Formal Methods
Module II: Formal Verification
Ch. 05: **Explicit-State CTL Model Checking**

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it
URL: https://disi.unitn.it/rseba/DIDATTICA/fm2024/
Teaching assistant: Giuseppe Spallitta – giuseppe.spallitta@unitn.it

M.S. in Computer Science, Mathematics, & Artificial Intelligence Systems
Academic year 2023-2024

last update: Thursday 18th April, 2024, 08:32

Outline

Outline

CTL Model Checking

CTL Model Checking is a formal verification technique where...

- ...the system is represented as a Finite State Machine M:
- ...the property is expressed a CTL formula φ:

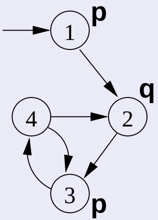
$$\mathbf{AG}(p \rightarrow \mathbf{AF}q)$$

- ...the model checking algorithm checks whether in all initial states of M all the executions of the model satisfy the formula ($M \models \varphi$).

CTL Model Checking

CTL Model Checking is a formal verification technique where...

- ...the system is represented as a Finite State Machine M:



- ...the property is expressed a CTL formula φ:

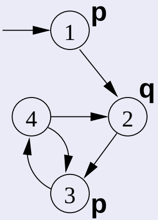
$$\mathbf{AG}(p \rightarrow \mathbf{AF}q)$$

- ...the model checking algorithm checks whether in all initial states of M all the executions of the model satisfy the formula ($M \models \varphi$).

CTL Model Checking

CTL Model Checking is a formal verification technique where...

- ...the system is represented as a Finite State Machine M:



- ...the property is expressed a CTL formula φ:

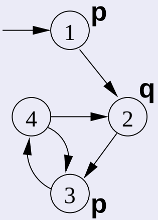
$$\mathbf{AG}(p \rightarrow \mathbf{AF}q)$$

- ...the model checking algorithm checks whether in all initial states of M all the executions of the model satisfy the formula ($M \models \varphi$).

CTL Model Checking

CTL Model Checking is a formal verification technique where...

- ...the system is represented as a Finite State Machine *M*:



- ...the property is expressed a CTL formula φ:

$$\mathbf{AG}(p \rightarrow \mathbf{AF}q)$$

- ...the model checking algorithm checks whether in all initial states of M all the executions of the model satisfy the formula ($M \models \varphi$).

CTL Model Checking: General Idea

Two macro-steps:

1. construct the set of states where the formula holds:
   $[\varphi] := \{s \in S : M, s \models \varphi\}$
   ($[\varphi]$ is called the denotation of φ)
2. then compare with the set of initial states:
   $I \subseteq [\varphi]$?

The lion's share of the effort in this process is on step 1: compute $[\varphi]$.

CTL Model Checking: General Idea

Two macro-steps:

1. construct the set of states where the formula holds:
   $[\varphi] := \{s \in S : M, s \models \varphi\}$
   ($[\varphi]$ is called the denotation of φ)
2. then compare with the set of initial states:
   $I \subseteq [\varphi]$?

The lion's share of the effort in this process is on step 1: compute $[\varphi]$.

CTL Model Checking: General Idea

Two macro-steps:

1. construct the set of states where the formula holds:
   $[\varphi] := \{s \in S : M, s \models \varphi\}$
   ($[\varphi]$ is called the denotation of φ)
2. then compare with the set of initial states:
   $I \subseteq [\varphi]$?

The lion's share of the effort in this process is on step 1: compute $[\varphi]$.

CTL Model Checking: General Idea

Two macro-steps:

1. construct the set of states where the formula holds:
   $[\varphi] := \{s \in S : M, s \models \varphi\}$
   ($[\varphi]$ is called the denotation of φ)
2. then compare with the set of initial states:
   $I \subseteq [\varphi]$?

The lion's share of the effort in this process is on step 1: compute $[\varphi]$.

CTL Model Checking: General Idea [cont.]

In order to compute $[\varphi]$:

- proceed "bottom-up" on the structure of the formula, computing $[\varphi_i]$ for each subformula φ_i of $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$:
  - $[q]$,
  - $[\mathbf{AF}q]$,
  - $[p]$,
  - $[p \rightarrow \mathbf{AF}q]$,
  - $[\mathbf{AG}(p \rightarrow \mathbf{AF}q)]$

CTL Model Checking: General Idea [cont.]

In order to compute $[\varphi]$:

- proceed "bottom-up" on the structure of the formula, computing $[\varphi_i]$ for each subformula φ_i of $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$:
  - $[q]$,
  - $[\mathbf{AF}q]$,
  - $[p]$,
  - $[p \rightarrow \mathbf{AF}q]$,
  - $[\mathbf{AG}(p \rightarrow \mathbf{AF}q)]$

CTL Model Checking: General Idea [cont.]

In order to compute $[\varphi]$:

- proceed "bottom-up" on the structure of the formula, computing $[\varphi_i]$ for each subformula φ_i of $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$:
  - $[q]$,
  - $[\mathbf{AF}q]$,
  - $[p]$,
  - $[p \rightarrow \mathbf{AF}q]$,
  - $[\mathbf{AG}(p \rightarrow \mathbf{AF}q)]$

CTL Model Checking: General Idea [cont.]

In order to compute $[\varphi]$:

- proceed "bottom-up" on the structure of the formula, computing $[\varphi_i]$ for each subformula φ_i of $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$:
  - $[q]$,
  - $[\mathbf{AF}q]$,
  - $[p]$,
  - $[p \rightarrow \mathbf{AF}q]$,
  - $[\mathbf{AG}(p \rightarrow \mathbf{AF}q)]$

CTL Model Checking: General Idea [cont.]

In order to compute $[\varphi]$:

- proceed "bottom-up" on the structure of the formula, computing $[\varphi_i]$ for each subformula φ_i of $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$:
  - $[q]$,
  - $[\mathbf{AF}q]$,
  - $[p]$,
  - $[p \rightarrow \mathbf{AF}q]$,
  - $[\mathbf{AG}(p \rightarrow \mathbf{AF}q)]$

CTL Model Checking: General Idea [cont.]

In order to compute $[\varphi]$:

- proceed "bottom-up" on the structure of the formula, computing $[\varphi_i]$ for each subformula φ_i of $\mathbf{AG}(p \to \mathbf{AF}q)$:
  - $[q]$,
  - $[\mathbf{AF}q]$,
  - $[p]$,
  - $[p \to \mathbf{AF}q]$,
  - $[\mathbf{AG}(p \to \mathbf{AF}q)]$

CTL Model Checking: General Idea [cont.]

In order to compute $[\varphi]$:

- proceed "bottom-up" on the structure of the formula, computing $[\varphi_i]$ for each subformula φ_i of $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$:
  - $[q]$,
  - $[\mathbf{AF}q]$,
  - $[p]$,
  - $[p \rightarrow \mathbf{AF}q]$,
  - $[\mathbf{AG}(p \rightarrow \mathbf{AF}q)]$

CTL Model Checking: General Idea [cont.]

In order to compute each $[\varphi_i]$:

- assign Propositional atoms by labeling function
- handle Boolean operators by standard set operations
- handle temporal operators **AX**, **EX** by computing pre-images
- handle temporal operators **AG**, **EG**, **AF**, **EF**, **AU**, **EU**, by (implicitly) applying tableaux rules, until a fixpoint is reached

CTL Model Checking: General Idea [cont.]

In order to compute each $[\varphi_i]$:

- assign Propositional atoms by labeling function
- handle Boolean operators by standard set operations
- handle temporal operators **AX**, **EX** by computing pre-images
- handle temporal operators **AG**, **EG**, **AF**, **EF**, **AU**, **EU**, by (implicitly) applying tableaux rules, until a fixpoint is reached

CTL Model Checking: General Idea [cont.]

In order to compute each $[\varphi_i]$:

- assign Propositional atoms by labeling function
- handle Boolean operators by standard set operations
- handle temporal operators **AX**, **EX** by computing pre-images
- handle temporal operators **AG**, **EG**, **AF**, **EF**, **AU**, **EU**, by (implicitly) applying tableaux rules, until a fixpoint is reached

CTL Model Checking: General Idea [cont.]

In order to compute each $[\varphi_i]$:

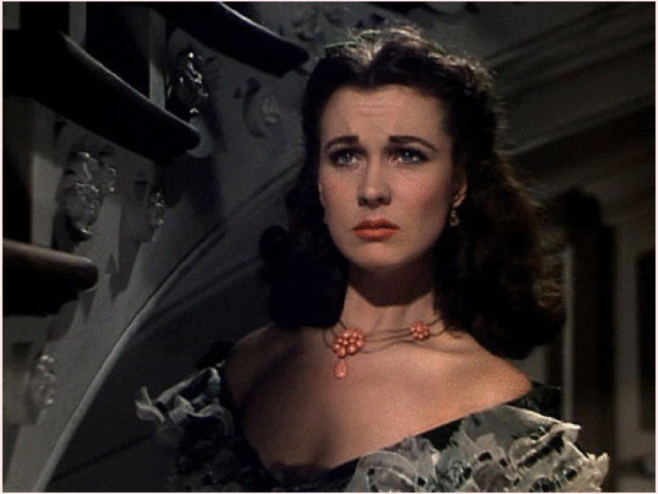
- assign Propositional atoms by labeling function
- handle Boolean operators by standard set operations
- handle temporal operators **AX**, **EX** by computing pre-images
- handle temporal operators **AG**, **EG**, **AF**, **EF**, **AU**, **EU**, by (implicitly) applying tableaux rules, until a fixpoint is reached

CTL Model Checking: General Idea [cont.]

In order to compute each $[\varphi_i]$:

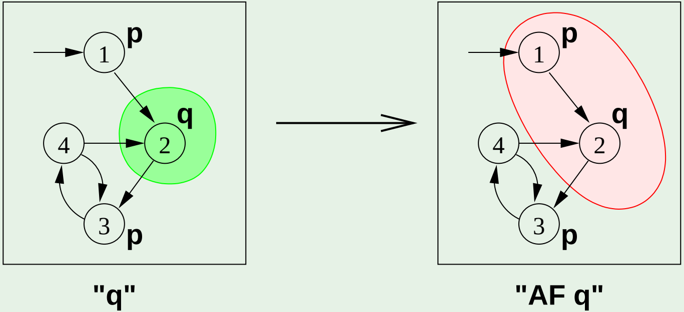
- assign Propositional atoms by labeling function
- handle Boolean operators by standard set operations
- handle temporal operators **AX**, **EX** by computing pre-images
- handle temporal operators **AG**, **EG**, **AF**, **EF**, **AU**, **EU**, by (implicitly) applying tableaux rules, until a fixpoint is reached

Tableaux Rules: a Quote



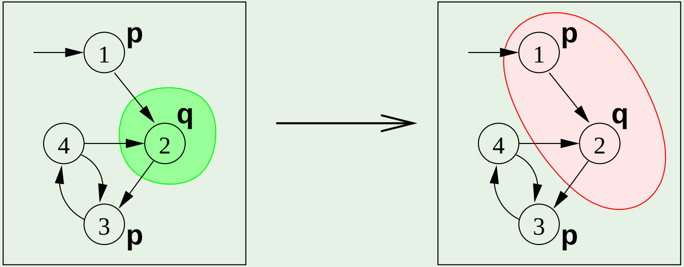
“After all... tomorrow is another day."
[Scarlett O’Hara, “Gone with the Wind”]

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$



- Recall the **AF** tableau rule: $\mathbf{AF}q \leftrightarrow (q \vee \mathbf{AXAF}q)$
- Iteration: $[\mathbf{AF}q]^{(1)} = [q]$; $\quad [\mathbf{AF}q]^{(i+1)} = [q] \cup \mathbf{AX}[\mathbf{AF}q]^{(i)}$
  - $[\mathbf{AF}q]^{(1)} = [q] = \{2\}$
  - $[\mathbf{AF}q]^{(2)} = [q \vee \mathbf{AX}q] = \{2\} \cup \{1\} = \{1,2\}$
  - $[\mathbf{AF}q]^{(3)} = [q \vee \mathbf{AX}(q \vee \mathbf{AX}q)] = \{2\} \cup \{1\} = \{1,2\}$

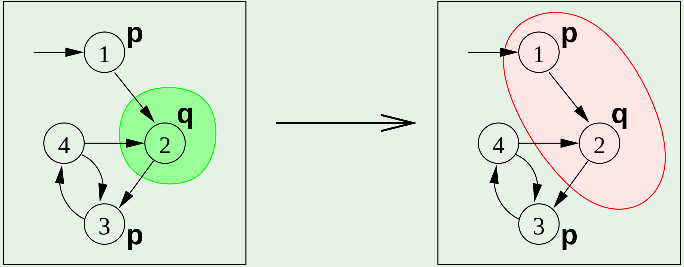
CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$



"q" "AF q"

- Recall the **AF** tableau rule: $\mathbf{AF}q \leftrightarrow (q \vee \mathbf{AXAF}q)$
- Iteration: $[\mathbf{AF}q]^{(1)} = [q]$; $[\mathbf{AF}q]^{(i+1)} = [q] \cup \mathbf{AX}[\mathbf{AF}q]^{(i)}$
  - $[\mathbf{AF}q]^{(1)} = [q] = \{2\}$
  - $[\mathbf{AF}q]^{(2)} = [q \vee \mathbf{AX}q] = \{2\} \cup \{1\} = \{1, 2\}$
  - $[\mathbf{AF}q]^{(3)} = [q \vee \mathbf{AX}(q \vee \mathbf{AX}q)] = \{2\} \cup \{1\} = \{1, 2\}$

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$

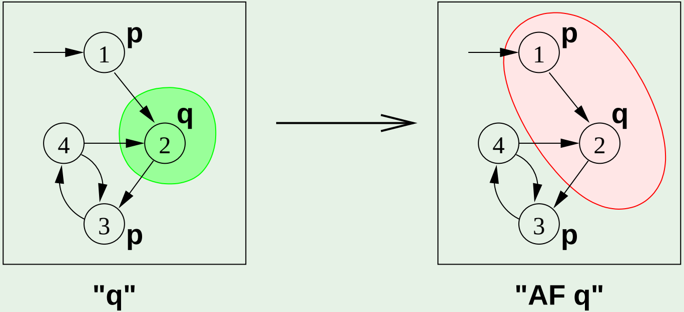


"q" "AF q"

- Recall the **AF** tableau rule: $\mathbf{AF}q \leftrightarrow (q \vee \mathbf{AXAF}q)$
- Iteration: $[\mathbf{AF}q]^{(1)} = [q]$; $\quad [\mathbf{AF}q]^{(i+1)} = [q] \cup \mathbf{AX}[\mathbf{AF}q]^{(i)}$
  - $[\mathbf{AF}q]^{(1)} = [q] = \{2\}$
  - $[\mathbf{AF}q]^{(2)} = [q \vee \mathbf{AX}q] = \{2\} \cup \{1\} = \{1, 2\}$
  - $[\mathbf{AF}q]^{(3)} = [q \vee \mathbf{AX}(q \vee \mathbf{AX}q)] = \{2\} \cup \{1\} = \{1, 2\}$

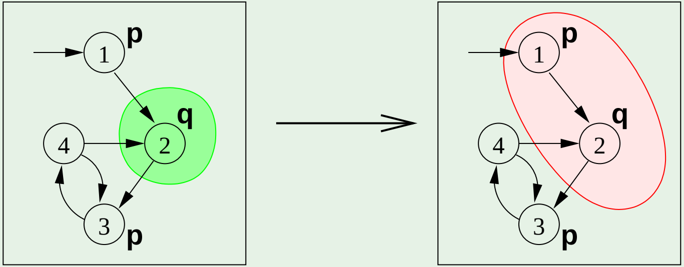
    $\Longrightarrow$ (fix point reached)

