M2 PLS

2024-2025

Complex systems
Part 1: Model checking

Étienne André

Université Sorbonne Paris Nord
Etienne.Andre@univ-paris13.fr

Version slides with holes: September 3, 2024

Objectives of the module

- introduce formal models for critical systems specification
  1. in an untimed setting
  2. in a timed setting
  3. in a parametric timed setting
- use model checking to verify their properties
  - properties expressed in extensions of the LTL and CTL logics

Objectives of this part of the module

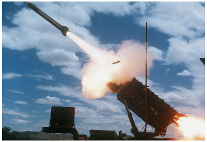
- introduce formal models for critical systems specification
  - finite-state automata
  - their extensions
- use model checking to verify their properties
  - reachability
  - properties expressed in LTL and CTL logics
- mention symbolic representations

Context: Verifying complex timed systems

- Critical systems: Failures may result in dramatic consequences
- Need for early bug detection
  - Bugs discovered when final testing: expensive

  ⇝ Need for a thorough specification and verification phase

Northeast blackout
(USA, 2003)



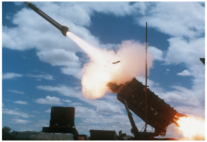
MIM-104 Patriot Missile Failure
(Iraq, 1991)

Sleipner A offshore platform
(Norway, 1991)

Context: Verifying complex timed systems

- Critical systems: Failures may result in dramatic consequences
- Need for early bug detection
  - Bugs discovered when final testing: expensive
  - $\rightsquigarrow$ Need for a thorough specification and verification phase

Northeast blackout
(USA, 2003)



MIM-104 Patriot Missile Failure
(Iraq, 1991)

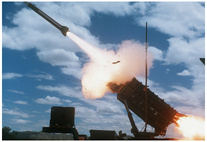
Sleipner A offshore platform
(Norway, 1991)

- Verification is needed to ensure the absence of bugs
- Verification techniques
  - Testing
  - Abstract interpretation
  - Theorem proving
  - Model checking

Context: Verifying complex timed systems

- Critical systems: Failures may result in dramatic consequences
- Need for early bug detection
  - Bugs discovered when final testing: expensive
  - ⇝ Need for a thorough specification and verification phase

Northeast blackout
(USA, 2003)



MIM-104 Patriot Missile Failure
(Iraq, 1991)

Sleipner A offshore platform
(Norway, 1991)

- Verification is needed to ensure the absence of bugs
- Verification techniques
  - Testing
  - Abstract interpretation
  - Theorem proving
  - Model checking

The Therac-25 radiation therapy machine (1/2)

- Radiation therapy machine used in the 1980s
- Involved in accidents between 1985 and 1987, in which patients were given massive overdoses of radiation
  - Approximately 100 times the intended dose!
  - Numerous causes, including race condition

The Therac-25 radiation therapy machine (1/2)

- Radiation therapy machine used in the 1980s
- Involved in accidents between 1985 and 1987, in which patients were given massive overdoses of radiation
  - Approximately 100 times the intended dose!
  - Numerous causes, including race condition

"The failure only occurred when a particular nonstandard sequence of keystrokes was entered on the VT-100 terminal which controlled the PDP-11 computer: an `X` *to (erroneously) select 25MV photon mode followed by* ↑, `E` *to (correctly) select 25 MeV Electron mode, then* `Enter`*, all within eight seconds."*

The Therac-25 radiation therapy machine (2/2)

The testing engineers could obviously not detect this strange (and quick!) sequence leading to the failure.

The Therac-25 radiation therapy machine (2/2)

The testing engineers could obviously not detect this strange (and quick!) sequence leading to the failure.

Limits of testing

This case illustrates the difficulty of bug detection without formal methods.

Bugs can be difficult to find

…and can have dramatic consequences for critical systems:

- health-related devices
- aeronautics and aerospace transportation
- smart homes and smart cities
- military devices
- etc.

Bugs can be difficult to find

…and can have dramatic consequences for critical systems:

- health-related devices
- aeronautics and aerospace transportation
- smart homes and smart cities
- military devices
- etc.

Hence, high need for formal verification

Outline

Outline

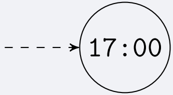
Automata

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.

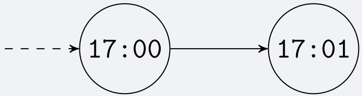
Example (Digital clock)



Automata

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

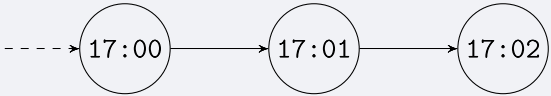
Example (Digital clock)



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)

Example: The modulo 3 counter

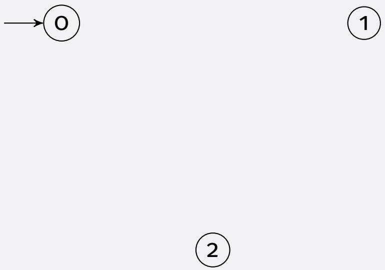
Example (Modulo 3 counter)

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

Example: The modulo 3 counter

Example (Modulo 3 counter)

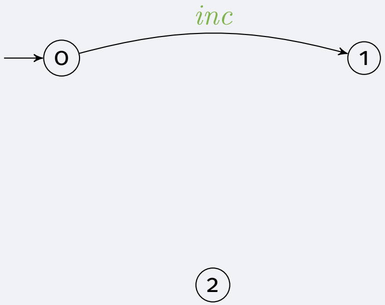
- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement



Example: The modulo 3 counter

Example (Modulo 3 counter)

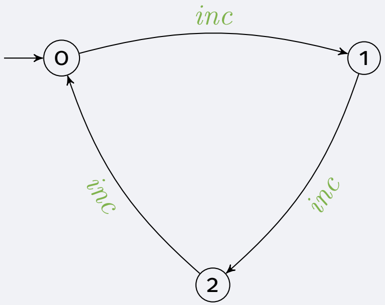
- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement



Example: The modulo 3 counter

Example (Modulo 3 counter)

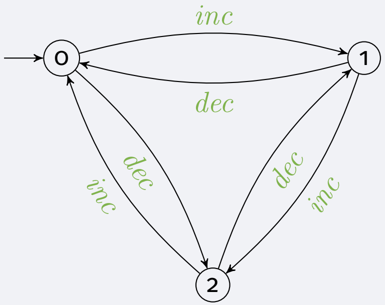
- counts 0, 1, 2
- initial value 0
- allows operations increment and decrement



Example: The modulo 3 counter

Example (Modulo 3 counter)

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



Example: The numerical code

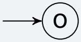
Example (Electronic lock with numerical code)

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

Example: The numerical code

Example (Electronic lock with numerical code)

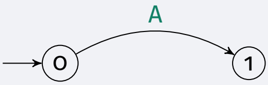
- 3 keys A, B, C
- code to open door: ABA
- if the wrong key is pressed the whole operation has to start again



Example: The numerical code

Example (Electronic lock with numerical code)

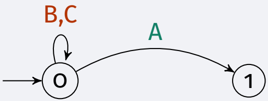
- 3 keys A, B, C
- code to open door: ABA
- if the wrong key is pressed the whole operation has to start again



Example: The numerical code

Example (Electronic lock with numerical code)

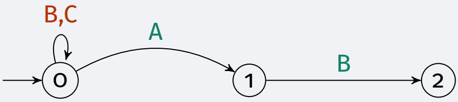
- 3 keys A, B, C
- code to open door: ABA
- if the wrong key is pressed the whole operation has to start again



Example: The numerical code

Example (Electronic lock with numerical code)

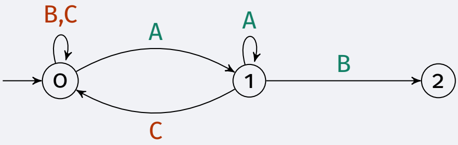
- 3 keys A, B, C
- code to open door: ABA
- if the wrong key is pressed the whole operation has to start again



Example: The numerical code

Example (Electronic lock with numerical code)

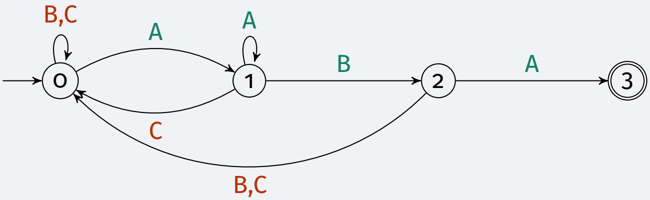
- 3 keys A, B, C
- code to open door: ABA
- if the wrong key is pressed the whole operation has to start again



Example: The numerical code

Example (Electronic lock with numerical code)

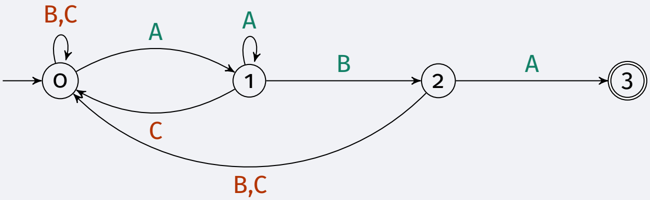
- 3 keys A, B, C
- code to open door: ABA
- if the wrong key is pressed the whole operation has to start again



Example: The numerical code

Example (Electronic lock with numerical code)

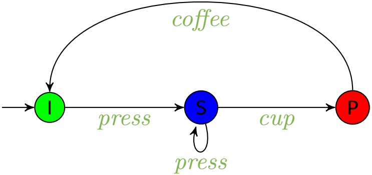
- 3 keys A, B, C
- code to open door: ABA
- if the wrong key is pressed the whole operation has to start again



Remark

The digits in the states represent the number of consecutive correct keys that have been pressed.

Example: The coffee machine

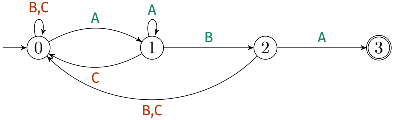


- I Idling
- S Adding sugar
- P Preparing coffee

Executions of a model (1/2)

Definition (Execution)

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

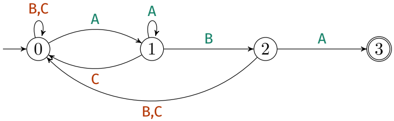


Examples of executions for the numerical code example:

Executions of a model (1/2)

Definition (Execution)

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

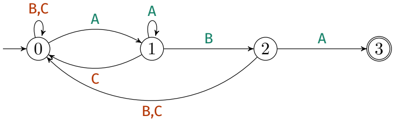


Examples of executions for the numerical code example:

Executions of a model (1/2)

Definition (Execution)

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

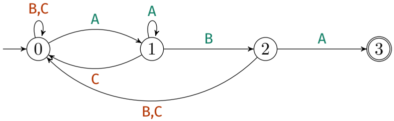


Examples of executions for the numerical code example:

Executions of a model (1/2)

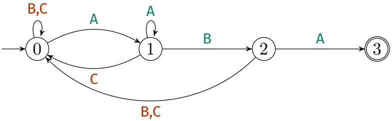
Definition (Execution)

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



Examples of executions for the numerical code example:

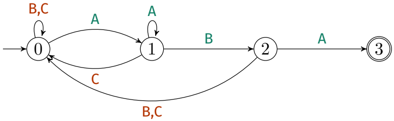
Executions of a model (2/2)



Examples of interesting questions on the numerical code example:

- Which executions lead to opening the door?

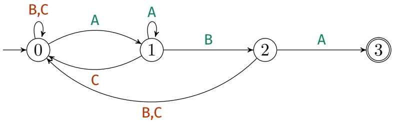
Executions of a model (2/2)



Examples of interesting questions on the numerical code example:

- Which executions lead to opening the door?

Executions of a model (2/2)

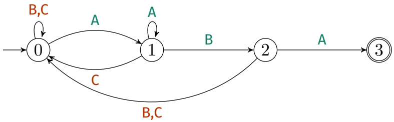


Examples of interesting questions on the numerical code example:

- Which executions lead to opening the door?

- Is there a possible infinite execution?

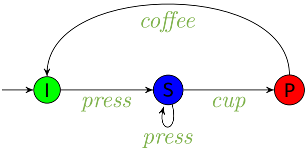
Executions of a model (2/2)



Examples of interesting questions on the numerical code example:

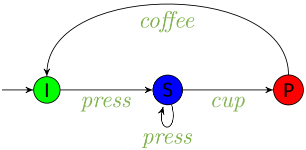
- Which executions lead to opening the door?
- Is there a possible infinite execution?

Executions: Example of the coffee machine



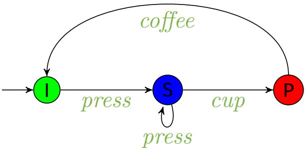
- Example of executions
  - Coffee with no sugar

Executions: Example of the coffee machine



- Example of executions
  - Coffee with no sugar
  - Coffee with 2 doses of sugar

Executions: Example of the coffee machine



- Example of executions
  - Coffee with no sugar
  - Coffee with 2 doses of sugar
  - And so on

Execution tree

Definition (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)

May be infinite!

Execution tree

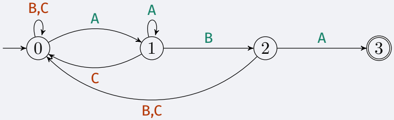
Definition (Execution tree)

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)

May be infinite!

Example (Execution tree for the numerical code example)



Execution tree

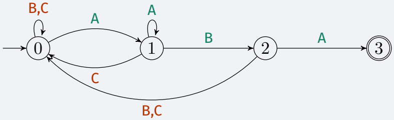
Definition (Execution tree)

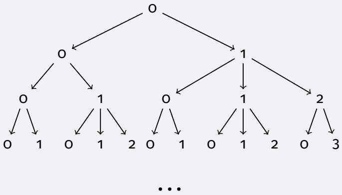
A tree to represent all possible executions

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

May be infinite!

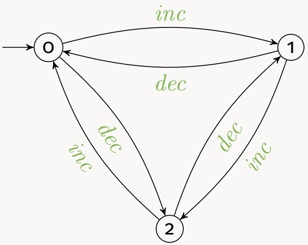
Example (Execution tree for the numerical code example)





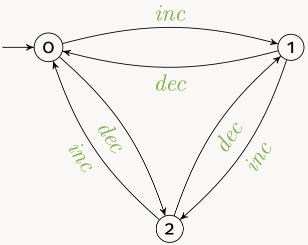
Exercise

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

Numerical code atomic properties

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

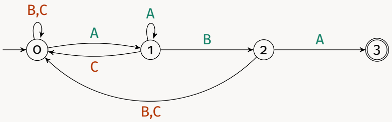
Atomic properties

- 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

Numerical code 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



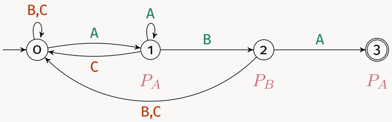
Atomic properties

- 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

Numerical code 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



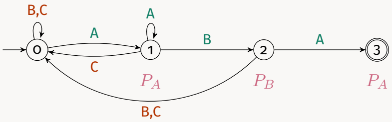
Atomic properties

- 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

Numerical code 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 that the correct code was entered when the door opens

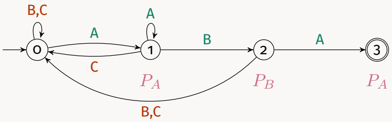
Atomic properties

- 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

Numerical code 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 that the correct code was entered when the door opens

"Whenever the system is in state 3, then the last three keys pressed were ABA."

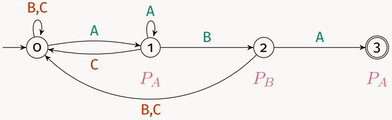
Atomic properties

- 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

Numerical code 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 that the correct code was entered when the door opens

"Whenever the system is in state 3, then the last three keys pressed were ABA."

Outline

Formal definition of automata

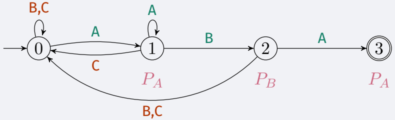
Definition (Automaton)

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

- Q is a finite set of states
- Σ is a finite set of transition labels
- $T \subseteq Q \times \Sigma \times Q$ is a set of transitions
- $q_0 \in Q$ is the (unique) initial state
- $lab : Q \to 2^{AP}$ associates with each state a finite set of atomic propositions
- $F \subseteq Q$ is a set of final states

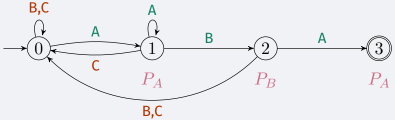
Example

Example (The numerical code example)



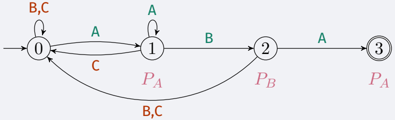
Example

Example (The numerical code example)



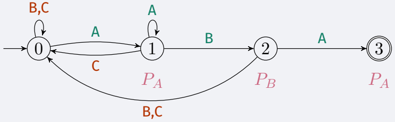
Example

Example (The numerical code example)



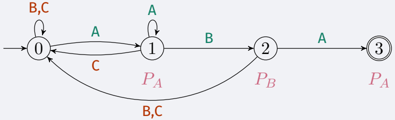
Example

Example (The numerical code example)



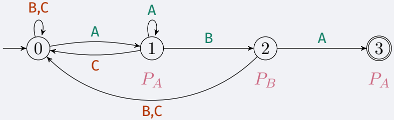
Example

Example (The numerical code example)



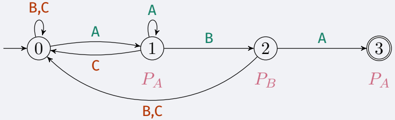
Example

Example (The numerical code example)



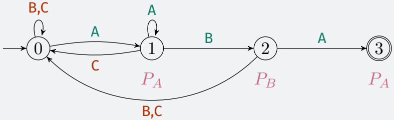
Example

Example (The numerical code example)



Example

Example (The numerical code example)



Exercise: draw the automaton

$\mathcal{A} = \langle Q, \Sigma, T, q_0, lab, F \rangle$, with

- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{a, b, c, d\}$
- $q_0 = q_1$
- $F = \{q_2\}$
- $\forall q \in Q : lab(q) = \emptyset$
- $T = \{(q_1, a, q_1), (q_1, b, q_2), (q_2, c, q_1), (q_2, d, q_2), (q_3, b, q_2)\}$

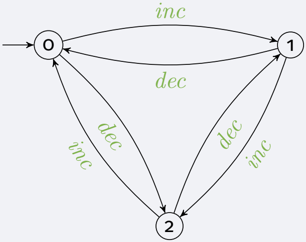
Exercise: draw the automaton

$\mathcal{A} = \langle Q, \Sigma, T, q_0, lab, F \rangle$, with

- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{a, b, c, d\}$
- $q_0 = q_1$
- $F = \{q_2\}$
- $\forall q \in Q : lab(q) = \emptyset$
- $T = \{(q_1, a, q_1), (q_1, b, q_2), (q_2, c, q_1), (q_2, d, q_2), (q_3, b, q_2)\}$

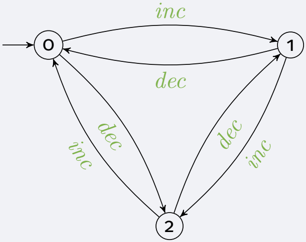
Exercise: formalize the automaton

Exercise (Formal representation of the modulo 3 counter)



Exercise: formalize the automaton

Exercise (Formal representation of the modulo 3 counter)



Behavior: runs

Definition (Run (or path))

- A run (or path) of an automaton $\mathcal{A}$ is a sequence ρ of successive transitions (q_i, a_i, q'_i) of $\mathcal{A}$, i. e., such that $\forall i, q_{i+1} = q'_i$.
  $\rho = q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \xrightarrow{a_3} q_4 \ldots$
- The length of a run ρ is its number of transitions $|\rho| \in \mathbb{N} \cup \{+\infty\}$
- The ith state of ρ is the state q_{i+1} reached after i transitions.

Behavior: executions

Definition (Execution)

- 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 behavior of the automaton.

Exercise: mutual exclusion

Exercise (Mutual exclusion between two processes)

Specification

- 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: mutual exclusion (solution)

Exercise: mutual exclusion (solution)

Exercise: mutual exclusion (solution)

Exercise: mutual exclusion (solution)

Exercise: mutual exclusion (solution)

Outline

Extension with variables

Why and how to use variables?

- More compact models, improving readability (but not necessarily more expressive than pure automata!)
- Guards and updates on transitions

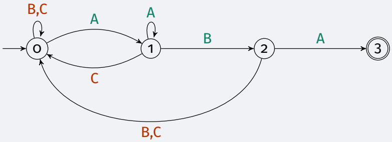
Extension with variables

Why and how to use variables?

- More compact models, improving readability (but not necessarily more expressive than pure automata!)
- Guards and updates on transitions

Example

Example: The numerical code limited to 3 errors



Extension with variables

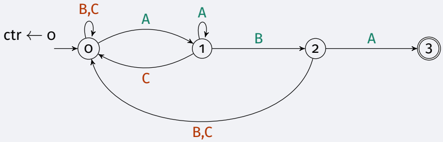
Why and how to use variables?

