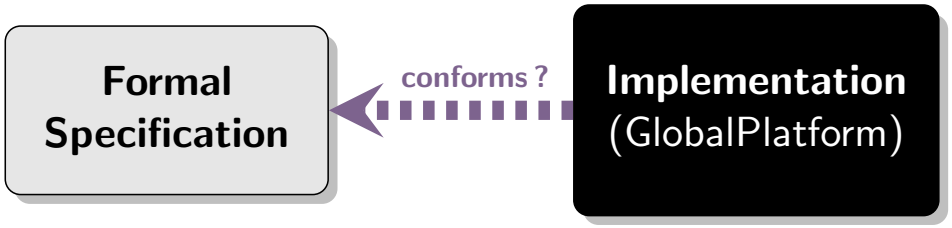
Testing Temporal Patterns

(Test Purpose Automata)

Safouan TAHA

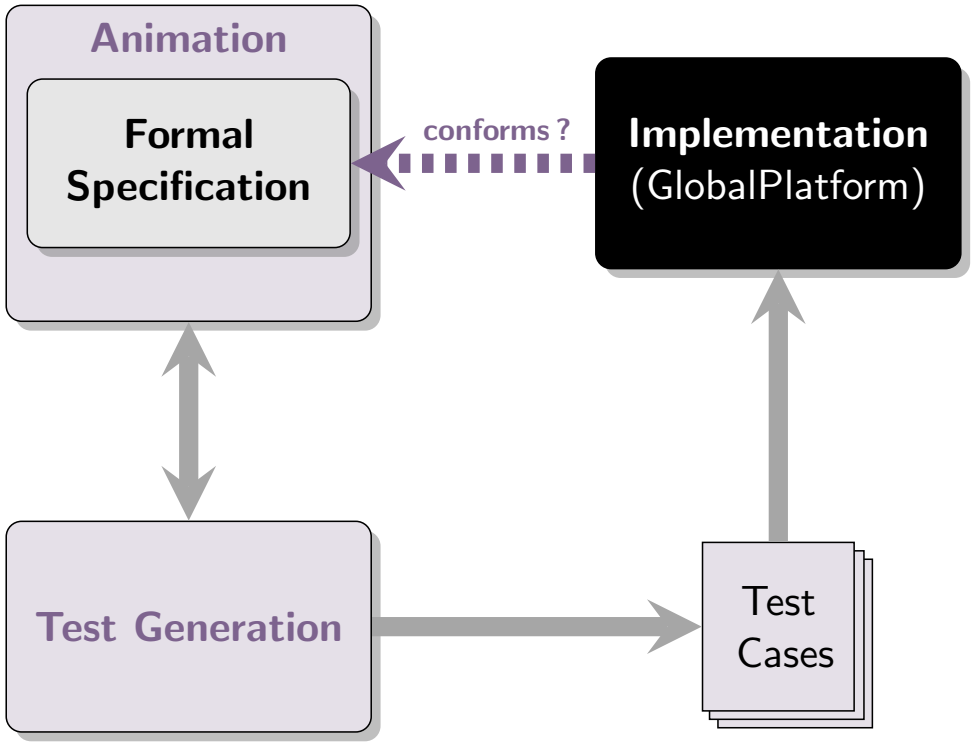
LRI – CentraleSupelec

July 8^{th} 2016



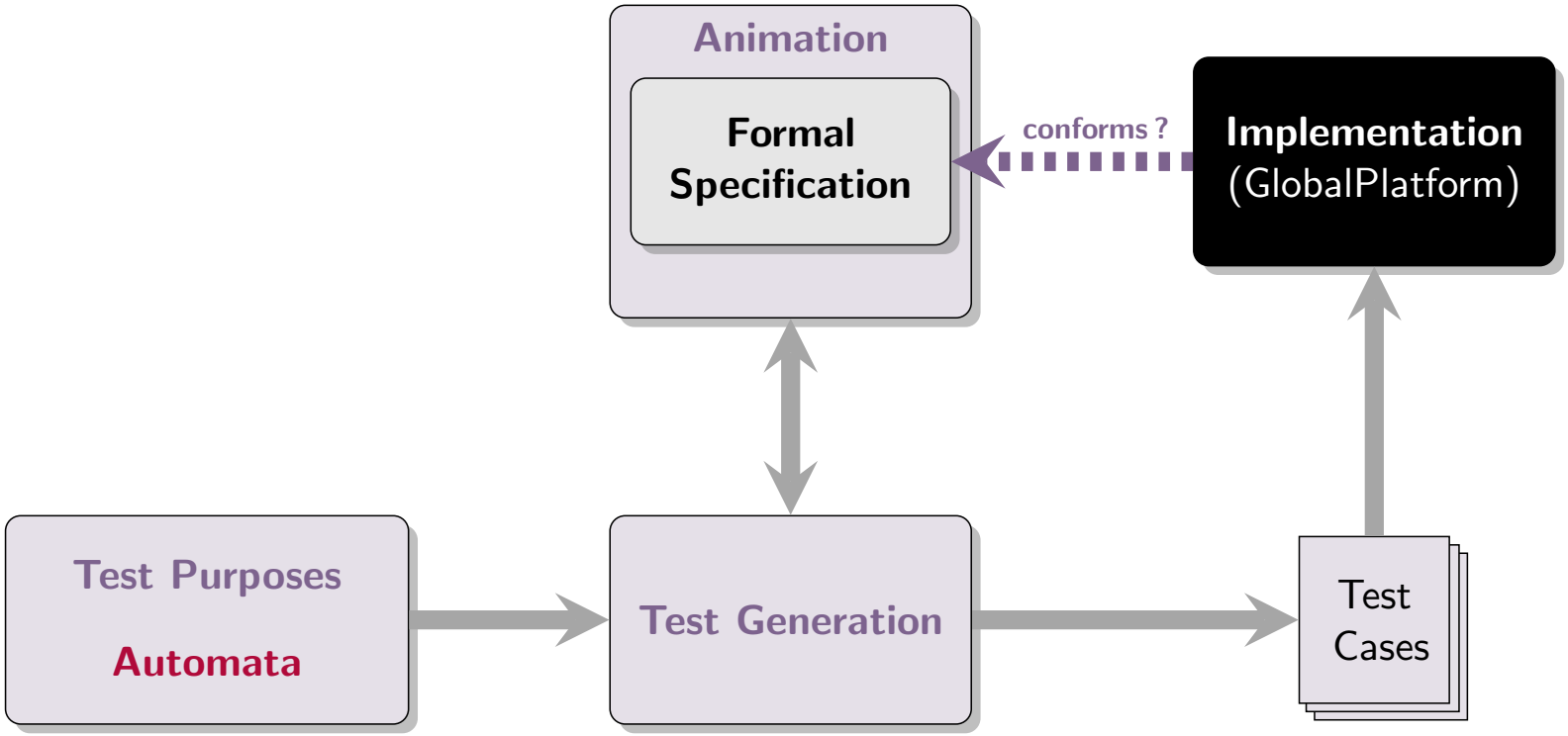
Context : Model-Based Testing

- Conformance testing : functional testing (black box)



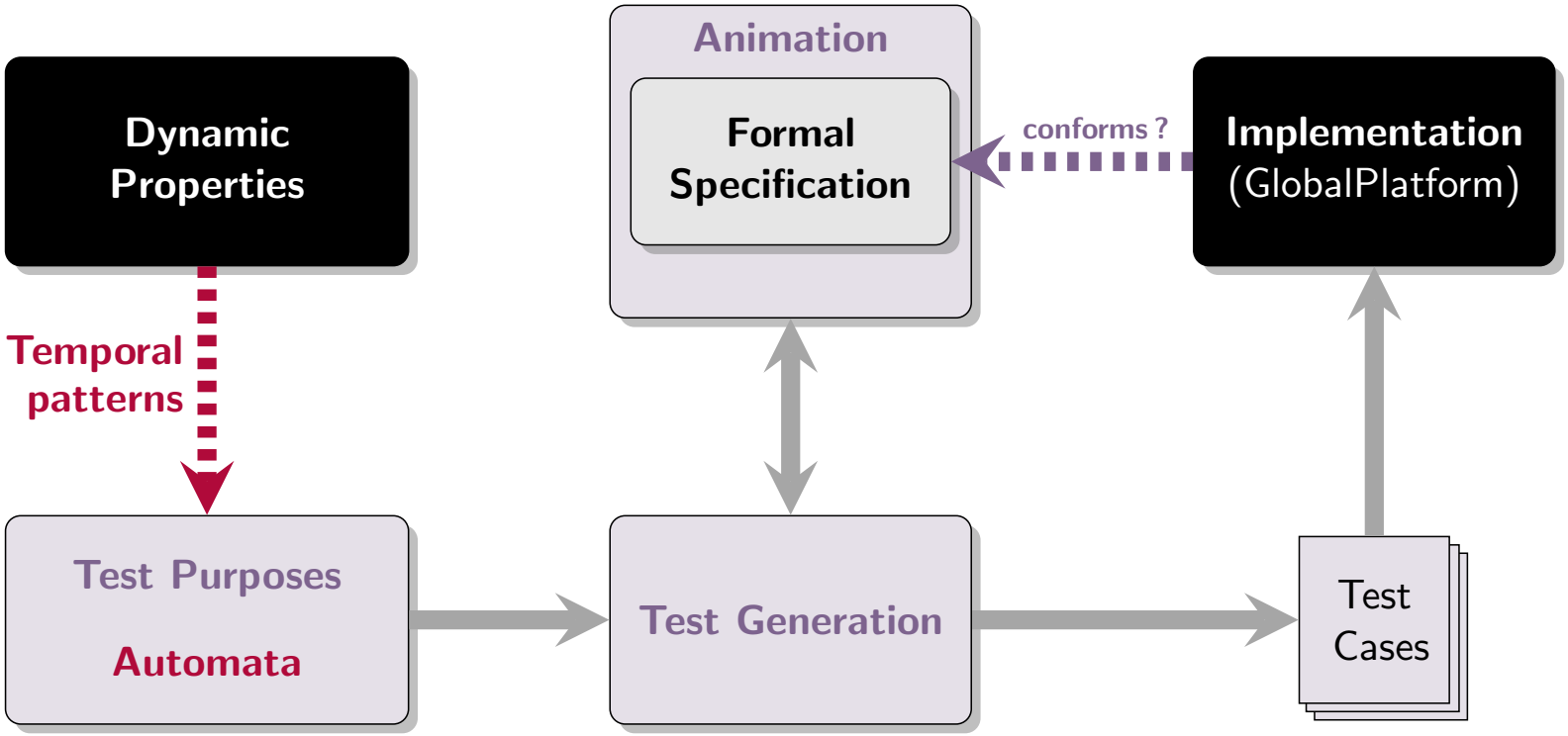
Context : Model-Based Testing

- Conformance testing : functional testing (black box)
- Generation/Animation of test cases



Context : Model-Based Testing

- Conformance testing : functional testing (black box)
- Generation/Animation of test cases
- Test Purposes automata



Objectives of Temporal patterns

1. Specify Dynamic Properties : Security, Safety, Liveness
2. Generate (automatically) Test Purposes Automata

Plan

Propositional ~~Linear Temporal~~ Logic (~~LTL~~)

$$\phi \quad ::= \quad true \quad | \quad a \quad | \quad \phi \quad \wedge \quad \phi \quad | \quad \neg \phi$$

where $a \in AP$

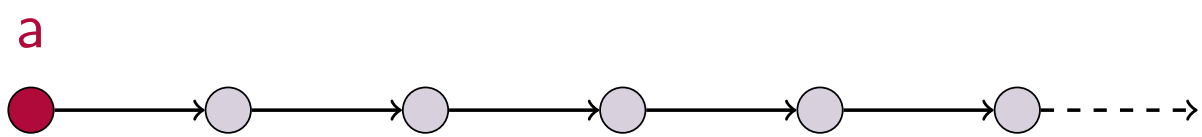
Propositional Linear Temporal Logic (LTL)

$$\phi \quad ::= \quad true \quad | \quad a \quad | \quad \phi \quad \wedge \quad \phi \quad | \quad \neg \phi \quad | \quad \bigcirc \phi$$

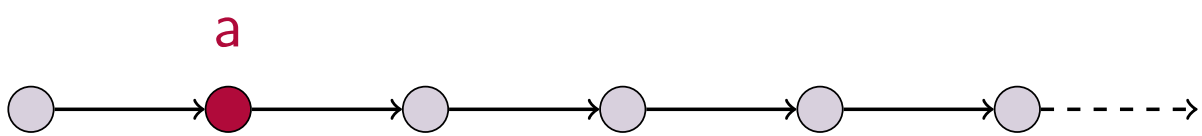
where $a \in AP$

$\bigcirc$ (next)

$\sigma \models a$



$\sigma \models \bigcirc a$



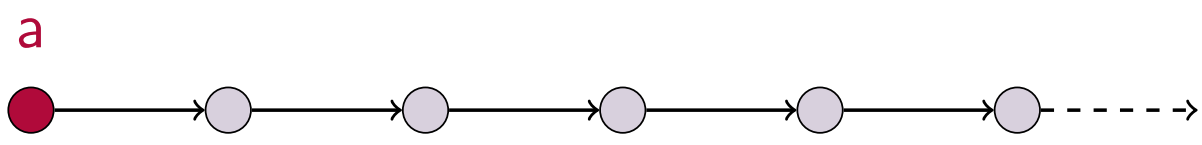
Propositional Linear Temporal Logic (LTL)

$$\phi \quad ::= \quad true \quad | \quad a \quad | \quad \phi \quad \wedge \quad \phi \quad | \quad \neg \phi \quad | \quad \bigcirc \phi \quad | \quad \Box \phi$$

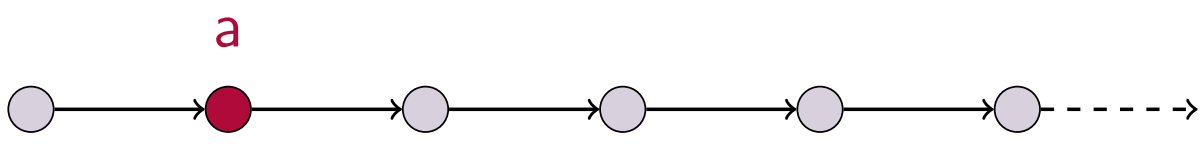
where $a \in AP$

$\bigcirc$ (next) $\Box$ (always)

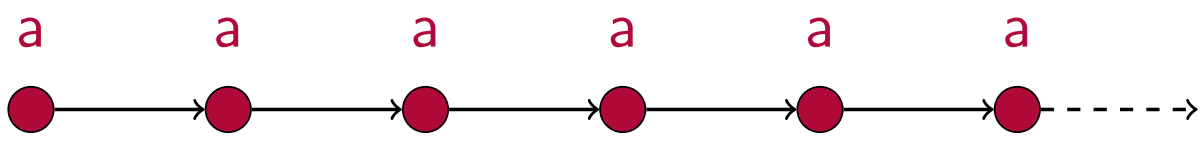
$\sigma \models a$



$\sigma \models \bigcirc a$



$\sigma \models \Box a$

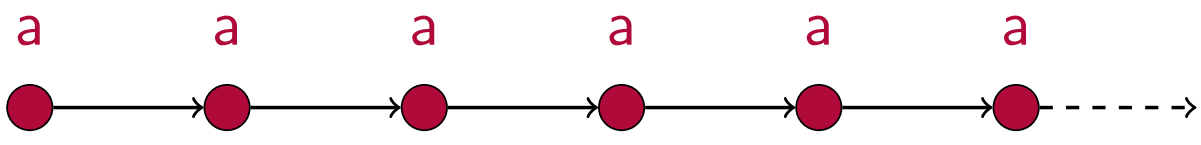


LTL : Derived Temporal Operators

$\Box\phi$

(always)

$\sigma \models \Box\ a$

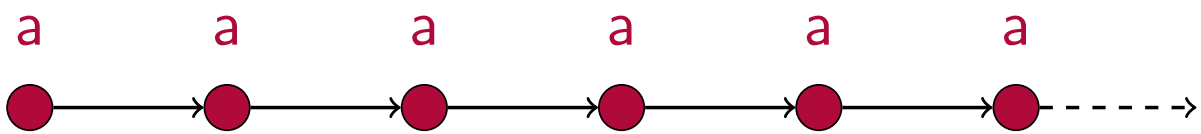


LTL : Derived Temporal Operators

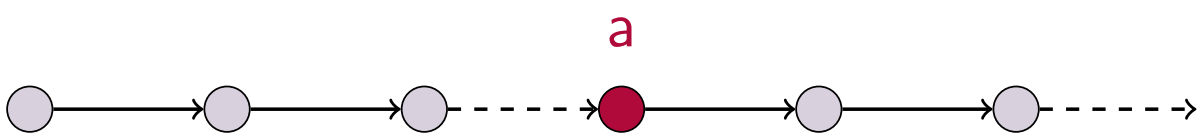
$\Box\phi$ (always)

$\Diamond\phi \equiv \neg\Box\neg\phi$ (eventually)

$\sigma \models \Box\ a$



$\sigma \models \Diamond\ a$

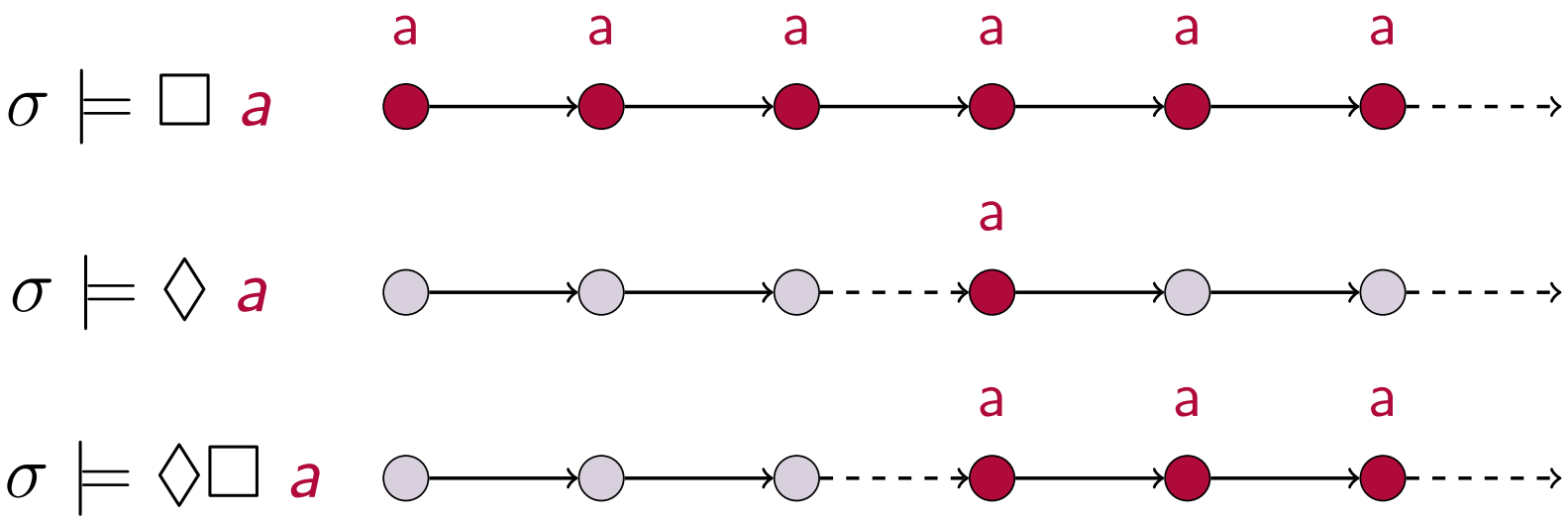


LTL : Derived Temporal Operators

$\Box\phi$ (always)

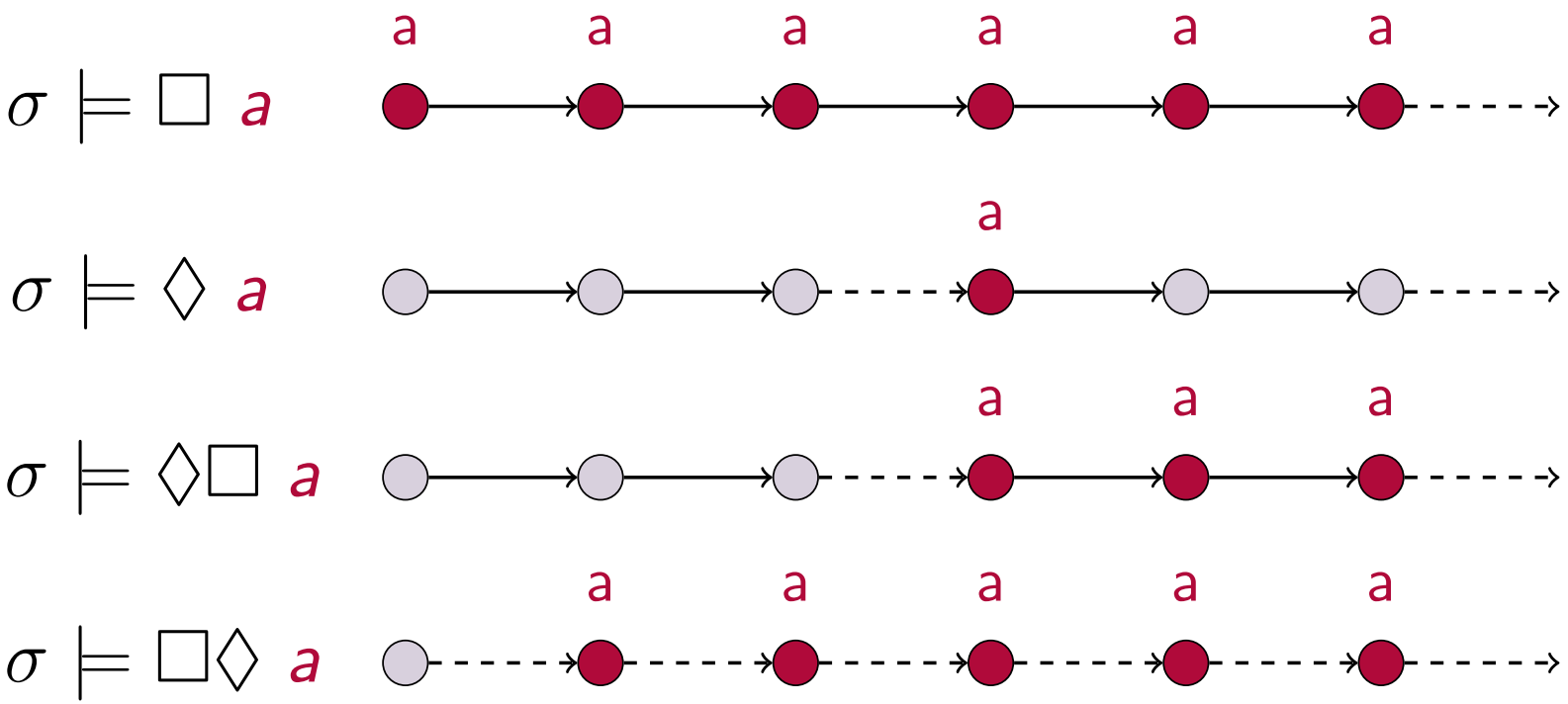
$\Diamond\phi \equiv \neg\Box\neg\phi$ (eventually)

