Automated Reasoning and Formal Verification
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/arfv2026/
Teaching assistant: Gabriele Masina – gabriele.masina@unitn.it

M.S. in Computer Science, Mathematics, & Artificial Intelligence Systems
Academic year 2025-2026

last update: Wednesday 18th February, 2026, 16:34

Outline

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:

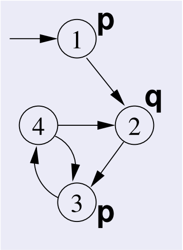
$$\mathbf{AG}(p \rightarrow \mathbf{AF}q)$$

- ...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:

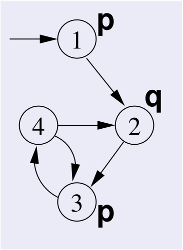
$$\mathbf{AG}(p \rightarrow \mathbf{AF}q)$$

- ...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:

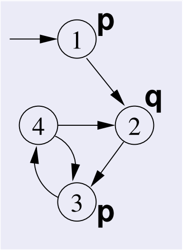
$$\mathbf{AG}(p \rightarrow \mathbf{AF}q)$$

- ...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 φ_i, computing $[\varphi_i]$ for each subformula φ_i

Example

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 φ_i, computing $[\varphi_i]$ for each subformula φ_i

Example

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 φ_i, computing $[\varphi_i]$ for each subformula φ_i

Example

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 φ_i, computing $[\varphi_i]$ for each subformula φ_i

Example

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 φ_i, computing $[\varphi_i]$ for each subformula φ_i

Example

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 φ_i, computing $[\varphi_i]$ for each subformula φ_i

Example

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 φ_i, computing $[\varphi_i]$ for each subformula φ_i

Example

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 $[\varphi_i]$:

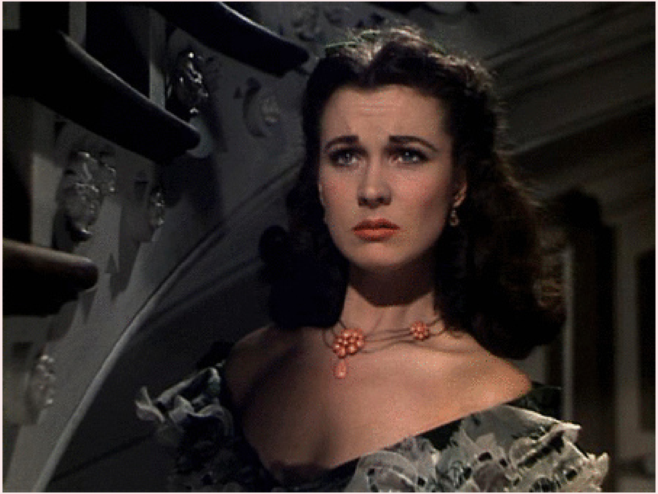
- assign Propositional atoms by labeling function
- handle Boolean operators by standard set operations
- handle temporal operators **AX**, **EX** by computing pre-images
- handle temporal operators **AG**, **EG**, **AF**, **EF**, **AU**, **EU**, by (implicitly) applying tableaux rules, until a fixpoint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

CTL Model Checking: General Idea [cont.]

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

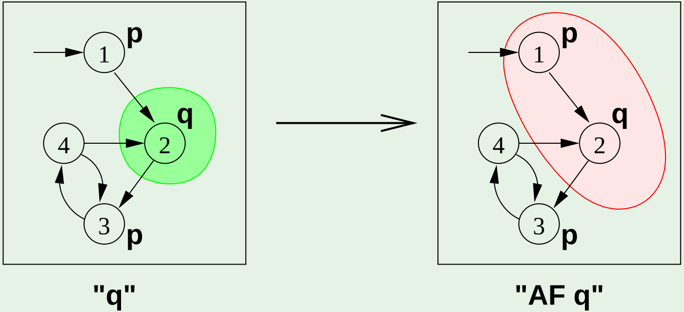
- assign Propositional atoms by labeling function
- handle Boolean operators by standard set operations
- handle temporal operators **AX**, **EX** by computing pre-images
- handle temporal operators **AG**, **EG**, **AF**, **EF**, **AU**, **EU**, by (implicitly) applying tableaux rules, until a fixpoint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

Tableaux Rules: a Quote



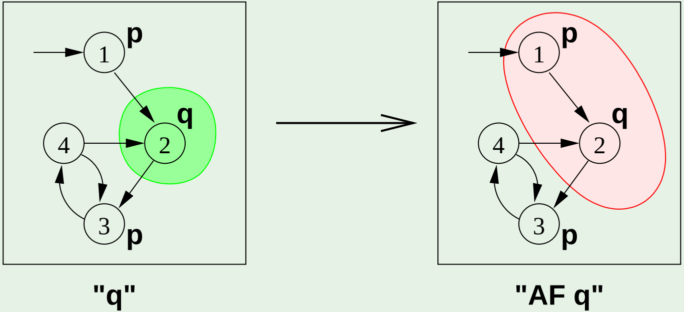
“After all... tomorrow is another day."
[Scarlett O’Hara, “Gone with the Wind”]

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



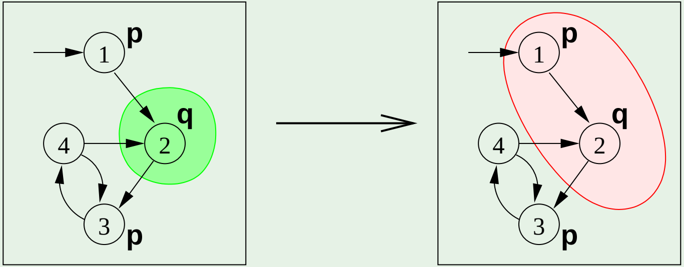
- Recall the **AF** tableau rule: $\mathbf{AF}q \leftrightarrow (q \vee \mathbf{AXAF}q)$
- Iteration: $[\mathbf{AF}q]^{(1)} = [q]$; $\quad [\mathbf{AF}q]^{(i+1)} = [q] \cup \mathbf{AX}[\mathbf{AF}q]^{(i)}$
  - $[\mathbf{AF}q]^{(1)} = [q] = \{2\}$
  - $[\mathbf{AF}q]^{(2)} = [q \vee \mathbf{AX}q] = \{2\}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

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



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

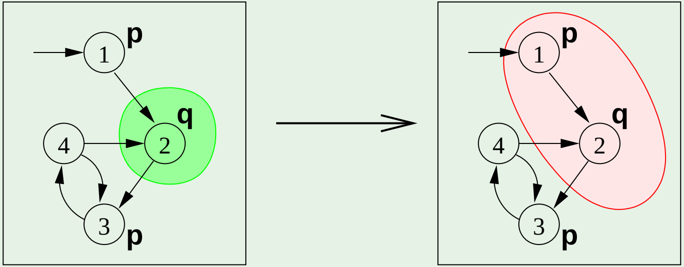


"q" "AF q"

- 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)

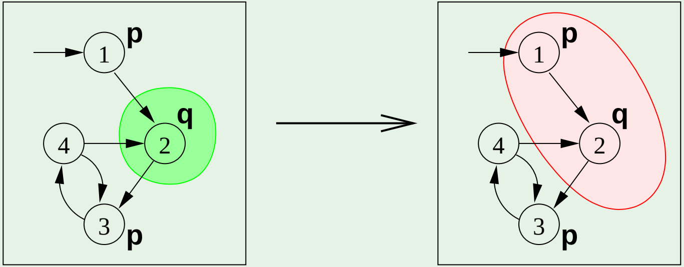
CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$



"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)

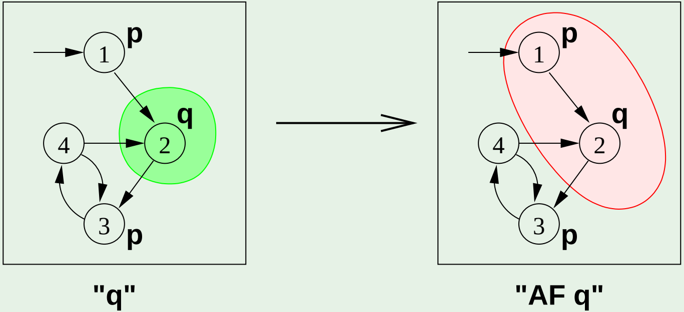


"q" "AF q"

- 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)

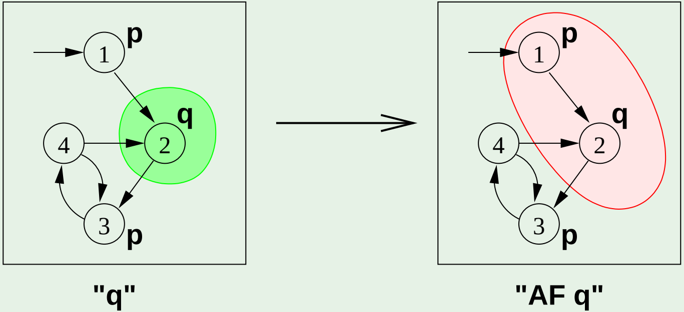
CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$



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

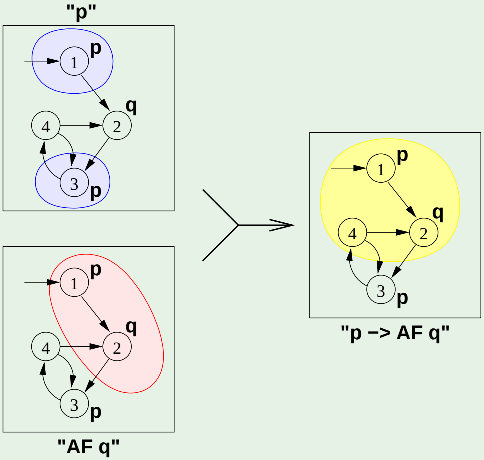
  $\Longrightarrow$ (fix point reached)

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

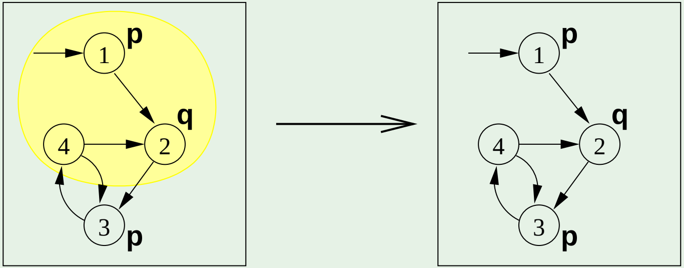


- Recall the **AF** tableau rule: $\mathbf{AF}q \leftrightarrow (q \vee \mathbf{AXAF}q)$
- Iteration: $[\mathbf{AF}q]^{(1)} = [q]; \quad [\mathbf{AF}q]^{(i+1)} = [q] \cup \mathbf{AX}[\mathbf{AF}q]^{(i)}$
  - $[\mathbf{AF}q]^{(1)} = [q] = \{2\}$
  - $[\mathbf{AF}q]^{(2)} = [q \vee \mathbf{AX}q] = \{2\}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

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



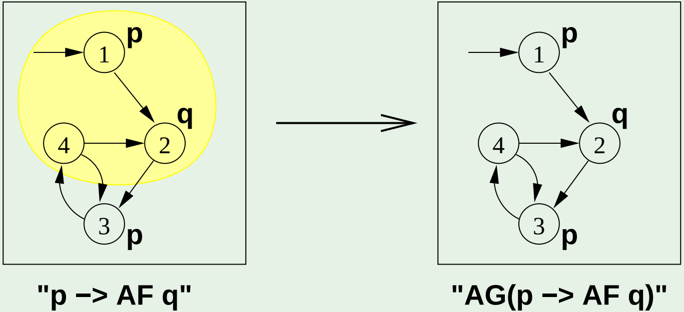
CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]



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

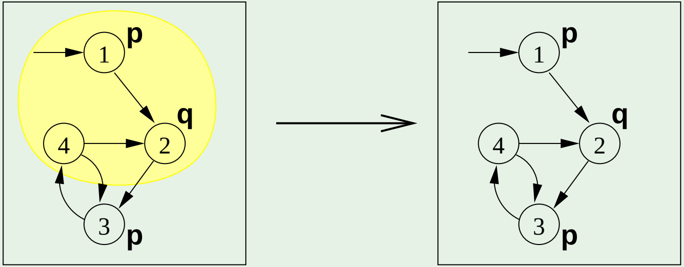
- Recall the **AG** tableau rule: $\mathbf{AG}\varphi \leftrightarrow (\varphi \wedge \mathbf{AXAG}\varphi)$
- Iteration: $[\mathbf{AG}\varphi^{(1)}] = [\varphi]$; $[\mathbf{AG}\varphi]^{(i+1)} = [\varphi] \cap \mathbf{AX}[\mathbf{AG}\varphi]^{(i)}$
  1. $[\mathbf{AG}\varphi]^{(1)} = [\varphi] = \{1,2,4\}$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

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.]



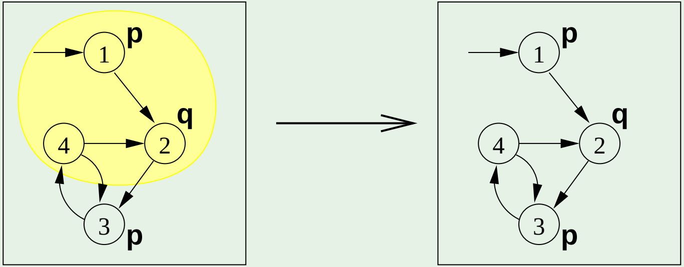
"p –> AF q"

"AG(p –> AF q)"

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

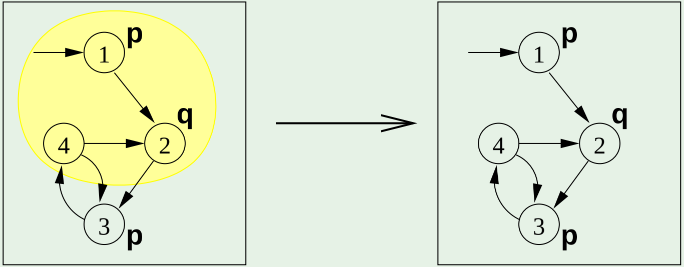


"p –> AF q" "AG(p –> AF q)"

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

  $\Longrightarrow$ (fix point reached)

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

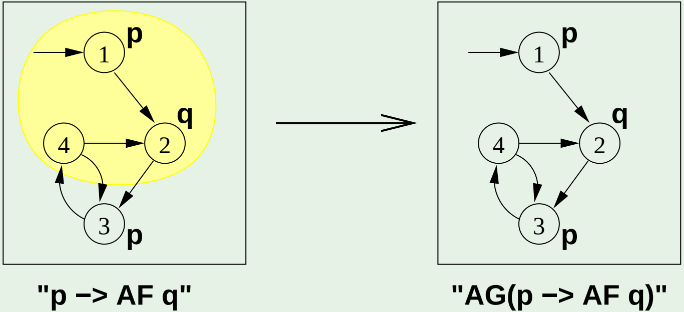


"p –> AF q" **"AG(p –> AF q)"**

- 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)

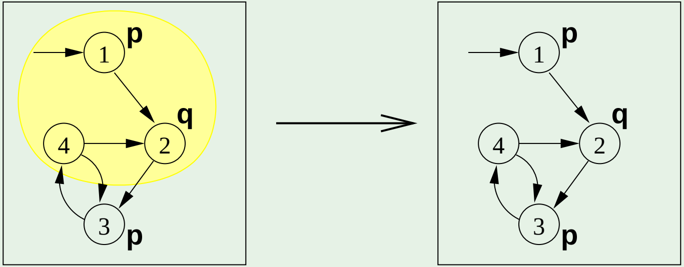
CTL Model Checking: Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [cont.]



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

  $\Longrightarrow$ (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.]

- 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: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [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: $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$ [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 $\emptyset \subseteq F(\emptyset) \subseteq F(F(\emptyset)) \dots$,
- the greatest fixed point of F is the limit of chain $S \supseteq F(S) \supseteq F(F(S)) \dots$

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 $[\varphi]$

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

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

Definition of $[\varphi]$

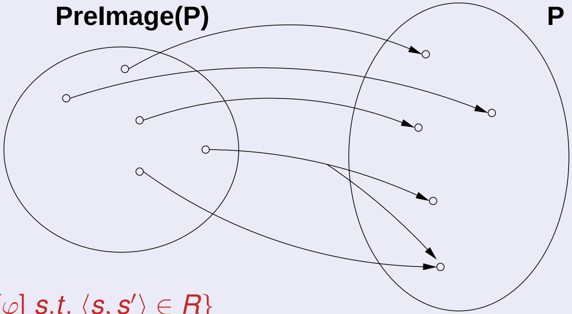
$[\varphi] := \{s \in S : M, s \models \varphi\}$

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 **EX**φ:



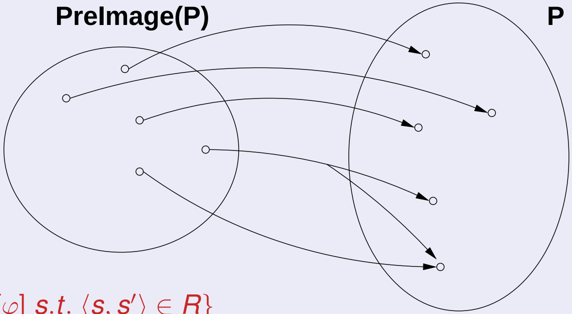
- $[\mathbf{EX}\varphi] = \{s \mid \exists s' \in [\varphi] \; s.t. \; \langle s, s' \rangle \in R\}$
- $[\mathbf{EX}\varphi]$ is said to be the Pre-image of $[\varphi]$ ($Preimage([\varphi])$)
- Key step of every CTL M.C. operation

Note

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

Case **EX**

Consider **EX**φ:



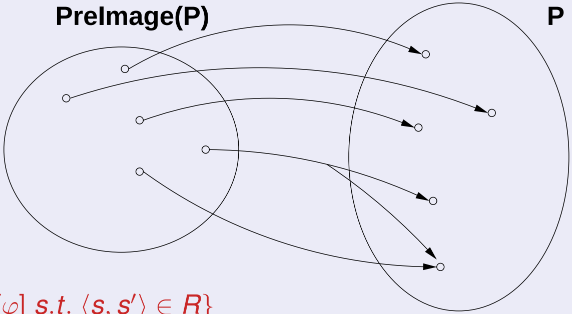
- $[\mathbf{EX}\varphi] = \{s \mid \exists s' \in [\varphi] \; s.t. \; \langle s, s' \rangle \in R\}$
- $[\mathbf{EX}\varphi]$ is said to be the Pre-image of $[\varphi]$ (*Preimage*($[\varphi]$))
- Key step of every CTL M.C. operation

Note

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

Case **EX**

Consider **EX**φ:



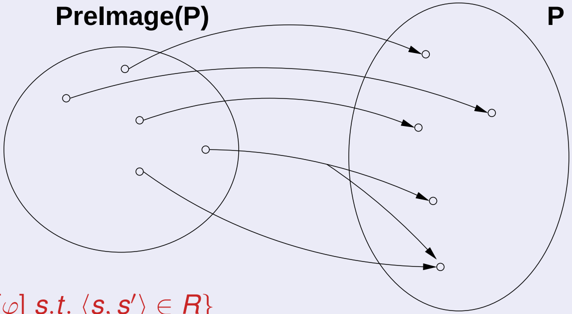
- $[\mathbf{EX}\varphi] = \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\}$
- $[\mathbf{EX}\varphi]$ is said to be the Pre-image of $[\varphi]$ ($Preimage([\varphi])$)
- Key step of every CTL M.C. operation

Note

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

Case **EX**

Consider **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 **EG**

Theorem (Clarke & Emerson)

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

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

- $\nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$: greatest fixed point of the function $F_\beta : 2^S \longrightarrow 2^S$, 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
    ... in a finite number of steps

Case **EG**

Theorem (Clarke & Emerson)

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

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

- $\nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$: 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.