- More compact models, improving readability (but not necessarily more expressive than pure automata!)
- Guards and updates on transitions

Example

Example: The numerical code limited to 3 errors

var ctr: int;



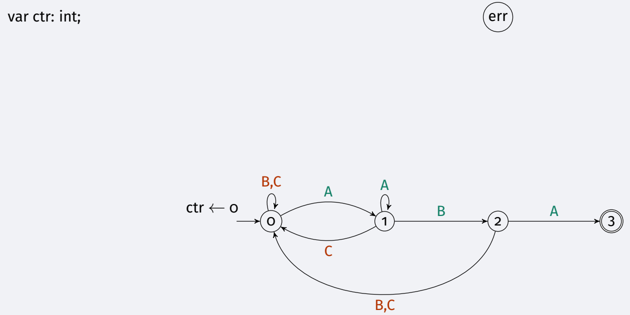
Extension with variables

Why and how to use variables?

- More compact models, improving readability (but not necessarily more expressive than pure automata!)
- Guards and updates on transitions

Example

Example: The numerical code limited to 3 errors



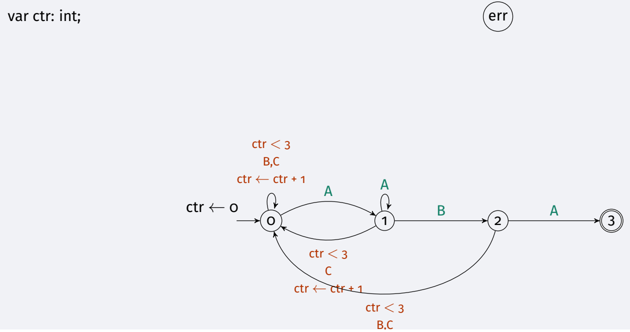
Extension with variables

Why and how to use variables?

- More compact models, improving readability (but not necessarily more expressive than pure automata!)
- Guards and updates on transitions

Example

Example: The numerical code limited to 3 errors



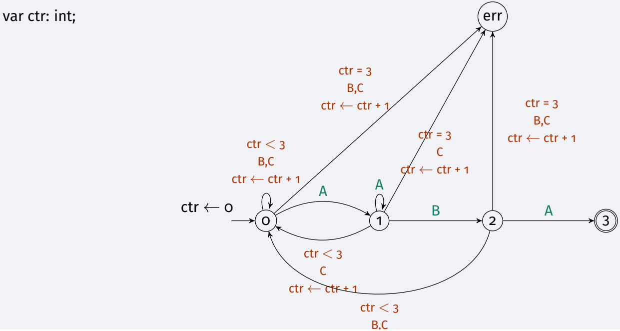
Extension with variables

Why and how to use variables?

- More compact models, improving readability (but not necessarily more expressive than pure automata!)
- Guards and updates on transitions

Example

Example: The numerical code limited to 3 errors



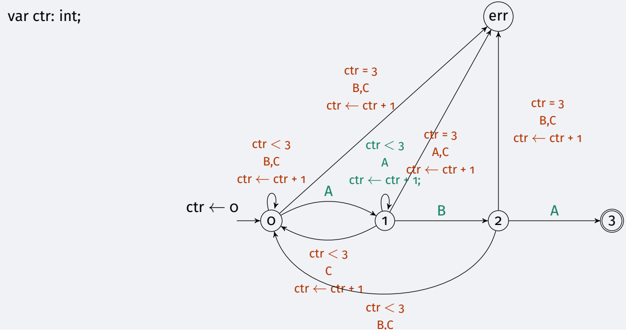
Extension with variables

Why and how to use variables?

- More compact models, improving readability (but not necessarily more expressive than pure automata!)
- Guards and updates on transitions

Example

Example: The numerical code limited to 3 errors



Extension with variables: without?

Exercise (The numerical code with 3 errors without variables)

Extension with variables: without?

Exercise (The numerical code with 3 errors without variables)

Synchronized product

Why?

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

Synchronized product

Why?

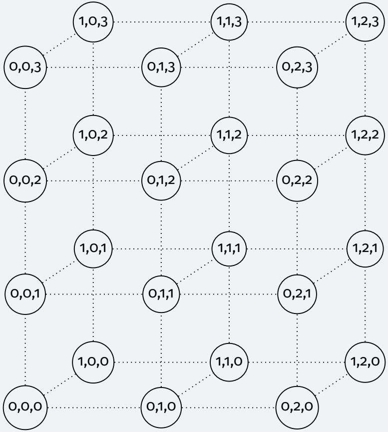
- each component of the system is designed as an automaton
- composition of automata

How?

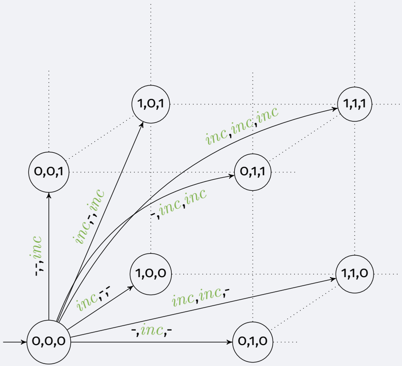
- independent actions lead to a Cartesian product of states
- synchronized actions occur simultaneously

Synchronized product: examples

Example (3 counters, modulo 2, 3, 4: states)

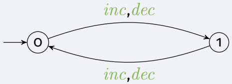


Example (3 counters: some transitions)

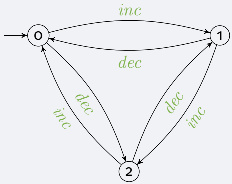


Example: Synchronized counters

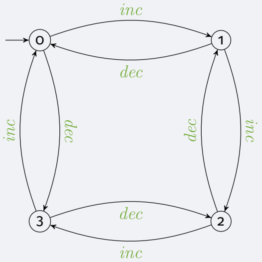
Modulo 2 counter



Modulo 3 counter

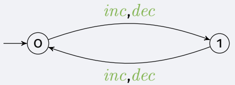


Modulo 4 counter

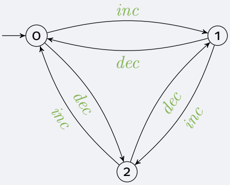


Example: Synchronized counters

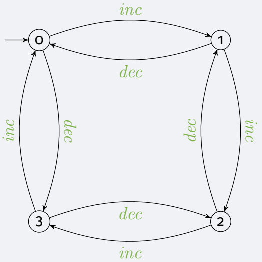
Modulo 2 counter



Modulo 3 counter



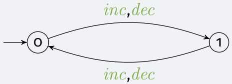
Modulo 4 counter



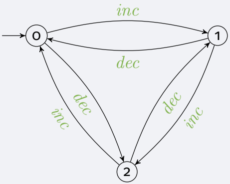
Synchronized actions: all counters increment or decrement simultaneously

Example: Synchronized counters

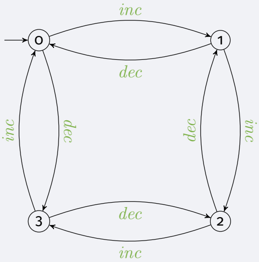
Modulo 2 counter



Modulo 3 counter



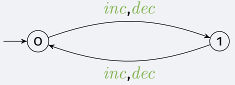
Modulo 4 counter



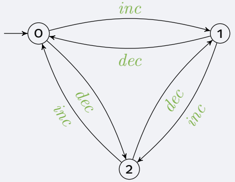
Synchronized actions: all counters increment or decrement simultaneously

Example: Synchronized counters

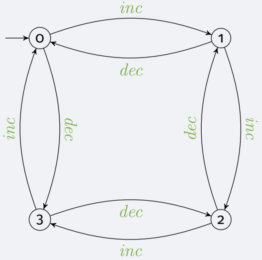
Modulo 2 counter



Modulo 3 counter



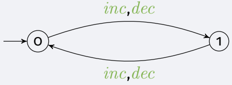
Modulo 4 counter



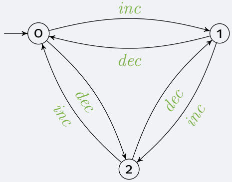
Synchronized actions: all counters increment or decrement simultaneously

Example: Synchronized counters

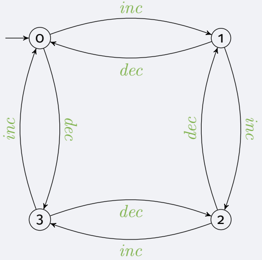
Modulo 2 counter



Modulo 3 counter



Modulo 4 counter



Synchronized 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 $\mathcal{A}_i = \langle Q_i, \Sigma_i, T_i, q_{0i}, lab_i, F_i \rangle$.

Definition (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 $\mathcal{A} = \langle Q, \Sigma, T, q_0, lab, F \rangle$ such that :

- $Q = Q_1 \times \cdots \times Q_n$
- $\Sigma = \prod_{1 \leq i \leq n} \big(\Sigma_i \cup \{\epsilon\} \big)$ (where ϵ represents a silent action)
- $$T = \Big\{ \big((q_1, \dots, q_n), (a_1, \dots, a_n), (q'_1, \dots, q'_n) \big) \mid \forall 1 \leq i \leq n, (a_i = \epsilon \wedge q'_i = q_i) \vee \big(a_i \neq \epsilon \wedge (q_i, a_i, q'_i) \in T_i \big) \Big\}$$
- $q_0 = (q_{01}, \dots, q_{0n})$
- $\forall (q_1, \dots, q_n) \in Q : lab\big((q_1, \dots, q_n)\big) = \bigcup_{1 \leq i \leq n} lab_i(q_i)$
- $F = \big\{ (q_1, \dots, q_n) \in Q \mid \exists 1 \leq i \leq n, q_i \in F_i \big\}$

Formal definition of the synchronized product

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

Definition (Synchronization set)

The synchronization set, denoted by $Sync$, describes all permitted simultaneous actions:

$$Sync \subseteq \prod_{1 \leq i \leq n} \left(\Sigma_i \cup \{\epsilon\} \right)$$

Formal definition of the synchronized product

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

Definition (Synchronization set)

The synchronization set, denoted by $Sync$, describes all permitted simultaneous actions:

$$Sync \subseteq \prod_{1 \leq i \leq n} \left(\Sigma_i \cup \{\epsilon\} \right)$$

Definition (Synchronized product of automata)

The synchronized product of $(\mathcal{A}_i)_{1 \leq i \leq n}$ over a set $Sync$ is the Cartesian product restricted to $\Sigma = Sync$.

Synchronization by message passing

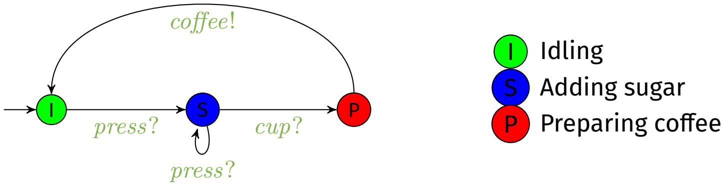
Message passing: a special case of synchronized product

$m!$ send a message m

$m?$ receive a message m

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

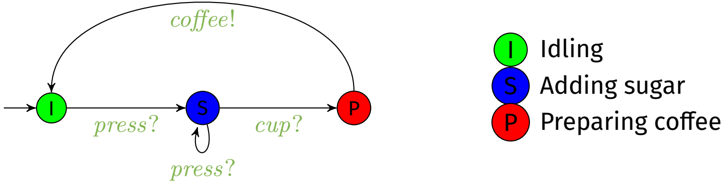
Synchronization by message passing: coffee machine



Exercise

Is this coffee machine environment-unfriendly (providing disposable cups) or environment-friendly (requesting the user to bring their own cup)?

Synchronization by message passing: coffee machine



Exercise

Is this coffee machine environment-unfriendly (providing disposable cups) or environment-friendly (requesting the user to bring their own cup)?

A coffee drinker (sugarless)

- Specify a coffee drinker automaton $\mathcal{A}_{D1}$ that performs forever the following actions:
  1. press the button once
  2. place the cup
  3. wait for the coffee
  4. drink the coffee
  5. put the cup to the washing machine

  and so on

A coffee drinker (sugarless)

- Specify a coffee drinker automaton $\mathcal{A}_{D1}$ that performs forever the following actions:
  1. press the button once
  2. place the cup
  3. wait for the coffee
  4. drink the coffee
  5. put the cup to the washing machine

  and so on

A coffee drinker (sugar-addicted)

- Specify a coffee drinker automaton $\mathcal{A}_{D2}$ that works just as $\mathcal{A}_{D1}$ except that they can nondeterministically ask for 0, 1 or 2 doses of sugar.

A coffee drinker (sugar-addicted)

- Specify a coffee drinker automaton $\mathcal{A}_{D2}$ that works just as $\mathcal{A}_{D1}$ except that they can nondeterministically ask for 0, 1 or 2 doses of sugar.

A washing machine

- Specify a washing machine automaton $\mathcal{A}_W$ that accepts cups to wash, and once 5 cups are placed into the washing machine, then the machine washes all cups.

A washing machine

- Specify a washing machine automaton $\mathcal{A}_W$ that accepts cups to wash, and once 5 cups are placed into the washing machine, then the machine washes all cups.

Synchronization by message passing: lift example

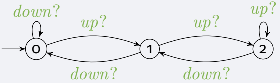
Example (A small lift)

Model of a lift in a 3-level building, made of:

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

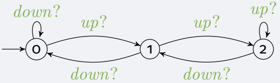
Synchronization by message passing: lift example (exercise)

Cabin



Synchronization by message passing: lift example (exercise)

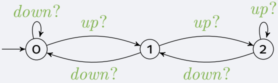
Cabin



i^{th} door

Synchronization by message passing: lift example (exercise)

Cabin

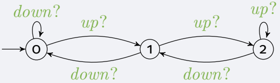


i^{th} door

Controller

Synchronization by message passing: lift example (exercise)

Cabin



i^{th} door

Controller

Examples of properties

Exercise: Mutual exclusion problem

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 generalize it to n processes

Exercise: Mutual exclusion problem (solution)

Exercise: Mutual exclusion problem (solution)

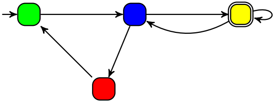
Exercise: Mutual exclusion problem (solution)

Exercise: Mutual exclusion problem (solution)

Outline

Model checking timed concurrent systems

- Principle of model checking [BK08]



A model of the system

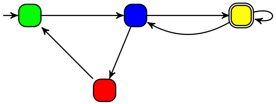
 is unreachable

A property to be verified

- [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* MIT Press, 2008. ISBN: 978-0-262-02649-9

Model checking timed concurrent systems

- Principle of model checking [BK08]



A model of the system

$\overset{?}{\models}$

 is unreachable

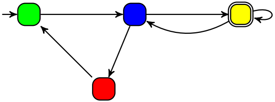
A property to be verified

- Question: does the model of the system satisfy the property?

- [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* MIT Press, 2008. ISBN: 978-0-262-02649-9

Model checking timed concurrent systems

- Principle of model checking [BK08]



A model of the system

$\stackrel{?}{\models}$

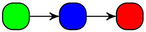
 is unreachable

A property to be verified

- Question: does the model of the system satisfy the property?

Yes

No



Counterexample

Turing award (2007) to Edmund M. Clarke, Allen Emerson and Joseph Sifakis

- [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9

Outline

Introduction to temporal logics

- Express dynamic behavior 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 behavior of the system
- Use formal syntax and semantics to avoid any ambiguity
- Capture statements and reasoning that involve the notion of order in time

Example (properties for the lift)

- “any request must ultimately be satisfied”

Introduction to temporal logics

- Express dynamic behavior of the system
- Use formal syntax and semantics to avoid any ambiguity
- Capture statements and reasoning that involve the notion of order in time

Example (properties for the lift)

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

Introduction to temporal logics

- Express dynamic behavior of the system
- Use formal syntax and semantics to avoid any ambiguity
- Capture statements and reasoning that involve the notion of order in time

Example (properties for the lift)

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

The logic CTL*

- Atomic propositions
- Logical (Boolean) operators:
  - `true`, `false`
  - $\neg$ (negation)
  - $\wedge$ (and), $\vee$ (or)
  - $\implies$ (logical implication), $\iff$ (if and only if)
- Temporal modal operators:
  - X (neXt), F (Future), G (Globally)
  - U (Until), W (Weak until)
  - R (Release)
- Quantifiers over paths: A (Always), E (Exists)

The logic CTL*

- Atomic propositions
- Logical (Boolean) operators:
  - `true`, `false`
  - $\neg$ (negation)
  - $\wedge$ (and), $\vee$ (or)
  - $\implies$ (logical implication), $\iff$ (if and only if)
- Temporal modal operators:
  - X (neXt), F (Future), G (Globally)
  - U (Until), W (Weak until)
  - R (Release)
- Quantifiers over paths: A (Always), E (Exists)

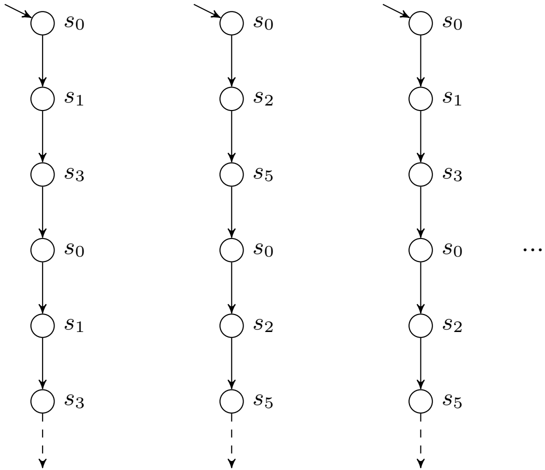
Two main subsets of CTL*

LTL Linear-time Temporal Logic: events are totally ordered

CTL Computation Tree Logic: events are partially ordered

Outline

LTL: Linear-time Temporal Logic



Syntax of LTL

LTL expresses formulas on the order between the future atomic propositions for one given path, over a set of atomic propositions AP

Definition (Minimal syntax of LTL)

$$LTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \psi$$

Syntax of LTL

LTL expresses formulas on the order between the future atomic propositions for one given path, over a set of atomic propositions AP

Definition (Minimal syntax of LTL)

$$LTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \psi$$

Additional operators:

- “Eventually”: $\mathsf{F}\varphi \equiv$

Syntax of LTL

LTL expresses formulas on the order between the future atomic propositions for one given path, over a set of atomic propositions AP

Definition (Minimal syntax of LTL)

$$LTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \lor \varphi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\psi$$

Additional operators:

- "Eventually": $\mathsf{F}\varphi \equiv$
- "Globally": $\mathsf{G}\varphi \equiv$

Syntax of LTL

LTL expresses formulas on the order between the future atomic propositions for one given path, over a set of atomic propositions AP

Definition (Minimal syntax of LTL)

$$LTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \psi$$

Additional operators:

- "Eventually": $\mathsf{F}\varphi \equiv$
- "Globally": $\mathsf{G}\varphi \equiv$
- "Weak until": $\varphi \mathsf{W} \psi \equiv$

Syntax of LTL

LTL expresses formulas on the order between the future atomic propositions for one given path, over a set of atomic propositions AP

Definition (Minimal syntax of LTL)

$$LTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \psi$$

Additional operators:

- "Eventually": $\mathsf{F}\varphi \equiv$
- "Globally": $\mathsf{G}\varphi \equiv$
- "Weak until": $\varphi \mathsf{W} \psi \equiv$
  - W is similar to U but ψ may never happen
- "Release": $\psi \mathsf{R} \varphi \equiv$

Syntax of LTL

LTL expresses formulas on the order between the future atomic propositions for one given path, over a set of atomic propositions AP

Definition (Minimal syntax of LTL)

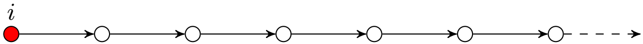
$$LTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \psi$$

Additional operators:

- "Eventually": $\mathsf{F}\varphi \equiv$
- "Globally": $\mathsf{G}\varphi \equiv$
- "Weak until": $\varphi \mathsf{W} \psi \equiv$
  - W is similar to U but ψ may never happen
- "Release": $\psi \mathsf{R} \varphi \equiv$

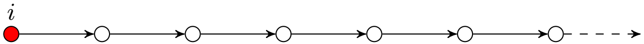
Informal illustration of the LTL semantics

- p

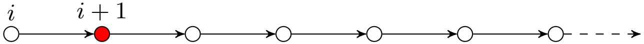


Informal illustration of the LTL semantics

- p

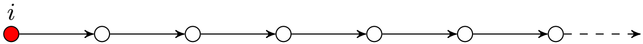


- $\mathsf{X}\varphi$

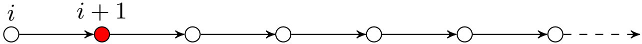


Informal illustration of the LTL semantics

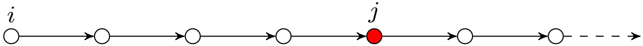
- p



- $\mathsf{X}\varphi$

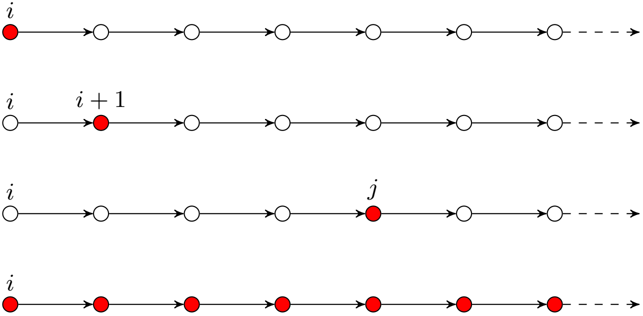


- $\mathsf{F}\varphi$



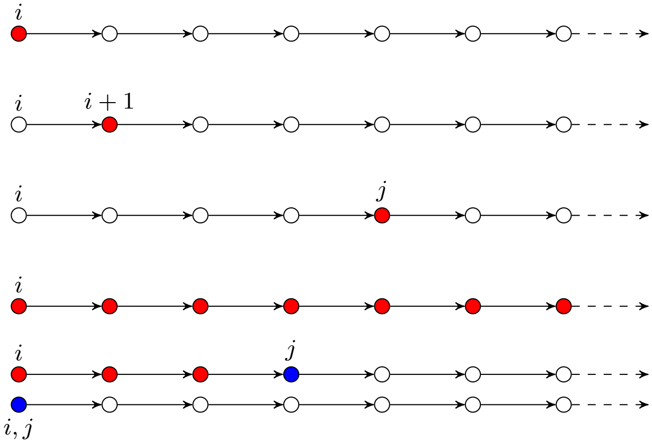
Informal illustration of the LTL semantics

- p
- $\mathsf{X}\varphi$
- $\mathsf{F}\varphi$
- $\mathsf{G}\varphi$



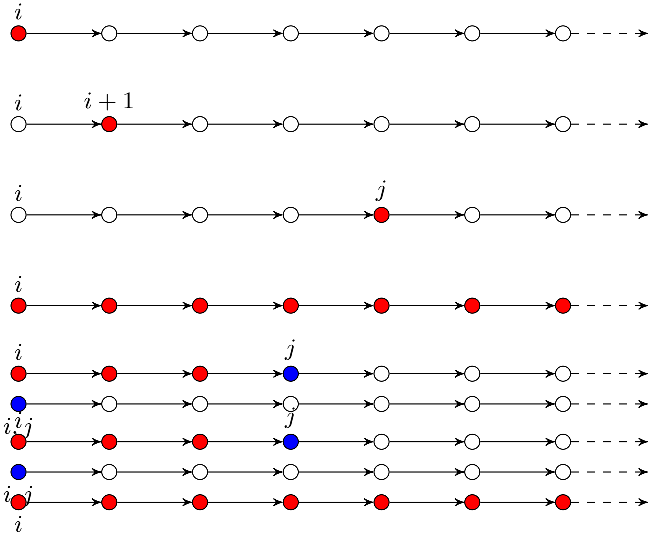
Informal illustration of the LTL semantics

- p
- $\mathsf{X}\varphi$
- $\mathsf{F}\varphi$
- $\mathsf{G}\varphi$
- $\varphi_1 \mathsf{U} \varphi_2$



Informal illustration of the LTL semantics

- p
- $\mathsf{X}\varphi$
- $\mathsf{F}\varphi$
- $\mathsf{G}\varphi$
- $\varphi_1 \mathsf{U} \varphi_2$
- $\varphi_1 \mathsf{W} \varphi_2$



Semantics of LTL

Let ρ be a finite run and $p \in AP$ an atomic proposition.
"$\rho, i \models \varphi$" denotes that, at position i of its execution, ρ satisfies formula φ.