$\Diamond\Box\phi$ (persistence)



LTL : Derived Temporal Operators

$\Box\phi$ (always) $\quad \Diamond\phi \equiv \neg\Box\neg\phi$ (eventually) $\quad \Diamond\Box\phi$ (persistence) $\quad \Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$ (infinitely many)



$\{\ i \mid a \in \sigma(i)\ \}$ is infinite

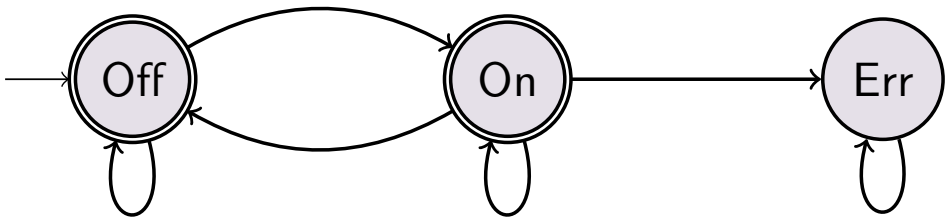
LTL : Example of Temporal Properties 1/2

- Safety :
  - cooling : $\Box\neg(temp_{high} \wedge cooling_{low})$
  - elevator : $\Box(moving \Rightarrow doors_{closed})$
  - traffic light : $\Box(yellow \Rightarrow \bigcirc red)$

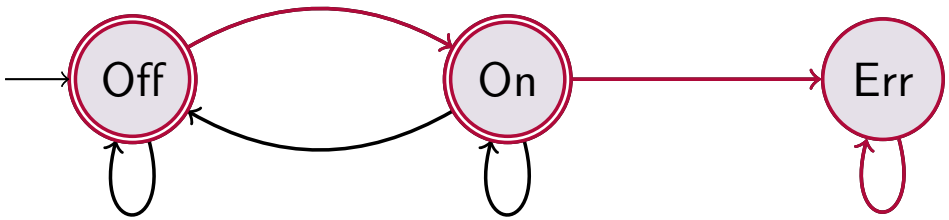
LTL : Example of Temporal Properties 1/2

- Safety :
  - cooling : $\Box\neg(temp_{high} \wedge cooling_{low})$
  - elevator : $\Box(moving \Rightarrow doors_{closed})$
  - traffic light : $\Box(yellow \Rightarrow \bigcirc red)$
- Liveness :
  - progress : $\Diamond\ progress$
  - response : $\Box(try\_to\_send \Rightarrow \Diamond delivered)$
  - termination : $\Diamond\Box\ terminated$

LTL : properties of a trace



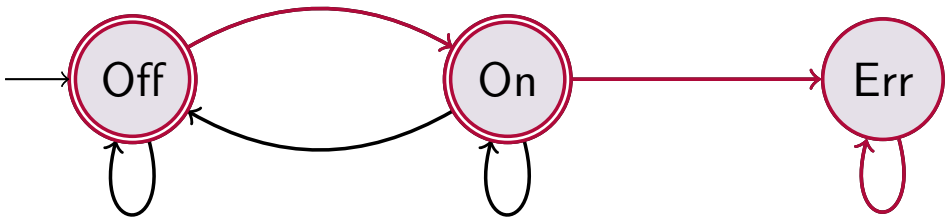
LTL : properties of a trace



have a path $\pi = Off\ On\ Err\ Err\ Err \ldots = Off\ On\ Err^{\omega}$

- $\pi \models \bigcirc On$

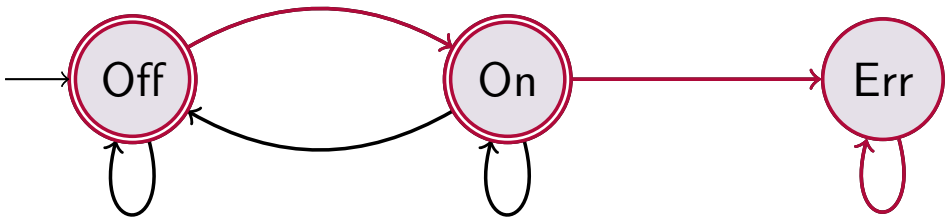
LTL : properties of a trace



have a path $\pi = Off\ On\ Err\ Err\ Err \ldots = Off\ On\ Err^{\omega}$

- $\pi \models \bigcirc On$
- $\pi \models \bigcirc \bigcirc Err$

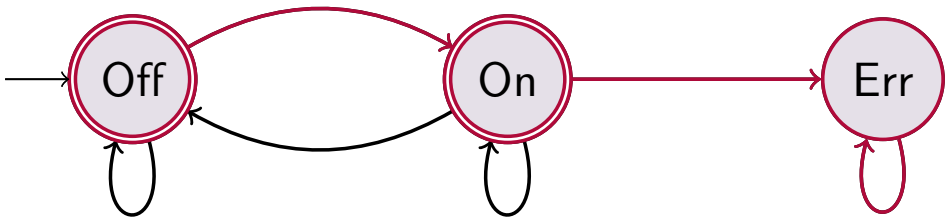
LTL : properties of a trace



have a path $\pi = Off\ On\ Err\ Err\ Err \ldots = Off\ On\ Err^{\omega}$

- $\pi \models \bigcirc On$
- $\pi \models \bigcirc \bigcirc Err$
- $\pi \models (Off \vee On)\ \cup\ Err$

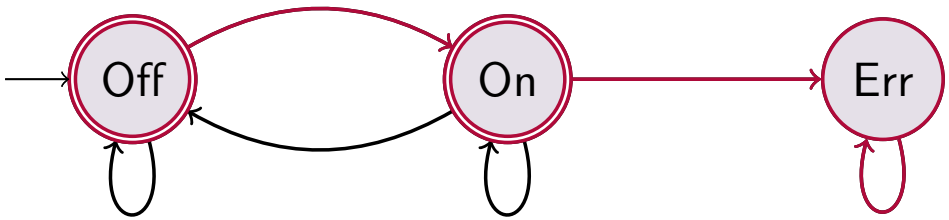
LTL : properties of a trace



have a path $\pi = Off\ On\ Err\ Err\ Err \ldots = Off\ On\ Err^{\omega}$

- $\pi \models \bigcirc On$
- $\pi \models \bigcirc \bigcirc Err$
- $\pi \models (Off \vee On)\ \cup\ Err$
- $\pi \models \Box(Err \Rightarrow \bigcirc Err)$

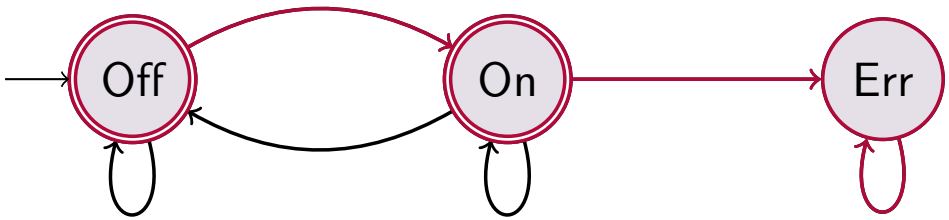
LTL : properties of a trace



have a path $\pi = Off\ On\ Err\ Err\ Err \ldots = Off\ On\ Err^{\omega}$

- $\pi \models \bigcirc On$
- $\pi \models \bigcirc \bigcirc Err$
- $\pi \models (Off \vee On)\ \cup\ Err$
- $\pi \models \Box(Err \Rightarrow \bigcirc Err)$
- $\pi \models \Box(Err \Rightarrow \Box Err)$

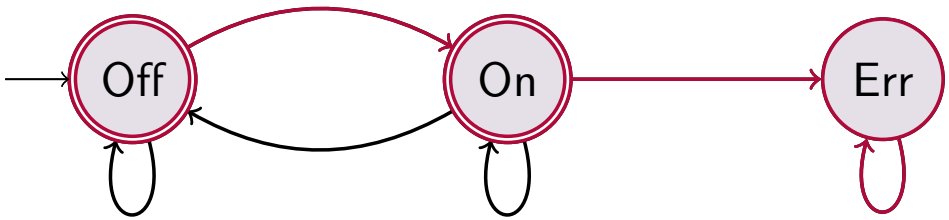
LTL : properties of a trace



have a path $\pi = Off\ On\ Err\ Err\ Err \ldots = Off\ On\ Err^{\omega}$

- $\pi \models \bigcirc On$
- $\pi \models \bigcirc \bigcirc Err$
- $\pi \models (Off \vee On)\ \cup\ Err$
- $\pi \models \Box(Err \Rightarrow \bigcirc Err)$
- $\pi \models \Box(Err \Rightarrow \Box Err)$
- $\pi \models \Diamond\Box Err$ $(persistence)$

LTL : properties of a trace



have a path $\pi = Off\ On\ Err\ Err\ Err \ldots = Off\ On\ Err^{\omega}$

- $\pi \models \bigcirc On$
- $\pi \models \bigcirc \bigcirc Err$
- $\pi \models (Off \vee On)\ \cup\ Err$
- $\pi \models \Box(Err \Rightarrow \bigcirc Err)$
- $\pi \models \Box(Err \Rightarrow \Box Err)$
- $\pi \models \Diamond\Box Err$ (*persistence*)
- $\pi \models \bigcirc \bigcirc \Box Err$

LTL : Laws

- negation :

$$\neg\Box\phi \equiv \Diamond \neg \phi, \qquad \neg\Diamond\phi \equiv \Box \neg \phi, \qquad \neg \bigcirc \phi \equiv \bigcirc \neg \phi$$

LTL : Laws

- negation :

$$\neg \overset{\curvearrowright}{\Box} \phi \equiv \Diamond \neg \phi, \qquad \neg \overset{\curvearrowright}{\Diamond} \phi \equiv \Box \neg \phi, \qquad \neg \overset{\curvearrowright}{\bigcirc} \phi \equiv \bigcirc \neg \phi$$

LTL : Laws

- negation :

$$\neg \overset{\curvearrowright}{\Box}\, \phi \equiv \Diamond \neg\, \phi, \qquad \neg \overset{\curvearrowright}{\Diamond}\, \phi \equiv \Box \neg\, \phi, \qquad \neg \overset{\curvearrowright}{\bigcirc}\, \phi \equiv \bigcirc \neg\, \phi$$

Corollary

$\boxtimes_i \in \{\Box, \Diamond, \bigcirc\}$.

$$\neg \boxtimes_1 \boxtimes_2 \ldots \boxtimes_{n-1} \boxtimes_n \phi \equiv \overset{\curvearrowright}{\boxtimes}_1 \overset{\curvearrowright}{\boxtimes}_2 \ldots \overset{\curvearrowright}{\boxtimes}_{n-1} \overset{\curvearrowright}{\boxtimes}_n \neg \phi$$

LTL : Laws

- negation :

$$\neg \overset{\curvearrowright}{\Box}\, \phi \equiv \Diamond \neg\, \phi, \qquad \neg \overset{\curvearrowright}{\Diamond}\, \phi \equiv \Box \neg\, \phi, \qquad \neg \overset{\curvearrowright}{\bigcirc}\, \phi \equiv \bigcirc \neg\, \phi$$

Corollary

$\boxtimes_i \in \{\Box, \Diamond, \bigcirc\}$.

$$\neg\, \boxtimes_1 \boxtimes_2 \ldots \boxtimes_{n-1} \boxtimes_n\, \phi \equiv \overset{\curvearrowright}{\boxtimes}_1 \overset{\curvearrowright}{\boxtimes}_2 \ldots \overset{\curvearrowright}{\boxtimes}_{n-1} \overset{\curvearrowright}{\boxtimes}_n \neg\, \phi$$

- next :

$$\Diamond \bigcirc \phi \overset{?}{\equiv} \bigcirc \Diamond\, \phi$$

LTL : Laws

- negation :

$$\neg \overset{\curvearrowright}{\Box}\, \phi \equiv \Diamond \neg\, \phi, \qquad \neg \overset{\curvearrowright}{\Diamond}\, \phi \equiv \Box \neg\, \phi, \qquad \neg \overset{\curvearrowright}{\bigcirc}\, \phi \equiv \bigcirc \neg\, \phi$$

Corollary

$\boxtimes_i \in \{\Box, \Diamond, \bigcirc\}$.

$$\neg\, \boxtimes_1 \boxtimes_2 \ldots \boxtimes_{n-1} \boxtimes_n\, \phi \equiv \overset{\curvearrowright}{\boxtimes}_1 \overset{\curvearrowright}{\boxtimes}_2 \ldots \overset{\curvearrowright}{\boxtimes}_{n-1} \overset{\curvearrowright}{\boxtimes}_n \neg\, \phi$$

- next :

$$\Diamond \bigcirc \phi \equiv \bigcirc \Diamond\, \phi\,, \qquad \Box \bigcirc \phi \overset{?}{\equiv} \bigcirc \Box\, \phi$$

LTL : Laws

- negation :

$$\neg \overset{\curvearrowright}{\Box}\, \phi \equiv \Diamond \neg\, \phi, \qquad \neg \overset{\curvearrowright}{\Diamond}\, \phi \equiv \Box \neg\, \phi, \qquad \neg \overset{\curvearrowright}{\bigcirc}\, \phi \equiv \bigcirc \neg\, \phi$$

Corollary

$\boxtimes_i \in \{\Box, \Diamond, \bigcirc\}$.

$$\neg\, \boxtimes_1 \boxtimes_2 \ldots \boxtimes_{n-1} \boxtimes_n\, \phi \equiv \overset{\curvearrowright}{\boxtimes}_1 \overset{\curvearrowright}{\boxtimes}_2 \ldots \overset{\curvearrowright}{\boxtimes}_{n-1} \overset{\curvearrowright}{\boxtimes}_n \neg\, \phi$$

- next :

$$\Diamond \bigcirc\, \phi \equiv \bigcirc \Diamond\, \phi\,, \qquad \Box \bigcirc\, \phi \equiv \bigcirc \Box\, \phi$$

LTL : Laws

- negation :

$$\neg \overset{\curvearrowright}{\Box}\, \phi \equiv \Diamond \neg\, \phi, \qquad \neg \overset{\curvearrowright}{\Diamond}\, \phi \equiv \Box \neg\, \phi, \qquad \neg \overset{\curvearrowright}{\bigcirc}\, \phi \equiv \bigcirc \neg\, \phi$$

Corollary

$\boxtimes_i \in \{\Box, \Diamond, \bigcirc\}$.

$$\neg\, \boxtimes_1 \boxtimes_2 \ldots \boxtimes_{n-1} \boxtimes_n\, \phi \equiv \overset{\curvearrowright}{\boxtimes}_1 \overset{\curvearrowright}{\boxtimes}_2 \ldots \overset{\curvearrowright}{\boxtimes}_{n-1} \overset{\curvearrowright}{\boxtimes}_n \neg\, \phi$$

- next :

$$\Diamond \bigcirc\, \phi \equiv \bigcirc \Diamond\, \phi\,, \qquad \Box \bigcirc\, \phi \equiv \bigcirc \Box\, \phi$$

Corollary

$\boxtimes_i \in \{\Box, \Diamond, \neg\}$.

$$\boxtimes_1 \boxtimes_2 \ldots \boxtimes_{n-1} \boxtimes_n \bigcirc\, \phi \equiv \bigcirc\, \boxtimes_1 \boxtimes_2 \ldots \boxtimes_{n-1} \boxtimes_n\, \phi$$

LTL : Laws

$\Box\phi$	$\Diamond\phi \equiv \neg\Box\neg\phi$	$\Diamond\Box\phi$	$\Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$
(always)	(eventually)	(persistence)	(infinitely many)

LTL : Laws

$\Box\phi$ (always) $\quad \Diamond\phi \equiv \neg\Box\neg\phi$ (eventually) $\quad \Diamond\Box\phi$ (persistence) $\quad \Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$ (infinitely many)

- Idempotency : $\Box\Box\ \phi \stackrel{?}{\equiv} \Box\ \phi$

LTL : Laws

$\Box\phi$ (always)

$\Diamond\phi \equiv \neg\Box\neg\phi$ (eventually)

$\Diamond\Box\phi$ (persistence)

$\Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$ (infinitely many)

- Idempotency : $\Box\Box\ \phi \equiv \Box\ \phi$

LTL : Laws

$\Box\phi$ (always)

$\Diamond\phi \equiv \neg\Box\neg\phi$ (eventually)

$\Diamond\Box\phi$ (persistence)

$\Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$ (infinitely many)

- Idempotency : $\Box\Box\ \phi \equiv \Box\ \phi$ $\qquad \Diamond\Diamond\ \phi \overset{?}{\equiv} \Diamond\ \phi$

LTL : Laws

$\Box\phi$	$\Diamond\phi \equiv \neg\Box\neg\phi$	$\Diamond\Box\phi$	$\Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$
(always)	(eventually)	(persistence)	(infinitely many)

- Idempotency : $\Box\Box\ \phi \equiv \Box\ \phi$ $\Diamond\Diamond\ \phi \equiv \Diamond\ \phi$

LTL : Laws

$\Box\phi$ (always) $\quad \Diamond\phi \equiv \neg\Box\neg\phi$ (eventually) $\quad \Diamond\Box\phi$ (persistence) $\quad \Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$ (infinitely many)

- Idempotency : $\Box\Box\ \phi \equiv \Box\ \phi \qquad \Diamond\Diamond\ \phi \equiv \Diamond\ \phi$
- Absorption : $\Diamond\Box\Diamond\ \phi \stackrel{?}{\equiv} \Box\Diamond\ \phi$

LTL : Laws

$\Box\phi$ (always) $\quad \Diamond\phi \equiv \neg\Box\neg\phi$ (eventually) $\quad \Diamond\Box\phi$ (persistence) $\quad \Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$ (infinitely many)

- Idempotency : $\Box\Box\ \phi \equiv \Box\ \phi \qquad \Diamond\Diamond\ \phi \equiv \Diamond\ \phi$
- Absorption : $\Diamond\Box\Diamond\ \phi \equiv \Box\Diamond\ \phi$

LTL : Laws

$\Box\phi$ (always) $\quad \Diamond\phi \equiv \neg\Box\neg\phi$ (eventually) $\quad \Diamond\Box\phi$ (persistence) $\quad \Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$ (infinitely many)

- Idempotency : $\Box\Box\ \phi \equiv \Box\ \phi \qquad \Diamond\Diamond\ \phi \equiv \Diamond\ \phi$
- Absorption : $\Diamond\Box\Diamond\ \phi \equiv \Box\Diamond\ \phi \qquad \Box\Diamond\Box\ \phi \overset{?}{\equiv} \Diamond\Box\ \phi$

LTL : Laws

$\Box\phi$ (always) $\quad \Diamond\phi \equiv \neg\Box\neg\phi$ (eventually) $\quad \Diamond\Box\phi$ (persistence) $\quad \Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$ (infinitely many)

- Idempotency : $\Box\Box\ \phi \equiv \Box\ \phi \qquad \Diamond\Diamond\ \phi \equiv \Diamond\ \phi$
- Absorption : $\Diamond\Box\Diamond\ \phi \equiv \Box\Diamond\ \phi \qquad \Box\Diamond\Box\ \phi \equiv \Diamond\Box\ \phi$

LTL : Laws

$\Box\phi$ (always) $\quad \Diamond\phi \equiv \neg\Box\neg\phi$ (eventually) $\quad \Diamond\Box\phi$ (persistence) $\quad \Box\Diamond\phi \equiv \neg\Diamond\Box\neg\phi$ (infinitely many)

- Idempotency : $\Box\Box\ \phi \equiv \Box\ \phi \qquad \Diamond\Diamond\ \phi \equiv \Diamond\ \phi$
- Absorption : $\Diamond\Box\Diamond\ \phi \equiv \Box\Diamond\ \phi \qquad \Box\Diamond\Box\ \phi \equiv \Diamond\Box\ \phi$

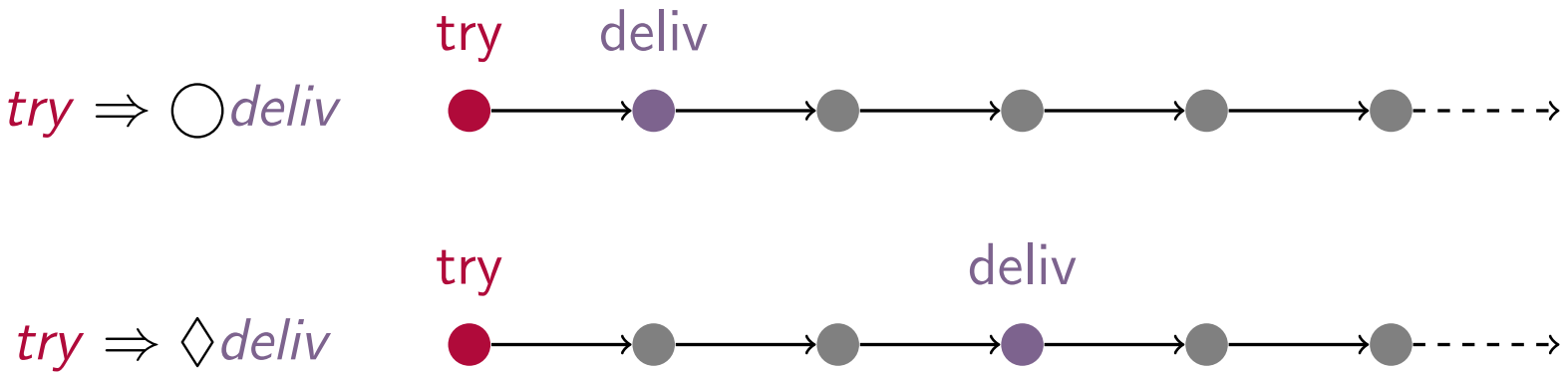
Corollary

$\boxtimes_i \in \{\Box, \Diamond\}$

$$\boxtimes_1 \boxtimes_2 \ldots \boxtimes_{n-1} \boxtimes_n\ \phi \equiv \begin{cases} \Box\ \phi & or \quad \Diamond\ \phi \quad or \\ \Box\Diamond\ \phi & or \quad \Diamond\Box\ \phi \end{cases}$$

