Verification by Model Checking

Bow-Yaw Wang

Institute of Information Science
Academia Sinica, Taiwan

November 22, 2017

Motivation I

- Computers are commonly used in modern society.
  - aircrafts, high-speed trains, cars, nuclear plants, banks, hospitals, governments, etc.
- What if computer programs go wrong?
  - Therac-25, Ariane 5, Pentium FDIV, high-speed rail, etc.
- A prominent application of logic in computer science is to verify critical computer systems.
- With logic, we are able to state and prove properties about computer systems formally.

Motivation II

- Engineering techniques are also used to build critical systems.
  - testing, metrics, documentation, good programming practices, etc.
- Such techniques cannot guarantee correctness.
- Since mid-1990's, formal logic has been deployed in computer industry.
- Many organizations are asking manufacturers to apply formal methods in development cycles.

Classification of Verification Techniques

- Proof- versus model-based. Is the technique syntactic or semantic?
- Degree of automation. Does the technique need human guidance? How much?
- Full- versus property-verification. Does the technique verify all or some requirements?
- Intended domain. What types of systems (hardware/software, interactive/reactive etc) the technique is designed for?
- Pre- versus post-development. Is the technique applied before or after system development?

Model Checking

- Model checking is a model-based, automatic, property-oriented verification technique for concurrent and reactive systems.
- We construct a model $\mathcal{M}$ of a system and specify a property ϕ in some logic.
- The model $\mathcal{M}$ describes system behaviors (variable values, messages sent or received, etc).
- $\mathcal{M}$ has a designated state s.
- The property ϕ is specified in temporal logics.
- Informally, $\mathcal{M}, s \vDash \phi$ holds if $\mathcal{M}$ satisfies ϕ at s.
- Model checking verifies whether $\mathcal{M}, s \vDash \phi$ holds.

Outline

Outline

1. Linear-time temporal logic
   - Syntax of LTL
   - Semantics of LTL
   - Practical patterns of specifications
   - Important equivalences between LTL formulae
   - Adequate sets of connectives for LTL
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL

Linear-time Temporal Logic (LTL)

- Linear-time temporal logic (LTL) models time as an infinite sequence of states.
  - Such an infinite sequence of states is called a computation path or simply path.
- LTL allows us to describe temporal properties about computation paths.
  - For instance, event P eventually happens, or event P happens until event Q does, etc.

Outline

Syntax of LTL

- Consider a fixed set $\mathtt{Atoms}$ of atomic formulae $p, q, r, \ldots$.

Definition

Linear-time temporal logic has the following syntax:

$$\begin{array}{rcl}\phi & ::= & \top \mid \bot \mid p \mid (\neg\phi) \mid (\phi \land \phi) \mid (\phi \lor \phi) \mid (\phi \implies \phi) \mid \\ & & (\mathbf{X}\phi) \mid (\mathbf{F}\phi) \mid (\mathbf{G}\phi) \mid (\phi\,\mathbf{U}\,\phi) \mid (\phi\,\mathbf{W}\,\phi) \mid (\phi\,\mathbf{R}\,\phi)\end{array}$$

where $p \in \mathtt{Atoms}$ is an atomic formula.

- The connectives $\mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{U}, \mathbf{R}$, and $\mathbf{W}$ are temporal connectives.
- Informally, $\mathbf{X}$ means "neXt state," $\mathbf{F}$ means "some Future state," $\mathbf{G}$ means "all future states (Globally)," $\mathbf{U}$ means "Until," $\mathbf{R}$ means "Release," and $\mathbf{W}$ means "Weak-until."

Convention and Examples

- By convention, binding powers of LTL connectives are:

strongest		$\rightarrow$	weakest
$\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}$	$\mathbf{U}, \mathbf{R}, \mathbf{W}$	$\wedge, \vee$	$\Longrightarrow$

- Examples:

$$\mathbf{F}p \wedge \mathbf{G}q \implies p\,\mathbf{W}\,r \qquad \mathbf{F}(p \implies \mathbf{G}r) \vee \neg q\,\mathbf{U}\,p$$
$$p\,\mathbf{W}\,(q\,\mathbf{W}\,r) \qquad \mathbf{GF}p \implies \mathbf{F}(q \vee s)$$

- Non-examples:

$$\cancel{\mathbf{U}r} \qquad \cancel{p\mathbf{G}q}$$

- A subformula of an LTL formula ϕ is a formula ψ whose parse tree is a subtree of ϕ's parse tree.

Outline

1. Linear-time temporal logic
   - Syntax of LTL
   - Semantics of LTL
   - Practical patterns of specifications
   - Important equivalences between LTL formulae
   - Adequate sets of connectives for LTL
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL

Semantics of LTL I

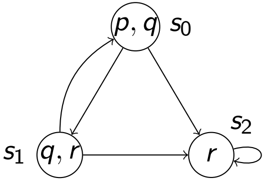
- Recall that LTL allows us to describe properties about computation paths.
- We will formalize computation paths by transition systems.

Definition

A transition system $\mathcal{M} = (S, \rightarrow, L)$ consists of a set S of states, a total transition relation $\rightarrow \subseteq S \times S$, and a labelling function $L : S \rightarrow 2^{\texttt{Atoms}}$.

- Instead of $(s, s') \in \rightarrow$, we will write $s \rightarrow s'$. not inform that s can only go to s'
- A transition relation $\rightarrow \subseteq S \times S$ is total if for every $s \in S$ there is an $s' \in S$ such that $s \rightarrow s'$.
- For any $s \in S$, $L(s)$ contains the set of atomic formulae which are true in s.
- A transition system is also called a model.

Semantics of LTL II



- Let $\mathcal{M} = (S, \rightarrow, L)$ with $S = \{s_0, s_1, s_2\}$, $\rightarrow = \{(s_0, s_1), (s_0, s_2), (s_1, s_0), (s_1, s_2), (s_2, s_2)\}$, and $L(s_0) = \{p, q\}$, $L(s_1) = \{q, r\}$, and $L(s_2) = \{r\}$.
- We can represent the transition system $\mathcal{M}$ as a directed graph.
- Note that transition relations must be total.
  - If the self loop at s_2 were removed, $\mathcal{M}$ would not be a transition system.
  - In order to model a state s without any outgoing transition, we add a new state s_d with a self loop and $s \rightarrow s_d$.

Computation Path and Suffix

Definition

A path in a model $\mathcal{M} = (S, \rightarrow, L)$ is an infinite sequence of states $s_0, s_1, \ldots, s_n, \ldots$ such that $s_i \rightarrow s_{i+1}$ for every $i \geq 0$. We also write $s_0 \rightarrow s_1 \rightarrow \cdots$ for the path $s_0, s_1, \ldots, s_n, \ldots$.

- For instance, $s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots$ is a path in the example.

Definition

Let $\pi = s_0 \rightarrow s_1 \rightarrow \cdots$ be a path in a model $\mathcal{M} = (S, \rightarrow, L)$. The *i*-suffix π^i is the suffix $s_i \rightarrow s_{i+1} \rightarrow \cdots$ of π.

- Let $\pi \triangleq s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots$. We have
  - $\pi^0 \triangleq \pi$;
  - $\pi^1 \triangleq s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots$;
  - $\pi^2 \triangleq s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow \cdots$;
  - $\pi^3 \triangleq s_2 \rightarrow s_2 \rightarrow \cdots$; etc.
- Suffixes of a path are paths.

$\pi \vDash \phi$ I

Definition

Let $\mathcal{M} = (S, \rightarrow, L)$ be a model, $\pi = s_0 \rightarrow s_1 \rightarrow \cdots$ a path in $\mathcal{M}$, and ϕ an LTL formula. Define the satisfaction relation $\pi \vDash \phi$ as follows.

1. $\pi \vDash \top$; $\pi \nvDash \bot$; and $\pi \vDash p$ if $p \in L(s_0)$;
2. $\pi \vDash \neg\phi$ if $\pi \nvDash \phi$; $\pi \vDash \phi \wedge \psi$ if $\pi \vDash \phi$ and $\pi \vDash \psi$; $\pi \vDash \phi \vee \psi$ if $\pi \vDash \phi$ or $\pi \vDash \psi$;
3. $\pi \vDash \phi \implies \psi$ if $\pi \vDash \psi$ whenever $\pi \vDash \phi$;
4. $\pi \vDash \mathbf{X}\phi$ if $\pi^1 \vDash \phi$;
5. $\pi \vDash \mathbf{G}\phi$ if $\pi^i \vDash \phi$ for every $i \geq 0$;
6. $\pi \vDash \mathbf{F}\phi$ if $\pi^i \vDash \phi$ for some $i \geq 0$;
7. $\pi \vDash \phi\,\mathbf{U}\,\psi$ if there is some $i \geq 0$ such that $\pi^i \vDash \psi$ and for every $0 \leq j < i$, $\pi^j \vDash \phi$;
8. $\pi \vDash \phi\,\mathbf{W}\,\psi$ if either there is some $i \geq 0$ such that $\pi^i \vDash \psi$ and for every $0 \leq j < i$ we have $\pi^j \vDash \phi$; or for every $k \geq 0$ we have $\pi^k \vDash \phi$;
9. $\pi \vDash \phi\,\mathbf{R}\,\psi$ if either there is some $i \geq 0$ such that $\pi^i \vDash \phi$ and for every $0 \leq j \leq i$ we have $\pi^j \vDash \psi$; or for every $k \geq 0$ we have $\pi^k \vDash \psi$.

1. $\top$ is always true; $\bot$ is always false; and $\pi \vDash p$ if p is true at the start of π;
2. $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, and $\phi \implies \psi$ have the usual semantics;
3. $\mathbf{X}\phi$ is true if ϕ is true at the 1-suffix of π;
4. $\mathbf{G}\phi$ is true if ϕ is always true in the future;
   - Note that "present" is a part of "future."
5. $\mathbf{F}\phi$ is true if ϕ is true for some future;
6. $\phi\,\mathbf{U}\,\psi$ is true if ϕ is true until exclusively ψ is true;
   - Note that ψ must be true in the future.
7. $\phi\,\mathbf{W}\,\psi$ is true if $\phi\,\mathbf{U}\,\psi$ or ϕ is always true;
   - Note that $\phi\,\mathbf{W}\,\psi$ does not require ψ to be true in the future.
8. $\phi\,\mathbf{R}\,\psi$ is true if ψ is true until inclusively ϕ releases ψ, or ψ is always true.
   - Note that $\phi\,\mathbf{R}\,\psi$ does not require ϕ to be true in the future.

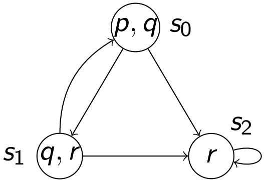
$\mathcal{M}, s \vDash \phi$

Definition

Let $\mathcal{M} = (S, \rightarrow, L)$ be a model, $s \in S$, and ϕ an LTL formula. $M, s \vDash \phi$ if for every path π of $\mathcal{M}$ starting at s, we have $\pi \vDash \phi$.

- Think of the model $\mathcal{M}$ as the description of a program and s a state of the program.
- $\mathcal{M}, s \vDash \phi$ holds if for every possible computation path satisfies the LTL formula ϕ.
- Particularly, consider programs that depends on user inputs and the initial state s_i of such a program.
- $\mathcal{M}, s_i \vDash \phi$ holds if all executions of the program satisfy ϕ no matter what user inputs are.
- This is a very strong statement.
  - Much stronger than testing programs.
  - Testing can only falsify properties; it cannot prove properties.

Examples I



- $\mathcal{M}, s_0 \vDash p \wedge q$;
- $\mathcal{M}, s_0 \vDash \neg r$;
- $\mathcal{M}, s_0 \vDash \top$;
- $\mathcal{M}, s_0 \vDash \mathbf{X} r$;
- $\mathcal{M}, s_0 \nvDash \mathbf{X}(q \wedge r)$;
- $\mathcal{M}, s_0 \vDash \mathbf{G}\neg(p \wedge r)$;
- $\mathcal{M}, s_0 \vDash \mathbf{GF} r$; not satisfy FGr
- $\mathcal{M}, s_0 \vDash \mathbf{GF} p \implies \mathbf{GF} r$;
- $\mathcal{M}, s_0 \nvDash \mathbf{GF} r \implies \mathbf{GF} p$.

Outline

Patterns of Specifications

- It is impossible to go to a state where *started* holds but *ready* does not: $\mathbf{G}\neg(started \land \neg ready)$
- For any state, if *request* holds then it will be *acknowledged* eventually: $\mathbf{G}(request \implies \mathbf{F}acknowledged)$
- *enabled* occurs infinitely often: $\mathbf{GF}enabled$
- *stable* will eventually occurs permanently: $\mathbf{FG}stable$
- If a process is *enabled* infinitely often, it is *running* infinitely often: $\mathbf{GF}enabled \implies \mathbf{GF}running$

Outline

1. Linear-time temporal logic
   - Syntax of LTL
   - Semantics of LTL
   - Practical patterns of specifications
   - Important equivalences between LTL formulae
   - Adequate sets of connectives for LTL
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL

Semantically Equivalence I

Definition

Let ϕ and ψ be LTL formulae. ϕ and ψ are semantically equivalent (written $\phi \equiv \psi$) if for all paths π, $\pi \vDash \phi$ iff $\pi \vDash \psi$.

- Semantically equivalent propositional formulae are still equivalent, for example:

$$\neg(\phi \land \psi) \equiv \neg\phi \lor \neg\psi \qquad \neg(\phi \lor \psi) \equiv \neg\phi \land \neg\psi.$$

- **F** and **G** are dual; **X** is dual to itself:

$$\neg\mathbf{G}\phi \equiv \mathbf{F}\neg\phi \qquad \neg\mathbf{F}\phi \equiv \mathbf{G}\neg\phi \qquad \neg\mathbf{X}\phi \equiv \mathbf{X}\neg\phi.$$

- **U** and **R** are dual (why?):

$$\neg(\phi\,\mathbf{U}\,\psi) \equiv \neg\phi\,\mathbf{R}\,\neg\psi \qquad \neg(\phi\,\mathbf{R}\,\psi) \equiv \neg\phi\,\mathbf{U}\,\neg\psi.$$

Semantically Equivalence II

- **F** distributes over $\vee$ and **G** over $\wedge$ (why?):

$$\mathbf{F}(\phi \vee \psi) \equiv \mathbf{F}\phi \vee \mathbf{F}\psi \qquad \mathbf{G}(\phi \wedge \psi) \equiv \mathbf{G}\phi \wedge \mathbf{G}\psi.$$

  - What about **F** over $\wedge$ and **G** over $\vee$?

-> : correct

$$\mathbf{F}(\phi \wedge \psi) \stackrel{?}{\equiv} \mathbf{F}\phi \wedge \mathbf{F}\psi$$

<- : not-correct

<- : correct

$$\mathbf{G}(\phi \vee \psi) \stackrel{?}{\equiv} \mathbf{G}\phi \vee \mathbf{G}\psi.$$

-> : not-correct

ex: phi, psi交替出現

- **F** and **G** in **U** and **R**:

$$\mathbf{F}\phi \equiv \top \,\mathbf{U}\, \phi \qquad \mathbf{G}\phi \equiv \bot \,\mathbf{R}\, \phi.$$

- **U** is equivalent to **W** and **F**:

$$\phi \,\mathbf{U}\, \psi \equiv \phi \,\mathbf{W}\, \psi \wedge \mathbf{F}\psi.$$

- **W** and **R** are closely related:

$$\phi \,\mathbf{W}\, \psi \equiv \psi \,\mathbf{R}\, (\phi \vee \psi) \qquad \phi \,\mathbf{R}\, \psi \equiv \psi \,\mathbf{W}\, (\phi \wedge \psi)$$

Semantically Equivalence III

Theorem

$\phi\,\mathbf{U}\,\psi \equiv \neg(\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi)) \wedge \mathbf{F}\psi$.