Definition (Semantics of LTL)

$\rho, i \models p$	if	$p \in lab\big(\rho(i)\big)$
$\rho, i \models \neg\varphi$	if	$\rho, i \not\models \varphi$
$\rho, i \models \varphi \wedge \psi$	if	$\rho, i \models \varphi$ and $\rho, i \models \psi$
$\rho, i \models \mathsf{X}\varphi$	if	$i < \lvert\rho\rvert$ and $\rho, i+1 \models \varphi$
$\rho, i \models \mathsf{F}\varphi$	if	$\exists j$ s.t. $i \leq j \leq \lvert\rho\rvert : \rho, j \models \varphi$
$\rho, i \models \mathsf{G}\varphi$	if	$\forall j$ s.t. $i \leq j \leq \lvert\rho\rvert : \rho, j \models \varphi$
$\rho, i \models \varphi \mathsf{U} \psi$	if	$\exists j$ s.t. $i \leq j \leq \lvert\rho\rvert : \rho, j \models \psi$ and $\forall k$ s.t. $i \leq k < j : \rho, k \models \varphi$

Exercise: Additional Boolean operators

Exercise

Express $\vee$, $\implies$, $\iff$ by using $\neg$ and $\wedge$

$$\varphi \vee \psi \quad \equiv$$

Exercise: Additional Boolean operators

Exercise

Express $\vee$, $\implies$, $\iff$ by using $\neg$ and $\wedge$

$\varphi \vee \psi \quad \equiv$

$\varphi \implies \psi \equiv$

Exercise: Additional Boolean operators

Exercise

Express $\vee$, $\implies$, $\iff$ by using $\neg$ and $\wedge$

$$\varphi \vee \psi \quad \equiv$$

$$\varphi \implies \psi \equiv$$

$$\varphi \iff \psi \equiv$$

Exercise: Additional Boolean operators

Exercise

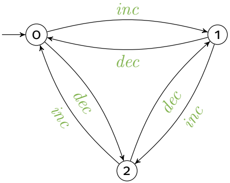
Express $\lor$, $\implies$, $\iff$ by using $\neg$ and $\land$

$$\varphi \lor \psi \equiv$$

$$\varphi \implies \psi \equiv$$

$$\varphi \iff \psi \equiv$$

Examples of LTL formulae (counter)



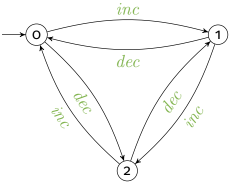
- What do the following formulae mean?
- Which runs satisfy the LTL property?

Example (Modulo 3 counter)

1. $XXX0$

Examples of LTL formulae (counter)

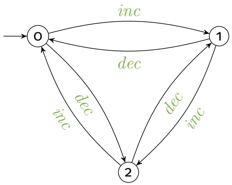
- What do the following formulae mean?
- Which runs satisfy the LTL property?



Example (Modulo 3 counter)

1. $\mathsf{XXX}0$

Examples of LTL formulae (counter)

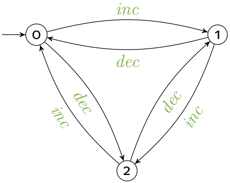


- What do the following formulae mean?
- Which runs satisfy the LTL property?

Example (Modulo 3 counter)

1. $\mathsf{XXX}0$
2. $\mathsf{F}(1 \vee 2)$

Examples of LTL formulae (counter)

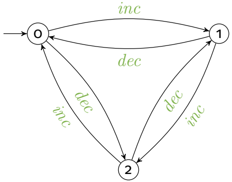


- What do the following formulae mean?
- Which runs satisfy the LTL property?

Example (Modulo 3 counter)

1. $\mathsf{XXX}0$
2. $\mathsf{F}(1 \vee 2)$

Examples of LTL formulae (counter)

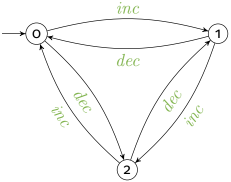


- What do the following formulae mean?
- Which runs satisfy the LTL property?

Example (Modulo 3 counter)

1. $\mathsf{XXX}0$
2. $\mathsf{F}(1 \vee 2)$
3. $\mathsf{F}1$

Examples of LTL formulae (counter)

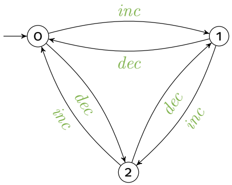


- What do the following formulae mean?
- Which runs satisfy the LTL property?

Example (Modulo 3 counter)

1. $\mathsf{XXX}0$
2. $\mathsf{F}(1 \lor 2)$
3. $\mathsf{F}1$

Examples of LTL formulae (counter)



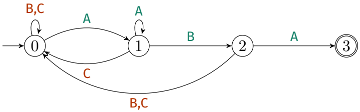
- What do the following formulae mean?
- Which runs satisfy the LTL property?

Example (Modulo 3 counter)

1. $\mathsf{XXX}0$
2. $\mathsf{F}(1 \vee 2)$
3. $\mathsf{F}1$

Examples of LTL formulae (numerical code)

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

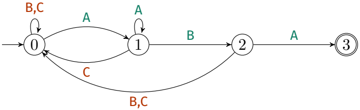


Example (The numerical code)

1. $\mathsf{F}3$

Examples of LTL formulae (numerical code)

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

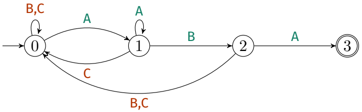


Example (The numerical code)

1. $\mathsf{F}3$

Examples of LTL formulae (numerical code)

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



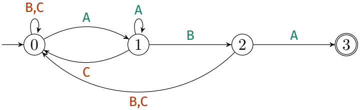
Example (The numerical code)

1. $\mathsf{F}3$

2. $\mathsf{G}\neg 3$

Examples of LTL formulae (numerical code)

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



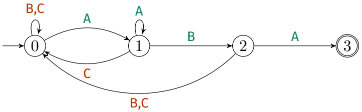
Example (The numerical code)

1. $\mathsf{F}3$

2. $\mathsf{G}\neg 3$

Examples of LTL formulae (numerical code)

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



Example (The numerical code)

1. $\mathsf{F}3$

2. $\mathsf{G}\neg 3$

Exercises: Writing LTL formulae (1/3)

Exercise

Express in LTL the following properties:

- "The plane will never crash"

Exercises: Writing LTL formulae (1/3)

Exercise

Express in LTL the following properties:

- “The plane will never crash”
- “I will eventually get a job”

Exercises: Writing LTL formulae (1/3)

Exercise

Express in LTL the following properties:

- “The plane will never crash”
- “I will eventually get a job”
- “Every time I ask a question, the teacher will eventually answer me”

Exercises: Writing LTL formulae (1/3)

Exercise

Express in LTL the following properties:

- “The plane will never crash”
- “I will eventually get a job”
- “Every time I ask a question, the teacher will eventually answer me”
- “If I ask for food infinitely often, then I will get food infinitely often”

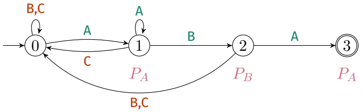
Exercises: Writing LTL formulae (1/3)

Exercise

Express in LTL the following properties:

- "The plane will never crash"
- "I will eventually get a job"
- "Every time I ask a question, the teacher will eventually answer me"
- "If I ask for food infinitely often, then I will get food infinitely often"

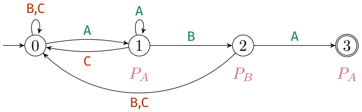
Exercises: Writing LTL formulae (2/3)



Exercise (Numerical code)

1. Write an LTL formula satisfied by all runs where keys A and B have successively been pressed

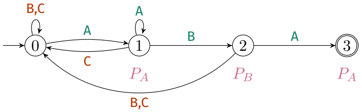
Exercises: Writing LTL formulae (2/3)



Exercise (Numerical code)

1. Write an LTL formula satisfied by all runs where keys A and B have successively been pressed
2. Write an LTL formula that characterizes the infinite loop on state 0

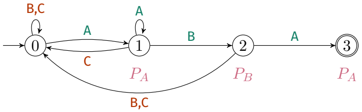
Exercises: Writing LTL formulae (2/3)



Exercise (Numerical code)

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

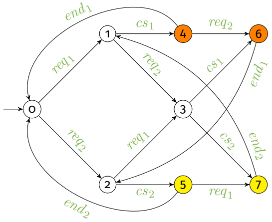
Exercises: Writing LTL formulae (2/3)



Exercise (Numerical code)

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

Exercises: Writing LTL formulae (3/3)



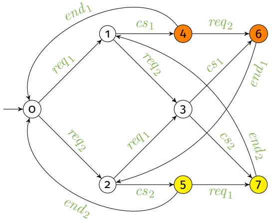
R_1: states 1, 3, 7; R_2: states 2, 3, 6;
CS_1: states 4, 6; CS_2: states 5, 7;
I_1: states 0, 2, 5; I_2: states 0, 1, 4

Exercise (Mutual exclusion between two processes)

Write an LTL formula satisfied by all runs where:

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

Exercises: Writing LTL formulae (3/3)



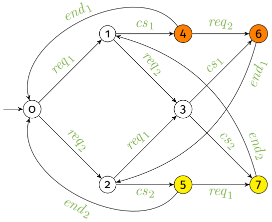
R_1: states 1, 3, 7; R_2: states 2, 3, 6;
CS_1: states 4, 6; CS_2: states 5, 7;
I_1: states 0, 2, 5; I_2: states 0, 1, 4

Exercise (Mutual exclusion between two processes)

Write an LTL formula satisfied by all runs where:

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

Exercises: Writing LTL formulae (3/3)



R_1: states 1, 3, 7; R_2: states 2, 3, 6;
CS_1: states 4, 6; CS_2: states 5, 7;
I_1: states 0, 2, 5; I_2: states 0, 1, 4

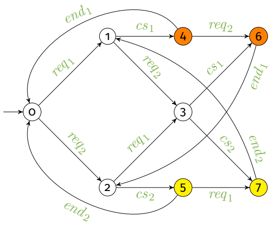
Exercise (Mutual exclusion between two processes)

Write an LTL formula satisfied by all runs where:

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

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

Exercises: Writing LTL formulae (3/3)



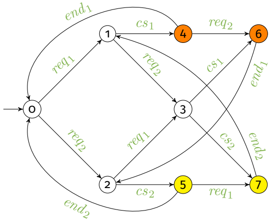
R_1: states 1, 3, 7; R_2: states 2, 3, 6;
CS_1: states 4, 6; CS_2: states 5, 7;
I_1: states 0, 2, 5; I_2: states 0, 1, 4

Exercise (Mutual exclusion between two processes)

Write an LTL formula satisfied by all runs where:

1. The two processes are not simultaneously in the critical section
2. Whenever process 1 requests to enter the critical section, it will eventually succeed

Exercises: Writing LTL formulae (3/3)



R_1: states 1, 3, 7; R_2: states 2, 3, 6;
CS_1: states 4, 6; CS_2: states 5, 7;
I_1: states 0, 2, 5; I_2: states 0, 1, 4

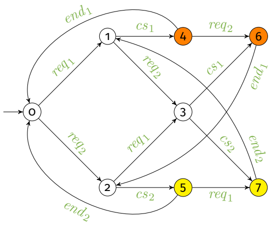
Exercise (Mutual exclusion between two processes)

Write an LTL formula satisfied by all runs where:

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

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

Exercises: Writing LTL formulae (3/3)



R_1: states 1, 3, 7; R_2: states 2, 3, 6;
CS_1: states 4, 6; CS_2: states 5, 7;
I_1: states 0, 2, 5; I_2: states 0, 1, 4

Exercise (Mutual exclusion between two processes)

Write an LTL formula satisfied by all runs where:

1. The two processes are not simultaneously in the critical section
2. Whenever process 1 requests to enter the critical section, it will eventually succeed

Exercise: Equivalences

Exercise

Using the aforementioned formal semantics, prove that:

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

Exercise: Equivalences

Exercise

Using the aforementioned formal semantics, prove that:

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

2. $\mathsf{G}\varphi \equiv \neg(\mathsf{F}\neg\varphi)$

Exercise: Equivalences

Exercise

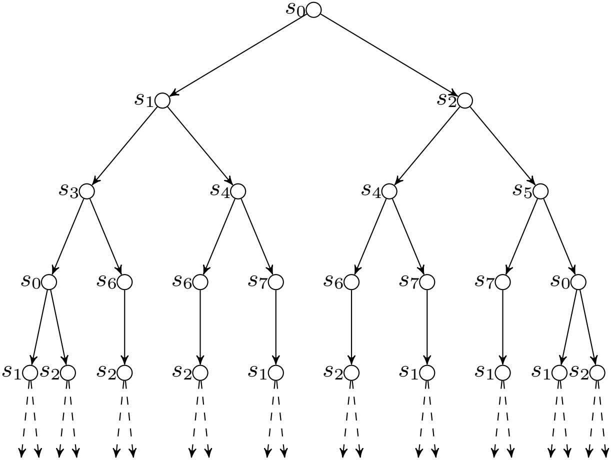
Using the aforementioned formal semantics, prove that:

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

2. $\mathsf{G}\varphi \equiv \neg(\mathsf{F}\neg\varphi)$

Outline

CTL: branching time



CTL (Computation tree logic)

CTL expresses formulas on the order between the future atomic propositions for some or for all paths, over a set of atomic propositions AP

Definition (Minimal syntax of CTL)

$$CTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$$

Additional operators: F, G, R, W

Semantics of CTL

Same as LTL, plus:

$$\rho, i \models \mathsf{E}\varphi \quad \text{if} \quad \exists \rho' : \rho(0) \dots \rho(i) = \rho'(0) \dots \rho'(i) \text{ and } \rho', i \models \varphi$$
$$\rho, i \models \mathsf{A}\varphi \quad \text{if} \quad \forall \rho' : \rho(0) \dots \rho(i) = \rho'(0) \dots \rho'(i) \text{ we have } \rho', i \models \varphi$$

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

(This restriction does not apply in CTL*)

CTL: Alternating quantifiers

A path quantifier must always be followed by a temporal operator.

Some useful combinations:

CTL: Alternating quantifiers

A path quantifier must always be followed by a temporal operator.

Some useful combinations:

"there exists a path for which the next state is..."

CTL: Alternating quantifiers

A path quantifier must always be followed by a temporal operator.

Some useful combinations:

"there exists a path for which the next state is..."
"for all possible paths, the next state is..."

CTL: Alternating quantifiers

A path quantifier must always be followed by a temporal operator.

Some useful combinations:

"there exists a path for which the next state is..."
"for all possible paths, the next state is..."
"it is possible that eventually..."

CTL: Alternating quantifiers

A path quantifier must always be followed by a temporal operator.

Some useful combinations:

“there exists a path for which the next state is. . . ”
“for all possible paths, the next state is. . . ”
“it is possible that eventually. . . ”
“in any case, eventually. . . ”

CTL: Alternating quantifiers

A path quantifier must always be followed by a temporal operator.

Some useful combinations:

"there exists a path for which the next state is…"
"for all possible paths, the next state is…"
"it is possible that eventually…"
"in any case, eventually…"
"in any case, for all states…"

CTL: Alternating quantifiers

A path quantifier must always be followed by a temporal operator.