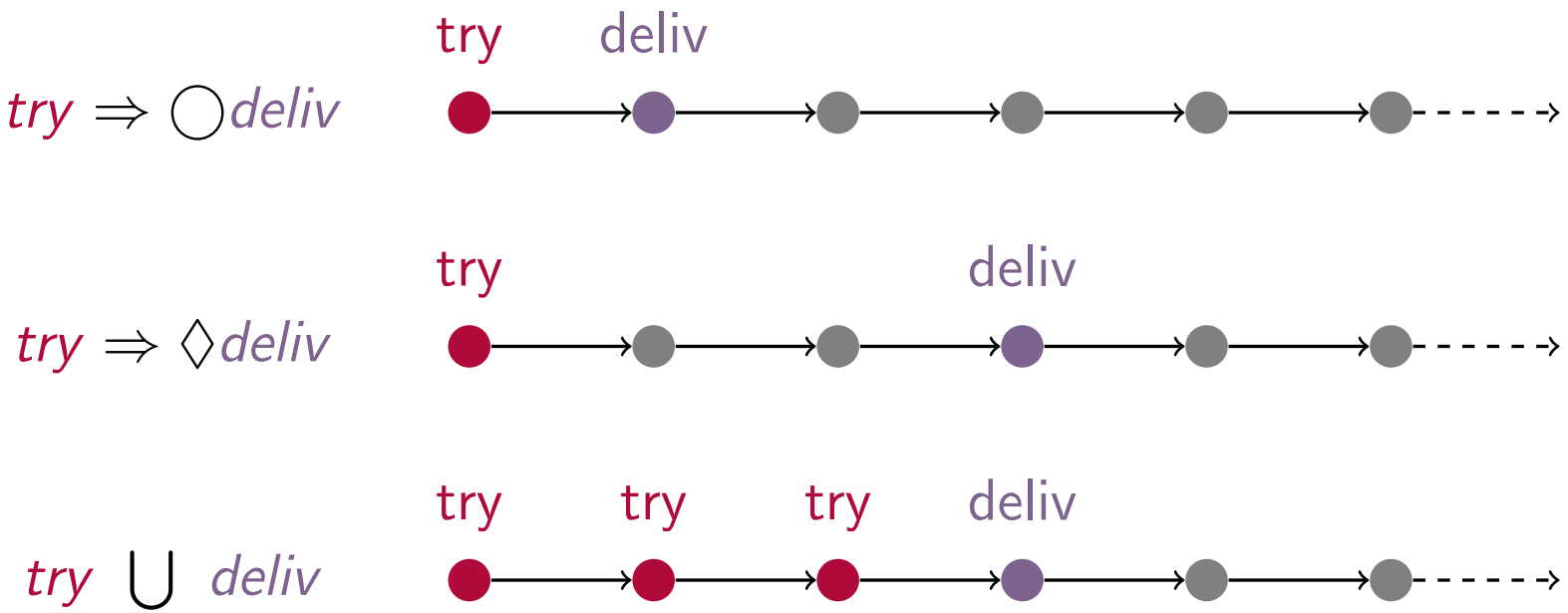
LTL : Until Operator

$$\phi \quad ::= \quad true \mid a \mid \phi \wedge \phi \mid \neg \phi \mid \bigcirc \phi \mid \Box \phi$$



LTL : Until Operator

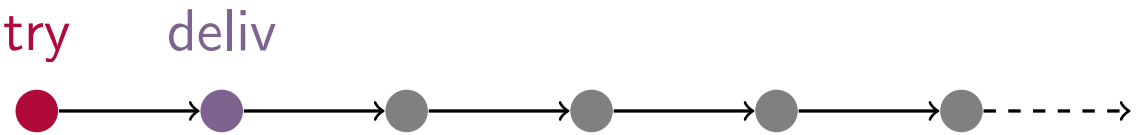
$$\phi \ ::= \ true \mid a \mid \phi \ \wedge \ \phi \mid \neg \phi \mid \bigcirc \phi \mid \Box \phi \mid \phi \bigcup \phi$$



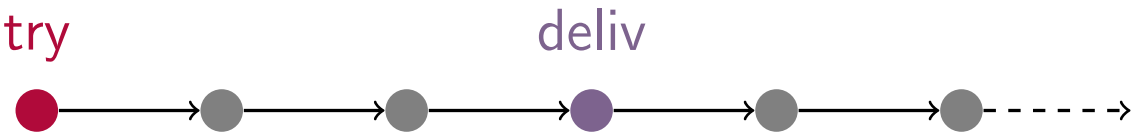
LTL : Until Operator

$$\phi ::= true \mid a \mid \phi \wedge \phi \mid \neg \phi \mid \bigcirc \phi \mid \Box \phi \mid \phi \bigcup \phi$$

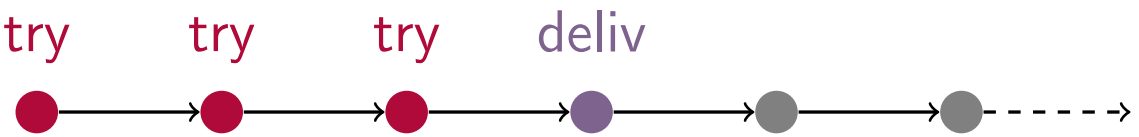
$try \Rightarrow \bigcirc deliv$



$try \Rightarrow \Diamond deliv$



$try \bigcup deliv$

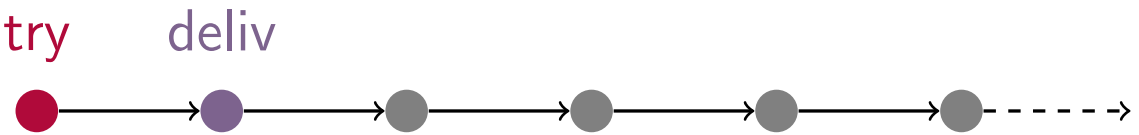


$$\Diamond \phi \equiv \mathbf{true} \cup \phi$$

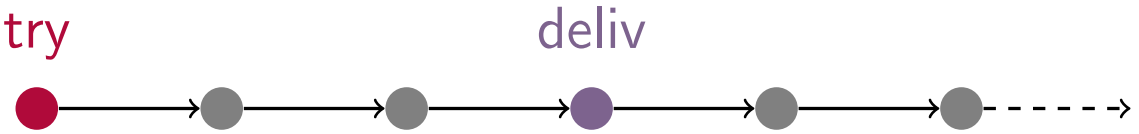
LTL : Until Operator

$$\phi ::= true \mid a \mid \phi \wedge \phi \mid \neg \phi \mid \bigcirc \phi \mid \not\Box \phi \mid \phi \bigcup \phi$$

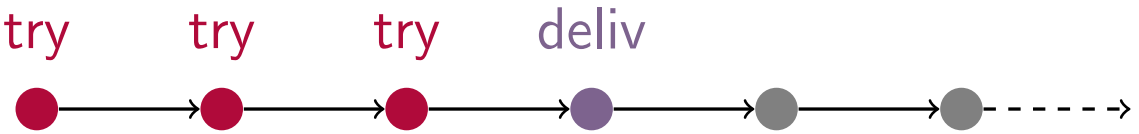
$try \Rightarrow \bigcirc deliv$



$try \Rightarrow \Diamond deliv$



$try \bigcup deliv$



$\Diamond \phi \equiv \mathbf{true} \cup \phi$ and $\Box \phi \equiv \neg \Diamond \neg \phi$

LTL : Past Operators

add the past-time dual of each temporal operator :

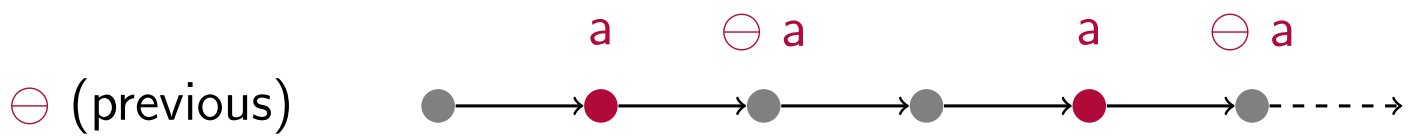
$$\phi \quad ::= \quad \ldots \quad | \quad \bigcirc \phi \quad | \quad \phi \quad \bigcup \quad \phi$$

LTL : Past Operators

add the past-time dual of each temporal operator :

$$\phi \quad ::= \quad \ldots \quad | \quad \bigcirc \phi \quad | \quad \phi \bigcup \phi \quad | \quad \ominus \phi$$

1. for $\bigcirc$ (next) : $\ominus$ (previous)

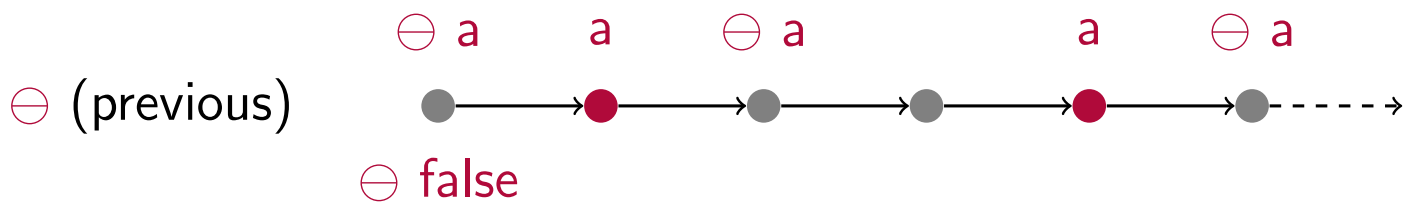


LTL : Past Operators

add the past-time dual of each temporal operator :

$$\phi \quad ::= \quad \ldots \quad | \quad \bigcirc \phi \quad | \quad \phi \bigcup \phi \quad | \quad \ominus \phi$$

1. for $\bigcirc$ (next) : $\ominus$ (previous) , **init** $\equiv \ominus$ *false*

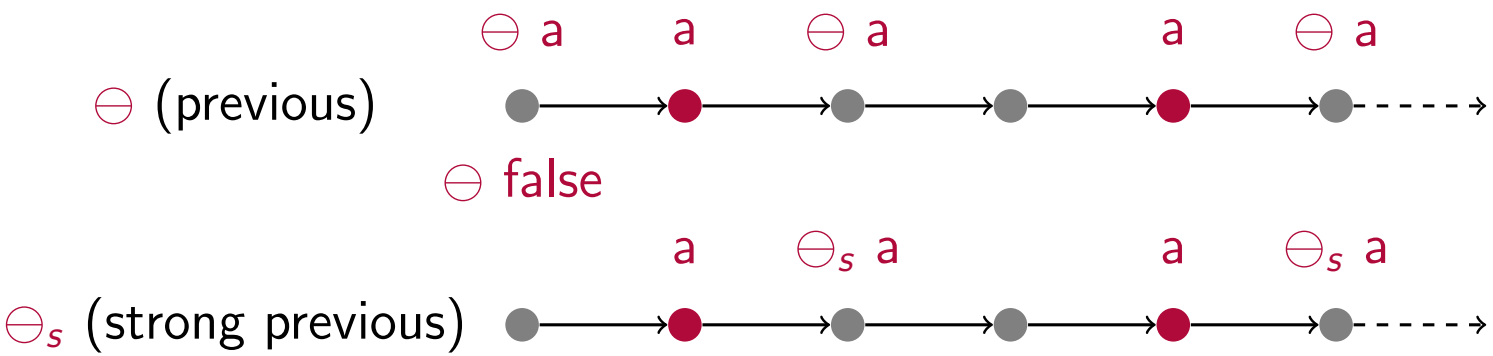


LTL : Past Operators

add the past-time dual of each temporal operator :

$$\phi \quad ::= \quad \ldots \quad | \quad \bigcirc \phi \quad | \quad \phi \bigcup \phi \quad | \quad \ominus \phi$$

1. for $\bigcirc$ (next) : $\ominus$ (previous) , **init** $\equiv \ominus$ *false* , $\ominus_s \phi \equiv \neg \ominus \neg \phi$



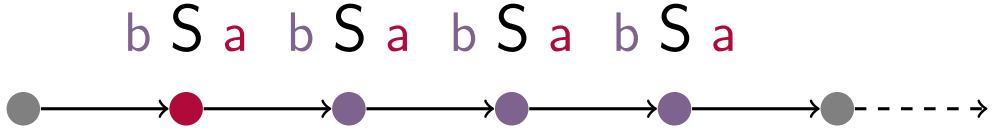
LTL : Past Operators

add the past-time dual of each temporal operator :

$$\phi \quad ::= \quad \ldots \quad | \quad \bigcirc \phi \quad | \quad \phi \bigcup \phi \quad | \quad \ominus \phi \quad | \quad \phi \ \mathsf{S} \ \phi$$

1. for $\bigcirc$ (next) : $\ominus$ (previous) , **init** $\equiv \ominus$ *false* , $\ominus_s \phi \equiv \neg \ominus \neg \phi$
2. for $\bigcup$ (until) : S (since)

b S a (since)



LTL : Past Operators

add the past-time dual of each temporal operator :

$$\phi \quad ::= \quad \ldots \quad | \quad \bigcirc \phi \quad | \quad \phi \bigcup \phi \quad | \quad \ominus \phi \quad | \quad \phi \ \mathsf{S} \ \phi$$

1. for $\bigcirc$ (next) : $\ominus$ (previous) , **init** $\equiv \ominus$ *false* , $\ominus_s \phi \equiv \neg \ominus \neg \phi$
2. for $\bigcup$ (until) : S (since)
3. for $\Box$ (always) : $\boxminus$ (always before), $\boxminus \phi \equiv (\textbf{init} \Rightarrow \phi) \wedge \phi \ \mathsf{S}$ **init**

$\boxminus$ (always before)

LTL : Past Operators

add the past-time dual of each temporal operator :

$$\phi \quad ::= \quad \ldots \quad | \quad \bigcirc \phi \quad | \quad \phi \bigcup \phi \quad | \quad \ominus \phi \quad | \quad \phi \ \mathsf{S} \ \phi$$

1. for $\bigcirc$ (next) : $\ominus$ (previous) , **init** $\equiv \ominus$ *false* , $\ominus_s \phi \equiv \neg \ominus \neg \phi$
2. for $\bigcup$ (until) : S (since)
3. for $\Box$ (always) : $\boxminus$ (always before), $\boxminus \phi \equiv (\textbf{init} \Rightarrow \phi) \wedge \phi \ \mathsf{S}\ \textbf{init}$
4. for $\Diamond$ (eventually) : $\diamondminus$ (eventually before), $\diamondminus \phi \equiv \neg \boxminus \neg \phi$

LTL : Past Operators

Theorem **LTL + past $\stackrel{?}{\equiv}$ LTL** [Gabbay]

- an (LTL + past) formula can be written as as disjunction of 3 pure formulas :

$$\phi \equiv \underbrace{\phi_{past}}_{\mathsf{S},\ominus,\wedge,\neg} \vee \underbrace{\phi_{present}}_{\wedge,\neg} \vee \underbrace{\phi_{future}}_{\mathsf{U},\bigcirc,\wedge,\neg}$$

- proof : by induction (FOLLO)

LTL : Past Operators

Theorem **LTL + past $\overset{exp}{\equiv}$ LTL** [Gabbay]

- an (LTL + past) formula can be written as as disjunction of 3 pure formulas :

$$\phi \equiv \underbrace{\phi_{past}}_{\textbf{true, false}} \vee \underbrace{\phi_{present}}_{\wedge, \neg} \vee \underbrace{\phi_{future}}_{\mathsf{U}, \bigcirc, \wedge, \neg}$$

- proof : by induction (FOLLO)

LTL : Past Operators

Theorem **LTL + past $\overset{exp}{\equiv}$ LTL** [Gabbay]

- an (LTL + past) formula can be written as as disjunction of 3 pure formulas :

$$\phi \equiv \underbrace{\phi_{past}}_{\textbf{true, false}} \vee \underbrace{\phi_{present}}_{\wedge, \neg} \vee \underbrace{\phi_{future}}_{\mathsf{U}, \bigcirc, \wedge, \neg}$$

- proof : by induction (FOLLO)

- example :

$$\Box((\boxminus \neg pin) \Longrightarrow \neg cash)$$

LTL : Past Operators

Theorem **LTL + past $\overset{exp}{\equiv}$ LTL** [Gabbay]

- an (LTL + past) formula can be written as as disjunction of 3 pure formulas :

$$\phi \equiv \underbrace{\phi_{past}}_{\textbf{true, false}} \vee \underbrace{\phi_{present}}_{\wedge, \neg} \vee \underbrace{\phi_{future}}_{\mathsf{U}, \bigcirc, \wedge, \neg}$$

- proof : by induction (FOLLO)

- example :

$$\Box((\boxminus \neg pin) \Longrightarrow \neg cash) \equiv \neg cash \bigcup pin$$

Finite Automata (recall)

- A (Nondeterministic) Finite Automaton (NFA) $\mathcal{A}$ is a tuple $(\Sigma, S, I, \delta, F)$, where
  - Σ is a finite alphabet
  - S is a finite set of states
  - $I \subseteq S$ is the set of initial states
  - $\delta : S \times \Sigma \to 2^S$ is a transition function
  - $F \subseteq S$ is the set of accepting (final) states

Finite Automata (recall)

- A (Nondeterministic) Finite Automaton (NFA) $\mathcal{A}$ is a tuple $(\Sigma, S, I, \delta, F)$, where
  - Σ is a finite alphabet
  - S is a finite set of states
  - $I \subseteq S$ is the set of initial states
  - $\delta : S \times \Sigma \to 2^S$ is a transition function
  - $F \subseteq S$ is the set of accepting (final) states

- A finite word $w = A_1 \dots A_n \in \Sigma^*$ is accepted by the NFA $\mathcal{A}$ if there exists a finite sequence $\pi = s_0 \dots s_n \in S^*$ such that :

$$s_0 \in I \quad \text{and} \quad \forall\, 0 \leq i < n,\ s_i \xrightarrow{A_{i+1}} s_{i+1} \quad \text{and} \quad s_n \in F$$

- We say that w belongs to the language of $\mathcal{A}$ noted $\mathcal{L}(\mathcal{A})$

Büchi Automata : Formal Definition

- A (Nondeterministic) Büchi Automaton (NBA) $\mathcal{A}$ is a tuple $(\Sigma, S, I, \delta, F)$ identical to the NFA definition.

Büchi Automata : Formal Definition

- A (Nondeterministic) Büchi Automaton (NBA) $\mathcal{A}$ is a tuple $(\Sigma, S, I, \delta, F)$ identical to the NFA definition.

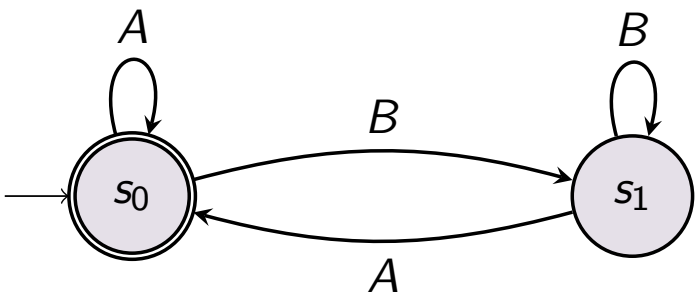
- An infinite word $w = A_1 A_2 \ldots \in \Sigma^\omega$ is accepted by the NBA $\mathcal{A}$ if there exists an infinite sequence $\pi = s_0 s_1 \ldots \in S^\omega$ such that :

  $s_0 \in I$ and $\forall\, 0 \leq i,\ s_i \xrightarrow{A_{i+1}} s_{i+1}$ and $s_i \in F$ for infinitely many i

- we note $\mathcal{L}_\omega(\mathcal{A}) \subseteq \Sigma^\omega$ the accepted language of $\mathcal{A}$

NFA/NBA example

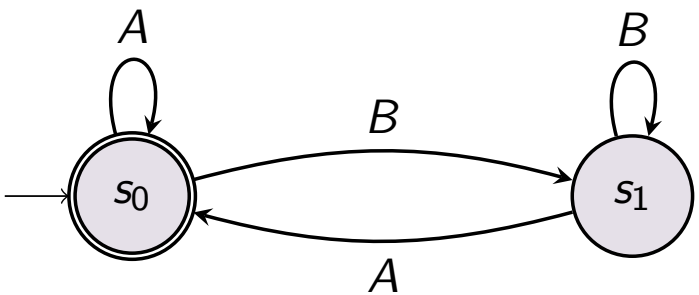
- Another example over the alphabet $\Sigma = \{A, B\}$



- when considered as NFA, this automaton recognizes ?

NFA/NBA example

- Another example over the alphabet $\Sigma = \{A, B\}$

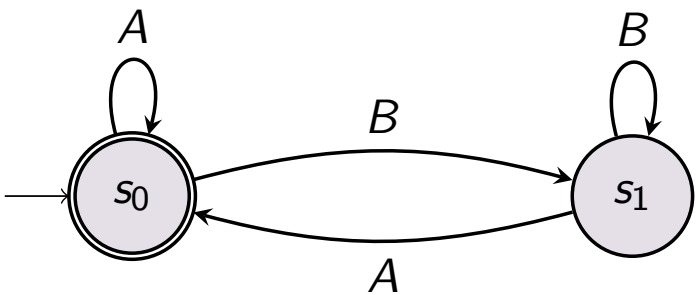


- when considered as NFA, this automaton recognizes :
  - all finite words ending with the letter A

$$\mathcal{L}(\mathcal{A}) \equiv (A + B)^*.A$$

NFA/NBA example

- Another example over the alphabet $\Sigma = \{A, B\}$



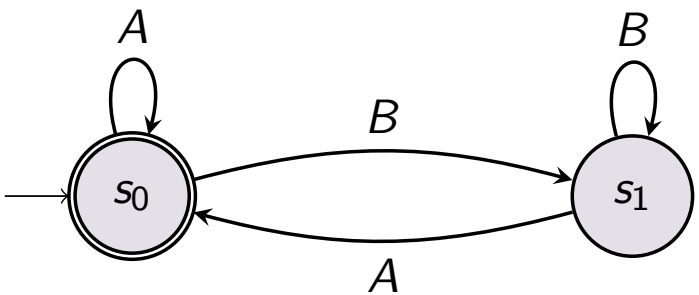
- when considered as NFA, this automaton recognizes :
  - all finite words ending with the letter A

$$\mathcal{L}(\mathcal{A}) \equiv (A + B)^*.A$$

- when considered as NBA, this automaton recognizes ?