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$



- Recall the **AF** tableau rule: $\mathbf{AF}q \leftrightarrow (q \vee \mathbf{AXAF}q)$
- Iteration: $[\mathbf{AF}q]^{(1)} = [q]$; $\quad [\mathbf{AF}q]^{(i+1)} = [q] \cup \mathbf{AX}[\mathbf{AF}q]^{(i)}$
  - $[\mathbf{AF}q]^{(1)} = [q] = \{2\}$
  - $[\mathbf{AF}q]^{(2)} = [q \vee \mathbf{AX}q] = \{2\} \cup \{1\} = \{1, 2\}$
  - $[\mathbf{AF}q]^{(3)} = [q \vee \mathbf{AX}(q \vee \mathbf{AX}q)] = \{2\} \cup \{1\} = \{1, 2\}$

    $\Longrightarrow$ (fix point reached)

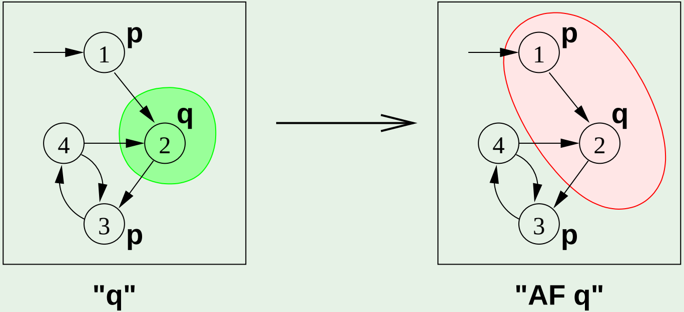


"q" **"AF q"**

- Recall the **AF** tableau rule: $\mathbf{AF}q \leftrightarrow (q \vee \mathbf{AXAF}q)$
- Iteration: $[\mathbf{AF}q]^{(1)} = [q]; \quad [\mathbf{AF}q]^{(i+1)} = [q] \cup \mathbf{AX}[\mathbf{AF}q]^{(i)}$
  - $[\mathbf{AF}q]^{(1)} = [q] = \{2\}$
  - $[\mathbf{AF}q]^{(2)} = [q \vee \mathbf{AX}q] = \{2\} \cup \{1\} = \{1, 2\}$
  - $[\mathbf{AF}q]^{(3)} = [q \vee \mathbf{AX}(q \vee \mathbf{AX}q)] = \{2\} \cup \{1\} = \{1, 2\}$

    $\Longrightarrow$ (fix point reached)

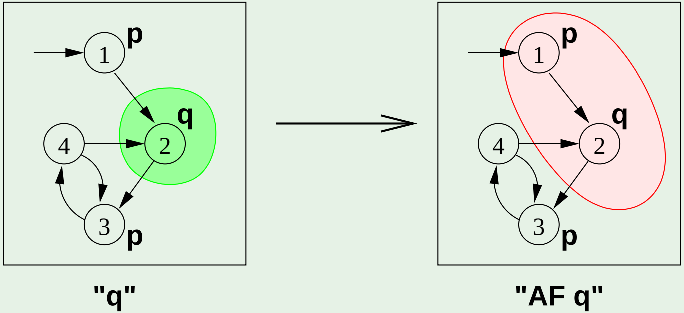
CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$



- Recall the **AF** tableau rule: $\mathbf{AF}q \leftrightarrow (q \vee \mathbf{AXAF}q)$
- Iteration: $[\mathbf{AF}q]^{(1)} = [q]; \quad [\mathbf{AF}q]^{(i+1)} = [q] \cup \mathbf{AX}[\mathbf{AF}q]^{(i)}$
  - $[\mathbf{AF}q]^{(1)} = [q] = \{2\}$
  - $[\mathbf{AF}q]^{(2)} = [q \vee \mathbf{AX}q] = \{2\} \cup \{1\} = \{1,2\}$
  - $[\mathbf{AF}q]^{(3)} = [q \vee \mathbf{AX}(q \vee \mathbf{AX}q)] = \{2\} \cup \{1\} = \{1,2\}$

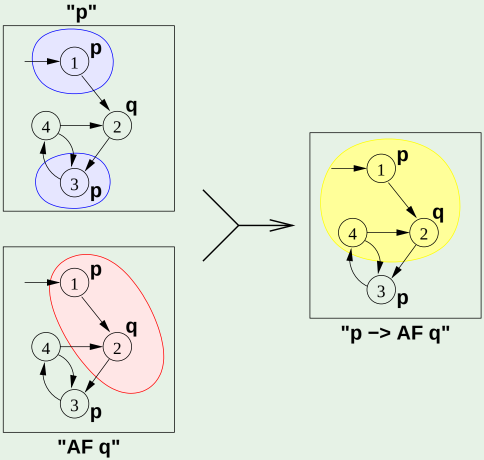
    $\Longrightarrow$ (fix point reached)

CTL Model Checking: Example: $\textbf{AG}(p \rightarrow \textbf{AF}q)$

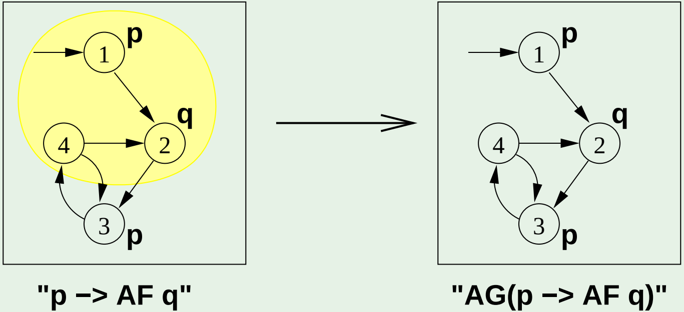


- Recall the **AF** tableau rule: $\textbf{AF}q \leftrightarrow (q \vee \textbf{AXAF}q)$
- Iteration: $[\textbf{AF}q]^{(1)} = [q]$; $\quad [\textbf{AF}q]^{(i+1)} = [q] \cup \textbf{AX}[\textbf{AF}q]^{(i)}$
  - $[\textbf{AF}q]^{(1)} = [q] = \{2\}$
  - $[\textbf{AF}q]^{(2)} = [q \vee \textbf{AX}q] = \{2\} \cup \{1\} = \{1, 2\}$
  - $[\textbf{AF}q]^{(3)} = [q \vee \textbf{AX}(q \vee \textbf{AX}q)] = \{2\} \cup \{1\} = \{1, 2\}$
    $\Longrightarrow$ (fix point reached)

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]

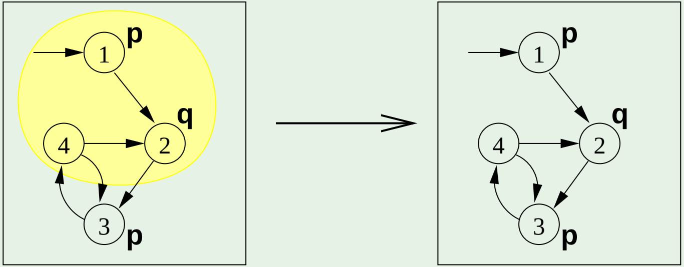


CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]



- Recall the AG tableau rule: $\mathbf{AG}\varphi \leftrightarrow (\varphi \wedge \mathbf{AXAG}\varphi)$
- Iteration: $[\mathbf{AG}\varphi^{(1)}] = [\varphi]$; $[\mathbf{AG}\varphi]^{(i+1)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(i)}$
  1. $[\mathbf{AG}\varphi]^{(1)} = [\varphi] = \{1, 2, 4\}$
  2. $[\mathbf{AG}\varphi]^{(2)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(1)} = \{1, 2, 4\} \cap \{1, 3\} = \{1\}$
  3. $[\mathbf{AG}\varphi]^{(3)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(2)} = \{1, 2, 4\} \cap \{\} = \{\}$

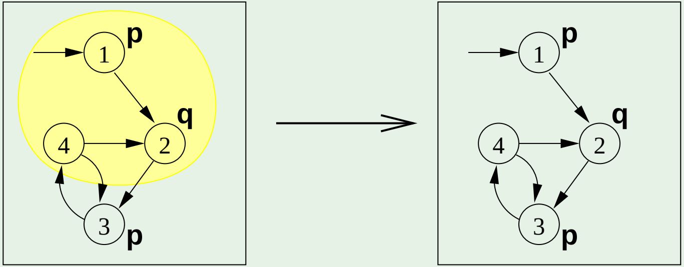
CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]



"p –> AF q"   "AG(p –> AF q)"

- Recall the **AG** tableau rule: $\mathbf{AG}\varphi \leftrightarrow (\varphi \wedge \mathbf{AXAG}\varphi)$
- Iteration: $[\mathbf{AG}\varphi^{(1)}] = [\varphi]; \quad [\mathbf{AG}\varphi]^{(i+1)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(i)}$
  1. $[\mathbf{AG}\varphi]^{(1)} = [\varphi] = \{1,2,4\}$
  2. $[\mathbf{AG}\varphi]^{(2)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(1)} = \{1,2,4\} \cap \{1,3\} = \{1\}$
  3. $[\mathbf{AG}\varphi]^{(3)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(2)} = \{1,2,4\} \cap \{\} = \{\}$

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]

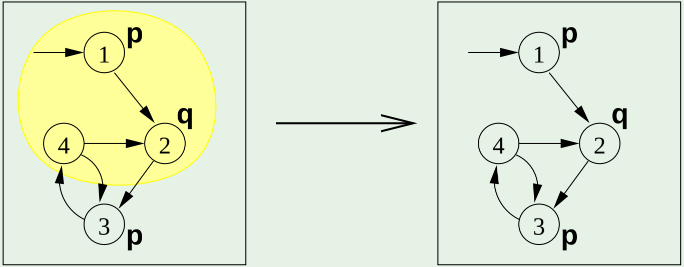


"p -> AF q" "AG(p -> AF q)"

- Recall the **AG** tableau rule: $\mathbf{AG}\varphi \leftrightarrow (\varphi \wedge \mathbf{AXAG}\varphi)$
- Iteration: $[\mathbf{AG}\varphi^{(1)}] = [\varphi]$; $[\mathbf{AG}\varphi]^{(i+1)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(i)}$
  1. $[\mathbf{AG}\varphi]^{(1)} = [\varphi] = \{1, 2, 4\}$
  2. $[\mathbf{AG}\varphi]^{(2)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(1)} = \{1, 2, 4\} \cap \{1, 3\} = \{1\}$
  3. $[\mathbf{AG}\varphi]^{(3)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(2)} = \{1, 2, 4\} \cap \{\} = \{\}$

  $\Longrightarrow$ (fix point reached)

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]

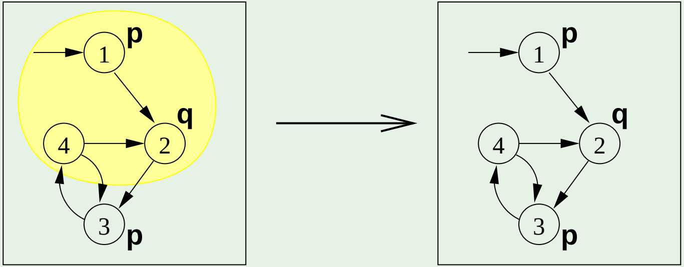


"p -> AF q" "AG(p -> AF q)"

- Recall the **AG** tableau rule: $\mathbf{AG}\varphi \leftrightarrow (\varphi \wedge \mathbf{AXAG}\varphi)$
- Iteration: $[\mathbf{AG}\varphi^{(1)}] = [\varphi]$; $[\mathbf{AG}\varphi]^{(i+1)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(i)}$
  1. $[\mathbf{AG}\varphi]^{(1)} = [\varphi] = \{1, 2, 4\}$
  2. $[\mathbf{AG}\varphi]^{(2)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(1)} = \{1, 2, 4\} \cap \{1, 3\} = \{1\}$
  3. $[\mathbf{AG}\varphi]^{(3)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(2)} = \{1, 2, 4\} \cap \{\} = \{\}$

  $\Longrightarrow$ (fix point reached)

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]

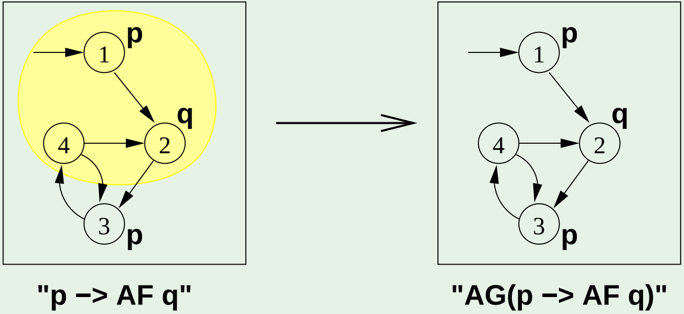


"p -> AF q" "AG(p -> AF q)"

- Recall the **AG** tableau rule: $\mathbf{AG}\varphi \leftrightarrow (\varphi \wedge \mathbf{AXAG}\varphi)$
- Iteration: $[\mathbf{AG}\varphi^{(1)}] = [\varphi]$; $[\mathbf{AG}\varphi]^{(i+1)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(i)}$
  1. $[\mathbf{AG}\varphi]^{(1)} = [\varphi] = \{1, 2, 4\}$
  2. $[\mathbf{AG}\varphi]^{(2)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(1)} = \{1, 2, 4\} \cap \{1, 3\} = \{1\}$
  3. $[\mathbf{AG}\varphi]^{(3)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(2)} = \{1, 2, 4\} \cap \{\} = \{\}$

  $\Longrightarrow$ (fix point reached)

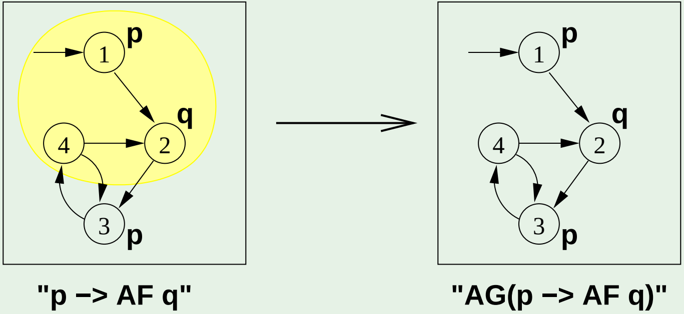
CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]



- Recall the **AG** tableau rule: $\mathbf{AG}\varphi \leftrightarrow (\varphi \wedge \mathbf{AXAG}\varphi)$
- Iteration: $[\mathbf{AG}\varphi^{(1)}] = [\varphi]$; $[\mathbf{AG}\varphi]^{(i+1)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(i)}$
  1. $[\mathbf{AG}\varphi]^{(1)} = [\varphi] = \{1, 2, 4\}$
  2. $[\mathbf{AG}\varphi]^{(2)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(1)} = \{1, 2, 4\} \cap \{1, 3\} = \{1\}$
  3. $[\mathbf{AG}\varphi]^{(3)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(2)} = \{1, 2, 4\} \cap \{\} = \{\}$

  $\Longrightarrow$ (fix point reached)

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]



- Recall the **AG** tableau rule: $\mathbf{AG}\varphi \leftrightarrow (\varphi \wedge \mathbf{AXAG}\varphi)$
- Iteration: $[\mathbf{AG}\varphi^{(1)}] = [\varphi]$; $[\mathbf{AG}\varphi]^{(i+1)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(i)}$
  1. $[\mathbf{AG}\varphi]^{(1)} = [\varphi] = \{1, 2, 4\}$
  2. $[\mathbf{AG}\varphi]^{(2)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(1)} = \{1, 2, 4\} \cap \{1, 3\} = \{1\}$
  3. $[\mathbf{AG}\varphi]^{(3)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(2)} = \{1, 2, 4\} \cap \{\} = \{\}$

  $\Longrightarrow$ (fix point reached)

CTL Model Checking: Example: $\mathbf{AG}(p \to \mathbf{AF}q)$ [cont.]

