C08 – Model checking

Program Verification

FMI · Denisa Diaconescu · Spring 2025

What is Model Checking?

The big picture

- Application domain: mostly concurrent, reactive systems
- Verify that a system satisfies a property
  1. Model the system in the model checker's description language. Call this model M.
  2. Express the property to be verified in the model checker's specification language. Call this formula ϕ.
  3. Run the model checker to show that M satisfies ϕ.
- Automatic for finite-state models

Example

- Two processes executed in parallel
- Each process undergoes transitions $n \to r \to c \to n \to \ldots$ where
  - n denotes "**n**ot in critical section"
  - r denotes "**r**equesting to enter critical section"
  - c denotes "**c**ritical section"

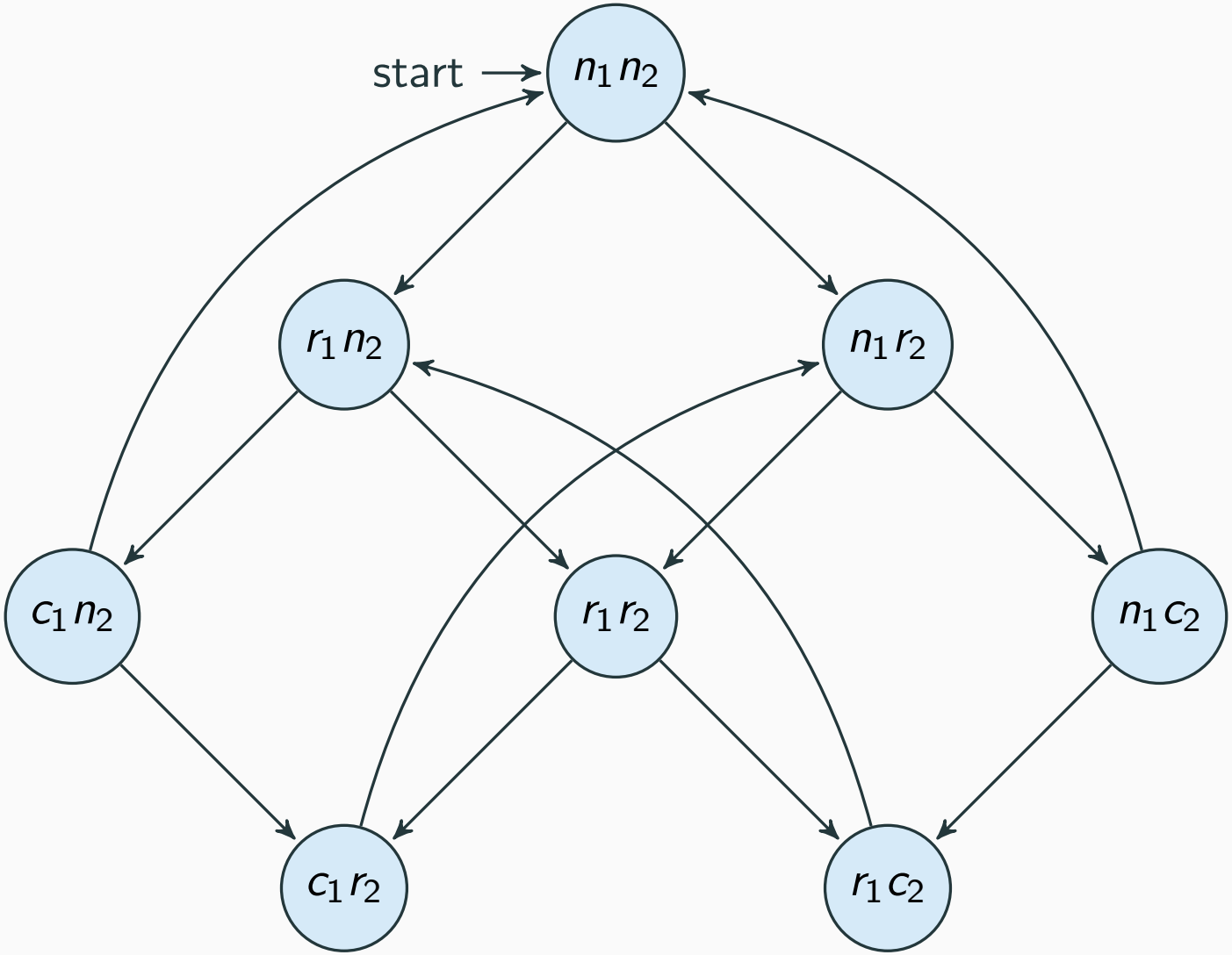
Example

- Two processes executed in parallel
- Each process undergoes transitions $n \to r \to c \to n \to \ldots$ where
  - n denotes "**n**ot in critical section"
  - r denotes "**r**equesting to enter critical section"
  - c denotes "**c**ritical section"
- Requirements
  - Safety: only one process may execute critical section code at any point
  - Liveness: whenever a process requests to enter its critical section, it will eventually be allowed to do so
  - Non-blocking: a process can always request to enter its critical section

Example (cont.)

- We write a program P to fulfill these requirements. But is it really doing its job?
- We construct a model M for P such that M captures the relevant behaviour of P.

Example (cont.)



Example (cont.)

- Based on a definition of when a model satisfies a property, we can determine whether M satisfies P's required properties.
- Example: M satisfies the safety requirement if no state reachable from the start state (including itself) is labeled $c_1 c_2$. Thus our M satisfies P's safety requirement.
- NOTE: the conclusion that P satisfies these requirements depends on the (unverified) assumption that M is a faithful representation of all the relevant aspects of P.

Uses of Model Checking

Verification of specific properties of

- Hardware circuits
- Communication protocols
- Control software
- Embedded systems
- ...

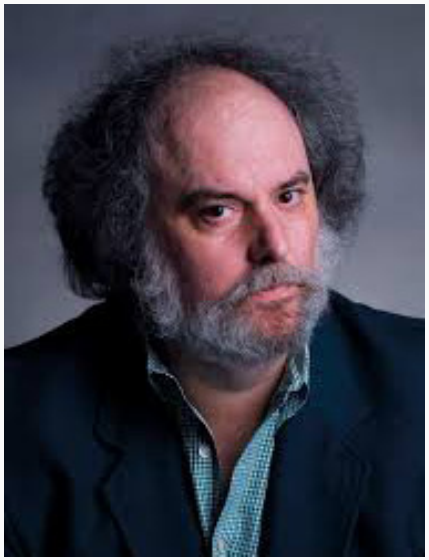
Uses of Model Checking

Model Checking has been used to

- Check Microsoft Windows device drivers for bugs
  - The Static Driver Verifier tool
  - SLAM project of Microsoft
- The SPIN tool
  - `http://spinroot.com`
  - `http://spinroot.com/spin/success.html`
  - Flood control barrier control software
  - Parts of *Mars Science Laboratory*, *Deep Space 1*, *Cassini*, *the Mars Exploration Rovers*, *Deep Impact*
- PEPA (Performance Evaluation Process Algebra)
  - `http://www.dcs.ed.ac.uk/pepa/`
  - Multiprocess systems
  - Biological systems
- ...

The ACM Turing Award in 2007

Edmund Clarke



Allen Emerson

Joseph Sifakis

"for their role in developing Model-Checking into a highly effective verification technology that is widely adopted in the hardware and software industries"

Model Checking - Models

A model of some system has

- A finite set of states
- A subset of states considered as the initial states
- A transition relation which, given a state, describes all the states that can be reached "in one time step"

Model Checking - Models

A model of some system has

- A finite set of states
- A subset of states considered as the initial states
- A transition relation which, given a state, describes all the states that can be reached "in one time step"

Refinements of this setup can handle:

- Infinite state spaces
- Continuous state spaces
- Probabilistic Transitions
- ...

Model Checking - Models

Models are always abstraction of reality.

- We must choose what to model and what not to model
- There are limitations forced by the formalism
  - e.g., here we are limited to finite state models
- There will be things we do not understand sufficiently to model
  - e.g., people

Source: *La trahison des images* by René Magritte. Licensed under Fair use via Wikipedia
http://en.wikipedia.org/wiki/File:MagrittePipe.jpg#mediaviewer/File:MagrittePipe.jpg

Model Checking - Specifications

We are interested in specifying behaviours of systems over time (use Temporal Logic).

Specifications are built from

- primitive properties of individual states
  - e.g., *is on*, *is off*, *is active*, *is reading*
- propositional connectives $\wedge, \vee, \neg, \rightarrow$
- temporal connectives
  - e.g., *At* **all times**, *the system is not simultaneously reading and writing.*
  - e.g., *If a request signal is asserted* **at some time**, *a corresponding grant signal will be asserted* **within 10 time units.**

Model Checking - Specifications

The exact set of temporal connectives differs across temporal logics.

Logics can differ in how they treat time:

- Continuous time vs. Discrete time
- Linear Time vs. Branching time
- ...

Linear vs. Branching Time

Linear Time

- Considers paths (sequences of states)
- Questions of the form
  - *For all paths, does some path property hold?*
  - *Does there exist a path such that some path property holds?*

Linear vs. Branching Time

Linear Time

- Considers paths (sequences of states)
- Questions of the form
  - *For all paths, does some path property hold?*
  - *Does there exist a path such that some path property holds?*

Branching Time

- Considers trees of possible future states from each initial state
- Questions can become more complex
  - *For all states reachable from an initial state, does there exist an onwards path to a state satisfying some property?*

LTL Logic

Syntax

LTL = Linear(-time) Temporal Logic

Assume some set *Atoms* of atomic propositions.

LTL formulas are defined by:

$$\varphi \;:=\; p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \to \varphi \mid \bigcirc\ \varphi \mid \Diamond\ \varphi \mid \Box\ \varphi \mid \varphi\ \mathbf{U}\ \varphi$$

Pronunciation:

- $\bigcirc\ \varphi$ – Next φ
- $\Diamond\ \varphi$ – Eventually φ
- $\Box\ \varphi$ – Always φ
- $\varphi\ \mathbf{U}\ \psi$ – φ Until ψ

Precedence high-to-low:

- $\bigcirc, \Diamond, \Box, \neg$
- $\mathbf{U}$
- $\wedge, \vee$
- $\to$

Syntax

Example
The following are LTL formulas:

- $\Diamond\ p \wedge \Box\ q \rightarrow p\ \mathbf{U}\ r$
- $\Diamond(p \rightarrow \Box\ r) \vee \neg q\ \mathbf{U}\ p$
- $p\ \mathbf{U}\ (q\ \mathbf{U}\ r)$
- $\Box\ \Diamond\ p \rightarrow \Diamond(q \vee s)$

Syntax

Example
The following are LTL formulas:

- $\Diamond\ p \wedge \Box\ q \rightarrow p\ \mathbf{U}\ r$
- $\Diamond(p \rightarrow \Box\ r) \vee \neg q\ \mathbf{U}\ p$
- $p\ \mathbf{U}\ (q\ \mathbf{U}\ r)$
- $\Box\ \Diamond\ p \rightarrow \Diamond(q \vee s)$

The following are not LTL formulas:

- $\mathbf{U}\ r$
- $q\ \Box\ p$

Informal semantics

A path is a sequence of states connected by transitions.

LTL formulas are evaluated at a position i along a path π through the system.

- An atomic proposition p holds if p is true in the state at position i.
- The propositional connectives $\neg, \wedge, \vee, \rightarrow$ have their usual meanings.
- Meaning of temporal connectives:
  - $\bigcirc\varphi$ holds if φ holds at the next position
  - $\Diamond\varphi$ holds if there exists a future position where φ holds
  - $\Box\varphi$ holds if φ holds in all future positions
  - $\varphi \mathbf{U} \psi$ holds if there exists a future position where ψ holds, and φ holds for all positions prior to that

Informal semantics

Example

$\square$ *invariant*

- *invariant* is true for all future positions

Informal semantics

Example

$\square$ *invariant*

- *invariant* is true for all future positions

$\square \neg(\mathit{read} \wedge \mathit{write})$

- In all future positions, it is not the case that *read* and *write*

Informal semantics

Example

$\Box$ *invariant*

- *invariant* is true for all future positions

$\Box \neg(\mathit{read} \wedge \mathit{write})$

- In all future positions, it is not the case that *read* and *write*

$\Box(\mathit{request} \rightarrow \Diamond\ \mathit{grant})$

- At every point in the future, a *request* implies that there exists a future point where *grant* holds.

Informal semantics

Example

$\square(\mathit{request} \rightarrow (\mathit{request}\ \mathbf{U}\ \mathit{grant}))$

- At every point in the future, a *request* implies that there exists a future point where *grant* holds, and *request* holds up until that point.

Informal semantics

Example

$\Box(\mathit{request} \rightarrow (\mathit{request}\ \mathbf{U}\ \mathit{grant}))$

- At every point in the future, a *request* implies that there exists a future point where *grant* holds, and *request* holds up until that point.

$\Box \Diamond\ \mathit{enabled}$

- In all future positions, there is a future position where *enabled* holds.

Informal semantics

Example

$\Box(\mathit{request} \rightarrow (\mathit{request}\ \mathbf{U}\ \mathit{grant}))$

- At every point in the future, a *request* implies that there exists a future point where *grant* holds, and *request* holds up until that point.

$\Box\ \Diamond\ \mathit{enabled}$

- In all future positions, there is a future position where *enabled* holds.

$\Diamond\ \Box\ \mathit{enabled}$

- There is a future position, from which all future positions have *enabled* holding.

Transition Systems and Paths

A transition system (or model) $\mathcal{M} = (S, \rightarrow, L)$ consists of

- S a finite set of states
- $\rightarrow \subseteq S \times S$ a transition relation
- $L : S \rightarrow \mathcal{P}(Atoms)$ a labelling function

such that for all $\forall s_1 \in S$ there exists $s_2 \in S$ with $s_1 \rightarrow s_2$ (*serial condition*).

Note:

- *Atoms* is a fixed set of atomic propositions and $\mathcal{P}(Atoms)$ is the powerset of *Atoms*. Thus, $L(s)$ is just the set of atomic propositions that are true in state s.

Transition Systems and Paths

A transition system (or model) $\mathcal{M} = (S, \rightarrow, L)$ consists of

- S a finite set of states
- $\rightarrow \subseteq S \times S$ a transition relation
- $L : S \rightarrow \mathcal{P}(Atoms)$ a labelling function

such that for all $\forall s_1 \in S$ there exists $s_2 \in S$ with $s_1 \rightarrow s_2$ (*serial condition*).

Note:

- *Atoms* is a fixed set of atomic propositions and $\mathcal{P}(Atoms)$ is the powerset of *Atoms*. Thus, $L(s)$ is just the set of atomic propositions that are true in state s.