Some useful combinations:

"there exists a path for which the next state is. . . "
"for all possible paths, the next state is. . . "
"it is possible that eventually. . . "
"in any case, eventually. . . "
"in any case, for all states. . . "

Illustration of the CTL semantics (1/8)

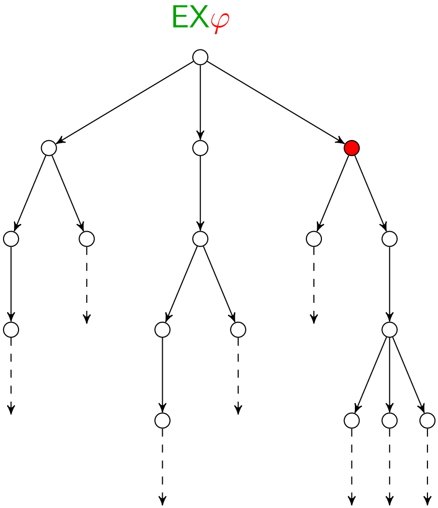


Illustration of the CTL semantics (2/8)

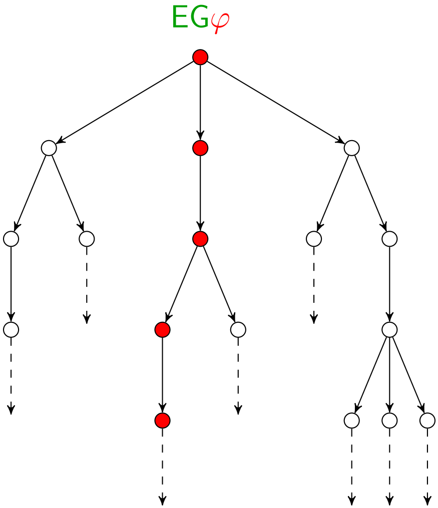


Illustration of the CTL semantics (3/8)

$\mathsf{E}\varphi\mathsf{U}\psi$

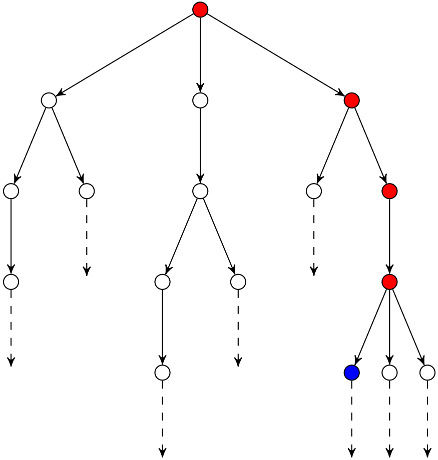


Illustration of the CTL semantics (4/8)

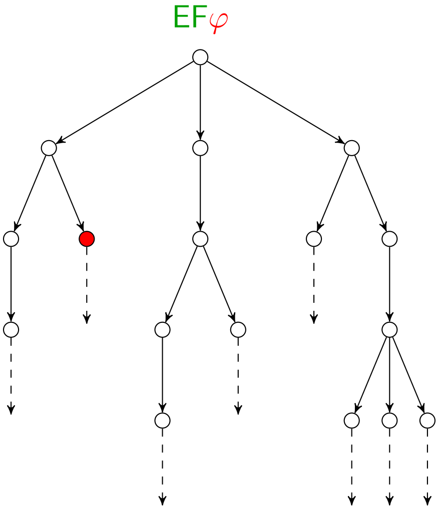


Illustration of the CTL semantics (5/8)

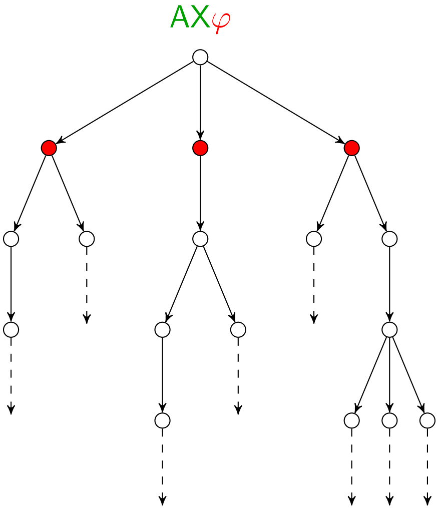


Illustration of the CTL semantics (6/8)

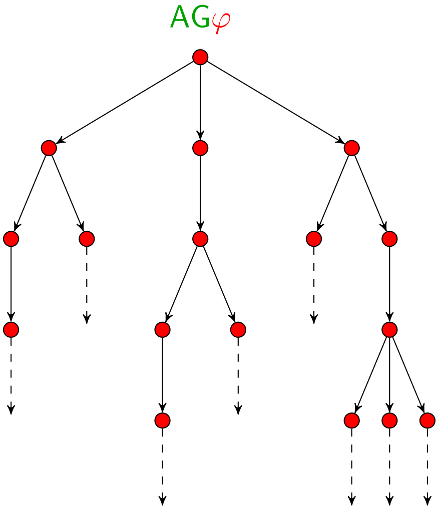


Illustration of the CTL semantics (7/8)

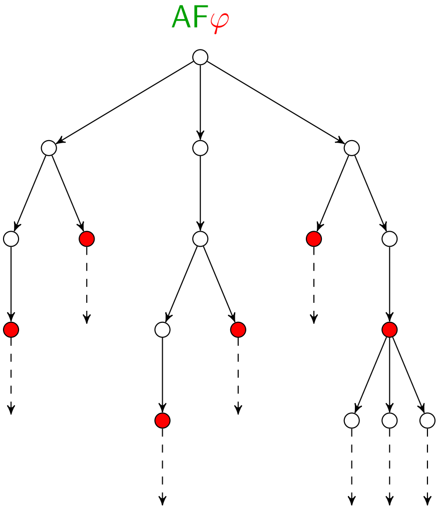


Illustration of the CTL semantics (8/8)

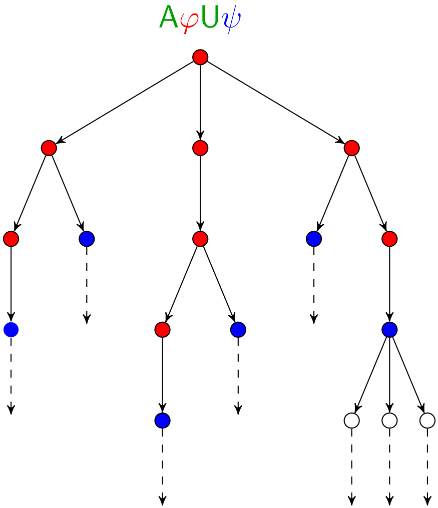
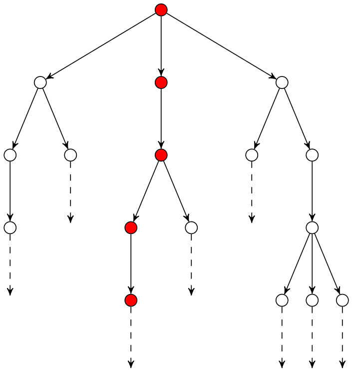


Illustration of the CTL semantics: Exercise

On which states are the following formulae valid?

1. $\mathsf{EX}\varphi$
2. $\mathsf{EF}\varphi$
3. $\mathsf{EG}\varphi$
4. $\mathsf{AX}\varphi$
5. $\mathsf{AG}\varphi$



Formally defining additional operators

Definition (Minimal syntax of CTL (recalled))

$$CTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$$

Formally defining additional operators

Definition (Minimal syntax of CTL (recalled))

$$CTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$$

Some additional operators:

- "Always next": $\mathsf{AX}\varphi \equiv$

Formally defining additional operators

Definition (Minimal syntax of CTL (recalled))

$$CTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$$

Some additional operators:

- “Always next”: $\mathsf{AX}\varphi \equiv$
- “Exists eventually”: $\mathsf{EF}\varphi \equiv$

Formally defining additional operators

Definition (Minimal syntax of CTL (recalled))

$$CTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$$

Some additional operators:

- "Always next": $\mathsf{AX}\varphi \equiv$
- "Exists eventually": $\mathsf{EF}\varphi \equiv$
- "Always eventually": $\mathsf{AF}\varphi \equiv$

Formally defining additional operators

Definition (Minimal syntax of CTL (recalled))

$$CTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$$

Some additional operators:

- "Always next": $\mathsf{AX}\varphi \equiv$
- "Exists eventually": $\mathsf{EF}\varphi \equiv$
- "Always eventually": $\mathsf{AF}\varphi \equiv$
- "Exists globally": $\mathsf{EG}\varphi \equiv$

Formally defining additional operators

Definition (Minimal syntax of CTL (recalled))

$$CTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$$

Some additional operators:

- "Always next": $\mathsf{AX}\varphi \equiv$
- "Exists eventually": $\mathsf{EF}\varphi \equiv$
- "Always eventually": $\mathsf{AF}\varphi \equiv$
- "Exists globally": $\mathsf{EG}\varphi \equiv$
- "Always globally": $\mathsf{AG}\varphi \equiv$

Formally defining additional operators

Definition (Minimal syntax of CTL (recalled))

$$CTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$$

Some additional operators:

- "Always next": $\mathsf{AX}\varphi \equiv$
- "Exists eventually": $\mathsf{EF}\varphi \equiv$
- "Always eventually": $\mathsf{AF}\varphi \equiv$
- "Exists globally": $\mathsf{EG}\varphi \equiv$
- "Always globally": $\mathsf{AG}\varphi \equiv$
- "Exists weak until": $\mathsf{E}\varphi\mathsf{W}\psi \equiv$

Formally defining additional operators

Definition (Minimal syntax of CTL (recalled))

$$CTL \ni \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi$$

Some additional operators:

- "Always next": $\mathsf{AX}\varphi \equiv$
- "Exists eventually": $\mathsf{EF}\varphi \equiv$
- "Always eventually": $\mathsf{AF}\varphi \equiv$
- "Exists globally": $\mathsf{EG}\varphi \equiv$
- "Always globally": $\mathsf{AG}\varphi \equiv$
- "Exists weak until": $\mathsf{E}\varphi\mathsf{W}\psi \equiv$

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)
- "I may eventually get a job" (reachability property)

CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)
- "I may eventually get a job" (reachability property)
- "I may study for the rest of my life (and beyond)"

CTL: Examples

Express in CTL the following properties:

- “Whatever happens, the plane will never crash” (safety property)
- “Whatever happens, I will eventually get a job” (liveness property)
- “I may eventually get a job” (reachability property)
- “I may study for the rest of my life (and beyond)”
- “It can always happen that suddenly I discover formal methods and then I may use them for the rest of time”

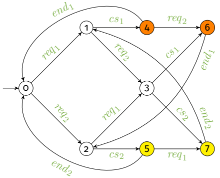
CTL: Examples

Express in CTL the following properties:

- "Whatever happens, the plane will never crash" (safety property)
- "Whatever happens, I will eventually get a job" (liveness property)
- "I may eventually get a job" (reachability property)
- "I may study for the rest of my life (and beyond)"
- "It can always happen that suddenly I discover formal methods and then I may use them for the rest of time"

Examples of CTL formulae: Mutual exclusion

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

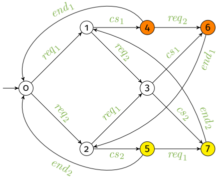


Exercise (Mutual exclusion between 2 processes)

1. $\mathsf{AG}\neg(CS_1 \wedge CS_2)$

Examples of CTL formulae: Mutual exclusion

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

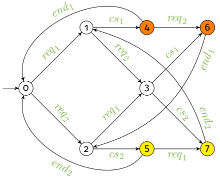


Exercise (Mutual exclusion between 2 processes)

1. $\text{AG}\neg(CS_1 \land CS_2)$

Examples of CTL formulae: Mutual exclusion

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



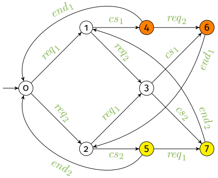
Exercise (Mutual exclusion between 2 processes)

1. $\mathsf{AG}\neg(CS_1 \wedge CS_2)$

2. $\mathsf{AG}(R_1 \implies \mathsf{AF}\, CS_1)$

Examples of CTL formulae: Mutual exclusion

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



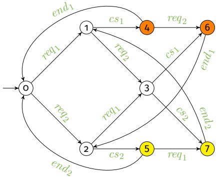
Exercise (Mutual exclusion between 2 processes)

1. $\mathsf{AG}\neg(CS_1 \wedge CS_2)$

2. $\mathsf{AG}(R_1 \implies \mathsf{AF}\,CS_1)$

Examples of CTL formulae: Mutual exclusion

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



Exercise (Mutual exclusion between 2 processes)

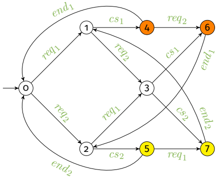
1. $\mathsf{AG}\neg(CS_1 \wedge CS_2)$

2. $\mathsf{AG}(R_1 \implies \mathsf{AF}\, CS_1)$

3. $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$

Examples of CTL formulae: Mutual exclusion

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



Exercise (Mutual exclusion between 2 processes)

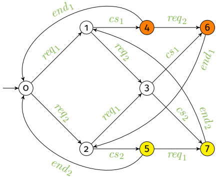
1. $\mathsf{AG}\neg(CS_1 \land CS_2)$

2. $\mathsf{AG}(R_1 \implies \mathsf{AF}\,CS_1)$

3. $\mathsf{AG}\big(\mathsf{EF}(I_1 \land I_2)\big)$

Examples of CTL formulae: Mutual exclusion

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



Exercise (Mutual exclusion between 2 processes)

1. $\mathsf{AG}\neg(CS_1 \land CS_2)$

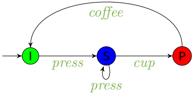
2. $\mathsf{AG}(R_1 \implies \mathsf{AF}\,CS_1)$

3. $\mathsf{AG}\big(\mathsf{EF}(I_1 \land I_2)\big)$

Examples of CTL formulae: Coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

(We reason here with actions instead of state predicates)

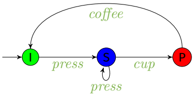


- "At anytime, if the button is pressed, a coffee is always eventually delivered."

Examples of CTL formulae: Coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

(We reason here with actions instead of state predicates)

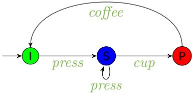


- "At anytime, if the button is pressed, a coffee is always eventually delivered."

Examples of CTL formulae: Coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

(We reason here with actions instead of state predicates)



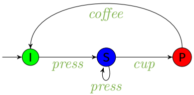
- "At anytime, if the button is pressed, a coffee is always eventually delivered."

- "At anytime, if the button is pressed, there exists an execution such that a coffee is eventually delivered."

Examples of CTL formulae: Coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

(We reason here with actions instead of state predicates)



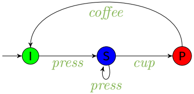
- "At anytime, if the button is pressed, a coffee is always eventually delivered."

- "At anytime, if the button is pressed, there exists an execution such that a coffee is eventually delivered."

Examples of CTL formulae: Coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

(We reason here with actions instead of state predicates)

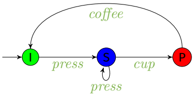


- "At anytime, if the button is pressed, a coffee is always eventually delivered."
- "At anytime, if the button is pressed, there exists an execution such that a coffee is eventually delivered."
- "Once the cup is placed, coffee will necessarily come next."

Examples of CTL formulae: Coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

(We reason here with actions instead of state predicates)

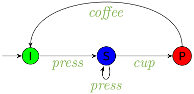


- "At anytime, if the button is pressed, a coffee is always eventually delivered."
- "At anytime, if the button is pressed, there exists an execution such that a coffee is eventually delivered."
- "Once the cup is placed, coffee will necessarily come next."

Examples of CTL formulae: Coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

(We reason here with actions instead of state predicates)

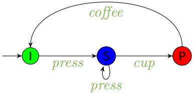


- "At anytime, if the button is pressed, a coffee is always eventually delivered."
- "At anytime, if the button is pressed, there exists an execution such that a coffee is eventually delivered."
- "Once the cup is placed, coffee will necessarily come next."
- "It is possible to get a coffee with exactly 2 doses of sugar."

Examples of CTL formulae: Coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

(We reason here with actions instead of state predicates)

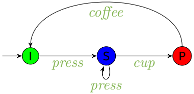


- "At anytime, if the button is pressed, a coffee is always eventually delivered."
- "At anytime, if the button is pressed, there exists an execution such that a coffee is eventually delivered."
- "Once the cup is placed, coffee will necessarily come next."
- "It is possible to get a coffee with exactly 2 doses of sugar."

Examples of CTL formulae: Coffee machine

Express in CTL the following properties, and decide whether they are satisfied for the coffee machine

(We reason here with actions instead of state predicates)



- "At anytime, if the button is pressed, a coffee is always eventually delivered."
- "At anytime, if the button is pressed, there exists an execution such that a coffee is eventually delivered."
- "Once the cup is placed, coffee will necessarily come next."
- "It is possible to get a coffee with exactly 2 doses of sugar."

Exercises: Proofs of equivalence in CTL

Prove that:

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

Exercises: Proofs of equivalence in CTL

Prove that:

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

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

Exercises: Proofs of equivalence in CTL

Prove that:

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

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

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

Exercises: Proofs of equivalence in CTL

Prove that:

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

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

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

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

Exercises: Proofs of equivalence in CTL

Prove that:

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

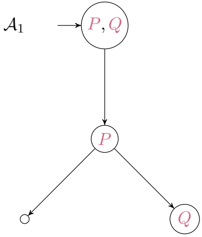
2. $\mathsf{AX}\varphi \equiv \neg\mathsf{EX}\neg\varphi$

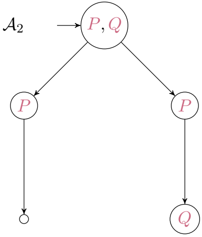
3. $\mathsf{AG}\varphi \equiv \neg(\mathsf{E}\,\mathtt{true}\,\mathsf{U}\neg\varphi)$

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

Outline

LTL and CTL do not recognize the same behaviors





LTL

Runs for both automata:

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

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

CTL

$\mathcal{A}_1 \models \mathsf{AX}(\mathsf{EX}Q \land \mathsf{EX}\neg Q)$

$\mathcal{A}_2 \not\models \mathsf{AX}(\mathsf{EX}Q \land \mathsf{EX}\neg Q)$

LTL or CTL?

1. $(P \mathsf{U} Q) \vee \mathsf{G} P$

LTL or CTL?

1. $(P \mathsf{U} Q) \vee \mathsf{G} P$

LTL or CTL?

1. $(P \mathsf{U} Q) \lor \mathsf{G} P$
2. $\mathsf{GF} P$

LTL or CTL?

1. $(P \mathsf{U} Q) \lor \mathsf{G} P$
2. $\mathsf{GF} P$

LTL or CTL?

1. $(P \mathsf{U} Q) \vee \mathsf{G} P$
2. $\mathsf{GF} P$
3. $\mathsf{AG}(P \implies \mathsf{AF} Q)$

LTL or CTL?

1. $(P \mathsf{U} Q) \lor \mathsf{G} P$
2. $\mathsf{GF} P$
3. $\mathsf{AG}(P \implies \mathsf{AF} Q)$

LTL or CTL?

1. $(P \mathsf{U} Q) \vee \mathsf{G} P$
2. $\mathsf{GF} P$
3. $\mathsf{AG}(P \implies \mathsf{AF} Q)$
4. $\mathsf{A}(\mathsf{GF} P \implies \mathsf{AG} Q)$

LTL or CTL?

1. $(P \mathsf{U} Q) \lor \mathsf{G} P$
2. $\mathsf{GF} P$
3. $\mathsf{AG}(P \implies \mathsf{AF} Q)$
4. $\mathsf{A}(\mathsf{GF} P \implies \mathsf{AG} Q)$

LTL or CTL?

1. $(P \mathsf{U} Q) \lor \mathsf{G} P$
2. $\mathsf{GF} P$
3. $\mathsf{AG}(P \implies \mathsf{AF} Q)$
4. $\mathsf{A}(\mathsf{GF} P \implies \mathsf{AG} Q)$

Outline

Outline

LTL model checking

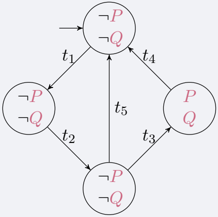
Algorithm working on path formulae

Principle for checking whether $\mathcal{A} \models \varphi$

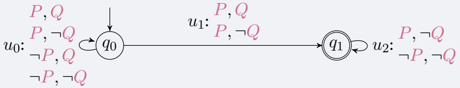
1. Construct the automaton $\mathcal{B}_{\neg\varphi}$ recognizing all executions not satisfying φ
2. Construct the synchronized product $\mathcal{A} \times \mathcal{B}_{\neg\varphi}$
3. If its recognized language is empty, then $\mathcal{A} \models \varphi$

Example

$\mathcal{A}$

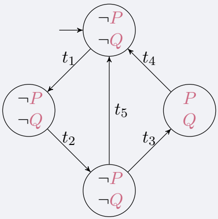


$\mathcal{B}_{\neg\varphi}$ for $\varphi = \mathsf{G}(P \implies \mathsf{XF}Q)$

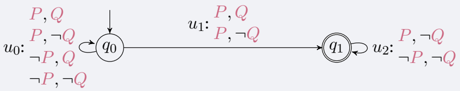


Example

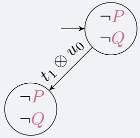
$\mathcal{A}$



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

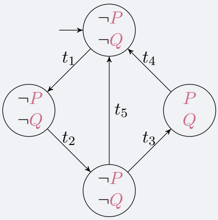


$\mathcal{A} \times \mathcal{B}_{\neg\varphi}$

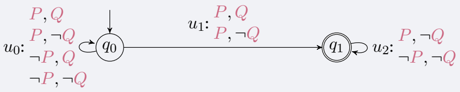


Example

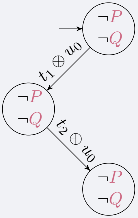
$\mathcal{A}$



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

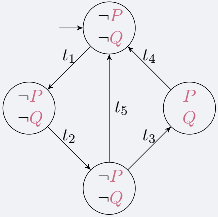


$\mathcal{A} \times \mathcal{B}_{\neg\varphi}$

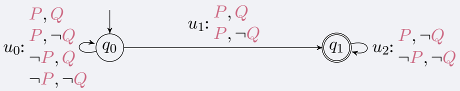


Example

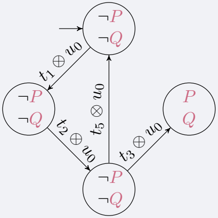
$\mathcal{A}$



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

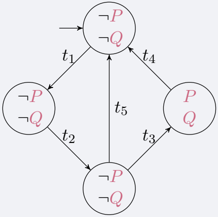


$\mathcal{A} \times \mathcal{B}_{\neg\varphi}$

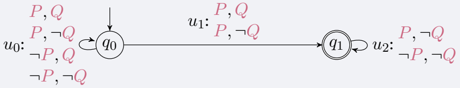


Example

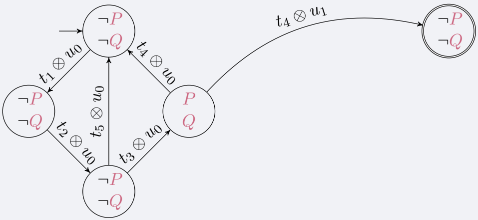
$\mathcal{A}$



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

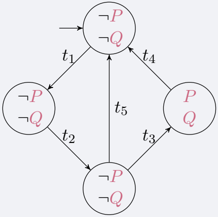


$\mathcal{A} \times \mathcal{B}_{\neg\varphi}$

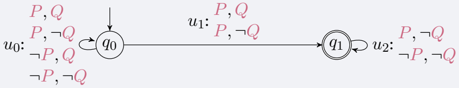


Example

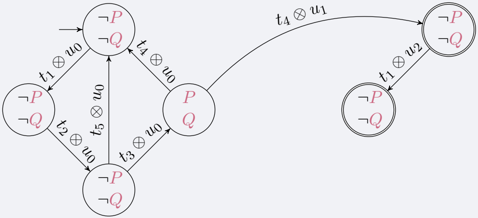
$\mathcal{A}$



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

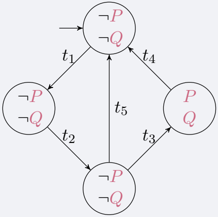


$\mathcal{A} \times \mathcal{B}_{\neg\varphi}$

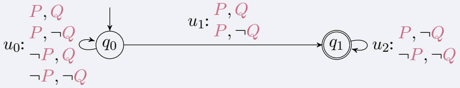


Example

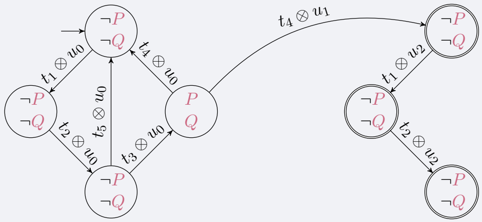
$\mathcal{A}$



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

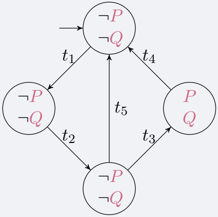


$\mathcal{A} \times \mathcal{B}_{\neg\varphi}$

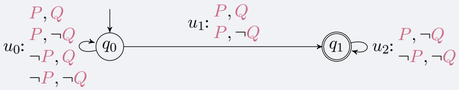


Example

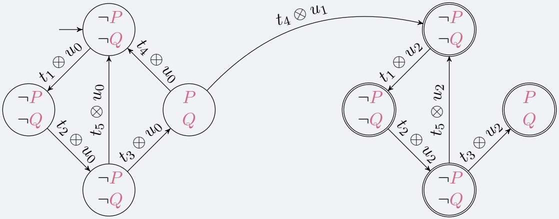
$\mathcal{A}$



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

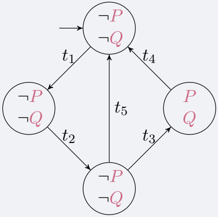


$\mathcal{A} \times \mathcal{B}_{\neg\varphi}$

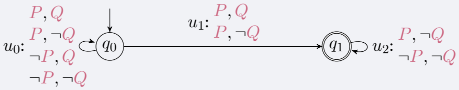


Example

$\mathcal{A}$

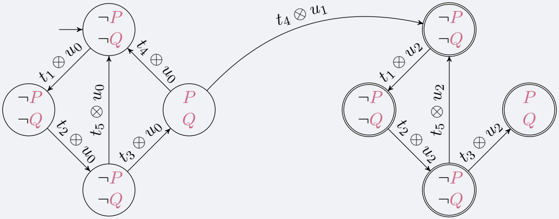


$\mathcal{B}_{\neg\varphi}$ for $\varphi = \mathsf{G}(P \implies \mathsf{XF}Q)$



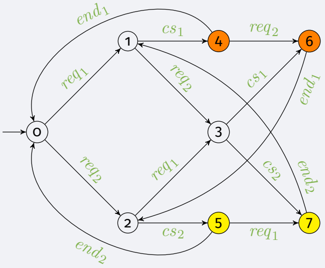
$\mathcal{A} \times \mathcal{B}_{\neg\varphi}$