- The set of states where the formula holds is empty
  $\Longrightarrow$ the initial state does not satisfy the property
  $\Longrightarrow M \not\models \mathbf{AG}(p \to \mathbf{AF}q)$
- Counterexample: a lazo-shaped path: $1, 2, \{3, 4\}^\omega$ (satisfying $\mathbf{EF}(p \wedge \mathbf{EG}\neg q)$)

Note

Counter-example reconstruction in general is not trivial, based on intermediate sets.

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]

- The set of states where the formula holds is empty
  $\Longrightarrow$ the initial state does not satisfy the property
  $\Longrightarrow M \not\models \mathbf{AG}(p \rightarrow \mathbf{AF}q)$
- Counterexample: a lazo-shaped path: $1, 2, \{3, 4\}^{\omega}$ (satisfying $\mathbf{EF}(p \wedge \mathbf{EG}\neg q)$)

Note

Counter-example reconstruction in general is not trivial, based on intermediate sets.

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]

- The set of states where the formula holds is empty
  $\Longrightarrow$ the initial state does not satisfy the property
  $\Longrightarrow M \not\models \mathbf{AG}(p \rightarrow \mathbf{AF}q)$
- Counterexample: a lazo-shaped path: $1, 2, \{3, 4\}^\omega$ (satisfying $\mathbf{EF}(p \wedge \mathbf{EG}\neg q)$)

Note

Counter-example reconstruction in general is not trivial, based on intermediate sets.

CTL Model Checking: Example: $\mathbf{AG}(p \to \mathbf{AF}q)$ [cont.]

- The set of states where the formula holds is empty
  $\Longrightarrow$ the initial state does not satisfy the property
  $\Longrightarrow M \not\models \mathbf{AG}(p \to \mathbf{AF}q)$
- Counterexample: a lazo-shaped path: $1, 2, \{3, 4\}^\omega$ (satisfying $\mathbf{EF}(p \land \mathbf{EG}\neg q)$)

Note

Counter-example reconstruction in general is not trivial, based on intermediate sets.

CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]

- The set of states where the formula holds is empty
  $\Longrightarrow$ the initial state does not satisfy the property
  $\Longrightarrow M \not\models \mathbf{AG}(p \rightarrow \mathbf{AF}q)$
- Counterexample: a lazo-shaped path: $1, 2, \{3, 4\}^{\omega}$ (satisfying $\mathbf{EF}(p \wedge \mathbf{EG}\neg q)$)

Note

Counter-example reconstruction in general is not trivial, based on intermediate sets.

Outline

The fixed-point theory of lattice of sets

Definition

Let 2^S denote the power set of S, i.e., the set of all subsets of S.

- For any finite set S, the structure $\langle 2^S, \subseteq \rangle$ forms a complete lattice with $\cup$ as join and $\cap$ as meet operations.
- A function $F : 2^S \longmapsto 2^S$ is monotonic provided $S_1 \subseteq S_2 \Rightarrow F(S_1) \subseteq F(S_2)$.

The fixed-point theory of lattice of sets

Definition

Let 2^S denote the power set of S, i.e., the set of all subsets of S.

- For any finite set S, the structure $\langle 2^S, \subseteq \rangle$ forms a complete lattice with $\cup$ as join and $\cap$ as meet operations.
- A function $F : 2^S \longmapsto 2^S$ is monotonic provided $S_1 \subseteq S_2 \;\Rightarrow\; F(S_1) \subseteq F(S_2)$.

Fixed Points

Definition

Let $\langle 2^S, \subseteq \rangle$ be a complete lattice, S finite.

- Given a function $F : 2^S \longmapsto 2^S$, $a \subseteq S$ is a fixed point of F iff

$$F(a) = a$$

- a is a least fixed point (LFP) of F, written $\mu x.F(x)$, iff, for every other fixed point a' of F, $a \subseteq a'$
- a is a greatest fixed point (GFP) of F, written $\nu x.F(x)$, iff, for every other fixed point a' of F, $a' \subseteq a$

Fixed Points

Definition

Let $\langle 2^S, \subseteq \rangle$ be a complete lattice, S finite.

- Given a function $F : 2^S \longmapsto 2^S$, $a \subseteq S$ is a fixed point of F iff

$$F(a) = a$$

- a is a least fixed point (LFP) of F, written $\mu x.F(x)$, iff, for every other fixed point a' of F, $a \subseteq a'$
- a is a greatest fixed point (GFP) of F, written $\nu x.F(x)$, iff, for every other fixed point a' of F, $a' \subseteq a$

Fixed Points

Definition

Let $\langle 2^S, \subseteq \rangle$ be a complete lattice, S finite.

- Given a function $F : 2^S \longmapsto 2^S$, $a \subseteq S$ is a fixed point of F iff

$$F(a) = a$$

- a is a least fixed point (LFP) of F, written $\mu x.F(x)$, iff, for every other fixed point a' of F, $a \subseteq a'$
- a is a greatest fixed point (GFP) of F, written $\nu x.F(x)$, iff, for every other fixed point a' of F, $a' \subseteq a$

Fixed Points

Definition

Let $\langle 2^S, \subseteq \rangle$ be a complete lattice, S finite.

- Given a function $F : 2^S \longmapsto 2^S$, $a \subseteq S$ is a fixed point of F iff

$$F(a) = a$$

- a is a least fixed point (LFP) of F, written $\mu x.F(x)$, iff, for every other fixed point a' of F, $a \subseteq a'$
- a is a greatest fixed point (GFP) of F, written $\nu x.F(x)$, iff, for every other fixed point a' of F, $a' \subseteq a$

Iteratively computing fixed points

Tarski's Theorem

A monotonic function over a complete finite lattice has a least and a greatest fixed point.

(A corollary of) Kleene's Theorem

A monotonic function F over a complete finite lattice has a least and a greatest fixed point, which can be computed as follows:

- the least fixed point of F is the limit of the chain $\emptyset \subseteq F(\emptyset) \subseteq F(F(\emptyset)) \ldots$,
- the greatest fixed point of F is the limit of chain $S \supseteq F(S) \supseteq F(F(S)) \ldots$

Since 2^S is finite, convergence is obtained in a finite number of steps.

Iteratively computing fixed points

Tarski's Theorem

A monotonic function over a complete finite lattice has a least and a greatest fixed point.

(A corollary of) Kleene's Theorem

A monotonic function F over a complete finite lattice has a least and a greatest fixed point, which can be computed as follows:

- the least fixed point of F is the limit of the chain $\emptyset \subseteq F(\emptyset) \subseteq F(F(\emptyset)) \ldots$,
- the greatest fixed point of F is the limit of chain $S \supseteq F(S) \supseteq F(F(S)) \ldots$

Since 2^S is finite, convergence is obtained in a finite number of steps.

Iteratively computing fixed points

Tarski's Theorem

A monotonic function over a complete finite lattice has a least and a greatest fixed point.

(A corollary of) Kleene's Theorem

A monotonic function F over a complete finite lattice has a least and a greatest fixed point, which can be computed as follows:

- the least fixed point of F is the limit of the chain $\emptyset \subseteq F(\emptyset) \subseteq F(F(\emptyset)) \ldots$,
- the greatest fixed point of F is the limit of chain $S \supseteq F(S) \supseteq F(F(S)) \ldots$

Since 2^S is finite, convergence is obtained in a finite number of steps.

Iteratively computing fixed points

Tarski's Theorem

A monotonic function over a complete finite lattice has a least and a greatest fixed point.

(A corollary of) Kleene's Theorem

A monotonic function F over a complete finite lattice has a least and a greatest fixed point, which can be computed as follows:

- the least fixed point of F is the limit of the chain $\emptyset \subseteq F(\emptyset) \subseteq F(F(\emptyset)) \ldots$,
- the greatest fixed point of F is the limit of chain $S \supseteq F(S) \supseteq F(F(S)) \ldots$

Since 2^S is finite, convergence is obtained in a finite number of steps.

Iteratively computing fixed points

Tarski's Theorem

A monotonic function over a complete finite lattice has a least and a greatest fixed point.

(A corollary of) Kleene's Theorem

A monotonic function F over a complete finite lattice has a least and a greatest fixed point, which can be computed as follows:

- the least fixed point of F is the limit of the chain $\emptyset \subseteq F(\emptyset) \subseteq F(F(\emptyset)) \ldots$,
- the greatest fixed point of F is the limit of chain $S \supseteq F(S) \supseteq F(F(S)) \ldots$

Since 2^S is finite, convergence is obtained in a finite number of steps.

CTL Model Checking and Lattices

- If $M = \langle S, I, R, L, AP \rangle$ is a Kripke structure, then $\langle 2^S, \subseteq \rangle$ is a complete lattice
- We identify φ with its denotation $[\varphi]$

$\Longrightarrow$ we can see logical operators as functions $F : 2^S \longmapsto 2^S$ on the complete lattice $\langle 2^S, \subseteq \rangle$

CTL Model Checking and Lattices

- If $M = \langle S, I, R, L, AP\rangle$ is a Kripke structure, then $\langle 2^S, \subseteq\rangle$ is a complete lattice
- We identify φ with its denotation $[\varphi]$

$\Longrightarrow$ we can see logical operators as functions $F : 2^S \longmapsto 2^S$ on the complete lattice $\langle 2^S, \subseteq\rangle$

CTL Model Checking and Lattices

- If $M = \langle S, I, R, L, AP \rangle$ is a Kripke structure, then $\langle 2^S, \subseteq \rangle$ is a complete lattice
- We identify φ with its denotation $[\varphi]$

$\Longrightarrow$ we can see logical operators as functions $F : 2^S \longmapsto 2^S$ on the complete lattice $\langle 2^S, \subseteq \rangle$

Denotation of a CTL formula φ: $[\varphi]$

Definition of $[\varphi]$

$[\varphi] := \{s \in S : M, s \models \varphi\}$

Recursive definition of $[\varphi]$

$$
\begin{array}{lcl}
[\top] & = & S \\
[\bot] & = & \{\} \\
[p] & = & \{s | p \in L(s)\} \\
[\neg\varphi_1] & = & S/[\varphi_1] \\
[\varphi_1 \wedge \varphi_2] & = & [\varphi_1] \cap [\varphi_2] \\
[\mathbf{EX}\varphi] & = & \{s \mid \exists s' \in [\varphi] \; s.t. \; \langle s, s' \rangle \in R\} \\
[\mathbf{EG}\beta] & = & \nu Z.(\; [\beta] \cap [\mathbf{EX}Z] \;) \\
[\mathbf{E}(\beta_1 \mathbf{U} \beta_2)] & = & \mu Z.(\; [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z]) \;)
\end{array}
$$

Denotation of a CTL formula φ: $[\varphi]$

Definition of $[\varphi]$

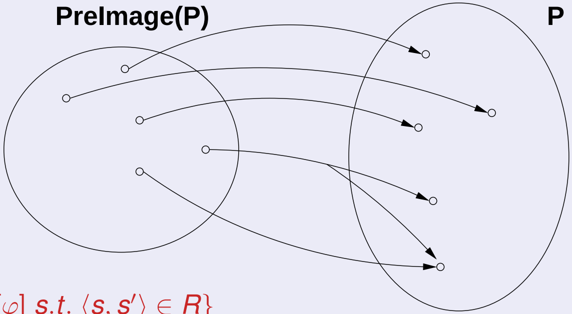
$[\varphi] := \{s \in S : M, s \models \varphi\}$

Recursive definition of $[\varphi]$

$$
\begin{array}{lcl}
[\top] & = & S \\
[\bot] & = & \{\} \\
[p] & = & \{s | p \in L(s)\} \\
[\neg\varphi_1] & = & S/[\varphi_1] \\
[\varphi_1 \wedge \varphi_2] & = & [\varphi_1] \cap [\varphi_2] \\
[\mathbf{EX}\varphi] & = & \{s \mid \exists s' \in [\varphi] \; s.t. \; \langle s, s' \rangle \in R\} \\
[\mathbf{EG}\beta] & = & \nu Z.(\; [\beta] \cap [\mathbf{EX}Z] \;) \\
[\mathbf{E}(\beta_1 \mathbf{U} \beta_2)] & = & \mu Z.(\; [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z]) \;)
\end{array}
$$

Case **EX**

Consider $\mathbf{EX}\varphi$:



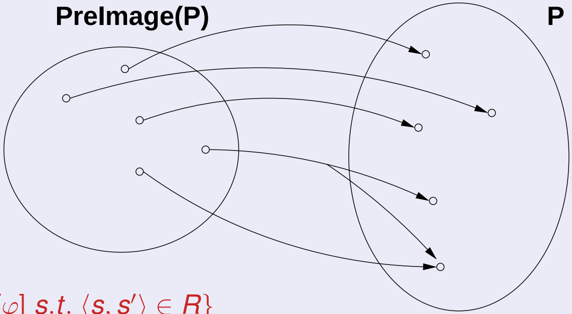
- $[\mathbf{EX}\varphi] = \{s \mid \exists s' \in [\varphi] \ s.t. \ \langle s, s' \rangle \in R\}$
- $[\mathbf{EX}\varphi]$ is said to be the Pre-image of $[\varphi]$ ($Preimage([\varphi])$)
- Key step of every CTL M.C. operation

Note

Preimage() is monotonic: $X \subseteq X' \Longrightarrow Preimage(X) \subseteq Preimage(X')$

Case EX

Consider **EX**φ:



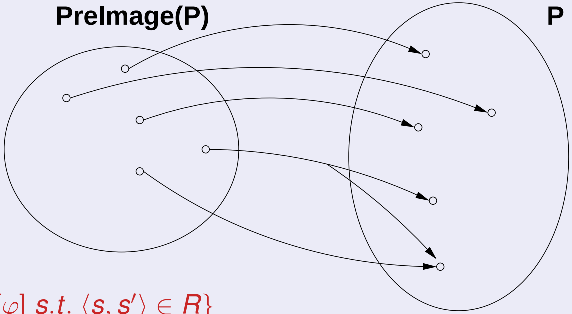
- $[\mathbf{EX}\varphi] = \{s \mid \exists s' \in [\varphi] \; s.t. \; \langle s, s' \rangle \in R\}$
- $[\mathbf{EX}\varphi]$ is said to be the Pre-image of $[\varphi]$ (*Preimage*($[\varphi]$))
- Key step of every CTL M.C. operation

Note

Preimage() is monotonic: $X \subseteq X' \Longrightarrow Preimage(X) \subseteq Preimage(X')$

Case **EX**

Consider $\mathbf{EX}\varphi$:



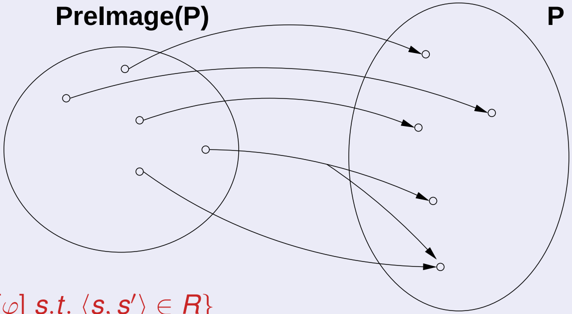
- $[\mathbf{EX}\varphi] = \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\}$
- $[\mathbf{EX}\varphi]$ is said to be the Pre-image of $[\varphi]$ ($Preimage([\varphi])$)
- Key step of every CTL M.C. operation

Note

Preimage() is monotonic: $X \subseteq X' \Longrightarrow Preimage(X) \subseteq Preimage(X')$

Case **EX**

Consider $\mathbf{EX}\varphi$:



- $[\mathbf{EX}\varphi] = \{s \mid \exists s' \in [\varphi] \ s.t. \ \langle s, s' \rangle \in R\}$
- $[\mathbf{EX}\varphi]$ is said to be the Pre-image of $[\varphi]$ ($Preimage([\varphi])$)
- Key step of every CTL M.C. operation

Note

Preimage() is monotonic: $X \subseteq X' \Longrightarrow Preimage(X) \subseteq Preimage(X')$

Case **EG**

Consider **EG**β:

- $\nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$: greatest fixed point of the function $F_\beta : 2^S \longmapsto 2^S$, s.t.

$$\begin{aligned} F_\beta([\varphi]) &= ([\beta] \cap Preimage([\varphi])) \\ &= ([\beta] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\}) \end{aligned}$$

- F_β Monotonic: $a \subseteq a' \Longrightarrow F_\beta(a) \subseteq F_\beta(a')$
  - (Tarski's theorem): $\nu x.F_\beta(x)$ always exists
  - (Kleene's theorem): $\nu x.F_\beta(x)$ can be computed as the limit $S \supseteq F_\beta(S) \supseteq F_\beta(F_\beta(S)) \supseteq \ldots$, in a finite number of steps.