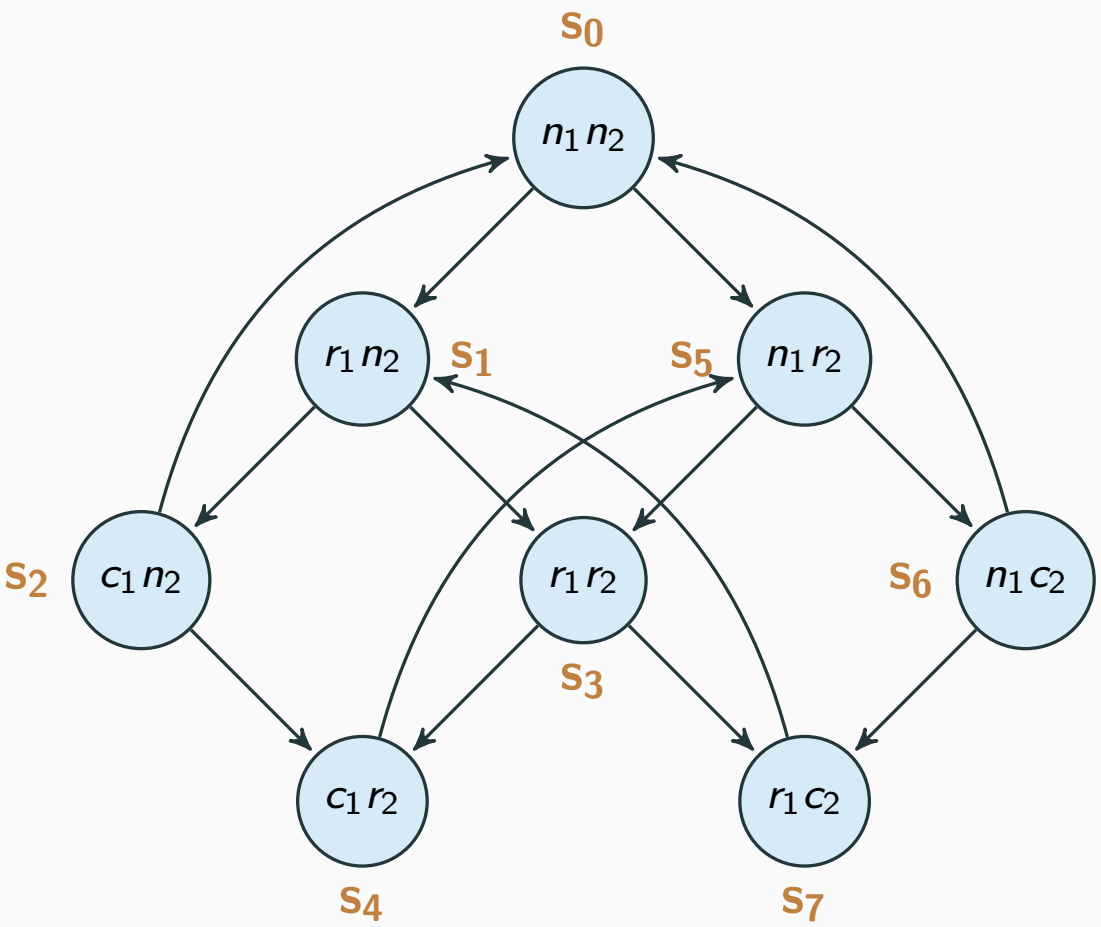
A path π in a transition system $\mathcal{M} = (S, \rightarrow, L)$ is an infinite sequence of states $s_0, s_1, s_2, \ldots$ such that for all $i \geq 0$, $s_i \rightarrow s_{i+1}$.

Paths are written as $\pi = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$

We write π^i for the suffix starting at s_i. For example, π^3 is $s_3 \rightarrow s_4 \rightarrow \ldots$

Transition Systems and Paths

Example



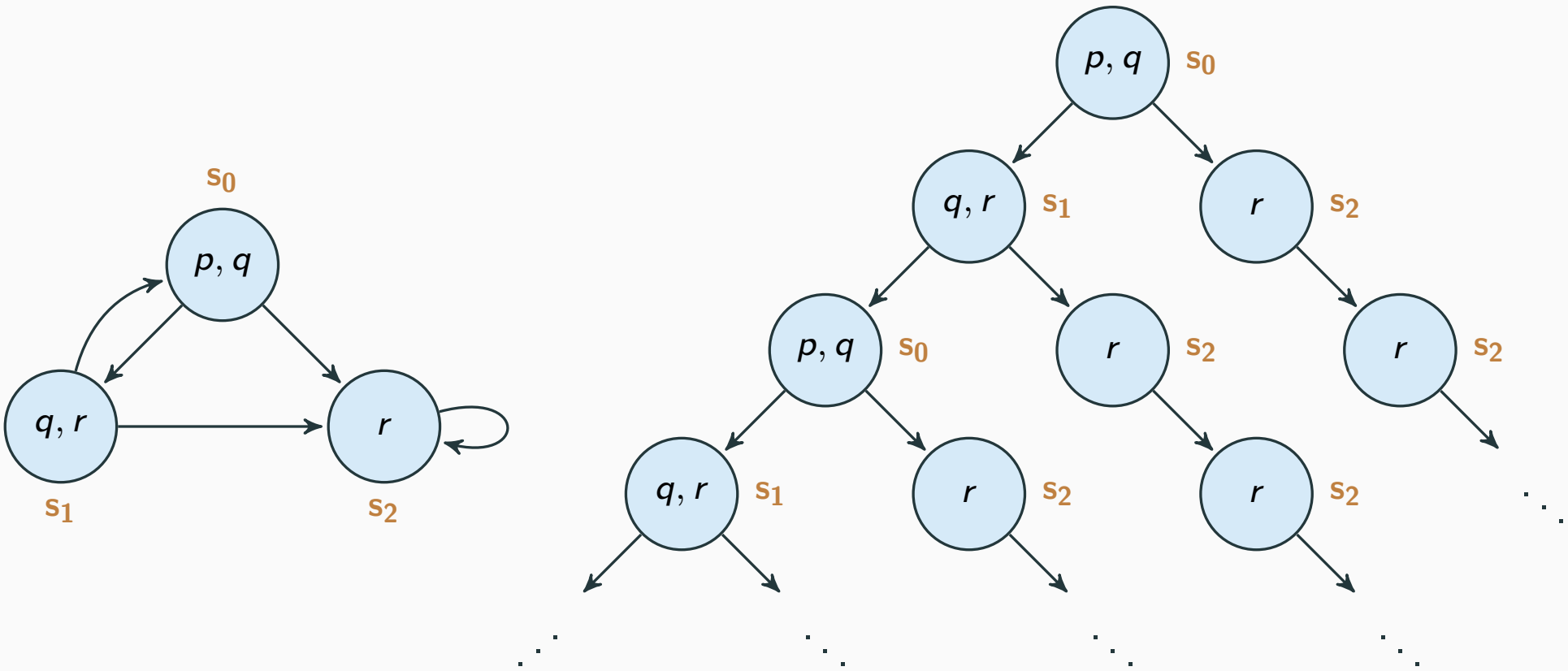
$Atoms = \{n_1, n_2, r_1, r_2, c_1, c_2\}$

$\mathcal{M} = (S, \rightarrow, L)$ where

- $S = \{s_0, s_1, \ldots, s_7\}$
- $\rightarrow = \{(s_0, s_1), (s_0, s_5), \ldots\}$
- $L(s_0) = \{n_1, n_2\}$
- $L(s_1) = \{r_1, n_2\}$
- $\ldots$

Unwinding a Transition System

Visualise all computational paths from a given state s by unwinding the transition system to obtain an infinite computation tree.



Satisfaction relation

Let $\mathcal{M} = (S, \to, L)$ be a model and $\pi = s_0 \to s_1 \to \ldots$ be a path in $\mathcal{M}$.

"The path π satisfies the LTL formula φ"

$$\pi \models \varphi$$

$\pi \models \top$

$\pi \not\models \bot$

$\pi \models p$	iff	$p \in L(s_0)$
$\pi \models \varphi \wedge \psi$	iff	$\pi \models \varphi$ and $\pi \models \psi$
$\pi \models \varphi \vee \psi$	iff	$\pi \models \varphi$ or $\pi \models \psi$
$\pi \models \varphi \to \psi$	iff	$\pi \models \varphi$ implies $\pi \models \psi$

Satisfaction relation

$$
\begin{array}{lll}
\pi \models \bigcirc\, \varphi & \text{iff} & \pi^1 \models \varphi \\
\pi \models \Diamond\, \varphi & \text{iff} & \text{there exists } i \geq 0 \text{ such that } \pi^i \models \varphi \\
\pi \models \Box\, \varphi & \text{iff} & \text{for all } i \geq 0 \text{ we have } \pi^i \models \varphi \\
\pi \models \varphi_1 \,\mathbf{U}\, \varphi_2 & \text{iff} & \text{there exists } i \geq 0 \text{ such that } \pi^i \models \varphi_2 \text{ and} \\
 & & \text{for all } j \in \{0, \ldots, i-1\} \text{ we have } \pi^j \models \varphi_1
\end{array}
$$

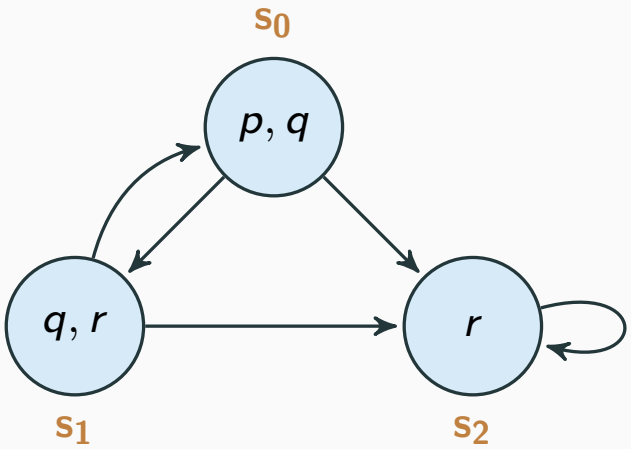
We write

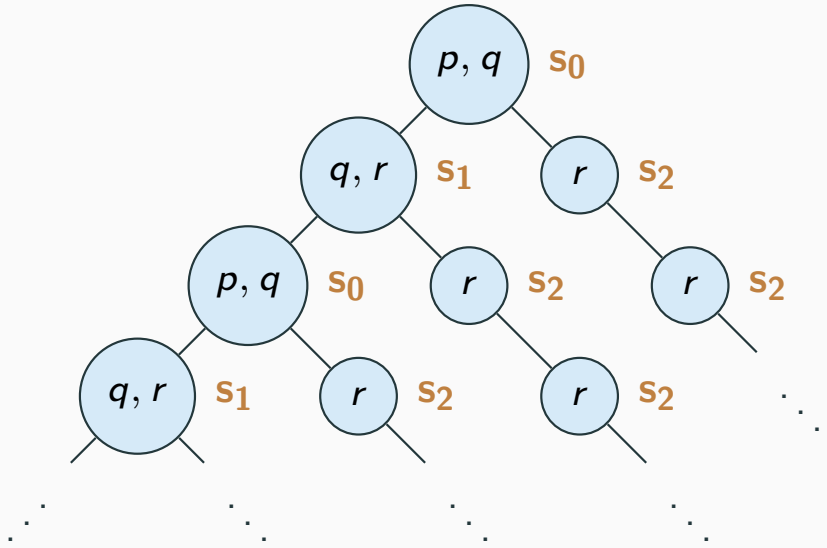
$$\mathcal{M}, s \models \varphi$$

if for every path π of a model $\mathcal{M}$ starting at state s we have $\pi \models \varphi$.