$\mathcal{A} \not\models \varphi$



Exercise

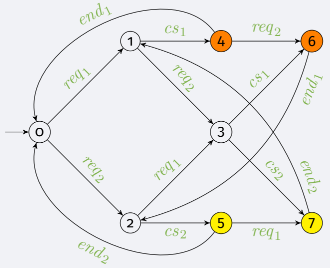
$\mathcal{A}$



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

Exercise

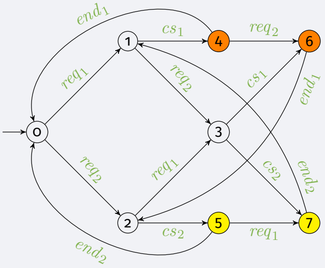
$\mathcal{A}$



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

Exercise

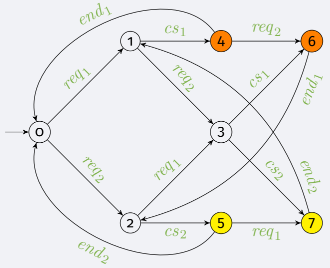
$\mathcal{A}$



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

Exercise

$\mathcal{A}$

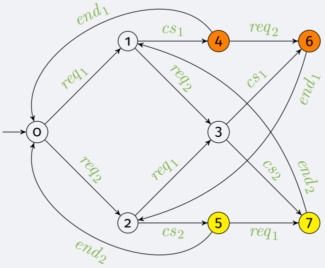


$\mathcal{B}_{\neg\varphi}$ for $\varphi = \mathsf{G}\neg(cs_1 \wedge cs_2)$

$\mathcal{A} \times \mathcal{B}_{\neg\varphi}$

Exercise

$\mathcal{A}$



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

$\mathcal{A} \times \mathcal{B}_{\neg\varphi}$

Outline

CTL model checking algorithm

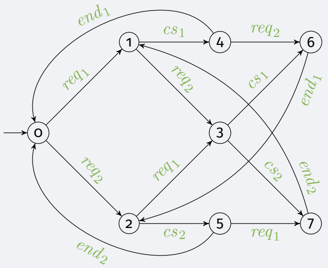
- Algorithm $markPred$ decides where a formula is satisfied
- Memorizes the already computed results
- Reuses the computed results of sub-formulae to compute new formulae

CTL model checking algorithm

Case 1: $\varphi = p$ (base case)

case $\varphi = p$ **do**
  forall $q \in Q$ **do**
    if $p \in lab(q)$ **then**
      $q.\varphi \leftarrow$ `true`
    else
      $q.\varphi \leftarrow$ `false`

$\varphi = R_1$

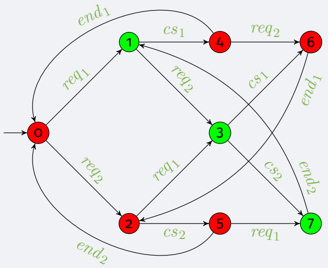


CTL model checking algorithm

Case 1: $\varphi = p$ (base case)

case $\varphi = p$ **do**
  forall $q \in Q$ **do**
    if $p \in lab(q)$ **then**
      $q.\varphi \leftarrow$ `true`
    else
      $q.\varphi \leftarrow$ `false`

$\varphi = R_1$

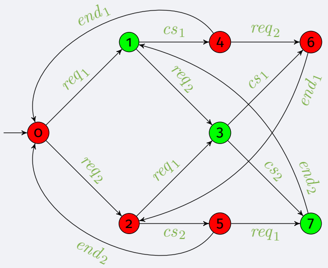


CTL model checking algorithm

Case 1: $\varphi = p$ (base case)

case $\varphi = p$ **do**
- **forall** $q \in Q$ **do**
  - **if** $p \in lab(q)$ **then**
    - $q.\varphi \leftarrow$ `true`
  - **else**
    - $q.\varphi \leftarrow$ `false`

$\varphi = R_1$

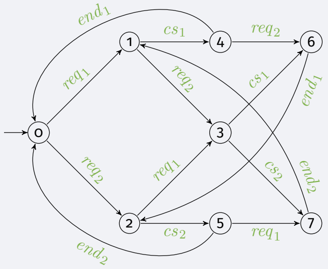


The formula is false on the initial state

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

$markPred(\psi)$
forall $q \in Q$ **do**
$\quad q.\varphi \leftarrow \neg q.\psi$

$\varphi = \neg R_1$



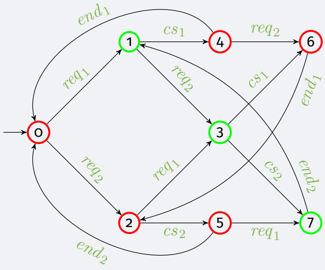
Case 2: $\varphi = \neg\psi$

$markPred(\psi)$

forall $q \in Q$ **do**

$\quad q.\varphi \leftarrow \neg q.\psi$

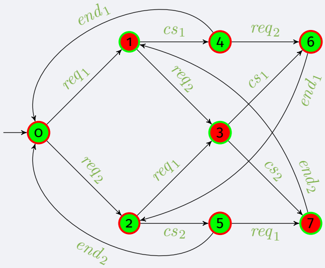
$\varphi = \neg R_1$



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

$markPred(\psi)$
forall $q \in Q$ **do**
$\quad q.\varphi \leftarrow \neg q.\psi$

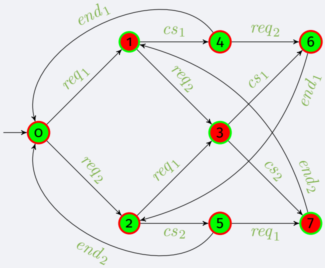
$\varphi = \neg R_1$



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

$markPred(\psi)$
forall $q \in Q$ **do**
$\quad q.\varphi \leftarrow \neg q.\psi$

$\varphi = \neg R_1$

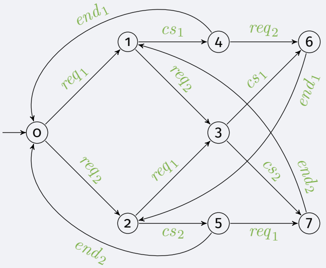


The formula is true on the initial state

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

$markPred(\psi_1)$
$markPred(\psi_2)$
forall $q \in Q$ **do**
  $q.\varphi \leftarrow q.\psi_1 \wedge q.\psi_2$

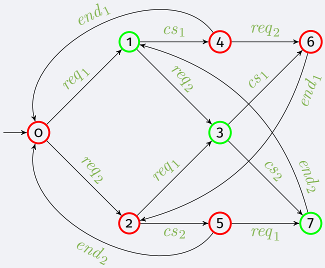
$\varphi = R_1 \wedge R_2$



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

$markPred(\psi_1)$
$markPred(\psi_2)$
forall $q \in Q$ **do**
$\quad q.\varphi \leftarrow q.\psi_1 \land q.\psi_2$

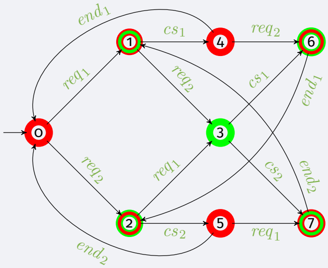
$\varphi = R_1 \land R_2$



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

$markPred(\psi_1)$
$markPred(\psi_2)$
forall $q \in Q$ **do**
  $q.\varphi \leftarrow q.\psi_1 \land q.\psi_2$

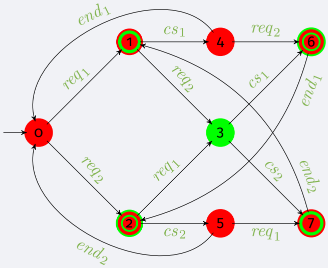
$\varphi = R_1 \land R_2$



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

$markPred(\psi_1)$
$markPred(\psi_2)$
forall $q \in Q$ **do**
$\quad q.\varphi \leftarrow q.\psi_1 \wedge q.\psi_2$

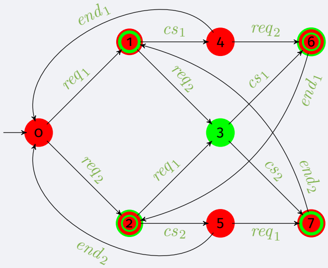
$\varphi = R_1 \wedge R_2$



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

$markPred(\psi_1)$
$markPred(\psi_2)$
forall $q \in Q$ **do**
  $q.\varphi \leftarrow q.\psi_1 \wedge q.\psi_2$

$\varphi = R_1 \wedge R_2$



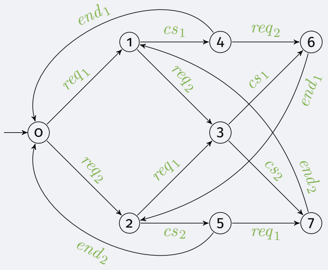
The formula is false on the initial state

Case 4: $\varphi = \mathsf{EX}\psi$

```
/* Init */
markPred(ψ)
forall q ∈ Q do
  q.φ ← false

/* Main loop */
forall (q, _, q') ∈ T do
  if q'.ψ = true then
    q.φ ← true
```

$\varphi = \mathsf{EX} R_1$



Case 4: $\varphi = \mathsf{EX}\psi$

```
/* Init */
```
$markPred(\psi)$

forall $q \in Q$ **do**

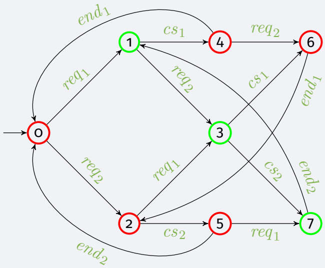
  $q.\varphi \leftarrow$ `false`

```
/* Main loop */
```
forall $(q, \_, q') \in T$ **do**

  if $q'.\psi =$ `true` **then**

    $q.\varphi \leftarrow$ `true`

$\varphi = \mathsf{EX} R_1$



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

/* Init */

$markPred(\psi)$

forall $q \in Q$ **do**

  $q.\varphi \leftarrow$ false

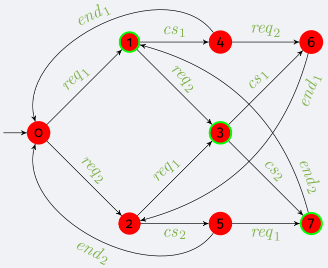
/* Main loop */

forall $(q, \_, q') \in T$ **do**

  if $q'.\psi =$ true **then**

    $q.\varphi \leftarrow$ true

$\varphi = \text{EX}R_1$



Case 4: $\varphi = \mathsf{EX}\psi$

/* Init */

$markPred(\psi)$

forall $q \in Q$ **do**

$\quad q.\varphi \leftarrow$ false

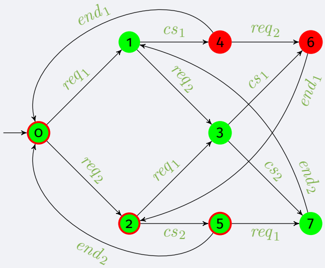
/* Main loop */

forall $(q, \_, q') \in T$ **do**

$\quad$ **if** $q'.\psi =$ true **then**

$\quad\quad q.\varphi \leftarrow$ true

$\varphi = \mathsf{EX} R_1$

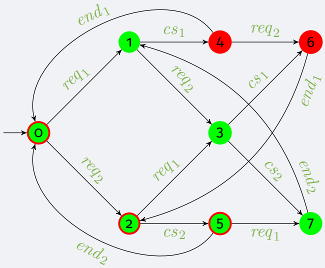


Case 4: $\varphi = \mathsf{EX}\psi$

```
/* Init */
```
$markPred(\psi)$
forall $q \in Q$ **do**
  $q.\varphi \leftarrow$ `false`

```
/* Main loop */
```
forall $(q, \_, q') \in T$ **do**
  if $q'.\psi =$ `true` **then**
    $q.\varphi \leftarrow$ `true`

$\varphi = \mathsf{EX} R_1$



The formula is true on the initial state

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

/* Init */

$markPred(\psi_1)$

$markPred(\psi_2)$

forall $q \in Q$ **do**

  $q.\varphi \leftarrow$ `false`

  $q.seenbefore \leftarrow$ `false`

$L \leftarrow \emptyset$

forall $q \in Q$ **do**

  if $q.\psi_2 =$ `true` **then**

    $L \leftarrow L \cup \{q\}$

/* Main loop */

while $L \neq \emptyset$ **do**

  pick q from L; $L \leftarrow L \setminus \{q\}$

  $q.\varphi \leftarrow$ `true`

  forall $(q', \_, q) \in T$ **do**

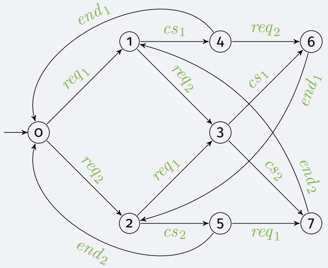
    if $q'.seenbefore =$ `false` **then**

      $q'.seenbefore \leftarrow$ `true`

      if $q'.\psi_1 =$ `true` **then**

        $L \leftarrow L \cup \{q'\}$

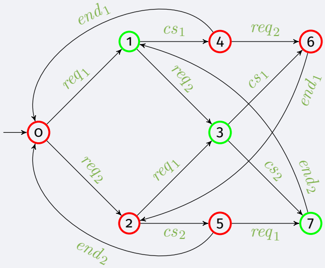
$\varphi = \mathsf{E}R_1\mathsf{U}CS_1$



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

```
/* Init */
markPred(ψ1)
markPred(ψ2)
forall q ∈ Q do
    q.φ ← false
    q.seenbefore ← false
L ← ∅
forall q ∈ Q do
    if q.ψ2 = true then
        L ← L ∪ {q}
/* Main loop */
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'}
```

$\varphi = \mathsf{E}R_1\mathsf{U}CS_1$



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

/* Init */

$markPred(\psi_1)$

$markPred(\psi_2)$

forall $q \in Q$ **do**

  $q.\varphi \leftarrow$ false

  $q.seenbefore \leftarrow$ false

$L \leftarrow \emptyset$

forall $q \in Q$ **do**

  if $q.\psi_2 =$ true **then**

    $L \leftarrow L \cup \{q\}$

/* Main loop */

while $L \neq \emptyset$ **do**

  pick q from L; $L \leftarrow L \setminus \{q\}$

  $q.\varphi \leftarrow$ true

  forall $(q', \_, q) \in T$ **do**

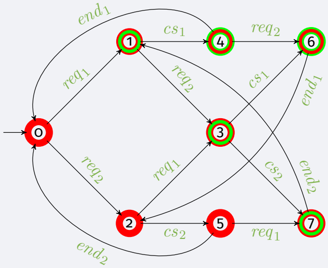
    if $q'.seenbefore =$ false **then**

      $q'.seenbefore \leftarrow$ true

      if $q'.\psi_1 =$ true **then**

        $L \leftarrow L \cup \{q'\}$

$\varphi = \mathsf{E} R_1 \mathsf{U} CS_1$

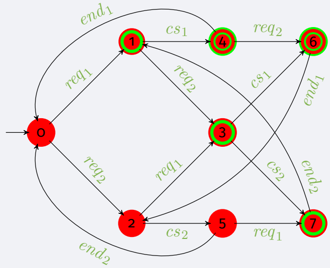


Case 5: $\varphi = \mathsf{E}\psi_1\mathsf{U}\psi_2$

```
/* Init */
markPred(ψ1)
markPred(ψ2)
forall q ∈ Q do
    q.φ ← false
    q.seenbefore ← false
L ← ∅
forall q ∈ Q do
    if q.ψ2 = true then
        L ← L ∪ {q}

/* Main loop */
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'}
```

$\varphi = \mathsf{E} R_1 \mathsf{U} CS_1$

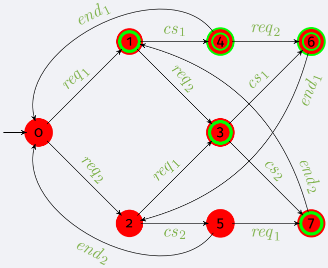


Case 5: $\varphi = \mathsf{E}\psi_1 \mathsf{U} \psi_2$

```
/* Init */
markPred(ψ1)
markPred(ψ2)
forall q ∈ Q do
    q.φ ← false
    q.seenbefore ← false
L ← ∅
forall q ∈ Q do
    if q.ψ2 = true then
        L ← L ∪ {q}

/* Main loop */
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′}
```

$\varphi = \mathsf{E} R_1 \mathsf{U}\, CS_1$

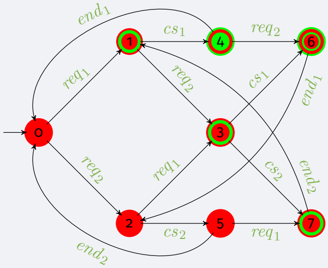


Case 5: $\varphi = \mathsf{E}\psi_1\mathsf{U}\psi_2$

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
forall $q \in Q$ **do**
  $q.\varphi \leftarrow$ false
  $q.seenbefore \leftarrow$ false
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  if $q.\psi_2 =$ true **then**
    $L \leftarrow L \cup \{q\}$

/* Main loop */
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ true
  forall $(q', \_, q) \in T$ **do**
    if $q'.seenbefore =$ false **then**
      $q'.seenbefore \leftarrow$ true
      if $q'.\psi_1 =$ true **then**
        $L \leftarrow L \cup \{q'\}$

$\varphi = \mathsf{E}R_1\mathsf{U}CS_1$

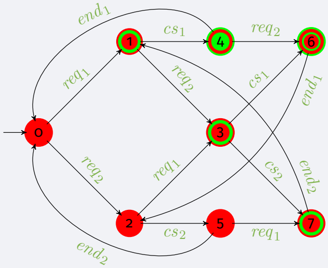


Case 5: $\varphi = \mathsf{E}\psi_1\mathsf{U}\psi_2$

```
/* Init */
```
$markPred(\psi_1)$
$markPred(\psi_2)$
forall $q \in Q$ **do**
  $q.\varphi \leftarrow$ `false`
  $q.seenbefore \leftarrow$ `false`
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  if $q.\psi_2 =$ `true` **then**
    $L \leftarrow L \cup \{q\}$

```
/* Main loop */
```
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ `true`
  forall $(q', \_, q) \in T$ **do**
    if $q'.seenbefore =$ `false` **then**
      $q'.seenbefore \leftarrow$ `true`
      if $q'.\psi_1 =$ `true` **then**
        $L \leftarrow L \cup \{q'\}$