Theorem (Clarke & Emerson)

$[\mathbf{EG}\beta] = \nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$

Case **EG**

Consider **EG**β:

- $\nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$: greatest fixed point of the function $F_\beta : 2^S \longmapsto 2^S$, s.t.

$$F_\beta([\varphi]) = ([\beta] \cap Preimage([\varphi])) = ([\beta] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\})$$

- F_β Monotonic: $a \subseteq a' \Longrightarrow F_\beta(a) \subseteq F_\beta(a')$
  - (Tarski's theorem): $\nu x.F_\beta(x)$ always exists
  - (Kleene's theorem): $\nu x.F_\beta(x)$ can be computed as the limit $S \supseteq F_\beta(S) \supseteq F_\beta(F_\beta(S)) \supseteq \ldots$, in a finite number of steps.

Theorem (Clarke & Emerson)

$[\mathbf{EG}\beta] = \nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$

Case EG

Consider **EG**β:

- $\nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$: greatest fixed point of the function $F_\beta : 2^S \longmapsto 2^S$, s.t.

$$F_\beta([\varphi]) = ([\beta] \cap Preimage([\varphi])) = ([\beta] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\})$$

- F_β Monotonic: $a \subseteq a' \Longrightarrow F_\beta(a) \subseteq F_\beta(a')$
  - (Tarski's theorem): $\nu x.F_\beta(x)$ always exists
  - (Kleene's theorem): $\nu x.F_\beta(x)$ can be computed as the limit $S \supseteq F_\beta(S) \supseteq F_\beta(F_\beta(S)) \supseteq \ldots$, in a finite number of steps.

Theorem (Clarke & Emerson)

$[\mathbf{EG}\beta] = \nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$

Case **EG**

Consider **EG**β:

- $\nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$: greatest fixed point of the function $F_\beta : 2^S \longmapsto 2^S$, s.t.

$$\begin{aligned} F_\beta([\varphi]) &= ([\beta] \cap Preimage([\varphi])) \\ &= ([\beta] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\}) \end{aligned}$$

- F_β Monotonic: $a \subseteq a' \Longrightarrow F_\beta(a) \subseteq F_\beta(a')$
  - (Tarski's theorem): $\nu x.F_\beta(x)$ always exists
  - (Kleene's theorem): $\nu x.F_\beta(x)$ can be computed as the limit $S \supseteq F_\beta(S) \supseteq F_\beta(F_\beta(S)) \supseteq \ldots$, in a finite number of steps.

Theorem (Clarke & Emerson)

$[\mathbf{EG}\beta] = \nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$

Case **EG**

Consider **EG**β:

- $\nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$: greatest fixed point of the function $F_\beta : 2^S \longmapsto 2^S$, s.t.

$$\begin{aligned} F_\beta([\varphi]) &= ([\beta] \cap Preimage([\varphi])) \\ &= ([\beta] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\}) \end{aligned}$$

- F_β Monotonic: $a \subseteq a' \Longrightarrow F_\beta(a) \subseteq F_\beta(a')$
  - (Tarski's theorem): $\nu x.F_\beta(x)$ always exists
  - (Kleene's theorem): $\nu x.F_\beta(x)$ can be computed as the limit $S \supseteq F_\beta(S) \supseteq F_\beta(F_\beta(S)) \supseteq \ldots$, in a finite number of steps.

Theorem (Clarke & Emerson)

$[\mathbf{EG}\beta] = \nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$

Case **EG** [cont.]

- We can compute $X := [\mathbf{EG}\beta]$ inductively as follows:

$$
\begin{array}{lllll}
X_0 & := & S & & \\
X_1 & := & F_\beta(S) & = & [\beta] \\
X_2 & := & F_\beta(F_\beta(S)) & = & [\beta] \cap Preimage(X_1) \\
\ldots & & & & \\
X_{j+1} & := & F_\beta^{j+1}(S) & = & [\beta] \cap Preimage(X_j)
\end{array}
$$

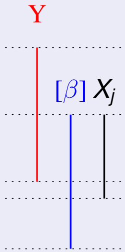
- Noticing that $X_1 = [\beta]$ and $X_{j+1} \subseteq X_j$ for every $j \geq 0$, and that $([\beta] \cap Y) \subseteq X_j \subseteq [\beta] \Longrightarrow ([\beta] \cap Y) = (X_j \cap Y)$, we can use instead the following inductive schema:
  - $X_1 \quad := [\beta]$
  - $X_{j+1} := X_j \cap Preimage(X_j)$

Case **EG** [cont.]

- We can compute $X := [\mathbf{EG}\beta]$ inductively as follows:

$$
\begin{array}{lcllcl}
X_0 & := & S & & & \\
X_1 & := & F_\beta(S) & = & [\beta] & \\
X_2 & := & F_\beta(F_\beta(S)) & = & [\beta] \cap Preimage(X_1) & \\
\ldots & & & & & \\
X_{j+1} & := & F_\beta^{j+1}(S) & = & [\beta] \cap Preimage(X_j) &
\end{array}
$$

- Noticing that $X_1 = [\beta]$ and $X_{j+1} \subseteq X_j$ for every $j \geq 0$, and that $([\beta] \cap Y) \subseteq X_j \subseteq [\beta] \Longrightarrow ([\beta] \cap Y) = (X_j \cap Y)$, we can use instead the following inductive schema:
  - $X_1 := [\beta]$
  - $X_{j+1} := X_j \cap Preimage(X_j)$



Case **EU**

Consider $\mathbf{E}(\beta_1 \mathbf{U} \beta_2)$:

- $\mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$: least fixed point of the function $F_{\beta_1,\beta_2} : 2^S \longmapsto 2^S$, s.t.

$$\begin{aligned} F_{\beta_1,\beta_2}([\varphi]) &= [\beta_2] \cup ([\beta_1] \cap Preimage([\varphi])) \\ &= [\beta_2] \cup ([\beta_1] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\}) \end{aligned}$$

- F_{β_1,β_2} Monotonic: $a \subseteq a' \Longrightarrow F_{\beta_1,\beta_2}(a) \subseteq F_{\beta_1,\beta_2}(a')$
  - (Tarski's theorem): $\mu x.F_{\beta_1,\beta_2}(x)$ always exists
  - (Kleene's theorem): $\mu x.F_{\beta_1,\beta_2}(x)$ can be computed as the limit $\emptyset \subseteq F_{\beta_1,\beta_2}(\emptyset) \subseteq F_{\beta_1,\beta_2}(F_{\beta_1,\beta_2}(\emptyset)) \subseteq \ldots$, in a finite number of steps.

Theorem (Clarke & Emerson)

$[\mathbf{E}(\beta_1 \mathbf{U} \beta_2)] = \mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$

Case **EU**

Consider $\mathbf{E}(\beta_1 \mathbf{U} \beta_2)$:

- $\mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$: least fixed point of the function $F_{\beta_1,\beta_2} : 2^S \longmapsto 2^S$, s.t.

$$F_{\beta_1,\beta_2}([\varphi]) = [\beta_2] \cup ([\beta_1] \cap Preimage([\varphi])) = [\beta_2] \cup ([\beta_1] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\})$$

- F_{β_1,β_2} Monotonic: $a \subseteq a' \Longrightarrow F_{\beta_1,\beta_2}(a) \subseteq F_{\beta_1,\beta_2}(a')$
  - (Tarski's theorem): $\mu x.F_{\beta_1,\beta_2}(x)$ always exists
  - (Kleene's theorem): $\mu x.F_{\beta_1,\beta_2}(x)$ can be computed as the limit $\emptyset \subseteq F_{\beta_1,\beta_2}(\emptyset) \subseteq F_{\beta_1,\beta_2}(F_{\beta_1,\beta_2}(\emptyset)) \subseteq \ldots$, in a finite number of steps.

Theorem (Clarke & Emerson)

$[\mathbf{E}(\beta_1 \mathbf{U} \beta_2)] = \mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$

Case **EU**

Consider $\mathbf{E}(\beta_1 \mathbf{U} \beta_2)$:

- $\mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$: least fixed point of the function $F_{\beta_1,\beta_2} : 2^S \longmapsto 2^S$, s.t.

$$F_{\beta_1,\beta_2}([\varphi]) = [\beta_2] \cup ([\beta_1] \cap Preimage([\varphi])) = [\beta_2] \cup ([\beta_1] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\})$$

- F_{β_1,β_2} Monotonic: $a \subseteq a' \Longrightarrow F_{\beta_1,\beta_2}(a) \subseteq F_{\beta_1,\beta_2}(a')$
  - (Tarski's theorem): $\mu x.F_{\beta_1,\beta_2}(x)$ always exists
  - (Kleene's theorem): $\mu x.F_{\beta_1,\beta_2}(x)$ can be computed as the limit $\emptyset \subseteq F_{\beta_1,\beta_2}(\emptyset) \subseteq F_{\beta_1,\beta_2}(F_{\beta_1,\beta_2}(\emptyset)) \subseteq \ldots$, in a finite number of steps.

Theorem (Clarke & Emerson)

$[\mathbf{E}(\beta_1 \mathbf{U} \beta_2)] = \mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$

Case **EU**

Consider $\mathbf{E}(\beta_1 \mathbf{U} \beta_2)$:

- $\mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$: least fixed point of the function $F_{\beta_1,\beta_2} : 2^S \longmapsto 2^S$, s.t.

$$\begin{aligned} F_{\beta_1,\beta_2}([\varphi]) &= [\beta_2] \cup ([\beta_1] \cap Preimage([\varphi])) \\ &= [\beta_2] \cup ([\beta_1] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\}) \end{aligned}$$

- F_{β_1,β_2} Monotonic: $a \subseteq a' \Longrightarrow F_{\beta_1,\beta_2}(a) \subseteq F_{\beta_1,\beta_2}(a')$
  - (Tarski's theorem): $\mu x.F_{\beta_1,\beta_2}(x)$ always exists
  - (Kleene's theorem): $\mu x.F_{\beta_1,\beta_2}(x)$ can be computed as the limit $\emptyset \subseteq F_{\beta_1,\beta_2}(\emptyset) \subseteq F_{\beta_1,\beta_2}(F_{\beta_1,\beta_2}(\emptyset)) \subseteq \ldots$, in a finite number of steps.

Theorem (Clarke & Emerson)

$[\mathbf{E}(\beta_1 \mathbf{U} \beta_2)] = \mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$

Case EU

Consider $\mathbf{E}(\beta_1 \mathbf{U} \beta_2)$:

- $\mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$: least fixed point of the function $F_{\beta_1,\beta_2} : 2^S \longmapsto 2^S$, s.t.

$$\begin{aligned} F_{\beta_1,\beta_2}([\varphi]) &= [\beta_2] \cup ([\beta_1] \cap Preimage([\varphi])) \\ &= [\beta_2] \cup ([\beta_1] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\}) \end{aligned}$$

- F_{β_1,β_2} Monotonic: $a \subseteq a' \Longrightarrow F_{\beta_1,\beta_2}(a) \subseteq F_{\beta_1,\beta_2}(a')$
  - (Tarski's theorem): $\mu x.F_{\beta_1,\beta_2}(x)$ always exists
  - (Kleene's theorem): $\mu x.F_{\beta_1,\beta_2}(x)$ can be computed as the limit $\emptyset \subseteq F_{\beta_1,\beta_2}(\emptyset) \subseteq F_{\beta_1,\beta_2}(F_{\beta_1,\beta_2}(\emptyset)) \subseteq \ldots$, in a finite number of steps.

Theorem (Clarke & Emerson)

$[\mathbf{E}(\beta_1 \mathbf{U} \beta_2)] = \mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$

Case **EU** [cont.]

- We can compute $X := [\mathbf{E}(\beta_1 \mathbf{U} \beta_2)]$ inductively as follows:

$$
\begin{array}{lcll}
X_0 & := & \emptyset & \\
X_1 & := & F_{\beta_1,\beta_2}(\emptyset) & = [\beta_2] \\
X_2 & := & F_{\beta_1,\beta_2}(F_{\beta_1,\beta_2}(\emptyset)) & = [\beta_2] \cup ([\beta_1] \cap \mathit{Preimage}(X_1)) \\
\ldots & & & \\
X_{j+1} & := & F^{j+1}_{\beta_1,\beta_2}(\emptyset)) & = [\beta_2] \cup ([\beta_1] \cap \mathit{Preimage}(X_j))
\end{array}
$$

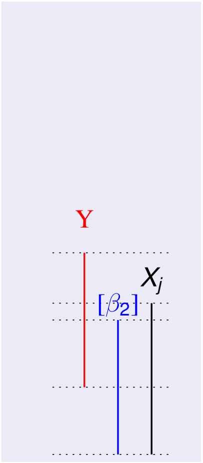
- Noticing that $X_1 = [\beta_2]$ and $X_{j+1} \supseteq X_j$ for every $j \geq 0$, and that $([\beta_2] \cup Y) \supseteq X_j \supseteq [\beta_2] \Longrightarrow ([\beta_2] \cup Y) = (X_j \cup Y)$,
we can use instead the following inductive schema:
  - $X_1 \quad := [\beta_2]$
  - $X_{j+1} := X_j \cup ([\beta_1] \cap \mathit{Preimage}(X_j))$

Case **EU** [cont.]