Case EG

Theorem (Clarke & Emerson)

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

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

- $\nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$: 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.

Case **EG**

Theorem (Clarke & Emerson)

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

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

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

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

- F_β Monotonic: $a \subseteq a' \implies F_\beta(a) \subseteq F_\beta(a')$
  - (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.

Case EG

Theorem (Clarke & Emerson)

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

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

- $\nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$: 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.

Case **EG** [cont.]

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

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

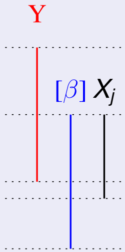
- Noticing that $X_1 = [\beta]$ and $X_{j+1} \subseteq X_j$ for every $j \geq 0$, and that $([\beta] \cap Y) \subseteq X_j \subseteq [\beta] \Longrightarrow ([\beta] \cap Y) = (X_j \cap Y)$, we can use instead the following inductive schema:
  - $X_1 \quad := [\beta]$
  - $X_{j+1} := X_j \cap Preimage(X_j)$

Case **EG** [cont.]

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

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

- 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 **EU**

Theorem (Clarke & Emerson)

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

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

- $\mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$: 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

    , in a finite number of steps.

Case EU

Theorem (Clarke & Emerson)

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

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

- $\mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$: 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.

Case **EU**

Theorem (Clarke & Emerson)

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

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

- $\mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )$: 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.

Case EU

Theorem (Clarke & Emerson)

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

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

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

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

- 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.

Case EU

Theorem (Clarke & Emerson)

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

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

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

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

- 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.

Case EU

Theorem (Clarke & Emerson)

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

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

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

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

- 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.

Case EU [cont.]

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

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

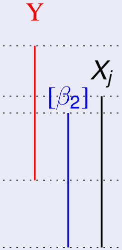
- Noticing that $X_1 = [\beta_2]$ and $X_{j+1} \supseteq X_j$ for every $j \geq 0$, and that $([\beta_2] \cup Y) \supseteq X_j \supseteq [\beta_2] \Longrightarrow ([\beta_2] \cup Y) = (X_j \cup Y)$, we can use instead the following inductive schema:
  - $X_1 \quad := [\beta_2]$
  - $X_{j+1} := X_j \cup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

Case **EU** [cont.]

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

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

- 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:
  - $X_1 := [\beta_2]$
  - $X_{j+1} := X_j \cup ([\beta_1] \cap Preimage(X_j))$



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:
  - $X_1 \quad := [\beta]$
  - $X_{j+1} := X_j \cup Preimage(X_j)$

A relevant subcase: **EF**

- $\mathbf{EF}\beta = \mathbf{E}(\top \mathbf{U} \beta)$
- $[\top] = S \Longrightarrow [\top] \cap Preimage(X_j) = Preimage(X_j)$
- 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:
  - $X_1 := [\beta]$
  - $X_{j+1} := X_j \cup \mathit{Preimage}(X_j)$

Outline

1. CTL Model Checking
   - General ideas
   - Some theoretical issues
   - Algorithms
   - Some examples
2. A relevant subcase: Invariant Checking
3. CTL Model Checking with Fair Kripke Models
   - Fairness & Fair Kripke Models
   - Fair CTL Model Checking
   - SCC-Based Approach
   - Emerson-Lei Algorithm
4. Exercises

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 [EXβ]

state_set PreImage(**state_set** [β]) {
    $X := \{\}$;
    for each $s \in S$ **do**
        for each s' $s.t.$ $s' \in [\beta]$ and $\langle s, s' \rangle \in R$ **do**
            $X := X \cup \{s\}$;
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;
}

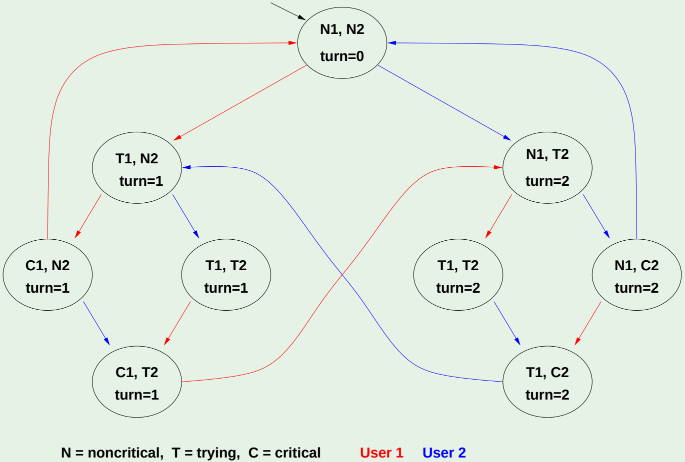
A relevant subcase: Check_EF

Compute [EFβ]

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

Outline

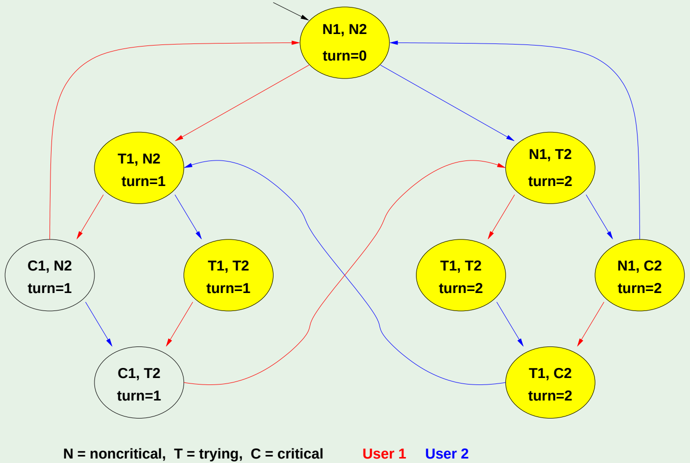
Example 1: fairness



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

Example 1: fairness

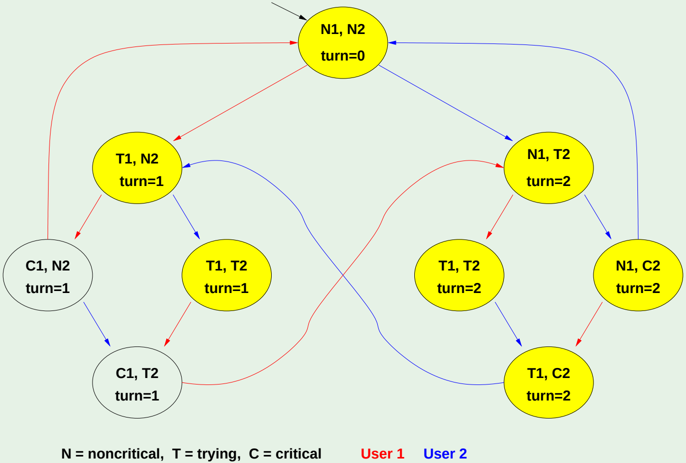
$[\neg C_1]$



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

Example 1: fairness

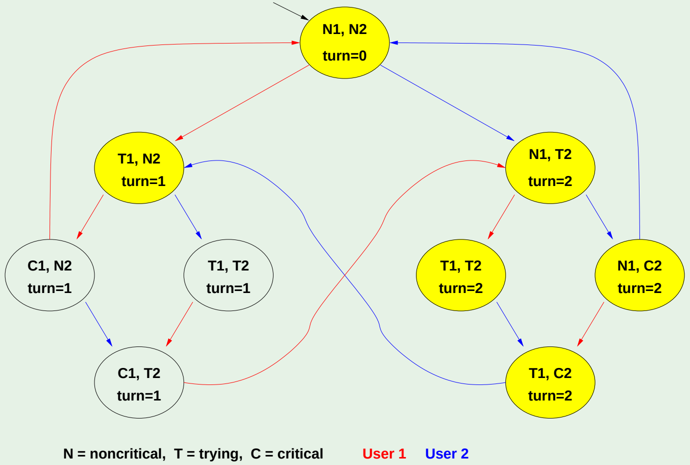
$[\mathbf{EG}\neg C_1]$, step 0:



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

Example 1: fairness

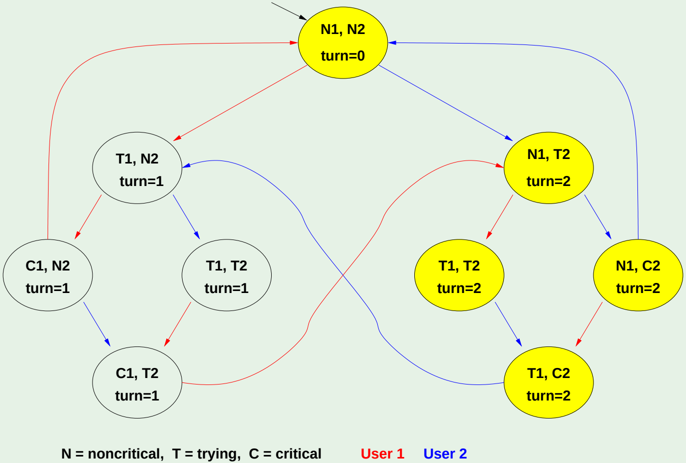
$[\mathbf{EG}\neg C_1]$, step 1:



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

Example 1: fairness

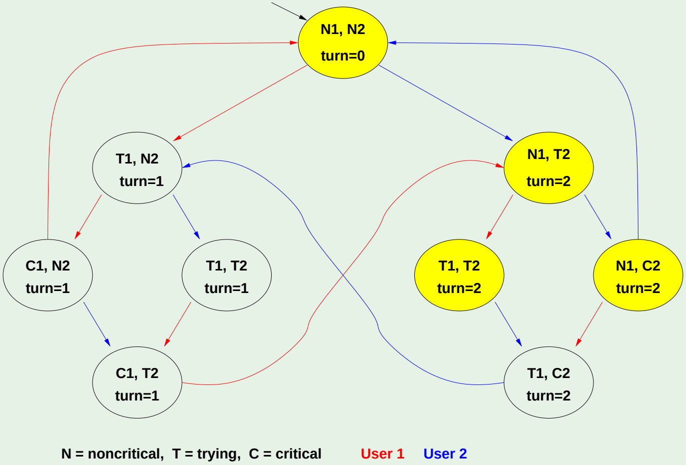
$[\mathbf{EG}\neg C_1]$, step 2:



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

Example 1: fairness

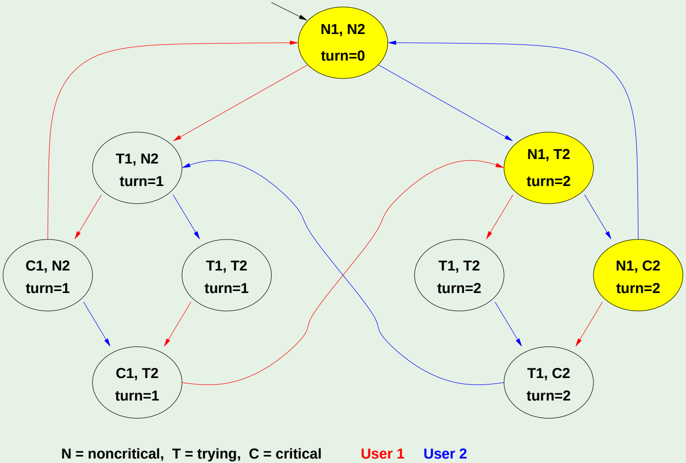
[$\mathbf{EG}\neg C_1$], step 3:



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

Example 1: fairness

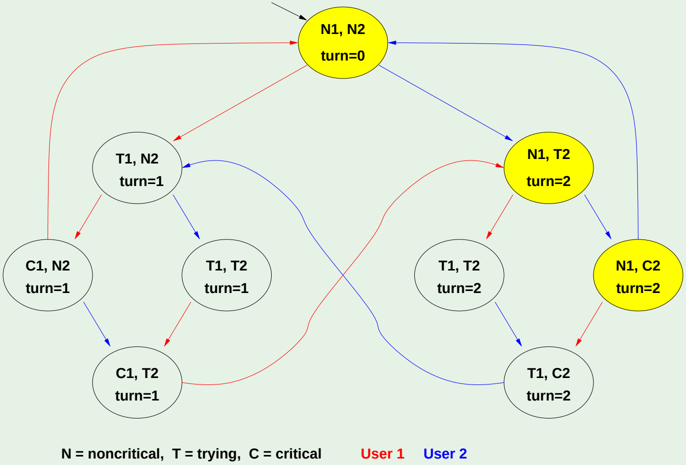
[**EG**$\neg C_1$], step 4:



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

Example 1: fairness

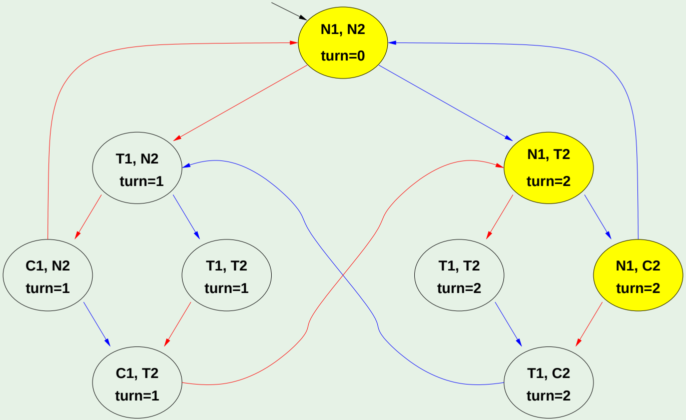
[**EG**$\neg C_1$], FIXPOINT!



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

Example 1: fairness

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

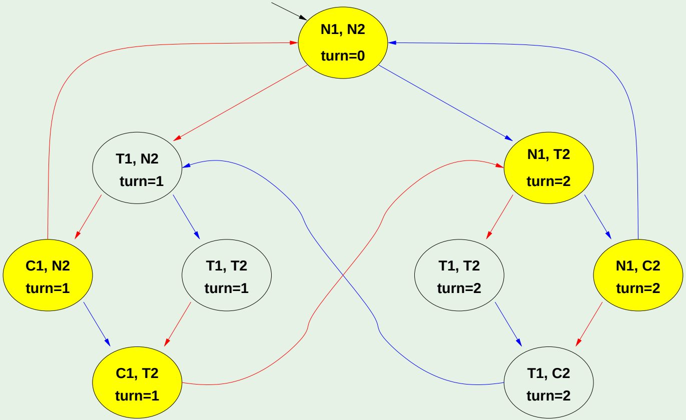


N = noncritical, T = trying, C = critical User 1 User 2

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

Example 1: fairness

$[\mathbf{EFEG}\neg C_1]$, STEP 1

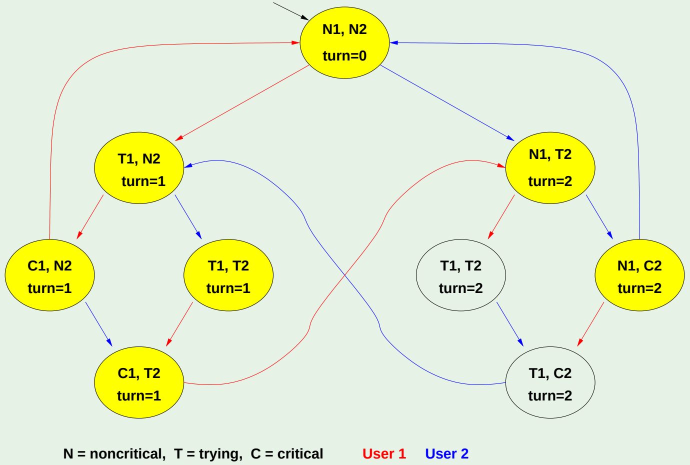


N = noncritical, T = trying, C = critical User 1 User 2

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

Example 1: fairness

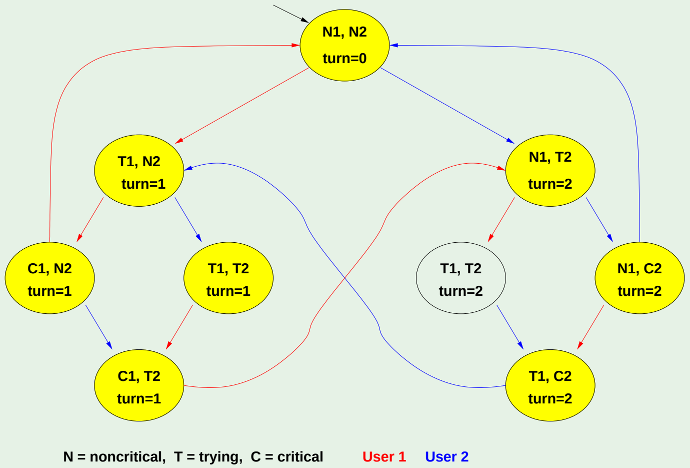
[EFEG$\neg C_1$], STEP 2



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

Example 1: fairness

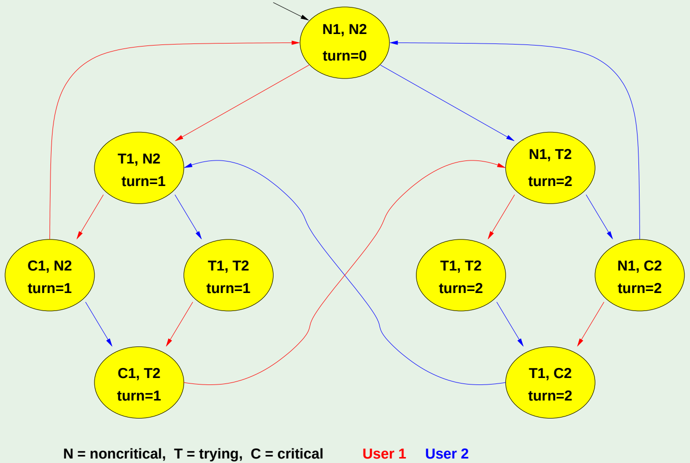
[**EFEG**$\neg C_1$], STEP 3



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

Example 1: fairness

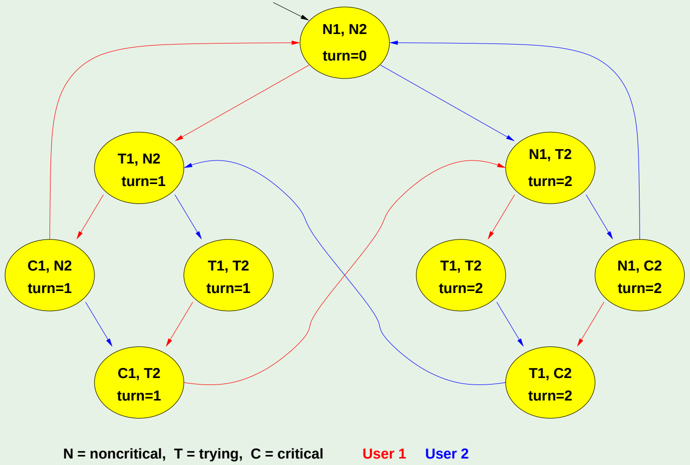
[**EFEG**$\neg C_1$], STEP 4



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

Example 1: fairness

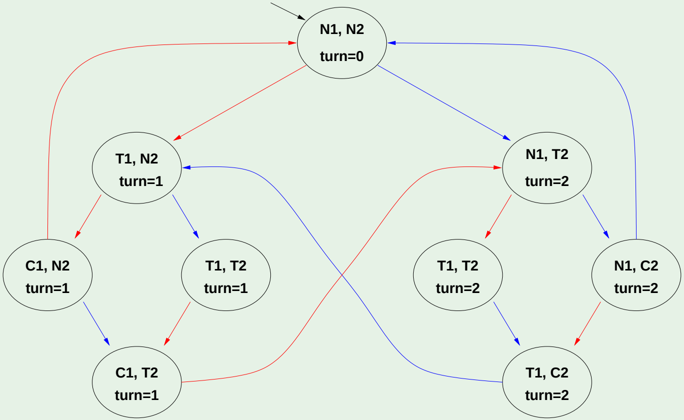
[**EFEG**$\neg C_1$], FIXPOINT!