$\varphi = \mathsf{E}R_1\mathsf{U}CS_1$



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

/* Init */

$markPred(\psi_1)$

$markPred(\psi_2)$

forall $q \in Q$ **do**

- $q.\varphi \leftarrow$ `false`
- $q.seenbefore \leftarrow$ `false`

$L \leftarrow \emptyset$

forall $q \in Q$ **do**

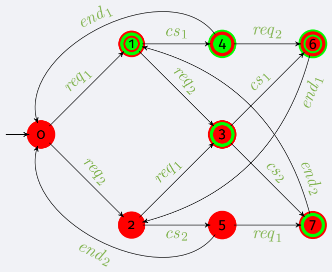
- **if** $q.\psi_2 =$ `true` **then**
  - $L \leftarrow L \cup \{q\}$

/* Main loop */

while $L \neq \emptyset$ **do**

- pick q from L; $L \leftarrow L \setminus \{q\}$
- $q.\varphi \leftarrow$ `true`
- **forall** $(q', \_, q) \in T$ **do**
  - **if** $q'.seenbefore =$ `false` **then**
    - $q'.seenbefore \leftarrow$ `true`
    - **if** $q'.\psi_1 =$ `true` **then**
      - $L \leftarrow L \cup \{q'\}$

$\varphi = \mathsf{E} R_1 \mathsf{U} CS_1$



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

/* Init */

$markPred(\psi_1)$

$markPred(\psi_2)$

forall $q \in Q$ **do**

- $q.\varphi \leftarrow$ `false`
- $q.seenbefore \leftarrow$ `false`

$L \leftarrow \emptyset$

forall $q \in Q$ **do**

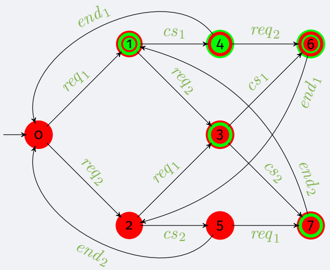
- **if** $q.\psi_2 =$ `true` **then**
  - $L \leftarrow L \cup \{q\}$

/* Main loop */

while $L \neq \emptyset$ **do**

- pick q from L; $L \leftarrow L \setminus \{q\}$
- $q.\varphi \leftarrow$ `true`
- **forall** $(q', \_, q) \in T$ **do**
  - **if** $q'.seenbefore =$ `false` **then**
    - $q'.seenbefore \leftarrow$ `true`
    - **if** $q'.\psi_1 =$ `true` **then**
      - $L \leftarrow L \cup \{q'\}$

$\varphi = \mathsf{E}R_1\mathsf{U}CS_1$



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

/* Init */

$markPred(\psi_1)$

$markPred(\psi_2)$

forall $q \in Q$ **do**

  $q.\varphi \leftarrow$ false

  $q.seenbefore \leftarrow$ false

$L \leftarrow \emptyset$

forall $q \in Q$ **do**

  if $q.\psi_2 =$ true **then**

    $L \leftarrow L \cup \{q\}$

/* Main loop */

while $L \neq \emptyset$ **do**

  pick q from L; $L \leftarrow L \setminus \{q\}$

  $q.\varphi \leftarrow$ true

  forall $(q', \_, q) \in T$ **do**

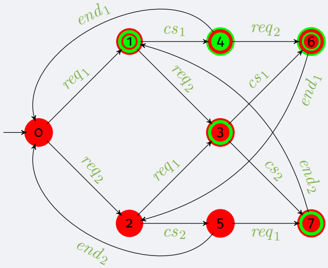
    if $q'.seenbefore =$ false **then**

      $q'.seenbefore \leftarrow$ true

      if $q'.\psi_1 =$ true **then**

        $L \leftarrow L \cup \{q'\}$

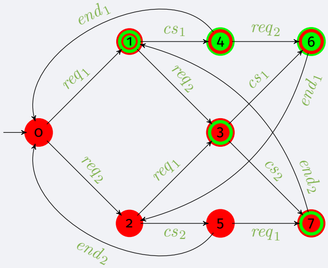
$\varphi = \mathsf{E} R_1 \mathsf{U} CS_1$



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

```
/* Init */
markPred(ψ1)
markPred(ψ2)
forall q ∈ Q do
    q.φ ← false
    q.seenbefore ← false
L ← ∅
forall q ∈ Q do
    if q.ψ2 = true then
        L ← L ∪ {q}
/* Main loop */
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'}
```

$\varphi = \mathsf{E}R_1\mathsf{U}CS_1$

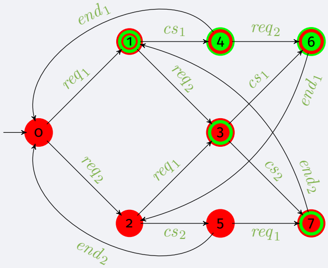


Case 5: $\varphi = \mathsf{E}\psi_1 \mathsf{U} \psi_2$

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
forall $q \in Q$ **do**
  $q.\varphi \leftarrow$ false
  $q.seenbefore \leftarrow$ false
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  if $q.\psi_2 =$ true **then**
    $L \leftarrow L \cup \{q\}$

/* Main loop */
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ true
  forall $(q', \_, q) \in T$ **do**
    if $q'.seenbefore =$ false **then**
      $q'.seenbefore \leftarrow$ true
      if $q'.\psi_1 =$ true **then**
        $L \leftarrow L \cup \{q'\}$

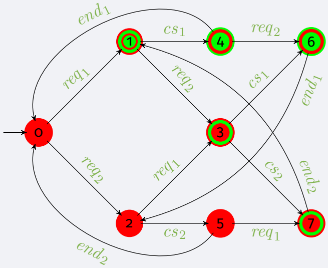
$\varphi = \mathsf{E} R_1 \mathsf{U} CS_1$



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

```
/* Init */
markPred(ψ1)
markPred(ψ2)
forall q ∈ Q do
    q.φ ← false
    q.seenbefore ← false
L ← ∅
forall q ∈ Q do
    if q.ψ2 = true then
        L ← L ∪ {q}
/* Main loop */
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'}
```

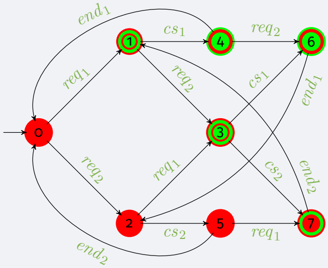
$\varphi = \mathsf{E}R_1\mathsf{U}CS_1$



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

```
/* Init */
markPred(ψ1)
markPred(ψ2)
forall q ∈ Q do
    q.φ ← false
    q.seenbefore ← false
L ← ∅
forall q ∈ Q do
    if q.ψ2 = true then
        L ← L ∪ {q}
/* Main loop */
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'}
```

$\varphi = \mathsf{E} R_1 \mathsf{U} CS_1$



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

/* Init */

$markPred(\psi_1)$

$markPred(\psi_2)$

forall $q \in Q$ **do**

  $q.\varphi \leftarrow$ `false`

  $q.seenbefore \leftarrow$ `false`

$L \leftarrow \emptyset$

forall $q \in Q$ **do**

  if $q.\psi_2 =$ `true` **then**

    $L \leftarrow L \cup \{q\}$

/* Main loop */

while $L \neq \emptyset$ **do**

  pick q from L; $L \leftarrow L \setminus \{q\}$

  $q.\varphi \leftarrow$ `true`

  forall $(q', \_, q) \in T$ **do**

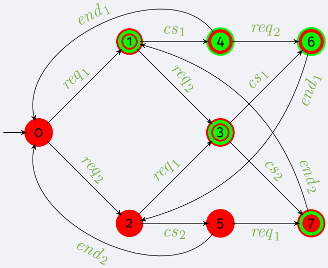
    if $q'.seenbefore =$ `false` **then**

      $q'.seenbefore \leftarrow$ `true`

      if $q'.\psi_1 =$ `true` **then**

        $L \leftarrow L \cup \{q'\}$

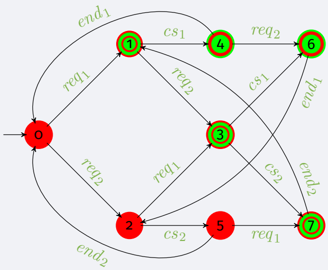
$\varphi = \mathsf{E}R_1\mathsf{U}CS_1$



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

```
/* Init */
markPred(ψ1)
markPred(ψ2)
forall q ∈ Q do
    q.φ ← false
    q.seenbefore ← false
L ← ∅
forall q ∈ Q do
    if q.ψ2 = true then
        L ← L ∪ {q}
/* Main loop */
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'}
```

$\varphi = \mathsf{E}R_1\mathsf{U}CS_1$

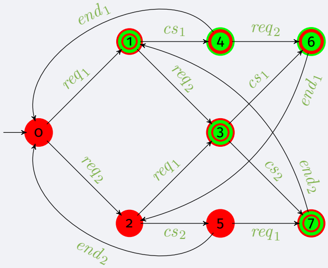


Case 5: $\varphi = \mathsf{E}\psi_1\mathsf{U}\psi_2$

```
/* Init */
markPred(ψ1)
markPred(ψ2)
forall q ∈ Q do
    q.φ ← false
    q.seenbefore ← false
L ← ∅
forall q ∈ Q do
    if q.ψ2 = true then
        L ← L ∪ {q}

/* Main loop */
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′}
```

$\varphi = \mathsf{E}R_1\mathsf{U}CS_1$

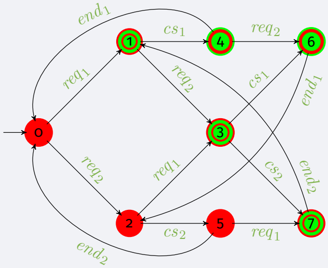


Case 5: $\varphi = \mathsf{E}\psi_1 \mathsf{U} \psi_2$

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
forall $q \in Q$ **do**
  $q.\varphi \leftarrow$ false
  $q.seenbefore \leftarrow$ false
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  if $q.\psi_2 =$ true **then**
    $L \leftarrow L \cup \{q\}$

/* Main loop */
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ true
  forall $(q', \_, q) \in T$ **do**
    if $q'.seenbefore =$ false **then**
      $q'.seenbefore \leftarrow$ true
      if $q'.\psi_1 =$ true **then**
        $L \leftarrow L \cup \{q'\}$

$\varphi = \mathsf{E} R_1 \mathsf{U} CS_1$

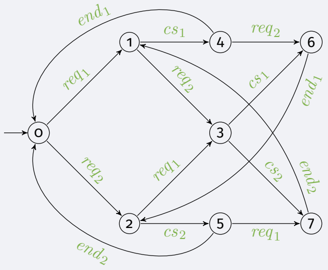


The formula is false on the initial state

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

```
/* Init */
markPred(ψ1)
markPred(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}
/* Main loop */
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 ∧ q'.ψ1 =
          true ∧ q'.φ = false then
            L ← L ∪ {q'}
```

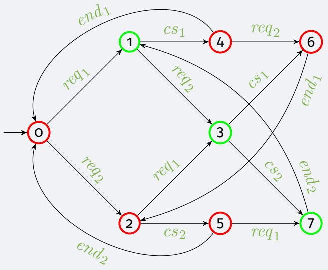
$\varphi = \mathsf{A}R_1\mathsf{U}CS_1$



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

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  $q.nb \leftarrow degree(q)$
  $q.\varphi \leftarrow$ false
forall $q \in Q$ **do**
  if $q.\psi_2 =$ true **then**
    $L \leftarrow L \cup \{q\}$
/* Main loop */
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ true
  forall $(q', \_, q) \in T$ **do**
    $q'.nb \leftarrow q'.nb - 1$
    if $q'.nb = 0 \wedge q'.\psi_1 =$ true $\wedge q'.\varphi =$ false **then**
      $L \leftarrow L \cup \{q'\}$

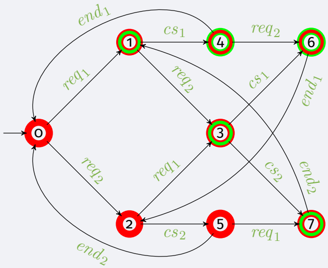
$\varphi = \mathsf{A} R_1 \mathsf{U} CS_1$



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

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  $q.nb \leftarrow degree(q)$
  $q.\varphi \leftarrow$ false
forall $q \in Q$ **do**
  if $q.\psi_2 =$ true **then**
    $L \leftarrow L \cup \{q\}$
/* Main loop */
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ true
  forall $(q', \_, q) \in T$ **do**
    $q'.nb \leftarrow q'.nb - 1$
    if $q'.nb = 0 \land q'.\psi_1 =$ true $\land q'.\varphi =$ false **then**
      $L \leftarrow L \cup \{q'\}$

$\varphi = \mathsf{A} R_1 \mathsf{U}\, CS_1$



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

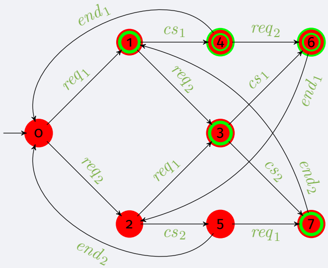
```
/* Init */
```
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$

forall $q \in Q$ **do**
- $q.nb \leftarrow degree(q)$
- $q.\varphi \leftarrow$ `false`

forall $q \in Q$ **do**
- **if** $q.\psi_2 =$ `true` **then**
  - $L \leftarrow L \cup \{q\}$

```
/* Main loop */
```
while $L \neq \emptyset$ **do**
- pick q from L; $L \leftarrow L \setminus \{q\}$
- $q.\varphi \leftarrow$ `true`
- **forall** $(q', \_, q) \in T$ **do**
  - $q'.nb \leftarrow q'.nb - 1$
  - **if** $q'.nb = 0 \wedge q'.\psi_1 =$ `true` $\wedge\, q'.\varphi =$ `false` **then**
    - $L \leftarrow L \cup \{q'\}$

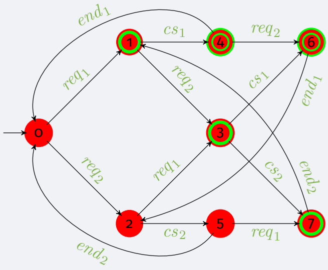
$\varphi = \mathsf{A}R_1\mathsf{U}CS_1$



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

```
/* Init */
markPred(ψ1)
markPred(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}
/* Main loop */
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 ∧ q′.ψ1 =
          true ∧ q′.φ = false then
            L ← L ∪ {q′}
```

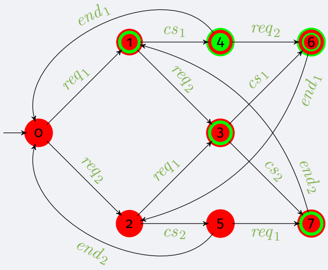
$\varphi = \mathsf{A} R_1 \mathsf{U}\, CS_1$



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

```
/* Init */
markPred(ψ1)
markPred(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}
/* Main loop */
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 ∧ q′.ψ1 =
          true ∧ q′.φ = false then
            L ← L ∪ {q′}
```

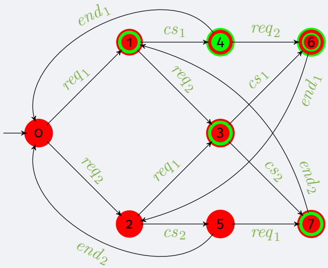
$\varphi = \mathsf{A}R_1\mathsf{U}CS_1$



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

```
/* Init */
markPred(ψ1)
markPred(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}
/* Main loop */
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 ∧ q'.ψ1 =
          true ∧ q'.φ = false then
            L ← L ∪ {q'}
```

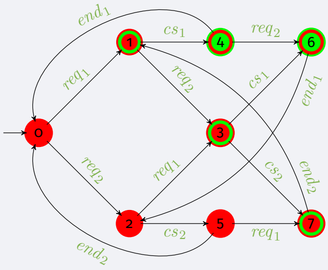
$\varphi = \mathsf{A} R_1 \mathsf{U}\, CS_1$



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

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
    $q.nb \leftarrow degree(q)$
    $q.\varphi \leftarrow$ false
forall $q \in Q$ **do**
    if $q.\psi_2 =$ true **then**
        $L \leftarrow L \cup \{q\}$
/* Main loop */
while $L \neq \emptyset$ **do**
    pick q from L; $L \leftarrow L \setminus \{q\}$
    $q.\varphi \leftarrow$ true
    forall $(q', \_, q) \in T$ **do**
        $q'.nb \leftarrow q'.nb - 1$
        if $q'.nb = 0 \wedge q'.\psi_1 =$ true $\wedge q'.\varphi =$ false **then**
            $L \leftarrow L \cup \{q'\}$

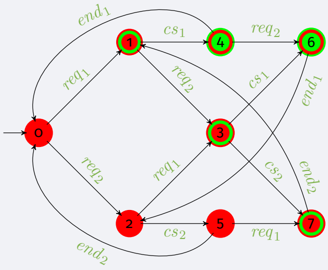
$\varphi = \mathsf{A}R_1\mathsf{U}CS_1$



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

```
/* Init */
markPred(ψ1)
markPred(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}
/* Main loop */
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 ∧ q′.ψ1 =
          true ∧ q′.φ = false then
            L ← L ∪ {q′}
```

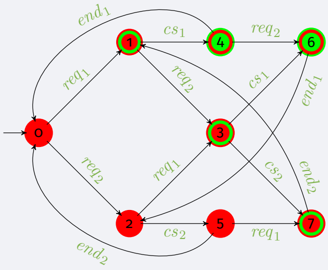
$\varphi = \mathsf{A} R_1 \mathsf{U}\, CS_1$



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

```
/* Init */
markPred(ψ1)
markPred(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}
/* Main loop */
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 ∧ q'.ψ1 =
          true ∧ q'.φ = false then
            L ← L ∪ {q'}
```

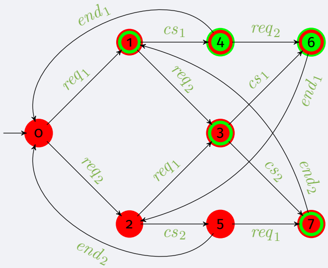
$\varphi = \mathsf{A} R_1 \mathsf{U}\, CS_1$



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

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  $q.nb \leftarrow degree(q)$
  $q.\varphi \leftarrow$ false
forall $q \in Q$ **do**
  if $q.\psi_2 =$ true **then**
    $L \leftarrow L \cup \{q\}$
/* Main loop */
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ true
  forall $(q', \_, q) \in T$ **do**
    $q'.nb \leftarrow q'.nb - 1$
    if $q'.nb = 0 \wedge q'.\psi_1 =$ true $\wedge q'.\varphi =$ false **then**
      $L \leftarrow L \cup \{q'\}$

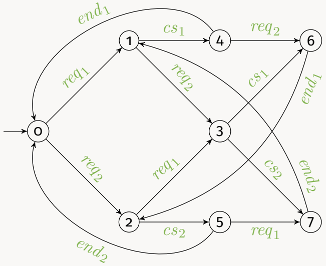
$\varphi = \mathsf{A}R_1\mathsf{U}CS_1$



The formula is false on the initial state

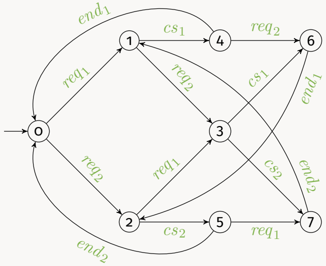
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



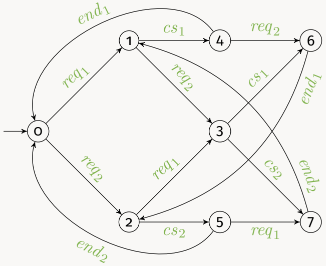
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



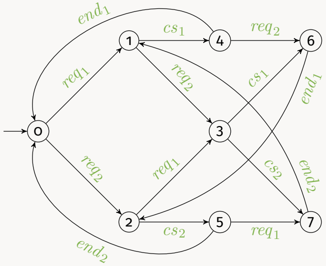
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



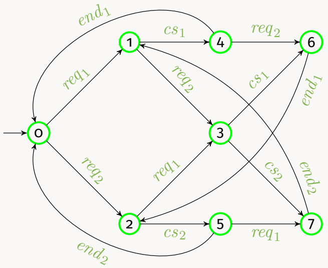
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \land I_2)\big)$



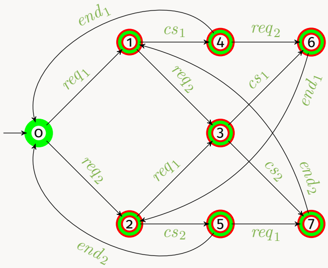
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



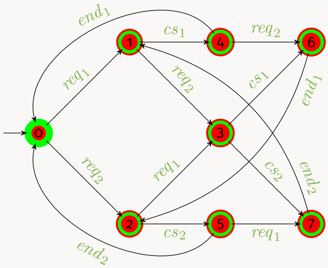
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



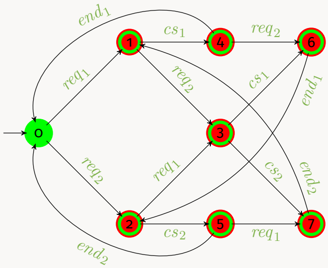
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