NFA/NBA example

- Another example over the alphabet $\Sigma = \{A, B\}$



- when considered as NFA, this automaton recognizes :
  - all finite words ending with the letter A

$$\mathcal{L}(\mathcal{A}) \equiv (A + B)^*.A$$

- when considered as NBA, this automaton recognizes :
  - all infinite words having infinitely often A

$$\mathcal{L}_\omega(\mathcal{A}) \equiv ((A + B)^*.A)^\omega$$

Büchi Automata : Emptiness

A language of a Büchi automaton $\mathcal{A}$ is said to be empty when

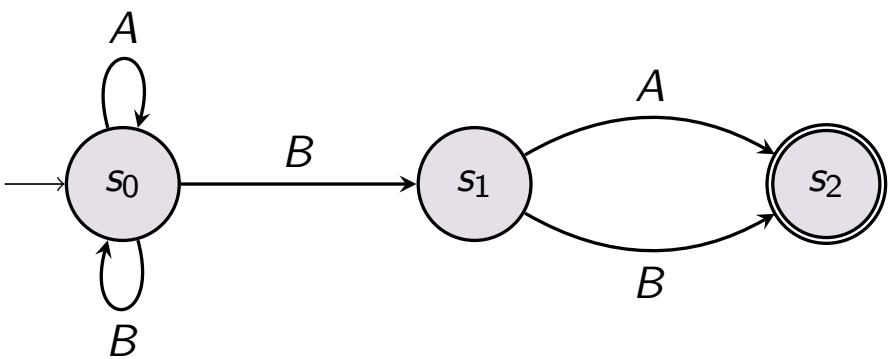
$$\mathcal{L}_\omega(\mathcal{A}) = \emptyset$$

Büchi Automata : Emptiness

A language of a Büchi automaton $\mathcal{A}$ is said to be empty when

$$\mathcal{L}_\omega(\mathcal{A}) = \emptyset$$

- How to automatically check emptiness ?
- A typical example :



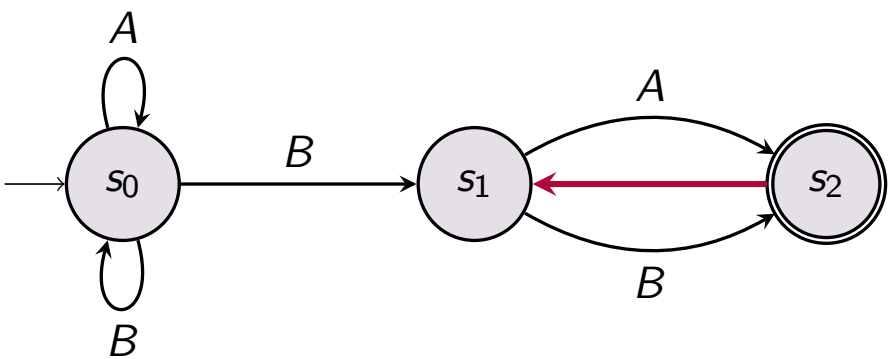
$$\mathcal{L}_\omega(\mathcal{A}) = \emptyset$$

Büchi Automata : Emptiness

A language of a Büchi automaton $\mathcal{A}$ is said to be empty when

$$\mathcal{L}_\omega(\mathcal{A}) = \emptyset$$

- How to automatically check emptiness ?
- A typical example :



$$\mathcal{L}_\omega(\mathcal{A}) \neq \emptyset$$

Büchi Automata : Checking Emptiness

- Checking Emptiness principle :
  - If there is a reachable final state on a cycle $\Rightarrow \mathcal{L}_\omega(\mathcal{A}) \neq \emptyset$
- Checking Emptiness Algorithm : using a traversal algorithm
  1. starting from initial states, checking for the next reachable final state
  2. starting from such state, checking if it may reach itself (cycle)
  3. loop if no cycle
- Thus, checking Emptiness can be solved in linear time

from LTL to NBA

Theorem

For any Linear Time Property $P_\phi \subseteq (2^{AP})^\omega$ corresponding to some LTL-formula ϕ (over AP), there exists an NBA $\mathcal{A}_\phi$ over 2^{AP} such that :

$$\mathcal{L}_\omega(\mathcal{A}_\phi) = P_\phi$$

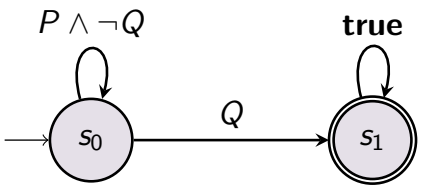
from LTL to NBA

Theorem

For any Linear Time Property $P_\phi \subseteq (2^{AP})^\omega$ corresponding to some LTL-formula ϕ (over AP), there exists an NBA $\mathcal{A}_\phi$ over 2^{AP} such that :

$$\mathcal{L}_\omega(\mathcal{A}_\phi) = P_\phi$$

- from LTL to NBA examples :
  - for $\phi = P \bigcup Q$



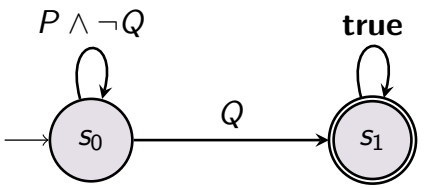
from LTL to NBA

Theorem

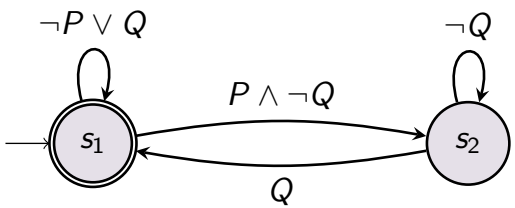
For any Linear Time Property $P_\phi \subseteq (2^{AP})^\omega$ corresponding to some LTL-formula ϕ (over AP), there exists an NBA $\mathcal{A}_\phi$ over 2^{AP} such that :

$$\mathcal{L}_\omega(\mathcal{A}_\phi) = P_\phi$$

- from LTL to NBA examples :
  - for $\phi = P \bigcup Q$



  - for $\phi = \Box\,(P \Rightarrow \Diamond\ Q)$



LTL : subFormulas

Definition

- Let ϕ be an LTL formula :

  $\varphi \in subF(\phi) \iff \varphi$ or $\neg\varphi$ is a subformula of ϕ

LTL : subFormulas

Definition

- Let ϕ be an LTL formula :

  $\varphi \in subF(\phi) \iff \varphi$ or $\neg\varphi$ is a subformula of ϕ

- Examples :
  - $subF(\bigcirc\ a) = \{a, \neg a, \bigcirc\ a, \neg \bigcirc\ a\}$

LTL : subFormulas

Definition

- Let ϕ be an LTL formula :

  $\varphi \in subF(\phi) \iff \varphi$ or $\neg\varphi$ is a subformula of ϕ

- Examples :
  - $subF(\bigcirc\ a) = \{a, \neg a, \bigcirc\ a, \neg \bigcirc\ a\}$
  - $subF(a\ \bigcup\ b) = \{a, \neg a, b, \neg b, a\ \bigcup\ b, \neg(a\ \bigcup\ b)\}$

LTL : subFormulas

Definition

- Let ϕ be an LTL formula :

  $\varphi \in subF(\phi) \iff \varphi$ or $\neg\varphi$ is a subformula of ϕ

- Examples :
  - $subF(\bigcirc\ a) = \{a, \neg a, \bigcirc\ a, \neg \bigcirc\ a\}$
  - $subF(a\ \bigcup\ b) = \{a, \neg a, b, \neg b, a\ \bigcup\ b, \neg(a\ \bigcup\ b)\}$
  - $\phi = (a \wedge \neg b)\ \bigcup\ b$

    $subF(\phi)$?

LTL : subFormulas

Definition

- Let ϕ be an LTL formula :

  $\varphi \in subF(\phi) \iff \varphi$ or $\neg\varphi$ is a subformula of ϕ

- Examples :
  - $subF(\bigcirc\ a) = \{a, \neg a, \bigcirc\ a, \neg \bigcirc\ a\}$
  - $subF(a\ \bigcup\ b) = \{a, \neg a, b, \neg b, a\ \bigcup\ b, \neg(a\ \bigcup\ b)\}$
  - $\phi = (a \wedge \neg b)\ \bigcup\ b$

    $subF(\phi) = \{a, \neg a, b, \neg b, a \wedge \neg b, \neg(a \wedge \neg b), \phi, \neg\phi\}$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

- Examples : $\phi = (a \wedge \neg b) \bigcup b$
  - $\mathcal{B} = \{a, \neg b, \phi\}$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

- Examples : $\phi = (a \wedge \neg b) \bigcup b$
  - $\mathcal{B} = \{a, \neg b, \phi\}$ (1) **X**

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

- Examples : $\phi = (a \wedge \neg b) \bigcup b$
  - $\mathcal{B} = \{a, \neg b, \phi\}$ (1) **X**
  - $\mathcal{B} = \{a, b, a \wedge \neg b, \phi\}$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

- Examples : $\phi = (a \wedge \neg b) \bigcup b$
  - $\mathcal{B} = \{a, \neg b, \phi\}$ (1) **X**
  - $\mathcal{B} = \{a, b, a \wedge \neg b, \phi\}$ (2) **X**

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

- Examples : $\phi = (a \wedge \neg b) \bigcup b$
  - $\mathcal{B} = \{a, \neg b, \phi\}$ (1) **X**
  - $\mathcal{B} = \{a, b, a \wedge \neg b, \phi\}$ (2) **X**
  - $\mathcal{B} = \{a, \neg b, a \wedge \neg b, \phi\}$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

- Examples : $\phi = (a \wedge \neg b) \bigcup b$
  - $\mathcal{B} = \{a, \neg b, \phi\}$ (1) **X**
  - $\mathcal{B} = \{a, b, a \wedge \neg b, \phi\}$ (2) **X**
  - $\mathcal{B} = \{a, \neg b, a \wedge \neg b, \phi\}$ $\surd$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

- Examples : $\phi = (a \wedge \neg b) \bigcup b$
  - $\mathcal{B} = \{a, \neg b, \phi\}$ (1) **X**
  - $\mathcal{B} = \{a, b, a \wedge \neg b, \phi\}$ (2) **X**
  - $\mathcal{B} = \{a, \neg b, a \wedge \neg b, \phi\}$ $\surd$
  - $\mathcal{B} = \{a, b, \neg(a \wedge \neg b), \neg\phi\}$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

- Examples : $\phi = (a \wedge \neg b) \bigcup b$
  - $\mathcal{B} = \{a, \neg b, \phi\}$ (1) **X**
  - $\mathcal{B} = \{a, b, a \wedge \neg b, \phi\}$ (2) **X**
  - $\mathcal{B} = \{a, \neg b, a \wedge \neg b, \phi\}$ $\surd$
  - $\mathcal{B} = \{a, b, \neg(a \wedge \neg b), \neg\phi\}$ (4) **X**

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

- Examples : $\phi = (a \wedge \neg b) \bigcup b$
  - $\mathcal{B} = \{a, \neg b, \phi\}$ (1) **X**
  - $\mathcal{B} = \{a, b, a \wedge \neg b, \phi\}$ (2) **X**
  - $\mathcal{B} = \{a, \neg b, a \wedge \neg b, \phi\}$ $\surd$
  - $\mathcal{B} = \{a, b, \neg(a \wedge \neg b), \neg\phi\}$ (4) **X**
  - $\mathcal{B} = \{\neg a, \neg b, \neg(a \wedge \neg b), \phi\}$

Consistent sets of subFormulas

Definition

- Let $\mathcal{B} \subseteq subF(\phi)$ a set of subFormulas of ϕ, $\mathcal{B}$ is consistent iff :
  1. $\varphi \in subF(\phi) \setminus \mathcal{B} \iff \neg\varphi \in \mathcal{B}$
  2. $\varphi_1 \wedge \varphi_2 \in \mathcal{B} \iff \varphi_1 \in \mathcal{B} \wedge \varphi_2 \in \mathcal{B} \wedge (\varphi_1 \wedge \varphi_2) \in subF(\phi)$
  3. $\varphi_1 \bigcup \varphi_2 \in \mathcal{B} \wedge \varphi_2 \notin \mathcal{B} \implies \varphi_1 \in \mathcal{B}$
  4. $\varphi_1 \bigcup \varphi_2 \in subF(\phi) \wedge \varphi_2 \in \mathcal{B} \implies \varphi_1 \bigcup \varphi_2 \in \mathcal{B}$

- Examples : $\phi = (a \wedge \neg b) \bigcup b$
  - $\mathcal{B} = \{a, \neg b, \phi\}$ (1) **X**
  - $\mathcal{B} = \{a, b, a \wedge \neg b, \phi\}$ (2) **X**
  - $\mathcal{B} = \{a, \neg b, a \wedge \neg b, \phi\}$ $\surd$
  - $\mathcal{B} = \{a, b, \neg(a \wedge \neg b), \neg\phi\}$ (4) **X**
  - $\mathcal{B} = \{\neg a, \neg b, \neg(a \wedge \neg b), \phi\}$ (3) **X**

GNBA for LTL formula

The GNBA $\mathcal{G}_\phi$ is a tuple $(\Sigma, S, I, \delta, \mathcal{F})$ where :

- $\Sigma = 2^{AP}$

GNBA for LTL formula

The GNBA $\mathcal{G}_\phi$ is a tuple $(\Sigma, S, I, \delta, \mathcal{F})$ where :

- $\Sigma = 2^{AP}$
- $S = \{\mathcal{B} \subseteq \mathit{subF}(\phi) \mid \mathcal{B} \text{ is consistent}\}$

GNBA for LTL formula

The GNBA $\mathcal{G}_\phi$ is a tuple $(\Sigma, S, I, \delta, \mathcal{F})$ where :

- $\Sigma = 2^{AP}$
- $S = \{\mathcal{B} \subseteq \mathit{subF}(\phi) \mid \mathcal{B} \text{ is consistent}\}$
- $I = \{\mathcal{B} \in S \mid \phi \in \mathcal{B}\}$

GNBA for LTL formula

The GNBA $\mathcal{G}_\phi$ is a tuple $(\Sigma, S, I, \delta, \mathcal{F})$ where :

- $\Sigma = 2^{AP}$
- $S = \{\mathcal{B} \subseteq subF(\phi) \mid \mathcal{B} \text{ is consistent}\}$
- $I = \{\mathcal{B} \in S \mid \phi \in \mathcal{B}\}$
- $\delta : S \times \Sigma \to 2^S$ is the transition function defined as follows :

$$\mathcal{B} \xrightarrow{A} \mathcal{B}' \in \delta \iff \begin{cases} A = \mathcal{B} \cap AP \\ \bigcirc \varphi \in \mathcal{B} \Leftrightarrow \varphi \in \mathcal{B}' \\ \varphi_1 \cup \varphi_2 \in \mathcal{B} \Leftrightarrow \varphi_2 \in \mathcal{B} \vee \\ \qquad (\varphi_1 \in \mathcal{B} \wedge \varphi_1 \cup \varphi_2 \in \mathcal{B}') \end{cases}$$

GNBA for LTL formula

The GNBA $\mathcal{G}_\phi$ is a tuple $(\Sigma, S, I, \delta, \mathcal{F})$ where :

- $\Sigma = 2^{AP}$
- $S = \{\mathcal{B} \subseteq \mathit{subF}(\phi) \mid \mathcal{B} \text{ is consistent}\}$
- $I = \{\mathcal{B} \in S \mid \phi \in \mathcal{B}\}$
- $\delta : S \times \Sigma \to 2^S$ is the transition function defined as follows :

$$\mathcal{B} \xrightarrow{A} \mathcal{B}' \in \delta \iff \begin{cases} A = \mathcal{B} \cap AP \\ \bigcirc \varphi \in \mathcal{B} \Leftrightarrow \varphi \in \mathcal{B}' \\ \varphi_1 \cup \varphi_2 \in \mathcal{B} \Leftrightarrow \varphi_2 \in \mathcal{B} \vee \\ \qquad (\varphi_1 \in \mathcal{B} \wedge \varphi_1 \cup \varphi_2 \in \mathcal{B}') \end{cases}$$

- $\mathcal{F} = \{F_{\varphi_1 \cup \varphi_2} \mid \varphi_1 \cup \varphi_2 \in \mathit{subF}(\phi)\}$ where

$$F_{\varphi_1 \cup \varphi_2} = \{\mathcal{B} \in S \mid \varphi_1 \cup \varphi_2 \notin \mathcal{B} \vee \varphi_2 \in \mathcal{B}\}$$

Decidability of LTL

Theorem

SAT(LTL) is decidable

- Proof ?

Decidability of LTL

Theorem

SAT(LTL) is decidable

- Proof ? transform LTL to BA then check emptiness

Decidability of LTL

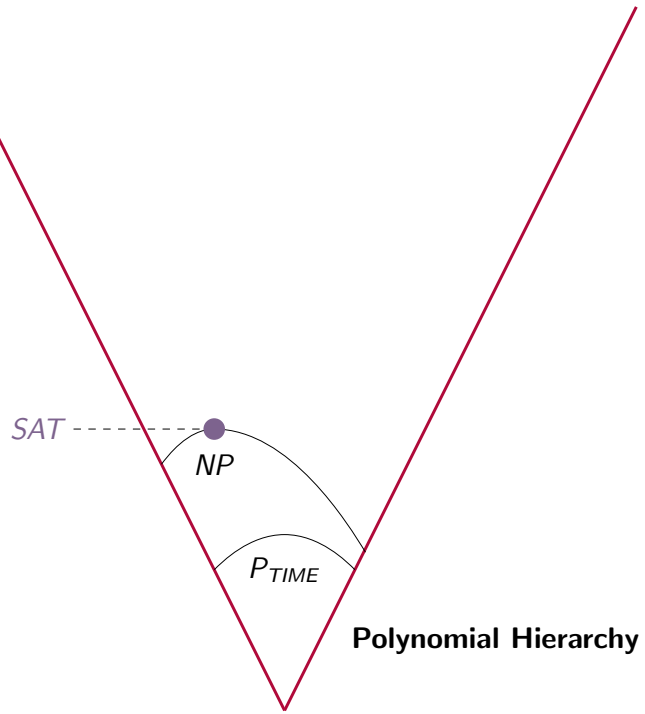
Theorem

SAT(LTL) is decidable

- Proof ? transform LTL to BA then check emptiness

- VAL(LTL) is decidable (ex : LTL vs LTL)
- MC(LTL) is decidable (LTL vs Autmaton)

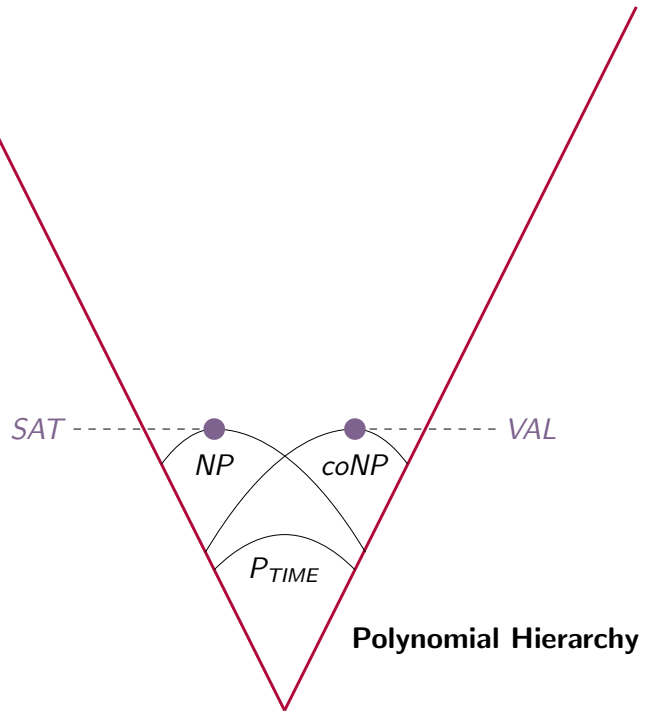
Complexity of LTL



Let $X_i = \{x_i^1, x_i^2, \ldots, x_i^n\}$ a vector of boolean variables,

The problem $SAT \equiv \exists X_1,\ Prop(X_1)$ is NP-complete

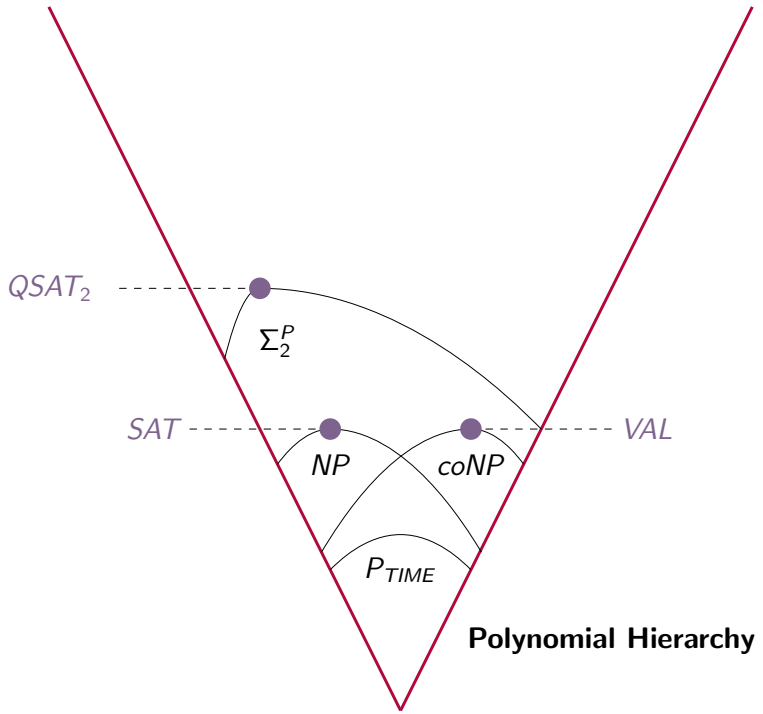
Complexity of LTL



Let $X_i = \{x_i^1, x_i^2, \ldots, x_i^n\}$ a vector of boolean variables,

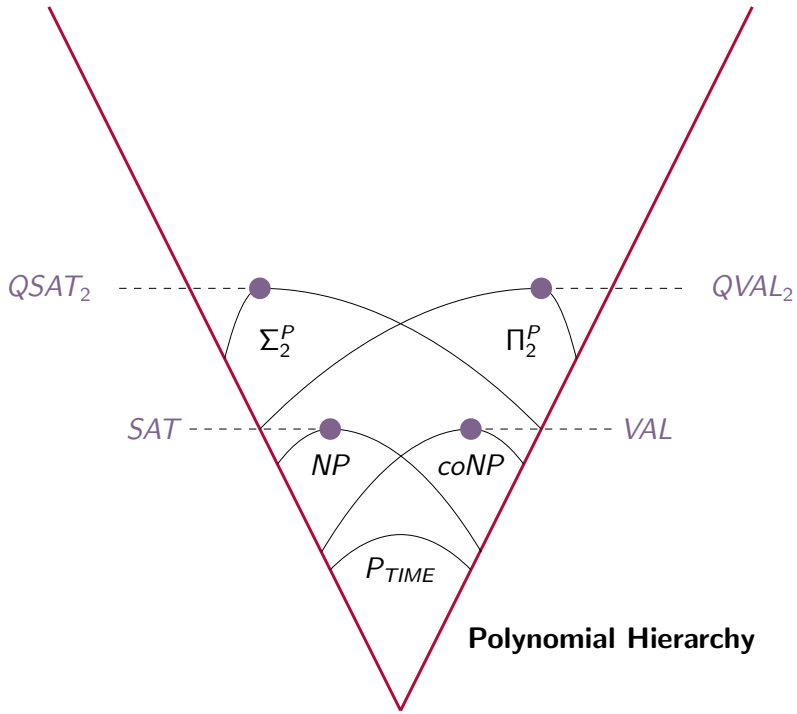
The problem $VAL \equiv \forall X_1,\ Prop(X_1)$ is **co**NP-complete