Satisfaction relation

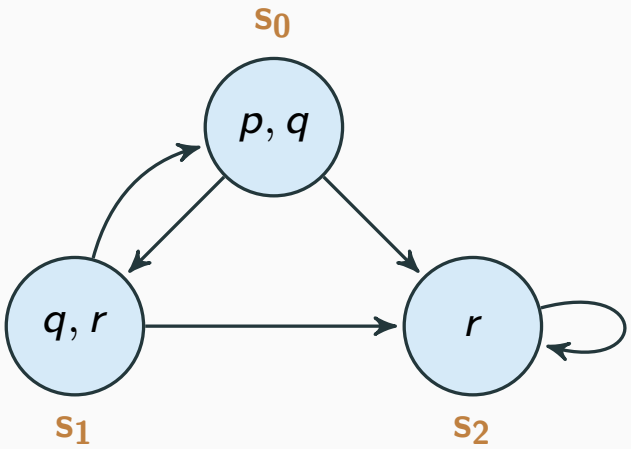
Example

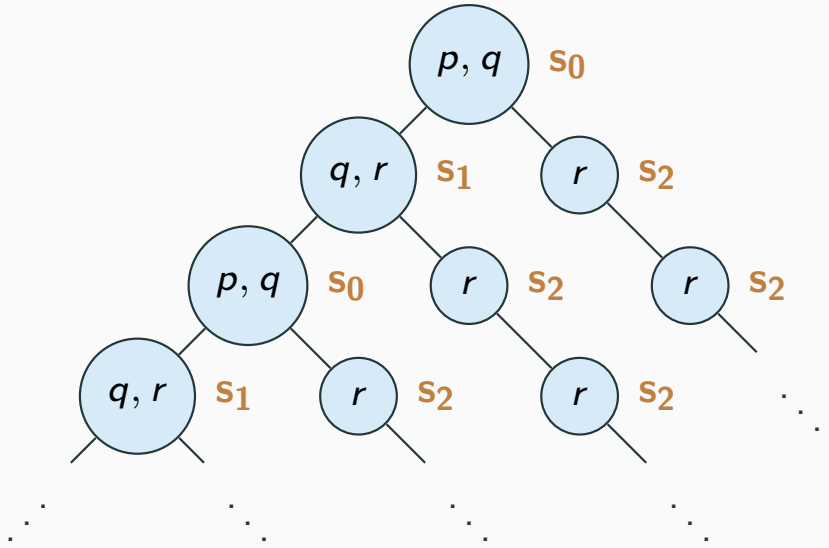




Satisfaction relation

Example

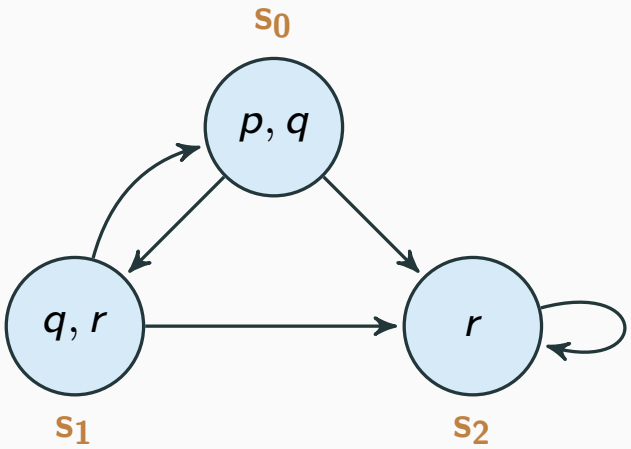


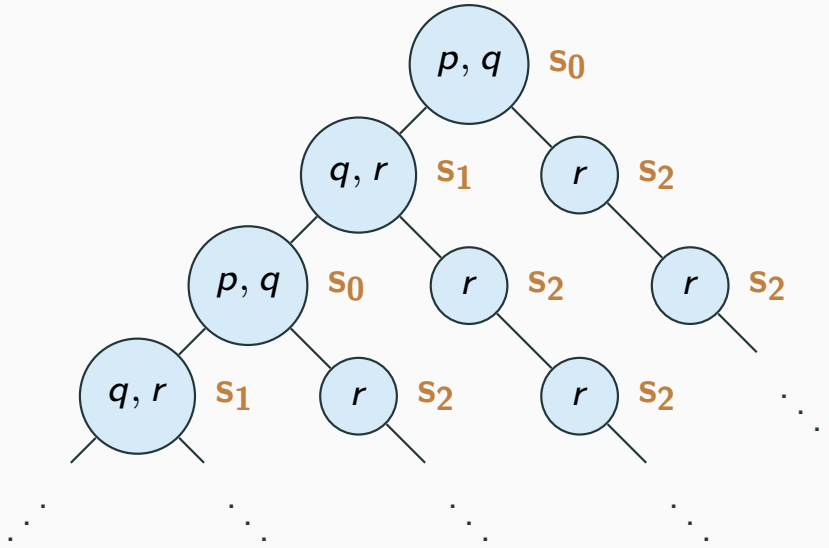


1. $\mathcal{M}, s_0 \models p \wedge q$

Satisfaction relation

Example

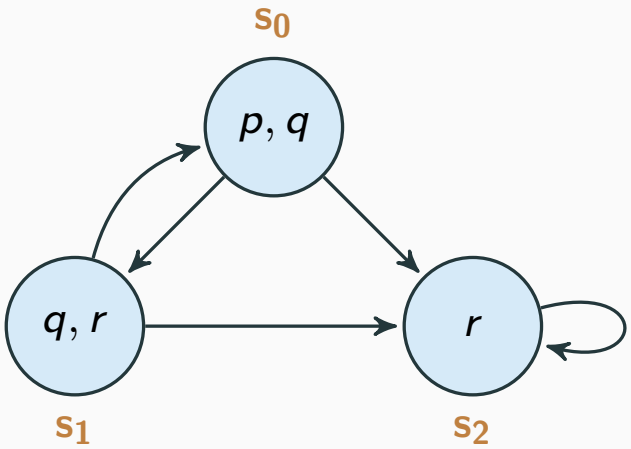


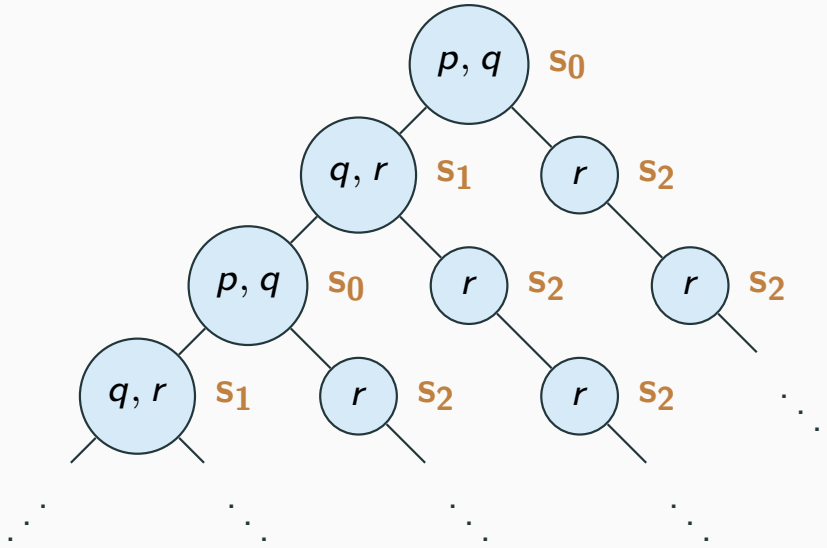


1. $\mathcal{M}, s_0 \models p \wedge q$
2. $\mathcal{M}, s_0 \models \neg r$

Satisfaction relation

Example

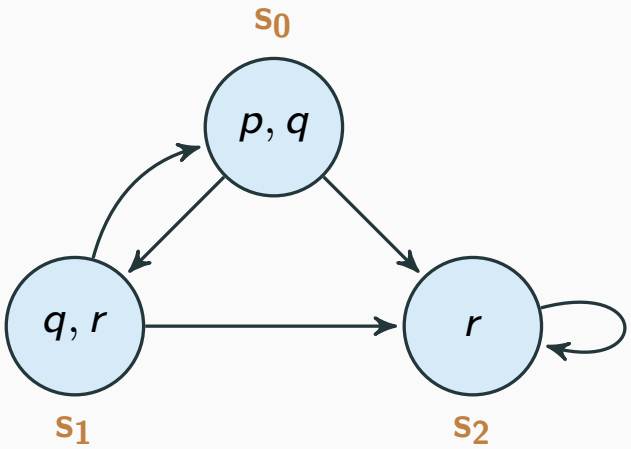


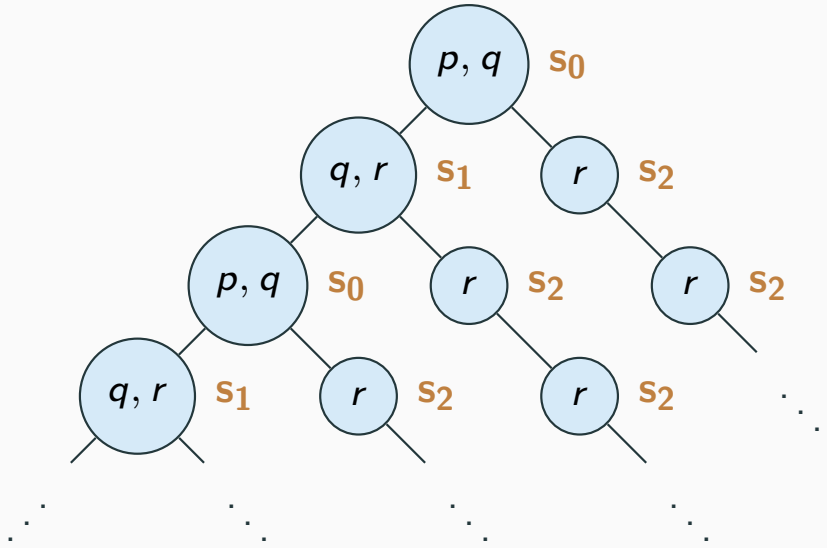


1. $\mathcal{M}, s_0 \models p \wedge q$
2. $\mathcal{M}, s_0 \models \neg r$
3. $\mathcal{M}, s_0 \models \bigcirc\ r$

Satisfaction relation

Example

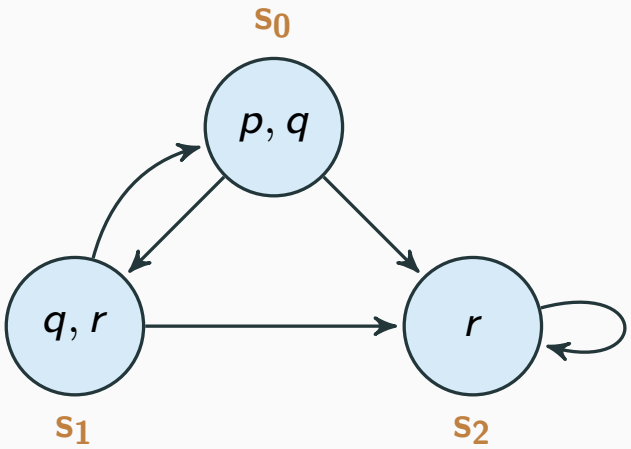


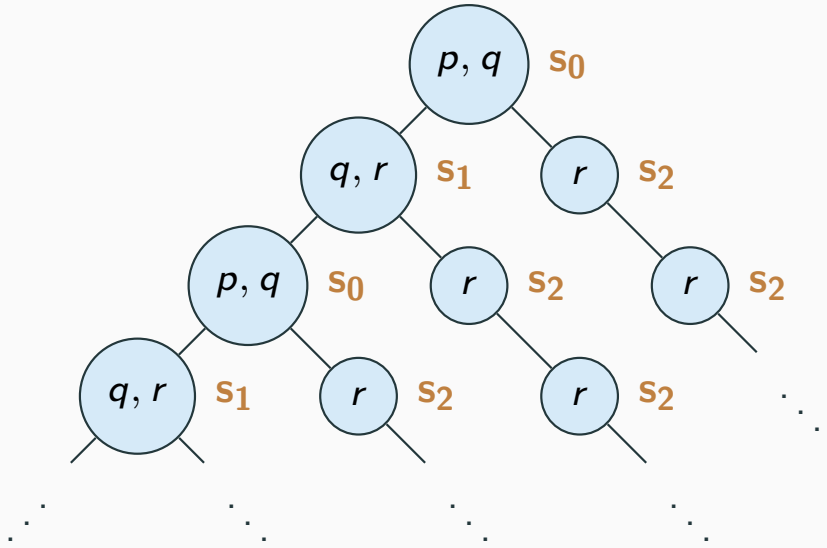


1. $\mathcal{M}, s_0 \models p \wedge q$
2. $\mathcal{M}, s_0 \models \neg r$
3. $\mathcal{M}, s_0 \models \bigcirc\ r$
4. $\mathcal{M}, s_0 \not\models \bigcirc\ (q \wedge r)$

Satisfaction relation

Example

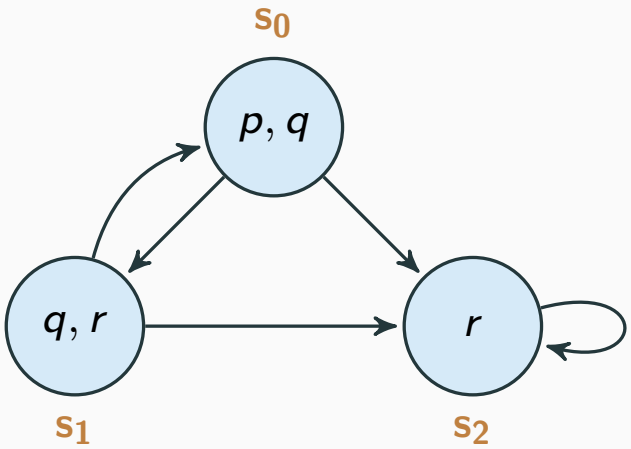


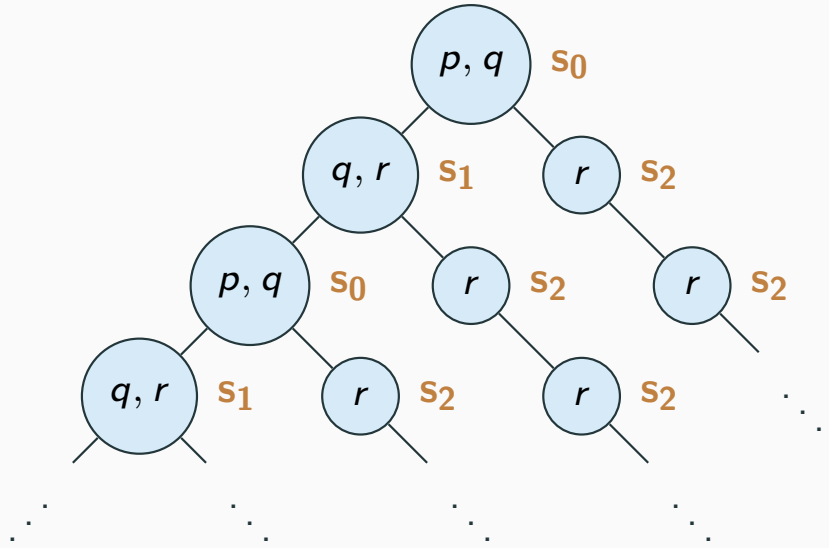


1. $\mathcal{M}, s_0 \models p \wedge q$
2. $\mathcal{M}, s_0 \models \neg r$
3. $\mathcal{M}, s_0 \models \bigcirc\ r$
4. $\mathcal{M}, s_0 \not\models \bigcirc\ (q \wedge r)$
5. $\mathcal{M}, s_0 \models \Box\ \neg(p \wedge r)$

Satisfaction relation

Example

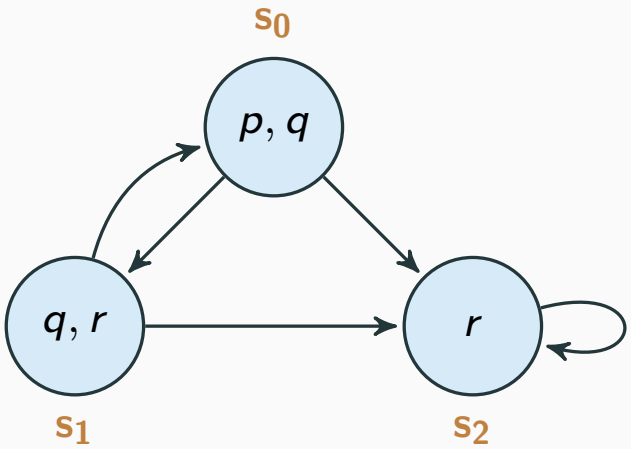


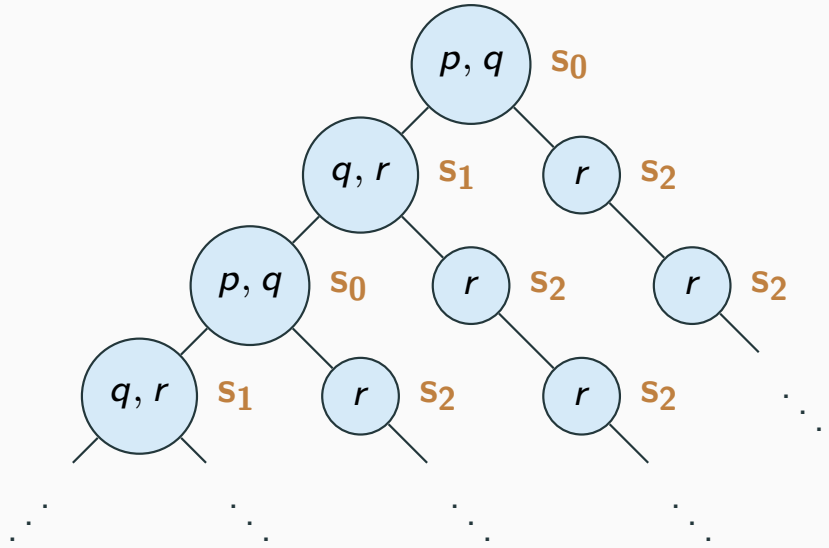


1. $\mathcal{M}, s_0 \models p \wedge q$
2. $\mathcal{M}, s_0 \models \neg r$
3. $\mathcal{M}, s_0 \models \bigcirc\ r$
4. $\mathcal{M}, s_0 \not\models \bigcirc\ (q \wedge r)$
5. $\mathcal{M}, s_0 \models \Box\ \neg(p \wedge r)$
6. $\mathcal{M}, s_2 \models \Box\ r$

Satisfaction relation

Example

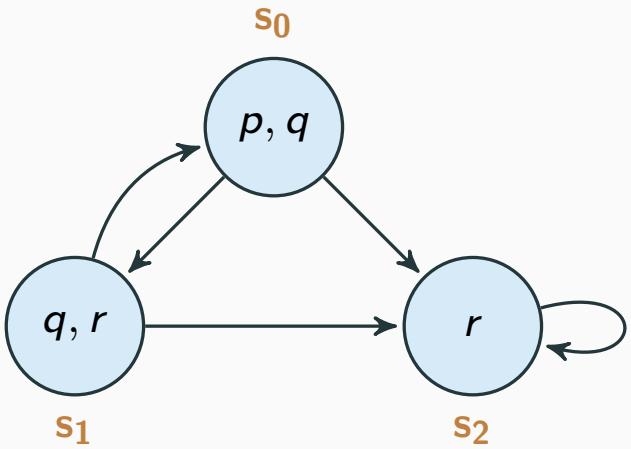


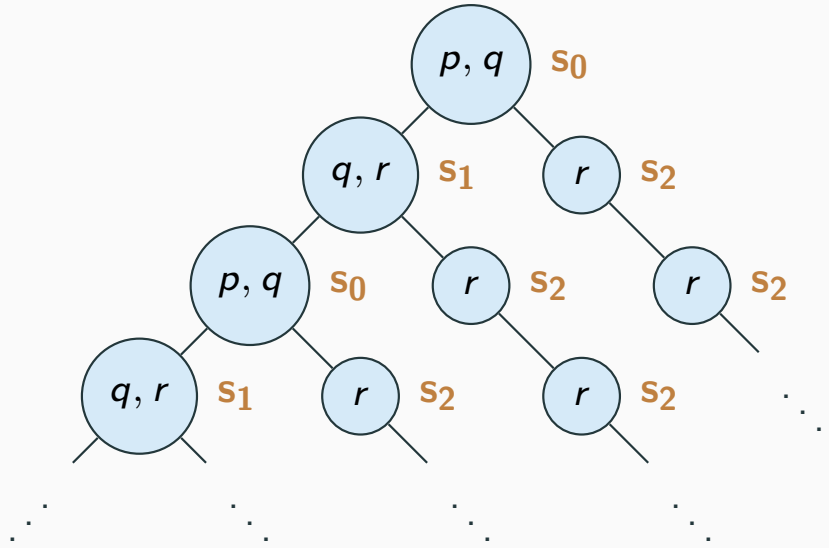


1. $\mathcal{M}, s_0 \models p \wedge q$
2. $\mathcal{M}, s_0 \models \neg r$
3. $\mathcal{M}, s_0 \models \bigcirc\ r$
4. $\mathcal{M}, s_0 \not\models \bigcirc\ (q \wedge r)$
5. $\mathcal{M}, s_0 \models \Box\ \neg(p \wedge r)$
6. $\mathcal{M}, s_2 \models \Box\ r$
7. $\mathcal{M}, s_0 \models$
   $\Diamond\ (\neg q \wedge r) \rightarrow \Diamond\ \Box\ r$