- We can compute $X := [\mathbf{E}(\beta_1 \mathbf{U} \beta_2)]$ inductively as follows:

$$
\begin{array}{lcl}
X_0 & := & \emptyset \\
X_1 & := & F_{\beta_1,\beta_2}(\emptyset) \quad = \quad [\beta_2] \\
X_2 & := & F_{\beta_1,\beta_2}(F_{\beta_1,\beta_2}(\emptyset)) \quad = \quad [\beta_2] \cup ([\beta_1] \cap \mathit{Preimage}(X_1)) \\
\ldots & & \\
X_{j+1} & := & F^{j+1}_{\beta_1,\beta_2}(\emptyset)) \quad = \quad [\beta_2] \cup ([\beta_1] \cap \mathit{Preimage}(X_j))
\end{array}
$$

- Noticing that $X_1 = [\beta_2]$ and $X_{j+1} \supseteq X_j$ for every $j \geq 0$, and that $([\beta_2] \cup Y) \supseteq X_j \supseteq [\beta_2] \Longrightarrow ([\beta_2] \cup Y) = (X_j \cup Y)$, we can use instead the following inductive schema:
  - $X_1 := [\beta_2]$
  - $X_{j+1} := X_j \cup ([\beta_1] \cap \mathit{Preimage}(X_j))$



A relevant subcase: **EF**

- $\mathbf{EF}\beta = \mathbf{E}(\top \mathbf{U} \beta)$
- $[\top] = S \implies [\top] \cap Preimage(X_j) = Preimage(X_j)$
- We can compute $X := [\mathbf{EF}\beta]$ inductively as follows:
  - $X_1 := [\beta]$
  - $X_{j+1} := X_j \cup Preimage(X_j)$

A relevant subcase: **EF**

- $\mathbf{EF}\beta = \mathbf{E}(\top\mathbf{U}\beta)$
- $[\top] = S \implies [\top] \cap Preimage(X_j) = Preimage(X_j)$
- We can compute $X := [\mathbf{EF}\beta]$ inductively as follows:
  - $X_1 := [\beta]$
  - $X_{j+1} := X_j \cup Preimage(X_j)$

A relevant subcase: **EF**

- $\mathbf{EF}\beta = \mathbf{E}(\top \mathbf{U} \beta)$
- $[\top] = S \Longrightarrow [\top] \cap \mathit{Preimage}(X_j) = \mathit{Preimage}(X_j)$
- We can compute $X := [\mathbf{EF}\beta]$ inductively as follows:
  - $X_1 \quad := [\beta]$
  - $X_{j+1} := X_j \cup \mathit{Preimage}(X_j)$

Outline

General Schema

- Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**
- A general M.C. algorithm (fix-point):
  1. for every $\varphi_i \in Sub(\varphi)$, find $[\varphi_i]$
  2. Check if $I \subseteq [\varphi]$
- Subformulas $Sub(\varphi)$ of φ are checked bottom-up
- To compute each $[\varphi_i]$: if the main operator of φ_i is a
  - Propositional atoms: apply labeling function
  - Boolean operator: apply standard set operations
  - temporal operator: appy recursively the tableaux rules, until a fixpoint is reached

General Schema

- Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**
- A general M.C. algorithm (fix-point):
  1. for every $\varphi_i \in Sub(\varphi)$, find $[\varphi_i]$
  2. Check if $I \subseteq [\varphi]$
- Subformulas $Sub(\varphi)$ of φ are checked bottom-up
- To compute each $[\varphi_i]$: if the main operator of φ_i is a
  - Propositional atoms: apply labeling function
  - Boolean operator: apply standard set operations
  - temporal operator: appy recursively the tableaux rules, until a fixpoint is reached

General Schema

- Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**
- A general M.C. algorithm (fix-point):
  1. for every $\varphi_i \in Sub(\varphi)$, find $[\varphi_i]$
  2. Check if $I \subseteq [\varphi]$
- Subformulas $Sub(\varphi)$ of φ are checked bottom-up
- To compute each $[\varphi_i]$: if the main operator of φ_i is a
  - Propositional atoms: apply labeling function
  - Boolean operator: apply standard set operations
  - temporal operator: appy recursively the tableaux rules, until a fixpoint is reached

General Schema

- Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**
- A general M.C. algorithm (fix-point):
  1. for every $\varphi_i \in Sub(\varphi)$, find $[\varphi_i]$
  2. Check if $I \subseteq [\varphi]$
- Subformulas $Sub(\varphi)$ of φ are checked bottom-up
- To compute each $[\varphi_i]$: if the main operator of φ_i is a
  - Propositional atoms: apply labeling function
  - Boolean operator: apply standard set operations
  - temporal operator: appy recursively the tableaux rules, until a fixpoint is reached

General Schema

- Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**
- A general M.C. algorithm (fix-point):
  1. for every $\varphi_i \in Sub(\varphi)$, find $[\varphi_i]$
  2. Check if $I \subseteq [\varphi]$
- Subformulas $Sub(\varphi)$ of φ are checked bottom-up
- To compute each $[\varphi_i]$: if the main operator of φ_i is a
  - Propositional atoms: apply labeling function
  - Boolean operator: apply standard set operations
  - temporal operator: appy recursively the tableaux rules, until a fixpoint is reached

General Schema

- Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**
- A general M.C. algorithm (fix-point):
  1. for every $\varphi_i \in Sub(\varphi)$, find $[\varphi_i]$
  2. Check if $I \subseteq [\varphi]$
- Subformulas $Sub(\varphi)$ of φ are checked bottom-up
- To compute each $[\varphi_i]$: if the main operator of φ_i is a
  - Propositional atoms: apply labeling function
  - Boolean operator: apply standard set operations
  - temporal operator: appy recursively the tableaux rules, until a fixpoint is reached

General Schema

- Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**
- A general M.C. algorithm (fix-point):
  1. for every $\varphi_i \in Sub(\varphi)$, find $[\varphi_i]$
  2. Check if $I \subseteq [\varphi]$
- Subformulas $Sub(\varphi)$ of φ are checked bottom-up
- To compute each $[\varphi_i]$: if the main operator of φ_i is a
  - Propositional atoms: apply labeling function
  - Boolean operator: apply standard set operations
  - temporal operator: appy recursively the tableaux rules, until a fixpoint is reached

General Schema

- Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**
- A general M.C. algorithm (fix-point):
  1. for every $\varphi_i \in Sub(\varphi)$, find $[\varphi_i]$
  2. Check if $I \subseteq [\varphi]$
- Subformulas $Sub(\varphi)$ of φ are checked bottom-up
- To compute each $[\varphi_i]$: if the main operator of φ_i is a
  - Propositional atoms: apply labeling function
  - Boolean operator: apply standard set operations
  - temporal operator: appy recursively the tableaux rules, until a fixpoint is reached

General Schema

- Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**
- A general M.C. algorithm (fix-point):
  1. for every $\varphi_i \in Sub(\varphi)$, find $[\varphi_i]$
  2. Check if $I \subseteq [\varphi]$
- Subformulas $Sub(\varphi)$ of φ are checked bottom-up
- To compute each $[\varphi_i]$: if the main operator of φ_i is a
  - Propositional atoms: apply labeling function
  - Boolean operator: apply standard set operations
  - temporal operator: appy recursively the tableaux rules, until a fixpoint is reached

General M.C. Procedure

```
state_set Check(CTL_formula β) {
    case β of
    ⊤:          return S;
    ⊥:          return {};
    p:          return {s | p ∈ L(s)};
    ¬β1:        return S / Check(β1);
    β1 ∧ β2:    return Check(β1) ∩ Check(β2);
    EXβ1:       return PreImage(Check(β1));
    EGβ1:       return Check_EG(Check(β1));
    E(β1Uβ2):   return Check_EU(Check(β1),Check(β2));
}
```

PreImage

Compute [$\mathbf{EX}\beta$]

```
state_set PreImage(state_set [β]) {
    X := {};
    for each s ∈ S do
        for each s' s.t. s' ∈ [β] and ⟨s, s'⟩ ∈ R do
            X := X ∪ {s};
return X;
}
```

Check_EG

Compute [$\mathbf{EG}\beta$]

state_set Check_EG(**state_set** [β]) {
    $X' := [\beta]$;
    repeat
        $X := X'$;
        $X' := X \cap PreImage(X)$;
    until $(X' = X)$;
return X;
}

Check_EU

Compute $[\mathbf{E}(\beta_1 \mathbf{U} \beta_2)]$

state_set Check_EU(**state_set** $[\beta_1]$,$[\beta_2]$) {
  $X' := [\beta_2]$;
  repeat
    $X := X'$;
    $X' := X \cup ([\beta_1] \cap PreImage(X))$;
  until $(X' = X)$;
return X;
}

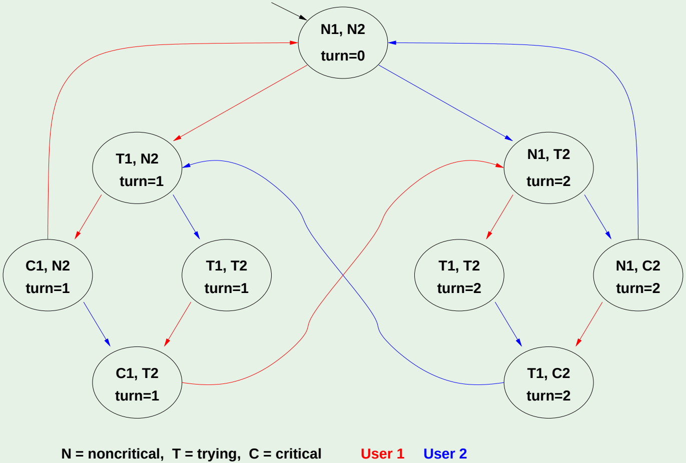
A relevant subcase: Check_EF

Compute [EFβ]

state_set Check_EF(**state_set** [β]) {
    $X' := [\beta]$;
    repeat
        $X := X'$;
        $X' := X \cup PreImage(X)$;
    until $(X' = X)$;
return X;
}

Outline

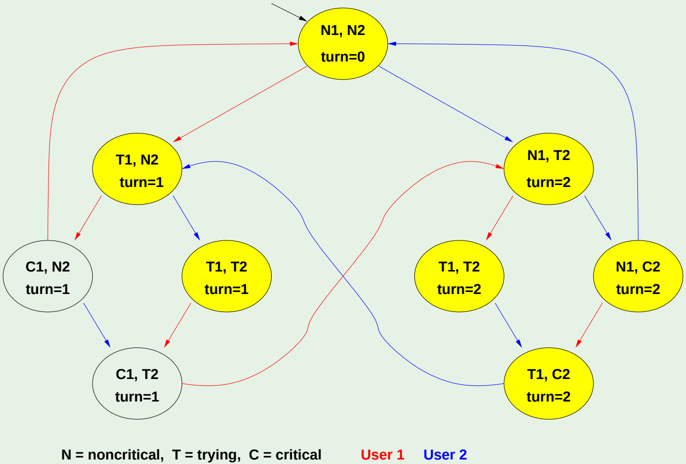
Example 1: fairness



$M \models \mathbf{AGAF}C_1$? $\Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1$?

Example 1: fairness

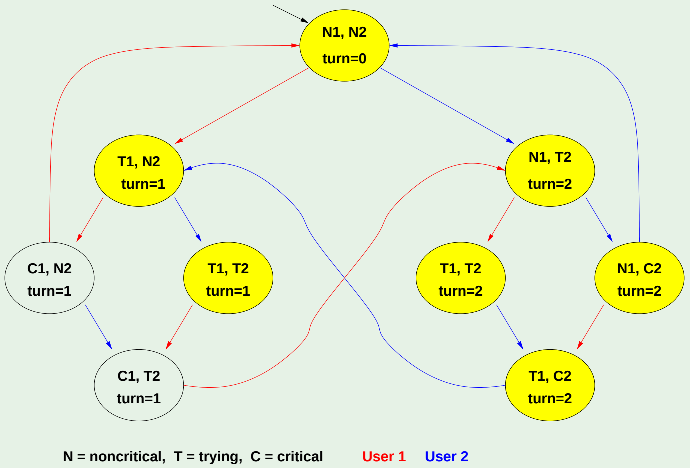
$[\neg C_1]$



$M \models \textbf{AGAF}C_1$? $\Longrightarrow M \models \neg\textbf{EFEG}\neg C_1$?

Example 1: fairness

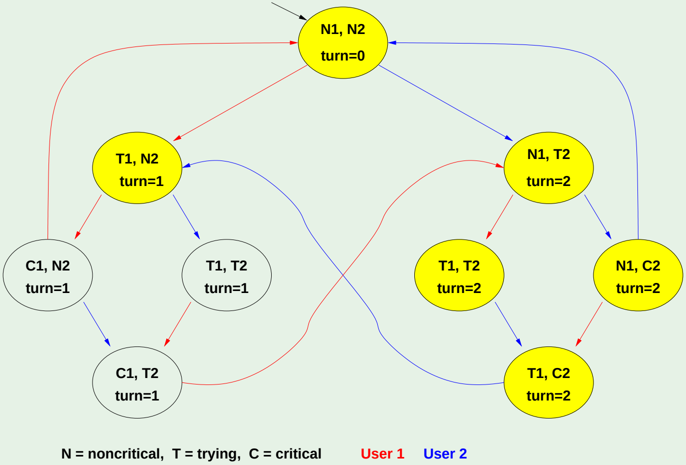
[**EG**$\neg C_1$], step 0:



$M \models$ **AGAF**C_1 ? $\Longrightarrow M \models \neg$**EFEG**$\neg C_1$?

Example 1: fairness

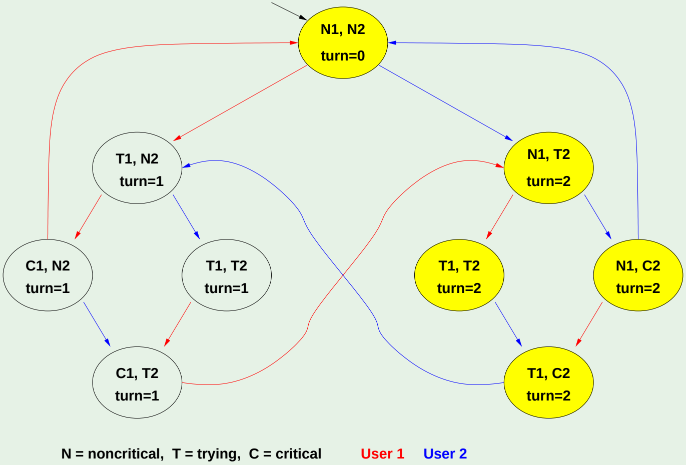
[**EG**$\neg C_1$], step 1:



$M \models$ **AGAF**C_1 ? $\Longrightarrow M \models \neg$**EFEG**$\neg C_1$?

Example 1: fairness

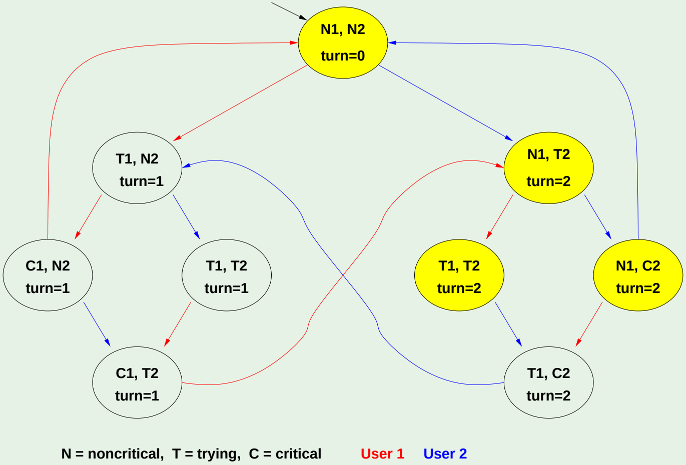
[**EG**$\neg C_1$], step 2:



$M \models$ **AGAF**C_1 ? $\Longrightarrow M \models \neg$**EFEG**$\neg C_1$?

Example 1: fairness

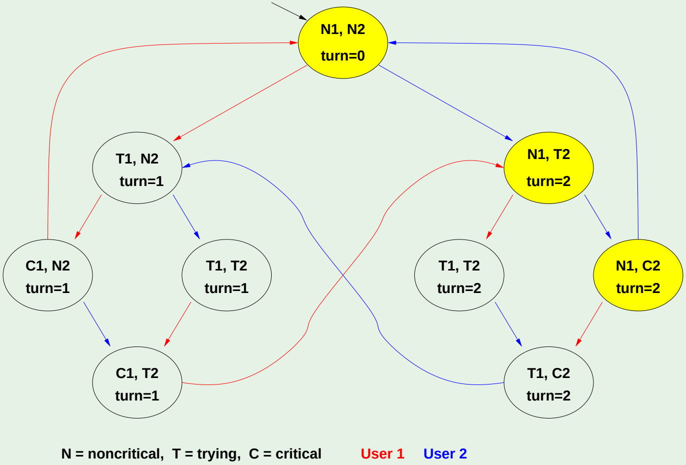
[**EG**$\neg C_1$], step 3:



$M \models \mathbf{AGAF}C_1$? $\Longrightarrow$ $M \models \neg\mathbf{EFEG}\neg C_1$?