Proof.

Consider any path $\pi = s_0 \to s_1 \to \cdots$.

Suppose $\pi \vDash \phi\,\mathbf{U}\,\psi$. Let n be the smallest number that $\pi^n \vDash \psi$. We have $\pi \vDash \mathbf{F}\psi$. It remains to show $\pi \vDash \neg(\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi))$. By the definition of n, $\pi^i \vDash \neg\psi$ for $0 \leq i < n$. Moreover, $\pi^n \vDash \psi$ and hence $\pi^n \nvDash \neg\phi \wedge \neg\psi$. That is, $\pi \nvDash \neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi)$.

Conversely, suppose $\pi \vDash \neg(\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi)) \wedge \mathbf{F}\psi$. Let n be the smallest number that $\pi^n \vDash \psi$ (by $\pi \vDash \mathbf{F}\psi$). Assume $\pi^i \vDash \neg\phi$ for some $0 \leq i < n$. Then $\pi^i \vDash \neg\psi$ by the minimality of n. Hence $\pi^i \vDash \neg\phi \wedge \neg\psi$. There is an $0 \leq j < i < n$ such that $\pi^j \vDash \psi$ since $\pi \vDash \neg(\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi))$. A contradiction to the minimality of n. □

Outline

1. Linear-time temporal logic
   - Syntax of LTL
   - Semantics of LTL
   - Practical patterns of specifications
   - Important equivalences between LTL formulae
   - Adequate sets of connectives for LTL
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL

Adequate Sets of LTL Connectives I

- An adequate set of connectives in a logic can express any connective in the same logic.
  - For instance, $\{\bot, \wedge, \neg\}$ is an adequate set of connectives in propositional logic.
- By semantical equivalences in LTL, we have the following adequate sets:
  - $\{\mathbf{U}, \mathbf{X}\}$. Recall $\phi \,\mathbf{R}\, \psi \equiv \neg(\neg\phi \,\mathbf{U}\, \neg\psi)$ and $\phi \,\mathbf{W}\, \psi \equiv \psi \,\mathbf{R}\, (\phi \vee \psi)$.
  - $\{\mathbf{R}, \mathbf{X}\}$. Recall that $\phi \,\mathbf{U}\, \psi \equiv \neg(\neg\phi \,\mathbf{R}\, \neg\psi)$ and $\phi \,\mathbf{W}\, \psi \equiv \psi \,\mathbf{R}\, (\phi \vee \psi)$.
  - $\{\mathbf{W}, \mathbf{X}\}$. Recall that $\phi \,\mathbf{R}\, \psi \equiv \psi \,\mathbf{W}\, (\phi \wedge \psi)$ and $\phi \,\mathbf{U}\, \psi \equiv \neg(\neg\phi \,\mathbf{R}\, \neg\psi)$.
- Note that $\mathbf{X}$ is independent of other connectives.

Adequate Sets of LTL Connectives II

- Consider the fragment of LTL without negation and $\mathbf{X}$.
- We have the following adequate sets:
  - $\{\mathbf{U}, \mathbf{R}\}$ since $\phi \,\mathbf{W}\, \psi \equiv \psi \,\mathbf{R}\, (\phi \vee \psi)$, $\mathbf{F}\phi \equiv \top \,\mathbf{U}\, \phi$, and $\mathbf{G}\phi \equiv \bot \,\mathbf{R}\, \phi$.
  - $\{\mathbf{U}, \mathbf{W}\}$ since $\phi \,\mathbf{R}\, \psi \equiv \psi \,\mathbf{W}\, (\phi \wedge \psi)$, $\mathbf{F}\phi \equiv \top \,\mathbf{U}\, \phi$, and $\mathbf{G}\phi \equiv \bot \,\mathbf{R}\, \phi$.
  - $\{\mathbf{U}, \mathbf{G}\}$ since $\phi \,\mathbf{W}\, \psi \equiv \phi \,\mathbf{U}\, \psi \vee \mathbf{G}\phi$, $\phi \,\mathbf{R}\, \psi \equiv \psi \,\mathbf{W}\, (\phi \wedge \psi)$, and $\mathbf{F}\phi \equiv \top \,\mathbf{U}\, \phi$.
  - $\{\mathbf{R}, \mathbf{F}\}$ since $\phi \,\mathbf{W}\, \psi \equiv \psi \,\mathbf{R}\, (\phi \vee \psi)$, $\phi \,\mathbf{U}\, \psi \equiv \phi \,\mathbf{W}\, \psi \wedge \mathbf{F}\psi$, and $\mathbf{G}\phi \equiv \bot \,\mathbf{R}\, \phi$.
  - $\{\mathbf{W}, \mathbf{F}\}$ since $\phi\mathbf{U} \equiv \phi \,\mathbf{W}\, \psi \wedge \mathbf{F}\phi$, $\phi \,\mathbf{R}\, \psi \equiv \psi \,\mathbf{W}\, (\phi \wedge \psi)$, and $\mathbf{G}\phi \equiv \bot \,\mathbf{R}\, \phi$.
- Note that $\{\mathbf{R}, \mathbf{G}\}$, $\{\mathbf{W}, \mathbf{G}\}$, and $\{\mathbf{U}, \mathbf{F}\}$ are not adequate.
  - $\mathbf{F}$ cannot be defined by $\{\mathbf{R}, \mathbf{G}\}$ nor $\{\mathbf{W}, \mathbf{G}\}$.
  - $\mathbf{G}$ cannot be defined by $\{\mathbf{U}, \mathbf{F}\}$.

Outline

Outline

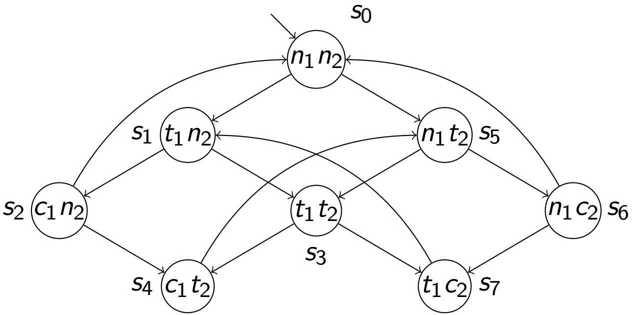
Mutual Exclusion I

- When two concurrent processes share a resource (printer, disk, etc), they sometimes need to access the resource exclusively.
- Critical sections are portions of process codes that have exclusive access to a shared resource.
- For efficiency, critical sections should be as small as possible.
- Moreover, at most one process can enter its critical section at any time.
- We will design a simple protocol to ensure mutually exclusive access to critical sections and verify our solution.

Mutual Exclusion II

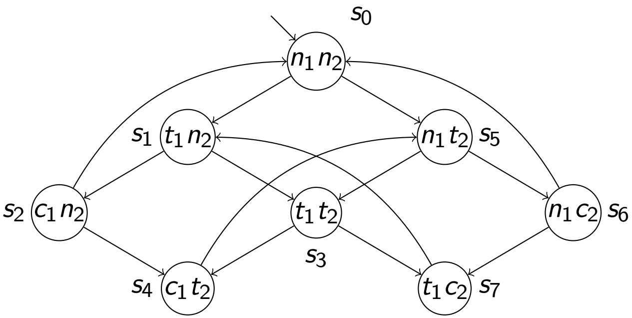
- Let us first try to specify our requirements informally.
- A protocol solving the mutual exclusion problem must ensure the following property:
  Safety: at most one process can enter its critical section at any time.
- Moreover, a protocol should not prevent any process from entering critical sections permanently:
  Liveness: When a process requests to enter its critical section, it will eventually be permitted to do so.
  Non-blocking: A process can always request to enter its critical section.

Mutual Exclusion: First Attempt I



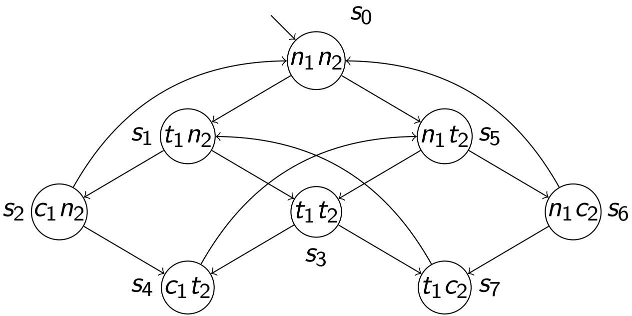
- Consider two processes P_1 and P_2.
- P_1 has three states: non-critical state (n_1), trying state (t_1), and critical state (c_1). Similarly, P_2 has states n_2, t_2, and c_2.
  - Local states are modeled as atomic formulae.
- Each process has transitions $n \to t \to c \to n \to t \to c \to \cdots$.

Mutual Exclusion: First Attempt II



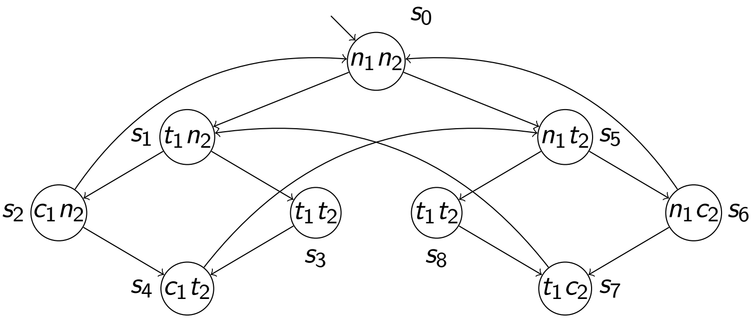
- The system starts with both processes at non-critical states (s_0).
- Exactly one process makes a transition at any time.
  - This is called an asynchronous interleaving model.

Mutual Exclusion: First Attempt III



- Safety. The property is expressed by $\mathbf{G}\neg(c_1 \wedge c_2)$ in LTL. It holds.
- Liveness. This is expressed by $\mathbf{G}(t_1 \implies \mathbf{F}c_1)$ in LTL. The property is not satisfied due to the path $s_0 \to s_1 \to s_3 \to s_7 \to s_1 \to s_3 \to s_7 \cdots$.
- Non-blocking. We would like to express: when a process at n, there is a successor at t. This is not expressible in LTL.

Mutual Exclusion: Second Attempt



- Liveness does not hold because the state s_3 does not record which process enters the trying state first.
- In our second design, we use two states to record which process enters the trying state first.
  - s_3 remembers P_1 enters t_1 first; s_8 remembers P_2 enters t_2 first.
- One can verify all three properties are satisfied.

Outline

The NuSMV Model Checker

- NuSMV is an open-sourced model checker.
- We specify a transition system $\mathcal{M} = (S, \rightarrow, L)$ with an initial state $s_0 \in S$, and a linear temporal logic formula ϕ.
- NuSMV checks whether $\mathcal{M}, s_0 \vDash \phi$ holds.
- We will learn how to specify a transition system in NuSMV.

NuSMV Modules I

- A NuSMV model consists of several modules.
- A module can be instantiated several times with different names.
- There is exactly one `main` module.
- An LTL formula uses the following symbols:

NuSMV	for	*LTL*	NuSMV	for	*LTL*
`!`		$\neg$	`->`		$\Longrightarrow$
`&`		$\wedge$	`|`		$\vee$
`G`		**G**	`F`		**F**
`X`		**X**	`U`		**U**

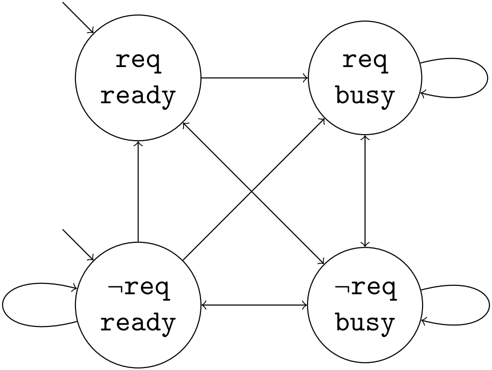
NuSMV Modules II

- Here is a NuSMV model with an LTL formula $\mathbf{G}(\text{request} \implies \mathbf{F}(\text{status} = \text{busy}))$:

```
MODULE main
VAR
  request : boolean;
  status : { ready, busy };
ASSIGN
 init (status) := ready;
 next (status) := case
                    request : busy;
                    TRUE : { ready, busy };
                  esac;
LTLSPEC
  G (request -> F status = busy);
```

NuSMV Modules III

- The NuSMV model specifies the following transition system:



One-bit Adder

```
MODULE adder (i0, i1)
VAR o : boolean; c : boolean;
ASSIGN
  o := i0 xor i1;
  c := i0 & i1;
MODULE main
VAR
  b0 : boolean; b1 : boolean;
  a : adder (b0, b1);
LTLSPEC G (!b0 & !b1 -> !a.c & !a.o);
LTLSPEC G ( b0 & !b1 -> !a.c &  a.o);
LTLSPEC G (!b0 &  b1 -> !a.c &  a.o);
LTLSPEC G ( b0 &  b1 ->  a.c & !a.o);
```

- Here is a possible trace:

	0	1	2	3	···
(b0, b1, a.o, a.c)	(F F F F)	(F T T F)	(F F F F)	(T T F T)	···

Two-bit Adder

```
MODULE adder (b0, b1, ic)
VAR o : word[1]; c : word[1];
ASSIGN
  o := word1(bool(b0) xor bool(b1) xor bool(ic));
  c := word1((bool(b0) & bool(b1)) | (bool(b0) & bool(ic)) |
             (bool(b1) & bool(ic)));
MODULE main
VAR
  i : word[2]; j : word[2];
  a0 : adder (i[0:0], j[0:0], 0b1_0);
  a1 : adder (i[1:1], j[1:1], a0.c);
DEFINE
  o := a1.o :: a0.o; c := a1.c;
LTLSPEC G (i = 0b2_10 & j = 0b2_10 -> c = 0b1_1 & o = 0b2_00);
LTLSPEC G (i = 0b2_10 & j = 0b2_11 -> c = 0b1_1 & o = 0b2_01);
```

- $id\,[high : low]$ selects bits in words
- 0b4_1010 = 0o2_12 = 0d2_10 = 0h1_a
- DEFINE defines a macro

A Two-bit Counter

```
MODULE counter
VAR b : array 0..1 of boolean;
ASSIGN
  init (b[0]) := FALSE; init (b[1]) := FALSE;
  next (b[0]) := !b[0];
  next (b[1]) := case
                    !b[0] :  b[1];
                     b[0] : !b[1];
                 esac;
MODULE main
VAR c : counter;
```

- Use `array of` to declare arrays
- Use `init` to refer the initial value of a variable
- The following shows how `c.b` changes

	0	1	2	3	4	⋯
(c.b[1], c.b[0])	(F F)	(F T)	(T F)	(T T)	(F F)	⋯

More about case

```
var := case
         bexp0 : exp0;
         bexp1 : exp1;
         ...
         bexpn : expn;
       esac;
```

- In a case expression, bexp's are evaluated consecutively
- The result of a case expression is exp*i* if bexp0 to bexp*i-1* all evaluate to false but bexp*i* evaluates to true
- NuSMV requires bexp0 $\vee$ bexp1 $\vee \cdots \vee$ bexpn is valid

Define by Constraints

```
MODULE delayed-inverter-constraint (b)
VAR o : boolean;
TRANS next (o) = !b;
MODULE main
VAR
  b : boolean;
  i : delayed-inverter-constraint (b);
INIT b = FALSE;
INVAR b = !i.o;
LTLSPEC G !b
```

- INIT *bexp* specifies a condition to be fulfilled at an initial state
- INVAR *bexp* specifies a condition to be fulfilled at any state
- TRANS *next-bexp* specifies a condition to be fulfilled at each transition

	0	1	2	···
(b, i.o)	(F T)	(F T)	(F T)	···

Synchronous and Asynchronous Composition

- NuSMV composes modules synchronously by default.
  - That is, every module make a transition at the same time.
- We may want to specify asynchronous modules sometimes.
  - That is, exactly one module makes a transition at any time.
- To specify asynchronous modules, we use the keyword `process` to instantiate a module.
- An asynchronous module has an implicit Boolean variable `running`.
- The variable `running` is true when the module takes a transition.