Complexity of LTL



The problem $QSAT_2 \equiv \exists X_2,\ \forall X_1,\ Prop(X_1, X_2)$ is Σ_2^P-complete

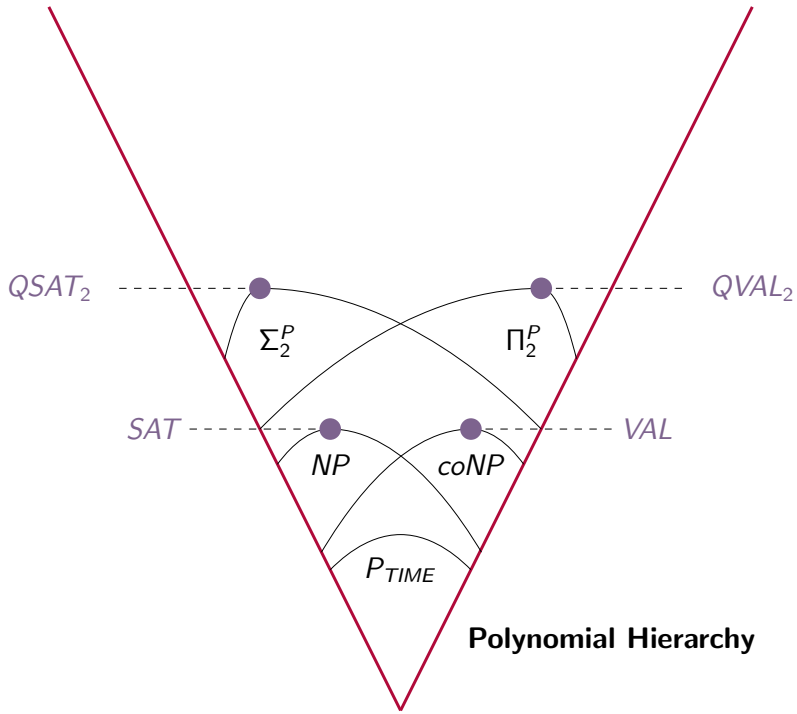
Complexity of LTL



The problem $QSAT_2 \equiv \exists X_2,\ \forall X_1,\ Prop(X_1, X_2)$ is Σ_2^P-complete

The problem $QVAL_2 \equiv \forall X_2,\ \exists X_1,\ Prop(X_1, X_2)$ is Π_2^P-complete

Complexity of LTL

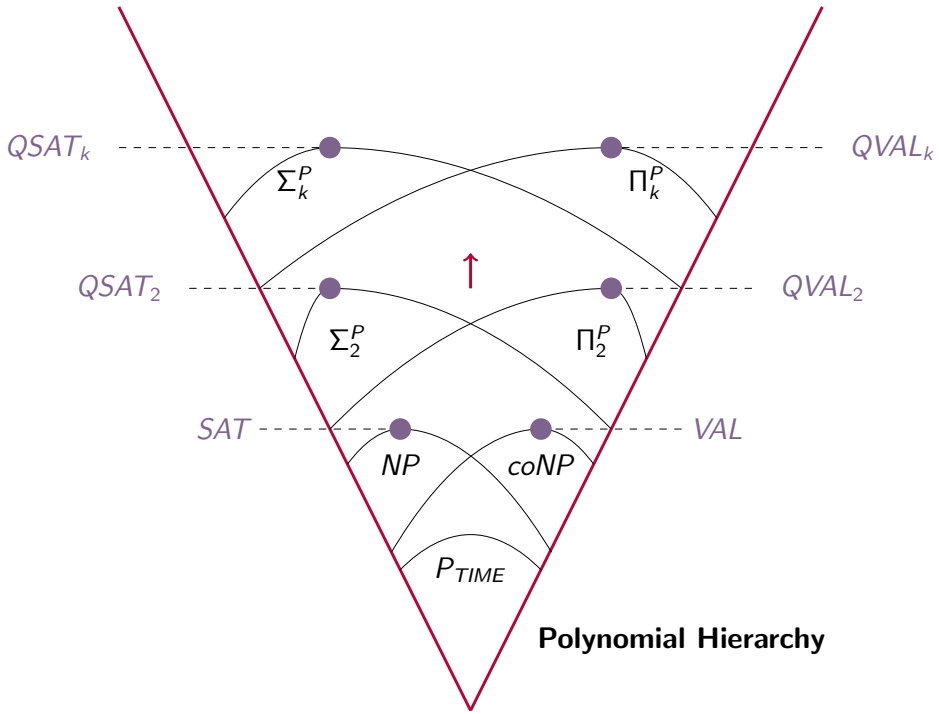


The problem $QSAT_2 \equiv \exists X_2, [\forall X_1, \ Prop(X_1, X_2)]$ is Σ_2^P-complete

The problem $QVAL_2 \equiv \forall X_2, [\exists X_1, \ Prop(X_1, X_2)]$ is Π_2^P-complete

$$\Sigma_2^P \equiv NP^{coNP} \qquad \Pi_2^P \equiv coNP^{NP}$$

Complexity of LTL

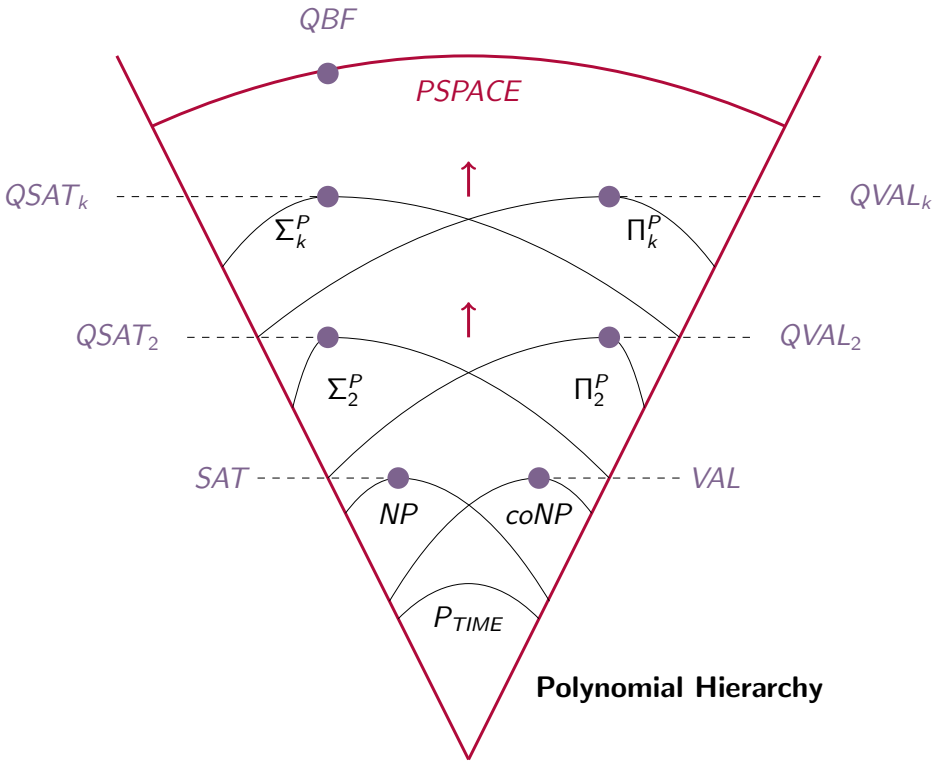


The problem $QSAT_k \equiv \exists X_k, \forall X_{k-1} \ldots, \; Prop(X_1, \ldots, X_k)$ is Σ_k^P-complete

The problem $QVAL_k \equiv \forall X_k, \exists X_{k-1} \ldots, \; Prop(X_1, \ldots, X_k)$ is Π_k^P-complete

$$\Sigma_k^P \equiv NP^{\Pi_{k-1}^P} \qquad \Pi_k^P \equiv coNP^{\Sigma_{k-1}^P}$$

Complexity of LTL

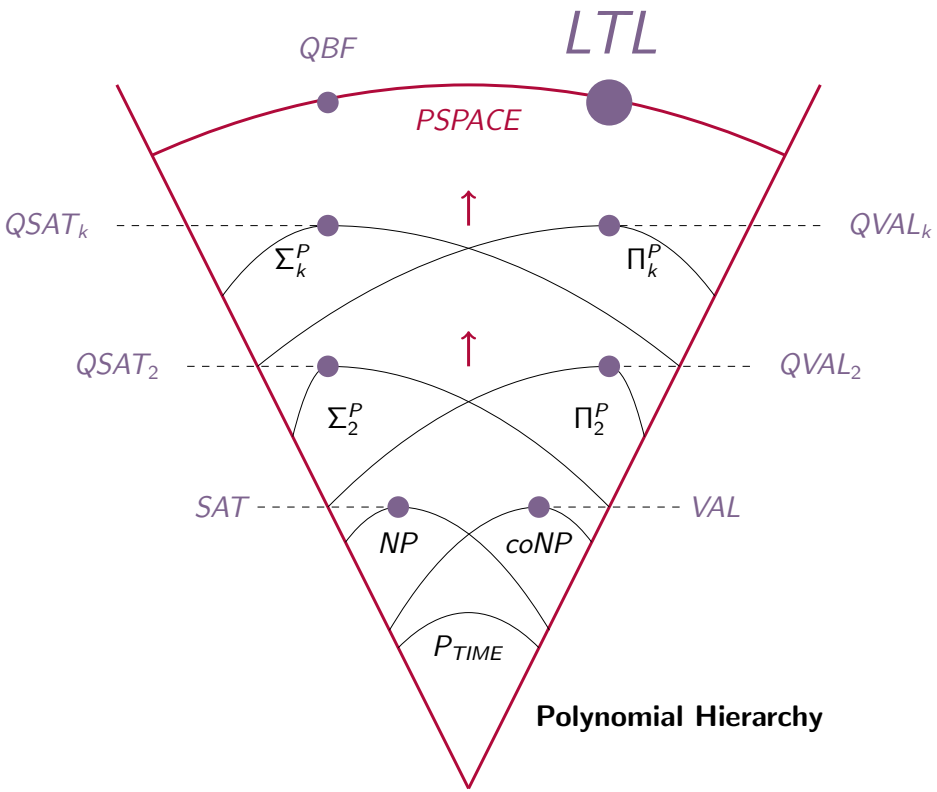


Let $Q_i \in \{\forall, \exists\}$, the general problem :

$QBF \equiv Q_n x_n, Q_{n-1} x_{n-1} \ldots, Q_1 x_1,\ Prop(x_n, \ldots, x_1)$ is $PSPACE$-complete

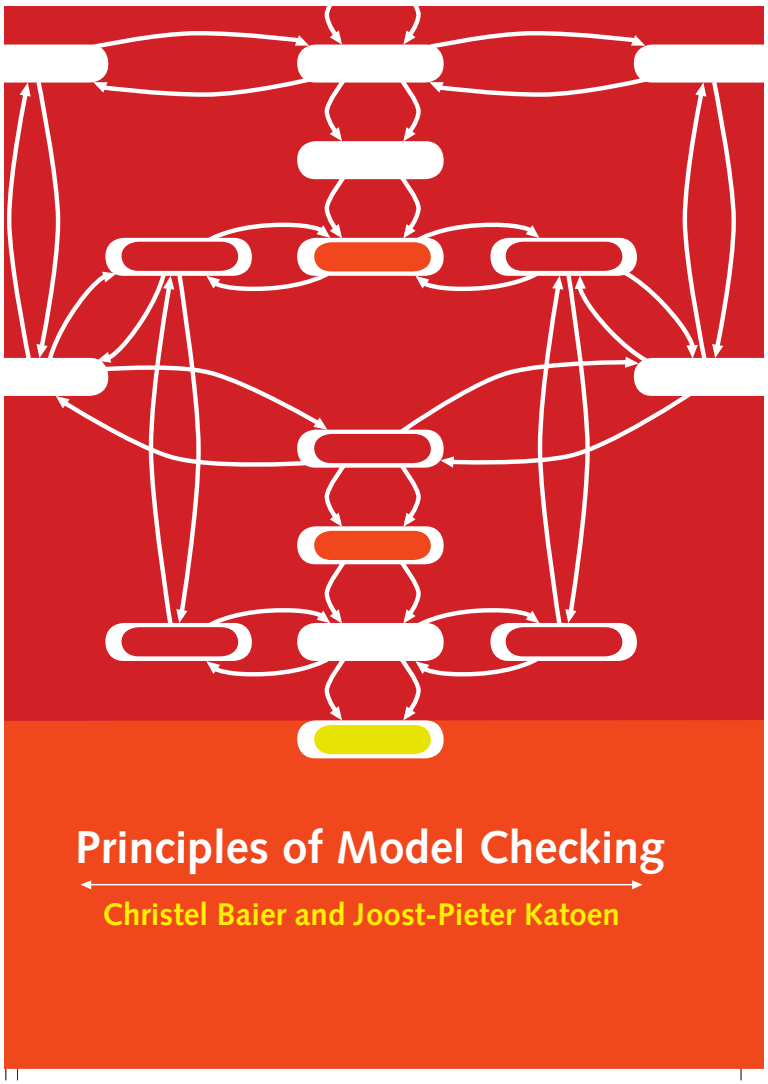
$$PSPACE \equiv NPSPACE \equiv coPSPACE$$

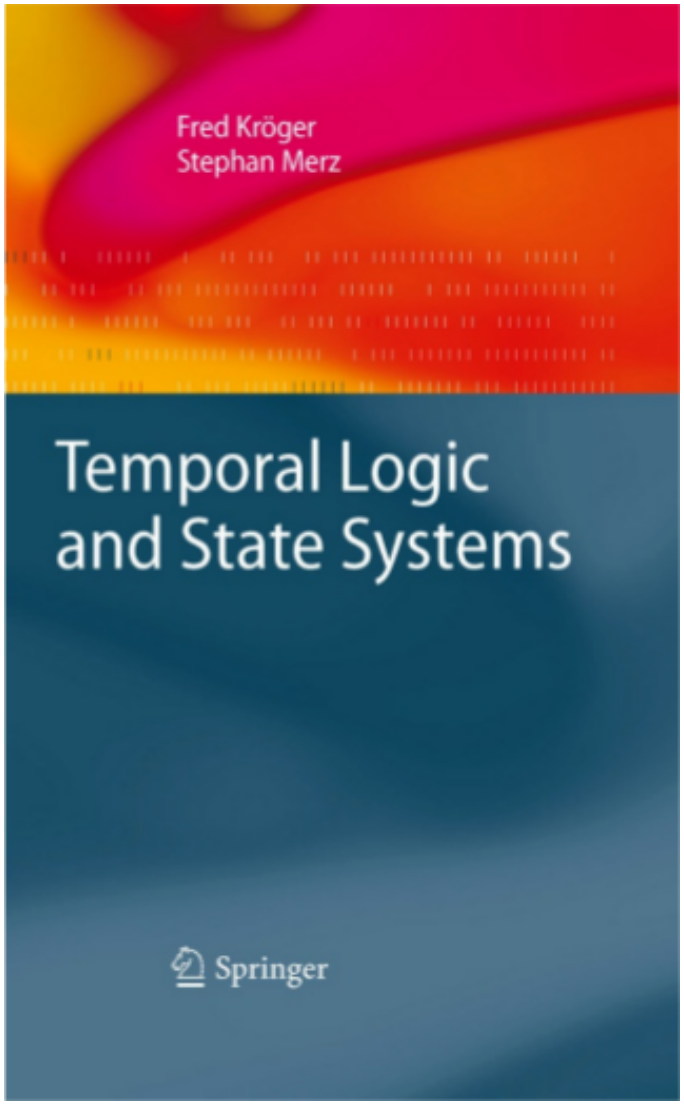
Complexity of LTL



The problems $SAT(LTL) \equiv VAL(LTL) \equiv MC(LTL)$ are *PSPACE*-complete

References


Principles of Model Checking
Christel Baier and Joost-Pieter Katoen


Fred Kröger
Stephan Merz
Temporal Logic
and State Systems
Springer

Plan

Context : Temporal Logics

Example

while a user is connected (between Login and Logout),
each user's request (Req) must be responded (Resp)

Context : Temporal Logics

Example

while a user is connected (between Login and Logout),
each user's request (Req) must be responded (Resp)

- in LTL :

$$\Box((Login \wedge \Diamond Logout) \Rightarrow (Req \Rightarrow (\neg Logout \ \mathsf{U} \ Resp)) \ \mathsf{U} \ Logout)$$

Context : Temporal Logics

Example

while a user is connected (between Login and Logout),
each user's request (Req) must be responded (Resp)

- in LTL :

$$\Box((Login \wedge \Diamond Logout) \Rightarrow (Req \Rightarrow (\neg Logout \ \mathsf{U} \ Resp)) \ \mathsf{U} \ Logout)$$

- this formula will be even more complex without assuming :

$$\Box(Login + Logout + Req + Resp \leq 1)$$

Context : Temporal Logics

Example

while a user is connected (between Login and Logout),
each user's request (Req) must be responded (Resp)

- in LTL :

$$\Box((Login \wedge \Diamond Logout) \Rightarrow (Req \Rightarrow (\neg Logout \ \mathsf{U} \ Resp)) \ \mathsf{U} \ Logout)$$

- this formula will be even more complex without assuming :

$$\Box(Login + Logout + Req + Resp \leq 1)$$

Issue

bridging the semantic gap between the natural language and Temporal Logics (the formal language)

Principle

- Can we provide formal semantics (of Temporal Logics) to the natural language ?

Principle

- Can we provide formal semantics (of Temporal Logics) to the natural language ?
  - one typical solution : a pattern-based language

Principle

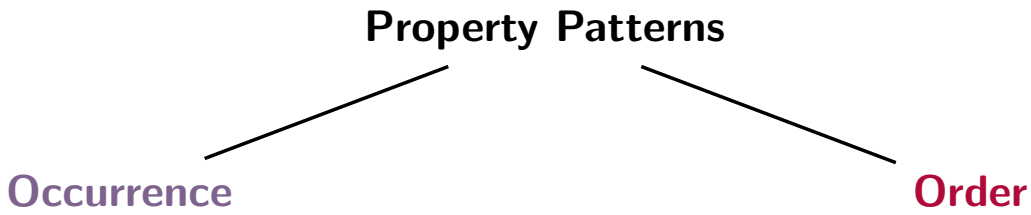
- Can we provide formal semantics (of Temporal Logics) to the natural language ?
  - one typical solution : a pattern-based language

Dwyer et al.'s Temporal Patterns [Dwyer et al., 1999]

- Dwyer et al. predefine many patterns of temporal properties
- Each temporal property is a combination of :