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

Example 1: fairness

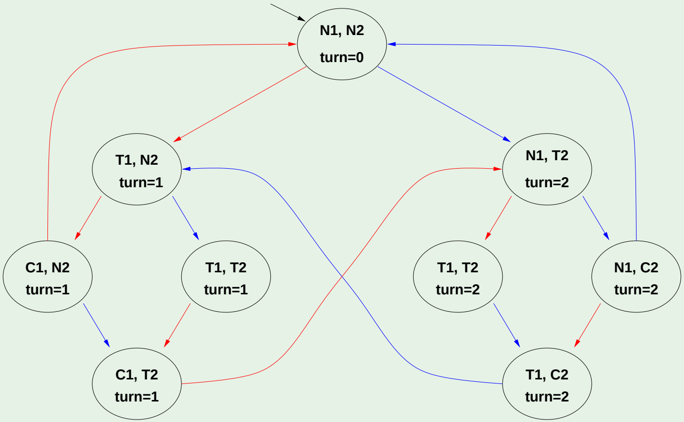
$[\neg \mathbf{EFEG} \neg C_1]$



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

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

Example 2: liveness

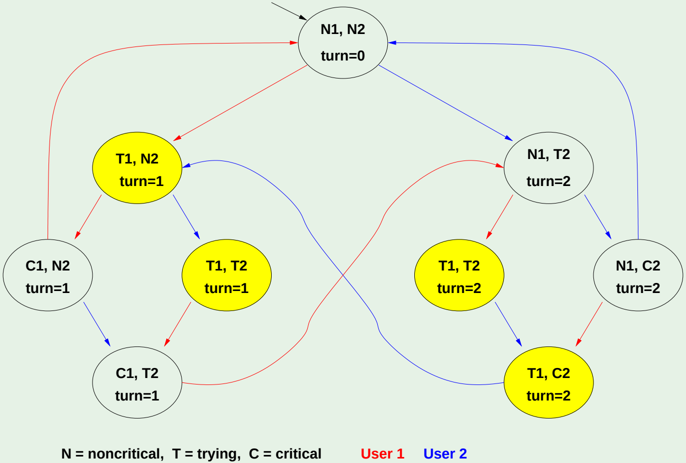


N = noncritical, T = trying, C = critical **User 1** **User 2**

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

Example 2: liveness

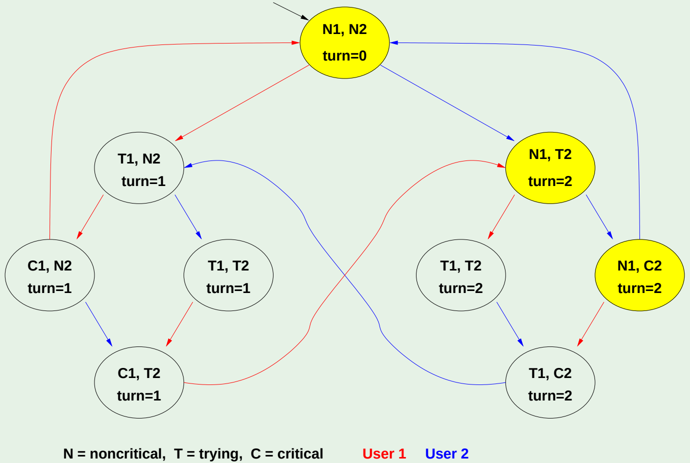
$[T_1]$:



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

Example 2: liveness

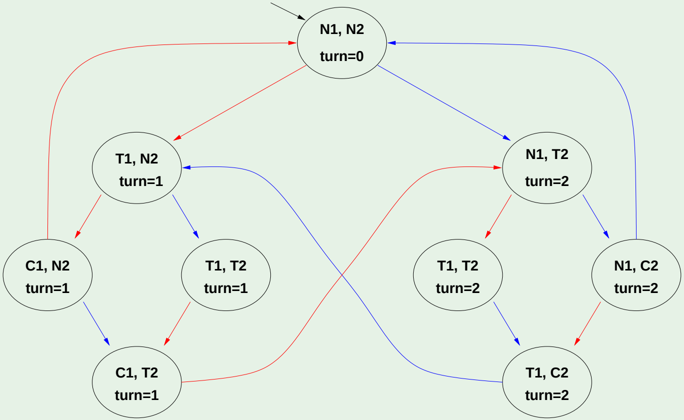
$[\mathbf{EG}\neg C_1]$, STEPS 0-4: (see previous example)



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

Example 2: liveness

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

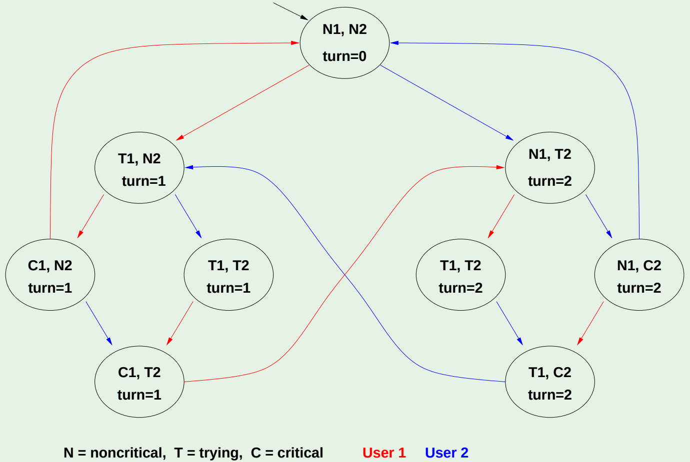


N = noncritical, T = trying, C = critical **User 1** **User 2**

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

Example 2: liveness

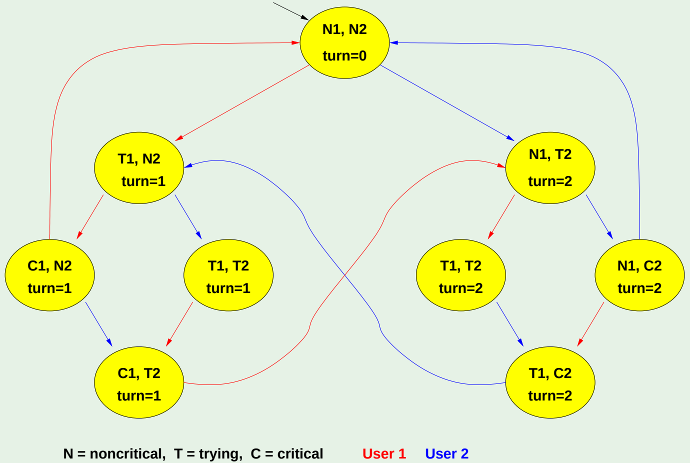
$[\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)]$:



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

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**¬*bad*)
- Checking invariants is the negation of a reachability problem:
  - is there a reachable state that is also a bad state? (**AG**¬*bad* = ¬**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

- 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:

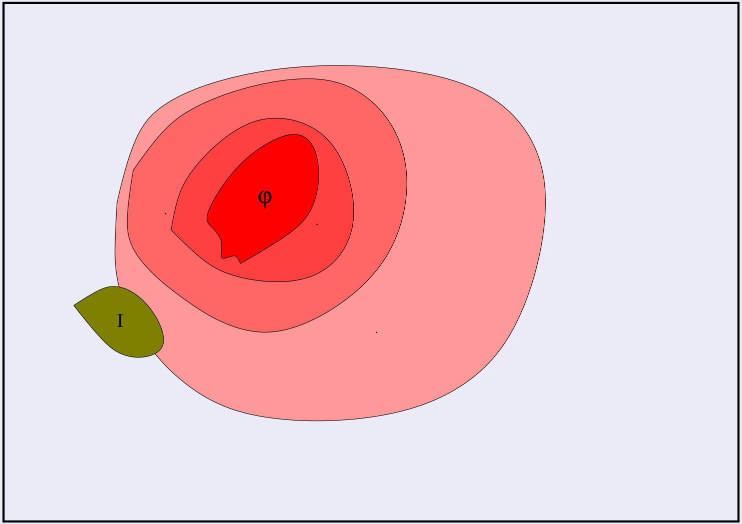
$$Y' := Y \cup PreImage(Y)$$

  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:

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

- 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;
}
```

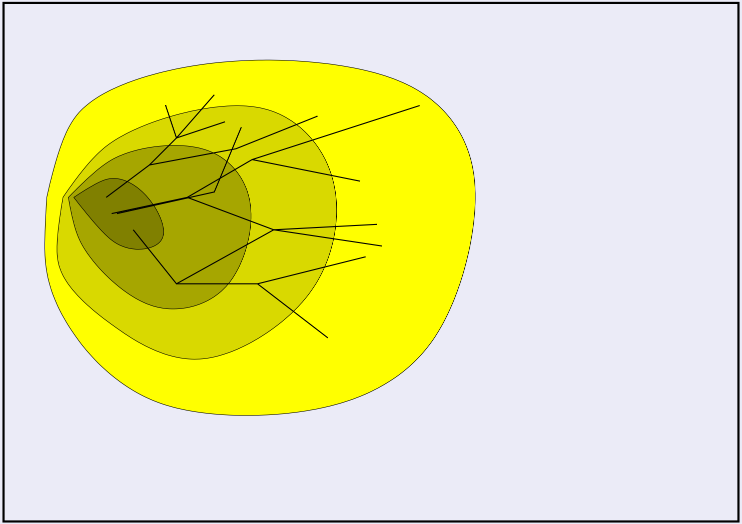
Y=reachable

Computing Reachable states: advanced

State_Set Compute_reachable() {
$\quad Y := F := I;$
$\quad$ **while** $(F \neq \emptyset)$
$\quad\quad F := Image(F) \setminus Y;$
$\quad\quad Y := Y \cup F;$
$\quad$ }
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' := ∅;
    while (Y' ≠ Y) and (Y' ∩ BAD) = ∅
        Y := Y';
        Y' := Y ∪ Image(Y);
    }
    if (Y' ∩ BAD) ≠ ∅    // counter-example
        return true
    else                 // fixpoint reached
        return false
}
```

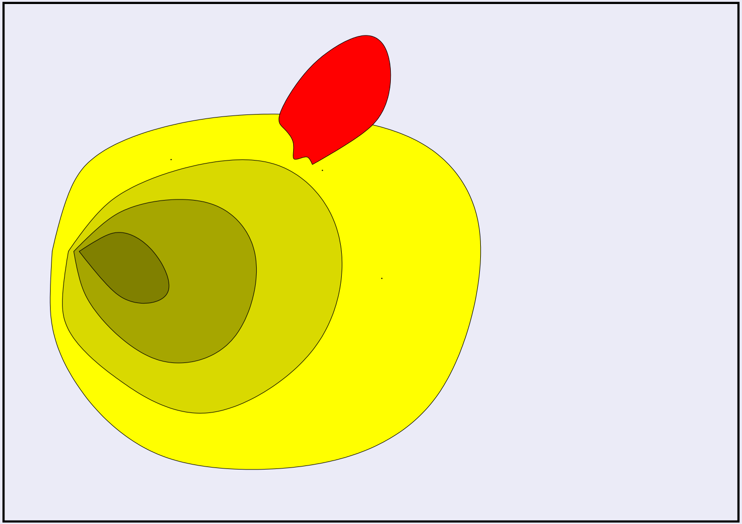
Y=reachable;