Outline

Mutual Exclusion Revisited I

```
MODULE prc (other-st, turn, myturn)
VAR st : { n, t, c };
ASSIGN
  init (st) := n;
  next (st) := case
    st = n                                    : { t, n };
    st = t & other-st = n                     : c;
    st = t & other-st = t & turn = myturn     : c;
    st = c                                    : { c, n };
    TRUE                                      : st;
  esac;
  next (turn) := case
    turn = myturn & st = c                    : !turn;
    TRUE                                      : turn;
  esac;
FAIRNESS running;
FAIRNESS !(st = c);
```

Mutual Exclusion Revisited II

```
MODULE main
VAR
  turn : boolean;
  pr1 : process prc (pr2.st, turn, FALSE);
  pr2 : process prc (pr1.st, turn, TRUE);
ASSIGN
  init (turn) := FALSE;
-- safety
LTLSPEC G !((pr1.st = c) & (pr2.st = c))
-- liveness
LTLSPEC G ((pr1.st = t) -> F (pr1.st = c))
LTLSPEC G ((pr2.st = t) -> F (pr2.st = c))
```

- `FAIRNESS` *`bexp`* `;` is a fairness constraint.
- NuSMV only considers paths where *`bexp`* is true infinitely often if fairness constraints present.
  - What happens if we remove fairness constraints?

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
   - Mutual exclusion revisited
   - The ferryman
   - The alternating bit protocol
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL
6. Model-checking algorithms

The Ferryman Puzzle I

- Consider the following puzzle: a ferryman, goat, cabbage, and a wolf are on one side of a river. The ferryman can cross the river with at most one passenger in his boat. However
  - the goat eats the cabbage; and
  - the wolf eats the goat.

  The ferryman must be on the same side of the river to prevent conflicts. Can the ferryman transport all goods to the other side safely?
- Let us use NuSMV to check if the ferryman is solvable.

The Ferryman Puzzle II

```
MODULE main
VAR     ferryman : boolean;    goat : boolean;
         cabbage : boolean;    wolf : boolean;
           carry : { g, c, w, n };
ASSIGN
  init (ferryman) := FALSE;   init (goat)      := FALSE;
  init (cabbage)  := FALSE;   init (wolf)      := FALSE;
  init (carry)    := n;
  next (ferryman) := { FALSE, TRUE };
  next (goat) := case
    ferryman = goat & next (carry) = g : next (ferryman);
    TRUE                               : goat;        esac;
  next (cabbage) := case
    ferryman = cabbage & next (carry) = c : next (ferryman);
    TRUE                                  : cabbage;   esac;
  next (wolf) := case
    ferryman = wolf & next (carry) = w : next (ferryman);
    TRUE                               : wolf;        esac;
TRANS                       (next (carry) = n) |
  (ferryman = goat     & next (carry) = g) |
  (ferryman = cabbage  & next (carry) = c) |
  (ferryman = wolf     & next (carry) = w);
```

The Ferryman Puzzle III

- Boolean variables `ferryman`, `goat`, `cabbage`, `wolf` denote the location of the ferryman, goat, cabbage, wolf.
- Initially, all are on the same side (`FALSE`).
- The variable `carry` denotes the good carried by the ferryman: `g` (goat), `c` (cabbage), `w` (wolf), or `n` (none).
- At any time, the ferryman moves to either side.
- At any time, we have
  - the ferryman does not carry anything;
  - the ferryman carries the goat;
  - the ferryman carries the cabbage; or
  - the ferryman carries the wolf.

The Ferryman Puzzle IV

- We verify if it is impossible that no conflict occurs until all are on the other side.

```
LTLSPEC  !((  (goat = cabbage | goat = wolf) -> goat = ferryman)
            U (cabbage & goat & wolf & ferryman))
```

- That is, any "bug" gives a solution to the ferryman puzzle.

The Ferryman Puzzle V

- Here is a solution found by NuSMV:

```
-> State: 1.1 <-            |   ferryman = FALSE
  ferryman = FALSE          |   carry = n
  goat = FALSE              | -> State: 1.8 <-
  cabbage = FALSE           |   ferryman = TRUE
  wolf = FALSE              |   goat = TRUE
  carry = n                 |   carry = g
-> State: 1.2 <-            | -> State: 1.9 <-
  ferryman = TRUE           |   ferryman = FALSE
  goat = TRUE               |   wolf = FALSE
  carry = g                 |   carry = w
-> State: 1.3 <-            | -> State: 1.10 <-
  ferryman = FALSE          |   ferryman = TRUE
  carry = n                 |   carry = n
-> State: 1.4 <-            | -> State: 1.11 <-
  ferryman = TRUE           |   ferryman = FALSE
  wolf = TRUE               |   cabbage = FALSE
  carry = w                 |   carry = c
-> State: 1.5 <-            | -> State: 1.12 <-
  ferryman = FALSE          |   ferryman = TRUE
  goat = FALSE              |   carry = n
  carry = g                 | -> State: 1.13 <-
-> State: 1.6 <-            |   ferryman = FALSE
  ferryman = TRUE           |   goat = FALSE
  cabbage = TRUE            |   carry = g
  carry = c                 | -> State: 1.14 <-
-> State: 1.7 <-            |   carry = n
```

The Ferryman Puzzle VI

Side A		Side B
ferryman, cabbage, goat, wolf		
cabbage, wolf		ferryman, goat
ferryman, cabbage, wolf		goat
cabbage		ferryman, wolf, goat
ferryman, cabbage, goat		wolf
goat		ferryman, cabbage, wolf
ferryman, goat		cabbage, wolf
		ferryman, cabbage, goat, wolf
ferryman, wolf		cabbage, goat
wolf		ferryman, cabbage, goat
ferryman, cabbage, wolf		goat
cabbage, wolf		ferryman, goat
ferryman, cabbage, goat, wolf		

Outline

1. Linear-time temporal logic

2. Model checking

3. The NuSMV model checker
   - Mutual exclusion revisited
   - The ferryman
   - The alternating bit protocol

4. Branching-time logic

5. CTL* and the expressive powers of LTL and CTL

6. Model-checking algorithms

The Alternating Bit Protocol I

- The alternating bit protocol (ABP) is a network protocol for transmitting messages over a lossy line.
  - That is, messages may be lost during transmission.
- Here is an infomal description of ABP:
  - Consider a sender, a receiver, a data channel, and an acknowledgment channel.
  - Every packet is associated with a control bit.
  - The sender sends a packet with the control bit 0 over the data channel.
  - When the receiver receives a packet with the control bit 0, he sends 0 over the acknowledgment channel.
  - When the sender receives 0 from the acknowledgment channel, she sends another packet with the control bit 1.
  - When the receiver receives a packet with the control bit 1, he sends 1 over the acknowledgment channel.
  - If anyone did not get expected messages, she or he resends the last message.
- The control bit makes sure that packets cannot be lost or duplicated.

The Alternating Bit Protocol II

```
MODULE sender (ack)
VAR     st : { sending, sent };
        message : boolean; control : boolean;
ASSIGN
  init (st) := sending;
  next (st) := case
    ack = control & !(st = sent) : sent;
    TRUE                         : sending;
  esac;
  next (message) := case
    st = sent                    : { FALSE, TRUE };
    TRUE                         : message;
  esac;
  next (control) := case
    st = sent                    : !control;
    TRUE                         :  control;
  esac;
FAIRNESS running
LTLSPEC G F (st = sent)
```

The Alternating Bit Protocol III

- The variable `st` denotes that a message is `sent` or `sending`.
- The variable `message` represents a 1-bit packet.
- The variable `control` is the current control bit.
- The variable `ack` is the current acknowledgment.
- The sender module is `sending` a message most of the time.
- A message is `sent` when the acknowledgment bit is equal to the control bit.
- The 1-bit packet selects an arbitrary value when a message is `sent`.
- The control bit is inverted when a message is `sent`.

The Alternating Bit Protocol IV

```
MODULE receiver (message, control)
VAR    st : { receiving, received };
       ack : boolean; expected : boolean;
ASSIGN
  init (st) := receiving;
  next (st) := case
    control = expected & !(st = received) : received;
    TRUE                                  : receiving;
  esac;
  next (ack) := case
    st = received                         : control;
    TRUE                                  : ack;
  esac;
  next (expected) := case
    st = received                          : !expected;
    TRUE                                   :  expected;
  esac;
FAIRNESS running
LTLSPEC G F (st = received)
```

The Alternating Bit Protocol V

- The variable `st` denotes that a message is `received` or `receiving`.
- The variable `message` represents a 1-bit packet.
- The variable `control` is the current control bit.
- The variable `ack` is the current acknowledgment.
- The variable `expected` is the expected control bit.
- The receiver module is `receiving` a message most of the time.
- A message is `received` when the control bit is equal to the expected control bit.
- The acknowledgment bit changes when a message is `received`.
- The expected acknowledgment bit is inverted when a message is `received`.

The Alternating Bit Protocol VI

```
MODULE one-bit-channel (input)
VAR    forget : boolean; output : boolean;
ASSIGN     next (output) :=          case
             forget     : output;
             TRUE       : input;     esac;
FAIRNESS running
FAIRNESS  input & !forget
FARINESS !input & !forget
MODULE two-bit-channel (input1, input2)
VAR    forget : boolean;
       output1 : boolean; output2 : boolean;
ASSIGN     next (output1) :=         case
             forget     : output1;
             TRUE       : input1;    esac;
           next (output2) :=         case
             forget     : output2;
             TRUE       : input2;    esac;
FAIRNESS running
FAIRNESS  input1 & !forget
FARINESS !input1 & !forget
FAIRNESS  input2 & !forget
FARINESS !input2 & !forget
```

The Alternating Bit Protocol VII

- The variable `input` denotes an input bit.
- The variable `output` denotes the output of a lossy channel.
- A lossy channel sends the old `output` when it `forget`; otherwise, it sends the `input`.
- It is important that a lossy channel does not always `forget`.
  - In fact, it cannot always `forget` the same `input`.
  - What if all FALSE (or TRUE) are lost?

The Alternating Bit Protocol VIII

```
MODULE main
VAR
  s : process sender (ack_chan.output);
  r : process receiver (data_chan.output1, data_chan.output2);
  data_chan : process two-bit-channel (s.message, s.control);
  ack_chan : process one-bit-channel (r.ack);
ASSIGN
  init(s.control) := FALSE;
  init(r.expected) := FALSE;
  init(r.ack)       := TRUE;
  init(data_chan.output2) := TRUE;
  init(ack_chan.output) := TRUE;
LTLSPEC
  G (s.st=sent & s.message -> data_chan.output1)
```

The Alternating Bit Protocol IX

- Our system has a sender, a receiver, a two-bit lossy data channel, and a one-bit acknowledgment channel.
- The initial control bit is FALSE.
- The initial expected control bit is FALSE.
- The initial acknowledgment bit is TRUE.
- THe initial control bit from the data channel is TRUE.
- THe initial acknowledgment bit from the acknowledgment channel is TRUE.
- Three properties are checked:
  - Messages are `sent` by the sender infinitely often.
  - Messages are `received` by the receiver infinitely often.
  - When TRUE is `sent`, the packet in data channel is TRUE.

Outline

1. Linear-time temporal logic

2. Model checking

3. The NuSMV model checker

4. Branching-time logic
   - Syntax of CTL
   - Semantics of computation tree logic
   - Practical patterns of specifications
   - Important equivalences between CTL formulae
   - Adequate sets of CTL connectives

5. CTL* and the expressive powers of LTL and CTL

Branching-time Logic I

- In LTL, we specify properties about paths.
- A model satisfies an LTL formula if all its paths from the initial state satisfy the formula.
- Sometimes, we may wish to consider properties about states.
  - Recall that non-blocking asks if a process can always enter the trying state when it is at the non-critical state.
- Branching-time logic allows us to specify such properties.
- We can ask if
  - all paths from a state satisfy certain properties; and
  - a path from a state satisfies certain properties.

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
   - Syntax of CTL
   - Semantics of computation tree logic
   - Practical patterns of specifications
   - Important equivalences between CTL formulae
   - Adequate sets of CTL connectives
5. CTL* and the expressive powers of LTL and CTL

Computation Tree Logic (CTL)

- Computation Tree Logic (CTL) is a branching-time logic.
- As usual, we fix a set `Atoms` of atomic formulae $p, q, \ldots$.

Definition

Computation Tree Logic (CTL) has the following syntax:

$$\begin{array}{rcl}\phi & ::= & \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \implies \phi) \mid \mathbf{AX}\phi \mid \mathbf{EX}\phi \mid \\ & & \mathbf{AF}\phi \mid \mathbf{EF}\phi \mid \mathbf{AG} \mid \mathbf{EG} \mid \mathbf{A}[\phi\,\mathbf{U}\,\phi] \mid \mathbf{E}[\phi\,\mathbf{U}\,\phi]\end{array}$$

where $p \in$ `Atoms` is an atomic formula.

- Observe that CTL temporal connectives are either **A** (for all paths) or **E** (there exists) followed by an LTL temporal connectives $\mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{U}$.

Convention and Examples

- By convention, binding powers of CTL connectives are:

strongest	$\rightarrow$	weakest
$\neg, \mathbf{AX}, \mathbf{EX}, \mathbf{AF}, \mathbf{EF}, \mathbf{AG}, \mathbf{EG}$	$\wedge, \vee$	$\Longrightarrow, \mathbf{AU}, \mathbf{EU}$

- Examples:

$$\begin{array}{lll} \mathbf{AG}(q \Longrightarrow \mathbf{EG}r) & \mathbf{EFE}[r\,\mathbf{U}\,q] & \mathbf{A}[p\,\mathbf{U}\,\mathbf{EF}r] \\ \mathbf{EFEG}p \Longrightarrow \mathbf{AF}r & \mathbf{A}[p_1\,\mathbf{U}\,\mathbf{A}[p_2\,\mathbf{U}\,p_3]] & \mathbf{E}[\mathbf{A}[p_1\,\mathbf{U}\,p_2]\,\mathbf{U}\,p_3] \end{array}$$

$$\mathbf{AG}(p \Longrightarrow \mathbf{A}[p\,\mathbf{U}\,(\neg p \wedge \mathbf{A}[\neg p\,\mathbf{U}\,q])])$$

- Non-examples:

$$\begin{array}{lll} \cancel{\mathbf{EFG}r} & \cancel{\mathbf{A}\neg\mathbf{G}\neg p} & \cancel{\mathbf{F}[r\,\mathbf{U}\,q]} \\ \cancel{\mathbf{EF}(r\,\mathbf{U}\,q)} & \cancel{\mathbf{AEF}r} & \cancel{\mathbf{A}[(r\,\mathbf{U}\,q) \wedge (p\,\mathbf{U}\,r)]} \end{array}$$

- A subformula of a CTL formula ϕ is any formula ψ whose parse tree is a subtree of ϕ's parse tree.