Satisfaction relation

Example

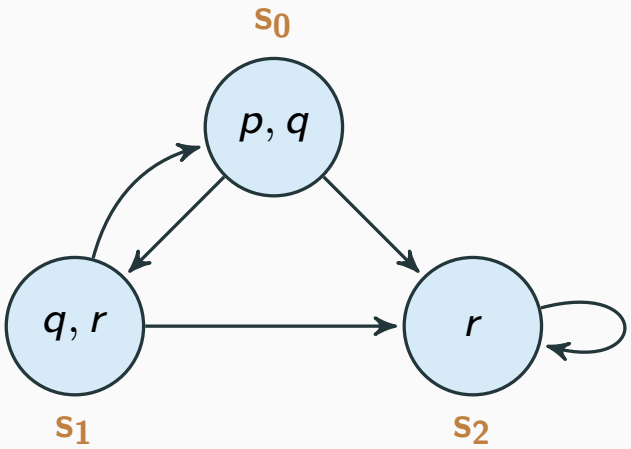


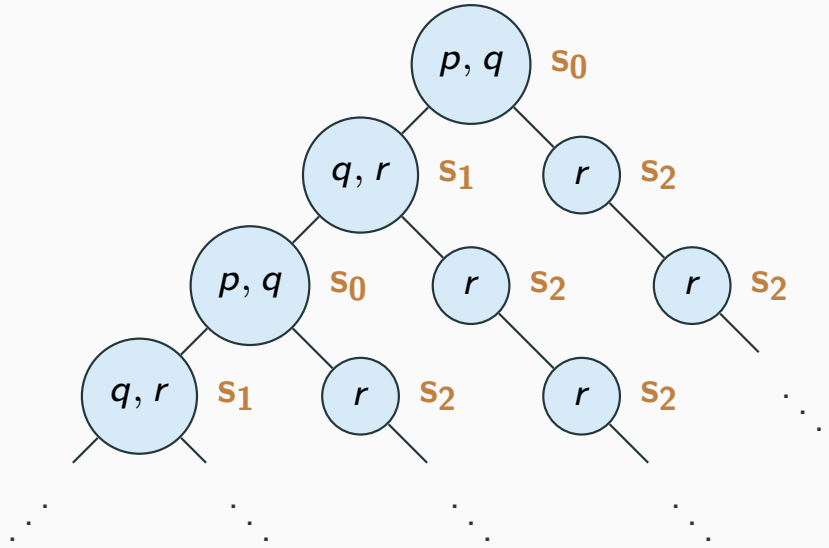


1. $\mathcal{M}, s_0 \models p \wedge q$
2. $\mathcal{M}, s_0 \models \neg r$
3. $\mathcal{M}, s_0 \models \bigcirc\ r$
4. $\mathcal{M}, s_0 \not\models \bigcirc\ (q \wedge r)$
5. $\mathcal{M}, s_0 \models \Box\ \neg(p \wedge r)$
6. $\mathcal{M}, s_2 \models \Box\ r$
7. $\mathcal{M}, s_0 \models$
   $\Diamond\ (\neg q \wedge r) \rightarrow \Diamond\ \Box\ r$
8. $\mathcal{M}, s_0 \not\models \Box\ \Diamond\ p$

Satisfaction relation

Example

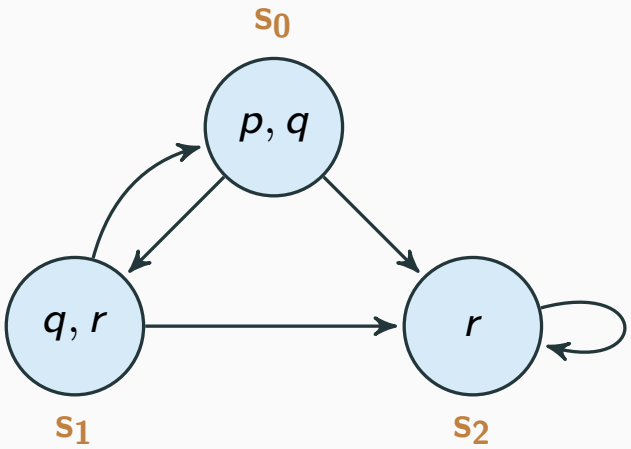


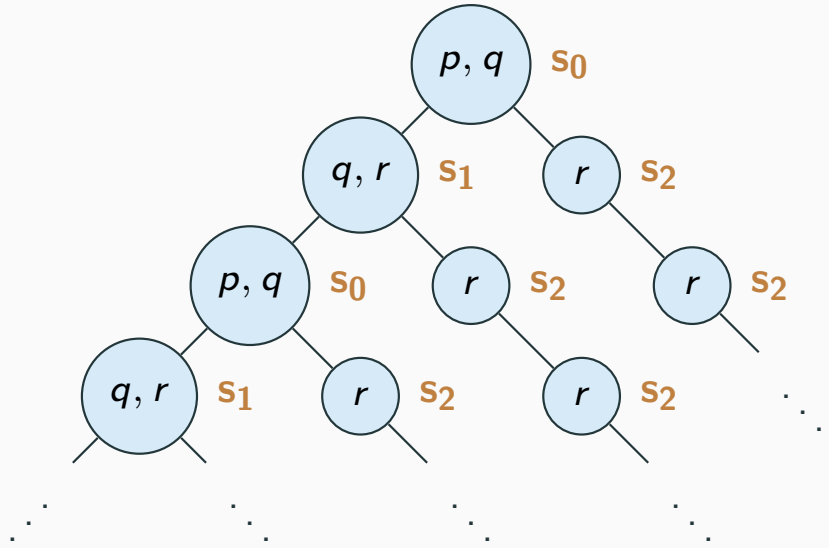


1. $\mathcal{M}, s_0 \models p \wedge q$
2. $\mathcal{M}, s_0 \models \neg r$
3. $\mathcal{M}, s_0 \models \bigcirc\ r$
4. $\mathcal{M}, s_0 \not\models \bigcirc\ (q \wedge r)$
5. $\mathcal{M}, s_0 \models \Box\ \neg(p \wedge r)$
6. $\mathcal{M}, s_2 \models \Box\ r$
7. $\mathcal{M}, s_0 \models$
   $\Diamond\ (\neg q \wedge r) \rightarrow \Diamond\ \Box\ r$
8. $\mathcal{M}, s_0 \not\models \Box\ \Diamond\ p$
9. $\mathcal{M}, s_0 \models \Box\ \Diamond\ p \rightarrow \Box\ \Diamond\ r$

Satisfaction relation

Example





1. $\mathcal{M}, s_0 \models p \wedge q$
2. $\mathcal{M}, s_0 \models \neg r$
3. $\mathcal{M}, s_0 \models \bigcirc\ r$
4. $\mathcal{M}, s_0 \not\models \bigcirc\ (q \wedge r)$
5. $\mathcal{M}, s_0 \models \Box\ \neg(p \wedge r)$
6. $\mathcal{M}, s_2 \models \Box\ r$
7. $\mathcal{M}, s_0 \models$
   $\Diamond\ (\neg q \wedge r) \rightarrow \Diamond\ \Box\ r$
8. $\mathcal{M}, s_0 \not\models \Box\ \Diamond\ p$
9. $\mathcal{M}, s_0 \models \Box\ \Diamond\ p \rightarrow \Box\ \Diamond\ r$
10. $\mathcal{M}, s_0 \not\models \Box\ \Diamond\ r \rightarrow \Box\ \Diamond\ p$

Equivalences

Two formulas are equivalent, denoted $\varphi \equiv \psi$, if they are satisfied by the same models.

Equivalences from Propositional Logic:

$$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi \qquad \neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$$

Equivalences from LTL:

$$\neg \bigcirc \varphi \equiv \bigcirc \neg\varphi \qquad \neg\Box\varphi \equiv \Diamond\neg\varphi \qquad \neg\Diamond\varphi \equiv \Box\neg\varphi$$

Distributive laws:

$$\Box(\varphi \wedge \psi) \equiv \Box\varphi \wedge \Box\psi \qquad \Diamond(\varphi \vee \psi) \equiv \Diamond\varphi \vee \Diamond\psi$$

Equivalences

Inter-definitions:

$$\Diamond\varphi \equiv \neg\Box\neg\varphi \qquad \Box\varphi \equiv \neg\Diamond\neg\varphi \qquad \Diamond\varphi \equiv \top \mathbf{U}\, \varphi$$

Idempotency:

$$\Diamond\Diamond\varphi \equiv \Diamond\varphi \qquad \Box\Box\varphi \equiv \Box\varphi$$

Some more surprising equivalences:

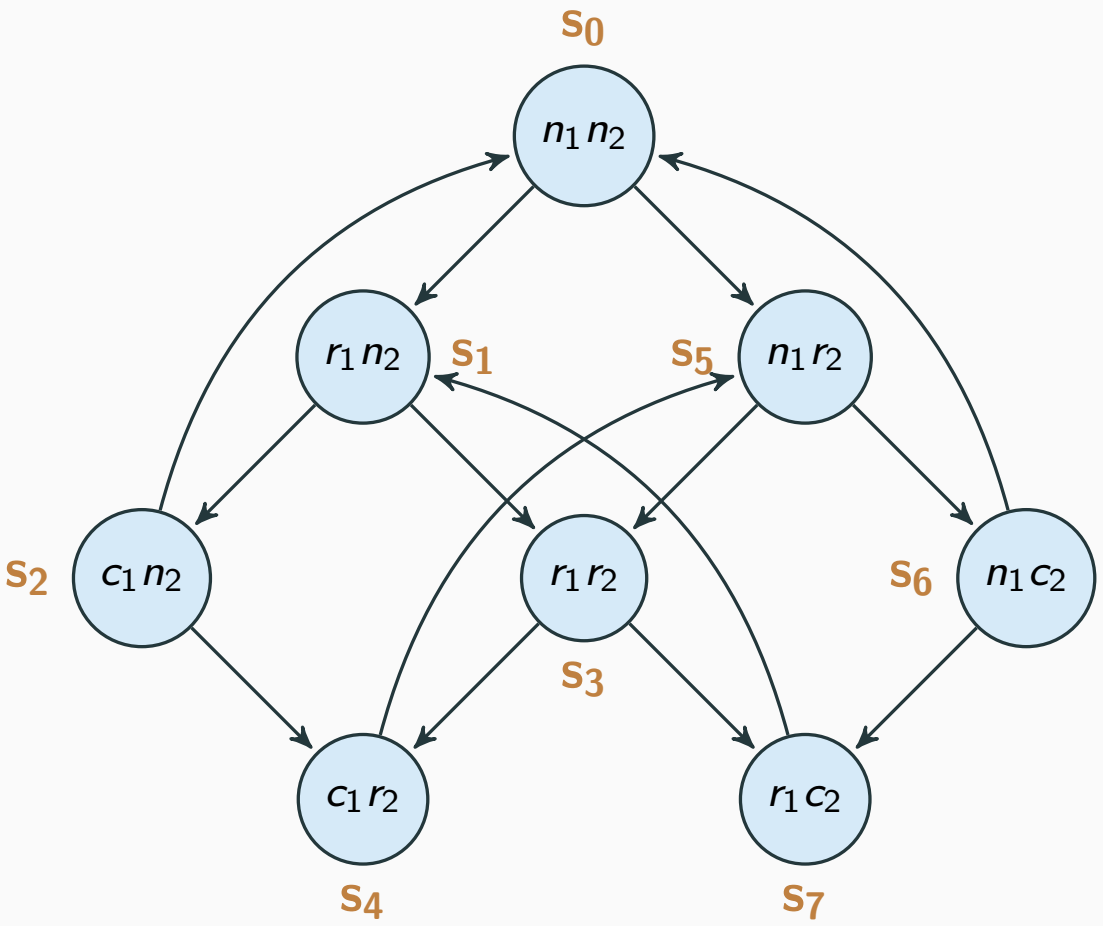
$$\Box\Diamond\Box\varphi \equiv \Diamond\Box\varphi \qquad \Diamond\Box\Diamond\varphi \equiv \Box\Diamond\varphi \qquad \Box(\Diamond\varphi \vee \Diamond\varphi) \equiv \Box\Diamond\varphi \vee \Box\Diamond\psi$$

Exercise: Prove these equivalences.

Satisfaction relation

Exercise: In the example with the two processes executed in parallel, establish if the following properties are expressible in LTL and if yes, if they hold:

- the safety property
- the liveness property
- the non-blocking property



Example

- Safety: only one process may execute critical section code at any point.

$$\Box \neg (c_1 \wedge c_2)$$

  It is satisfied in every state.

Example

- Safety: only one process may execute critical section code at any point.

  $$\Box \neg(c_1 \wedge c_2)$$

  It is satisfied in every state.

- Liveness: whenever a process requests to enter its critical section, it will eventually be allowed to do so.

  $$\Box(r_1 \rightarrow \Diamond c_1)$$

  However, $\mathcal{M}, s_0 \not\models \Box(r_1 \rightarrow \Diamond c_1)$. A counterexample path is $s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow \ldots$.

Example

- Safety: only one process may execute critical section code at any point.

  $$\Box \neg(c_1 \wedge c_2)$$

  It is satisfied in every state.

- Liveness: whenever a process requests to enter its critical section, it will eventually be allowed to do so.

  $$\Box(r_1 \rightarrow \Diamond c_1)$$

  However, $\mathcal{M}, s_0 \not\models \Box(r_1 \rightarrow \Diamond c_1)$. A counterexample path is $s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow \ldots$.

- Non-blocking: a process can always request to enter its critical section. This property cannot be expressed in LTL!
  We need to express *for every state satisfying n_1, there is a state satisfying r_1*. Unfortunately, this existence quantifier on paths (*there is a state satisfying...*) cannot be expressed in LTL.

CTL Logic

Syntax

CTL = Computation Tree Logic

Assume some set *Atoms* of atomic propositions.

CTL formulas are defined by:

$$\varphi \;:=\; p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \to \varphi \mid$$
$$\mathbf{AX}\varphi \mid \mathbf{EX}\varphi \mid \mathbf{AF}\varphi \mid \mathbf{EF}\varphi \mid \mathbf{AG}\varphi \mid \mathbf{EG}\varphi \mid A[\varphi \mathbf{U} \varphi] \mid E[\varphi \mathbf{U} \varphi]$$

Each temporal connective is a pair of a path quantifier:

- **A** – for all paths
- **E** – there exists a path

and an LTL-like temporal operator $\bigcirc$, $\Diamond$, $\Box$, **U**.