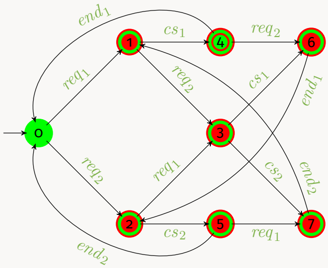
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



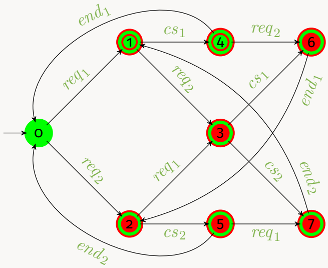
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



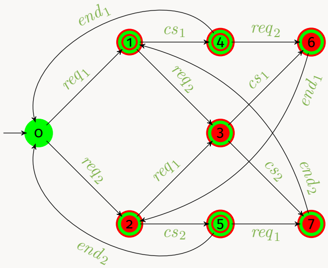
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



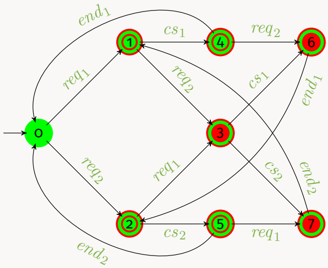
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



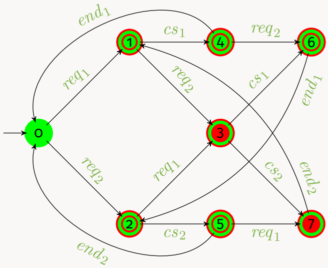
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



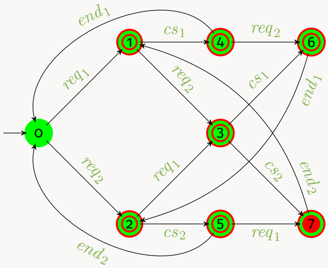
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



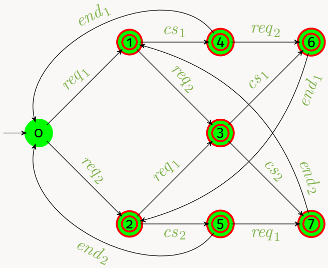
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



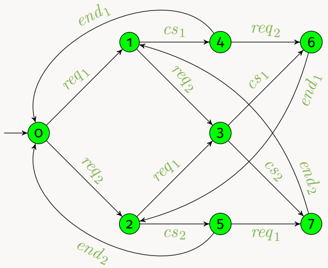
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



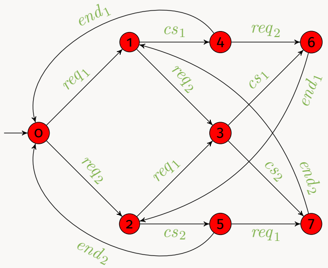
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



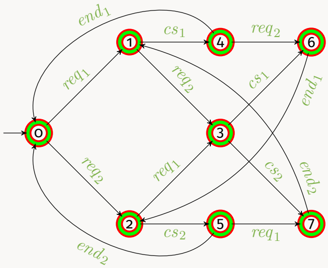
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



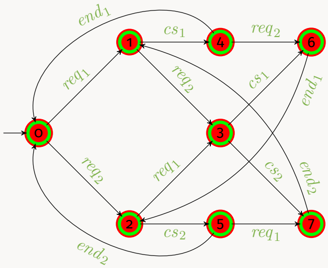
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



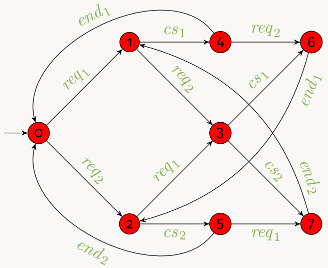
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \land I_2)\big)$



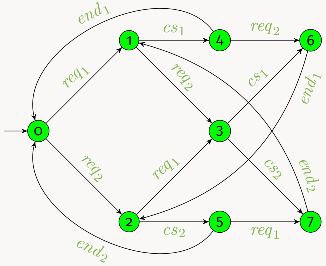
Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \wedge I_2)\big)$



Exercise: Check a CTL formula (mutual exclusion)

Check $\mathsf{AG}\big(\mathsf{EF}(I_1 \land I_2)\big)$

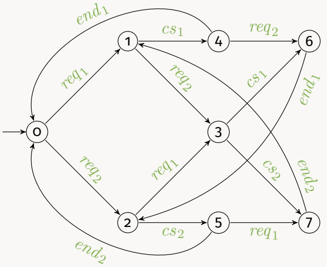


The formula is true

Exercise: Check another CTL formula (mutual exclusion)

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
    $q.nb \leftarrow degree(q)$
    $q.\varphi \leftarrow$ false
forall $q \in Q$ **do**
    if $q.\psi_2 =$ true **then**
        $L \leftarrow L \cup \{q\}$
/* Main loop */
while $L \neq \emptyset$ **do**
    pick q from L; $L \leftarrow L \setminus \{q\}$
    $q.\varphi \leftarrow$ true
    forall $(q', \_, q) \in T$ **do**
        $q'.nb \leftarrow q'.nb - 1$
        if $q'.nb = 0 \wedge q'.\psi_1 =$ true $\wedge\, q'.\varphi =$ false **then**
            $L \leftarrow L \cup \{q'\}$

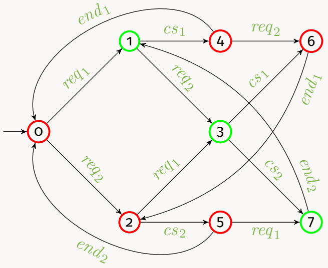
$\varphi = \mathsf{A} R_1 \mathsf{U} CS_2$



Exercise: Check another CTL formula (mutual exclusion)

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  $q.nb \leftarrow degree(q)$
  $q.\varphi \leftarrow$ false
forall $q \in Q$ **do**
  if $q.\psi_2 =$ true **then**
    $L \leftarrow L \cup \{q\}$
/* Main loop */
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ true
  forall $(q', \_, q) \in T$ **do**
    $q'.nb \leftarrow q'.nb - 1$
    if $q'.nb = 0 \land q'.\psi_1 =$ true $\land q'.\varphi =$ false **then**
      $L \leftarrow L \cup \{q'\}$

$\varphi = \mathsf{A} R_1 \mathsf{U} CS_2$



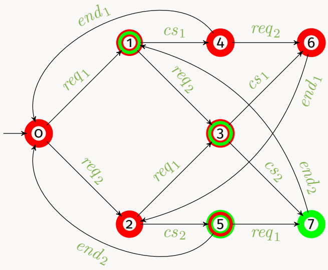
Exercise: Check another CTL formula (mutual exclusion)

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  $q.nb \leftarrow degree(q)$
  $q.\varphi \leftarrow$ false

forall $q \in Q$ **do**
  if $q.\psi_2 =$ true **then**
    $L \leftarrow L \cup \{q\}$

/* Main loop */
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ true
  forall $(q', \_, q) \in T$ **do**
    $q'.nb \leftarrow q'.nb - 1$
    if $q'.nb = 0 \wedge q'.\psi_1 =$ true $\wedge q'.\varphi =$ false **then**
      $L \leftarrow L \cup \{q'\}$

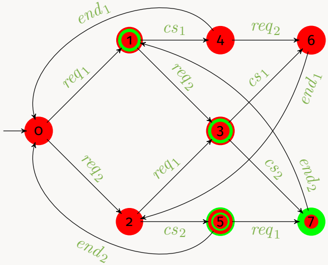
$\varphi = \mathsf{A} R_1 \mathsf{U} CS_2$



Exercise: Check another CTL formula (mutual exclusion)

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
    $q.nb \leftarrow degree(q)$
    $q.\varphi \leftarrow$ false
forall $q \in Q$ **do**
    if $q.\psi_2 =$ true **then**
        $L \leftarrow L \cup \{q\}$
/* Main loop */
while $L \neq \emptyset$ **do**
    pick q from L; $L \leftarrow L \setminus \{q\}$
    $q.\varphi \leftarrow$ true
    forall $(q', \_, q) \in T$ **do**
        $q'.nb \leftarrow q'.nb - 1$
        if $q'.nb = 0 \wedge q'.\psi_1 =$ true $\wedge q'.\varphi =$ false **then**
            $L \leftarrow L \cup \{q'\}$

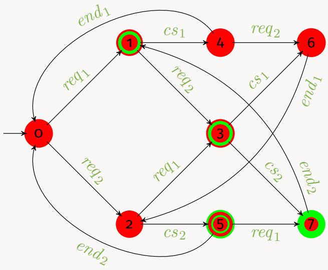
$\varphi = \mathsf{A} R_1 \mathsf{U} CS_2$



Exercise: Check another CTL formula (mutual exclusion)

```
/* Init */
markPred(ψ1)
markPred(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}
/* Main loop */
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 ∧ q′.ψ1 =
          true ∧ q′.φ = false then
            L ← L ∪ {q′}
```

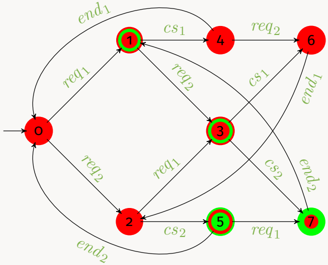
$\varphi = \mathsf{A} R_1 \mathsf{U} CS_2$



Exercise: Check another CTL formula (mutual exclusion)

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
    $q.nb \leftarrow degree(q)$
    $q.\varphi \leftarrow$ `false`
forall $q \in Q$ **do**
    if $q.\psi_2 =$ `true` **then**
        $L \leftarrow L \cup \{q\}$
/* Main loop */
while $L \neq \emptyset$ **do**
    pick q from L; $L \leftarrow L \setminus \{q\}$
    $q.\varphi \leftarrow$ `true`
    forall $(q', \_, q) \in T$ **do**
        $q'.nb \leftarrow q'.nb - 1$
        if $q'.nb = 0 \land q'.\psi_1 =$ `true` $\land q'.\varphi =$ `false` **then**
            $L \leftarrow L \cup \{q'\}$

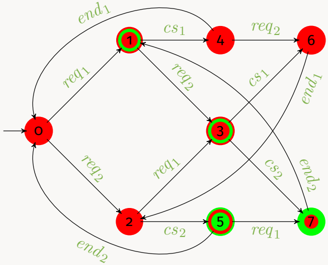
$\varphi = \mathsf{A} R_1 \mathsf{U} CS_2$



Exercise: Check another CTL formula (mutual exclusion)

```
/* Init */
```
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  $q.nb \leftarrow degree(q)$
  $q.\varphi \leftarrow$ `false`
forall $q \in Q$ **do**
  if $q.\psi_2 =$ `true` **then**
    $L \leftarrow L \cup \{q\}$
```
/* Main loop */
```
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ `true`
  forall $(q', \_, q) \in T$ **do**
    $q'.nb \leftarrow q'.nb - 1$
    if $q'.nb = 0 \wedge q'.\psi_1 =$ `true` $\wedge q'.\varphi =$ `false` **then**
      $L \leftarrow L \cup \{q'\}$

$\varphi = \mathsf{A} R_1 \mathsf{U} CS_2$



Exercise: Check another CTL formula (mutual exclusion)

/* Init */

$markPred(\psi_1)$

$markPred(\psi_2)$

$L \leftarrow \emptyset$

forall $q \in Q$ **do**

  $q.nb \leftarrow degree(q)$

  $q.\varphi \leftarrow$ false

forall $q \in Q$ **do**

  if $q.\psi_2 =$ true **then**

    $L \leftarrow L \cup \{q\}$

/* Main loop */

while $L \neq \emptyset$ **do**

  pick q from L; $L \leftarrow L \setminus \{q\}$

  $q.\varphi \leftarrow$ true

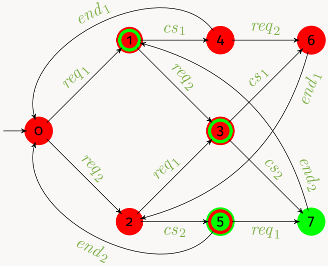
  forall $(q', \_, q) \in T$ **do**

    $q'.nb \leftarrow q'.nb - 1$

    if $q'.nb = 0 \land q'.\psi_1 =$ true $\land\, q'.\varphi =$ false **then**

      $L \leftarrow L \cup \{q'\}$

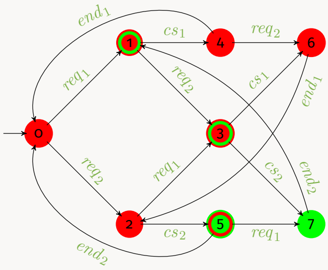
$\varphi = \mathsf{A} R_1 \mathsf{U} CS_2$



Exercise: Check another CTL formula (mutual exclusion)

```
/* Init */
markPred(ψ1)
markPred(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}
/* Main loop */
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 ∧ q'.ψ1 =
          true ∧ q'.φ = false then
            L ← L ∪ {q'}
```

$\varphi = \mathsf{A} R_1 \mathsf{U} CS_2$



Exercise: Check another CTL formula (mutual exclusion)

/* Init */

$markPred(\psi_1)$

$markPred(\psi_2)$

$L \leftarrow \emptyset$

forall $q \in Q$ **do**

- $q.nb \leftarrow degree(q)$
- $q.\varphi \leftarrow$ `false`

forall $q \in Q$ **do**

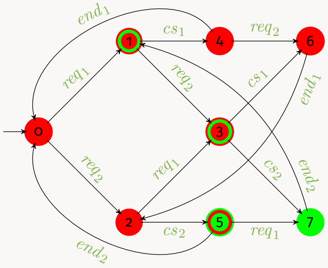
- **if** $q.\psi_2 =$ `true` **then**
  - $L \leftarrow L \cup \{q\}$

/* Main loop */

while $L \neq \emptyset$ **do**

- pick q from L; $L \leftarrow L \setminus \{q\}$
- $q.\varphi \leftarrow$ `true`
- **forall** $(q', \_, q) \in T$ **do**
  - $q'.nb \leftarrow q'.nb - 1$
  - **if** $q'.nb = 0 \wedge q'.\psi_1 =$ `true` $\wedge\, q'.\varphi =$ `false` **then**
    - $L \leftarrow L \cup \{q'\}$

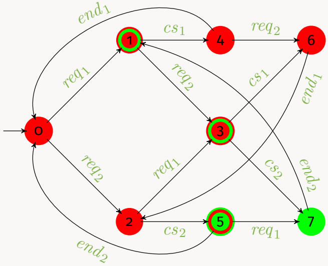
$\varphi = \mathsf{A} R_1 \mathsf{U}\, CS_2$



Exercise: Check another CTL formula (mutual exclusion)

/* Init */
$markPred(\psi_1)$
$markPred(\psi_2)$
$L \leftarrow \emptyset$
forall $q \in Q$ **do**
  $q.nb \leftarrow degree(q)$
  $q.\varphi \leftarrow$ `false`
forall $q \in Q$ **do**
  if $q.\psi_2 =$ `true` **then**
    $L \leftarrow L \cup \{q\}$
/* Main loop */
while $L \neq \emptyset$ **do**
  pick q from L; $L \leftarrow L \setminus \{q\}$
  $q.\varphi \leftarrow$ `true`
  forall $(q', \_, q) \in T$ **do**
    $q'.nb \leftarrow q'.nb - 1$
    if $q'.nb = 0 \land q'.\psi_1 =$ `true` $\land q'.\varphi =$ `false` **then**
      $L \leftarrow L \cup \{q'\}$

$\varphi = \mathsf{A} R_1 \mathsf{U} CS_2$



Outline

Reachability properties

How to characterize 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 (safety) 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 return to the initial state
- we can always 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 return to the initial state
- we can always 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 return to the initial state
- we can always 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 return to the initial state
- we can always 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 return to the initial state
- we can always 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 return to the initial state
- we can always 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 return to the initial state (simple)
- we can always 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 return to the initial state (simple)
- we can always return to the initial state (any reachable state)

Outline

Reachability in CTL

Form of formulae in CTL

- use the EF modal operator: EFφ
- φ is a propositional formula without temporal operator
- EU for conditional reachability
- Nesting AG and EF when considering any reachable state

Reachability in CTL: examples

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 CTL: examples

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 CTL: examples

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 CTL: examples

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 CTL: examples

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 CTL: examples

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 CTL: examples

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 CTL: examples

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:

Outline

4 Reachability Properties

- Reachability in CTL
- Computing the state space
- Specifying properties using observers

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

Outline

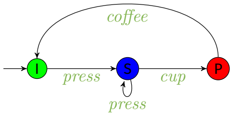
Verifying properties using observers

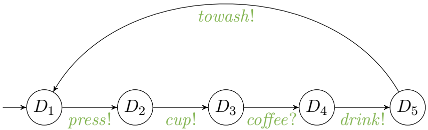
An observer is an automaton that observes the system behavior

- It synchronizes with other automata's actions
- It must be non-blocking (see example on the white board)
  - Note: a complete automaton is never blocking
- Its location(s) give an indication on the system property

Then verifying the property reduces to a reachability property on the observer (in parallel with the system)

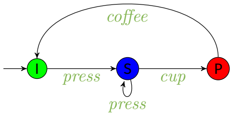
Observers for the coffee machine (1/3)

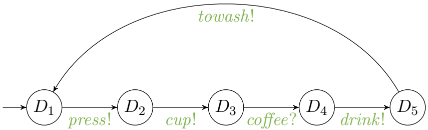




Design an observer for the coffee machine and the drinker verifying that it is possible to order a coffee with exactly one dose of sugar.
(…and check the validity of the property)

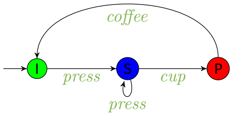
Observers for the coffee machine (2/3)

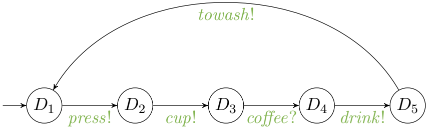




Design an observer for the coffee machine and the drinker verifying that it is possible to order a coffee with *at least* one dose of sugar.
(...and check the validity of the property)

Observers for the coffee machine (3/3)

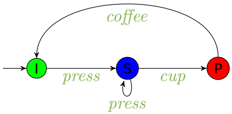


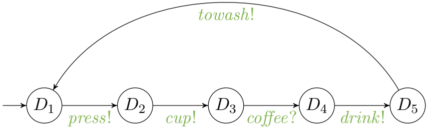


Consider the following property on the coffee machine and the coffee drinker: "whenever the (first) coffee comes, no cup was put to the washing machine before"

1. express this property using CTL
2. write an observer verifying this property

Observers for the coffee machine (3/3)





Consider the following property on the coffee machine and the coffee drinker: “whenever the (first) coffee comes, no cup was put to the washing machine before”

1. express this property using CTL $\mathsf{A}(\neg towash)\mathsf{U}\,coffee$
2. write an observer verifying this property

Outline

The state space explosion problem

Systems are often designed in a compositional manner. Example:

- a server
- 10 clients
- a network

The synchronized product becomes exponentially larger

- even worse in the presence of variables

The state space explosion problem

Systems are often designed in a compositional manner. Example:

- a server
- 10 clients
- a network

The synchronized product becomes exponentially larger

- even worse in the presence of variables

State space explosion problem

The state space explosion problem is the main obstacle to model checking algorithms, because of the necessity to construct the entire state space.

The state space explosion problem: examples

Example

For a system made of 10 components with 10 states each, the (maximum) number of different states in the global system is

The state space explosion problem: examples

Example

For a system made of 10 components with 10 states each, the (maximum) number of different states in the global system is

The state space explosion problem: examples

Example

For a system made of 10 components with 10 states each, the (maximum) number of different states in the global system is

Example

For a system made of 10 components with each of them having 10 states and one variable with (only!) 10 possible values each, the (maximum) number of different states in the global system is

The state space explosion problem: examples

Example

For a system made of 10 components with 10 states each, the (maximum) number of different states in the global system is

Example

For a system made of 10 components with each of them having 10 states and one variable with (only!) 10 possible values each, the (maximum) number of different states in the global system is

Example

For a system made of 100 components with 10 states each, the (maximum) number of different states in the global system is

The state space explosion problem: examples

Example

For a system made of 10 components with 10 states each, the (maximum) number of different states in the global system is

Example

For a system made of 10 components with each of them having 10 states and one variable with (only!) 10 possible values each, the (maximum) number of different states in the global system is

Example

For a system made of 100 components with 10 states each, the (maximum) number of different states in the global system is

Alleviating the state space explosion problem

An active research field since the 1980s!