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
   - Syntax of CTL
   - Semantics of computation tree logic
   - Practical patterns of specifications
   - Important equivalences between CTL formulae
   - Adequate sets of CTL connectives
5. CTL* and the expressive powers of LTL and CTL

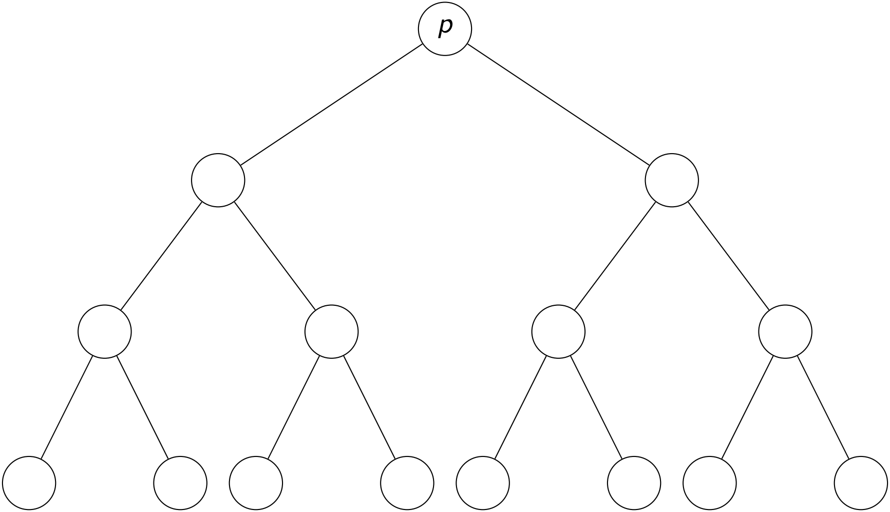
$\mathcal{M}, s \vDash \phi$

Definition

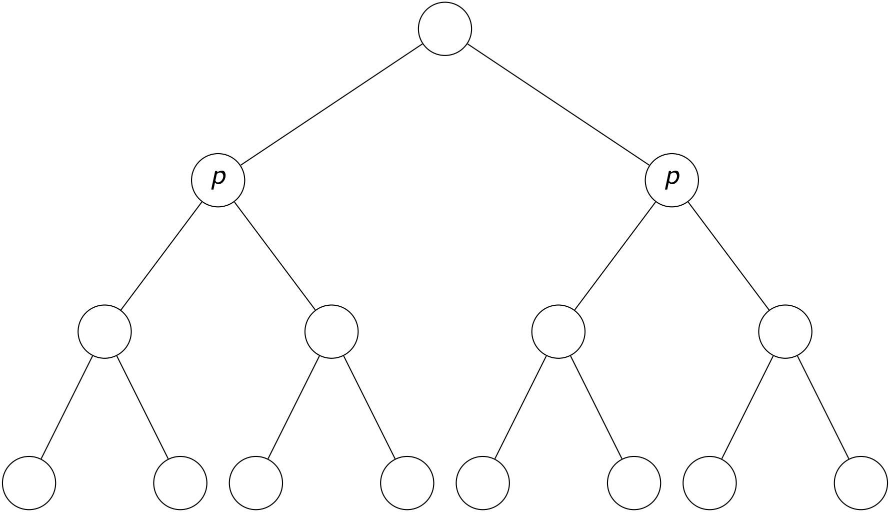
Let $\mathcal{M} = (S, \rightarrow, L)$ be a transition system, $s \in S$, and ϕ a CTL formula. $\underline{\mathcal{M}, s \vDash \phi}$ is defined as follows.

1. $\mathcal{M}, s \vDash \top$; $\mathcal{M}, s \nvDash \bot$; $\mathcal{M}, s \vDash p$ if $p \in L(s)$; $\mathcal{M}, s \vDash \neg\phi$ if $\mathcal{M}, s \nvDash \phi$;
2. $\mathcal{M}, s \vDash \phi \wedge \psi$ if $\mathcal{M}, s \vDash \phi$ and $\mathcal{M}, s \vDash \psi$; $\mathcal{M}, s \vDash \phi \vee \psi$ if $\mathcal{M}, s \vDash \phi$ or $\mathcal{M}, s \vDash \psi$;
3. $\mathcal{M}, s \vDash \phi \implies \psi$ if $\mathcal{M}, s \vDash \psi$ whenever $\mathcal{M}, s \vDash \phi$;
4. $\mathcal{M}, s \vDash \mathbf{AX}\phi$ if for all s' that $s \rightarrow s'$ we have $\mathcal{M}, s' \vDash \phi$;
5. $\mathcal{M}, s \vDash \mathbf{EX}\phi$ if for some s' that $s \rightarrow s'$ we have $\mathcal{M}, s' \vDash \phi$;
6. $\mathcal{M}, s \vDash \mathbf{AG}\phi$ if for all paths $s_0(= s) \rightarrow s_1 \rightarrow \cdots$ we have $\mathcal{M}, s_i \vDash \phi$ for every $i \geq 0$;
7. $\mathcal{M}, s \vDash \mathbf{EG}\phi$ if for some path $s_0(= s) \rightarrow s_1 \rightarrow \cdots$ we have $\mathcal{M}, s_i \vDash \phi$ for every $i \geq 0$;
8. $\mathcal{M}, s \vDash \mathbf{AF}\phi$ if for all paths $s_0(= s) \rightarrow s_1 \rightarrow \cdots$ we have $\mathcal{M}, s_i \vDash \phi$ for some $i \geq 0$;
9. $\mathcal{M}, s \vDash \mathbf{EF}\phi$ if for some path $s_0(= s) \rightarrow s_1 \rightarrow \cdots$ we have $\mathcal{M}, s_i \vDash \phi$ for some $i \geq 0$;
10. $\mathcal{M}, s \vDash \mathbf{A}[\phi \,\mathbf{U}\, \psi]$ if for all paths $s_0(= s) \rightarrow s_1 \rightarrow \cdots$ there is $i \geq 0$ that $\mathcal{M}, s_i \vDash \psi$ and for every $0 \leq j < i$ we have $\mathcal{M}, s_j \vDash \phi$;
11. $\mathcal{M}, s \vDash \mathbf{E}[\phi \,\mathbf{U}\, \psi]$ if for some path $s_0(= s) \rightarrow s_1 \rightarrow \cdots$ there is $i \geq 0$ that $\mathcal{M}, s_i \vDash \psi$ and for every $0 \leq j < i$ we have $\mathcal{M}, s_j \vDash \phi$.

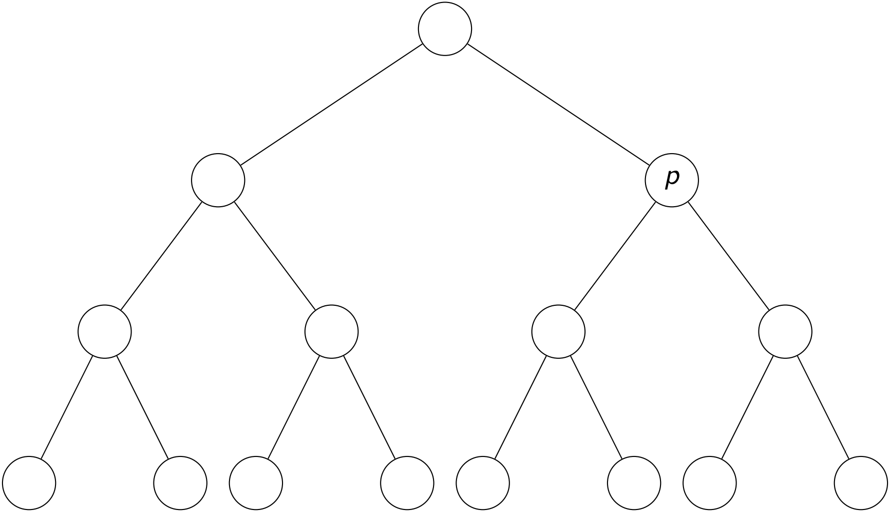
Visualization – $\mathcal{M}, s \vDash p$



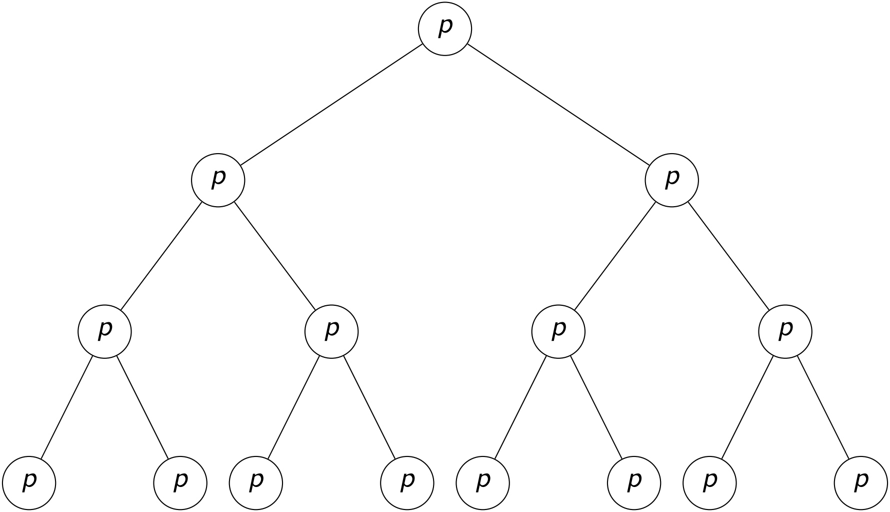
Visualization – $\mathcal{M}, s \vDash \mathbf{AX}p$



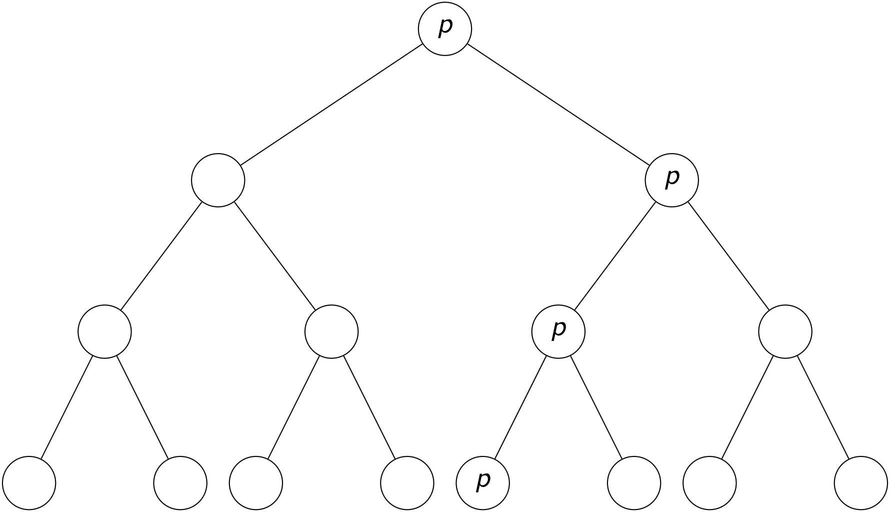
Visualization – $\mathcal{M}, s \vDash \mathbf{EX}p$



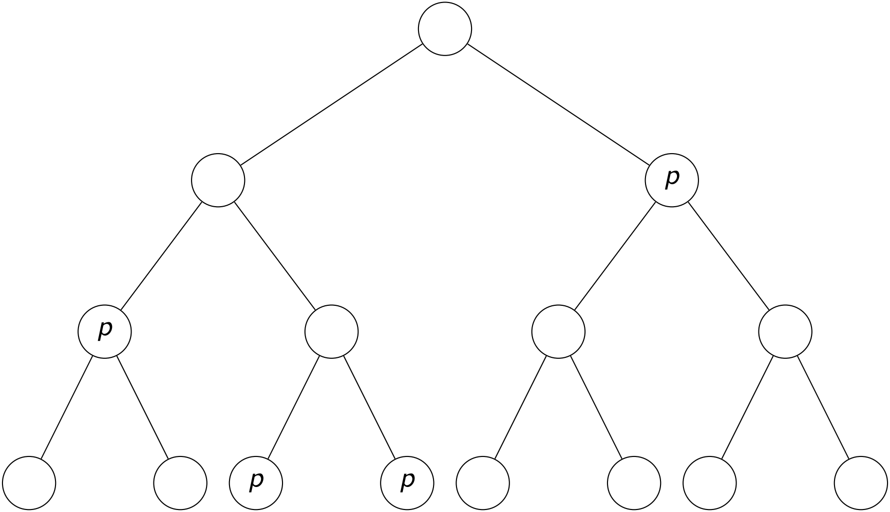
Visualization – $\mathcal{M}, s \models \mathbf{AG}p$



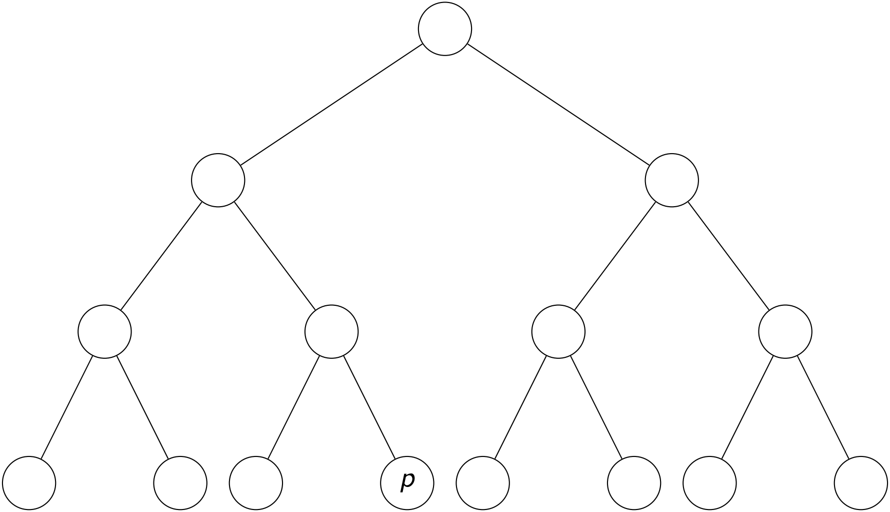
Visualization – $\mathcal{M}, s \vDash \mathbf{EG}p$