Pattern / Scope

Patterns

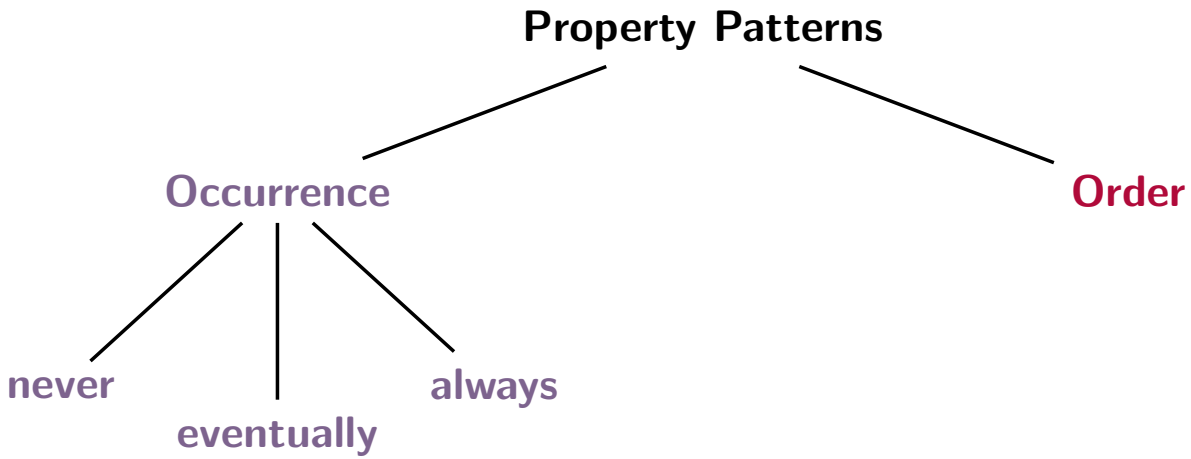


Property patterns

Occurrence patterns : Absence/Presence of events

Order patterns : Order between events

Patterns

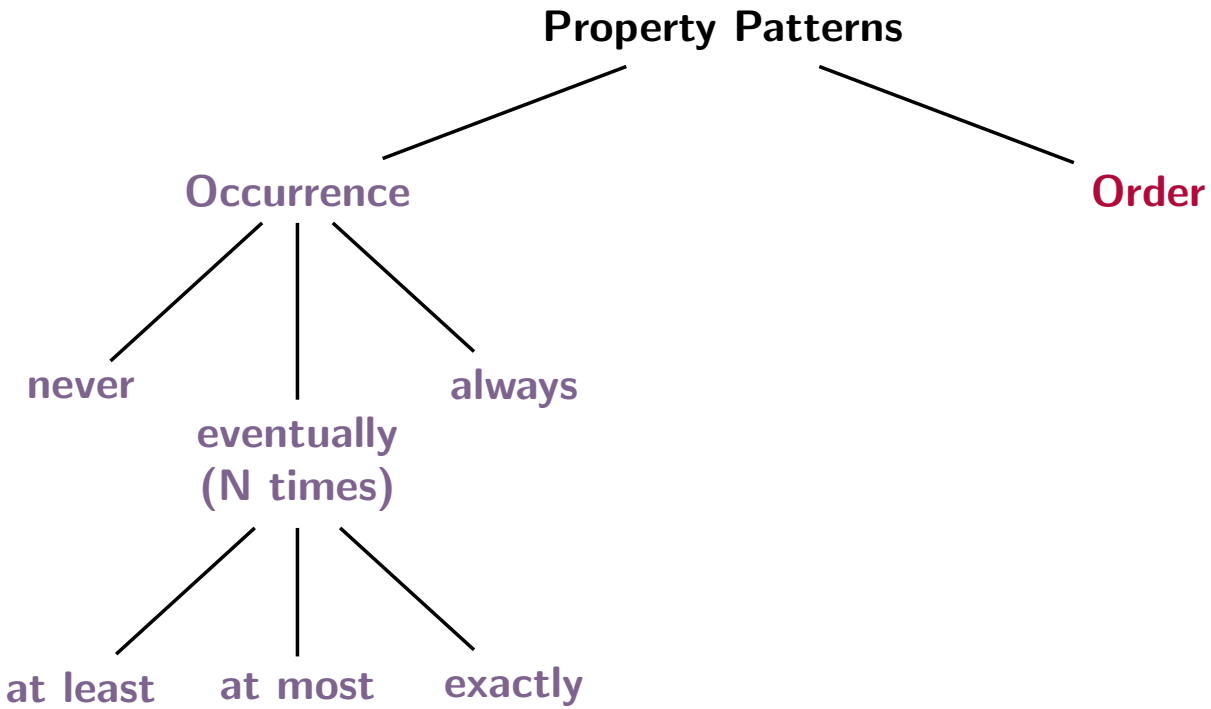


Property patterns

Occurrence patterns : Absence/Presence of events

Order patterns : Order between events

Patterns

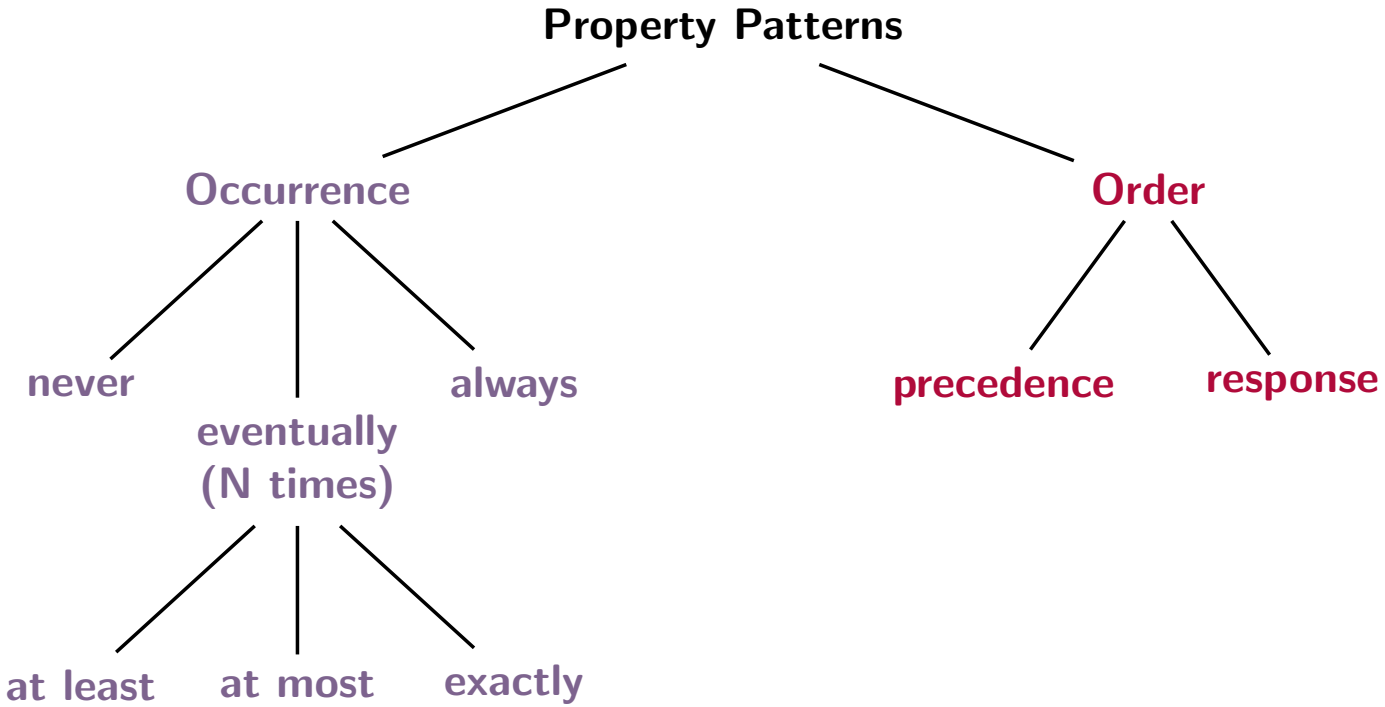


Property patterns

Occurrence patterns : Absence/Presence of events

Order patterns : Order between events

Patterns

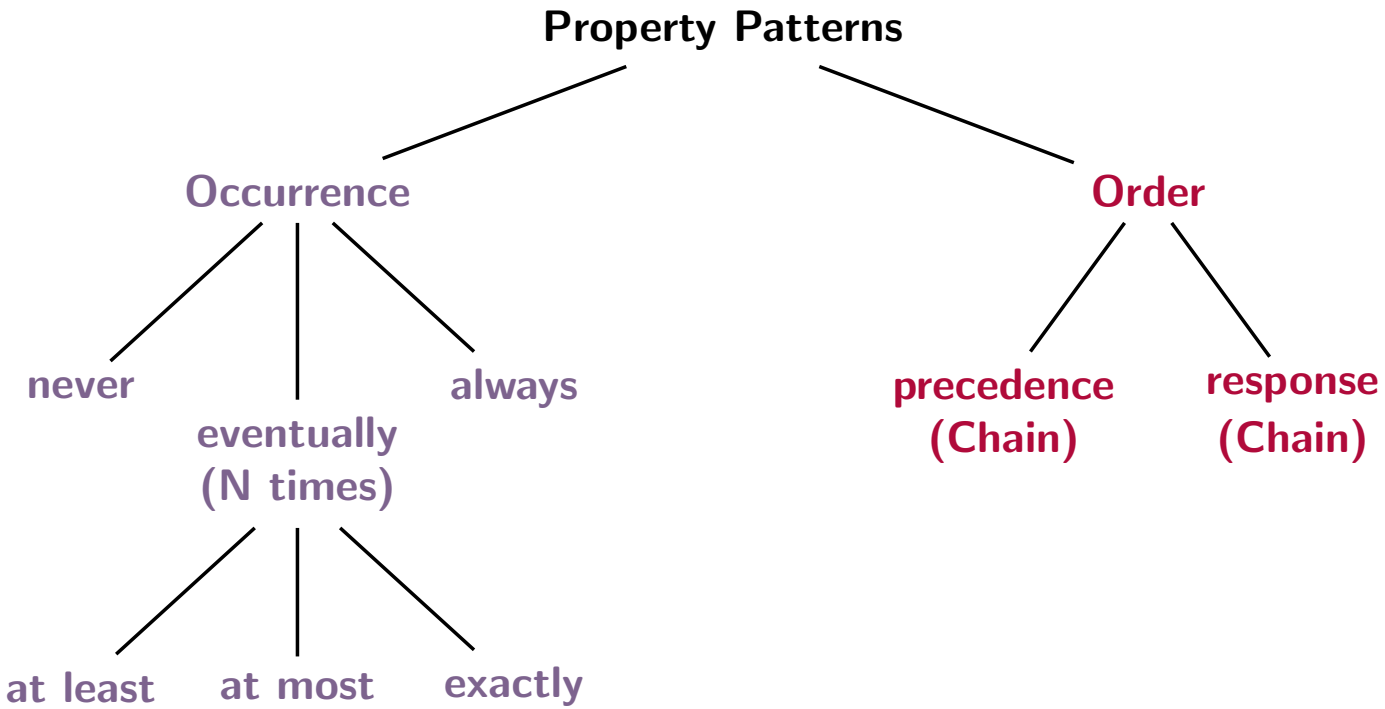


Property patterns

Occurrence patterns : Absence/Presence of events

Order patterns : Order between events

Patterns

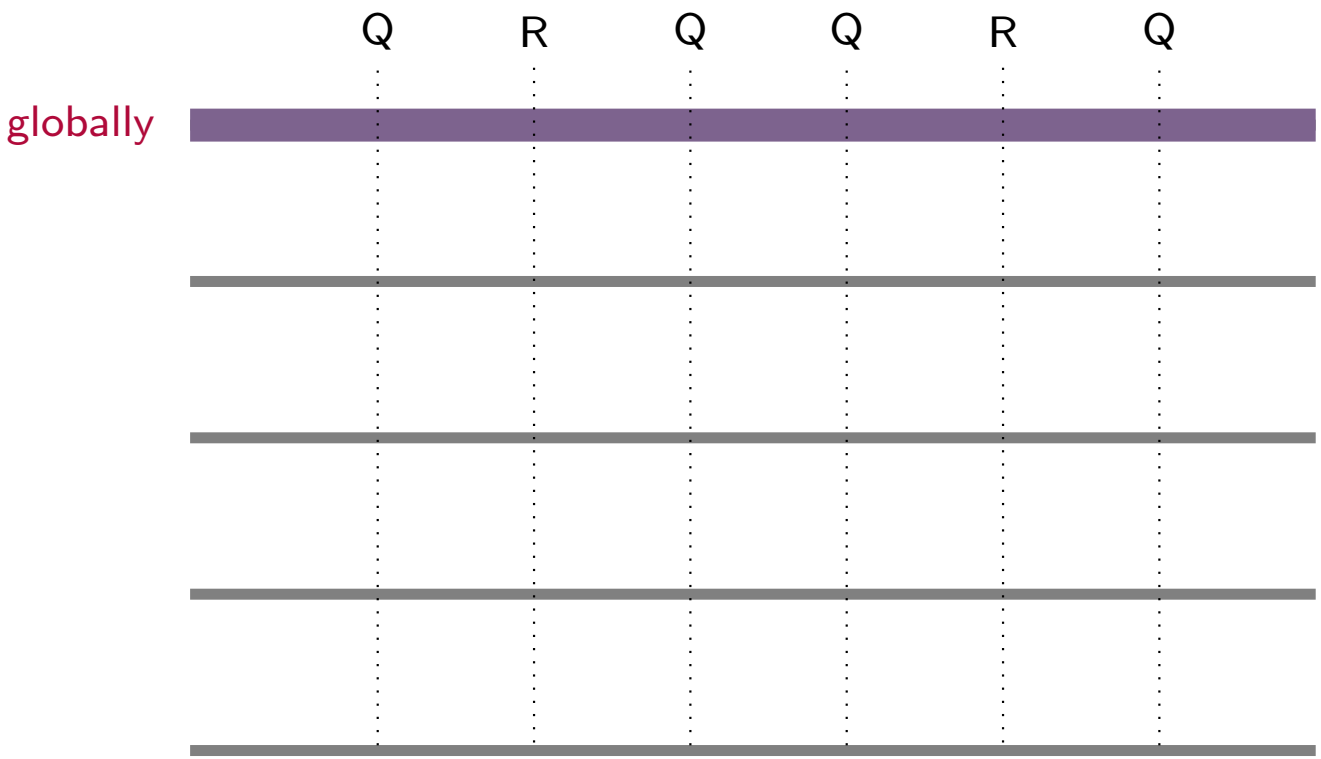


Property patterns

Occurrence patterns : Absence/Presence of events

Order patterns : Order between events

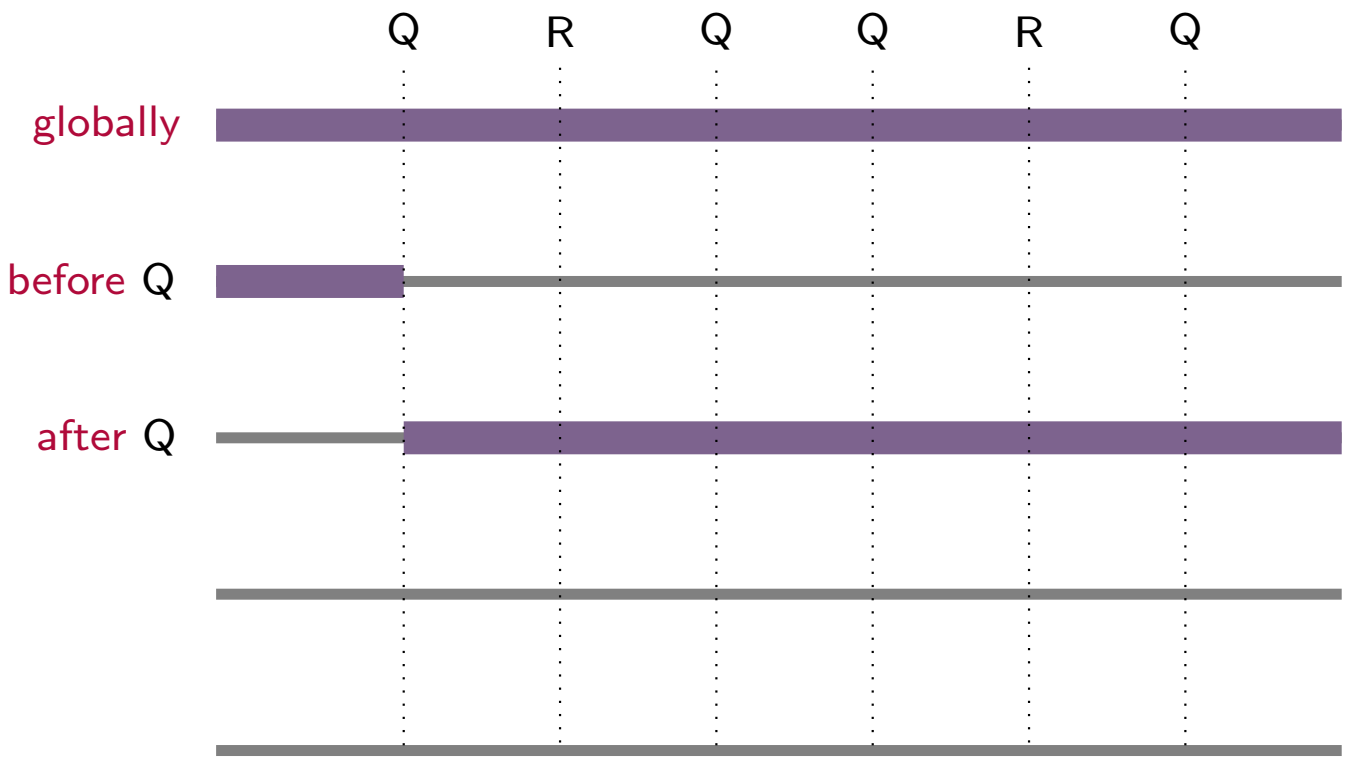
Scopes



Scopes

The Time Interval(s) over which a pattern holds

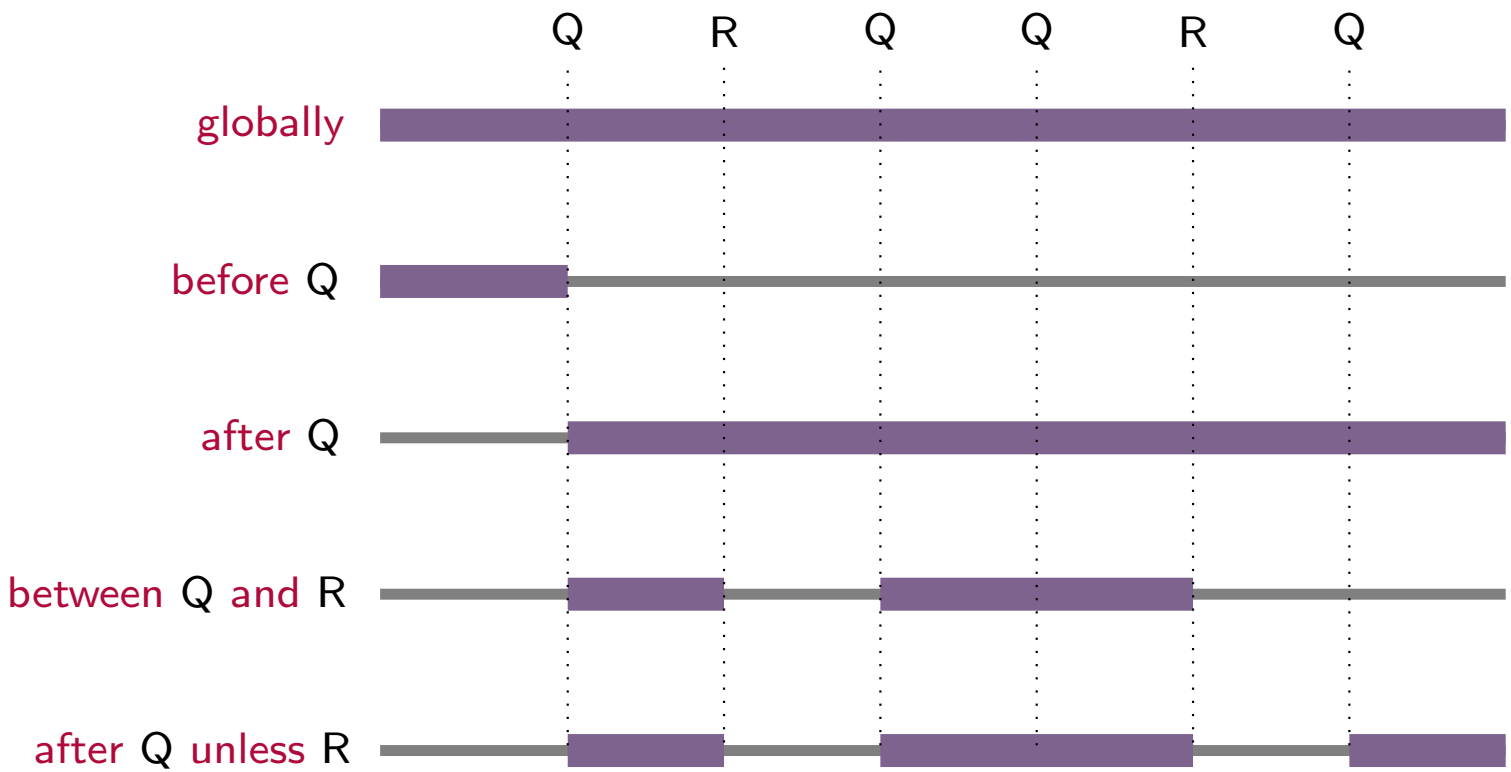
Scopes



Scopes

The Time Interval(s) over which a pattern holds

Scopes



Scopes

The Time Interval(s) over which a pattern holds

Dwyer et al.'s Spec Patterns

- back to our example :

while a user is connected (between Login and Logout),
each user's request (Req) must be responded (Resp)

LTL : $\Box((Login \wedge \Diamond Logout) \Rightarrow (Req \Rightarrow (\neg Logout \ \mathsf{U} \ Resp)) \ \mathsf{U} \ Logout)$

Dwyer et al.'s Spec Patterns

- back to our example :

while a user is connected (between Login and Logout),
each user's request (Req) must be responded (Resp)

LTL : $\Box((Login \wedge \Diamond Logout) \Rightarrow (Req \Rightarrow (\neg Logout \ \mathsf{U} \ Resp)) \ \mathsf{U} \ Logout)$

Pattern : Resp *responds to* Req (Response)
Scope : *between* Login *and* Logout (Between)

Dwyer et al.'s Spec Patterns

- back to our example :

while a user is connected (between Login and Logout),
each user's request (Req) must be responded (Resp)

LTL : $\Box((Login \wedge \Diamond Logout) \Rightarrow (Req \Rightarrow (\neg Logout \ \mathsf{U} \ Resp)) \ \mathsf{U} \ Logout)$

Pattern : Resp *responds to* Req (Response)
Scope : *between* Login *and* Logout (Between)

- easy to use
- over $\simeq$ 600 specs : 92% of coverage
  - `http://patterns.projects.cis.ksu.edu/`
- adopted by many works : Bandera, JTPL, Propel ...