Checking of Invariant Properties: advanced

```
bool Forward_Check_EF(State_Set BAD) {
    Y := F := I;
    while (F ≠ ∅) and (F ∩ BAD) = ∅
        F := Image(F) \ Y;
        Y := Y ∪ F;
    }
    if (F ∩ BAD) ≠ ∅      // 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
  1. (i) select any state of $BAD \cap F[n]$ (we know it is satisfiable), call it $t[n]$
  2. (ii) compute $Preimage(t[n])$, i.e. the states that can result in $t[n]$ in one step
  3. (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

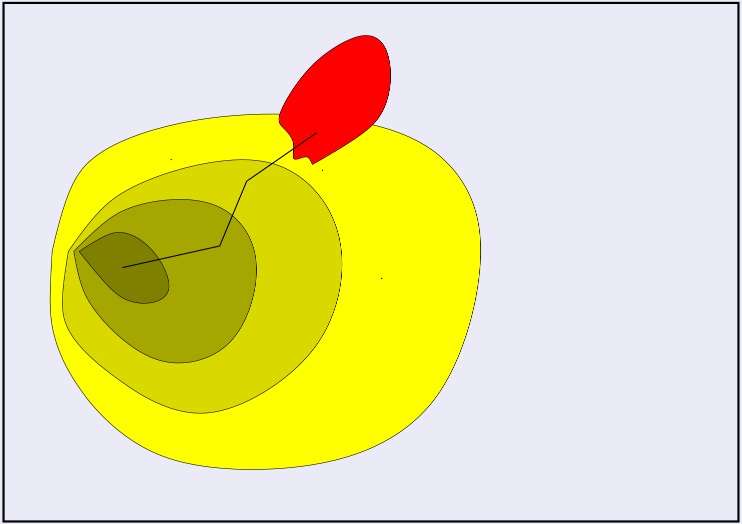
Checking of Invariants: Counterexamples

- 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

Outline

The Need for Fairness Conditions: Intuition

Consider a public restroom. A standard access policy is “first come first served” (e.g., a queue-based protocol).

- Does this policy guarantee that everybody entering the queue will eventually access the restroom?
  - No: in principle, somebody might remain in the restroom forever, hindering the access to everybody else
  - In practice, it is considered reasonable to assume that everybody exits the restroom after a finite amount of time

$\Longrightarrow$ It is reasonable enough to assume the protocol suitable under the condition that each user is infinitely often outside the restroom

- Such a condition is called fairness condition

The Need for Fairness Conditions: Intuition

Consider a public restroom. A standard access policy is “first come first served” (e.g., a queue-based protocol).

- Does this policy guarantee that everybody entering the queue will eventually access the restroom?
  - **No**: in principle, somebody might remain in the restroom forever, hindering the access to everybody else
  - In practice, it is considered reasonable to assume that everybody exits the restroom after a finite amount of time

$\Longrightarrow$ It is reasonable enough to assume the protocol suitable under the condition that each user is infinitely often outside the restroom

- Such a condition is called fairness condition

The Need for Fairness Conditions: Intuition

Consider a public restroom. A standard access policy is “first come first served” (e.g., a queue-based protocol).

- Does this policy guarantee that everybody entering the queue will eventually access the restroom?
  - **No**: in principle, somebody might remain in the restroom forever, hindering the access to everybody else
  - In practice, it is considered reasonable to assume that everybody exits the restroom after a finite amount of time

$\Longrightarrow$ It is reasonable enough to assume the protocol suitable under the condition that each user is infinitely often outside the restroom

- Such a condition is called fairness condition

The Need for Fairness Conditions: Intuition

Consider a public restroom. A standard access policy is “first come first served” (e.g., a queue-based protocol).

- Does this policy guarantee that everybody entering the queue will eventually access the restroom?
  - **No**: in principle, somebody might remain in the restroom forever, hindering the access to everybody else
  - In practice, it is considered reasonable to assume that everybody exits the restroom after a finite amount of time

$\Longrightarrow$ It is reasonable enough to assume the protocol suitable under the condition that each user is infinitely often outside the restroom

- Such a condition is called fairness condition

The Need for Fairness Conditions: Intuition

Consider a public restroom. A standard access policy is “first come first served” (e.g., a queue-based protocol).

- Does this policy guarantee that everybody entering the queue will eventually access the restroom?
  - **No**: in principle, somebody might remain in the restroom forever, hindering the access to everybody else
  - In practice, it is considered reasonable to assume that everybody exits the restroom after a finite amount of time

$\Longrightarrow$ It is reasonable enough to assume the protocol suitable under the condition that each user is infinitely often outside the restroom

- Such a condition is called fairness condition

The Need for Fairness Conditions: Intuition

Consider a public restroom. A standard access policy is “first come first served” (e.g., a queue-based protocol).

- Does this policy guarantee that everybody entering the queue will eventually access the restroom?
  - **No**: in principle, somebody might remain in the restroom forever, hindering the access to everybody else
  - In practice, it is considered reasonable to assume that everybody exits the restroom after a finite amount of time

$\Longrightarrow$ It is reasonable enough to assume the protocol suitable under the condition that each user is infinitely often outside the restroom

- Such a condition is called fairness condition

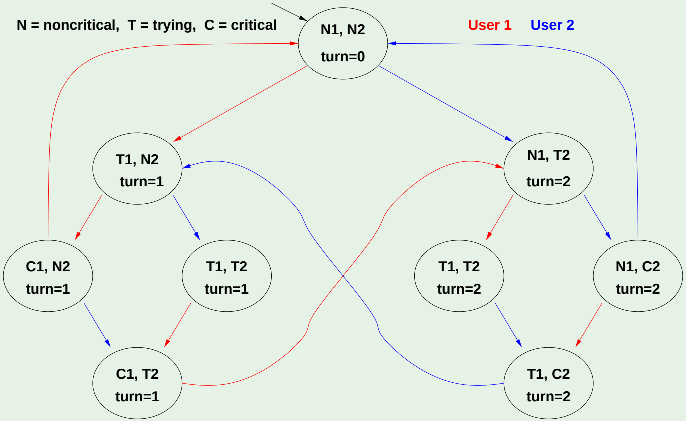
The Need for Fairness Conditions: An Example

- Consider a variant of the mutual exclusion in which one process can stay permanently in the critical zone
- Do $M \models \mathbf{G}(T_1 \rightarrow \mathbf{F}C_1)$, $M \models \mathbf{G}(T_2 \rightarrow \mathbf{F}C_2)$ still hold?

The Need for Fairness Conditions: An Example

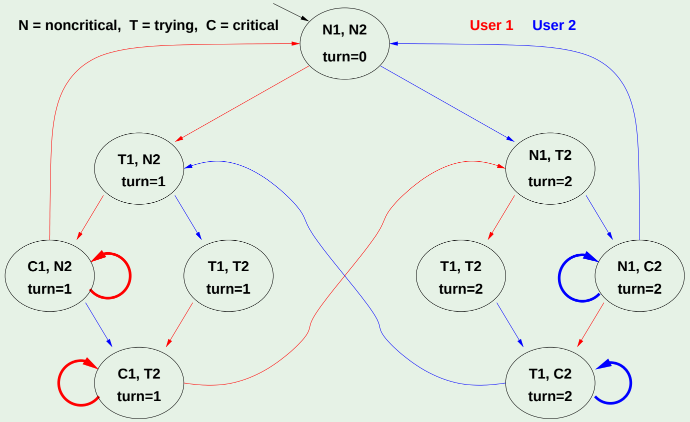
- Consider a variant of the mutual exclusion in which one process can stay permanently in the critical zone
- Do $M \models \mathbf{G}(T_1 \rightarrow \mathbf{F}C_1)$, $M \models \mathbf{G}(T_2 \rightarrow \mathbf{F}C_2)$ still hold?

The Need for Fairness Conditions: An Example [cont.]



$M \models \mathbf{G}(T_1 \rightarrow \mathbf{F}C_1)$ $M \models \mathbf{G}(T_2 \rightarrow \mathbf{F}C_2)$

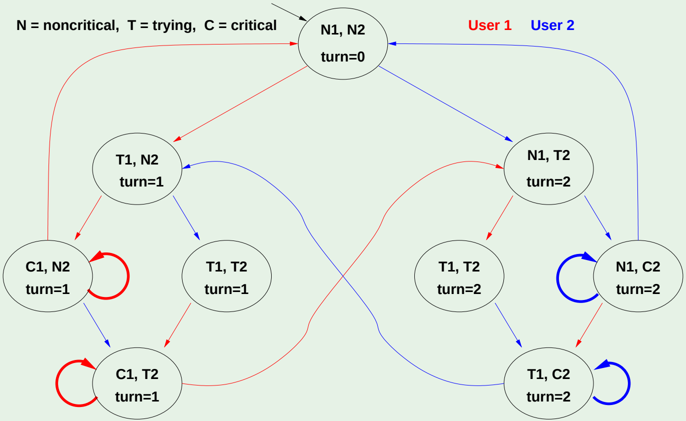
The need for fairness conditions: an example [cont.]



$M \models \mathbf{G}(T_1 \rightarrow \mathbf{F}C_1)$?

$M \models \mathbf{G}(T_2 \rightarrow \mathbf{F}C_2)$?

The need for fairness conditions: an example [cont.]



$\mathbf{G}(T_1 \rightarrow \mathbf{F}C_1)$? $\mathbf{G}(T_2 \rightarrow \mathbf{F}C_2)$?

NO: E.g., it can cycle forever in $\{C_1, T_2, turn = 1\}$

$\Longrightarrow$ Unfair protocol: one process might never be served

Fairness Conditions

- It is desirable that certain (typically Boolean) conditions φ's hold infinitely often: **GF**φ
- **GF**φ is called fairness condition
- Intuitively, fairness conditions are used to eliminate behaviours in which a certain condition φ never holds:
  GFφ: "it is never reached a state from which φ is forever false"
- Example: it is not desirable that, once a process is in the critical section, it never exits: **GF**$\neg C_1$
- A fair condition φ_i can be represented also by the set f_i of states where φ_i holds ($f_i := \{s : \pi, s \models \varphi_i,\ \textit{for each}\ \pi \in M\}$)

Fairness Conditions

- It is desirable that certain (typically Boolean) conditions φ's hold infinitely often: **GF**φ
- **GF**φ is called fairness condition
- Intuitively, fairness conditions are used to eliminate behaviours in which a certain condition φ never holds:
  GFφ: "it is never reached a state from which φ is forever false"
- Example: it is not desirable that, once a process is in the critical section, it never exits: **GF**$\neg C_1$
- A fair condition φ_i can be represented also by the set f_i of states where φ_i holds ($f_i := \{s : \pi, s \models \varphi_i,$ *for each* $\pi \in M\}$)

Fairness Conditions

- It is desirable that certain (typically Boolean) conditions φ's hold infinitely often: **GF**φ
- **GF**φ is called fairness condition
- Intuitively, fairness conditions are used to eliminate behaviours in which a certain condition φ never holds:
  GFφ: "it is never reached a state from which φ is forever false"
- Example: it is not desirable that, once a process is in the critical section, it never exits: **GF**$\neg C_1$
- A fair condition φ_i can be represented also by the set f_i of states where φ_i holds ($f_i := \{s : \pi, s \models \varphi_i,\ \textit{for each}\ \pi \in M\}$)

Fairness Conditions

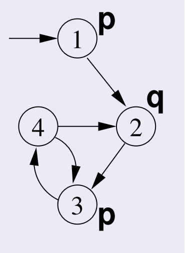
- It is desirable that certain (typically Boolean) conditions φ's hold infinitely often: **GF**φ
- **GF**φ is called fairness condition
- Intuitively, fairness conditions are used to eliminate behaviours in which a certain condition φ never holds:
  GFφ: "it is never reached a state from which φ is forever false"
- Example: it is not desirable that, once a process is in the critical section, it never exits: **GF**$\neg C_1$
- A fair condition φ_i can be represented also by the set f_i of states where φ_i holds ($f_i := \{s : \pi, s \models \varphi_i, \textit{ for each } \pi \in M\}$)

Fairness Conditions

- It is desirable that certain (typically Boolean) conditions φ's hold infinitely often: $\mathbf{GF}\varphi$
- $\mathbf{GF}\varphi$ is called fairness condition
- Intuitively, fairness conditions are used to eliminate behaviours in which a certain condition φ never holds:
  $\mathbf{GF}\varphi$: "it is never reached a state from which φ is forever false"
- Example: it is not desirable that, once a process is in the critical section, it never exits: $\mathbf{GF}\neg C_1$
- A fair condition φ_i can be represented also by the set f_i of states where φ_i holds
  $(f_i := \{s : \pi, s \models \varphi_i, \ \textit{for each}\ \pi \in M\})$

Fair Kripke models

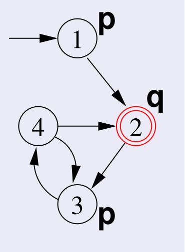
- A Fair Kripke model $M_F := \langle S, R, I, AP, L, F \rangle$ consists of:
  - a set of states S;
  - a set of initial states $I \subseteq S$;
  - a set of transitions $R \subseteq S \times S$;
  - a set of atomic propositions AP;
  - a labeling function $L : S \longmapsto 2^{AP}$;
  - a set of fairness conditions $F = \{f_1, \ldots, f_n\}$, with $f_i \subseteq S$.



- E.g., $\{\{2\}\} := \{\{s : L(s) = \{q\}\}\} = \{\mathbf{GF}q\}$ is the set of fairness conditions of the Kripke model above
- Fair path π: at least one state for each f_i occurs infinitely often in π (φ_i holds infinitely often in π: $\pi \models \mathbf{GF}\varphi_i$)
  - E.g., every path visiting infinitely often state 2 is a fair path.
- Fair state: a state through which at least one fair path passes
  - E.g., all states 1,2,3,4 are fair states
- Note: fair state $\neq$ state belonging to a fairness condition

Fair Kripke models

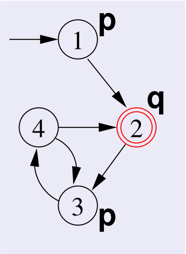
- A Fair Kripke model $M_F := \langle S, R, I, AP, L, F \rangle$ consists of:
  - a set of states S;
  - a set of initial states $I \subseteq S$;
  - a set of transitions $R \subseteq S \times S$;
  - a set of atomic propositions AP;
  - a labeling function $L : S \longmapsto 2^{AP}$;
  - a set of fairness conditions $F = \{f_1, \ldots, f_n\}$, with $f_i \subseteq S$.



- E.g., $\{\{2\}\} := \{\{s : L(s) = \{q\}\}\} = \{\mathbf{GF}q\}$ is the set of fairness conditions of the Kripke model above
- Fair path π: at least one state for each f_i occurs infinitely often in π (φ_i holds infinitely often in π: $\pi \models \mathbf{GF}\varphi_i$)
  - E.g., every path visiting infinitely often state 2 is a fair path.
- Fair state: a state through which at least one fair path passes
  - E.g., all states 1,2,3,4 are fair states
- Note: fair state $\neq$ state belonging to a fairness condition

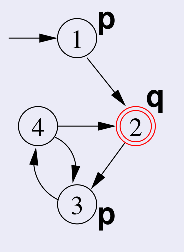
Fair Kripke models

- A Fair Kripke model $M_F := \langle S, R, I, AP, L, F \rangle$ consists of:
  - a set of states S;
  - a set of initial states $I \subseteq S$;
  - a set of transitions $R \subseteq S \times S$;
  - a set of atomic propositions AP;
  - a labeling function $L : S \longmapsto 2^{AP}$;
  - a set of fairness conditions $F = \{f_1, \ldots, f_n\}$, with $f_i \subseteq S$.



- E.g., $\{\{2\}\} := \{\{s : L(s) = \{q\}\}\} = \{\mathbf{GF}q\}$ is the set of fairness conditions of the Kripke model above
- Fair path π: at least one state for each f_i occurs infinitely often in π (φ_i holds infinitely often in π: $\pi \models \mathbf{GF}\varphi_i$)
  - E.g., every path visiting infinitely often state 2 is a fair path.
- Fair state: a state through which at least one fair path passes
  - E.g., all states 1,2,3,4 are fair states
- Note: fair state $\neq$ state belonging to a fairness condition

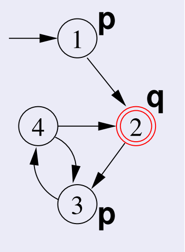
Fair Kripke models



- A Fair Kripke model $M_F := \langle S, R, I, AP, L, F \rangle$ consists of:
  - a set of states S;
  - a set of initial states $I \subseteq S$;
  - a set of transitions $R \subseteq S \times S$;
  - a set of atomic propositions AP;
  - a labeling function $L : S \longmapsto 2^{AP}$;
  - a set of fairness conditions $F = \{f_1, \ldots, f_n\}$, with $f_i \subseteq S$.
- E.g., $\{\{2\}\} := \{\{s : L(s) = \{q\}\}\} = \{\mathbf{GF}q\}$ is the set of fairness conditions of the Kripke model above
- Fair path π: at least one state for each f_i occurs infinitely often in π (φ_i holds infinitely often in π: $\pi \models \mathbf{GF}\varphi_i$)
  - E.g., every path visiting infinitely often state 2 is a fair path.
- Fair state: a state through which at least one fair path passes
  - E.g., all states 1,2,3,4 are fair states
- Note: fair state $\neq$ state belonging to a fairness condition

Fair Kripke models

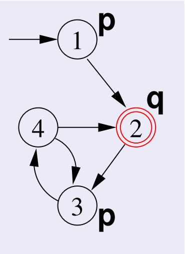
- A Fair Kripke model $M_F := \langle S, R, I, AP, L, F \rangle$ consists of:
  - a set of states S;
  - a set of initial states $I \subseteq S$;
  - a set of transitions $R \subseteq S \times S$;
  - a set of atomic propositions AP;
  - a labeling function $L : S \longmapsto 2^{AP}$;
  - a set of fairness conditions $F = \{f_1, \ldots, f_n\}$, with $f_i \subseteq S$.



- E.g., $\{\{2\}\} := \{\{s : L(s) = \{q\}\}\} = \{\mathbf{GF}q\}$ is the set of fairness conditions of the Kripke model above
- Fair path π: at least one state for each f_i occurs infinitely often in π (φ_i holds infinitely often in π: $\pi \models \mathbf{GF}\varphi_i$)
  - E.g., every path visiting infinitely often state 2 is a fair path.
- Fair state: a state through which at least one fair path passes
  - E.g., all states 1,2,3,4 are fair states
- Note: fair state $\neq$ state belonging to a fairness condition

Fair Kripke models

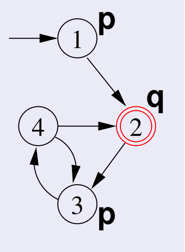
- A Fair Kripke model $M_F := \langle S, R, I, AP, L, F \rangle$ consists of:
  - a set of states S;
  - a set of initial states $I \subseteq S$;
  - a set of transitions $R \subseteq S \times S$;
  - a set of atomic propositions AP;
  - a labeling function $L : S \longmapsto 2^{AP}$;
  - a set of fairness conditions $F = \{f_1, \ldots, f_n\}$, with $f_i \subseteq S$.



- E.g., $\{\{2\}\} := \{\{s : L(s) = \{q\}\}\} = \{\mathbf{GF}q\}$ is the set of fairness conditions of the Kripke model above
- Fair path π: at least one state for each f_i occurs infinitely often in π (φ_i holds infinitely often in π: $\pi \models \mathbf{GF}\varphi_i$)
  - E.g., every path visiting infinitely often state 2 is a fair path.
- Fair state: a state through which at least one fair path passes
  - E.g., all states 1,2,3,4 are fair states
- Note: fair state $\neq$ state belonging to a fairness condition

Fair Kripke models

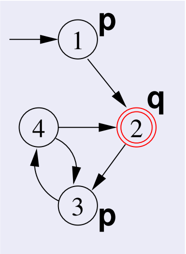
- A Fair Kripke model $M_F := \langle S, R, I, AP, L, F\rangle$ consists of:
  - a set of states S;
  - a set of initial states $I \subseteq S$;
  - a set of transitions $R \subseteq S \times S$;
  - a set of atomic propositions AP;
  - a labeling function $L : S \longmapsto 2^{AP}$;
  - a set of fairness conditions $F = \{f_1, \ldots, f_n\}$, with $f_i \subseteq S$.



- E.g., $\{\{2\}\} := \{\{s : L(s) = \{q\}\}\} = \{\mathbf{GF}q\}$ is the set of fairness conditions of the Kripke model above
- Fair path π: at least one state for each f_i occurs infinitely often in π (φ_i holds infinitely often in π: $\pi \models \mathbf{GF}\varphi_i$)
  - E.g., every path visiting infinitely often state 2 is a fair path.
- Fair state: a state through which at least one fair path passes