Example 1: fairness

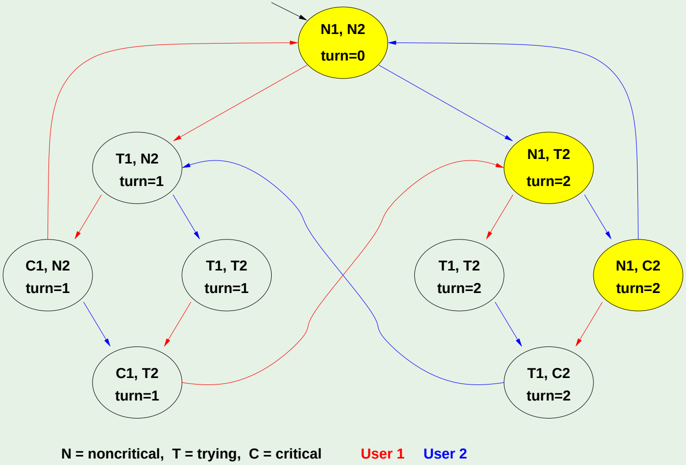
[**EG**$\neg C_1$], step 4:



$M \models \mathbf{AGAF} C_1$? $\Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1$?

Example 1: fairness

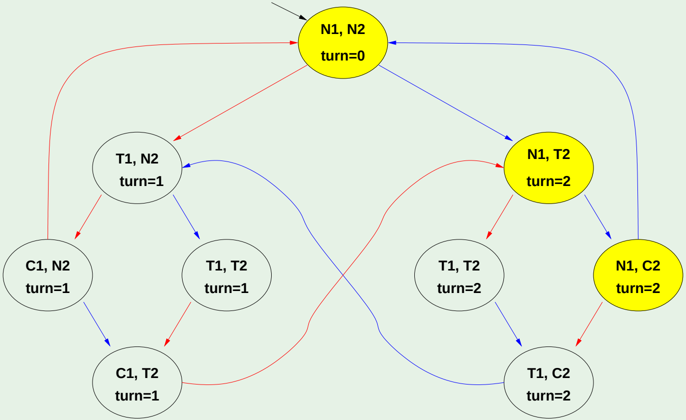
$[\mathbf{EG}\neg C_1]$, FIXPOINT!



$M \models \mathbf{AGAF} C_1$? $\Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1$?

Example 1: fairness

[**EFEG**$\neg C_1$], STEP 0

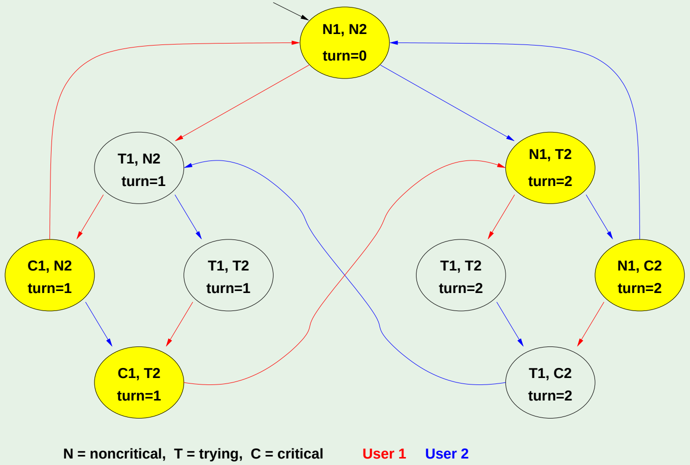


N = noncritical, T = trying, C = critical **User 1** **User 2**

$M \models$ **AGAF**C_1 ? $\Longrightarrow M \models \neg$**EFEG**$\neg C_1$?

Example 1: fairness

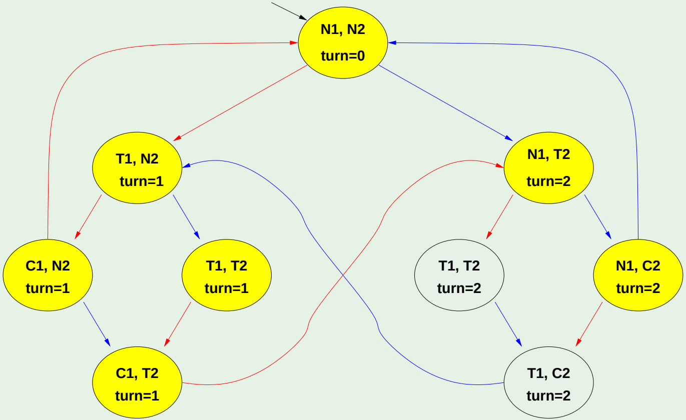
[**EFEG**$\neg C_1$], STEP 1



$M \models$ **AGAF**C_1 ? $\Longrightarrow M \models \neg$**EFEG**$\neg C_1$?

Example 1: fairness

$[\mathbf{EFEG}\neg C_1]$, STEP 2

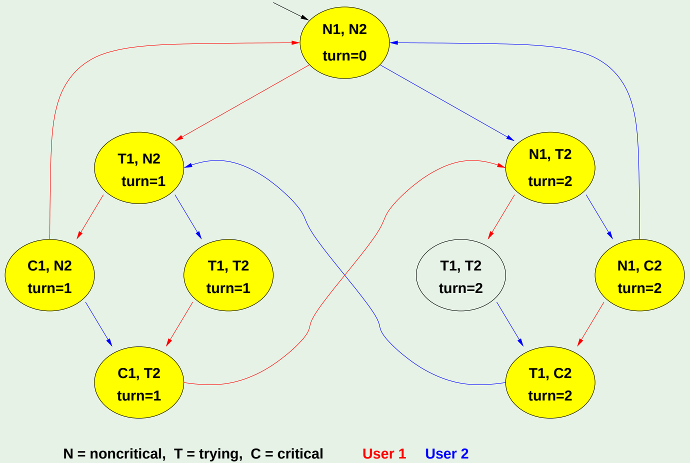


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1$? $\Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1$?

Example 1: fairness

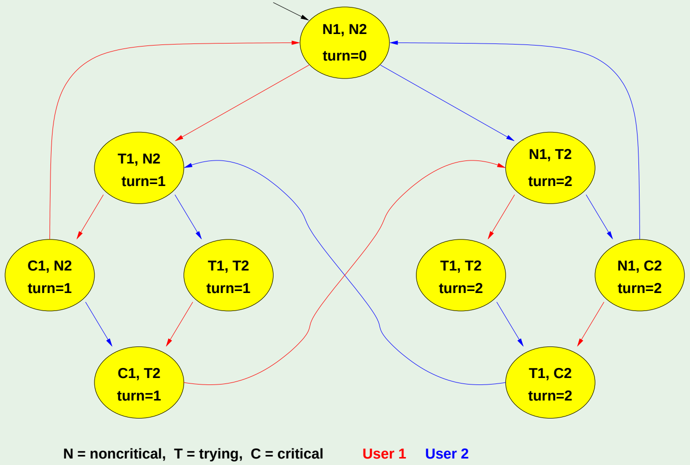
$[\mathbf{EFEG}\neg C_1]$, STEP 3



$M \models \mathbf{AGAF}C_1 \; ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \; ?$

Example 1: fairness

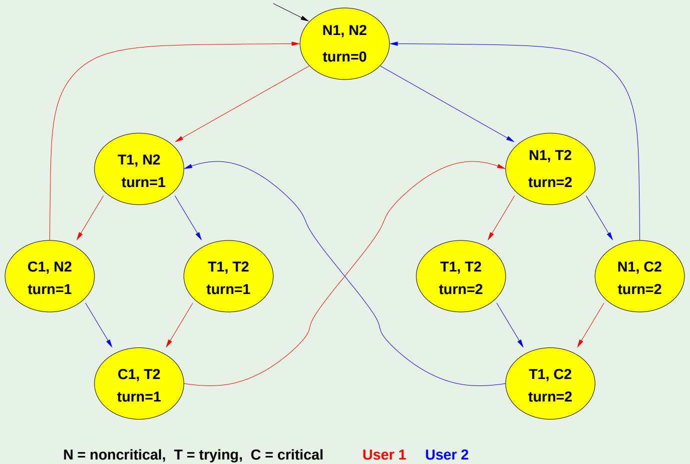
[**EFEG**$\neg C_1$], STEP 4



$M \models$ **AGAF**C_1 ? $\Longrightarrow M \models \neg$**EFEG**$\neg C_1$?

Example 1: fairness

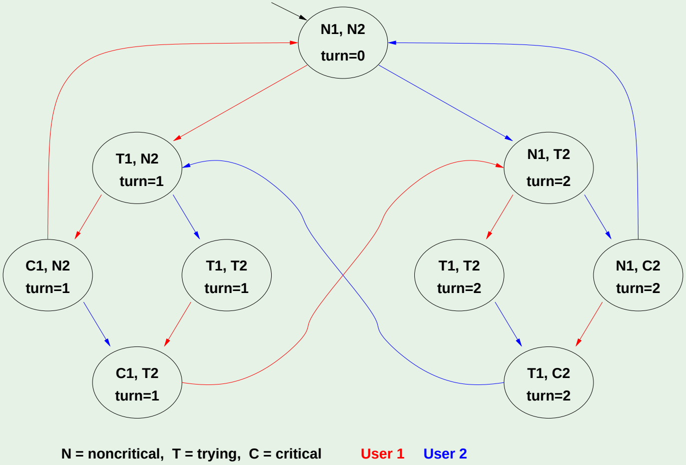
$[\mathbf{EFEG}\neg C_1]$, FIXPOINT!



$M \models \mathbf{AGAF} C_1 \ ? \Longrightarrow M \models \neg \mathbf{EFEG} \neg C_1 \ ?$

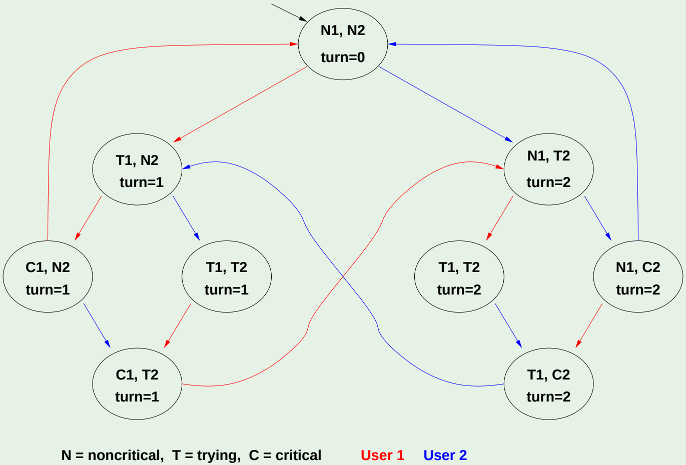
Example 1: fairness

$[\neg \mathbf{EFEG} \neg C_1]$



$M \models \mathbf{AGAF} C_1 \ ? \Longrightarrow M \models \neg \mathbf{EFEG} \neg C_1 \ ? \Longrightarrow$ NO!

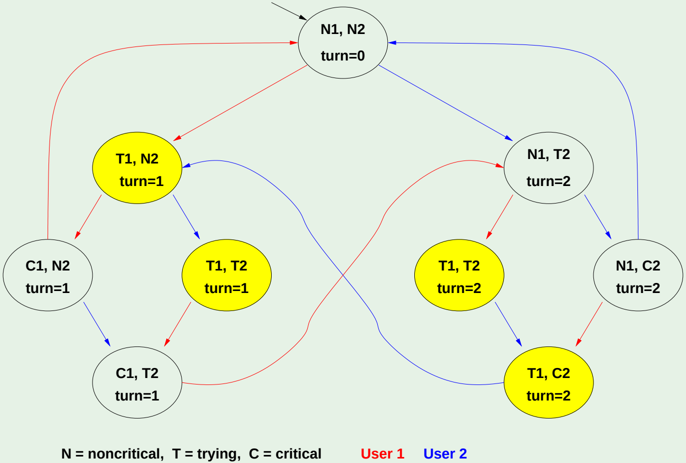
Example 2: liveness



$$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF} C_1)\ ? \Longrightarrow M \models \neg \mathbf{EF}(T_1 \wedge \mathbf{EG} \neg C_1)\ ?$$

Example 2: liveness

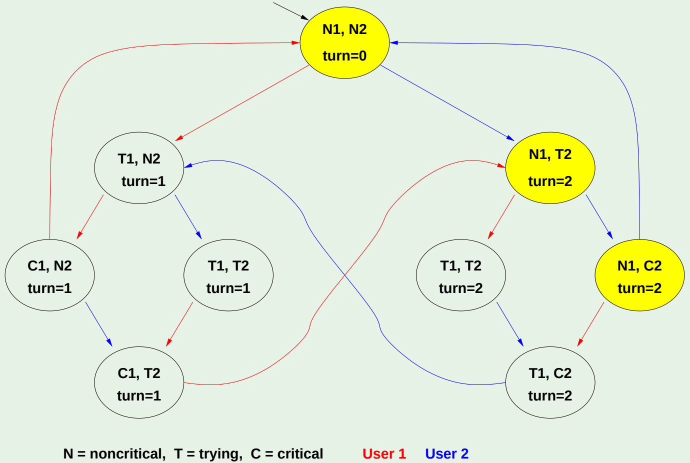
$[T_1]$:



$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF} C_1)$? $\Longrightarrow$ $M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)$?

Example 2: liveness

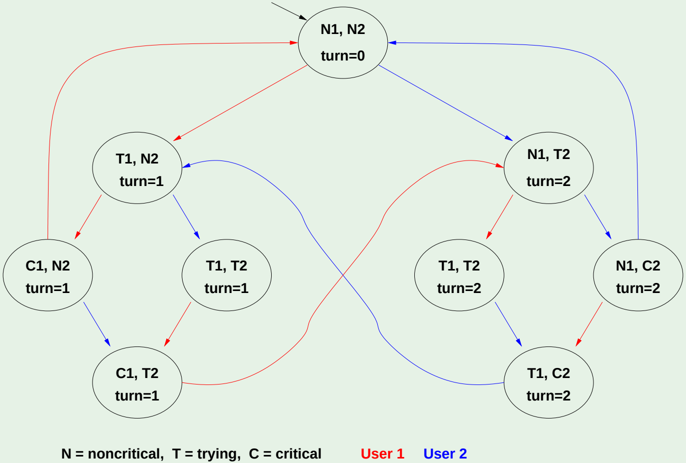
[**EG**$\neg C_1$], STEPS 0-4: (see previous example)



$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF} C_1)$? $\Longrightarrow M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)$?