Precedence (high-to low): $(\mathbf{AX}, \mathbf{EX}, \mathbf{AF}, \mathbf{EF}, \mathbf{AG}, \mathbf{EG}, \neg)$, $(\mathbf{AU}, \mathbf{EU})$, $(\wedge, \vee)$, $\to$

Syntax

Example
The following are CTL formulas:

- $\mathbf{AG}(q \rightarrow \mathbf{EG}\ r)$
- $\mathbf{EF}\ \mathbf{E}[r\ \mathbf{U}\ q]$
- $\mathbf{A}[p\ \mathbf{U}\ \mathbf{EF}\ r]$
- $\mathbf{EF}\ \mathbf{EG}\ p \rightarrow \mathbf{AF}\ r$

Syntax

Example
The following are CTL formulas:

- $\mathbf{AG}(q \rightarrow \mathbf{EG}\ r)$
- $\mathbf{EF}\ \mathbf{E}[r\ \mathbf{U}\ q]$
- $\mathbf{A}[p\ \mathbf{U}\ \mathbf{EF}\ r]$
- $\mathbf{EF}\ \mathbf{EG}\ p \rightarrow \mathbf{AF}\ r$

The following are not CTL formulas:

- $\mathbf{EF}\ \Box\ r$
- $\mathbf{A}\neg\Box\neg p$
- $\Diamond[r\ \mathbf{U}\ q]$
- $\mathbf{EF}(r\ \mathbf{U}\ q)$

Transition Systems and Paths

(This is the same as for LTL)

A transition system (or model) $\mathcal{M} = (S, \rightarrow, L)$ consists of

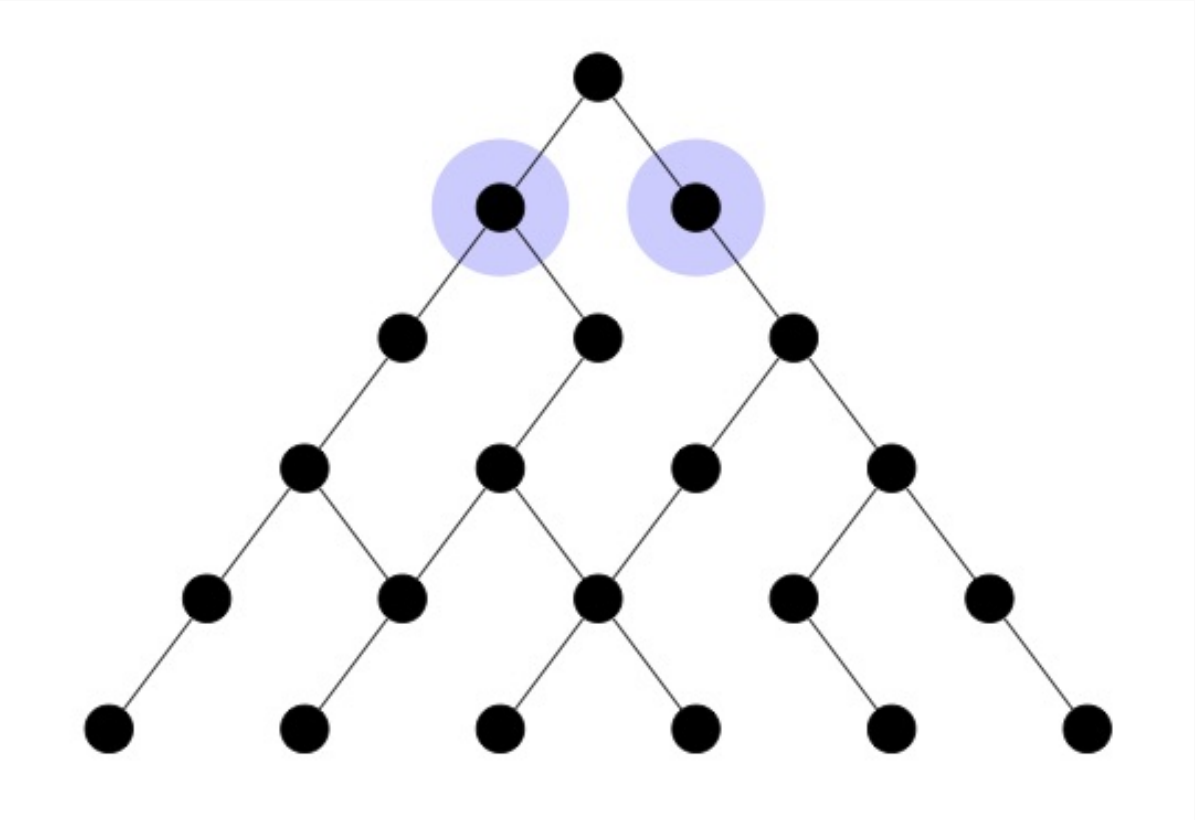
- S a finite set of states
- $\rightarrow \subseteq S \times S$ a transition relation
- $L : S \rightarrow \mathcal{P}(Atoms)$ a labelling function

such that for all $\forall s_1 \in S$ there exists $s_2 \in S$ with $s_1 \rightarrow s_2$ (*serial condition*).

A path π in a transition system $\mathcal{M} = (S, \rightarrow, L)$ is an infinite sequence of states $s_0, s_1, s_2, \ldots$ such that for all $i \geq 0$, $s_i \rightarrow s_{i+1}$.

Paths are written as $\pi = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$

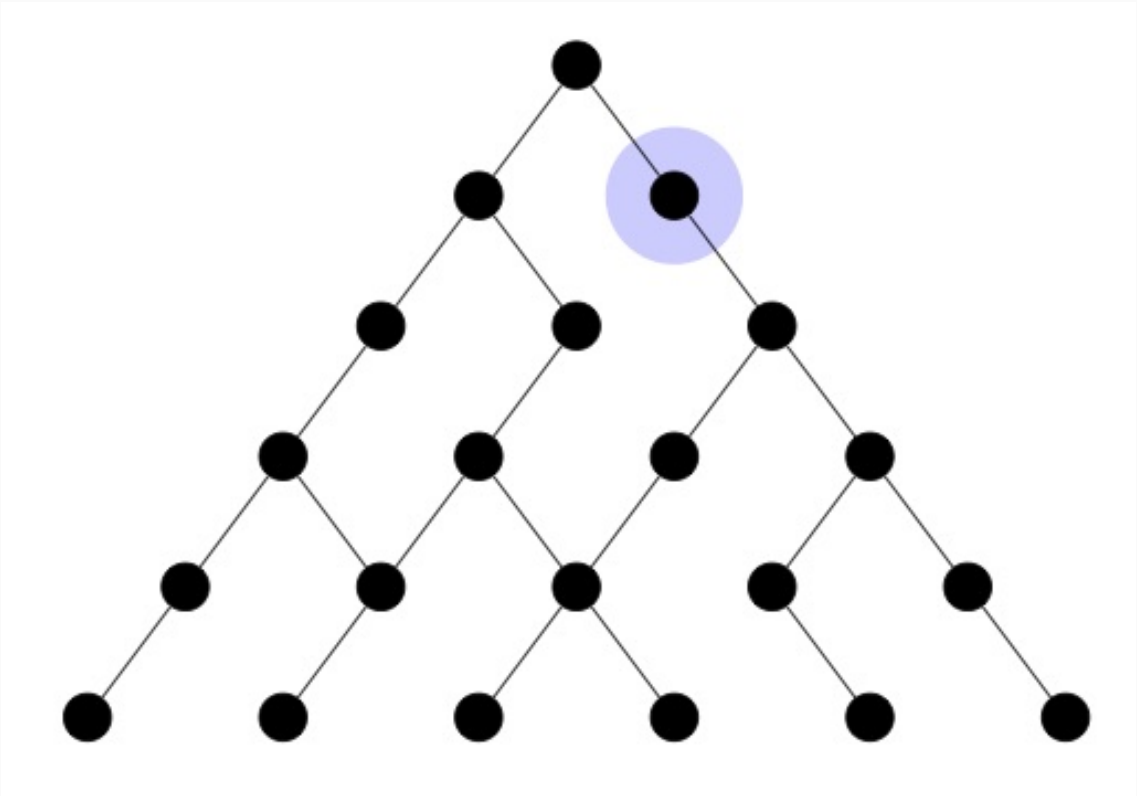
CTL in pictures



AX φ

For every next state, φ holds.

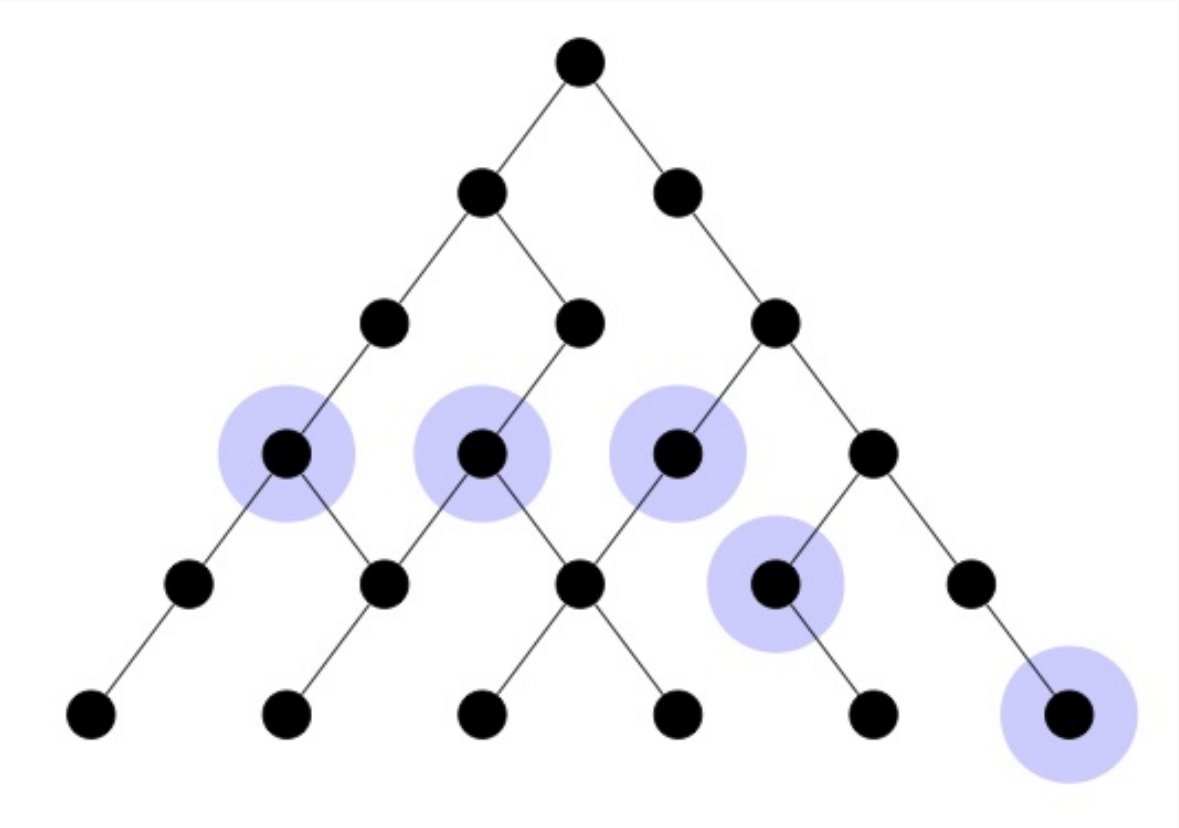
CTL in pictures



EX φ

There exists a next state where φ holds.

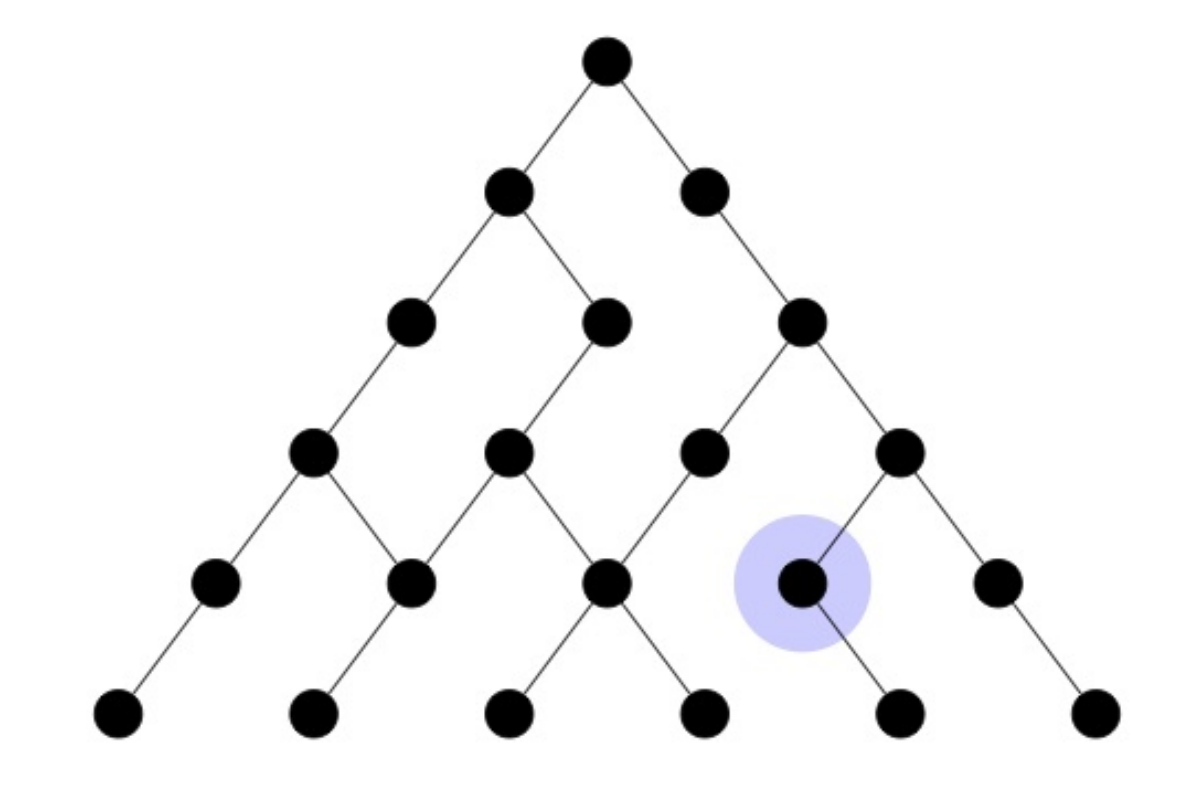
CTL in pictures



AF φ

For all paths, there exists a future state where φ holds.

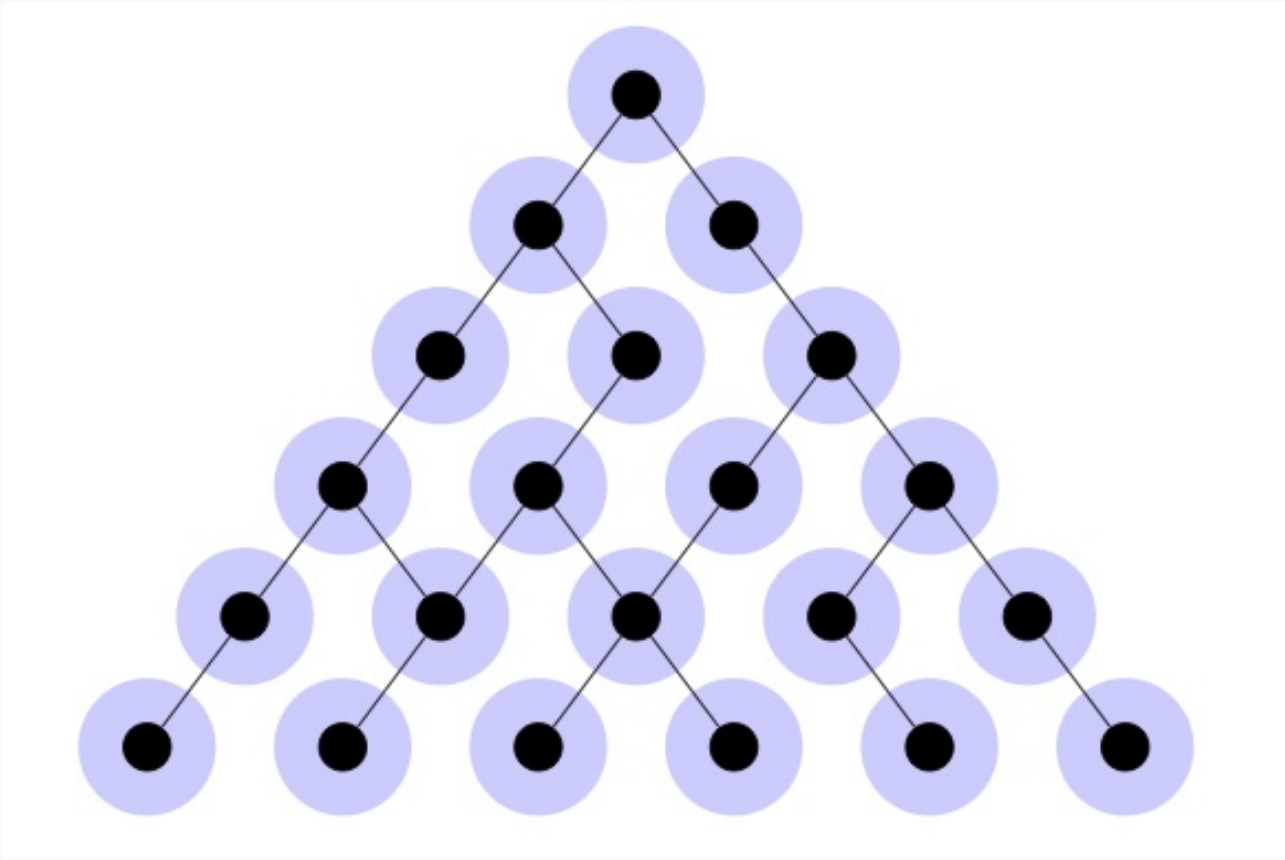
CTL in pictures



EF φ

There exists a path with a future state where φ holds.