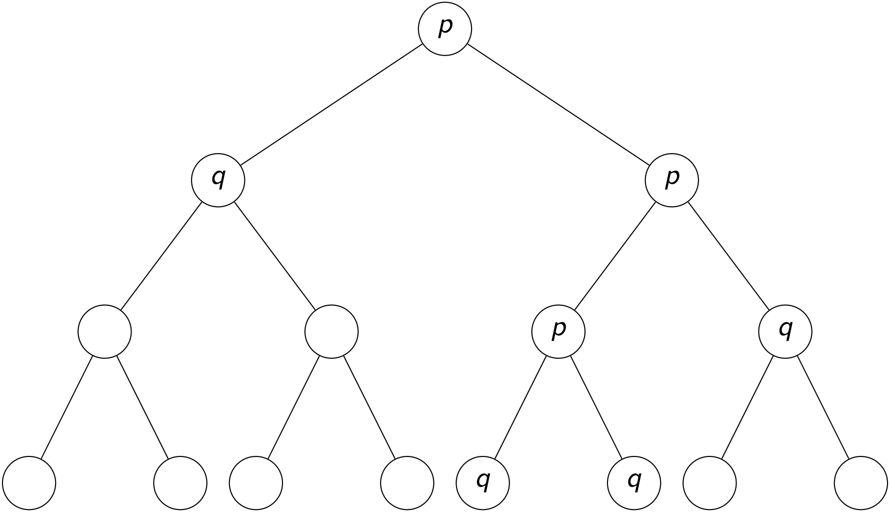
Visualization – $\mathcal{M}, s \vDash \mathbf{AF}p$



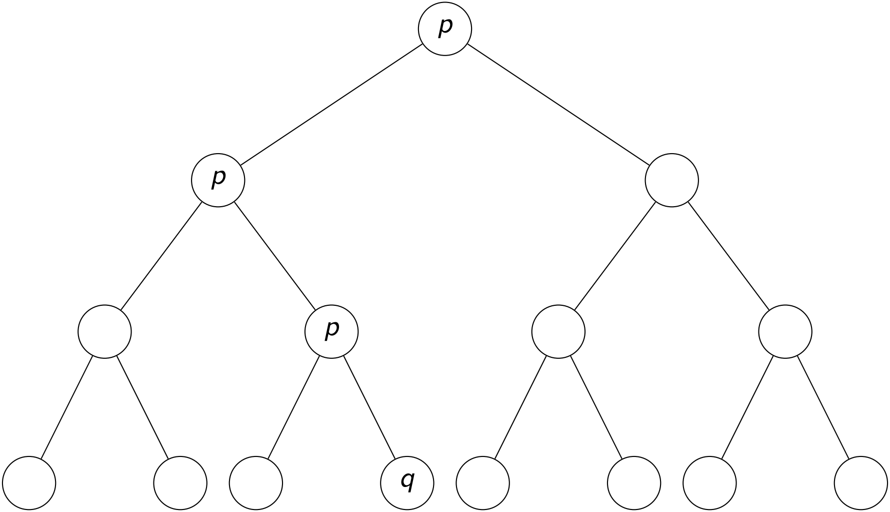
Visualization – $\mathcal{M}, s \models \mathbf{EF}p$



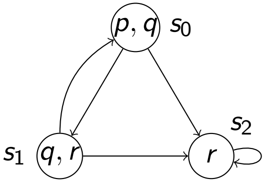
Visualization – $\mathcal{M}, s \vDash \mathbf{A}[p \,\mathbf{U}\, q]$



Visualization – $\mathcal{M}, s \vDash \mathbf{E}[p \,\mathbf{U}\, q]$

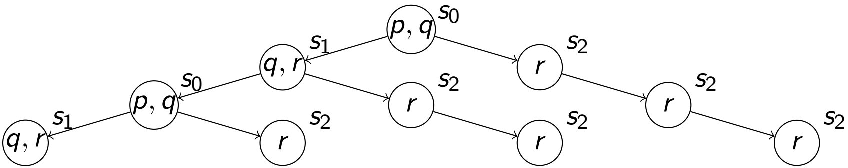


Example I



- Recall the transition system $\mathcal{M}$ above.
- We will give examples of $\mathcal{M}, s \vDash \phi$.
- To do so, we first "unroll" $\mathcal{M}$ from its initial state s_0.

Example II



- $\mathcal{M}, s_0 \vDash p \wedge q$; $\mathcal{M}, s_0 \vDash \neg r$; $\mathcal{M}, s_0 \vDash \top$;
- $\mathcal{M}, s_0 \vDash \mathbf{EX}(q \wedge r)$; $\mathcal{M}, s_0 \vDash \neg\mathbf{AX}(q \wedge r)$;
- $\mathcal{M}, s_0 \vDash \neg\mathbf{EF}(p \wedge r)$ since a state with p, r is not reachable from s_0;
- $\mathcal{M}, s_2 \vDash \mathbf{EG}r$ since $s_2 \to s_2 \to \cdots$;
- $\mathcal{M}, s_0 \vDash \mathbf{AF}r$ since r is always reachable from s_0;
- $\mathcal{M}, s_0 \vDash \mathbf{E}[(p \wedge q)\,\mathbf{U}\,r]$ since $s_0 \to s_2 \to s_2 \to \cdots$;
- $\mathcal{M}, s_0 \vDash \mathbf{A}[p\,\mathbf{U}\,r]$ since $s_0 \to s_1 \to \cdots$ and $s_0 \to s_2 \to \cdots$;
- $\mathcal{M}, s_0 \vDash \mathbf{AG}(p \vee q \vee r \implies \mathbf{EFEG}r)$.

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
   - Syntax of CTL
   - Semantics of computation tree logic
   - Practical patterns of specifications
   - Important equivalences between CTL formulae
   - Adequate sets of CTL connectives
5. CTL^* and the expressive powers of LTL and CTL

Patterns of Specifications

- It is possible to get to a state where *started* holds but *ready* doesn't: $\mathbf{EF}(started \wedge \neg ready)$;
- For any state, if a *requested* occurs, it will be *acknowledged* eventually: $\mathbf{AG}(requested \implies \mathbf{AF} acknowledged)$;
- A process is *enabled* infinitely often on all paths: $\mathbf{AG}(\mathbf{AF} enabled)$;
- A process will eventually be *stable* permanently: $\mathbf{AF}(\mathbf{AG} stable)$;
- From any state it is possible to get to a *restart* state: $\mathbf{AG}(\mathbf{EF} restart)$;
- Process P_1 can always request to enter its critical section: $\mathbf{AG}(n_1 \implies \mathbf{EX} t_1)$.
  - Recall that non-blocking is not expressible in LTL.
- Consider the following property: "if a process is *enabled* infinitely often, it is *running* infinitely often. This is not expressed by $\mathbf{AGAF} enabled \implies \mathbf{AGAF} running$.
  - In fact, this property is not expressible in CTL.
  - We can express the property by $\mathbf{GF} enabled \implies \mathbf{GF} running$ in LTL.

Outline

Semantic Equivalences

Definition

Let ϕ and ψ be CTL formulae. ϕ and ψ are semantically equivalent (written $\phi \equiv \psi$) if for all models $\mathcal{M}$ and all state s in $\mathcal{M}$, $\mathcal{M}, s \vDash \phi$ iff $\mathcal{M}, s \vDash \psi$.

- Since **A** is a universal quantifier and **E** an existential quantifier, it is not hard to see the following semantically equivalent formulae:

$$\neg \mathbf{AF}\phi \equiv \mathbf{EG}\neg\phi \qquad \neg \mathbf{EF}\phi \equiv \mathbf{AG}\neg\phi \qquad \neg \mathbf{AX}\phi \equiv \mathbf{EX}\neg\phi.$$

- Moreover, by similar arguments in LTL, we have

$$\mathbf{AF}\phi \equiv \mathbf{A}[\top\,\mathbf{U}\,\phi] \qquad \mathbf{EF}\phi \equiv \mathbf{E}[\top\,\mathbf{U}\,\phi].$$

- Finally, we have

$$\begin{array}{rclcrcl} \mathbf{AG}\phi & \equiv & \phi \wedge \mathbf{AXAG}\phi & \quad & \mathbf{EG}\phi & \equiv & \phi \wedge \mathbf{EXEG}\phi \\ \mathbf{AF}\phi & \equiv & \phi \vee \mathbf{AXAF}\phi & & \mathbf{EF}\phi & \equiv & \phi \vee \mathbf{EXEF}\phi \end{array}$$

$$\mathbf{A}[\phi\,\mathbf{U}\,\psi] \equiv \psi \vee (\phi \wedge \mathbf{AXA}[\phi\,\mathbf{U}\,\psi])$$

$$\mathbf{E}[\phi\,\mathbf{U}\,\psi] \equiv \psi \vee (\phi \wedge \mathbf{EXE}[\phi\,\mathbf{U}\,\psi]).$$

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
   - Syntax of CTL
   - Semantics of computation tree logic
   - Practical patterns of specifications
   - Important equivalences between CTL formulae
   - Adequate sets of CTL connectives
5. CTL* and the expressive powers of LTL and CTL

Adequate Sets of CTL Connectives I

- $\{\mathbf{AU}, \mathbf{EU}, \mathbf{AX}\}$ is an adequate set of CTL connectives.
  - Observe that

$$\mathbf{AX}\phi \equiv \neg\mathbf{EX}\neg\phi$$

$$\begin{array}{rclcrcl}
\mathbf{AG}\phi & \equiv & \neg\mathbf{EF}\neg\phi & \quad & \mathbf{EG}\phi & \equiv & \neg\mathbf{AF}\neg\phi \\
\mathbf{AF}\phi & \equiv & \mathbf{A}[\top\,\mathbf{U}\,\phi] & \quad & \mathbf{EF}\phi & \equiv & \mathbf{E}[\top\,\mathbf{U}\,\phi].
\end{array}$$

- $\{\mathbf{EG}, \mathbf{EU}, \mathbf{EX}\}$ is an adequate set of CTL connectives.
  - Recall $\phi\,\mathbf{U}\,\psi \equiv \neg(\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi)) \wedge \mathbf{F}\psi$ in LTL.
  - Observe that $\mathbf{A}[\phi\,\mathbf{U}\,\psi] \equiv \neg(\mathbf{E}[\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi)] \vee \mathbf{EG}\neg\psi)$.

$$\begin{array}{rcl}
\mathbf{A}[\phi\,\mathbf{U}\,\psi] & \equiv & \mathbf{A}[\neg(\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi)) \wedge \mathbf{F}\psi] \\
& \equiv & \neg\mathbf{E}\neg[\neg(\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi)) \wedge \mathbf{F}\psi] \\
& \equiv & \neg\mathbf{E}[(\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi)) \vee \mathbf{G}\neg\psi] \\
& \equiv & \neg(\mathbf{E}[\neg\psi\,\mathbf{U}\,(\neg\phi \wedge \neg\psi)] \vee \mathbf{EG}\neg\psi).
\end{array}$$

  - Strictly speaking, we need CTL* for the proof.

Adequate Sets of CTL Connectives II

- More generally, we have

Theorem

A set of CTL temporal connectives is adequate iff it contains at least one of $\{\mathbf{AX}, \mathbf{EX}\}$, *at least one of* $\{\mathbf{EG}, \mathbf{AF}, \mathbf{AU}\}$, *and* $\mathbf{EU}$.

- We can also define **AR**, **ER**, **AW**, and **EW** by **EU**:

$$\begin{aligned} \mathbf{A}[\phi\,\mathbf{R}\,\psi] &\triangleq \neg\mathbf{E}[\neg\phi\,\mathbf{U}\,\neg\psi] & \mathbf{E}[\phi\,\mathbf{R}\,\psi] &\triangleq \neg\mathbf{A}[\neg\phi\,\mathbf{U}\,\neg\psi] \\ \mathbf{A}[\phi\,\mathbf{W}\,\psi] &\triangleq \mathbf{A}[\psi\,\mathbf{R}\,(\phi\vee\psi)] & \mathbf{E}[\phi\,\mathbf{W}\,\psi] &\triangleq \mathbf{E}[\psi\,\mathbf{R}\,(\phi\vee\psi)] \end{aligned}$$

Outline

LTL and CTL

- In LTL, we have temporal connectives $\mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{U}$.
- In CTL, we have temporal connectives $\mathbf{AX}, \mathbf{EX}, \mathbf{AF}, \mathbf{EF}, \mathbf{AG}, \mathbf{EG}, \mathbf{AU}, \mathbf{EU}$.
- Observe that CTL temporal connectives are LTL connectives prefixed by path quantifiers **A** or **E**.
- Consider the property: for some path, if p occurs eventually, then q occurs eventually.
- Using LTL, one would write $\mathbf{F}p \implies \mathbf{F}q$.
  - But LTL considers all paths from a specified state.
- Using CTL, one would write $\mathbf{EF}p \implies \mathbf{EF}q$.
  - But this is not what we want either.
- How about $\mathbf{E}(\mathbf{F}p \implies \mathbf{F}q)$?
  - But this is not in LTL nor CTL!
- We need a more expressive logic.

The Syntax of CTL*

Definition

CTL* has the following syntax:

- state formulae

$$\phi ::= \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid \mathbf{A}[\alpha] \mid \mathbf{E}[\alpha]$$

  where $p \in \texttt{Atoms}$ is an atomic formula

- path formulae

$$\alpha ::= \phi \mid (\neg\alpha) \mid (\alpha \wedge \alpha) \mid (\alpha\, \mathbf{U}\, \alpha) \mid (\mathbf{G}\alpha) \mid (\mathbf{F}\alpha) \mid (\mathbf{X}\alpha)$$

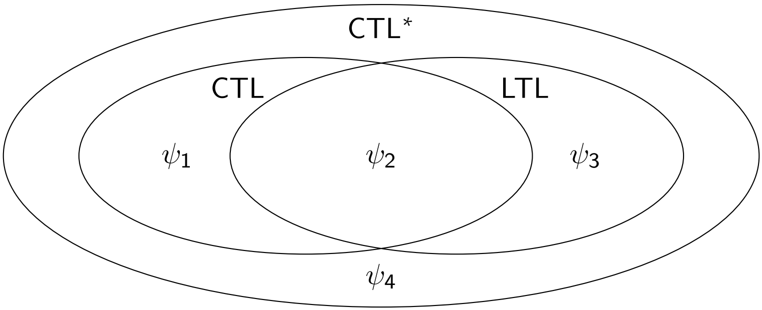
LTL, CTL, and CTL*

- An LTL formula is in fact a CTL* path formula.
- Since we consider all paths in LTL, an LTL formula α is in fact the CTL* state formula $\mathbf{A}[\alpha]$.
- We obtain CTL by restricting path formulae to

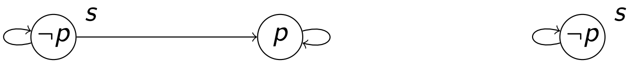
$$\alpha ::= (\phi\, \mathbf{U}\, \phi) \mid (\mathbf{G}\phi) \mid (\mathbf{F}\phi) \mid (\mathbf{X}\phi)$$

- Hence both LTL and CTL are subclasses of CTL*.

Expressive Powers of LTL, CTL, and CTL*



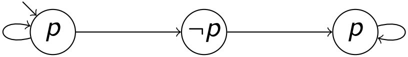
- $\psi_1 = \mathbf{AGEF}p$.



- $\psi_2 = \mathbf{AG}(p \implies \mathbf{AF}q)(\text{CTL}) = \mathbf{G}(p \implies \mathbf{F}q)(\text{LTL})$.
- $\psi_3 = \mathbf{GF}p \implies \mathbf{F}q$.
- $\psi_4 = \mathbf{E}[\mathbf{GF}p]$.

More Examples

- **FG**p and **AFAG**p are not equivalent. why



- **XF**$p \equiv$ **FX**p (LTL) and **AXAF**p (CTL) are equivalent. But **AFAX**$p \not\equiv$ **AXAF**p. why
- So, between LTL and CTL, which one is better?
  - In 1980's, many debates (and papers) were fought for the question.
  - Now, people generally believe they are just different: one is not better than the other.
  - Try to compare LTL and CTL.