Example 2: liveness

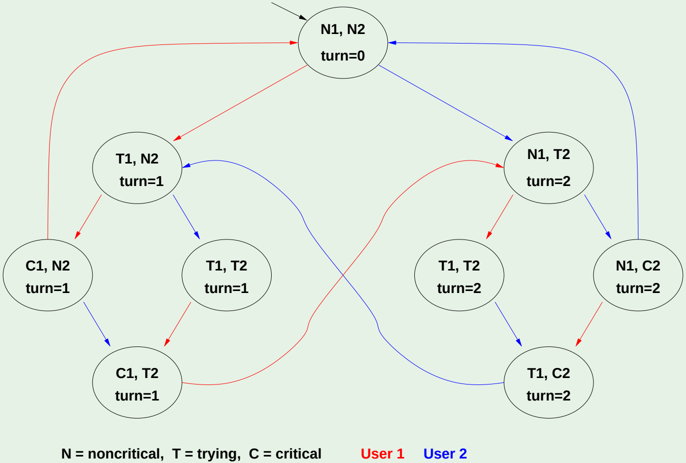
$[T_1 \wedge \mathbf{EG}\neg C_1]$:



$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}C_1)\ ? \Longrightarrow M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)\ ?$

Example 2: liveness

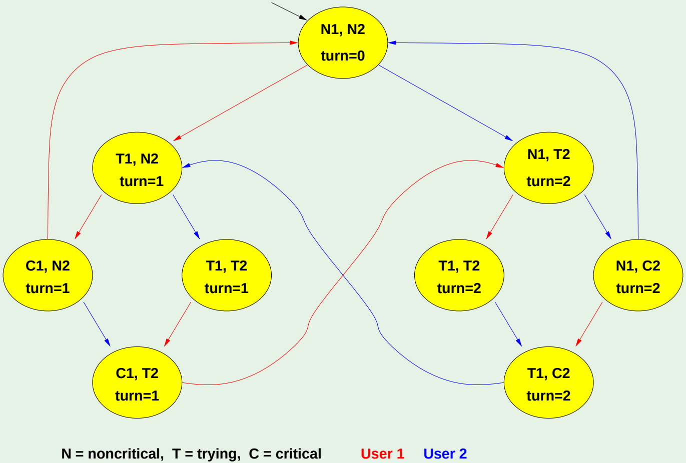
$[\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)]$:



$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}C_1)\ ? \Longrightarrow M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)\ ?$

Example 2: liveness

$[\neg \mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)]$:



$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}C_1)$? $\Longrightarrow M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)$? YES!

The property verified is...

Homework

Apply the same process to all the CTL examples of Chapter 3.

Complexity of CTL Model Checking: $M \models \varphi$

- Step 1: compute $[\varphi]$
  - Compute $[\varphi]$ bottom-up on the $O(|\varphi|)$ sub-formulas of φ: $O(|\varphi|)$ steps...
  - ... each requiring at most exploring $O(|M|)$ states

  $\Longrightarrow O(|M| \cdot |\varphi|)$ steps
- Step 2: check $I \subseteq [\varphi]$: $O(|M|)$

$\Longrightarrow O(|M| \cdot |\varphi|)$

Complexity of CTL Model Checking: $M \models \varphi$

- Step 1: compute $[\varphi]$
  - Compute $[\varphi]$ bottom-up on the $O(|\varphi|)$ sub-formulas of φ: $O(|\varphi|)$ steps...
  - ... each requiring at most exploring $O(|M|)$ states

  $\Longrightarrow O(|M| \cdot |\varphi|)$ steps
- Step 2: check $I \subseteq [\varphi]$: $O(|M|)$

$\Longrightarrow O(|M| \cdot |\varphi|)$

Complexity of CTL Model Checking: $M \models \varphi$

- Step 1: compute $[\varphi]$
  - Compute $[\varphi]$ bottom-up on the $O(|\varphi|)$ sub-formulas of φ: $O(|\varphi|)$ steps...
  - ... each requiring at most exploring $O(|M|)$ states

  $\Longrightarrow O(|M| \cdot |\varphi|)$ steps
- Step 2: check $I \subseteq [\varphi]$: $O(|M|)$

$\Longrightarrow O(|M| \cdot |\varphi|)$

Complexity of CTL Model Checking: $M \models \varphi$

- Step 1: compute $[\varphi]$
  - Compute $[\varphi]$ bottom-up on the $O(|\varphi|)$ sub-formulas of φ: $O(|\varphi|)$ steps...
  - ... each requiring at most exploring $O(|M|)$ states

  $\Longrightarrow O(|M| \cdot |\varphi|)$ steps
- Step 2: check $I \subseteq [\varphi]$: $O(|M|)$

$\Longrightarrow O(|M| \cdot |\varphi|)$

Outline

Model Checking of Invariants

- Invariant properties have the form **AG p**, where **p** in Boolean (e.g., **AG**$\neg$*bad*)
- Checking invariants is the negation of a reachability problem:
  - is there a reachable state that is also a bad state? ($\mathbf{AG}\neg bad = \neg\mathbf{EF} bad$)
- Standard M.C. algorithm reasons backward from the *bad* by iteratively applying PreImage:

$$Y' := Y \cup PreImage(Y)$$

  until a fixed point is reached.
  Then the complement is computed and I is checked for inclusion in the resulting set.
- Better algorithm: reasons backward from the *bad* by iteratively applying PreImage:

$$Y' := Y \cup PreImage(Y)$$

  until (i) it intersect $[I]$ or (ii) a fixed point is reached

Model Checking of Invariants

- Invariant properties have the form **AG p**, where **p** in Boolean (e.g., **AG**$\neg$*bad*)
- Checking invariants is the negation of a reachability problem:
  - is there a reachable state that is also a bad state? (**AG**$\neg$*bad* $=$ $\neg$**EF***bad*)
- Standard M.C. algorithm reasons backward from the *bad* by iteratively applying PreImage:

$$Y' := Y \cup PreImage(Y)$$

  until a fixed point is reached.
  Then the complement is computed and I is checked for inclusion in the resulting set.
- Better algorithm: reasons backward from the *bad* by iteratively applying PreImage:

$$Y' := Y \cup PreImage(Y)$$

  until (i) it intersect [I] or (ii) a fixed point is reached

Model Checking of Invariants

- Invariant properties have the form **AG p**, where **p** in Boolean (e.g., $\mathbf{AG}\neg bad$)
- Checking invariants is the negation of a reachability problem:
  - is there a reachable state that is also a bad state? ($\mathbf{AG}\neg bad = \neg\mathbf{EF} bad$)
- Standard M.C. algorithm reasons backward from the *bad* by iteratively applying PreImage:

$$Y' := Y \cup PreImage(Y)$$

  until a fixed point is reached.
  Then the complement is computed and I is checked for inclusion in the resulting set.
- Better algorithm: reasons backward from the *bad* by iteratively applying PreImage:

$$Y' := Y \cup PreImage(Y)$$

  until (i) it intersect [I] or (ii) a fixed point is reached

Model Checking of Invariants

- Invariant properties have the form **AG p**, where **p** in Boolean (e.g., **AG**$\neg$*bad*)
- Checking invariants is the negation of a reachability problem:
  - is there a reachable state that is also a bad state? (**AG**$\neg$*bad* $=$ $\neg$**EF***bad*)
- Standard M.C. algorithm reasons backward from the *bad* by iteratively applying PreImage:

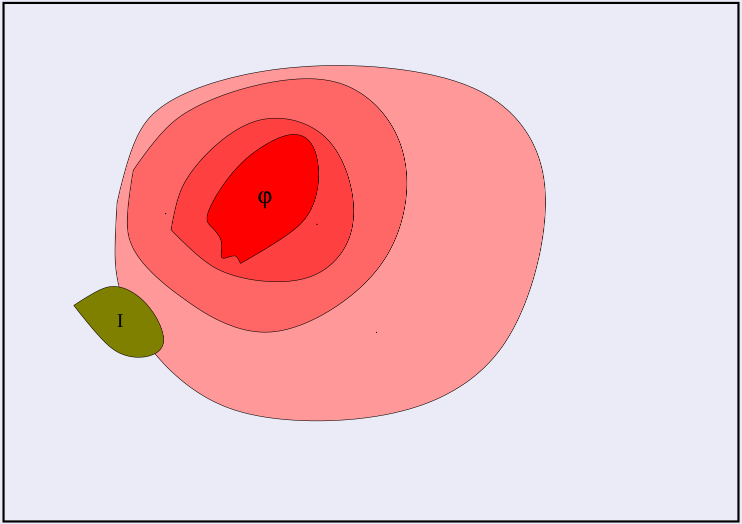
$$Y' := Y \cup PreImage(Y)$$

  until a fixed point is reached.
  Then the complement is computed and I is checked for inclusion in the resulting set.
- Better algorithm: reasons backward from the *bad* by iteratively applying PreImage:

$$Y' := Y \cup PreImage(Y)$$

  until (i) it intersect $[I]$ or (ii) a fixed point is reached

Model Checking of Invariants [cont.]



Forward Model Checking of Invariants

Alternative algorithm (often more efficient): forward checking

- Compute the set of bad states $[bad]$
- Compute the set of initial states I
- Compute incrementally the set of reachable states from I until (i) it intersect $[bad]$ or (ii) a fixed point is reached
- Basic step is the (Forward) Image:

$$Image(Y) \stackrel{\text{def}}{=} \{s' \mid s \in Y \text{ and } R(s, s') \text{ holds}\}$$

- Simplest form: compute the set of reachable states.

Forward Model Checking of Invariants

Alternative algorithm (often more efficient): forward checking

- Compute the set of bad states $[bad]$
- Compute the set of initial states I
- Compute incrementally the set of reachable states from I until (i) it intersect $[bad]$ or (ii) a fixed point is reached
- Basic step is the (Forward) Image:

$$Image(Y) \stackrel{\text{def}}{=} \{s' \mid s \in Y \text{ and } R(s, s') \text{ holds}\}$$

- Simplest form: compute the set of reachable states.

Forward Model Checking of Invariants

Alternative algorithm (often more efficient): forward checking

- Compute the set of bad states $[bad]$
- Compute the set of initial states I
- Compute incrementally the set of reachable states from I until (i) it intersect $[bad]$ or (ii) a fixed point is reached
- Basic step is the (Forward) Image:

$$Image(Y) \stackrel{\text{def}}{=} \{s' \mid s \in Y \text{ and } R(s, s') \text{ holds}\}$$

- Simplest form: compute the set of reachable states.

Forward Model Checking of Invariants

Alternative algorithm (often more efficient): forward checking

- Compute the set of bad states [*bad*]
- Compute the set of initial states I
- Compute incrementally the set of reachable states from I until (i) it intersect [*bad*] or (ii) a fixed point is reached
- Basic step is the (Forward) Image:

$$Image(Y) \stackrel{\text{def}}{=} \{s' \mid s \in Y \text{ and } R(s, s') \text{ holds}\}$$

- Simplest form: compute the set of reachable states.

Forward Model Checking of Invariants

Alternative algorithm (often more efficient): forward checking

- Compute the set of bad states $[bad]$
- Compute the set of initial states I
- Compute incrementally the set of reachable states from I until (i) it intersect $[bad]$ or (ii) a fixed point is reached
- Basic step is the (Forward) Image:

$$Image(Y) \stackrel{\text{def}}{=} \{s' \mid s \in Y \text{ and } R(s, s') \text{ holds}\}$$

- Simplest form: compute the set of reachable states.

Forward Model Checking of Invariants

Alternative algorithm (often more efficient): forward checking

- Compute the set of bad states $[bad]$
- Compute the set of initial states I
- Compute incrementally the set of reachable states from I until (i) it intersect $[bad]$ or (ii) a fixed point is reached
- Basic step is the (Forward) Image:

$$Image(Y) \stackrel{\text{def}}{=} \{s' \mid s \in Y \textit{ and } R(s, s') \textit{ holds}\}$$

- Simplest form: compute the set of reachable states.

Computing Reachable states: basic

```
State_Set Compute_reachable() {
    Y' := I; Y := ∅;
    while (Y' ≠ Y)
        Y := Y';
        Y' := Y ∪ Image(Y);
    }
return Y;
}
```

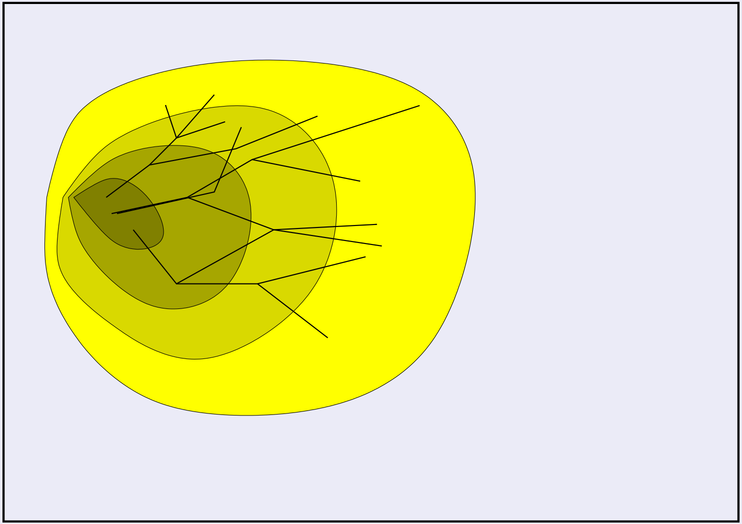
Y=reachable

Computing Reachable states: advanced

State_Set Compute_reachable() {
$Y := F := I$;
while $(F \neq \emptyset)$
$F := Image(F) \setminus Y$;
$Y := Y \cup F$;
}
return Y;
}

Y=reachable;F=frontier (new)

Computing Reachable states [cont.]



Checking of Invariant Properties: basic

bool Forward_Check_EF(**State_Set** *BAD*) {
    $Y := I;\ \ Y' := \emptyset;$
    while $(Y' \neq Y)$ **and** $(Y' \cap BAD) = \emptyset$
        $Y := Y';$
        $Y' := Y \cup Image(Y);$
    }
    if $(Y' \cap BAD) \neq \emptyset$ // counter-example
        return *true*
    else   // fixpoint reached
        return *false*
}

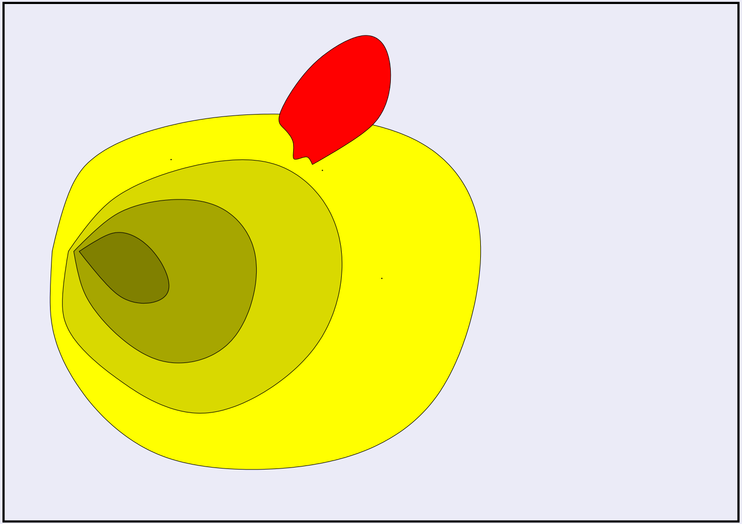
Y=reachable;

Checking of Invariant Properties: advanced

bool Forward_Check_EF(**State_Set** BAD) {
    $Y := F := I$;
    while $(F \neq \emptyset)$ **and** $(F \cap BAD) = \emptyset$
        $F := Image(F) \setminus Y$;
        $Y := Y \cup F$;
    }
    if $(F \cap BAD) \neq \emptyset$ // counter-example
        return *true*
    else   // fixpoint reached
        return *false*
}

Y=reachable;F=frontier (new)

Checking of Invariant Properties [cont.]