Some techniques (among many others):

- construct only the part of the state space needed
  - on-the-fly construction of the synchronized product
- parallel/distributed verification
  - more computational power
  - ...but not all algorithms can be parallelized!
- parameterized verification [AD16]
- symmetry reductions, partial order reductions...
  - remove "similar" parts of the state spaces
- symbolic model checking [Ake78] [Bur+92]

- [AD16] Parosh Aziz Abdulla and Giorgio Delzanno. « Parameterized verification ». In: *International Journal on Software Tools for Technology Transfer* 18.5 (2016), pp. 469–473
- [Ake78] Sheldon B. Akers Jr. « Binary Decision Diagrams ». In: *IEEE Transactions on Computers* 27.6 (1978), pp. 509–516
- [Bur+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. « Symbolic Model Checking: 10^{20} States and Beyond ». In: *Information and Computation* 98.2 (1992), pp. 142–170

Motivation for symbolic approaches

- Idea: represent symbolically states and transitions

- A symbolic state aims at representing concisely large sets of states

Outline

Symbolic computation of state sets

Let $\mathcal{A} = \langle Q, \Sigma, T, q_0, lab, F \rangle$ 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
- $Sat(\varphi) = \{q \in Q \mid q \models \varphi\}$ 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 $Sat(\varphi)$

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

Symbolic features

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

Example

- suppose the automaton has 2 integer variables $a, b \in \{0, \ldots, 255\}$
- 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

Symbolic features

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

Example

- suppose the automaton has 2 integer variables $a, b \in \{0, \ldots, 255\}$
- 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 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
  - Can encode

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 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
  - Can encode

Requirements for symbolic model checking

1. symbolic representation of $Sat(p)$ for each proposition $p \in AP$
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

Outline

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, …, x_n

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

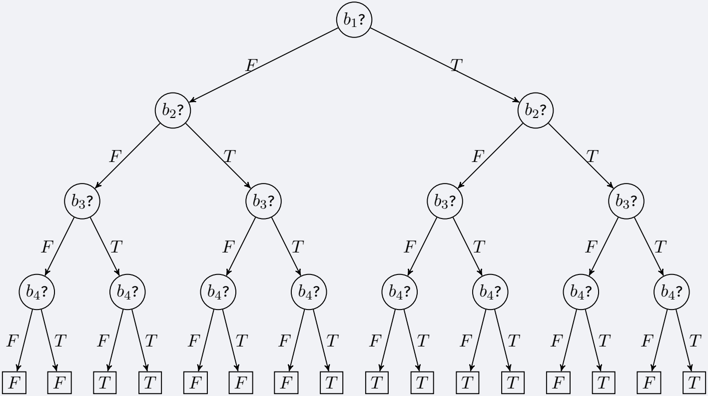
BDD structure

n Boolean variables x_1, ..., x_n

- suppose $n = 4$.
  $\langle b_1, b_2, b_3, b_4 \rangle$ associates values with x_1, ..., 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:
  - $|S| = 9$: $$\begin{aligned} S = \{ \quad & \langle F, F, T, F \rangle, \langle F, F, T, T \rangle, \langle F, T, T, T \rangle, \\ & \langle T, F, F, F \rangle, \langle T, F, F, T \rangle, \langle T, F, T, F \rangle, \\ & \langle T, F, T, T \rangle, \langle T, T, F, T \rangle, \langle T, T, T, T \rangle \} \end{aligned}$$
  - 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

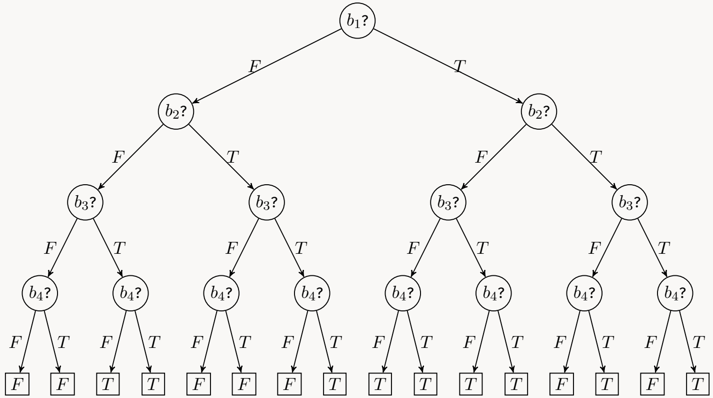
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

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

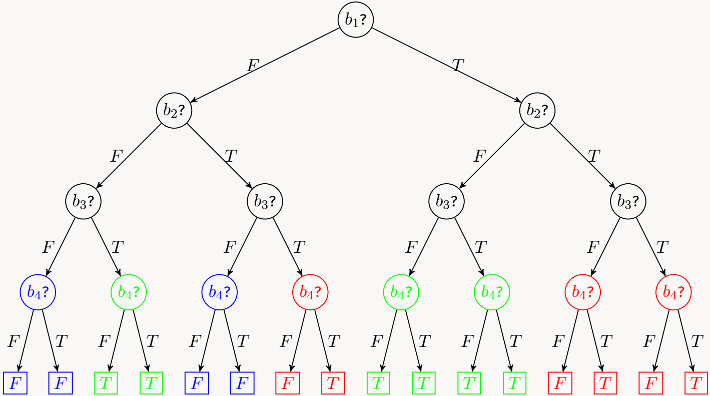
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

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

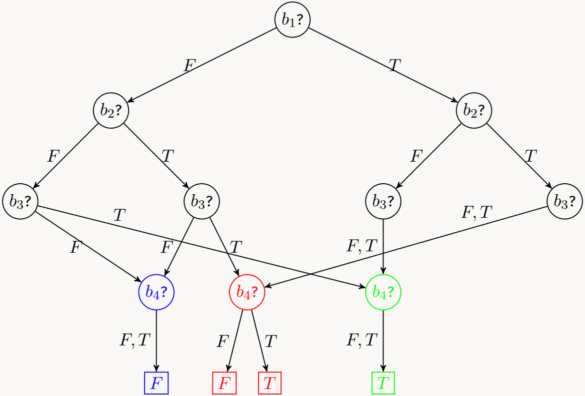
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

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

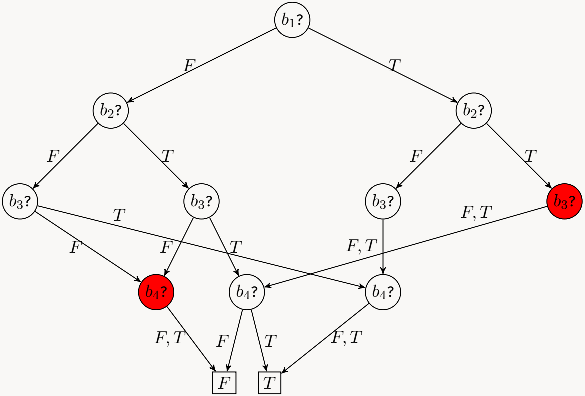
$$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$$



BDD: a reduced decision tree

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

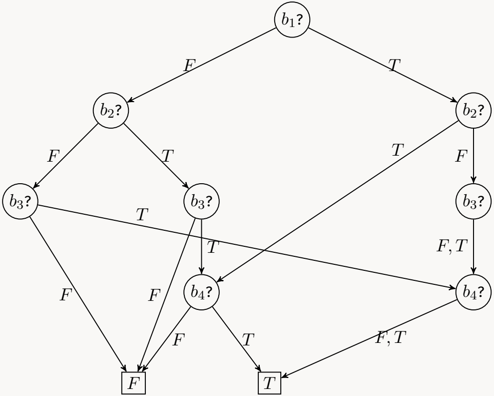
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

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

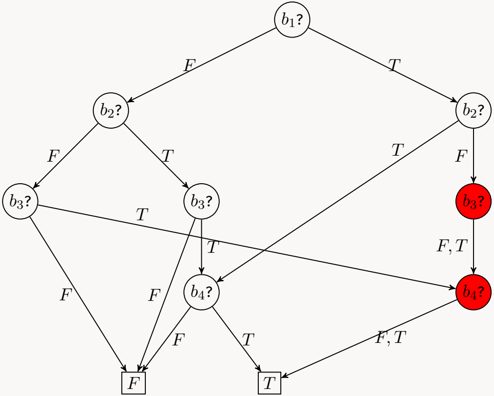
$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$



BDD: a reduced decision tree

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

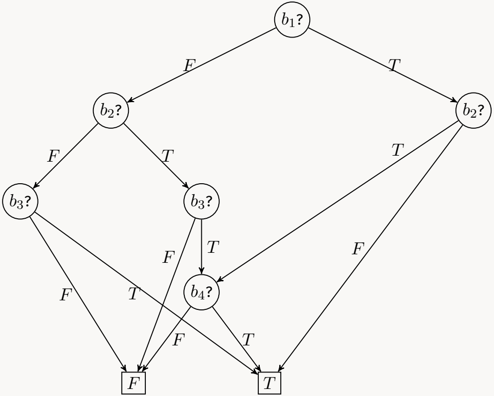
$$(b_1 \vee b_3) \wedge (b_2 \implies b_4)$$



BDD: a reduced decision tree

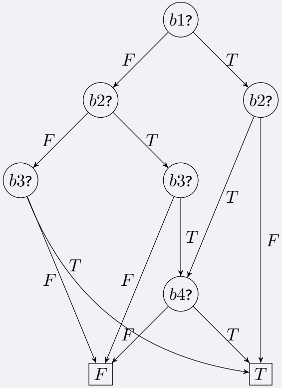
- identical subtrees are shared $\rightsquigarrow$ directed acyclic graph (*dag*)
- internal superfluous nodes are deleted (where no choice is possible)

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



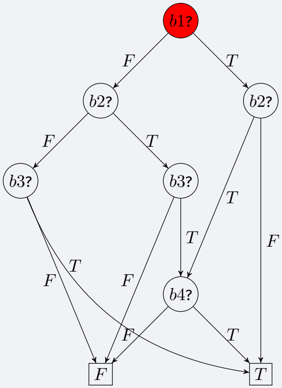
Testing whether a tuple belongs to the set

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



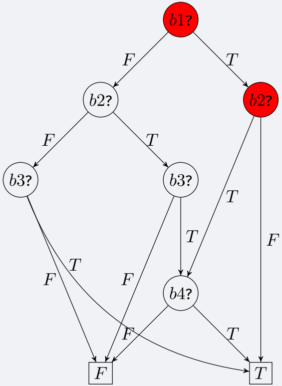
Testing whether a tuple belongs to the set

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



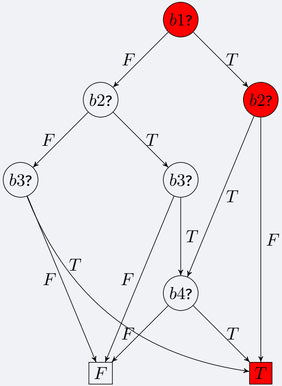
Testing whether a tuple belongs to the set

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



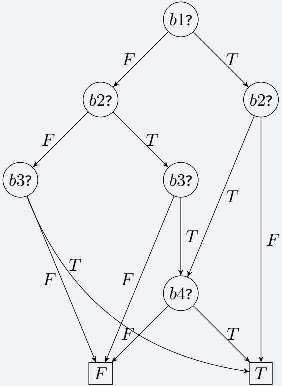
Testing whether a tuple belongs to the set

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



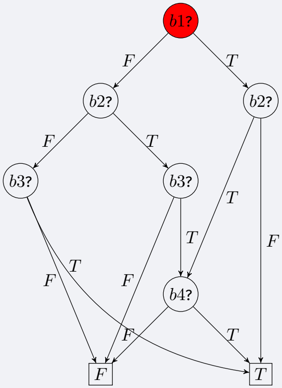
Testing whether a tuple belongs to the set

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



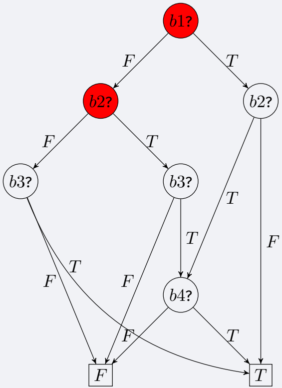
Testing whether a tuple belongs to the set

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



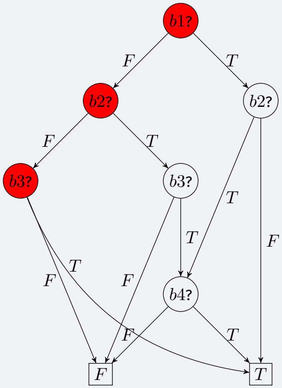
Testing whether a tuple belongs to the set

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



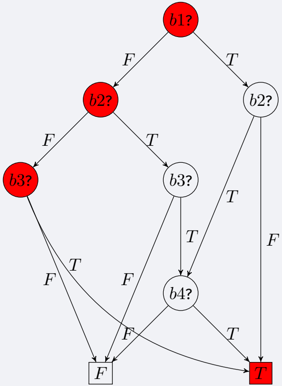
Testing whether a tuple belongs to the set

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

Exercise

Give the BDD for $\neg\Big((b_1 \land (b_2 \lor b_4) \land b_5\Big) \lor \neg b_3) \lor \Big(b_4 \implies (b_3 \land b_5)\Big)$

Exercise

Exercise

Give the BDD for $\neg\Big((b_1 \land (b_2 \lor b_4) \land b_5\Big) \lor \neg b_3) \lor \Big(b_4 \implies (b_3 \land b_5)\Big)$

Exercise

Exercise

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

Exercise

Exercise

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

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 tautology

Advantages of BDDs

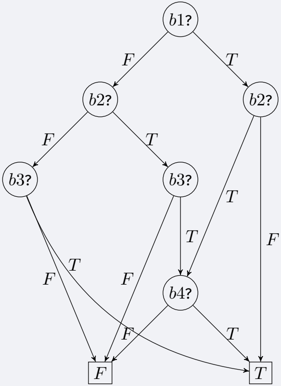
- small representations
- existence of a canonical BDD structure:
  - unicity for a fixed order of the variables
  - test the equivalence of two symbolic representations
  - test the tautology
  - test the emptiness

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 tautology
  - test the emptiness
  - simple operations: complement, union, intersection, projection

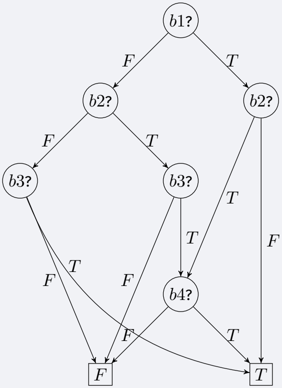
Exercise: Complement

Exercise (Complement)



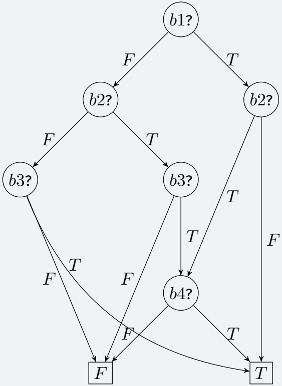
Exercise: Complement

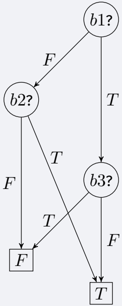
Exercise (Complement)



Exercise: Union

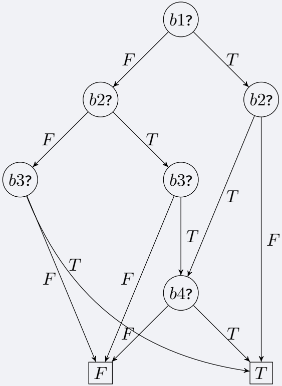
Exercise (Union)

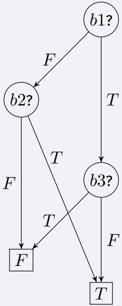




Exercise: Union

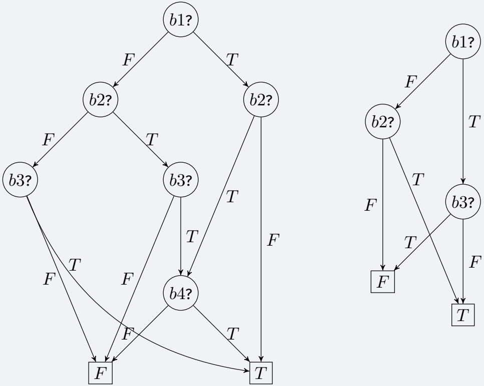
Exercise (Union)





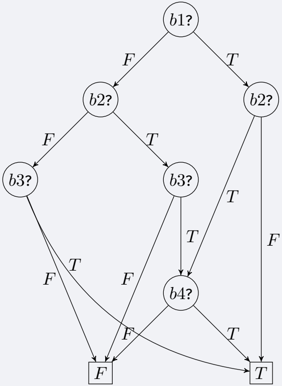
Exercise: Intersection

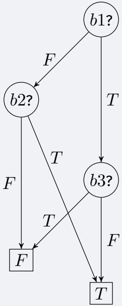
Exercise (Intersection)



Exercise: Intersection

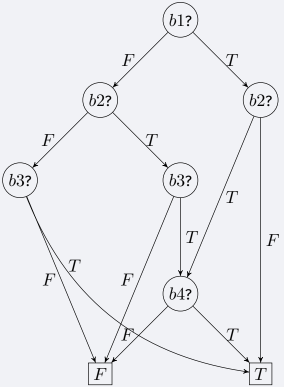
Exercise (Intersection)





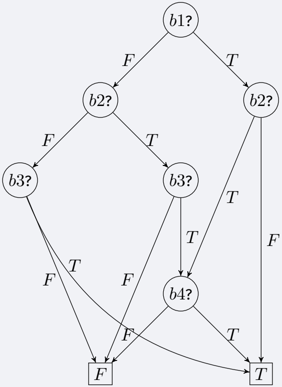
Exercise: Projection

Exercise (Projection $S[b_3 \leftarrow T]$)



Exercise: Projection

Exercise (Projection $S[b_3 \leftarrow T]$)



Outline

Representing automata by BDDs

Encoding of states

- Boolean encoding of states
- Boolean encoding of each individual variable

Representing automata by BDDs: example

Example

Let us consider an automaton with:

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

How many bits are necessary to encode a state $\langle q, d, r \rangle$?

Representing automata by BDDs: example

Example

Let us consider an automaton with:

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

How many bits are necessary to encode a state $\langle q, d, r \rangle$?

bits for the state

Representing automata by BDDs: example

Example

Let us consider an automaton with:

- $Q = \{q_0, \ldots, q_6\}$
- an integer variable $digit \in \{0, \ldots, 9\}$
- a Boolean variable $ready$

How many bits are necessary to encode a state $\langle q, d, r \rangle$?

bits for the state

bits for $digit$

Representing automata by BDDs: example

Example

Let us consider an automaton with:

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

How many bits are necessary to encode a state $\langle q, d, r \rangle$?

bits for the state

bits for $digit$

bit for $ready$.

So a state can be encoded with

Representing automata by BDDs: example

Example

Let us consider an automaton with:

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

How many bits are necessary to encode a state $\langle q, d, r \rangle$?

bits for the state

bits for $digit$

bit for $ready$.

So a state can be encoded with bits.

For example, $\langle q_3, 8, F \rangle$ is represented by:

Representing automata by BDDs: example

Example

Let us consider an automaton with:

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

How many bits are necessary to encode a state $\langle q, d, r \rangle$?

bits for the state

bits for $digit$

bit for $ready$.

So a state can be encoded with bits.

For example, $\langle q_3, 8, F \rangle$ is represented by:

Representing a set of states

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

Representing a set of states

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

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:

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

Bibliography

References I

[AD16] Parosh Aziz Abdulla and Giorgio Delzanno. « Parameterized verification ». In: *International Journal on Software Tools for Technology Transfer* 18.5 (2016), pp. 469–473. DOI: 10.1007/s10009-016-0424-3.

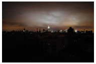
[Ake78] Sheldon B. Akers Jr. « Binary Decision Diagrams ». In: *IEEE Transactions on Computers* 27.6 (1978), pp. 509–516. DOI: 10.1109/TC.1978.1675141.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9.

[Bur+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. « Symbolic Model Checking: 10^{20} States and Beyond ». In: *Information and Computation* 98.2 (1992), pp. 142–170. DOI: 10.1016/0890-5401(92)90017-A.

Additional information

Explanation for the 3 pictures in the beginning



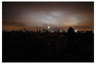
Allusion to the Northeast blackout (USA, 2003)
Computer bug
Consequences: 11 fatalities, huge cost
(Picture actually from the Sandy Hurricane, 2012)

Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
28 fatalities, hundreds of injured
Computer bug: software error (clock drift)
(Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)

Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
No fatalities
Computer bug: inaccurate finite element analysis modeling
(Picture actually from the Deepwater Horizon Offshore Drilling Platform)

Credits

Source of the graphics used I



Titre : Hurricane Sandy Blackout New York Skyline
Auteur : David Shankbone
Source : https://commons.wikimedia.org/wiki/File:Hurricane_Sandy_Blackout_New_York_Skyline.JP
Licence : BY

Titre : Deepwater Horizon Offshore Drilling Platform on Fire
Auteur : ideum
Source : https://secure.flickr.com/photos/ideum/4711481781/
Licence : BY SA

Titre : DA-SC-88-01663
Auteur : imcomkorea
Source : https://secure.flickr.com/photos/imcomkorea/3017886760
Licence : BY NC ND

Titre : Smiley green alien big eyes (aaah)
Auteur : LadyofHats
Source : https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
Licence : PUBLIC DOMAIN

Titre : Smiley green alien big eyes (cry)
Auteur : LadyofHats

Source of the graphics used II

Source : https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
Licence : PUBLIC DOMAIN

Titre : Coffee machine drawing
Auteur : Ysangkok
Source : https://commons.wikimedia.org/wiki/File:Coffee_machine.svg
Licence : PUBLIC DOMAIN

Titre : taking a coffee break
Auteur : chris
Source : https://commons.wikimedia.org/wiki/File:SMirC-coffeebreak.svg
Licence : CC BY

License of this document

Authors: **Laure Petrucci** and **Étienne André**

(LaTeX source available to academic teachers upon request)

Version: September 3, 2024