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL
6. Model-checking algorithms
   - The CTL model-checking algorithm
   - CTL model checking with fairness
   - The LTL model-checking algorithm

The Model Checking Problem

- Let $\mathcal{M} = (S, \rightarrow, L)$ be a model, $s \in S$ a state, and ϕ a temporal logic formula.
- The model checking problem is to decide whether $\mathcal{M}, s \vDash \phi$ holds.
- We will discuss two model checking algorithms: one for LTL, the other for CTL.
- These algorithms help us understand basic principles of various verification tools (such as NuSMV).
  - NuSMV does not implement the algorithms we discuss here.
  - Yet the basic ideas are not very different.

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL
6. Model-checking algorithms
   - The CTL model-checking algorithm
   - CTL model checking with fairness
   - The LTL model-checking algorithm

The CTL Model Checking Algorithm I

- Let us first consider deciding whether $\mathcal{M}, s_0 \vDash \phi$ where $\mathcal{M}$ is a finite transition system and ϕ a CTL formula.
  - That is, $\mathcal{M} = (S, \rightarrow, L)$ with S a finite set of states.
  - There are algorithms that solve the model checking problem for certain infinite transition systems.
- Our algorithm in fact computes all states that satisfy the top CTL formula.
  - That is, it computes the set $\{s \in S : \mathcal{M}, s \vDash \phi\}$.
- After computing all states satisfying the given CTL formula, the model checking problem is solved easily.
  - We simply check if $s_0 \in \{s \in S : \mathcal{M}, s \vDash \phi\}$.

The CTL Model Checking Algorithm II

- In our algorithm, we only consider temporal connectives $\{\mathbf{EX}, \mathbf{AF}, \mathbf{EU}\}$.
  - $\{\mathbf{EX}, \mathbf{AF}, \mathbf{EU}\}$ is adequate.
- For each state, we label it with subformulae of the given CTL formula.
  - A state satisfies all subformulae in its label.
- Start from smallest subformulae; the algorithm works on each subformula until the given CTL formula.

The CTL Model Checking Algorithm III

Input: $\mathcal{M} = (S, \rightarrow, L)$: a mode; ϕ : a CTL formula
Output: $\{s \in S : \mathcal{M}, s \vDash \phi\}$
foreach subformula ψ of ϕ **do**
  switch ψ **do**
    case $\perp$: **do** **continue**;
    case p: **do** label all s with p if $p \in L(s)$;
    case $\psi_1 \wedge \psi_2$: **do** label all s with ψ if it is lablled with ψ_1, ψ_2 ;
    case $\neg\psi_1$: **do** label all s with ψ if it is not labelled with ψ_1 ;
    case $\mathbf{EX}\psi_1$: **do** label all s with ψ if one of its successors is labelled with ψ_1 ;
    case $\mathbf{AF}\psi_1$: **do** label all s with ψ if it is labelled with ψ_1, or all successors of s are labeleed with ψ until no change ;
    case $\mathbf{E}[\psi_1 \mathbf{U} \psi_2]$: **do** label all s with ψ if it is labelled with ψ_2, or s is labelled with ψ_1 and one of its successors is labelled with ψ until no change ;

The CTL Model Checking Algorithm IV

- Let $|\phi|$ be the number of connectives in ϕ.
- Let $\mathcal{M} = (S, \rightarrow, L)$ be a transition system.
- The complexity of the algorithm is $O(|\phi| \cdot |S| \cdot (|S| + |\rightarrow|))$.
  - There are $O(|\phi|)$ subformulae.
  - For $\mathbf{AF}\psi_1$ and $\mathbf{E}[\psi_1 \,\mathbf{U}\, \psi_2]$, each iteration takes $O(|S| + |\rightarrow|)$ steps; there are at most $O(|S|)$ iterations.
- We now present the pseudo algorithm $\mathtt{SAT}(\mathcal{M}, \phi)$.

switch $\underline{\phi}$ **do**

- **case** $\underline{\top}$: **do** **return** S ;
- **case** $\underline{\bot}$: **do** **return** $\varnothing$;
- **case** $\underline{p}$: **do** **return** $\{s \in S : \phi \in L(s)\}$;
- **case** $\underline{\neg\phi_1}$: **do** **return** $S \setminus \mathtt{SAT}(\mathcal{M}, \phi_1)$;
- **case** $\underline{\phi_1 \wedge \phi_2}$: **do** **return** $\mathtt{SAT}(\mathcal{M}, \phi_1) \cap \mathtt{SAT}(\mathcal{M}, \phi_2)$;
- **case** $\underline{\mathbf{EX}\phi_1}$: **do** **return** $\mathtt{SAT}_{\mathtt{EX}}(\mathcal{M}, \phi_1)$;
- **case** $\underline{\mathbf{E}[\phi_1 \mathbf{\ U\ } \phi_2]}$: **do** **return** $\mathtt{SAT}_{\mathtt{EU}}(\mathcal{M}, \phi_1, \phi_2)$;
- **case** $\underline{\mathbf{AF}\phi_1}$: **do** **return** $\mathtt{SAT}_{\mathtt{AF}}(\mathcal{M}, \phi_1)$;

$\text{pre}_\exists(\mathcal{M}, Y)$ and $\text{pre}_\forall(\mathcal{M}, Y)$

- In order to give the pseudo algorithm for $\texttt{SAT}_{\texttt{EX}}(\mathcal{M}, \phi)$, $\texttt{SAT}_{\texttt{EU}}(\mathcal{M}, \phi, \psi)$, and $\texttt{SAT}_{\texttt{AF}}(\mathcal{M}, \phi)$, we need two functions:

$$
\begin{aligned}
\text{pre}_\exists(\mathcal{M}, Y) &\stackrel{\triangle}{=} \{s \in S : \text{there exists } s'(s \to s' \text{ and } s' \in Y)\} \\
\text{pre}_\forall(\mathcal{M}, Y) &\stackrel{\triangle}{=} \{s \in S : \text{for all } s'(s \to s' \text{ implies } s' \in Y)\}
\end{aligned}
$$

- $\text{pre}_\exists(\mathcal{M}, Y)$ consists of states whose successors intersect Y is not empty.
- $\text{pre}_\forall(\mathcal{M}, Y)$ consists of states whose successors are contained in Y.
- Observe that

$$\text{pre}_\forall(\mathcal{M}, Y) = S \setminus \text{pre}_\exists(\mathcal{M}, S \setminus Y)$$

$X \leftarrow \mathtt{SAT}(\mathcal{M}, \phi)$;
$Y \leftarrow \mathsf{pre}_{\exists}(\mathcal{M}, X)$;
return Y;

$\mathtt{SAT}_{\mathtt{AF}}(\mathcal{M}, \phi)$

$X \leftarrow S$;
$Y \leftarrow \mathtt{SAT}(\mathcal{M}, \phi)$;
repeat
  $X \leftarrow Y$;
  $Y \leftarrow Y \cup \mathrm{pre}_{\forall}(\mathcal{M}, Y)$;
until $\underline{X = Y}$;
return Y;

$\texttt{SAT}_{\texttt{EU}}(\mathcal{M}, \phi, \psi)$

$W \leftarrow \texttt{SAT}(\mathcal{M}, \phi)$;
$Y \leftarrow \texttt{SAT}(\mathcal{M}, \psi)$;
repeat
    $X \leftarrow Y$;
    $Y \leftarrow Y \cup (W \cap \mathsf{pre}_{\exists}(\mathcal{M}, Y))$;
until $\underline{X = Y}$;
return Y;

A More Efficient Model Checking Algorithm

- Using backward breadth-first search, we can in fact do better.
- We use the adequate set $\{\textbf{EX}, \textbf{EG}, \textbf{EU}\}$ instead.
- Observe that $\textbf{EX}\psi_1$ and $\textbf{E}[\psi_1 \,\textbf{U}\, \psi_2]$ can be labelled in time $O(|S| + |\rightarrow|)$ if we perform backward breadth-first search.
- For the case $\textbf{EG}\psi_1$,
  - Consider the subgraph with states satisfying ψ_1;
  - Find maximal strongly connected components (SCC's) in the subgraph;
  - Use backward breadth-first search on the subgraph to find states that can reach an SCC.
- The new algorithm takes time $O(|\phi| \cdot (|S| + |\rightarrow|))$.

The State Explosion Problem

- Although the complexity of CTL model checking algorithm is linear in the number of states, the number of states can be exponential in the number of variables and the number of components of the system.
  - Adding a Boolean variable can double the number of states.
- This is called the state explosion problem.
- Lots of researches try to overcome the state explosion problem.
  - Efficient data structures.
  - Abstraction.
  - Partial order reduction.
  - Induction.
  - Compositional reasoning.

Outline

Fairness I

- We often simplify the system when we build a model for it.
  - This is called abstraction.
- Abstraction sometimes introduces unrealistic model behaviors.
  - For instance, a process may stay in its critical section forever.
- Such unrealistic behaviors may disprove intended properties.
  - For instance, it is possible that the other process won't get to its critical section forever.
- In order to consider realistic model behaviors, we impose fairness constraints.

Fairness II

- We consider only fair computation paths specified by fair constraints.
- Instead of standard path quantifiers **A** (for all) and **E** (for some), we use their variants $\mathbf{A}_C$ (for all fair paths) and $\mathbf{E}_C$ (for some fair paths).
- As an example, we can ask
  "a process will eventually be permitted to enter its critical section if it requests so along all fair paths."
- Clearly, we have to slightly modify the CTL model checking algorithm.

Fair Computation Paths

Definition

Let $C = \{\psi_1, \psi_2, \ldots, \psi_n\}$ be fairness constraints. A computation path $s_0 \rightarrow s_1 \rightarrow \cdots$ is fair with respect to C if for every i there are infinitely s_j's such that $s_j \vDash \psi_i$. We write $\mathbf{A}_C$ and $\mathbf{E}_C$ for the path quantifers $\mathbf{A}$ and $\mathbf{E}$ restricted to fair paths.

- For example, $\mathcal{M}, s_0 \vDash \mathbf{A}_C\mathbf{G}\phi$ if ϕ holds in every state along all fair paths.

CTL Model Checking Algorithm with Fairness Constraints

- Let $C = \{\psi_1, \psi_2, \ldots, \psi_n\}$ be a set of fairness constraints.
- We consider the adequate set $\{\mathbf{E}_C\mathbf{U}, \mathbf{E}_C\mathbf{G}, \mathbf{E}_C\mathbf{X}\}$.
- Observe that

$$\begin{aligned}\mathbf{E}_C[\phi\, \mathbf{U}\, \psi] &\equiv \mathbf{E}[\phi\, \mathbf{U}\, (\psi \wedge \mathbf{E}_C\mathbf{G}\top)]\\ \mathbf{E}_C\mathbf{X}\phi &\equiv \mathbf{EX}(\phi \wedge \mathbf{E}_C\mathbf{G}\top).\end{aligned}$$

  - Note that a computation path is fair iff all its suffixes are fair.
  - $\mathcal{M}, s \vDash \mathbf{E}_C\mathbf{G}\top$ ensures that s is on a fair path.
- Since $\mathbf{E}_C\mathbf{U}$ and $\mathbf{E}_C\mathbf{X}$ can be reduced to $\mathbf{E}_C\mathbf{G}$, it remains to compute $\{s \in S : \mathcal{M}, s \vDash \mathbf{E}_C\mathbf{G}\phi\}$.

Computing $\{s \in S : \mathcal{M}, s \vDash \mathbf{E}_C\mathbf{G}\phi\}$

- It turns out that the algorithm for computing $\mathbf{E}_C\mathbf{G}$ is similar for computing $\mathbf{EG}$.
- Let $C = \{\psi_1, \psi_2, \ldots, \psi_n\}$ be a set of fairness constraints.
- To compute $\{s \in S : \mathcal{M}, s \vDash \mathbf{E}_C\mathbf{G}\phi\}$, do the following:
  - Consider the subgraph with states satisfying ϕ;
  - Find maximal strongly connected components (SCC's) in the subgraph;
  - Remove an SCC if it does not contain a state satisfying ψ_i for some i. The remaining SCC's are fair SCC's;
  - Use backward breadth-first search on the subgraph to find states that can reach a fair SCC.
- The complexity of the algorithm with fairness constraints is $O(|C| \cdot |\phi| \cdot (|S| + |\rightarrow|))$.

Fairness in NuSMV

- In NuSMV two different types of fair paths can be specified.
- Justice
  - Let $C = \{p_1, p_2, \dots, p_n\}$ be a set of atomic formulae.
  - A path $s_0 \to s_1 \to \cdots$ satisfies the justice constraint C if for every i, there are infinitely s_j's such that $s_j \vDash p_i$.
  - NuSMV uses the keywords `FAIRNESS` or `JUSTICE`.
- Compassion
  - Let $C = \{(p_1, q_1), (p_2, q_2), \dots, (p_n, q_n)\}$ be a set of pairs of atomic formulae.
  - A path $s_0 \to s_1 \to \cdots$ satisfies the compassion constraint C if for every i, there are infinitely s_j's such that $s_j \vDash p_i$ then there are infinitely s_j's such that $s_j \vDash q_i$.
  - NuSMV uses the keyword `COMPASSION`.

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL
6. Model-checking algorithms
   - The CTL model-checking algorithm
   - CTL model checking with fairness
   - The LTL model-checking algorithm

LTL Model Checking Algorithm

- The CTL model checking algorithm is rather straightforward.
  - It labels each state by satisfied subformulae.
  - It works because CTL formulae specify state properties.
- LTL formulae, on the other hand, specify path properties.
  - Labelling states no longer work.
- We need a formal model for paths.
- Automata theory is required!

Paths and Traces

- Let $\mathcal{M} = (S, \to, L)$ be a model.
- Recall that a path $s_0 \to s_1 \to \cdots$ is a sequence of states such that $s_i \to s_{i+1}$ for every $i \geq 0$.
- A trace is a sequence of valuations of propositional atoms.
- The trace of a path $s_0 \to s_1 \to \cdots$ is $L(s_0)L(s_1)\cdots$.
  - Recall that $L : S \to 2^{\texttt{Atoms}}$.
  - $L(s_i)$ is the set of propositional atoms that hold in s_i.
  - $L(s_i)$ hence is a valuation of propositional atoms.
- Note that different paths may have the same trace.

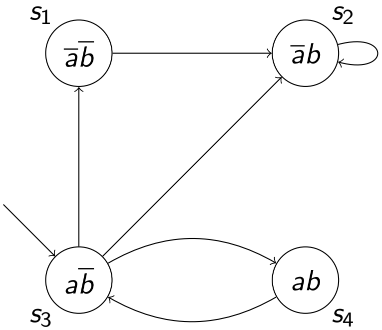
Basic Ideas

- Let $\mathcal{M} = (S, \rightarrow, L)$ be a model, $s \in S$, and ϕ an LTL formula.
- We check whether $\mathcal{M}, s \vDash \phi$ holds as follows.
  1. Construct an automaton $A_{\neg\phi}$. $A_{\neg\phi}$ accepts the traces satisfying $\neg\phi$;
  2. Construct the combination of the automaton $A_{\neg\phi}$ and $\mathcal{M}$.
     - ★ Each path of the combination is a path of $A_{\neg\phi}$ and also a path of $\mathcal{M}$.
  3. Check if there is an accepting path starting from a combined state including s.
     - ★ Such a path is a path of $\mathcal{M}$ from s.
     - ★ Moreover its trace satisfying $\neg\phi$.