  - E.g., all states 1,2,3,4 are fair states
- Note: fair state $\neq$ state belonging to a fairness condition

Fair Kripke models

- A Fair Kripke model $M_F := \langle S, R, I, AP, L, F \rangle$ consists of:
  - a set of states S;
  - a set of initial states $I \subseteq S$;
  - a set of transitions $R \subseteq S \times S$;
  - a set of atomic propositions AP;
  - a labeling function $L : S \longmapsto 2^{AP}$;
  - a set of fairness conditions $F = \{f_1, \ldots, f_n\}$, with $f_i \subseteq S$.



- E.g., $\{\{2\}\} := \{\{s \ : \ L(s) = \{q\}\}\} = \{\mathbf{GF}q\}$ is the set of fairness conditions of the Kripke model above
- Fair path π: at least one state for each f_i occurs infinitely often in π (φ_i holds infinitely often in π: $\pi \models \mathbf{GF}\varphi_i$)
  - E.g., every path visiting infinitely often state 2 is a fair path.
- Fair state: a state through which at least one fair path passes
  - E.g., all states 1,2,3,4 are fair states
- Note: fair state $\neq$ state belonging to a fairness condition

Computing an NBA A_M from a Fair Kripke Model M

- Transforming a fair K.S. $M = \langle S, S_0, R, L, AP, FT \rangle$, $FT = \{F_1, ..., F_n\}$, into a generalized NBA $A_M = \langle Q, \Sigma, \delta, I, FT' \rangle$ s.t.:
  - States: $Q := S \cup \{init\}$, *init* being a new initial state
  - Alphabet: $\Sigma := 2^{AP}$
  - Initial State: $I := \{init\}$
  - Accepting States: $FT' := FT$
  - Transitions:

$$\delta : \quad \begin{array}{l} q \xrightarrow{a} q' \text{ iff } (q, q') \in R \text{ and } L(q') = a \\ init \xrightarrow{a} q \text{ iff } q \in S_0 \text{ and } L(q) = a \end{array}$$

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$

Computing an NBA A_M from a Fair Kripke Model M

- Transforming a fair K.S. $M = \langle S, S_0, R, L, AP, FT \rangle$, $FT = \{F_1, ..., F_n\}$, into a generalized NBA $A_M = \langle Q, \Sigma, \delta, I, FT' \rangle$ s.t.:
  - States: $Q := S \cup \{init\}$, *init* being a new initial state
  - Alphabet: $\Sigma := 2^{AP}$
  - Initial State: $I := \{init\}$
  - Accepting States: $FT' := FT$
  - Transitions:

$$\delta : \quad q \xrightarrow{a} q' \text{ iff } (q, q') \in R \text{ and } L(q') = a$$
$$init \xrightarrow{a} q \text{ iff } q \in S_0 \text{ and } L(q) = a$$

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$

Computing an NBA A_M from a Fair Kripke Model M

- Transforming a fair K.S. $M = \langle S, S_0, R, L, AP, FT \rangle$, $FT = \{F_1, ..., F_n\}$, into a generalized NBA $A_M = \langle Q, \Sigma, \delta, I, FT' \rangle$ s.t.:
  - States: $Q := S \cup \{init\}$, *init* being a new initial state
  - Alphabet: $\Sigma := 2^{AP}$
  - Initial State: $I := \{init\}$
  - Accepting States: $FT' := FT$
  - Transitions:

$$\begin{aligned} \delta: \quad & q \xrightarrow{a} q' \text{ iff } (q, q') \in R \text{ and } L(q') = a \\ & init \xrightarrow{a} q \text{ iff } q \in S_0 \text{ and } L(q) = a \end{aligned}$$

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$

Computing an NBA A_M from a Fair Kripke Model M

- Transforming a fair K.S. $M = \langle S, S_0, R, L, AP, FT \rangle$, $FT = \{F_1, ..., F_n\}$, into a generalized NBA $A_M = \langle Q, \Sigma, \delta, I, FT' \rangle$ s.t.:
  - States: $Q := S \cup \{init\}$, *init* being a new initial state
  - Alphabet: $\Sigma := 2^{AP}$
  - Initial State: $I := \{init\}$
  - Accepting States: $FT' := FT$
  - Transitions:

$$\delta : \quad q \xrightarrow{a} q' \text{ iff } (q, q') \in R \text{ and } L(q') = a$$
$$init \xrightarrow{a} q \text{ iff } q \in S_0 \text{ and } L(q) = a$$

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$

Computing an NBA A_M from a Fair Kripke Model M

- Transforming a fair K.S. $M = \langle S, S_0, R, L, AP, FT \rangle$, $FT = \{F_1, ..., F_n\}$, into a generalized NBA $A_M = \langle Q, \Sigma, \delta, I, FT' \rangle$ s.t.:
  - States: $Q := S \cup \{init\}$, *init* being a new initial state
  - Alphabet: $\Sigma := 2^{AP}$
  - Initial State: $I := \{init\}$
  - Accepting States: $FT' := FT$
  - Transitions:

$$\delta : \quad q \xrightarrow{a} q' \text{ iff } (q, q') \in R \text{ and } L(q') = a$$
$$init \xrightarrow{a} q \text{ iff } q \in S_0 \text{ and } L(q) = a$$

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$

Computing an NBA A_M from a Fair Kripke Model M

- Transforming a fair K.S. $M = \langle S, S_0, R, L, AP, FT \rangle$, $FT = \{F_1, ..., F_n\}$, into a generalized NBA $A_M = \langle Q, \Sigma, \delta, I, FT' \rangle$ s.t.:
  - States: $Q := S \cup \{init\}$, *init* being a new initial state
  - Alphabet: $\Sigma := 2^{AP}$
  - Initial State: $I := \{init\}$
  - Accepting States: $FT' := FT$
  - Transitions:

$$\begin{aligned} \delta : \quad & q \xrightarrow{a} q' \text{ iff } (q, q') \in R \textit{ and } L(q') = a \\ & init \xrightarrow{a} q \text{ iff } q \in S_0 \textit{ and } L(q) = a \end{aligned}$$

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$

Computing an NBA A_M from a Fair Kripke Model M

- Transforming a fair K.S. $M = \langle S, S_0, R, L, AP, FT \rangle$, $FT = \{F_1, ..., F_n\}$, into a generalized NBA $A_M = \langle Q, \Sigma, \delta, I, FT' \rangle$ s.t.:
  - States: $Q := S \cup \{init\}$, *init* being a new initial state
  - Alphabet: $\Sigma := 2^{AP}$
  - Initial State: $I := \{init\}$
  - Accepting States: $FT' := FT$
  - Transitions:

$$\delta : \quad \begin{array}{l} q \xrightarrow{a} q' \text{ iff } (q, q') \in R \text{ and } L(q') = a \\ init \xrightarrow{a} q \text{ iff } q \in S_0 \text{ and } L(q) = a \end{array}$$

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$

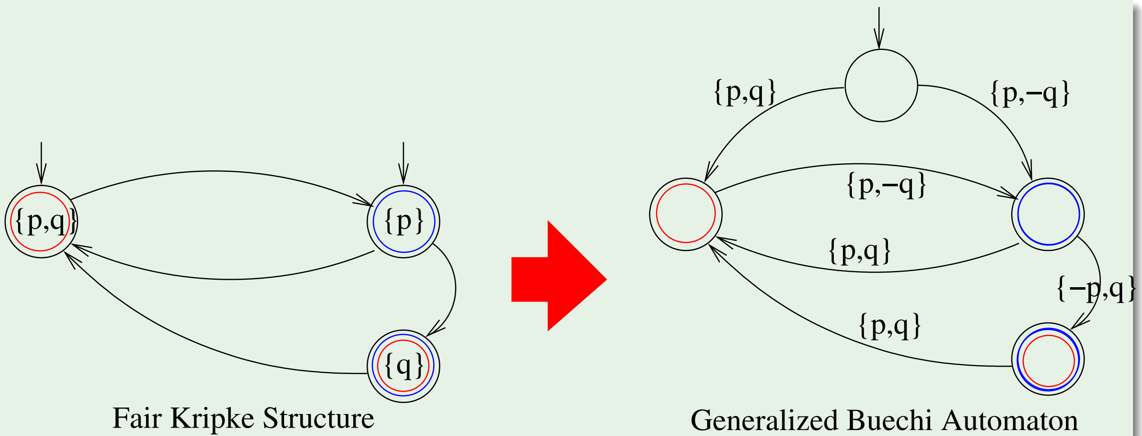
Computing an NBA A_M from a Fair Kripke Model M

- Transforming a fair K.S. $M = \langle S, S_0, R, L, AP, FT\rangle$, $FT = \{F_1, ..., F_n\}$, into a generalized NBA $A_M = \langle Q, \Sigma, \delta, I, FT'\rangle$ s.t.:
  - States: $Q := S \cup \{init\}$, *init* being a new initial state
  - Alphabet: $\Sigma := 2^{AP}$
  - Initial State: $I := \{init\}$
  - Accepting States: $FT' := FT$
  - Transitions:

$$\delta : \quad \begin{array}{l} q \xrightarrow{a} q' \text{ iff } (q, q') \in R \textit{ and } L(q') = a \\ init \xrightarrow{a} q \text{ iff } q \in S_0 \textit{ and } L(q) = a \end{array}$$

- $\mathcal{L}(A_M) = \mathcal{L}(M)$
- $|A_M| = |M| + 1$

Computing a (Generalized) BA A_M from a Fair Kripke Structure M: Example



$\Longrightarrow$ Substantially, add one initial state, move labels from states to incoming edges, set fair states as accepting states

Outline

CTL M.C. with Fair Kripke Models

Fair Kripke Models restrict the M.C. process to fair paths:

- $M_f \models \varphi$ iff $\pi \models \varphi$ for every fair path π
- Path quantifiers (from CTL) apply only to fair paths:
  - $M_F, s \models \mathbf{A}\varphi$ iff $\pi, s \models \varphi$ for every fair path π s.t. $s \in \pi$
  - $M_F, s \models \mathbf{E}\varphi$ iff $\pi, s \models \varphi$ for some fair path π s.t. $s \in \pi$

$\Longrightarrow$ a fair state s is a state in M_F iff $M_F, s \models \mathbf{EG}\mathit{true}$.

- We need a procedure to compute the set of fair states: `Check_FairEG(true)`

Example

- $M_f \models \mathbf{EG}\mathit{true}$? yes
- $M_f \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? yes
- $M \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? no

CTL M.C. with Fair Kripke Models

Fair Kripke Models restrict the M.C. process to fair paths:

- $M_f \models \varphi$ iff $\pi \models \varphi$ for every fair path π
- Path quantifiers (from CTL) apply only to fair paths:
  - $M_F, s \models \mathbf{A}\varphi$ iff $\pi, s \models \varphi$ for every fair path π s.t. $s \in \pi$
  - $M_F, s \models \mathbf{E}\varphi$ iff $\pi, s \models \varphi$ for some fair path π s.t. $s \in \pi$

$\Longrightarrow$ a fair state s is a state in M_F iff $M_F, s \models \mathbf{EG}\mathit{true}$.

- We need a procedure to compute the set of fair states: `Check_FairEG(true)`

Example

- $M_f \models \mathbf{EG}\mathit{true}$? yes
- $M_f \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? yes
- $M \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? no

CTL M.C. with Fair Kripke Models

Fair Kripke Models restrict the M.C. process to fair paths:

- $M_f \models \varphi$ iff $\pi \models \varphi$ for every fair path π
- Path quantifiers (from CTL) apply only to fair paths:
  - $M_F, s \models \mathbf{A}\varphi$ iff $\pi, s \models \varphi$ for every fair path π s.t. $s \in \pi$
  - $M_F, s \models \mathbf{E}\varphi$ iff $\pi, s \models \varphi$ for some fair path π s.t. $s \in \pi$

$\Longrightarrow$ a fair state s is a state in M_F iff $M_F, s \models \mathbf{EG}\mathit{true}$.

- We need a procedure to compute the set of fair states: `Check_FairEG(true)`

Example

- $M_f \models \mathbf{EG}\mathit{true}$? yes
- $M_f \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? yes
- $M \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? no

CTL M.C. with Fair Kripke Models

Fair Kripke Models restrict the M.C. process to fair paths:

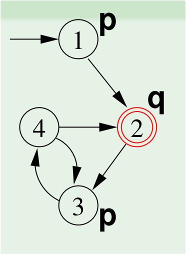
- $M_f \models \varphi$ iff $\pi \models \varphi$ for every fair path π
- Path quantifiers (from CTL) apply only to fair paths:
  - $M_F, s \models \mathbf{A}\varphi$ iff $\pi, s \models \varphi$ for every fair path π s.t. $s \in \pi$
  - $M_F, s \models \mathbf{E}\varphi$ iff $\pi, s \models \varphi$ for some fair path π s.t. $s \in \pi$

$\Longrightarrow$ a fair state s is a state in M_F iff $M_F, s \models \mathbf{EG}\mathit{true}$.

- We need a procedure to compute the set of fair states: `Check_FairEG(true)`

Example

- $M_f \models \mathbf{EG}\mathit{true}$? yes
- $M_f \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? yes
- $M \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? no



CTL M.C. with Fair Kripke Models

Fair Kripke Models restrict the M.C. process to fair paths:

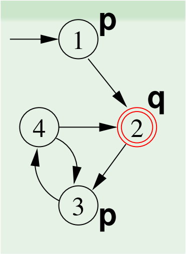
- $M_f \models \varphi$ iff $\pi \models \varphi$ for every fair path π
- Path quantifiers (from CTL) apply only to fair paths:
  - $M_F, s \models \mathbf{A}\varphi$ iff $\pi, s \models \varphi$ for every fair path π s.t. $s \in \pi$
  - $M_F, s \models \mathbf{E}\varphi$ iff $\pi, s \models \varphi$ for some fair path π s.t. $s \in \pi$

$\Longrightarrow$ a fair state s is a state in M_F iff $M_F, s \models \mathbf{EG}\mathit{true}$.

- We need a procedure to compute the set of fair states: `Check_FairEG(true)`

Example

- $M_f \models \mathbf{EG}\mathit{true}$? yes
- $M_f \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? yes
- $M \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? no



CTL M.C. with Fair Kripke Models

Fair Kripke Models restrict the M.C. process to fair paths:

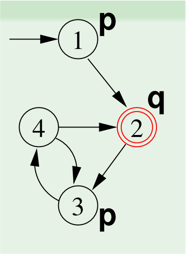
- $M_f \models \varphi$ iff $\pi \models \varphi$ for every fair path π
- Path quantifiers (from CTL) apply only to fair paths:
  - $M_F, s \models \mathbf{A}\varphi$ iff $\pi, s \models \varphi$ for every fair path π s.t. $s \in \pi$
  - $M_F, s \models \mathbf{E}\varphi$ iff $\pi, s \models \varphi$ for some fair path π s.t. $s \in \pi$

$\Longrightarrow$ a fair state s is a state in M_F iff $M_F, s \models \mathbf{EG}true$.

- We need a procedure to compute the set of fair states: `Check_FairEG(true)`

Example

- $M_f \models \mathbf{EG}true$? yes
- $M_f \models \mathbf{G}(p \to \mathbf{F}q)$? yes
- $M \models \mathbf{G}(p \to \mathbf{F}q)$? no



CTL M.C. with Fair Kripke Models

Fair Kripke Models restrict the M.C. process to fair paths:

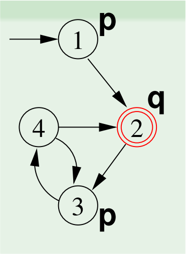
- $M_f \models \varphi$ iff $\pi \models \varphi$ for every fair path π
- Path quantifiers (from CTL) apply only to fair paths:
  - $M_F, s \models \mathbf{A}\varphi$ iff $\pi, s \models \varphi$ for every fair path π s.t. $s \in \pi$
  - $M_F, s \models \mathbf{E}\varphi$ iff $\pi, s \models \varphi$ for some fair path π s.t. $s \in \pi$

$\Longrightarrow$ a fair state s is a state in M_F iff $M_F, s \models \mathbf{EG}true$.

- We need a procedure to compute the set of fair states: `Check_FairEG(true)`

Example

- $M_f \models \mathbf{EG}true$? yes
- $M_f \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? yes
- $M \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? no



CTL M.C. with Fair Kripke Models

Fair Kripke Models restrict the M.C. process to fair paths:

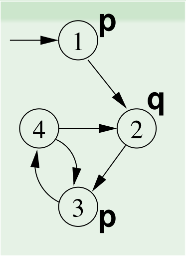
- $M_f \models \varphi$ iff $\pi \models \varphi$ for every fair path π
- Path quantifiers (from CTL) apply only to fair paths:
  - $M_F, s \models \mathbf{A}\varphi$ iff $\pi, s \models \varphi$ for every fair path π s.t. $s \in \pi$
  - $M_F, s \models \mathbf{E}\varphi$ iff $\pi, s \models \varphi$ for some fair path π s.t. $s \in \pi$

$\Longrightarrow$ a fair state s is a state in M_F iff $M_F, s \models \mathbf{EG}\mathit{true}$.

- We need a procedure to compute the set of fair states: `Check_FairEG(true)`

Example

- $M_f \models \mathbf{EG}\mathit{true}$? yes
- $M_f \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? yes
- $M \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? no



CTL M.C. with Fair Kripke Models

Fair Kripke Models restrict the M.C. process to fair paths:

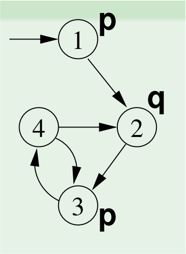
- $M_f \models \varphi$ iff $\pi \models \varphi$ for every fair path π
- Path quantifiers (from CTL) apply only to fair paths:
  - $M_F, s \models \mathbf{A}\varphi$ iff $\pi, s \models \varphi$ for every fair path π s.t. $s \in \pi$
  - $M_F, s \models \mathbf{E}\varphi$ iff $\pi, s \models \varphi$ for some fair path π s.t. $s \in \pi$

$\Longrightarrow$ a fair state s is a state in M_F iff $M_F, s \models \mathbf{EG}\mathit{true}$.

- We need a procedure to compute the set of fair states: `Check_FairEG(true)`

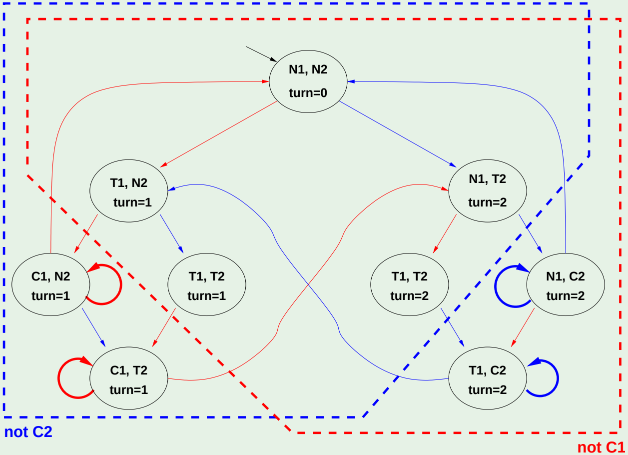
Example

- $M_f \models \mathbf{EG}\mathit{true}$? yes
- $M_f \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? yes
- $M \models \mathbf{G}(p \rightarrow \mathbf{F}q)$? no



Fair CTL Model Checking: Example

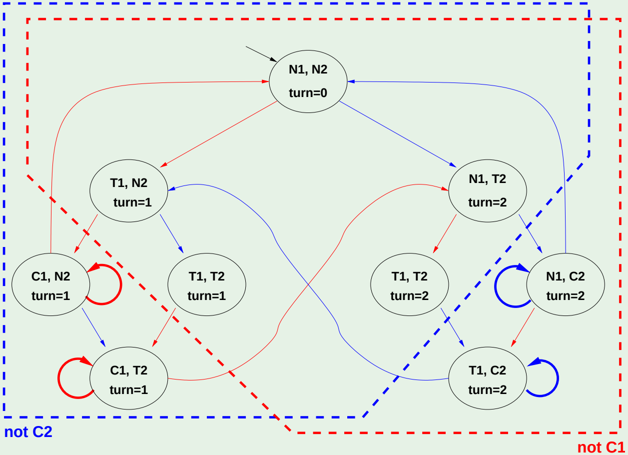
F := {{ not C1},{not C2}}



$M_F \models \mathbf{G}(T_1 \rightarrow \mathbf{F}C_1)$? $M_F \models \mathbf{G}(T_2 \rightarrow \mathbf{F}C_2)$?

Fair CTL Model Checking: Example

F := {{ not C1},{not C2}}



$M_F \models \mathbf{G}(T_1 \rightarrow \mathbf{F}C_1)$? $M_F \models \mathbf{G}(T_2 \rightarrow \mathbf{F}C_2)$? YES: every fair path satisfies the conditions

CTL M.C. vs. LTL M.C. with Fair Kripke Models

Remark: fair CTL M.C.

When model checking a CTL formula ψ, fairness conditions cannot be encoded into the formula:

$$M_{\{f_1,...,f_n\}} \models \psi \not\Longleftrightarrow M \models (\bigwedge_{i=1}^{n} \mathbf{AGAF} f_i) \rightarrow \psi.$$

$$M_{\{f_1,...,f_n\}} \models \psi \not\Longleftrightarrow M \models (\bigwedge_{i=1}^{n} \mathbf{EGEF} f_i) \rightarrow \psi.$$

$\Longrightarrow$ We need specific procedures for Fair CTL Model Checking.

Remark: fair LTL M.C.

When model checking an LTL formula ψ, fairness conditions can be encoded into the formula:

$$M_{\{f_1,...,f_n\}} \models \psi \Longleftrightarrow M \models (\bigwedge_{i=1}^{n} \mathbf{GF} f_i) \rightarrow \psi.$$

$\Longrightarrow$ There is no need for Fair LTL Model Checking procedures.

CTL M.C. vs. LTL M.C. with Fair Kripke Models

Remark: fair CTL M.C.

When model checking a CTL formula ψ, fairness conditions cannot be encoded into the formula:

$$M_{\{f_1,\ldots,f_n\}} \models \psi \not\Longleftrightarrow M \models (\bigwedge_{i=1}^{n} \mathbf{AGAF} f_i) \rightarrow \psi.$$

$$M_{\{f_1,\ldots,f_n\}} \models \psi \not\Longleftrightarrow M \models (\bigwedge_{i=1}^{n} \mathbf{EGEF} f_i) \rightarrow \psi.$$

$\Longrightarrow$ We need specific procedures for Fair CTL Model Checking.

Remark: fair LTL M.C.

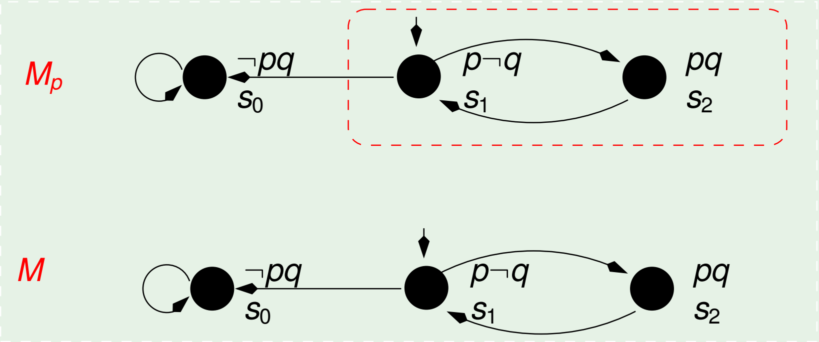
When model checking an LTL formula ψ, fairness conditions can be encoded into the formula:

$$M_{\{f_1,\ldots,f_n\}} \models \psi \Longleftrightarrow M \models (\bigwedge_{i=1}^{n} \mathbf{GF} f_i) \rightarrow \psi.$$

$\Longrightarrow$ There is no need for Fair LTL Model Checking procedures.

Ex. CTL: $M_{\{f_1,\ldots,f_n\}} \models \psi \not\Longleftrightarrow M \models (\bigwedge_{i=1}^{n} \mathbf{AGAF} f_i) \to \psi$.

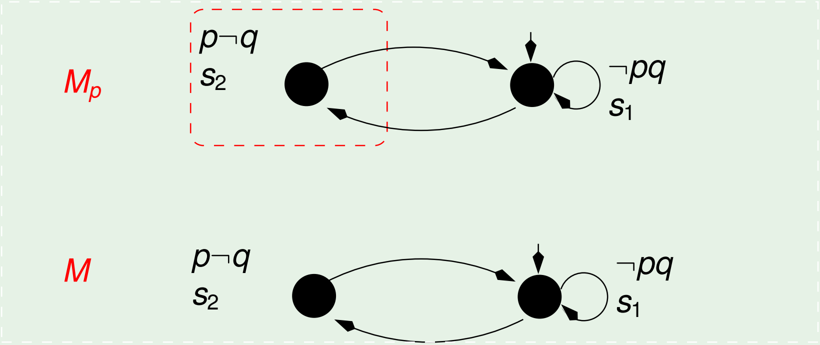
[Example provided by the student Davide Kirchner, 2014]



- $M_p \not\models \mathbf{AG}q$
- $M \models (\mathbf{AGAF}p) \to \mathbf{AG}q$

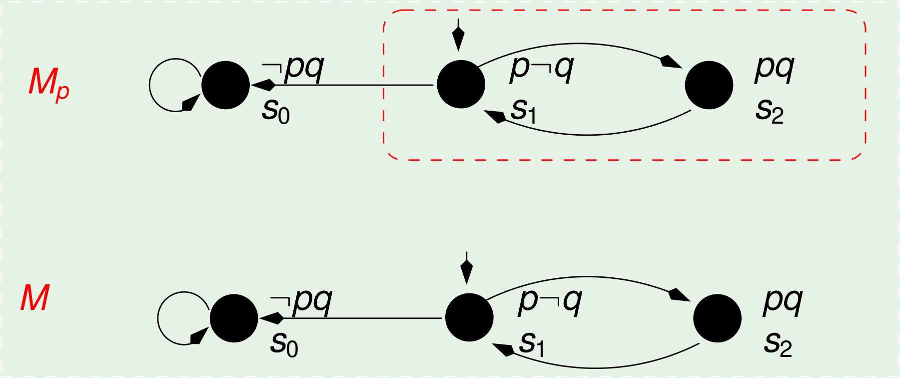
Ex. CTL: $M_{\{f_1,\ldots,f_n\}} \models \psi \not\Longleftrightarrow M \models (\bigwedge_{i=1}^{n} \mathbf{EGEF} f_i) \rightarrow \psi$.

[Example provided by the student Daniele Giuliani, 2019]



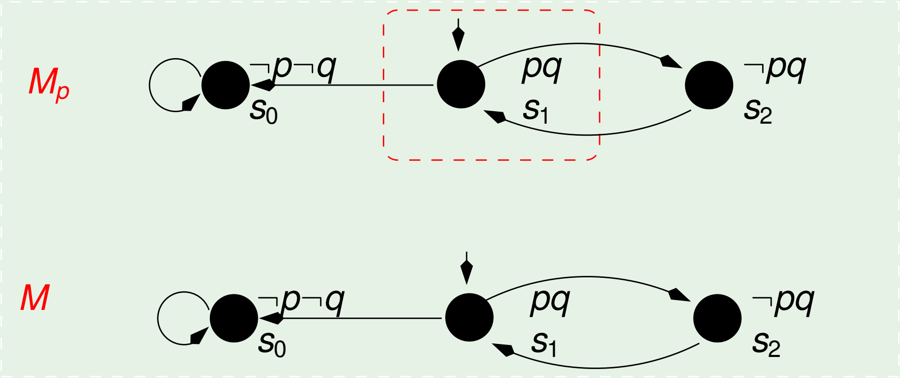
- $M_p \not\models \mathbf{EFEG}q$
- $M \models (\mathbf{EGEF}p) \rightarrow \mathbf{EFEG}q$

Ex. LTL (1): $M_{\{f_1,\ldots,f_n\}} \models \psi \iff M \models (\bigwedge_{i=1}^{n} \mathbf{GF} f_i) \rightarrow \psi$.



- $M_p \not\models \mathbf{G}q$
- $M \not\models (\mathbf{GF}p) \rightarrow \mathbf{G}q$

Ex. LTL (2): $M_{\{f_1,\ldots,f_n\}} \models \psi \iff M \models (\bigwedge_{i=1}^{n} \mathbf{GF} f_i) \rightarrow \psi$.



- $M_p \models \mathbf{G} q$
- $M \models (\mathbf{GF} p) \rightarrow \mathbf{G} q$

Fair CTL Model Checking

- In order to solve the fair CTL model checking problem, we must be able to compute:
  - $[\varphi_f]$ s.t. φ Boolean (i.e. $[\varphi]$ under fairness conditions f)
  - $[\mathbf{E}_f\mathbf{X}(\varphi)]$ (i.e. $[\mathbf{EX}\varphi]$ under fairness conditions f)
  - $[\mathbf{E}_f(\varphi\mathbf{U}\psi)]$ (i.e. $[\mathbf{E}(\varphi\mathbf{U}\psi)]$ under fairness conditions f)
  - $[\mathbf{E}_f\mathbf{G}\varphi]$ (i.e. $[\mathbf{EG}\varphi]$ under fairness conditions f).
- Suppose we have a procedure `Check_FairEG` to compute $[\mathbf{E}_f\mathbf{G}\varphi]$.
- Let $fair \stackrel{\text{def}}{=} \mathbf{E}_f\mathbf{G}true$. ($M, s \models \mathbf{E}_f\mathbf{G}true$ if s is a fair state.)
- if φ is Boolean, then $M_f, s \models \varphi$ iff $M, s \models (\varphi \wedge fair)$
- We can rewrite all the other fair operators:
  - $\mathbf{E}_f\mathbf{X}(\varphi) \equiv \mathbf{EX}(\varphi \wedge fair)$
  - $\mathbf{E}_f(\varphi\mathbf{U}\psi) \equiv \mathbf{E}(\varphi\mathbf{U}(\psi \wedge fair))$

Fair CTL Model Checking

- In order to solve the fair CTL model checking problem, we must be able to compute:
  - $[\varphi_f]$ s.t. φ Boolean (i.e. $[\varphi]$ under fairness conditions f)
  - $[\mathbf{E}_f\mathbf{X}(\varphi)]$ (i.e. $[\mathbf{EX}\varphi]$ under fairness conditions f)
  - $[\mathbf{E}_f(\varphi\mathbf{U}\psi)]$ (i.e. $[\mathbf{E}(\varphi\mathbf{U}\psi)]$ under fairness conditions f)
  - $[\mathbf{E}_f\mathbf{G}\varphi]$ (i.e. $[\mathbf{EG}\varphi]$ under fairness conditions f).
- Suppose we have a procedure `Check_FairEG` to compute $[\mathbf{E}_f\mathbf{G}\varphi]$.
- Let $fair \stackrel{\text{def}}{=} \mathbf{E}_f\mathbf{G}true$. ($M, s \models \mathbf{E}_f\mathbf{G}true$ if s is a fair state.)
- if φ is Boolean, then $M_f, s \models \varphi$ iff $M, s \models (\varphi \wedge fair)$
- We can rewrite all the other fair operators:
  - $\mathbf{E}_f\mathbf{X}(\varphi) \equiv \mathbf{EX}(\varphi \wedge fair)$
  - $\mathbf{E}_f(\varphi\mathbf{U}\psi) \equiv \mathbf{E}(\varphi\mathbf{U}(\psi \wedge fair))$

Fair CTL Model Checking

- In order to solve the fair CTL model checking problem, we must be able to compute:
  - $[\varphi_f]$ s.t. φ Boolean (i.e. $[\varphi]$ under fairness conditions f)
  - $[\mathbf{E}_f\mathbf{X}(\varphi)]$ (i.e. $[\mathbf{EX}\varphi]$ under fairness conditions f)
  - $[\mathbf{E}_f(\varphi\mathbf{U}\psi)]$ (i.e. $[\mathbf{E}(\varphi\mathbf{U}\psi)]$ under fairness conditions f)
  - $[\mathbf{E}_f\mathbf{G}\varphi]$ (i.e. $[\mathbf{EG}\varphi]$ under fairness conditions f).
- Suppose we have a procedure `Check_FairEG` to compute $[\mathbf{E}_f\mathbf{G}\varphi]$.
- Let $fair \stackrel{\text{def}}{=} \mathbf{E}_f\mathbf{G}true$. ($M, s \models \mathbf{E}_f\mathbf{G}true$ if s is a fair state.)
- if φ is Boolean, then $M_f, s \models \varphi$ iff $M, s \models (\varphi \wedge fair)$
- We can rewrite all the other fair operators:
  - $\mathbf{E}_f\mathbf{X}(\varphi) \equiv \mathbf{EX}(\varphi \wedge fair)$
  - $\mathbf{E}_f(\varphi\mathbf{U}\psi) \equiv \mathbf{E}(\varphi\mathbf{U}(\psi \wedge fair))$

Fair CTL Model Checking

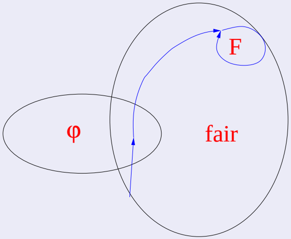
- In order to solve the fair CTL model checking problem, we must be able to compute:
  - $[\varphi_f]$ s.t. φ Boolean (i.e. $[\varphi]$ under fairness conditions f)
  - $[\mathbf{E}_f\mathbf{X}(\varphi)]$ (i.e. $[\mathbf{EX}\varphi]$ under fairness conditions f)
  - $[\mathbf{E}_f(\varphi\mathbf{U}\psi)]$ (i.e. $[\mathbf{E}(\varphi\mathbf{U}\psi)]$ under fairness conditions f)
  - $[\mathbf{E}_f\mathbf{G}\varphi]$ (i.e. $[\mathbf{EG}\varphi]$ under fairness conditions f).
- Suppose we have a procedure `Check_FairEG` to compute $[\mathbf{E}_f\mathbf{G}\varphi]$.
- Let $fair \stackrel{\text{def}}{=} \mathbf{E}_f\mathbf{G}true$. ($M, s \models \mathbf{E}_f\mathbf{G}true$ if s is a fair state.)
- if φ is Boolean, then $M_f, s \models \varphi$ iff $M, s \models (\varphi \wedge fair)$
- We can rewrite all the other fair operators:
  - $\mathbf{E}_f\mathbf{X}(\varphi) \equiv \mathbf{EX}(\varphi \wedge fair)$
  - $\mathbf{E}_f(\varphi\mathbf{U}\psi) \equiv \mathbf{E}(\varphi\mathbf{U}(\psi \wedge fair))$

Fair CTL Model Checking

- In order to solve the fair CTL model checking problem, we must be able to compute:
  - $[\varphi_f]$ s.t. φ Boolean (i.e. $[\varphi]$ under fairness conditions f)
  - $[\mathbf{E}_f\mathbf{X}(\varphi)]$ (i.e. $[\mathbf{EX}\varphi]$ under fairness conditions f)
  - $[\mathbf{E}_f(\varphi\mathbf{U}\psi)]$ (i.e. $[\mathbf{E}(\varphi\mathbf{U}\psi)]$ under fairness conditions f)
  - $[\mathbf{E}_f\mathbf{G}\varphi]$ (i.e. $[\mathbf{EG}\varphi]$ under fairness conditions f).
- Suppose we have a procedure `Check_FairEG` to compute $[\mathbf{E}_f\mathbf{G}\varphi]$.
- Let $fair \stackrel{\text{def}}{=} \mathbf{E}_f\mathbf{G}true$. ($M, s \models \mathbf{E}_f\mathbf{G}true$ if s is a fair state.)
- if φ is Boolean, then $M_f, s \models \varphi$ iff $M, s \models (\varphi \wedge fair)$
- We can rewrite all the other fair operators:
  - $\mathbf{E}_f\mathbf{X}(\varphi) \equiv \mathbf{EX}(\varphi \wedge fair)$
  - $\mathbf{E}_f(\varphi\mathbf{U}\psi) \equiv \mathbf{E}(\varphi\mathbf{U}(\psi \wedge fair))$

Fair CTL Model Checking

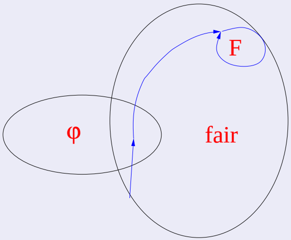
- $\mathbf{E}_f\mathbf{X}(\varphi) \equiv \mathbf{EX}(\varphi \wedge \mathit{fair})$:



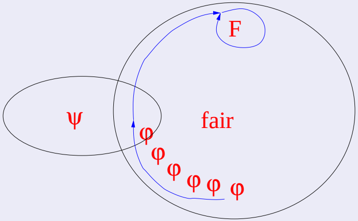
- $\mathbf{E}_f(\varphi \mathbf{U} \psi) \equiv \mathbf{E}(\varphi \mathbf{U}(\psi \wedge \mathit{fair}))$:

Fair CTL Model Checking

- $\mathbf{E}_f\mathbf{X}(\varphi) \equiv \mathbf{EX}(\varphi \wedge \mathit{fair})$:



- $\mathbf{E}_f(\varphi\mathbf{U}\psi) \equiv \mathbf{E}(\varphi\mathbf{U}(\psi \wedge \mathit{fair}))$:



Language-Emptiness Checking for Fair Kripke Models

Fair_CheckEG

Given: a fair Kripke model $M_F := \langle S, R, I, AP, L, F \rangle$ and a CTL formula φ s.t. $[\varphi] \subseteq S$,
Fair_CheckEG(φ) returns the subset of the states s in $[\varphi]$ from which at least one fair path π entirely included in $[\varphi]$ passes through

Fair_CheckEG(*true*) computes the set of fair states of M_f

$\Longrightarrow I \subseteq$ Fair_CheckEG(*true*) iff $\mathcal{L}(M_f) \neq \emptyset$

Language-Emptiness Checking for Fair Kripke Models

Fair_CheckEG

Given: a fair Kripke model $M_F := \langle S, R, I, AP, L, F \rangle$ and a CTL formula φ s.t. $[\varphi] \subseteq S$, Fair_CheckEG(φ) returns the subset of the states s in $[\varphi]$ from which at least one fair path π entirely included in $[\varphi]$ passes through

Fair_CheckEG(*true*) computes the set of fair states of M_f

$\Longrightarrow$ $I \subseteq$ Fair_CheckEG(*true*) iff $\mathcal{L}(M_f) \neq \emptyset$

Language-Emptiness Checking for Fair Kripke Models

Fair_CheckEG

Given: a fair Kripke model $M_F := \langle S, R, I, AP, L, F \rangle$ and a CTL formula φ s.t. $[\varphi] \subseteq S$, `Fair_CheckEG`(φ) returns the subset of the states s in $[\varphi]$ from which at least one fair path π entirely included in $[\varphi]$ passes through

`Fair_CheckEG`(*true*) computes the set of fair states of M_f

$\Longrightarrow$ $I \subseteq$ `Fair_CheckEG`(*true*) iff $\mathcal{L}(M_f) \neq \emptyset$

Ingredients (from CTL Model Checking)

Some primitive functions from CTL Model Checking:

- `Check_EX`(ϕ): returns the set of states from which a path verifying $\mathbf{X}\phi$ holds (i.e., the preimage of the set of states where ϕ holds)
- `Check_EG`(ϕ): returns the set of states from which a path verifying $\mathbf{G}\phi$ holds
- `Check_EU`(ϕ_1, ϕ_2): returns the set of states from which a path verifying $\phi_1 \mathbf{U} \phi_2$ holds

Outline

SCC-based Check_FairEG

A Strongly Connected Component (SCC) of a directed graph is a maximal subgraph s.t. all its nodes are reachable from each other.

Given a fair Kripke model M, a fair non-trivial SCC is an SCC with at least one edge that contains at least one state for every fair condition

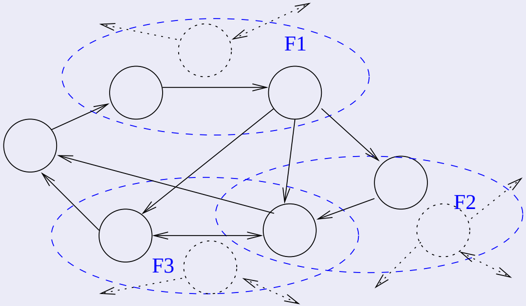
$\Longrightarrow$ all states in a fair (non-trivial) SCC are fair states

SCC-based Check_FairEG

A Strongly Connected Component (SCC) of a directed graph is a maximal subgraph s.t. all its nodes are reachable from each other.

Given a fair Kripke model M, a fair non-trivial SCC is an SCC with at least one edge that contains at least one state for every fair condition

$\Longrightarrow$ all states in a fair (non-trivial) SCC are fair states

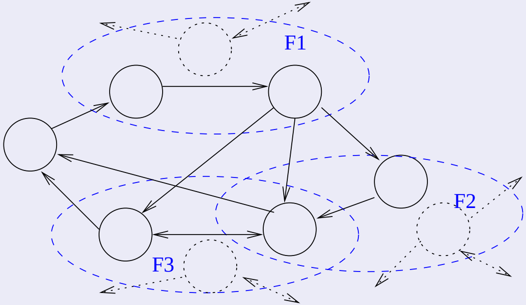


SCC-based Check_FairEG

A Strongly Connected Component (SCC) of a directed graph is a maximal subgraph s.t. all its nodes are reachable from each other.

Given a fair Kripke model M, a fair non-trivial SCC is an SCC with at least one edge that contains at least one state for every fair condition

$\Longrightarrow$ all states in a fair (non-trivial) SCC are fair states



SCC-based Check_FairEG (cont.)

`Check_FairEG(`$[\phi]$`):`

(i) restrict the graph of M to $[\phi]$;
(ii) find all fair non-trivial SCCs C_i
(iii) build $C := \cup_i C_i$;
(iv) compute the states that can reach C (`Check_EU(`$[\phi]$`, `C`)`).

$[\phi]$: set of states where ϕ holds (aka denotation of ϕ)

SCC-based Check_FairEG (cont.)

`Check_FairEG`($[\phi]$):

(i) restrict the graph of M to $[\phi]$;

(ii) find all fair non-trivial SCCs C_i

(iii) build $C := \cup_i C_i$;

(iv) compute the states that can reach C (`Check_EU`($[\phi], C$)).

$[\phi]$: set of states where ϕ holds (aka denotation of ϕ)

SCC-based Check_FairEG (cont.)

`Check_FairEG([`ϕ`]):`

(i) restrict the graph of M to $[\phi]$;
(ii) find all fair non-trivial SCCs C_i
(iii) build $C := \cup_i C_i$;
(iv) compute the states that can reach C (`Check_EU(`$[\phi], C$`)`).

$[\phi]$: set of states where ϕ holds (aka denotation of ϕ)

SCC-based Check_FairEG (cont.)

`Check_FairEG(`$[\phi]$`):`

(i) restrict the graph of M to $[\phi]$;
(ii) find all fair non-trivial SCCs C_i
(iii) build $C := \cup_i C_i$;
(iv) compute the states that can reach C (`Check_EU(`$[\phi]$, C`)`).

$[\phi]$: set of states where ϕ holds (aka denotation of ϕ)

SCC-based Check_FairEG (cont.)

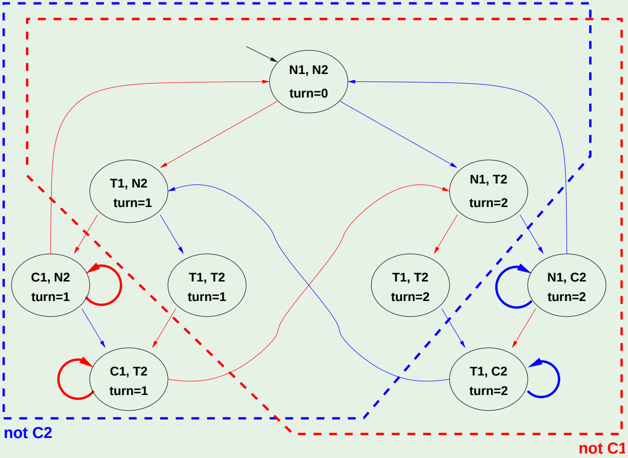
`Check_FairEG(`$[\phi]$`):`

(i) restrict the graph of M to $[\phi]$;
(ii) find all fair non-trivial SCCs C_i
(iii) build $C := \cup_i C_i$;
(iv) compute the states that can reach C (`Check_EU(`$[\phi]$`,` C`)`).

$[\phi]$: set of states where ϕ holds (aka denotation of ϕ)

Example: Check_FairEG

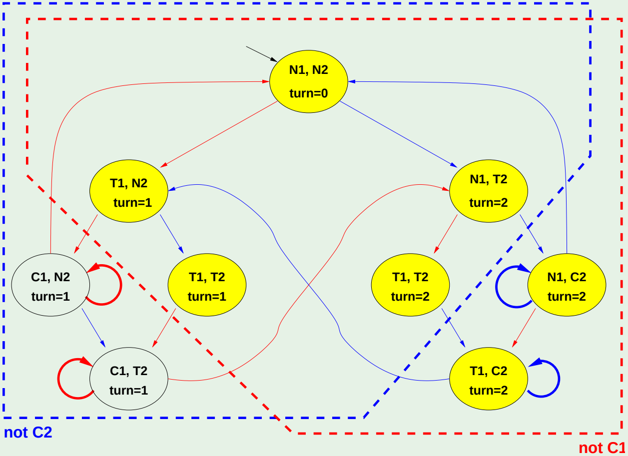
F := {{ not C1},{not C2}}



EG$\neg C_1$

Example: Check_FairEG

F := {{ not C1},{not C2}}

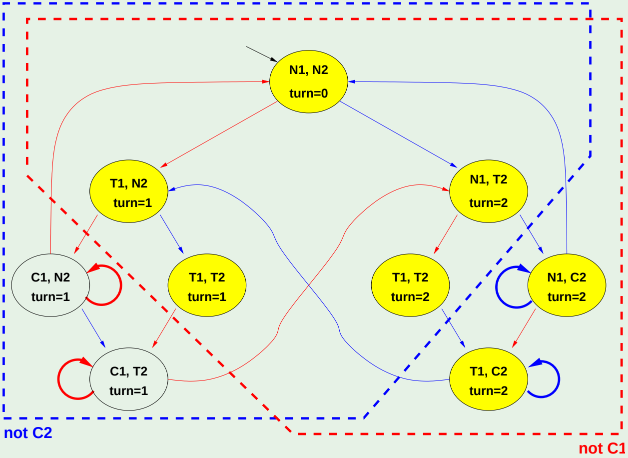


EG $\neg C_1$

Check_FairEG($\neg C_1$): 1. compute $[\neg C_1]$

Example: Check_FairEG

F := {{ not C1},{not C2}}

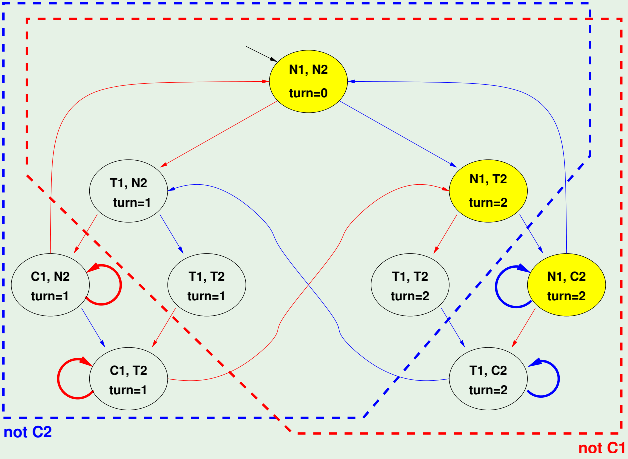


EG$\neg C_1$

Check_FairEG($\neg C_1$): 2. restrict the graph to $[\neg C_1]$

Example: Check_FairEG

F := {{ not C1},{not C2}}

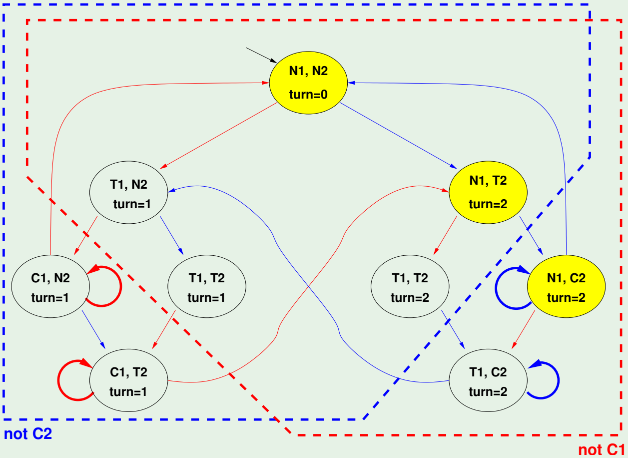


EG$\neg C_1$

Check_FairEG($\neg C_1$): 3. find all fair non-trivial SCC's

Example: Check_FairEG

F := {{ not C1},{not C2}}

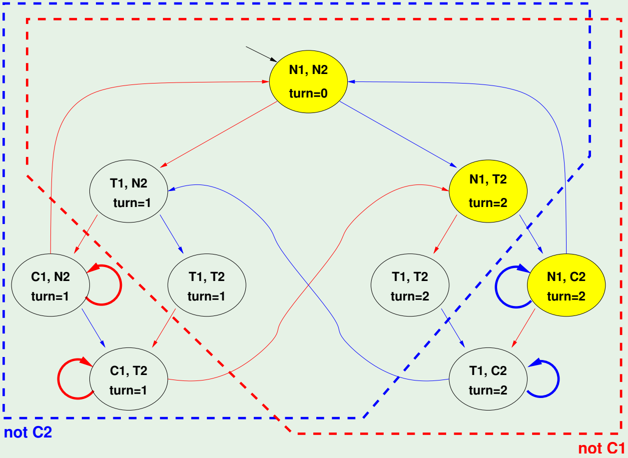


EG$\neg C_1$

Check_FairEG($\neg C_1$): 4. build the union C of all SCC's

Example: Check_FairEG

F := {{ not C1},{not C2}}



EG $\neg C_1$

Check_FairEG($\neg C_1$): 5. compute the states which can reach it

SCC-based Check_FairEG - Drawbacks

- SCCs computation requires a linear ($O(\#nodes + \#edges)$) DFS (Tarjan).
- The DFS manipulates the states explicitly, storing information for every state.
- A DFS is not suitable for symbolic model checking where we manipulate sets of states.

$\Longrightarrow$ We want an algorithm based on (symbolic) preimage computation.

SCC-based Check_FairEG - Drawbacks

- SCCs computation requires a linear ($O(\#nodes + \#edges)$) DFS (Tarjan).
- The DFS manipulates the states explicitly, storing information for every state.
- A DFS is not suitable for symbolic model checking where we manipulate sets of states.

$\Longrightarrow$ We want an algorithm based on (symbolic) preimage computation.

SCC-based Check_FairEG - Drawbacks

- SCCs computation requires a linear ($O(\#nodes + \#edges)$) DFS (Tarjan).
- The DFS manipulates the states explicitly, storing information for every state.
- A DFS is not suitable for symbolic model checking where we manipulate sets of states.

$\Longrightarrow$ We want an algorithm based on (symbolic) preimage computation.

SCC-based Check_FairEG - Drawbacks

- SCCs computation requires a linear ($O(\#nodes + \#edges)$) DFS (Tarjan).
- The DFS manipulates the states explicitly, storing information for every state.
- A DFS is not suitable for symbolic model checking where we manipulate sets of states.

$\Longrightarrow$ We want an algorithm based on (symbolic) preimage computation.

Outline

Emerson-Lei Algorithm

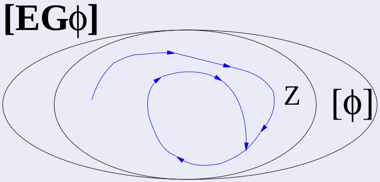
Fixpoint characterization of **EG** and fair **EG**

"[ϕ]" denotes the set of states where ϕ holds

- Theorem (Emerson & Clarke): $[\mathbf{EG}\phi] = \nu Z.([\phi] \cap [\mathbf{EX}Z])$
  The greatest set Z s.t. every state z in Z satisfies ϕ and reaches another state in Z in one step.

We can characterize fair **EG** (aka "$\mathbf{E}_f\mathbf{G}$") similarly:

- Theorem (Emerson & Lei): $[\mathbf{E}_f\mathbf{G}\phi] = \nu Z.([\phi] \cap \bigcap_{F_i \in FT} [\mathbf{EX}\ \mathbf{E}(Z\mathbf{U}(Z \cap F_i))])$
  The greatest set Z s.t. every state z in Z satisfies ϕ and, for every set $F_i \in FT$, z reaches a state in $F_i \cap Z$ by means of a non-trivial path that lies in Z.



Emerson-Lei Algorithm

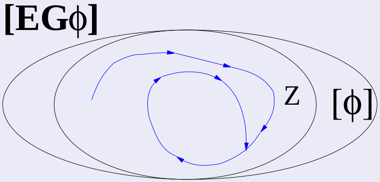
Fixpoint characterization of **EG** and fair **EG**

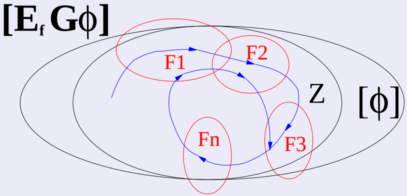
"[ϕ]" denotes the set of states where ϕ holds

- Theorem (Emerson & Clarke): $[\mathbf{EG}\phi] = \nu Z.([\phi] \cap [\mathbf{EX}Z])$
  The greatest set Z s.t. every state z in Z satisfies ϕ and reaches another state in Z in one step.

We can characterize fair **EG** (aka "$\mathbf{E}_f\mathbf{G}$") similarly:

- Theorem (Emerson & Lei): $[\mathbf{E}_f\mathbf{G}\phi] = \nu Z.([\phi] \cap \bigcap_{F_i \in FT}[\mathbf{EX}\ \mathbf{E}(Z\mathbf{U}(Z \cap F_i))])$
  The greatest set Z s.t. every state z in Z satisfies ϕ and, for every set $F_i \in FT$, z reaches a state in $F_i \cap Z$ by means of a non-trivial path that lies in Z.





Emerson-Lei Algorithm

Recall: $[\mathbf{E}_f\mathbf{G}\phi] = \nu Z.([\phi] \cap \bigcap_{F_i \in FT}[\mathbf{EX}\ \mathbf{E}(Z\mathbf{U}(Z \cap F_i))])$