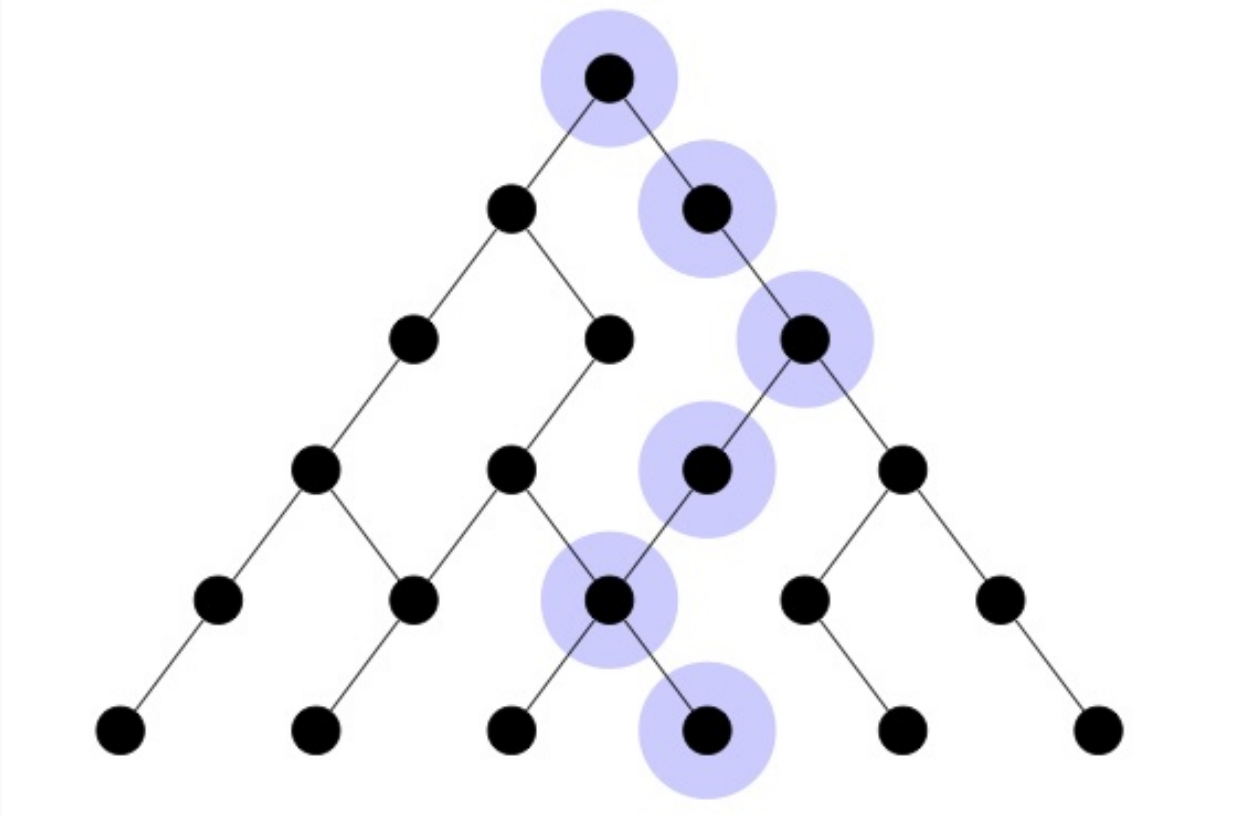
CTL in pictures



AG φ

For all paths, for all states along them, φ holds.

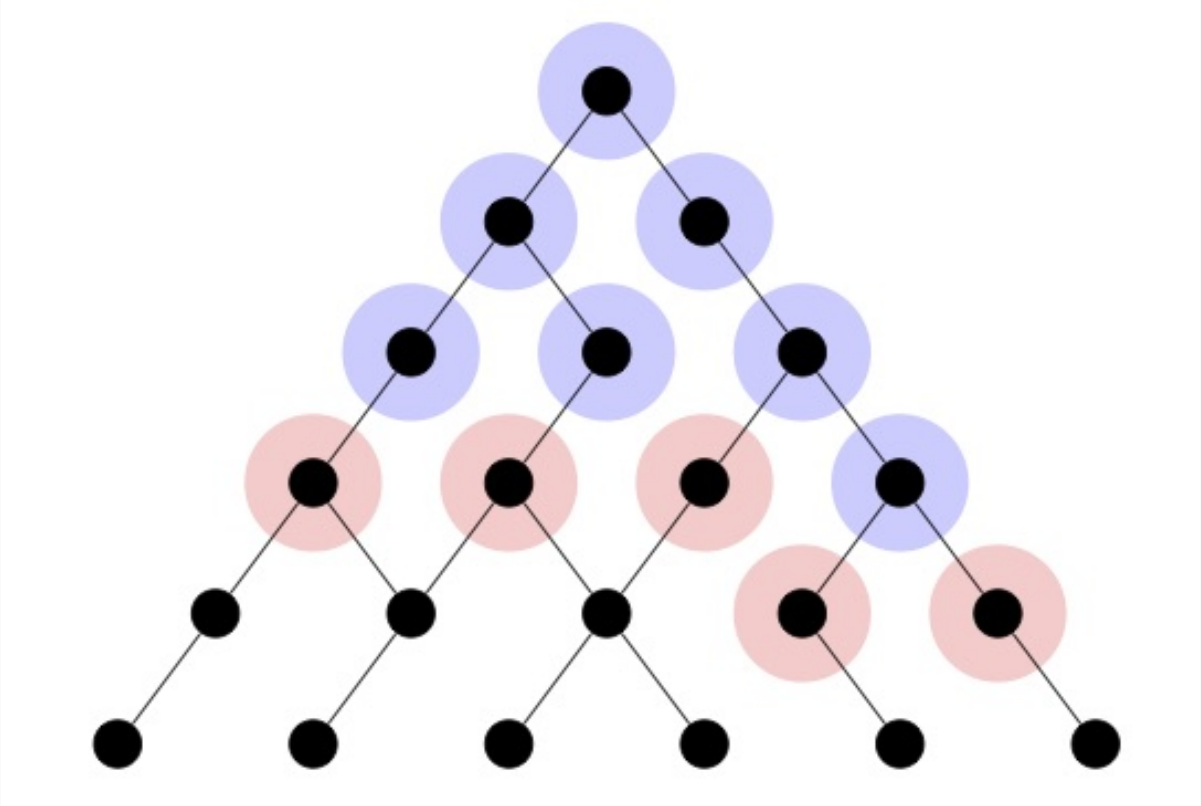
CTL in pictures



EG φ

There exists a path such that, for all states along it, φ holds.

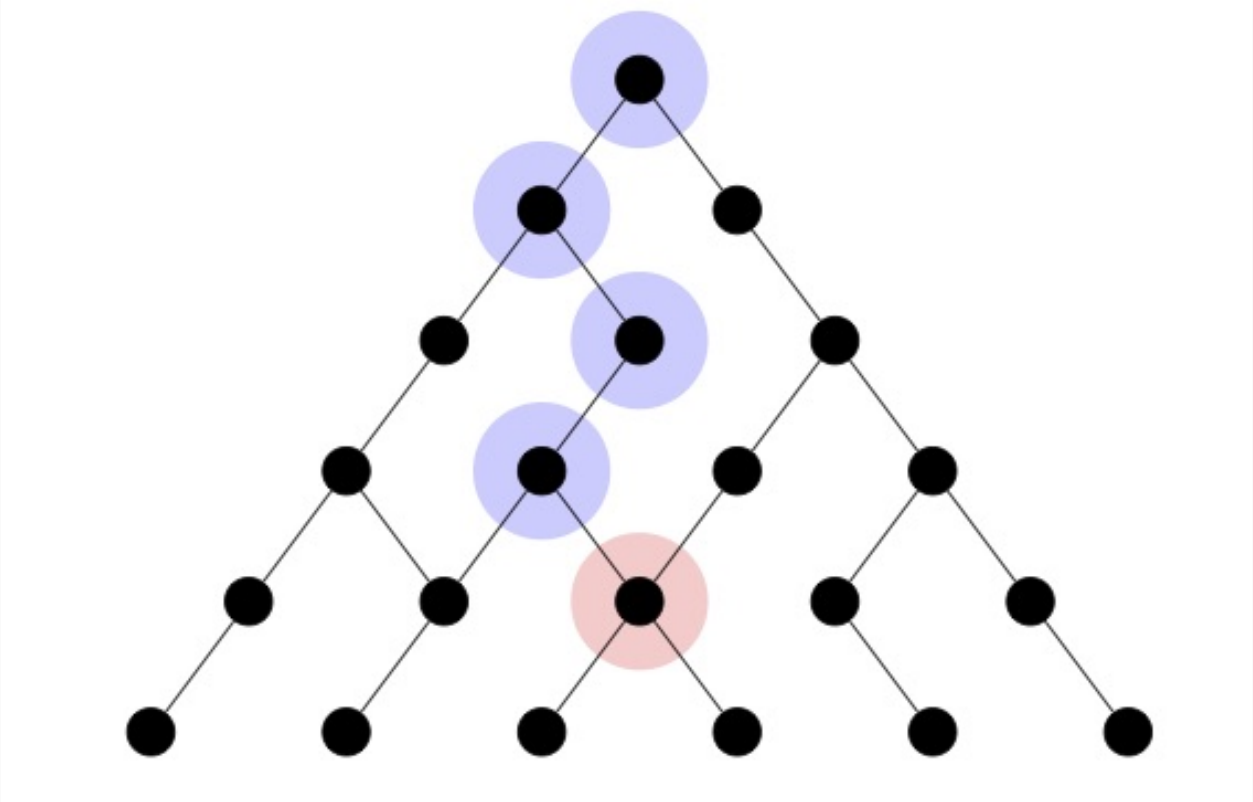
CTL in pictures



$\mathbf{A}[\varphi\ \mathbf{U}\ \psi]$

For all paths, ψ eventually holds, and φ holds at all states earlier.

CTL in pictures



E[φ **U** ψ]

There exists a path wehre ψ eventually holds, and φ holds at all states earlier.

Informal semantics

Example

EF φ

- there exists a future state where eventually φ is true

AG AF φ

- for all future states, φ will eventually hold

AG$(\varphi \rightarrow$ **AF** $\psi)$

- for all future states, if φ holds, then ψ will eventually hold

Informal semantics

Example

AG$(\varphi \rightarrow \mathbf{E}[\varphi \mathbf{U} \psi])$

- for all future states, if φ holds, then there is a future where ψ will eventually hold, and φ holds for all points in between

AG$(\varphi \rightarrow \mathbf{EG}\, \psi)$

- for all future states, if φ holds, then there is a future where ψ always holds

EF AG φ

- there exists a possible state in the future from where φ is always true

Satisfaction Relation

"The state s of the model $\mathcal{M}$ satisfies the CTL formula φ"

$$\mathcal{M}, s \models \varphi$$

$\mathcal{M}, s \models \top$

$\mathcal{M}, s \not\models \bot$

$\mathcal{M}, s \models p$	iff	$p \in L(s)$
$\mathcal{M}, s \models \varphi \wedge \psi$	iff	$\mathcal{M}, s \models \varphi$ and $\mathcal{M}, s \models \psi$
$\mathcal{M}, s \models \varphi \vee \psi$	iff	$\mathcal{M}, s \models \varphi$ or $\mathcal{M}, s \models \psi$
$\mathcal{M}, s \models \varphi \rightarrow \psi$	iff	$\mathcal{M}, s \models \varphi$ implies $\mathcal{M}, s \models \psi$
$\mathcal{M}, s \models \mathbf{AX}\ \varphi$	iff	for all $s' \in S$ such that $s \rightarrow s'$ we have $\mathcal{M}, s' \models \varphi$
$\mathcal{M}, s \models \mathbf{EX}\ \varphi$	iff	there exists $s' \in S$ such that $s \rightarrow s'$ and $\mathcal{M}, s' \models \varphi$

Satisfaction Relation

Assume that $\pi = s_0 \to s_1 \to s_2 \to \dots$

$\mathcal{M}, s \models \mathbf{AF}\ \varphi$	iff	for all paths π such that $s_0 = s$ there exists i such that $\mathcal{M}, s_i \models \varphi$
$\mathcal{M}, s \models \mathbf{EF}\ \varphi$	iff	there exists a path π such that $s_0 = s$ and there exists i such that $\mathcal{M}, s_i \models \varphi$
$\mathcal{M}, s \models \mathbf{AG}\ \varphi$	iff	for all paths π such that $s_0 = s$ for all i, $\mathcal{M}, s_i \models \varphi$
$\mathcal{M}, s \models \mathbf{EG}\ \varphi$	iff	there exists a path π such that $s_0 = s$ and for all i, $\mathcal{M}, s_i \models \varphi$
$\mathcal{M}, s \models A[\varphi_1 \mathbf{U}\ \varphi_2]$	iff	for all paths π such that $s_0 = s$ there exists i such that $\mathcal{M}, s_i \models \varphi_2$ and for all $j < i$, $\mathcal{M}, s_j \models \varphi_1$
$\mathcal{M}, s \models E[\varphi_1 \mathbf{U}\ \varphi_2]$	iff	there exists a path π such that $s_0 = s$ and there exists i such that $\mathcal{M}, s_i \models \varphi_2$ and for all $j < i$, $\mathcal{M}, s_j \models \varphi_1$

CTL - Equivalences

$$
\begin{aligned}
\neg \mathbf{EX}\, \varphi &\equiv \mathbf{AX}\, \neg\varphi \\
\neg \mathbf{EF}\, \varphi &\equiv \mathbf{AG}\, \neg\varphi \\
\neg \mathbf{EG}\, \varphi &\equiv \mathbf{AF}\, \neg\varphi \\
\mathbf{AF}\, \varphi &\equiv \mathbf{A}[\top\, \mathbf{U}\, \varphi] \\
\mathbf{EF}\, \varphi &\equiv \mathbf{E}[\top\, \mathbf{U}\, \varphi] \\
\mathbf{A}[\varphi\, \mathbf{U}\, \psi] &\equiv \neg(\mathbf{E}[\neg\psi\, \mathbf{U}\, (\neg\varphi \wedge \neg\psi)] \vee \mathbf{EG}\, \neg\psi)
\end{aligned}
$$

From these, one can show that the sets $\{\mathbf{AU}, \mathbf{EU}, \mathbf{EX}\}$ and $\{\mathbf{EU}, \mathbf{EG}, \mathbf{EX}\}$ are both adequate sets of temporal connectives.

Differences between LTL and CTL

LTL allows questions of the form:

- For all paths, does the LTL formula φ hold?
- Intuitively, in LTL we cannot fix a path π and talk about it.

CTL allows mixing of path quantifiers:

- $\mathbf{AG}(p \rightarrow \mathbf{EG}\, q)$
  for all paths, if p is true then there is a path on which q is always true.

Going back to the Example

- Safety: only one process may execute critical section code at any point.

$$\mathbf{AG}\,\neg(c_1 \wedge c_2)$$

  It is satisfied in every state.