Checking of Invariants: Counterexamples

- if layer n intersects with the bad states, then the property is violated
- a counterexample can be reconstructed proceeding backwards
  - (i) select any state of $BAD \cap F[n]$ (we know it is satisfiable), call it $t[n]$
  - (ii) compute $Preimage(t[n])$, i.e. the states that can result in $t[n]$ in one step
  - (iii) compute $Preimage(t[n]) \cap F[n-1]$, and select one state $t[n-1]$
- iterate (i)-(iii) until the initial states are reached
- $t[0], t[1], \ldots, t[n]$ is our counterexample

Checking of Invariants: Counterexamples

- if layer n intersects with the bad states, then the property is violated
- a counterexample can be reconstructed proceeding backwards
  - (i) select any state of $BAD \cap F[n]$ (we know it is satisfiable), call it $t[n]$
  - (ii) compute $Preimage(t[n])$, i.e. the states that can result in $t[n]$ in one step
  - (iii) compute $Preimage(t[n]) \cap F[n-1]$, and select one state $t[n-1]$
- iterate (i)-(iii) until the initial states are reached
- $t[0], t[1], \ldots, t[n]$ is our counterexample

Checking of Invariants: Counterexamples

- if layer n intersects with the bad states, then the property is violated
- a counterexample can be reconstructed proceeding backwards
  (i) select any state of $BAD \cap F[n]$ (we know it is satisfiable), call it $t[n]$
  (ii) compute $Preimage(t[n])$, i.e. the states that can result in $t[n]$ in one step
  (iii) compute $Preimage(t[n]) \cap F[n-1]$, and select one state $t[n-1]$
- iterate (i)-(iii) until the initial states are reached
- $t[0], t[1], \ldots, t[n]$ is our counterexample

Checking of Invariants: Counterexamples

- if layer n intersects with the bad states, then the property is violated
- a counterexample can be reconstructed proceeding backwards
  (i) select any state of $BAD \cap F[n]$ (we know it is satisfiable), call it $t[n]$
  (ii) compute $Preimage(t[n])$, i.e. the states that can result in $t[n]$ in one step
  (iii) compute $Preimage(t[n]) \cap F[n-1]$, and select one state $t[n-1]$
- iterate (i)-(iii) until the initial states are reached
- $t[0], t[1], \ldots, t[n]$ is our counterexample

Checking of Invariants: Counterexamples

- if layer n intersects with the bad states, then the property is violated
- a counterexample can be reconstructed proceeding backwards
  (i) select any state of $BAD \cap F[n]$ (we know it is satisfiable), call it $t[n]$
  (ii) compute $Preimage(t[n])$, i.e. the states that can result in $t[n]$ in one step
  (iii) compute $Preimage(t[n]) \cap F[n-1]$, and select one state $t[n-1]$
- iterate (i)-(iii) until the initial states are reached
- $t[0], t[1], \ldots, t[n]$ is our counterexample

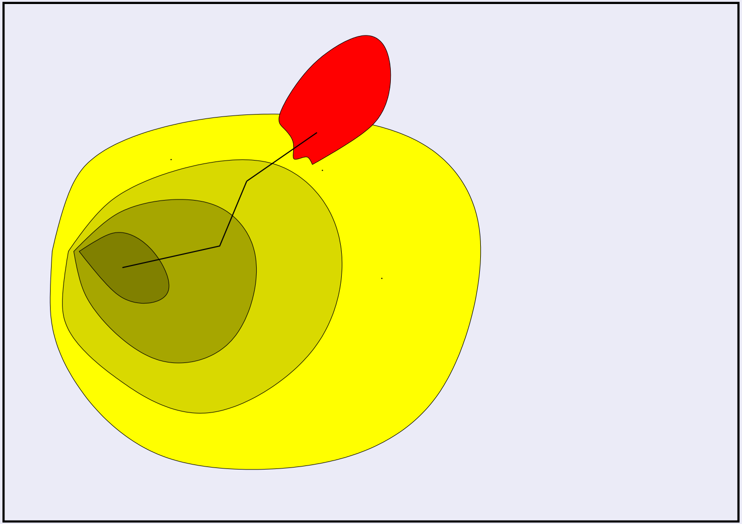
Checking of Invariants: Counterexamples

- if layer n intersects with the bad states, then the property is violated
- a counterexample can be reconstructed proceeding backwards
  (i) select any state of $BAD \cap F[n]$ (we know it is satisfiable), call it $t[n]$
  (ii) compute $Preimage(t[n])$, i.e. the states that can result in $t[n]$ in one step
  (iii) compute $Preimage(t[n]) \cap F[n-1]$, and select one state $t[n-1]$
- iterate (i)-(iii) until the initial states are reached
- $t[0], t[1], \ldots, t[n]$ is our counterexample

Checking of Invariants: Counterexamples

- if layer n intersects with the bad states, then the property is violated
- a counterexample can be reconstructed proceeding backwards
  (i) select any state of $BAD \cap F[n]$ (we know it is satisfiable), call it $t[n]$
  (ii) compute $Preimage(t[n])$, i.e. the states that can result in $t[n]$ in one step
  (iii) compute $Preimage(t[n]) \cap F[n-1]$, and select one state $t[n-1]$
- iterate (i)-(iii) until the initial states are reached
- $t[0], t[1], \ldots, t[n]$ is our counterexample

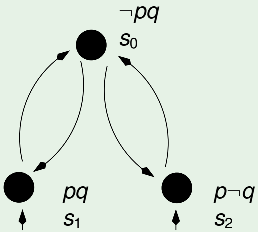
Checking of Invariants: Counterexamples [cont.]



Outline

Ex: CTL Model Checking

Consider the Kripke Model M below, and the CTL property $\varphi \stackrel{\text{def}}{=} \mathbf{AG}((p \wedge q) \rightarrow \mathbf{EG}q)$.



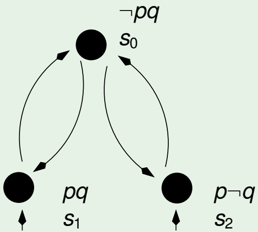
(a) Rewrite φ into an equivalent formula φ' expressed in terms of **EX**, **EG**, **EU**/**EF** only.

(b) Compute bottom-up the denotations of all subformulas of φ'. (Ex: $[p] = \{s_1, s_2\}$)

(c) As a consequence of point (b), say whether $M \models \varphi$ or not.

Ex: CTL Model Checking

Consider the Kripke Model M below, and the CTL property $\varphi \stackrel{\text{def}}{=} \mathbf{AG}((p \wedge q) \rightarrow \mathbf{EG}q)$.



(a) Rewrite φ into an equivalent formula φ' expressed in terms of **EX**, **EG**, **EU**/**EF** only.

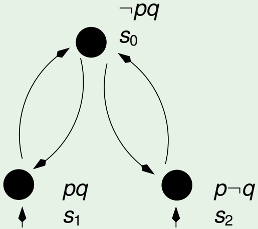
[Solution: $\varphi' = \neg\mathbf{EF}\neg((\neg p \vee \neg q) \vee \mathbf{EG}q) = \neg\mathbf{EF}((p \wedge q) \wedge \neg\mathbf{EG}q)$]

(b) Compute bottom-up the denotations of all subformulas of φ'. (Ex: $[p] = \{s_1, s_2\}$)

(c) As a consequence of point (b), say whether $M \models \varphi$ or not.

Ex: CTL Model Checking

Consider the Kripke Model M below, and the CTL property $\varphi \stackrel{\text{def}}{=} \mathbf{AG}((p \wedge q) \rightarrow \mathbf{EG}q)$.



(a) Rewrite φ into an equivalent formula φ' expressed in terms of **EX**, **EG**, **EU**/**EF** only.

[Solution: $\varphi' = \neg\mathbf{EF}\neg((\neg p \vee \neg q) \vee \mathbf{EG}q) = \neg\mathbf{EF}((p \wedge q) \wedge \neg\mathbf{EG}q)$]

(b) Compute bottom-up the denotations of all subformulas of φ'. (Ex: $[p] = \{s_1, s_2\}$)

[Solution:

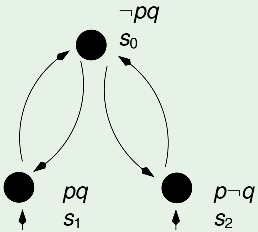
$[p]$	$=$	$\{s_1, s_2\}$	$[\neg\mathbf{EG}q]$	$=$	$\{s_2\}$
$[q]$	$=$	$\{s_0, s_1\}$	$[((p \wedge q) \wedge \neg\mathbf{EG}q)]$	$=$	$\{\}$
$[(p \wedge q)]$	$=$	$\{s_1\}$	$[\mathbf{EF}((p \wedge q) \wedge \neg\mathbf{EG}q)]$	$=$	$\{\}$
$[\mathbf{EG}q]$	$=$	$\{s_0, s_1\}$	$[\neg\mathbf{EF}((p \wedge q) \wedge \neg\mathbf{EG}q)]$	$=$	$\{s_0, s_1, s_2\}$

]

(c) As a consequence of point (b), say whether $M \models \varphi$ or not.

Ex: CTL Model Checking

Consider the Kripke Model M below, and the CTL property $\varphi \stackrel{\text{def}}{=} \mathbf{AG}((p \wedge q) \rightarrow \mathbf{EG}q)$.



(a) Rewrite φ into an equivalent formula φ' expressed in terms of **EX**, **EG**, **EU**/**EF** only.

[Solution: $\varphi' = \neg\mathbf{EF}\neg((\neg p \vee \neg q) \vee \mathbf{EG}q) = \neg\mathbf{EF}((p \wedge q) \wedge \neg\mathbf{EG}q)$]

(b) Compute bottom-up the denotations of all subformulas of φ'. (Ex: $[p] = \{s_1, s_2\}$)

[Solution:

$[p]$	$=$	$\{s_1, s_2\}$	$[\neg\mathbf{EG}q]$	$=$	$\{s_2\}$
$[q]$	$=$	$\{s_0, s_1\}$	$[((p \wedge q) \wedge \neg\mathbf{EG}q)]$	$=$	$\{\}$
$[(p \wedge q)]$	$=$	$\{s_1\}$	$[\mathbf{EF}((p \wedge q) \wedge \neg\mathbf{EG}q)]$	$=$	$\{\}$
$[\mathbf{EG}q]$	$=$	$\{s_0, s_1\}$	$[\neg\mathbf{EF}((p \wedge q) \wedge \neg\mathbf{EG}q)]$	$=$	$\{s_0, s_1, s_2\}$

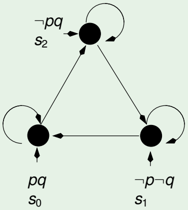
]

(c) As a consequence of point (b), say whether $M \models \varphi$ or not.

[Solution: Yes, $\{s_1, s_2\} \subseteq [\varphi']$.]

Ex: CTL Model Checking

Consider the Kripke Model M below, and the CTL property $\mathbf{AG}(\mathbf{AF}p \rightarrow \mathbf{AF}q)$.



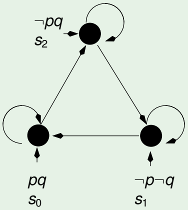
(a) Rewrite φ into an equivalent formula φ' expressed in terms of $\mathbf{EX}, \mathbf{EG}, \mathbf{EU}/\mathbf{EF}$ only.

(b) Compute bottom-up the denotations of all subformulas of φ'. (Ex: $[p] = \{s_1, s_2\}$)

(c) As a consequence of point (b), say whether $M \models \varphi$ or not.

Ex: CTL Model Checking

Consider the Kripke Model M below, and the CTL property $\mathbf{AG}(\mathbf{AF}p \rightarrow \mathbf{AF}q)$.



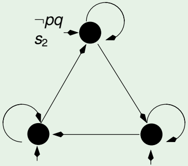
(a) Rewrite φ into an equivalent formula φ' expressed in terms of **EX**, **EG**, **EU**/**EF** only.
[Solution: $\varphi' = \mathbf{AG}(\mathbf{AF}p \rightarrow \mathbf{AF}q) = \neg\mathbf{EF}\neg(\neg\mathbf{EG}\neg p \rightarrow \neg\mathbf{EG}\neg q) = \neg\mathbf{EF}(\neg\mathbf{EG}\neg p \wedge \mathbf{EG}\neg q)$]

(b) Compute bottom-up the denotations of all subformulas of φ'. (Ex: $[p] = \{s_1, s_2\}$)

(c) As a consequence of point (b), say whether $M \models \varphi$ or not.

Ex: CTL Model Checking

Consider the Kripke Model M below, and the CTL property $\mathbf{AG}(\mathbf{AF}p \rightarrow \mathbf{AF}q)$.



(a) Rewrite φ into an equivalent formula φ' expressed in terms of **EX**, **EG**, **EU**/**EF** only.

[Solution: $\varphi' = \mathbf{AG}(\mathbf{AF}p \rightarrow \mathbf{AF}q) = \neg\mathbf{EF}\neg(\neg\mathbf{EG}\neg p \rightarrow \neg\mathbf{EG}\neg q) = \neg\mathbf{EF}(\neg\mathbf{EG}\neg p \wedge \mathbf{EG}\neg q)$]

(b) Compute bottom-up the denotations of all subformulas of φ'. (Ex: $[p] = \{s_1, s_2\}$)

[Solution:

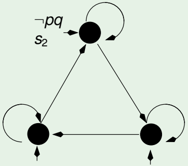
$$
\begin{array}{lcllcl}
[p] & = & \{s_0\} & [\neg q] & = & \{s_1\} \\
[\neg p] & = & \{s_1, s_2\} & [\mathbf{EG}\neg q] & = & \{s_1\} \\
[\mathbf{EG}\neg p] & = & \{s_1, s_2\} & [\neg\mathbf{EG}\neg p \wedge \mathbf{EG}\neg q] & = & \{\} \\
[\neg\mathbf{EG}\neg p] & = & \{s_0\} & [\mathbf{EF}(\neg\mathbf{EG}\neg p \wedge \mathbf{EG}\neg q)] & = & \{\} \\
[q] & = & \{s_0, s_2\} & [\neg\mathbf{EF}(\neg\mathbf{EG}\neg p \wedge \mathbf{EG}\neg q)] & = & \{s_0, s_1, s_2\}
\end{array}
$$

]

(c) As a consequence of point (b), say whether $M \models \varphi$ or not.

Ex: CTL Model Checking

Consider the Kripke Model M below, and the CTL property $\mathbf{AG}(\mathbf{AF}p \rightarrow \mathbf{AF}q)$.



(*a*) Rewrite φ into an equivalent formula φ' expressed in terms of **EX**, **EG**, **EU**/**EF** only.
[Solution: $\varphi' = \mathbf{AG}(\mathbf{AF}p \rightarrow \mathbf{AF}q) = \neg\mathbf{EF}\neg(\neg\mathbf{EG}\neg p \rightarrow \neg\mathbf{EG}\neg q) = \neg\mathbf{EF}(\neg\mathbf{EG}\neg p \wedge \mathbf{EG}\neg q)$]

(*b*) Compute bottom-up the denotations of all subformulas of φ'. (Ex: $[p] = \{s_1, s_2\}$)

[Solution:

$$
\begin{array}{lcl|lcl}
[p] & = & \{s_0\} & [\neg q] & = & \{s_1\} \\
[\neg p] & = & \{s_1, s_2\} & [\mathbf{EG}\neg q] & = & \{s_1\} \\
[\mathbf{EG}\neg p] & = & \{s_1, s_2\} & [\neg\mathbf{EG}\neg p \wedge \mathbf{EG}\neg q] & = & \{\} \\
[\neg\mathbf{EG}\neg p] & = & \{s_0\} & [\mathbf{EF}(\neg\mathbf{EG}\neg p \wedge \mathbf{EG}\neg q)] & = & \{\} \\
[q] & = & \{s_0, s_2\} & [\neg\mathbf{EF}(\neg\mathbf{EG}\neg p \wedge \mathbf{EG}\neg q)] & = & \{s_0, s_1, s_2\}
\end{array}
$$

]

(*c*) As a consequence of point (*b*), say whether $M \models \varphi$ or not.
[Solution: Yes, $\{s_0, s_1, s_2\} \subseteq [\varphi']$.]