```
state_set Check_FairEG( state_set [φ]) {
    Z′ := [φ];
    repeat
        Z:= Z′;
        for each F_i in FT
            Y:= Check_EU(Z,F_i ∩ Z);
            Z′ := Z′ ∩ PreImage(Y));
        end for;
    until (Z′ = Z);
    return Z;
}
```

Implementation of the above formula

Emerson-Lei Algorithm

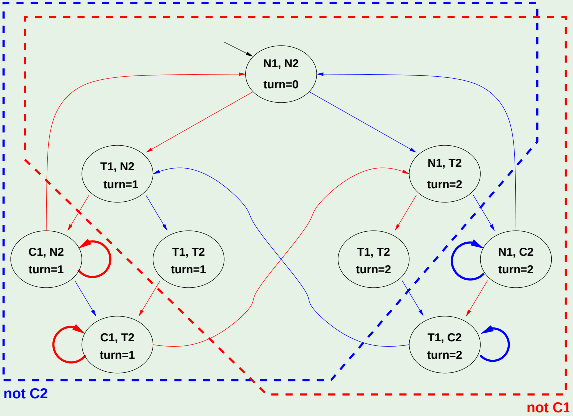
Recall: $[\mathbf{E}_f\mathbf{G}\phi] = \nu Z.([\phi] \cap \bigcap_{F_i \in FT}[\mathbf{EX}\ \mathbf{E}(Z\mathbf{U}(Z \cap F_i))])$