  If an accepting path is found, we report "$\mathcal{M}, s \nvDash \phi$"; Otherwise, "$\mathcal{M}, s \vDash \phi$."

Illustration I

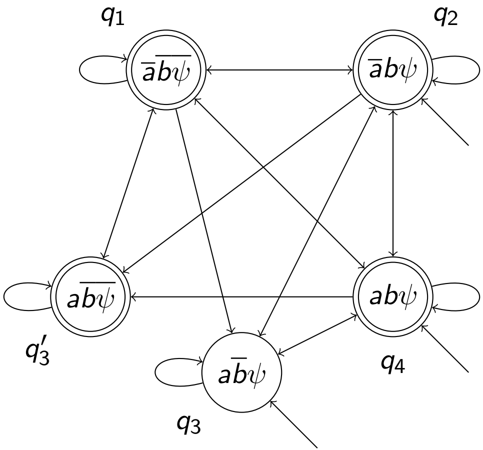
```
init (a) := 1;
init (b) := 0;
next (a) := case
          !a :  0;
           b :  1;
           1 :  { 0, 1 };
               esac;
next (b) := case
           a & next (a) :  !b;
          !a :  1;
           1 :  { 0, 1 };
               esac;
```



$\mathcal{M} = (S, \rightarrow, L)$ with $\texttt{Atoms} = \{a, b\}$ and $\phi = \neg(a\ \mathbf{U}\ b)$

Does $\mathcal{M}, s_3 \vDash \phi$ hold?

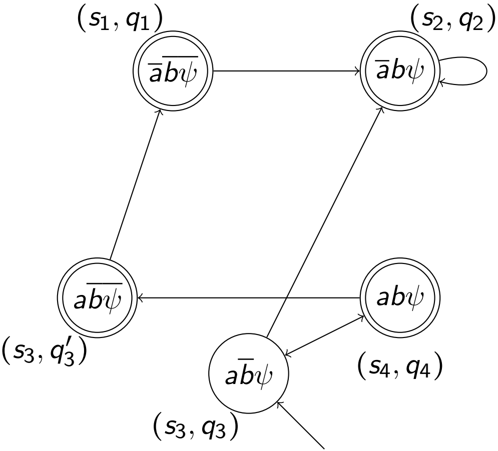
Illustration II



- A trace t is accepting if there is a path π whose trace is t such that an accepting state occurs infinitely often.
- $\{a\}\{a\}\{a,b\}\{a\}\{a\}\cdots$ is accepting.
- $\{a\}\{a\}\cdots$ is not.

$$\phi = \neg(a \mathbf{U} b), \psi = (a \mathbf{U} b) \text{ and } A_\psi$$

Illustration III



- $(s_3, q_3)(s_2, q_2)\cdots$ is accepting.
- Hence $\mathcal{M}, s_3 \not\models \neg(a \mathbf{U} b)$ because of the path $s_3 s_2 \cdots$.
- In fact, $s_3 s_4 s_3 s_4 \cdots$ is another counterexample.

Combination of $\mathcal{M}$ and A_ψ

Construct A_ϕ I

- Let ϕ be an LTL formula.
- We want to construct an automaton A_ϕ such that A_ϕ accepts precisely traces on which ϕ holds.
- We assume ϕ contains only the temporal connectives **U** and **X**.
  - Recall that $\{\mathbf{U}, \mathbf{X}\}$ is adequate.
- Define the closure $\mathcal{C}(\phi)$ of an LTL formula ϕ by

$$\mathcal{C}(\phi) \triangleq \{\psi, \neg\psi : \psi \text{ is a subformula of } \phi\}$$

  where we identify $\neg\neg\psi$ and ψ.
- Example:
  - $\mathcal{C}(a\,\mathbf{U}\,b) = \{a, b, \neg a, \neg b, a\,\mathbf{U}\,b, \neg(a\,\mathbf{U}\,b)\}$.

Construct A_ϕ II

- Let ϕ be an LTL formula.
- A maximal subset q of $\mathcal{C}(\phi)$ satisfies the following:
  - for all (non-negated) $\psi \in \mathcal{C}(\phi)$, either $\psi \in q$ or $\neg\psi \in q$;
  - $\psi_1 \vee \psi_2 \in q$ iff $\psi_1 \in q$ or $\psi_2 \in q$;
  - Conditions for other Boolean connectives are similar;
  - If $\psi_1 \,\mathbf{U}\, \psi_2 \in q$, then $\psi_2 \in q$ or $\psi_1 \in q$; and
  - If $\neg(\psi_1 \,\mathbf{U}\, \psi_2) \in q$, then $\neg\psi_2 \in q$.
- The states of A_ϕ are the maximal subsets of $\mathcal{C}(\phi)$. That is, $\{q \subseteq \mathcal{C}(\phi) : q \text{ is maximal}\}$.
  - Informally, $\psi \in q$ means $\psi \in \mathcal{C}$ is true at state q.
- The initial states of A_ϕ are those containing ϕ. Formally, $\{q \subseteq \mathcal{C}(\phi) : q \text{ is maximal and } \phi \in q\}$.

Construct A_ϕ III

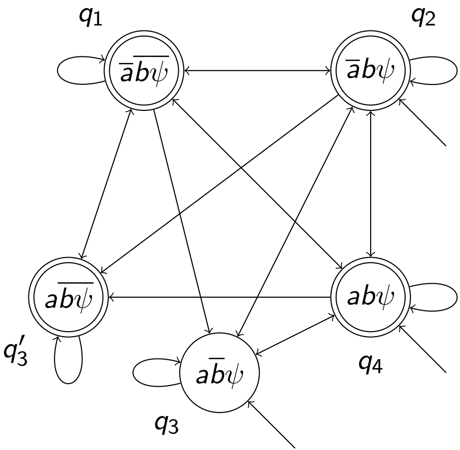
- The transition relation δ of A_ϕ is defined as follows. $(q, q') \in \delta$ if
  - $\mathbf{X}\psi \in q$ implies $\psi \in q'$;
  - $\neg\mathbf{X}\psi \in q$ implies $\neg\psi \in q'$;
  - $\psi_1 \mathbf{U} \psi_2 \in q$ and $\psi_2 \notin q$ imply $\psi_1 \mathbf{U} \psi_2 \in q'$; and
  - $\neg(\psi_1 \mathbf{U} \psi_2) \in q$ and $\psi_1 \in q$ imply $\neg(\psi_1 \mathbf{U} \psi_2) \in q'$.
- Informally, $\psi \in q$ enforces certain $\psi' \in q'$.
  - Recall the definition of maximal subsets of $\mathcal{C}(\phi)$.
  - Observe that

$$\begin{array}{rcl} \psi_1 \mathbf{U} \psi_2 & \equiv & \psi_2 \vee (\psi_1 \wedge \mathbf{X}(\psi_1 \mathbf{U} \psi_2) \\ \neg(\psi_1 \mathbf{U} \psi_2) & \equiv & \neg\psi_2 \wedge (\neg\psi_1 \vee \mathbf{X}\neg(\psi_1 \mathbf{U} \psi_2)). \end{array}$$

- Consider $\mathcal{C}(a \mathbf{U} b) = \{a, \neg a, b, \neg b, a \mathbf{U} b, \neg(a \mathbf{U} b)\}$.
- Let $q = \{a, \neg b, a \mathbf{U} b\}$ be a maximal subset of $\mathcal{C}(a \mathbf{U} b)$.
- We have $(q, q) \in \delta$, the transition relation of $A_{a\mathbf{U}b}$.
- Informally, $a \mathbf{U} b \in q$ means $a \mathbf{U} b$ holds at all traces from q.
  - Apparently, $qq\cdots$ does not satisfy $a \mathbf{U} b$.
- We define acceptance conditions to disallow such traces.

Construct A_ϕ IV

- Let $\chi_1 \, \mathbf{U} \, \psi_1, \ldots, \chi_k \, \mathbf{U} \, \psi_k$ be all formulae of this form in $\mathcal{C}(\phi)$.
- The acceptance condition of A_ϕ is as follows.
  A state q of A_ϕ is i-accepting if $\{\neg(\chi_i \, \mathbf{U} \, \psi_i), \psi_i\} \cap q \neq \varnothing$.
  A path π is accepted if for every $1 \leq i \leq k$, π has infinitely many i-accepting states.
  - A state can be i-accepting for every $1 \leq i \leq k$.
- To see why it works, consider a path $\pi = q_0 \to q_1 \to \cdots$ with only finitely many i-accepting states. Hence for some h, we have $\pi^h = q_h \to q_{h+1} \to \cdots$ with $q_j \cap \{\neg(\chi_i \, \mathbf{U} \, \psi_i), \psi_i\} = \varnothing$ for every $j \geq h$. By the maximality of q_j, we have $\{\chi_i \, \mathbf{U} \, \psi_i, \neg\psi_i\} \subseteq q_j$ for every $j \geq h$. The path π is precisely what we wan to eliminate.

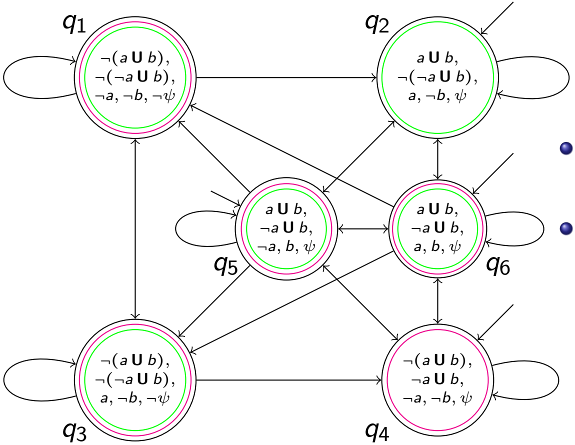


- $\mathcal{C}(a \,\mathbf{U}\, b) =$ $\{a, \neg a, b, \neg b, a \,\mathbf{U}\, b, \neg(a \,\mathbf{U}\, b)\}$.
- Maximal subsets of $\mathcal{C}(a \,\mathbf{U}\, b)$:

$$
\begin{array}{lcl}
q_1 & = & \{\neg a, \neg b, \neg(a \,\mathbf{U}\, b)\} \\
q_2 & = & \{\neg a, b, a \,\mathbf{U}\, b\} \\
q_3 & = & \{a, \neg b, a \,\mathbf{U}\, b\} \\
q_3' & = & \{a, \neg b, \neg(a \,\mathbf{U}\, b)\} \\
q_4 & = & \{a, b, a \,\mathbf{U}\, b\}
\end{array}
$$

- Accepting states are $\{q_i : \neg(a \,\mathbf{U}\, b) \in q_i \text{ or } b \in q_i\}$.

$\psi = a \,\mathbf{U}\, b$ and A_ψ



- For $a \,\mathbf{U}\, b$, the accepting states are $\{q_1, q_3, q_4, q_5, q_6\}$.
- For $\neg a \,\mathbf{U}\, b$, the accepting states are $\{q_1, q_2, q_3, q_5, q_6\}$.

$\psi = (a \,\mathbf{U}\, b) \vee (\neg a \,\mathbf{U}\, b)$ and A_ψ

LTL Model Checking with Fairness Constraints

- Our CTL model checking algorithm is slightly modified to check CTL properties with fairness constraints.
- However, it is not necessary for LTL model checking.
- Consider, for example, checking an LTL formula ϕ with a justice constraint ψ.
  - Recall that a justice constraint ψ considers only paths that ψ occurs infinitely often.
- We would like to check if ϕ holds on all fair paths.
- This is equivalent to checking $\mathbf{GF}\psi \implies \phi$.
  - because LTL can specify justice constraints.
- Hence the LTL model checking algorithm works even if there are fairness constraints.

NuSMV LTL Model Checking Algorithm

- We can in fact implement the LTL model checking algorithm by the CTL model checking algorithm with justice constraints.
- Let $\mathcal{M}$ be a NuSMV model and ϕ an LTL formula.
- Here is how it works:
  - Construct $A_{\neg\phi}$ as a NuSMV model.
  - Construct $\mathcal{M} \times A_{\neg\phi}$.
  - Check $\mathbf{EG}\top$ with justice constraint $\neg(\chi \mathbf{U} \psi) \vee \psi$ for each $\chi \mathbf{U} \psi$ in ϕ.
- The NuSMV LTL model checking algorithm uses justice constraints to search paths accepted by $\mathcal{M} \times A_{\neg\phi}$.

Outline

The CTL Model Checking Algorithm Revisited I

- We have presented a CTL model checking algorithm.
- Let $\mathcal{M} = (S, \rightarrow, L)$ be a transition system.
- The algorithm computes the set $[\![\phi]\!]$ for any CTL formula ϕ, where

$$[\![\phi]\!] = \{s \in S : \mathcal{M}, s \vDash \phi\}.$$

- We use the following equations:

$$\begin{array}{rcl}
[\![p]\!] & = & \{s \in S : p \in L(s)\} \\
[\![\bot]\!] & = & \varnothing \\
[\![\neg\phi]\!] & = & S \setminus [\![\phi]\!] \\
[\![\phi \wedge \psi]\!] & = & [\![\phi]\!] \cap [\![\psi]\!] \\
[\![\mathbf{EX}\phi]\!] & = & \mathsf{pre}_{\exists}(\mathcal{M}, [\![\phi]\!]) \\
[\![\mathbf{AF}\phi]\!] & = & [\![\phi]\!] \cup \mathsf{pre}_{\forall}(\mathcal{M}, [\![\mathbf{AF}\phi]\!]) \\
[\![\mathbf{E}[\phi \,\mathbf{U}\, \psi]]\!] & = & [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathsf{pre}_{\exists}(\mathcal{M}, [\![\mathbf{E}[\phi \,\mathbf{U}\, \psi]]\!]))
\end{array}$$

- The last two recursive equations are puzzling.
  - Circular definitions are always problematic!

- To simplify our presentation, we will use a different adequate set of temporal connectives $\{\mathbf{EX}, \mathbf{EG}, \mathbf{EU}\}$.
  - To get rid of $\mathsf{pre}_{\forall}(\bullet, \bullet)$.
- Hence we use the following puzzling equations:

$$
\begin{array}{rcl}
[\![\mathbf{EG}\phi]\!] & = & [\![\phi]\!] \cap \mathsf{pre}_{\exists}(\mathcal{M}, [\![\mathbf{EG}\phi]\!]) \\
[\![\mathbf{E}[\phi\, \mathbf{U}\, \psi]]\!] & = & [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathsf{pre}_{\exists}(\mathcal{M}, [\![\mathbf{E}[\phi\, \mathbf{U}\, \psi]]\!]))
\end{array}
$$

- We will explain them by fixed-point theory.
- The theory will also establish the correctness of the algorithm.

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL
6. Model-checking algorithms
7. The fixed-point characterisation of CTL
   - Monotone functions
   - The correctness of $\mathtt{SAT}_{\mathtt{EG}}$

Monotone Functions

Definition

Let S be a set of states and $F : 2^S \to 2^S$.

1. F is <u>monotone</u> if $X \subseteq Y$ implies $F(X) \subseteq F(Y)$ for every $X, Y \subseteq S$;
2. $X \subseteq S$ is a <u>fixed point</u> of F if $F(X) = X$.