- Liveness: whenever a process requests to enter its critical section, it will eventually be allowed to do so.

$$\mathbf{AG}(r_1 \rightarrow \mathbf{AF}c_1) \wedge \mathbf{AG}(r_2 \rightarrow \mathbf{AF}c_2)$$

- Non-blocking: a process can always request to enter its critical section.

$$\mathbf{AG}(n_1 \rightarrow \mathbf{EX}r_1) \wedge \mathbf{AG}(n_2 \rightarrow \mathbf{EX}r_2)$$

Model checking algorithm for CTL

Model checking - The big picture

- An efficient algorithm to decide $\mathcal{M}, s \models \varphi$
- Typically without user interaction (fully automated)
- Different approaches: semantic, automata, tableau,...
- Provide a counterexample when $\mathcal{M}, s \not\models \varphi$.
  The counterexample provides a clue to what is wrong:
  - The system might be incorrect
  - The model might be too abstract (in need of refinement)
  - The specification might not be the intended one

CTL model checking

- We present an algorithm which, given a model and a CTL formula, outputs the set of states of the model that satisfy the formula.
- The algorithm handles CTL formulas containing the connectives

$$\{\bot, \neg, \wedge, \mathbf{AF}, \mathbf{EU}, \mathbf{EX}\}$$

- Given an arbitrary CTL formula φ, we simply pre-process φ in order to write it in an equivalent form in terms of the above set of connectives.

The labelling algorithm

INPUT: a CTL model $\mathcal{M} = (S, \rightarrow, L)$ and a CTL formula φ

OUTPUT: the set of states of $\mathcal{M}$ which satisfy φ

The labelling algorithm

INPUT: a CTL model $\mathcal{M} = (S, \to, L)$ and a CTL formula φ

OUTPUT: the set of states of $\mathcal{M}$ which satisfy φ

1. Write φ in terms of the connectives $\{\bot, \neg, \wedge, \mathbf{AF}, \mathbf{EU}, \mathbf{EX}\}$ using the equivalences in CTL

The labelling algorithm

INPUT: a CTL model $\mathcal{M} = (S, \to, L)$ and a CTL formula φ

OUTPUT: the set of states of $\mathcal{M}$ which satisfy φ

1. Write φ in terms of the connectives $\{\bot, \neg, \wedge, \mathbf{AF}, \mathbf{EU}, \mathbf{EX}\}$ using the equivalences in CTL

2. Label the states of $\mathcal{M}$ with subformulas of φ that are satisfied there, starting with the smallest subformulas and working outwards towards φ.

The labelling algorithm

INPUT: a CTL model $\mathcal{M} = (S, \to, L)$ and a CTL formula φ

OUTPUT: the set of states of $\mathcal{M}$ which satisfy φ

1. Write φ in terms of the connectives $\{\bot, \neg, \wedge, \mathbf{AF}, \mathbf{EU}, \mathbf{EX}\}$ using the equivalences in CTL

2. Label the states of $\mathcal{M}$ with subformulas of φ that are satisfied there, starting with the smallest subformulas and working outwards towards φ.

Suppose ψ is a subformula of φ. If ψ is

The labelling algorithm

INPUT: a CTL model $\mathcal{M} = (S, \to, L)$ and a CTL formula φ

OUTPUT: the set of states of $\mathcal{M}$ which satisfy φ

1. Write φ in terms of the connectives $\{\bot, \neg, \wedge, \mathbf{AF}, \mathbf{EU}, \mathbf{EX}\}$ using the equivalences in CTL

2. Label the states of $\mathcal{M}$ with subformulas of φ that are satisfied there, starting with the smallest subformulas and working outwards towards φ.

Suppose ψ is a subformula of φ. If ψ is

- $\bot$: then no states are labelled with $\bot$

The labelling algorithm

INPUT: a CTL model $\mathcal{M} = (S, \to, L)$ and a CTL formula φ

OUTPUT: the set of states of $\mathcal{M}$ which satisfy φ

1. Write φ in terms of the connectives $\{\bot, \neg, \wedge, \mathbf{AF}, \mathbf{EU}, \mathbf{EX}\}$ using the equivalences in CTL

2. Label the states of $\mathcal{M}$ with subformulas of φ that are satisfied there, starting with the smallest subformulas and working outwards towards φ.

Suppose ψ is a subformula of φ. If ψ is

- $\bot$: then no states are labelled with $\bot$
- p: then label s with p if $p \in L(s)$

The labelling algorithm

INPUT: a CTL model $\mathcal{M} = (S, \to, L)$ and a CTL formula φ

OUTPUT: the set of states of $\mathcal{M}$ which satisfy φ

1. Write φ in terms of the connectives $\{\bot, \neg, \wedge, \mathbf{AF}, \mathbf{EU}, \mathbf{EX}\}$ using the equivalences in CTL

2. Label the states of $\mathcal{M}$ with subformulas of φ that are satisfied there, starting with the smallest subformulas and working outwards towards φ.

Suppose ψ is a subformula of φ. If ψ is

- $\bot$: then no states are labelled with $\bot$
- p: then label s with p if $p \in L(s)$
- $\psi_1 \wedge \psi_2$: label s with $\psi_1 \wedge \psi_2$ if s is already labelled with ψ_1 and ψ_2

The labelling algorithm

INPUT: a CTL model $\mathcal{M} = (S, \to, L)$ and a CTL formula φ

OUTPUT: the set of states of $\mathcal{M}$ which satisfy φ

1. Write φ in terms of the connectives $\{\bot, \neg, \wedge, \mathbf{AF}, \mathbf{EU}, \mathbf{EX}\}$ using the equivalences in CTL

2. Label the states of $\mathcal{M}$ with subformulas of φ that are satisfied there, starting with the smallest subformulas and working outwards towards φ.

Suppose ψ is a subformula of φ. If ψ is

- $\bot$: then no states are labelled with $\bot$
- p: then label s with p if $p \in L(s)$
- $\psi_1 \wedge \psi_2$: label s with $\psi_1 \wedge \psi_2$ if s is already labelled with ψ_1 and ψ_2
- $\neg\psi_1$: label s with $\neg\psi_1$ if s is not already labelled with ψ_1

The labelling algorithm

- **EX** ψ_1: label a state with **EX** ψ_1 if one of its successors is labelled with ψ_1

The labelling algorithm

- **EX** ψ_1: label a state with **EX** ψ_1 if one of its successors is labelled with ψ_1
- **AF** ψ_1:
  - If any state s is labelled with ψ_1, label it with **AF** ψ_1
  - **Repeat:** label any state with **AF** ψ_1 if all its successor states are labelled with **AF** ψ_1, until there is no change.

The labelling algorithm

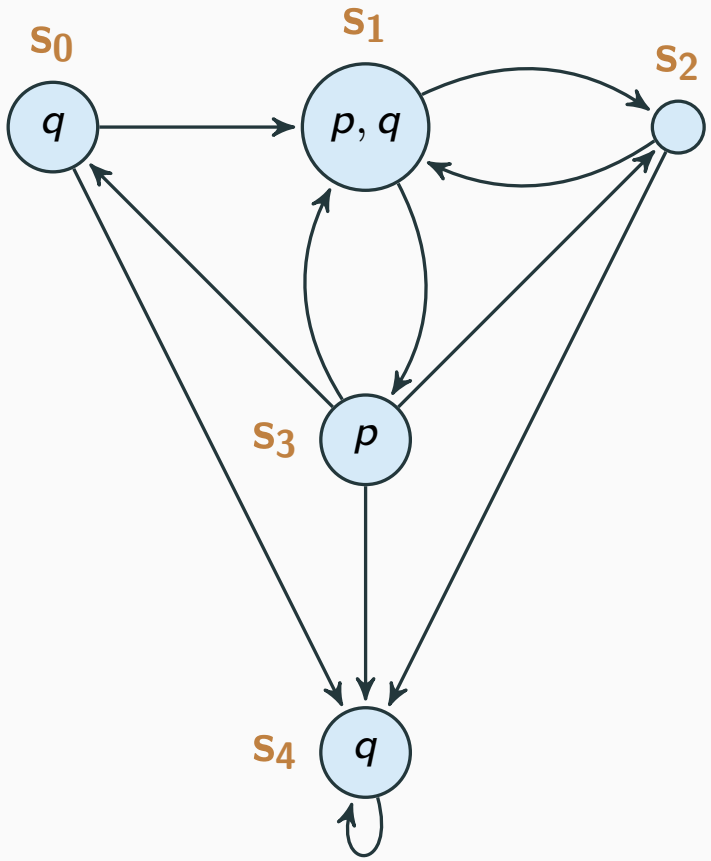
- **EX** ψ_1: label a state with **EX** ψ_1 if one of its successors is labelled with ψ_1
- **AF** ψ_1:
  - If any state s is labelled with ψ_1, label it with **AF** ψ_1
  - **Repeat:** label any state with **AF** ψ_1 if all its successor states are labelled with **AF** ψ_1, until there is no change.
- **E**$[\psi_1$ **U** $\psi_2]$:
  - If any state s is labelled with ψ_2, label it with **E**$[\psi_1$ **U** $\psi_2]$
  - **Repeat:** label any state with **E**$[\psi_1$ **U** $\psi_2]$ if it is labelled with ψ_1 and at least one of its successors is labelled with **E**$[\psi_1$ **U** $\psi_2]$, until there is no change.

The labelling algorithm

Example

$\mathbf{AG}(\mathbf{q} \rightarrow \mathbf{EG\ p}) \equiv \neg\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$

- Label q: $\{s_0, s_1, s_4\}$
- Label p: $\{s_1, s_3\}$

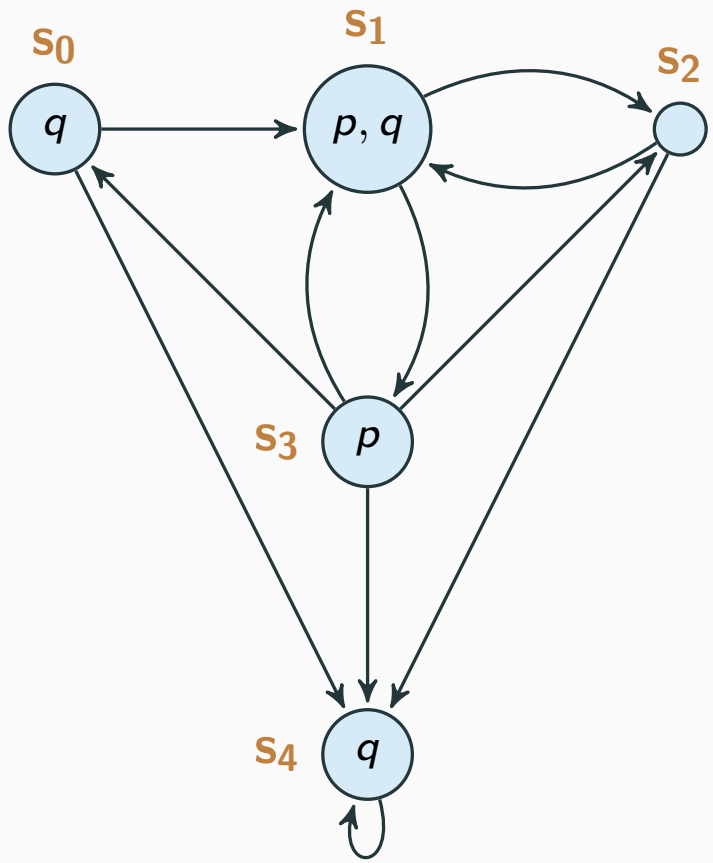


The labelling algorithm

Example

$\mathbf{AG(q \rightarrow EG\ p)} \equiv \neg\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$

- Label q: $\{s_0, s_1, s_4\}$
- Label p: $\{s_1, s_3\}$
- Label $\neg p$: $\{s_0, s_2, s_4\}$

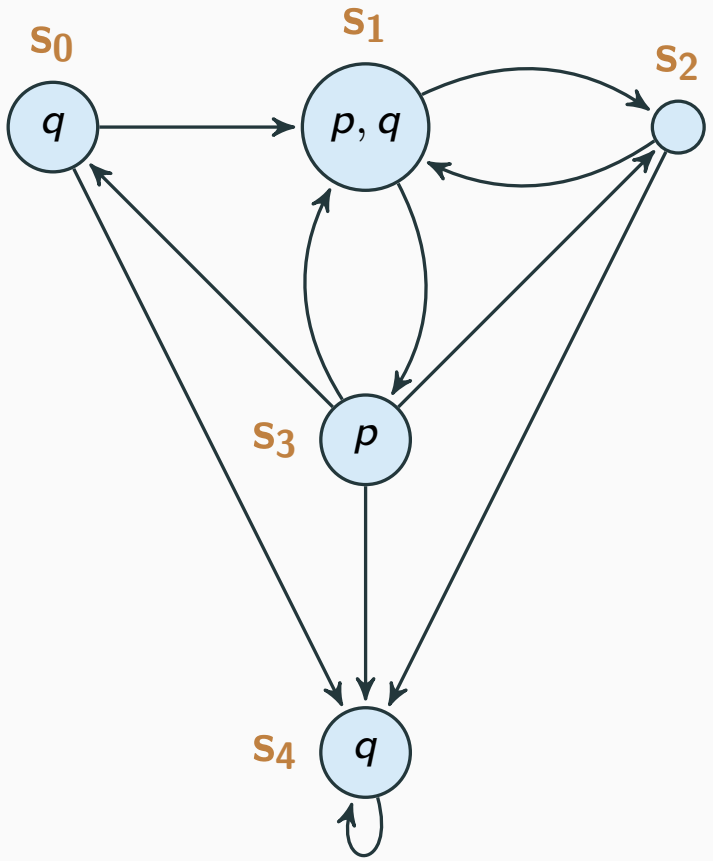


The labelling algorithm

Example

$\mathbf{AG}(\mathbf{q} \rightarrow \mathbf{EG\ p}) \equiv \neg\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$

- Label q: $\{s_0, s_1, s_4\}$
- Label p: $\{s_1, s_3\}$
- Label $\neg p$: $\{s_0, s_2, s_4\}$
- Label $\mathbf{AF}\neg p$: $\{s_0, s_2, s_4\}$
  1. s1 cannot be labeled because of s3
  2. s3 cannot be labeled because of s1

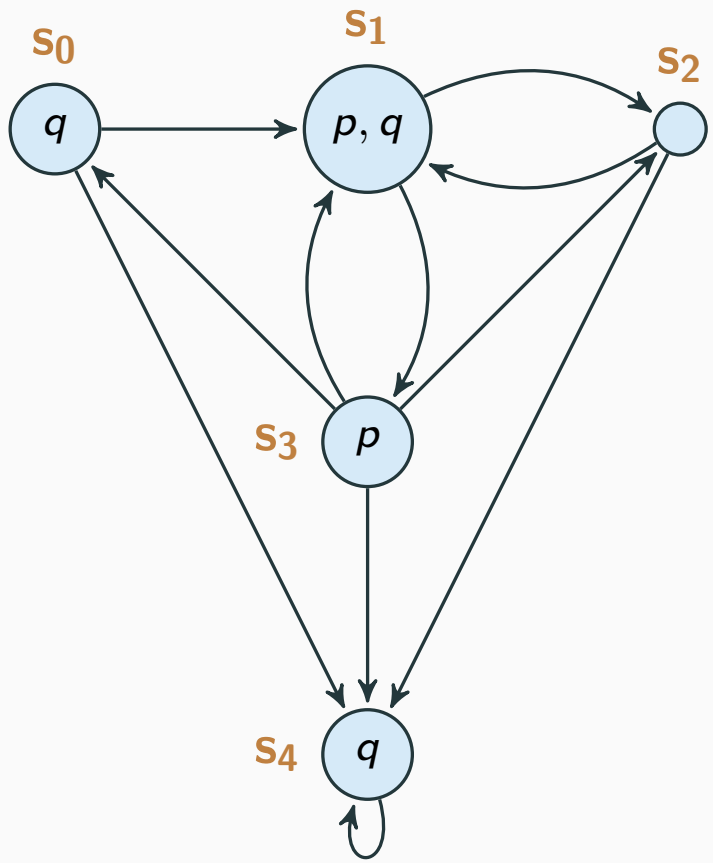


The labelling algorithm

Example

$\mathbf{AG(q \rightarrow EG\ p)} \equiv \neg\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$