Dwyer et al.'s Translational Semantics

- Besides the natural semantics of patterns and scopes
- Dwyer et al. provide formal semantics by translation :
  - $Pattern \times Scope \longrightarrow LTL$

Dwyer et al.'s Translational Semantics

- Besides the natural semantics of patterns and scopes
- Dwyer et al. provide formal semantics by translation :
  - $Pattern \times Scope \longrightarrow LTL$

- Example : $\{Response : P' \text{ responds to } P\} \times Scope \longrightarrow LTL$

Scope s	LTL
globally	$\Box(P \Rightarrow \Diamond P')$
before Q	$\Diamond Q \Rightarrow (P \Rightarrow (\neg Q \ \mathsf{U} \ (P' \wedge \neg Q))) \ \mathsf{U} \ Q$
after Q	$\Box(Q \Rightarrow \Box(P \Rightarrow \Diamond P'))$
between Q `and` R	$\Box((Q \wedge \neg R \wedge \Diamond R) \Rightarrow (P \Rightarrow (\neg R \ \mathsf{U} \ (P' \wedge \neg R))) \ \mathsf{U} \ R)$
after Q `unless` R	$\Box(Q \wedge \neg R \Rightarrow ((P \Rightarrow (\neg R \ \mathsf{U} \ (P' \wedge \neg R))) \ \mathsf{W} \ R)$

Dwyer et al.'s Translational Semantics

- Besides the natural semantics of patterns and scopes
- Dwyer et al. provide formal semantics by translation :
  - $Pattern \times Scope \longrightarrow LTL, CTL, \mu\text{-}calculus, QRE \ldots$
  - Library : `http://patterns.projects.cis.ksu.edu/`

- Example : $\{Response : P' \text{ responds to } P\} \times Scope \longrightarrow LTL$

Scope s	LTL
globally	$\Box(P \Rightarrow \Diamond P')$
before Q	$\Diamond Q \Rightarrow (P \Rightarrow (\neg Q \text{ U } (P' \wedge \neg Q))) \text{ U } Q$
after Q	$\Box(Q \Rightarrow \Box(P \Rightarrow \Diamond P'))$
between Q and R	$\Box((Q \wedge \neg R \wedge \Diamond R) \Rightarrow (P \Rightarrow (\neg R \text{ U } (P' \wedge \neg R))) \text{ U } R)$
after Q unless R	$\Box(Q \wedge \neg R \Rightarrow ((P \Rightarrow (\neg R \text{ U } (P' \wedge \neg R))) \text{ W } R)$

1^{st} Limit : Genericity

- Many of the Dwyer et al.'s patterns are generic :

1^{st} Limit : Genericity

- Many of the Dwyer et al.'s patterns are generic :
  - eventually P (at most | at least | exactly) k($\in \mathbb{N}$) times

1^{st} Limit : Genericity

- Many of the Dwyer et al.'s patterns are generic :
  - eventually P (at most | at least | exactly) k($\in \mathbb{N}$) times

- Because of the Translational Semantics, Dwyer et al. only consider :
  - eventually P (at least once) $k = 1$
  - eventually P at most 2 times $k = 2$

1^{st} Limit : Genericity

- Many of the Dwyer et al.'s patterns are generic :
  - eventually P (at most | at least | exactly) k($\in \mathbb{N}$) times
  - ($P_1, \ldots, P_{n(\in\mathbb{N})}$) precedes | responds to ($P'_1, \ldots, P'_{m(\in\mathbb{N})}$)
- Because of the Translational Semantics, Dwyer et al. only consider :
  - eventually P (at least once) $k = 1$
  - eventually P at most 2 times $k = 2$

1^{st} Limit : Genericity

- Many of the Dwyer et al.'s patterns are generic :
  - eventually P (at most | at least | exactly) k($\in \mathbb{N}$) times
  - $(P_1, \ldots, P_{n(\in\mathbb{N})})$ precedes | responds to $(P'_1, \ldots, P'_{m(\in\mathbb{N})})$
- Because of the Translational Semantics, Dwyer et al. only consider :
  - eventually P (at least once) $k = 1$
  - eventually P at most 2 times $k = 2$
  - P precedes | responds to P′ $n = 1, m = 1$
  - (P_1, P_2) precedes | responds to P′ $n = 2, m = 1$
  - P precedes | responds to (P'_1, P'_2) $n = 1, m = 2$

2^{nd} Limit : Scalability

- Dwyer et al.'s adopt Translational Semantics :
  - to Temporal Logics : $Pattern \times Scope \longrightarrow LTL$

2^{nd} Limit : Scalability

- Dwyer et al.'s adopt Translational Semantics :
  - to Temporal Logics : $Pattern \times Scope \longrightarrow LTL$
  - 10 patterns $\times$ 5 scopes $=$ 50 LTL formulas

2^{nd} Limit : Scalability

- Dwyer et al.'s adopt Translational Semantics :
  - to Temporal Logics : $Pattern \times Scope \longrightarrow LTL$
  - 10 patterns $\times$ 5 scopes $=$ 50 LTL formulas
- Dwyer et al. informally propose many variants to increase expressiveness and improve the 92% coverage :
  - Patterns variants
  - Scopes variants
  - `http://patterns.projects.cis.ksu.edu/`

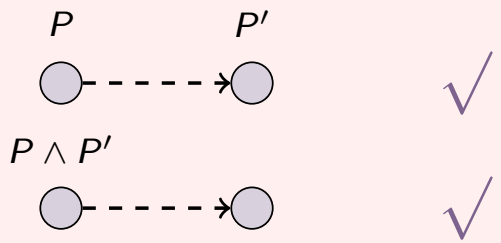
2^{nd} Limit : Scalability

- Dwyer et al.'s adopt Translational Semantics :
  - to Temporal Logics : $Pattern \times Scope \longrightarrow LTL$
  - 10 patterns $\times$ 5 scopes $=$ 50 LTL formulas
- Dwyer et al. informally propose many variants to increase expressiveness and improve the 92% coverage :
  - Patterns variants
  - Scopes variants
  - `http://patterns.projects.cis.ksu.edu/`
- **without** translating variants :
  - 14 patterns $\times$ 21 scopes $\simeq$ **300** combinations

Patterns Variants

Order Patterns Semantics

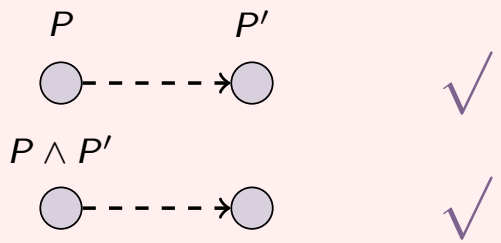
- Dwyer et al. consider **non strict** semantics :
  - ex : *P′ responds to P*



Patterns Variants

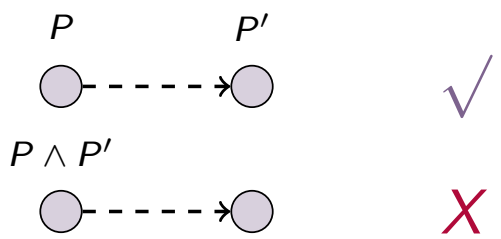
Order Patterns Semantics

- Dwyer et al. consider **non strict** semantics :
  - ex : P' *responds to* P



✓
✓

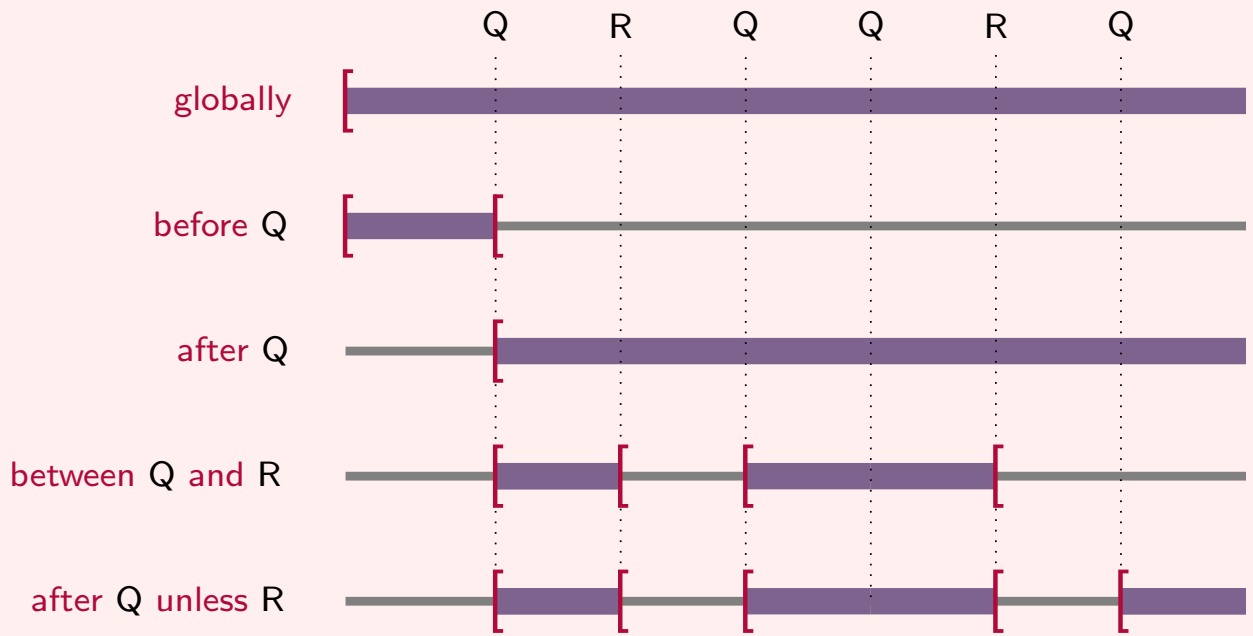
1. Variant : **strict** Order Patterns
   - ex : P' *responds*$_{strictly}$ *to* P



✓
X

Scopes Variants

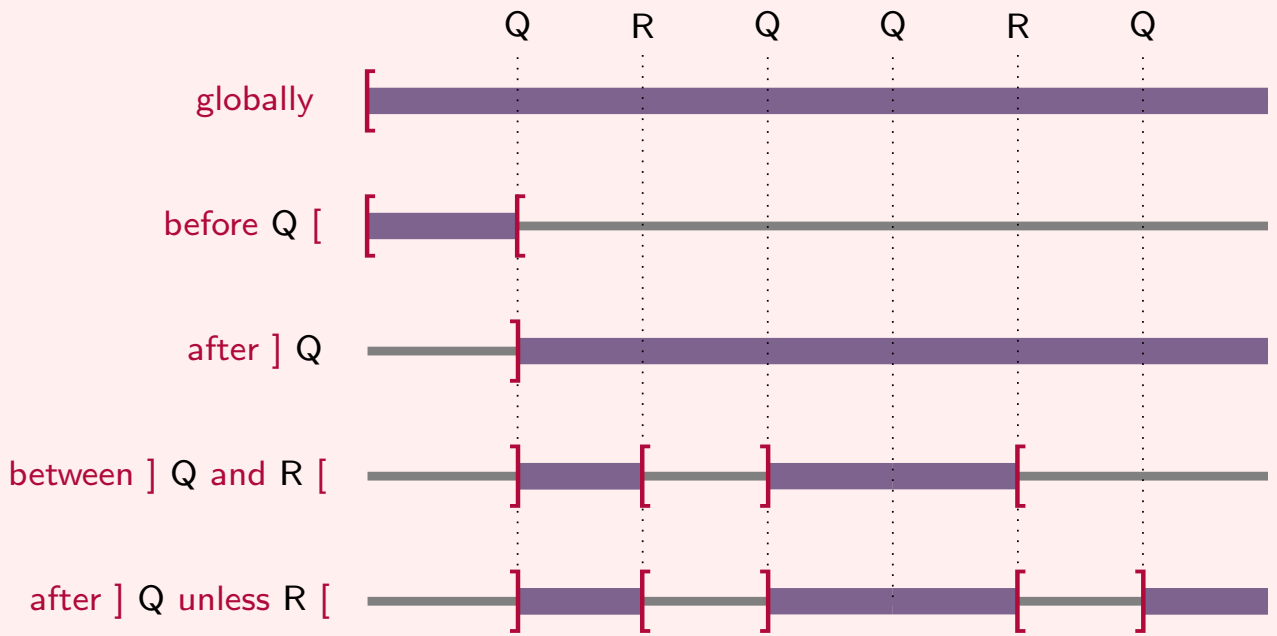
Scopes Semantics



- Dwyer et al. consider right-open intervals
  - [_ , _ [: left-closed and right-open

Scopes Variants

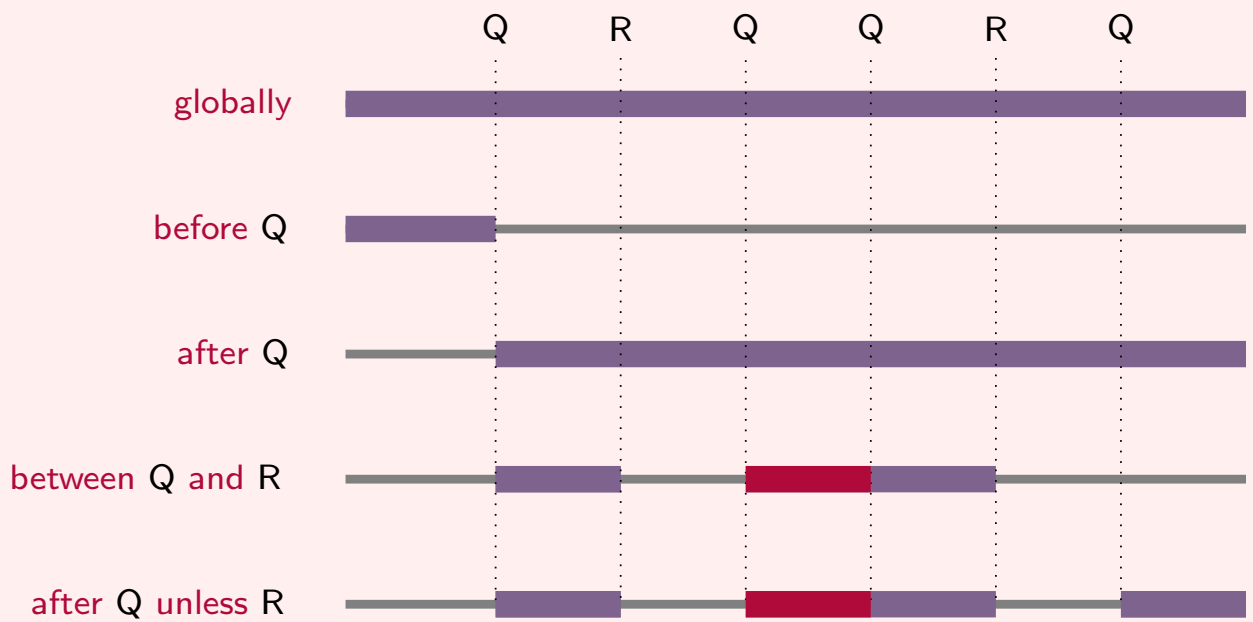
Scopes Semantics



3. Variant : other intervals [_ , _],] _ , _ [et] _ , _]
   - ex : open intervals] _ , _ [

Scopes Variants

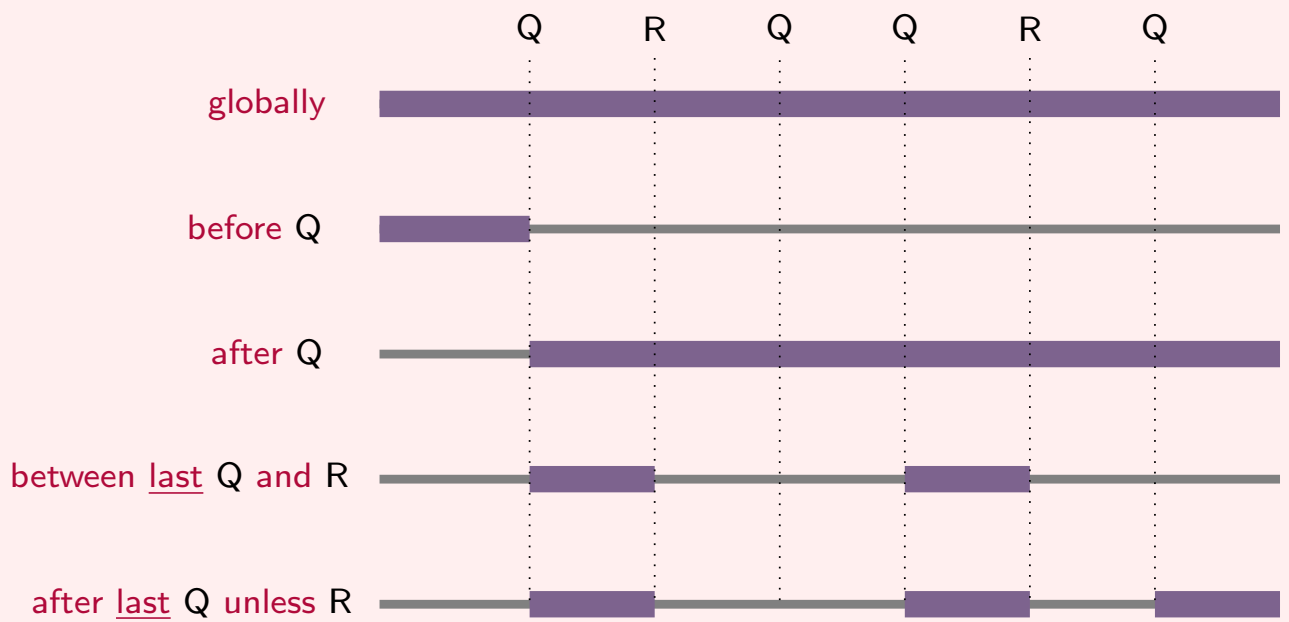
Scopes Semantics



- Dwyer et al. consider the first occurrence

Scopes Variants

Scopes Semantics



4. Variant : select the last occurrence

3^{rd} Limit : Faithfulness

- Patterns and Scopes have intuitive natural semantics

3^{rd} Limit : Faithfulness

- Patterns and Scopes have intuitive natural semantics
- Are the Translational Semantics faithful to the natural one ?

3^{rd} Limit : Faithfulness

- Patterns and Scopes have intuitive natural semantics
- Are the Translational Semantics faithful to the natural one ?
  - Peer-review among the Dwyer et al. project members
  - Model-checking the LTL formulas against some (un)satisfying scenarios (w.r.t. natural semantics)

3rd Limit : Faithfulness

- Patterns and Scopes have intuitive natural semantics
- Are the Translational Semantics faithful to the natural one ?
  - Peer-review among the Dwyer et al. project members
  - Model-checking the LTL formulas against some (un)satisfying scenarios (w.r.t. natural semantics)
- Common issue between natural and formal semantics :
  - No way ! to measure faithfulness !

3^{rd} Limit : Faithfulness

- Dwyer et al. build the Property Specification Language over a clear separation between Patterns and Scopes
  - A Pattern (resp. Scope) has a unique natural semantics regardless the Scope (resp. Pattern) with which it is combined

3^{rd} Limit : Faithfulness $\Longrightarrow$ Homogeneity

- Dwyer et al. build the Property Specification Language over a clear separation between Patterns and Scopes
  - A Pattern (resp. Scope) has a unique natural semantics regardless the Scope (resp. Pattern) with which it is combined
- Does the Translational Semantics keep Homogeneity ?
  - Translational Semantics flattens the key separation
  - ex : a Pattern is translated 5 times and may have up to 5 different interpretations

3^{rd} Limit : Faithfulness $\Longrightarrow$ Homogeneity

- Dwyer et al. build the Property Specification Language over a clear separation between Patterns and Scopes
  - A Pattern (resp. Scope) has a unique natural semantics regardless the Scope (resp. Pattern) with which it is combined
- Does the Translational Semantics keep Homogeneity ?
  - Translational Semantics flattens the key separation
  - ex : a Pattern is translated 5 times and may have up to 5 different interpretations
- Homogeneity is part of Faithfulness
  - a part that we will be able to measure !

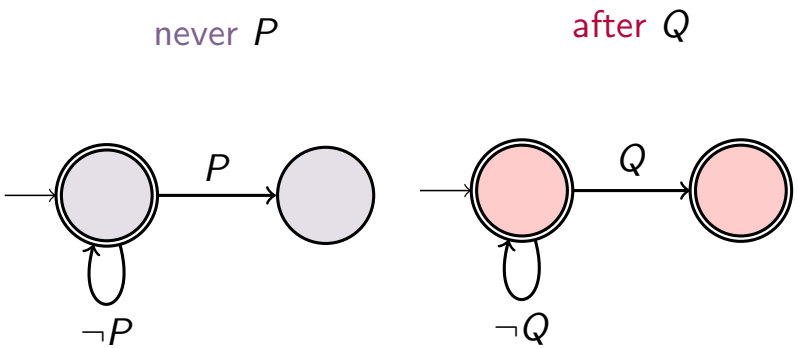
Plan

Principle [Taha et al., 2015]

- Compositional Semantics are based on Büchi automata

Principle [Taha et al., 2015]

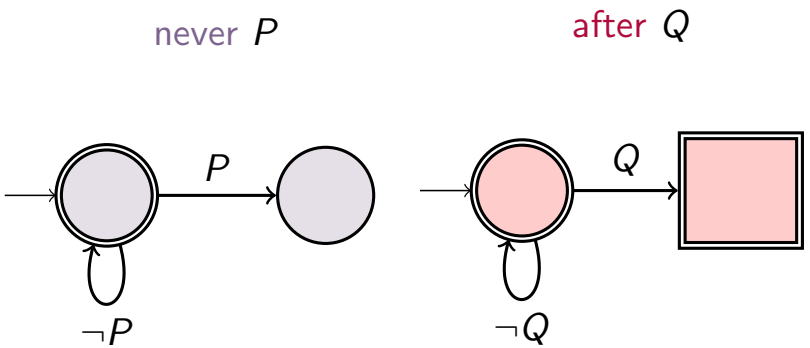
- Compositional Semantics are based on Büchi automata



1. Pattern and Scope are Büchi automata

Principle [Taha et al., 2015]

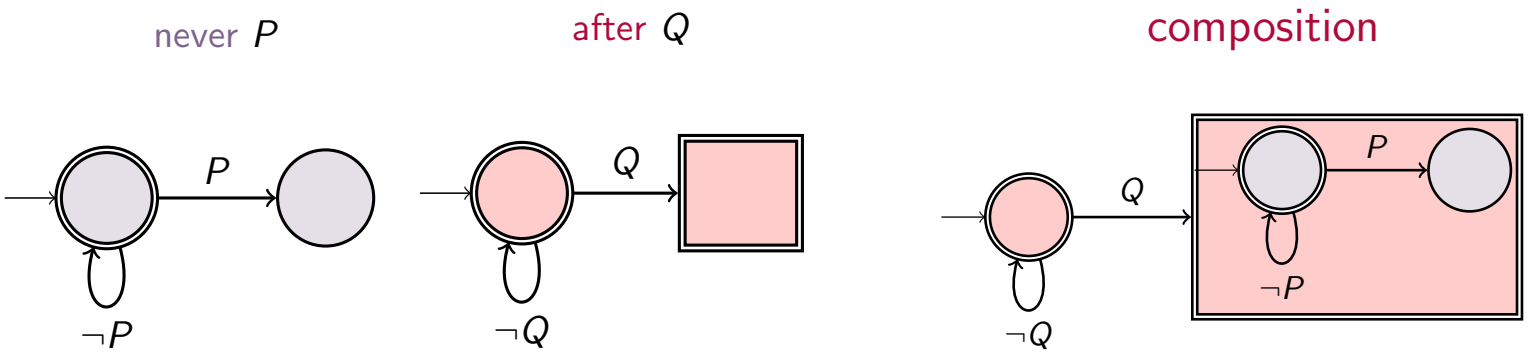
- Compositional Semantics are based on Büchi automata



1. Pattern and Scope are Büchi automata
2. we distinguish the composition state within the Scope

Principle [Taha et al., 2015]

