreachable

Computation of the reachability graph

On-the-fly exploration

- check the property during exploration
- only partially construct the state space