- Examples: let $S \triangleq \{s_0, s_1\}$.
  - $F(Y) \triangleq Y \cup \{s_0\}$. $F(Y)$ is monotone. $\{s_0\}$ and $\{s_0, s_1\}$ are fixed points of $F(Y)$.
  - $G(Y) \triangleq \begin{cases} \{s_1\} & \text{if } Y = \{s_0\} \\ \{s_0\} & \text{otherwise} \end{cases}$. $G(Y)$ is not monotone. $G(Y)$ has no fixed points.
- Let $F : 2^S \to 2^S$. We write $F^i(X)$ for the expression

$$\overbrace{F(F \cdots F}^{i}(X) \cdots).$$

Fixpoint Theorem I

Theorem

Let S be a set with $|S| = n$. If $F : 2^S \to 2^S$ is a monotone function, $F^n(\varnothing)$ is the least fixed point of F and $F^n(S)$ is the greatest fixed point of F (by set inclusion order).

Proof.

Since $\varnothing \subseteq F(\varnothing)$ and F is monotone, $F(\varnothing) \subseteq F^2(\varnothing)$. More generally,

$$F(\varnothing) \subseteq F^2(\varnothing) \subseteq \cdots \subseteq F^n(\varnothing) \subseteq F^{n+1}(\varnothing).$$

Since $|S| = n$ and F is monotone, there is $1 \leq k \leq n$ that $F^k(\varnothing) = F^{k+1}(\varnothing)$. Thus $F^n(\varnothing)$ is a fixed point of F.
Suppose $H = F(H)$ is a fixed point of F. Since $\varnothing \subseteq H$, $F(\varnothing) \subseteq F(H) = H$. $F^2(\varnothing) \subseteq F(H) = H$. Hence $F^i(\varnothing) \subseteq H$ for every i. Particularly, $F^n(\varnothing) \subseteq H$. The case for the greatest fixed point is similar. □

Fixpoint Theorem II

- Knaster-Tarski theorem in fact shows the least and greatest fixed points exist for any (not necessarily finite) set.
- Our fixpoint theorem is a special case of Knaster-Tarski theorem.
- The fixpoint theorem shows how to compute least and greatest fixed points for monotone functions over finite sets.
- We will show $[\![\mathbf{EG}\phi]\!]$ and $[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!]$ are in fact greatest and least fixed points of certain monotone functions respectively.

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL
6. Model-checking algorithms
7. The fixed-point characterisation of CTL
   - Monotone functions
   - The correctness of $\mathtt{SAT}_{\mathtt{EG}}$

Computing $[\![\mathbf{EG}\phi]\!]$

- Recall that $\mathbf{EG}\phi \equiv \phi \wedge \mathbf{EXEG}\phi$.
- Hence $[\![\mathbf{EG}\phi]\!] = [\![\phi]\!] \cap \mathrm{pre}_{\exists}(\mathcal{M}, [\![\mathbf{EG}\phi]\!])$.
- $[\![\mathbf{EG}\phi]\!]$ is a fixed point of

$$F(X) \triangleq [\![\phi]\!] \cap \mathrm{pre}_{\exists}(\mathcal{M}, X).$$

- We will show that $[\![\mathbf{EG}\phi]\!]$ is in fact a greatest fixed point of F.

$[\![\mathbf{EG}\phi]\!]$ as a Greatest Fixed Point

Theorem

Let $\mathcal{M} = (S, \to, L)$ be a model with $|S| = n$ and $F(X) \triangleq [\![\phi]\!] \cap pre_{\exists}(\mathcal{M}, X)$. Then F is monotone, $[\![\mathbf{EG}\phi]\!]$ is the greatest fixed point of F, and $[\![\mathbf{EG}\phi]\!] = F^n(S)$.

Proof.

Let $X, Y \subseteq S$ and $X \subseteq Y$, and $s \in F(X)$. Then $s \in [\![\phi]\!]$ and there is an s' such that $s \to s'$ and $s' \in X \subseteq Y$. That is, $s' \in F(Y)$ and hence $F(X) \subseteq F(Y)$. F is monotone.

Since $[\![\mathbf{EG}\phi]\!] = F([\![\mathbf{EG}\phi]\!])$, $[\![\mathbf{EG}\phi]\!]$ is a fixed point of F. It remains to show that $[\![\mathbf{EG}\phi]\!]$ is the greatest fixed point of F. Consider any $X \subseteq S$ with $X = F(X)$. Let $s_0 \in X$. Then $s_0 \in F(X) = [\![\phi]\!] \cap \text{pre}_{\exists}(\mathcal{M}, X)$. $s_0 \in [\![\phi]\!]$ and there is an s_1 with $s_0 \to s_1$ such that $s_1 \in X$. By induction, we have a path $s_0 \to s_1 \to \cdots$ such that $s_i \in [\![\phi]\!]$ for $s \geq 0$. Hence $s_0 \in [\![\mathbf{EG}\phi]\!]$.

Therefore $X \subseteq [\![\mathbf{EG}\phi]\!]$ for every fixed point X of F.

The greatest fixed point of F is $F^n(S)$. □

$Y \leftarrow \mathtt{SAT}(\phi)$;
$X \leftarrow \varnothing$;
repeat
  $X \leftarrow Y$;
  $Y \leftarrow Y \cap \mathsf{pre}_{\exists}(\mathcal{M}, Y)$;
until $\underline{X = Y}$;
return (Y);

- Note that $\mathtt{SAT}_{\mathtt{EG}}(\mathcal{M}, \phi)$ does not apply the previous theorem exactly.
- Recall that

$$F(X) \stackrel{\triangle}{=} [\![\phi]\!] \cap \mathsf{pre}_{\exists}(\mathcal{M}, X).$$

$\texttt{SAT}_{\texttt{EG}}(\mathcal{M}, \phi)$ II

- By the theorem, we would have

$$\begin{array}{rcl}
\overline{Y}_0 & = & F^0(S) = S \\
\overline{Y}_1 & = & F(\overline{Y}_0) = [\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_0) \\
 & = & [\![\phi]\!] \\
\overline{Y}_2 & = & F(\overline{Y}_1) = \overline{Y}_1 \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_1) \\
\overline{Y}_3 & = & F(\overline{Y}_2) = \overline{Y}_1 \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_2)
\end{array}$$

$$\text{where } \overline{Y}_0 \supseteq \overline{Y}_1 \supseteq \overline{Y}_2 \supseteq \cdots$$

- $\texttt{SAT}_{\texttt{EG}}(\mathcal{M}, \phi)$ on the other hand computes

$$\begin{array}{rcl}
Y_0 & = & [\![\phi]\!] = \overline{Y}_1 \\
Y_1 & = & Y_0 \cap \mathsf{pre}_\exists(\mathcal{M}, Y_0) \\
 & = & \overline{Y}_1 \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_1) = \overline{Y}_2 \\
Y_2 & = & Y_1 \cap \mathsf{pre}_\exists(\mathcal{M}, Y_1) \\
 & = & \overline{Y}_1 \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_1) \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_2) \\
 & = & \overline{Y}_1 \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_2) = \overline{Y}_3 \\
Y_3 & = & Y_2 \cap \mathsf{pre}_\exists(\mathcal{M}, Y_2) \\
 & = & \overline{Y}_1 \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_2) \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_3) \\
 & = & \overline{Y}_1 \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_3) = \overline{Y}_4
\end{array}$$

Outline

1. Linear-time temporal logic
2. Model checking
3. The NuSMV model checker
4. Branching-time logic
5. CTL* and the expressive powers of LTL and CTL
6. Model-checking algorithms
7. The fixed-point characterisation of CTL
   - Monotone functions
   - The correctness of $\mathtt{SAT}_{\mathtt{EG}}$

Computing $[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!]$

- Recall that $\mathbf{E}[\phi\,\mathbf{U}\,\psi] \equiv \psi \vee (\phi \wedge \mathbf{EXE}[\phi\,\mathbf{U}\,\psi])$.
- Hence $[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!] = [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, [\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!]))$.
- $[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!]$ is a fixed point of

$$G(X) \triangleq [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, X)).$$

- We will show that $[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!]$ is in fact a least fixed point of G.

$[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!]$ as a Least Fixed Point

Theorem

Let $\mathcal{M} = (S, \rightarrow, L)$ be a model with $|S| = n$ and $G(X) \triangleq [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathit{pre}_\exists(\mathcal{M}, X))$. Then G is monotone, $[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!]$ is the least fixed point of G, and $[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!] = G^n(\varnothing)$.

Proof.

Since $\mathrm{pre}_\exists(\mathcal{M}, X)$ is monotone, G is monotone.
Since $G^n(\varnothing)$ is the least fixed point of G, it remains to show that $[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!] = G^n(\varnothing)$.

Recall $\mathcal{M}, s \vDash \mathbf{E}[\phi\,\mathbf{U}\,\psi]$ if for some path $s_0 (= s) \rightarrow s_1 \rightarrow \cdots$ there is $i \geq 0$ that $\mathcal{M}, s_i \vDash \psi$ and for every $0 \leq j < i$ we have $\mathcal{M}, s_j \vDash \phi$.

Observe that $G^1(\varnothing) = [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathrm{pre}_\exists(\mathcal{M}, \varnothing)) = [\![\psi]\!]$. $s \in G^1(\varnothing)$ iff $\mathcal{M}, s \vDash \mathbf{E}[\phi\,\mathbf{U}\,\psi]$ by taking $i = 0$. Similarly, $G^2(\varnothing) = G(G^1(\varnothing)) = [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathrm{pre}_\exists(\mathcal{M}, G^1(\varnothing)))$. That is, $s \in G^2(\varnothing)$ iff $\mathcal{M}, s \vDash \mathbf{E}[\phi\,\mathbf{U}\,\psi]$ by taking $i = 0, 1$. By induction, one can show $s \in G^k(\varnothing)$ iff $\mathcal{M}, s \vDash \mathbf{E}[\phi\,\mathbf{U}\,\psi]$ by taking $i = 0, \ldots, k-1$. Hence $[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!] = \bigcup_{i \in \mathbb{N}} G^i(\varnothing)$.

Now recall that $G^0(\varnothing) \subseteq G^1(\varnothing) \subseteq G^2(\varnothing) \subseteq \cdots$ and $G^n(\varnothing)$ is a fixed point. We have $\bigcup_{i \in \mathbb{N}} G^i(\varnothing) = G^n(\varnothing)$. That is, $[\![\mathbf{E}[\phi\,\mathbf{U}\,\psi]]\!] = G^n(\varnothing)$. □

$\mathtt{SAT}_{\mathtt{EU}}(\mathcal{M}, \phi, \psi)$ Revisited I

$W \leftarrow \mathtt{SAT}(\mathcal{M}, \phi)$;
$X \leftarrow S$;
$Y \leftarrow \mathtt{SAT}(\mathcal{M}, \psi)$;
repeat
  $X \leftarrow Y$;
  $Y \leftarrow Y \cup (W \cap \mathsf{pre}_\exists(\mathcal{M}, Y))$;
until $\underline{X = Y}$;
return Y;

- Note again that $\mathtt{SAT}_{\mathtt{EU}}(\mathcal{M}, \phi, \psi)$ does not exactly follow the theorem.
- Recall

$$G(X) \triangleq [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, X)).$$

$\mathtt{SAT}_{\mathtt{EU}}(\mathcal{M}, \phi, \psi)$ Revisited II

- By the theorem, we would have

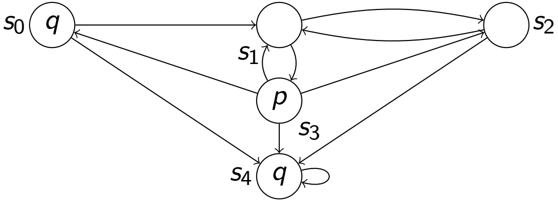
$$
\begin{array}{rcl}
\overline{Y}_0 & = & G^0(\emptyset) = \emptyset \\
\overline{Y}_1 & = & G(\overline{Y}_0) = [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_0)) = [\![\psi]\!] \\
\overline{Y}_2 & = & G(\overline{Y}_1) = [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_1)) \\
 & = & \overline{Y}_1 \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_1)) \\
\overline{Y}_3 & = & G(\overline{Y}_2) = \overline{Y}_1 \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_2)) \\
\end{array}
$$

$$\text{where } \overline{Y}_0 \subseteq \overline{Y}_1 \subseteq \overline{Y}_2 \subseteq \cdots$$

- $\mathtt{SAT}_{\mathtt{EU}}(\mathcal{M}, \phi, \psi)$ on the other hand computes

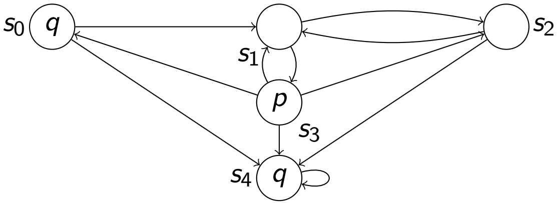
$$
\begin{array}{rcl}
Y_0 & = & [\![\psi]\!] = \overline{Y}_1 \\
Y_1 & = & Y_0 \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, Y_0)) \\
 & = & \overline{Y}_1 \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_1)) = \overline{Y}_2 \\
Y_2 & = & Y_1 \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, Y_1)) \\
 & = & \overline{Y}_1 \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_1)) \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_2)) \\
 & = & \overline{Y}_1 \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_2)) = \overline{Y}_3 \\
Y_3 & = & Y_2 \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, Y_2)) \\
 & = & \overline{Y}_1 \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_2)) \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_3)) \\
 & = & \overline{Y}_1 \cup ([\![\phi]\!] \cap \mathsf{pre}_\exists(\mathcal{M}, \overline{Y}_3)) = \overline{Y}_4
\end{array}
$$

Example I



- Compute $[\![\mathbf{EF}p]\!]$.
  - Since $[\![\mathbf{EF}p]\!] = [\![\mathbf{E}[\top\ \mathbf{U}\ p]]\!]$, consider $G(X) \triangleq [\![p]\!] \cup ([\![\top]\!] \cap \mathrm{pre}_\exists(\mathcal{M}, X)) = \{s_3\} \cup \mathrm{pre}_\exists(\mathcal{M}, X)$. $G^1(\varnothing) = \{s_3\}$, $G^2(\varnothing) = G(\{s_3\}) = \{s_3, s_1\}$, $G^3(\varnothing) = G(\{s_1, s_3\}) = \{s_3, s_0, s_2, s_1\}$, $G^4(\varnothing) = G(\{s_0, s_1, s_2, s_3\}) = \{s_3, s_0, s_2, s_1\} = G^3(\varnothing)$. Hence $[\![\mathbf{EF}p]\!] = \{s_0, s_1, s_2, s_3\}$.

Example II



- Compute $[\![\mathbf{EG}q]\!]$.
  - Consider $F(X) \triangleq [\![q]\!] \cap \text{pre}_\exists(\mathcal{M}, X) = \{s_0, s_4\} \cap \text{pre}_\exists(\mathcal{M}, X)$. $F^1(S) = \{s_0, s_4\}$. $F^2(S) = F(\{s_0, s_4\}) = \{s_0, s_4\} \cap \{s_3, s_0, s_2, s_4\} = \{s_0, s_4\} = F^1(S)$. Hence $[\![\mathbf{EG}q]\!] = \{s_0, s_4\}$.

Where to go?

- Model checking has been applied in industry.
  - IC design, Windows device drivers, NASA Curiosity, Amazon, etc.
- We only discuss the most elementary backgrounds.
- Lots of techniques are needed to make it practical.
  - implicit algorithms;
  - counterexample guided abstraction refinement;
  - partial order reduction;
  - compositional reasoning;
  - and many more.
- There are also model checking algorithms for infinite state systems.
- Contact me if you are interested in research opportunities.