```
state_set Check_FairEG( state_set [ϕ]) {
    Z' := [ϕ];
    repeat
        Z:= Z';
        for each F_i in FT
            Y:= Check_EU(Z',F_i ∩ Z');
            Z' := Z' ∩ PreImage(Y));
        end for;
    until (Z' = Z);
    return Z;
}
```

Slight improvement: do not consider states in $Z \setminus Z'$

Example: Check_FairEG

F := { { not C1},{not C2}}

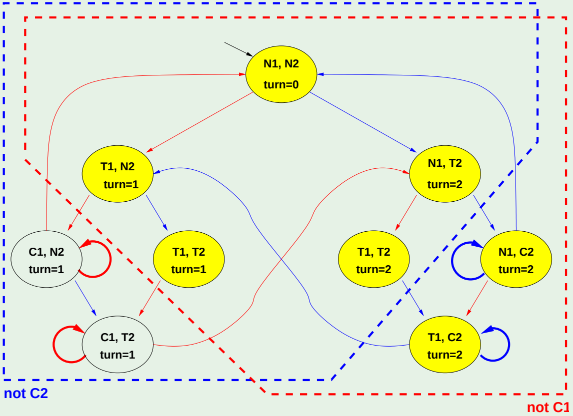


$[\mathbf{E}_f\mathbf{G}\neg C_1]$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}

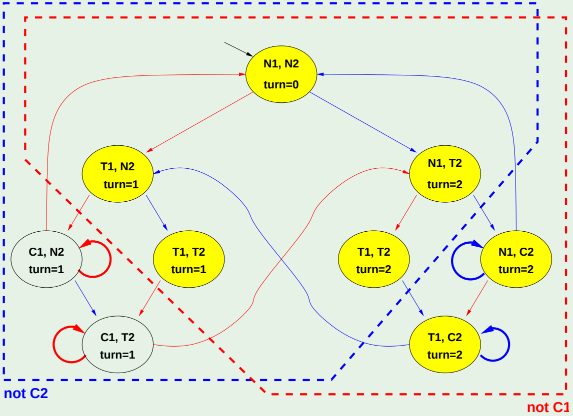


$[\mathbf{E}_f\mathbf{G}\neg C_1]$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}



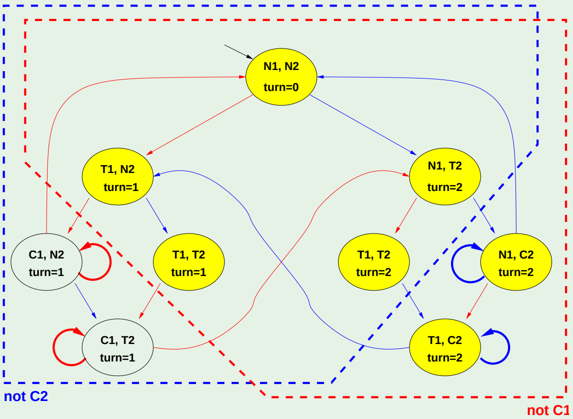
$[\mathbf{E}_f\mathbf{G}\neg C_1]$

$[\mathbf{E}_f\mathbf{G}g] = \nu Z.[g] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_1))] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_2))]$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}



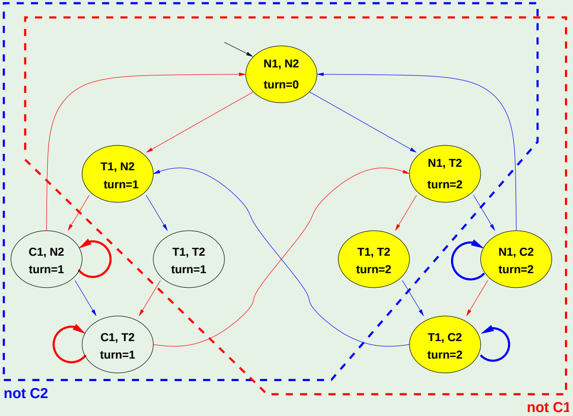
$[\mathbf{E}_f\mathbf{G}\neg C_1]$

$[\mathbf{E}_f\mathbf{G}g] = \nu Z.[g] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_1))] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_2))]$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}



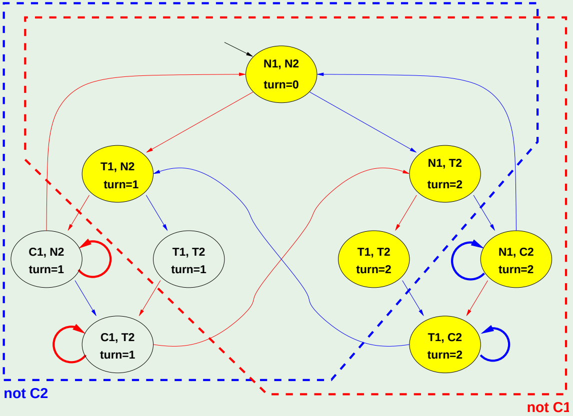
$[\mathbf{E}_f\mathbf{G}\neg C_1]$

$[\mathbf{E}_f\mathbf{G}g] = \nu Z.[g] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_1))] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_2))]$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}



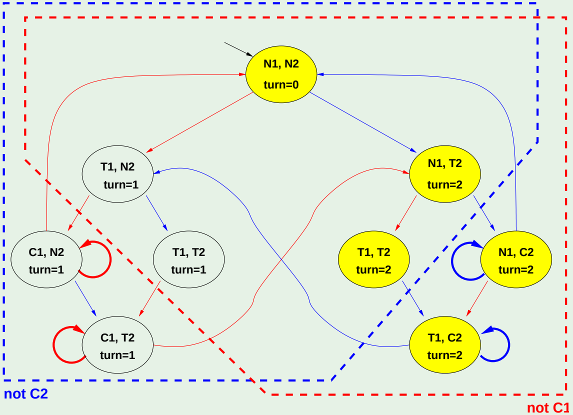
$[\mathbf{E}_f\mathbf{G}\neg C_1]$

$[\mathbf{E}_f\mathbf{G}g] = \nu Z.[g] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_1))] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_2))]$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}



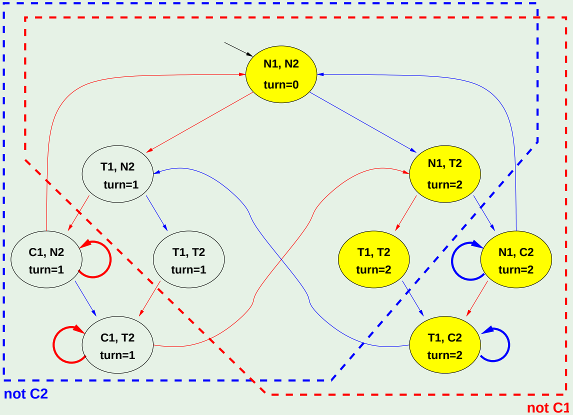
$[\mathbf{E}_f\mathbf{G}\neg C_1]$

$$[\mathbf{E}_f\mathbf{G}g] = \nu Z.[g] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_1))] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_2))]$$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}



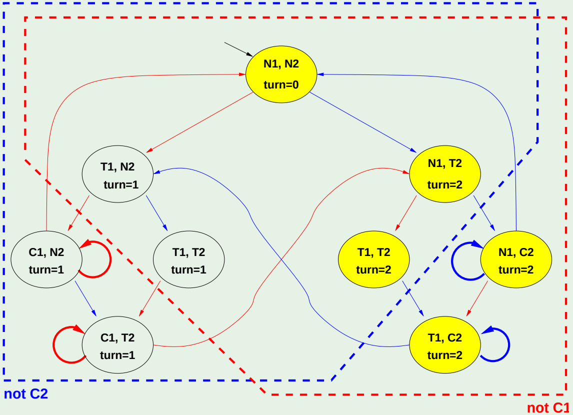
$[\mathbf{E}_f\mathbf{G}\neg C_1]$

$[\mathbf{E}_f\mathbf{G}g] = \nu Z.[g] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_1))] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_2))]$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}



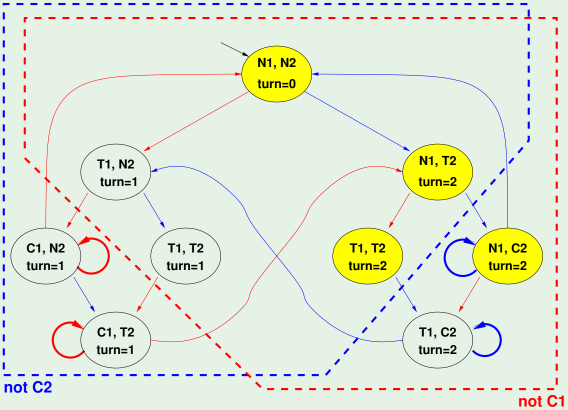
$[\mathbf{E}_f\mathbf{G}\neg C_1]$

$[\mathbf{E}_f\mathbf{G}g] = \nu Z.[g] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_1))] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_2))]$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}



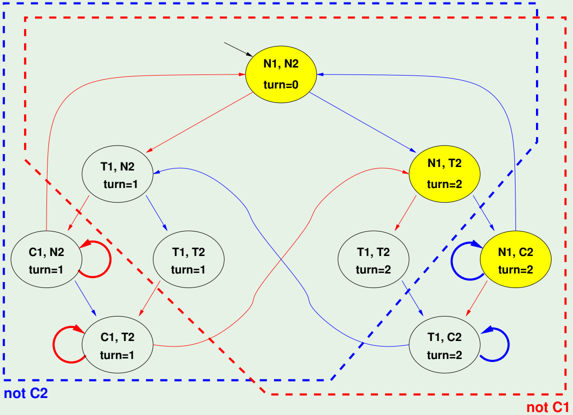
$[\mathbf{E}_f\mathbf{G}\neg C_1]$

$[\mathbf{E}_f\mathbf{G}g] = \nu Z.[g] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_1))] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_2))]$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}



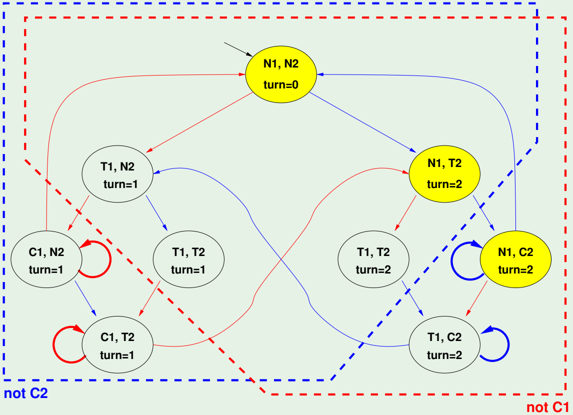
$[\mathbf{E}_f\mathbf{G}\neg C_1]$

$[\mathbf{E}_f\mathbf{G}g] = \nu Z.[g] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_1))] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_2))]$

Fixpoint reached

Example: Check_FairEG

F := { { not C1},{not C2}}



$[\mathbf{E}_f\mathbf{G}\neg C_1]$

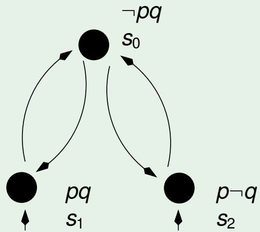
$[\mathbf{E}_f\mathbf{G}g] = \nu Z.[g] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_1))] \cap [\mathbf{EXE}(Z\mathbf{U}(Z \cap F_2))]$

Fixpoint reached

Outline

Ex: CTL Model Checking

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



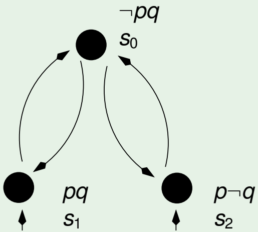
(*a*) Rewrite φ into an equivalent formula φ' expressed in terms of **EX**, **EG**, **EU**/**EF** only.

(*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.

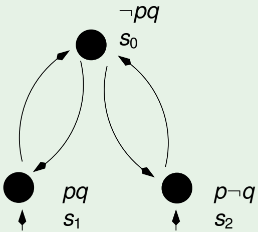
[Solution: $\varphi' = \neg\mathbf{EF}\neg((\neg p \vee \neg q) \vee \mathbf{EG}q) = \neg\mathbf{EF}((p \wedge q) \wedge \neg\mathbf{EG}q)$]

(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:

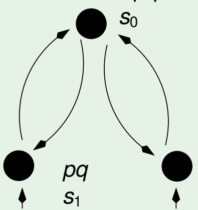
$$
\begin{array}{lcllcl}
[p] & = & \{s_1, s_2\} & [\neg\mathbf{EG}q] & = & \{s_2\} \\
[q] & = & \{s_0, s_1\} & [((p \wedge q) \wedge \neg\mathbf{EG}q)] & = & \{\} \\
[(p \wedge q)] & = & \{s_1\} & [\mathbf{EF}((p \wedge q) \wedge \neg\mathbf{EG}q)] & = & \{\} \\
[\mathbf{EG}q] & = & \{s_0, s_1\} & [\neg\mathbf{EF}((p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

]

(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 $\varphi \stackrel{\text{def}}{=} \mathbf{AG}((p \wedge q) \rightarrow \mathbf{EG}q)$.



(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:

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

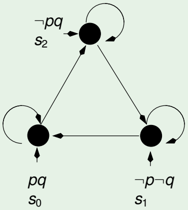
]

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



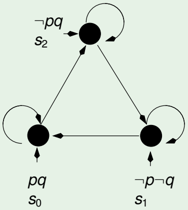
(a) Rewrite φ into an equivalent formula φ' expressed in terms of **EX**, **EG**, **EU**/**EF** only.

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



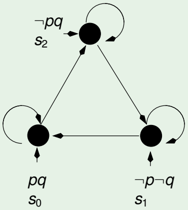
(a) Rewrite φ into an equivalent formula φ' expressed in terms of **EX**, **EG**, **EU**/**EF** only.
[Solution: $\varphi' = \mathbf{AG}(\mathbf{AF}p \rightarrow \mathbf{AF}q) = \neg\mathbf{EF}\neg(\neg\mathbf{EG}\neg p \rightarrow \neg\mathbf{EG}\neg q) = \neg\mathbf{EF}(\neg\mathbf{EG}\neg p \wedge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

(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:

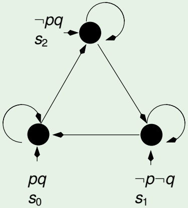
$$
\begin{array}{lcllcl}
[p] & = & \{s_0\} & [\neg q] & = & \{s_1\} \\
[\neg p] & = & \{s_1, s_2\} & [\mathbf{EG}\neg q] & = & \{s_1\} \\
[\mathbf{EG}\neg p] & = & \{s_1, s_2\} & [\neg\mathbf{EG}\neg p \wedge \mathbf{EG}\neg q] & = & \{\} \\
[\neg\mathbf{EG}\neg p] & = & \{s_0\} & [\mathbf{EF}(\neg\mathbf{EG}\neg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

]

(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:

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

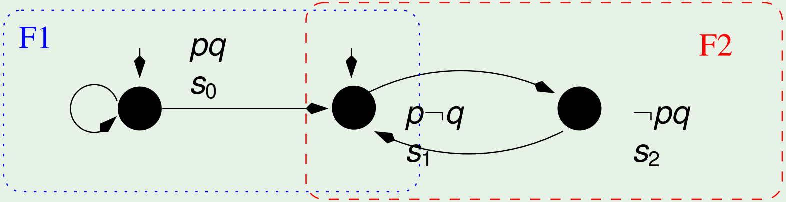
]

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

[Solution: Yes, $\{s_0, s_1, s_2\} \subseteq [\varphi']$.]

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



For each of the following facts, say if it is true or false in CTL.

(a) $M \models \mathbf{AF}\neg p$

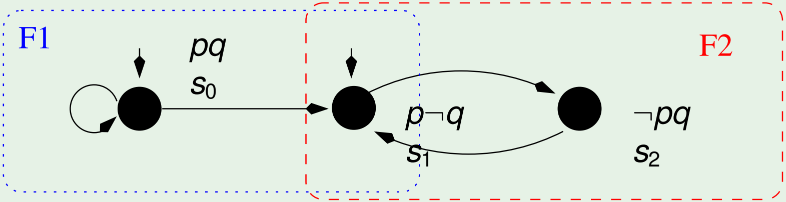
(b) $M \models \mathbf{A}(p\mathbf{U}\neg q)$

(c) $M \models \mathbf{AX}\neg q$

(d) $M \models \mathbf{AGAF}\neg p$

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



For each of the following facts, say if it is true or false in CTL.

(*a*) $M \models \mathbf{AF}\neg p$
[Solution: true]

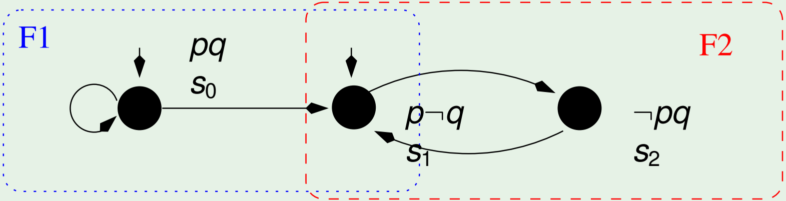
(*b*) $M \models \mathbf{A}(p\mathbf{U}\neg q)$

(*c*) $M \models \mathbf{AX}\neg q$

(*d*) $M \models \mathbf{AGAF}\neg p$

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



For each of the following facts, say if it is true or false in CTL.

(*a*) $M \models \mathbf{AF}\neg p$
[Solution: true]

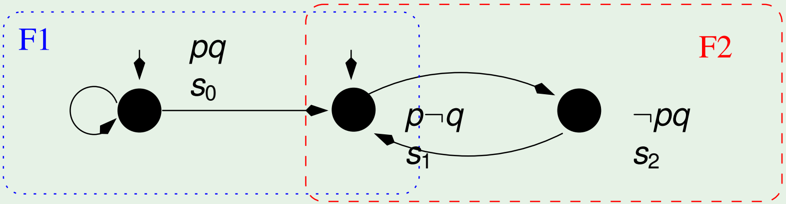
(*b*) $M \models \mathbf{A}(p\mathbf{U}\neg q)$
[Solution: true]

(*c*) $M \models \mathbf{AX}\neg q$

(*d*) $M \models \mathbf{AGAF}\neg p$

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



For each of the following facts, say if it is true or false in CTL.

(*a*) $M \models \mathbf{AF}\neg p$
[Solution: true]

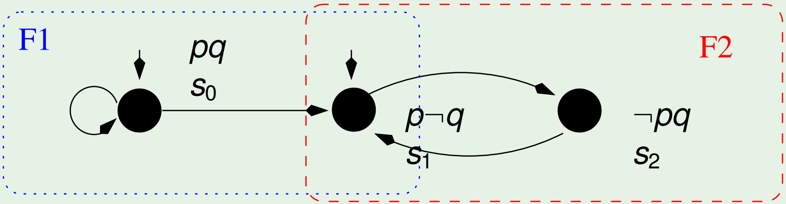
(*b*) $M \models \mathbf{A}(p\mathbf{U}\neg q)$
[Solution: true]

(*c*) $M \models \mathbf{AX}\neg q$
[Solution: false]

(*d*) $M \models \mathbf{AGAF}\neg p$

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



For each of the following facts, say if it is true or false in CTL.

(*a*) $M \models \mathbf{AF}\neg p$
[Solution: true]

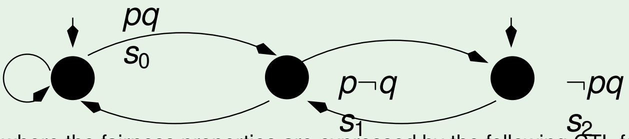
(*b*) $M \models \mathbf{A}(p\mathbf{U}\neg q)$
[Solution: true]

(*c*) $M \models \mathbf{AX}\neg q$
[Solution: false]

(*d*) $M \models \mathbf{AGAF}\neg p$
[Solution: true]

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



where the fairness properties are expressed by the following CTL formula: $\mathbf{AGAF}\neg q$.

For each of the following facts, say if it is true or false in CTL.

(a) $M \models \mathbf{EF}(p \wedge q)$

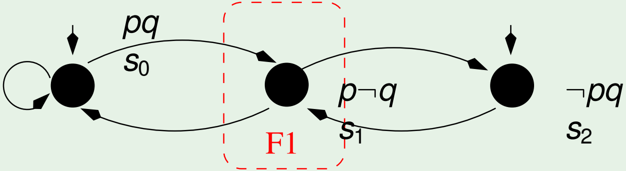
(b) $M \models \mathbf{AGAF}p$

(c) $M \models \mathbf{AF}\neg q$

(d) $M \models \mathbf{AG}(\neg p \vee \neg q)$

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



For each of the following facts, say if it is true or false in CTL.

(a) $M \models \mathbf{EF}(p \land q)$

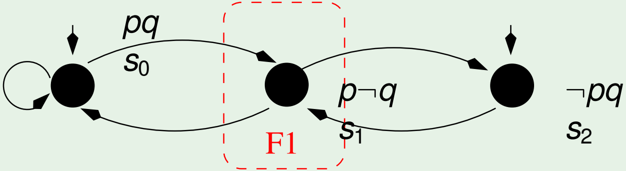
(b) $M \models \mathbf{AGAF}p$

(c) $M \models \mathbf{AF}\neg q$

(d) $M \models \mathbf{AG}(\neg p \lor \neg q)$

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



For each of the following facts, say if it is true or false in CTL.

(a) $M \models \mathbf{EF}(p \wedge q)$
[Solution: true]

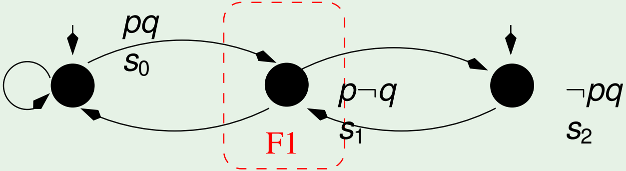
(b) $M \models \mathbf{AGAF}p$

(c) $M \models \mathbf{AF}\neg q$

(d) $M \models \mathbf{AG}(\neg p \vee \neg q)$

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



For each of the following facts, say if it is true or false in CTL.

(a) $M \models \mathbf{EF}(p \wedge q)$
[Solution: true]

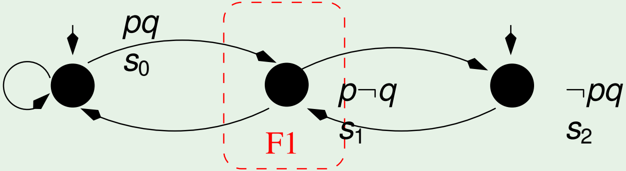
(b) $M \models \mathbf{AGAF}p$
[Solution: true]

(c) $M \models \mathbf{AF}\neg q$

(d) $M \models \mathbf{AG}(\neg p \vee \neg q)$

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



For each of the following facts, say if it is true or false in CTL.

(a) $M \models \mathbf{EF}(p \land q)$
[Solution: true]

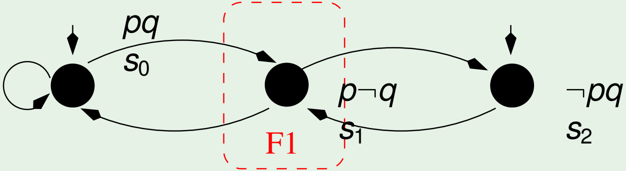
(b) $M \models \mathbf{AGAF}p$
[Solution: true]

(c) $M \models \mathbf{AF}\neg q$
[Solution: true]

(d) $M \models \mathbf{AG}(\neg p \lor \neg q)$

Ex: Fair CTL Model Checking

Consider the following *fair* Kripke Model M:



For each of the following facts, say if it is true or false in CTL.

(a) $M \models \mathbf{EF}(p \wedge q)$
[Solution: true]

(b) $M \models \mathbf{AGAF}p$
[Solution: true]

(c) $M \models \mathbf{AF}\neg q$
[Solution: true]

(d) $M \models \mathbf{AG}(\neg p \vee \neg q)$
[Solution: false]