- Label q: $\{s_0, s_1, s_4\}$
- Label p: $\{s_1, s_3\}$
- Label $\neg p$: $\{s_0, s_2, s_4\}$
- Label $\mathbf{AF}\neg p$: $\{s_0, s_2, s_4\}$
  1. s1 cannot be labeled because of s3
  2. s3 cannot be labeled because of s1
- Label $q \wedge \mathbf{AF}\neg p$: $\{s_0, s_4\}$

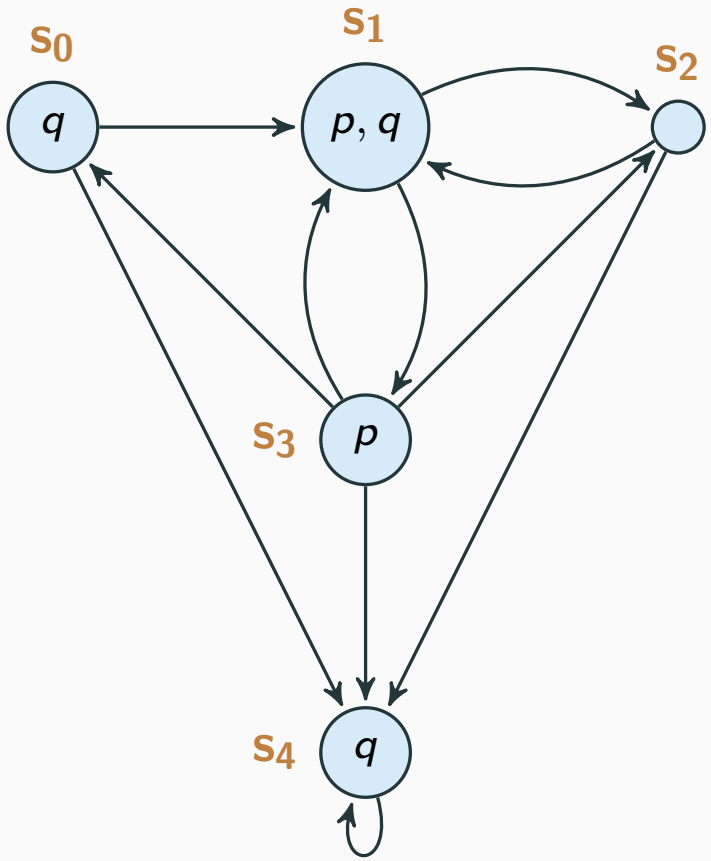


The labelling algorithm

Example

$\mathbf{AG(q \rightarrow EG\ p)} \equiv \neg\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$

- Label q: $\{s_0, s_1, s_4\}$
- Label p: $\{s_1, s_3\}$
- Label $\neg p$: $\{s_0, s_2, s_4\}$
- Label $\mathbf{AF}\neg p$: $\{s_0, s_2, s_4\}$
  1. s1 cannot be labeled because of s3
  2. s3 cannot be labeled because of s1
- Label $q \wedge \mathbf{AF}\neg p$: $\{s_0, s_4\}$
- Label $\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$:
  1. $\{s_0, s_4\}$

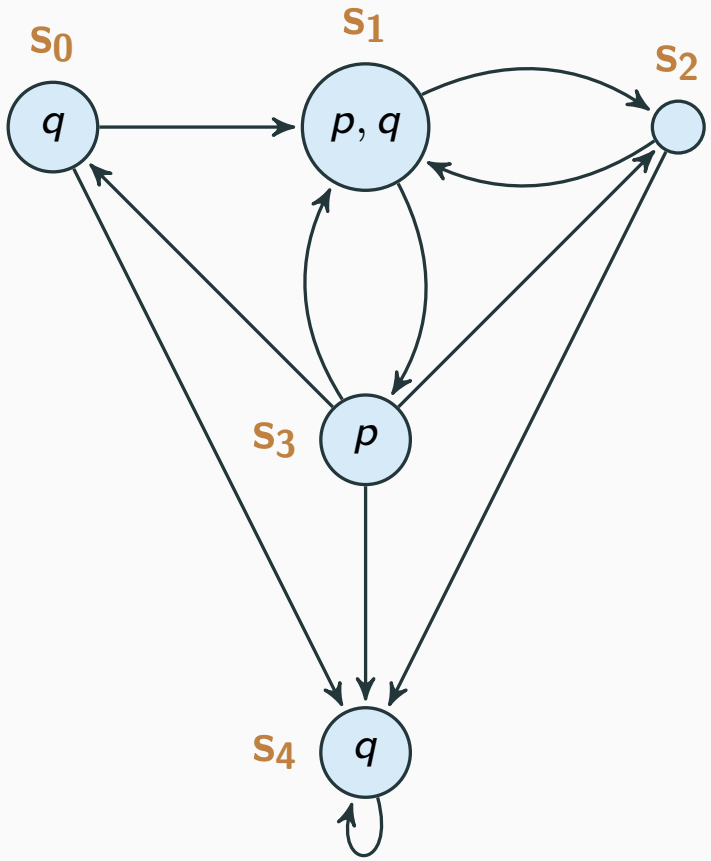


The labelling algorithm

Example

$\mathbf{AG(q \rightarrow EG\ p)} \equiv \neg\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$

- Label q: $\{s_0, s_1, s_4\}$
- Label p: $\{s_1, s_3\}$
- Label $\neg p$: $\{s_0, s_2, s_4\}$
- Label $\mathbf{AF}\neg p$: $\{s_0, s_2, s_4\}$
  1. s1 cannot be labeled because of s3
  2. s3 cannot be labeled because of s1
- Label $q \wedge \mathbf{AF}\neg p$: $\{s_0, s_4\}$
- Label $\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$:
  1. $\{s_0, s_4\}$
  2. $\{s_0, s_4, s_2\}$

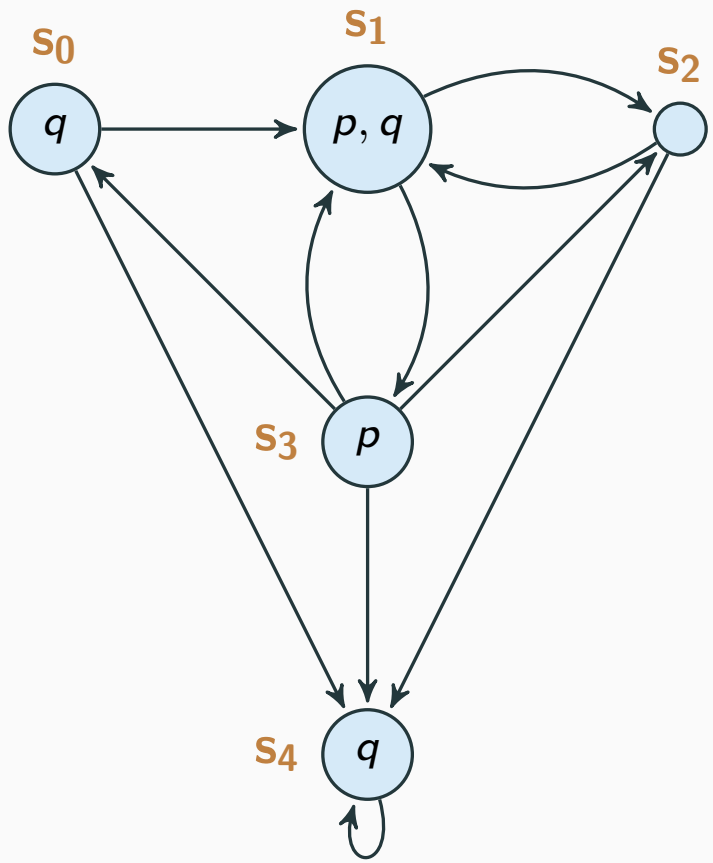


The labelling algorithm

Example

AG(q $\rightarrow$ **EG p)** $\equiv \neg\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$

- Label q: $\{s_0, s_1, s_4\}$
- Label p: $\{s_1, s_3\}$
- Label $\neg p$: $\{s_0, s_2, s_4\}$
- Label $\mathbf{AF}\neg p$: $\{s_0, s_2, s_4\}$
  1. s1 cannot be labeled because of s3
  2. s3 cannot be labeled because of s1
- Label $q \wedge \mathbf{AF}\neg p$: $\{s_0, s_4\}$
- Label $\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$:
  1. $\{s_0, s_4\}$
  2. $\{s_0, s_4, s_2\}$
  3. $\{s_0, s_4, s_2, s_1\}$

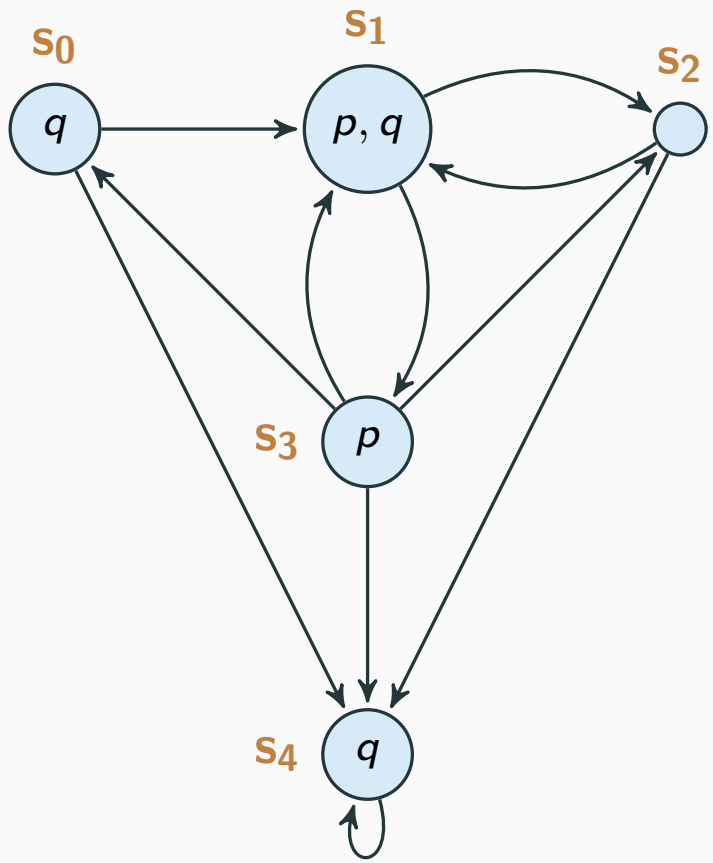


The labelling algorithm

Example

AG(q $\rightarrow$ EG p) $\equiv \neg\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$

- Label q: $\{s_0, s_1, s_4\}$
- Label p: $\{s_1, s_3\}$
- Label $\neg p$: $\{s_0, s_2, s_4\}$
- Label $\mathbf{AF}\neg p$: $\{s_0, s_2, s_4\}$
  1. s1 cannot be labeled because of s3
  2. s3 cannot be labeled because of s1
- Label $q \wedge \mathbf{AF}\neg p$: $\{s_0, s_4\}$
- Label $\mathbf{E}[\neg\bot\ \mathbf{U}\ (q \wedge \mathbf{AF}\neg p)]$:
  1. $\{s_0, s_4\}$
  2. $\{s_0, s_4, s_2\}$
  3. $\{s_0, s_4, s_2, s_1\}$
  4. $\{s_0, s_4, s_2, s_1, s_3\}$

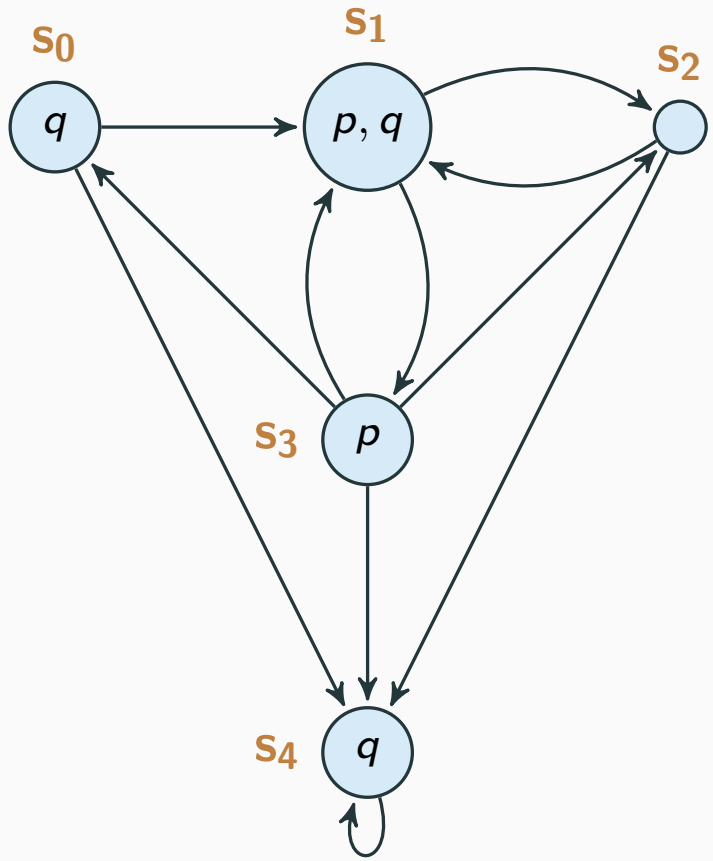


The labelling algorithm

Example

AG(q → EG p) $\equiv \neg \mathbf{E}[\neg \bot \ \mathbf{U} \ (q \wedge \mathbf{AF} \neg p)]$

- Label q: $\{s_0, s_1, s_4\}$
- Label p: $\{s_1, s_3\}$
- Label $\neg p$: $\{s_0, s_2, s_4\}$
- Label $\mathbf{AF} \neg p$: $\{s_0, s_2, s_4\}$
  1. s1 cannot be labeled because of s3
  2. s3 cannot be labeled because of s1
- Label $q \wedge \mathbf{AF} \neg p$: $\{s_0, s_4\}$
- Label $\mathbf{E}[\neg \bot \ \mathbf{U} \ (q \wedge \mathbf{AF} \neg p)]$:
  1. $\{s_0, s_4\}$
  2. $\{s_0, s_4, s_2\}$
  3. $\{s_0, s_4, s_2, s_1\}$
  4. $\{s_0, s_4, s_2, s_1, s_3\}$
- Label $\neg \mathbf{E}[\neg \bot \ \mathbf{U} \ (q \wedge \mathbf{AF} \neg p)]$: $\emptyset$



The state explosion problem

Although the labelling algorithm is linear in the size of the model, unfortunately the size of the model is itself more often than not exponential in the number of variables and the number of components of the system which execute in parallel.

For example, adding a boolean variable to a program will double the complexity of verifying a property of it.

The tendency of state spaces to become very large is known as the state explosion problem.

The state explosion problem

A lot of research had gone into finding ways to overcoming it:

- Efficient data structures, called ordered binary decision diagrams (OBDDs) which represent sets of states instead of individual states.
- Abstraction: that one may interpret a model abstractly, uniformly or for a specific property.
- Partial order reduction: for asynchronous systems, several interleavings of components traces may be equivalent as far as satisfaction of the formula to be checked is concerned.
- Induction: model-checking systems with (e.g.) large numbers of identical, or similar, components can often be implemented by "induction" on this number.
- ...

References

- Lecture Notes on "Program Analysis", ETH Zurich, Martin Vechev.
- Lecture Notes on "Techniques for Program Analysis and Verification", Stanford, Clark Barrett.
- Lecture Notes on "Program verification", ETH Zurich, Alexander Summers.
- Lecture Notes on "Computer-Aided Reasoning for Software Engineering", University of Washington, Emina Torlak.