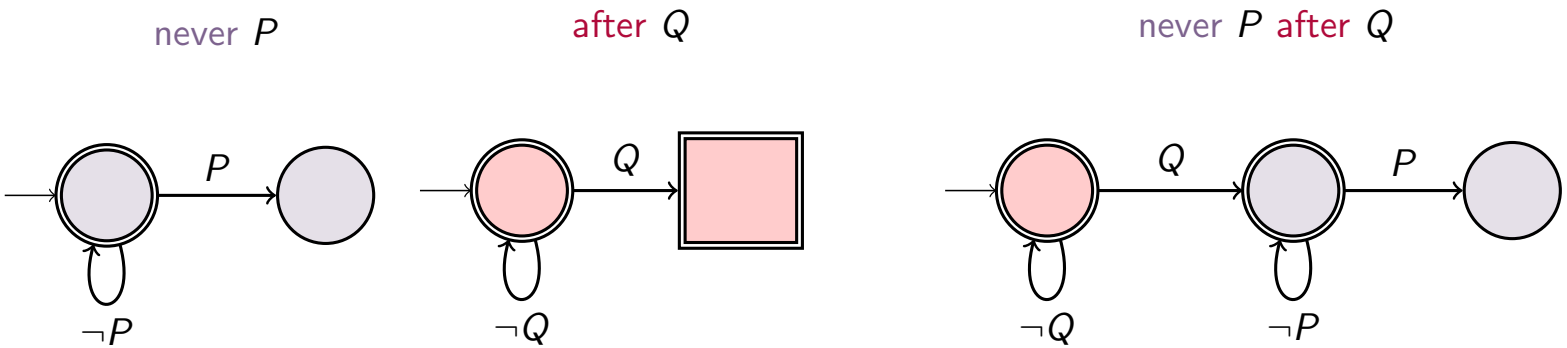
- Compositional Semantics are based on Büchi automata



1. Pattern and Scope are Büchi automata
2. we distinguish the composition state within the Scope
3. we substitute the composition state by the Pattern automaton

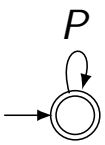
Principle [Taha et al., 2015]

- Compositional Semantics are based on Büchi automata

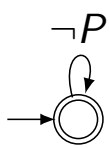


1. Pattern and Scope are Büchi automata
2. we distinguish the composition state within the Scope
3. we substitute the composition state by the Pattern automaton
4. We obtain the automaton of the property : [Pattern/Scope]
   - by applying composition rules

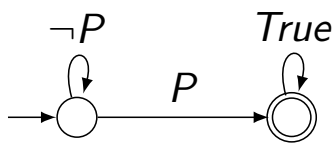
Patterns Automata



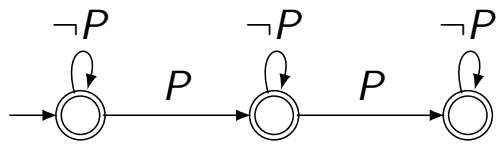
always P



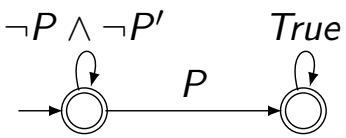
never P



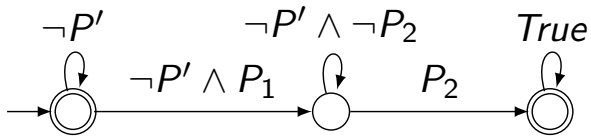
eventually P



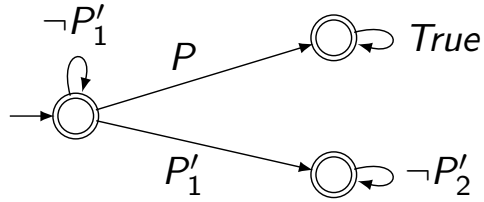
eventually P at most 2 times



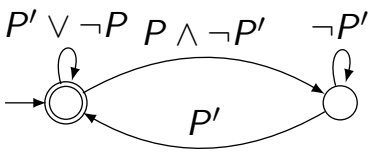
P precedes P′



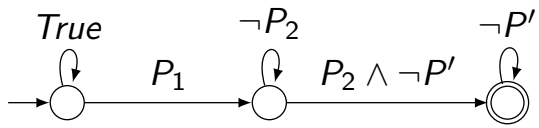
(P_1, P_2) precedes P′



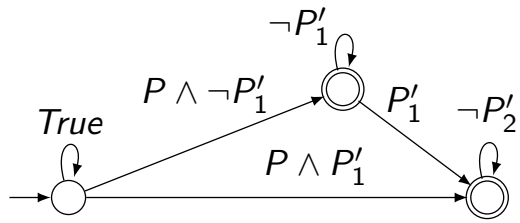
P precedes (P'_1, P'_2)



P′ responds to P

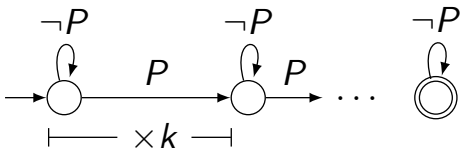


¬[P′ responds to (P_1, P_2)]

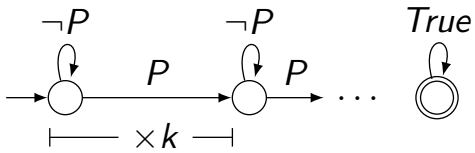


¬[(P'_1, P'_2) responds to P]

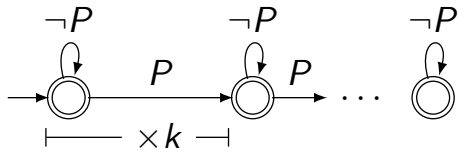
Patterns Automata + generic



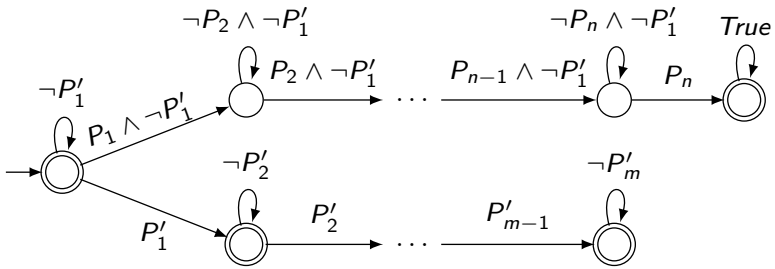
eventually P k times



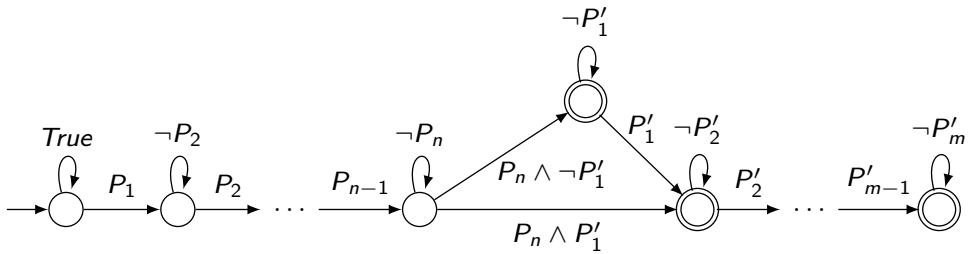
eventually P at least k times



eventually P at most k times

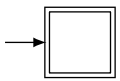


$(P_1, \ldots, P_n)$ precedes $(P'_1, \ldots, P'_m)$

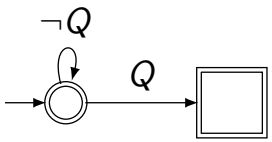


$\neg[(P'_1, \ldots, P'_m)$ responds to $(P_1, \ldots, P_n)]$

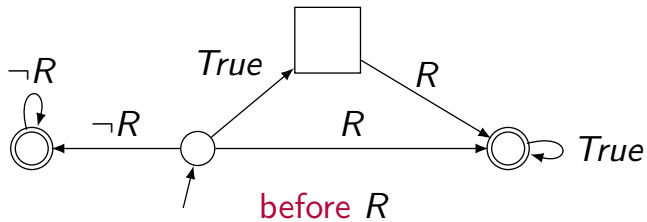
Scopes Automata



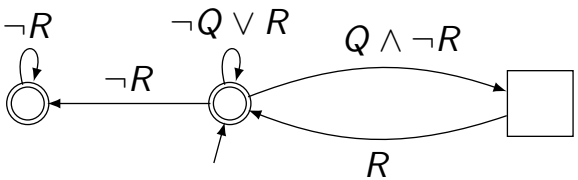
globally



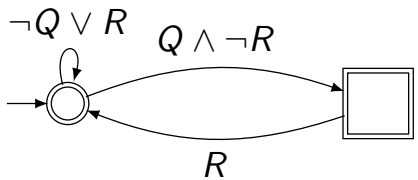
after Q



before R

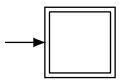


between Q and R

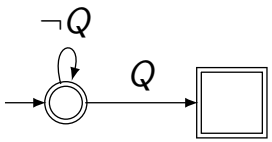


after Q unless R

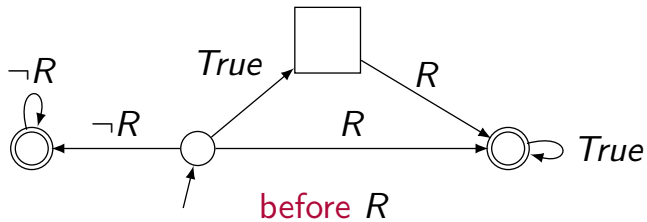
Scopes Automata + variants



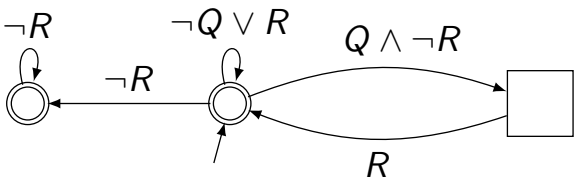
globally



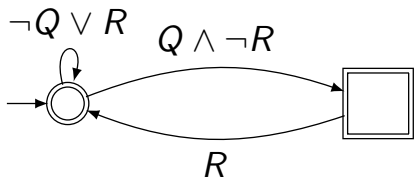
after Q



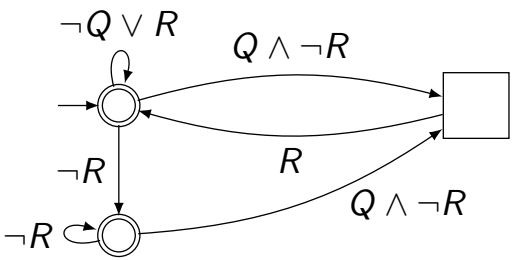
before R



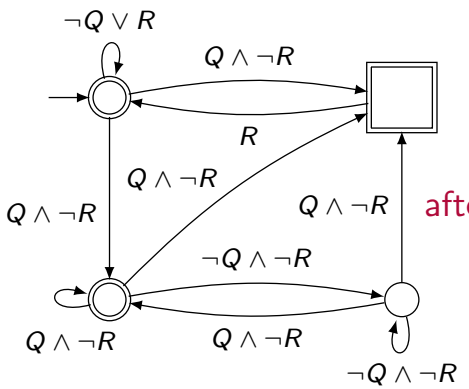
between Q and R



after Q unless R



between last Q and R



after last Q unless R

Composition Rules : States

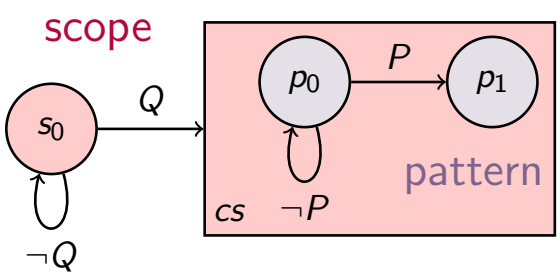
$$\begin{cases} \text{pattern} & \stackrel{\text{def}}{=} (Q_p, \quad init_p, F_p, T_p) \quad with \quad init_p \in Q_p, F_p \subseteq Q_p \\ \text{scope} & \stackrel{\text{def}}{=} (Q_s \cup \{cs\}, init_s, F_s, T_s) \quad with \quad init_s \in Q_s \cup \{cs\}, F_s \subseteq Q_s \cup \{cs\} \end{cases}$$

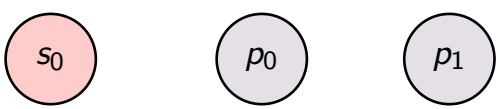
Composition Rules : States

$$\begin{cases} \text{pattern} & \stackrel{\text{def}}{=} (Q_p, \quad init_p, F_p, T_p) \quad with \quad init_p \in Q_p, F_p \subseteq Q_p \\ \text{scope} & \stackrel{\text{def}}{=} (Q_s \cup \{cs\}, init_s, F_s, T_s) \quad with \quad init_s \in Q_s \cup \{cs\}, F_s \subseteq Q_s \cup \{cs\} \end{cases}$$

pattern/scope $\stackrel{\text{def}}{=} (Q, init, F, T)$

$Q = Q_p \cup Q_s$





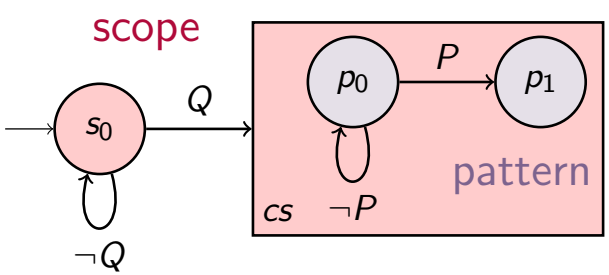
Composition Rules : States

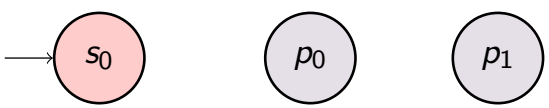
$$\begin{cases} \text{pattern} & \stackrel{\text{def}}{=} (Q_p, \quad init_p, F_p, T_p) \quad with \quad init_p \in Q_p, F_p \subseteq Q_p \\ \text{scope} & \stackrel{\text{def}}{=} (Q_s \cup \{cs\}, init_s, F_s, T_s) \quad with \quad init_s \in Q_s \cup \{cs\}, F_s \subseteq Q_s \cup \{cs\} \end{cases}$$

pattern/scope $\stackrel{\text{def}}{=} (Q, init, F, T)$

$$Q = Q_p \cup Q_s$$

$$init = \begin{cases} init_s & \text{if} \quad init_s \neq cs \end{cases}$$





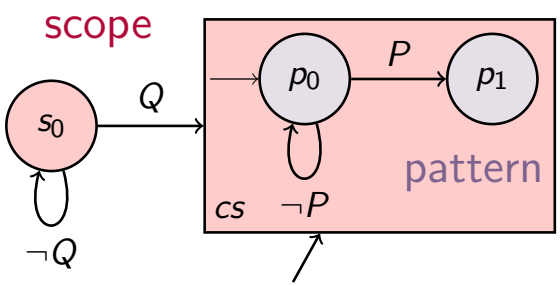
Composition Rules : States

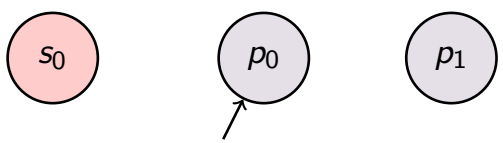
$$\begin{cases} \text{pattern} & \stackrel{\text{def}}{=} (Q_p, \quad init_p, F_p, T_p) \quad with \quad init_p \in Q_p, F_p \subseteq Q_p \\ \text{scope} & \stackrel{\text{def}}{=} (Q_s \cup \{cs\}, init_s, F_s, T_s) \quad with \quad init_s \in Q_s \cup \{cs\}, F_s \subseteq Q_s \cup \{cs\} \end{cases}$$

pattern/scope $\stackrel{\text{def}}{=} (Q, init, F, T)$

$$Q = Q_p \cup Q_s$$

$$init = \begin{cases} init_s & \text{if} \quad init_s \neq cs \\ init_p & \text{else} \end{cases}$$





Composition Rules : States

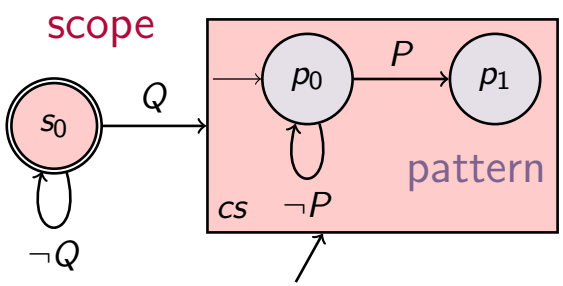
$$\begin{cases} \text{pattern} & \stackrel{\text{def}}{=} (Q_p, \quad init_p, F_p, T_p) \quad with \quad init_p \in Q_p, F_p \subseteq Q_p \\ \text{scope} & \stackrel{\text{def}}{=} (Q_s \cup \{cs\}, init_s, F_s, T_s) \quad with \quad init_s \in Q_s \cup \{cs\}, F_s \subseteq Q_s \cup \{cs\} \end{cases}$$

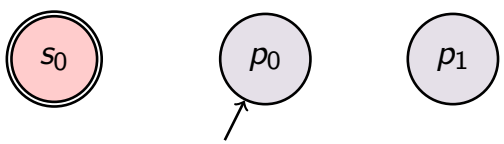
$$\text{pattern/scope} \stackrel{\text{def}}{=} (Q, init, F, T)$$

$$Q = Q_p \cup Q_s$$

$$init = \begin{cases} init_s & \text{if} \quad init_s \neq cs \\ init_p & \text{else} \end{cases}$$

$$F = \begin{cases} F_s & \text{if} \quad cs \notin F_s \\ \end{cases}$$





Composition Rules : States

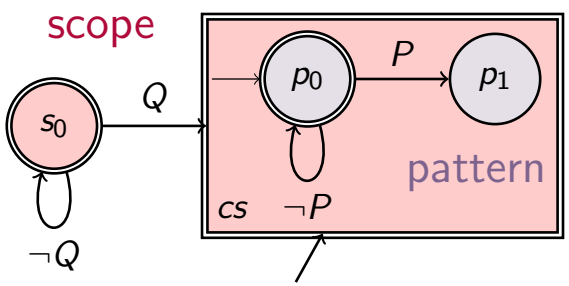
$$\begin{cases} \text{pattern} & \stackrel{\text{def}}{=} (Q_p, \quad init_p, F_p, T_p) \quad with \quad init_p \in Q_p, F_p \subseteq Q_p \\ \text{scope} & \stackrel{\text{def}}{=} (Q_s \cup \{cs\}, init_s, F_s, T_s) \quad with \quad init_s \in Q_s \cup \{cs\}, F_s \subseteq Q_s \cup \{cs\} \end{cases}$$

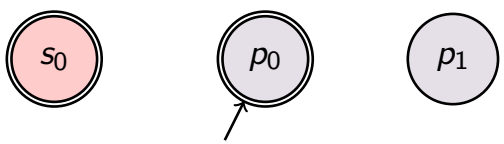
$$\text{pattern/scope} \stackrel{\text{def}}{=} (Q, init, F, T)$$

$$Q = Q_p \cup Q_s$$

$$init = \begin{cases} init_s & \text{if} \quad init_s \neq cs \\ init_p & \text{else} \end{cases}$$

$$F = \begin{cases} F_s & \text{if} \quad cs \notin F_s \\ (F_s \setminus \{cs\}) \cup F_p & \text{else} \end{cases}$$

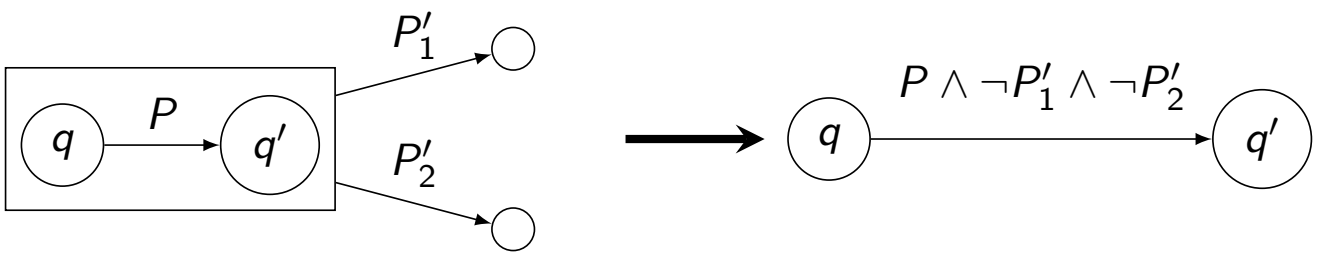




Composition Rules : Transitions (1/3)

1. Transitions from pattern to pattern : restriction

$$\frac{q \xrightarrow{P} q' \in T_p}{q \xrightarrow{P \wedge R} q' \in T} \quad \text{avec} \quad \begin{cases} R & = \bigwedge_{P' \in Out(cs)} \neg P' \\ Out(cs) & = \{P' \mid \exists q'', q'' \in Q_s, cs \xrightarrow{P'} q'' \in T_s\} \end{cases}$$



Composition Rules : Transitions (1/3)

1. Transitions from pattern to pattern : restriction

$$\frac{q \xrightarrow{P} q' \in T_p}{q \xrightarrow{P \wedge R} q' \in T} \quad \text{avec} \quad \begin{cases} R & = \bigwedge_{P' \in Out(cs)} \neg P' \\ Out(cs) & = \{P' \mid \exists q'', q'' \in Q_s, cs \xrightarrow{P'} q'' \in T_s\} \end{cases}$$

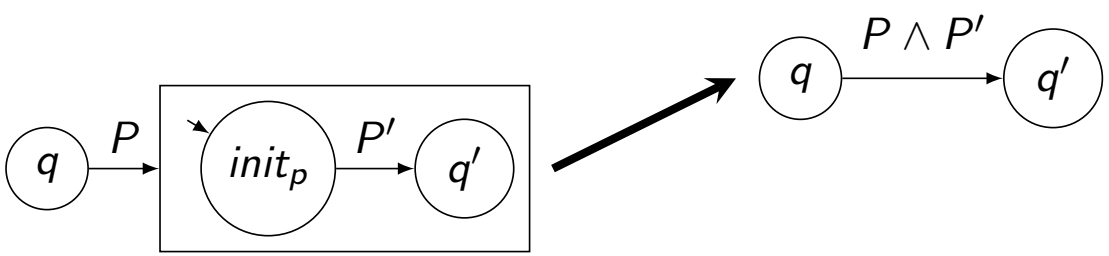
2. Transitions from scope to scope :

$$\frac{q \xrightarrow{P} q' \in T_s,\ q,\ q' \in Q_s}{q \xrightarrow{P} q' \in T}$$

Composition Rules : Transitions (2/3)

3. Transitions from scope to pattern :

$$\frac{q \xrightarrow{P} cs \in T_s,\ q \in Q_s,\ init_p \xrightarrow{P'} q' \in T_p}{q \xrightarrow{P \wedge P'} q' \in T}$$



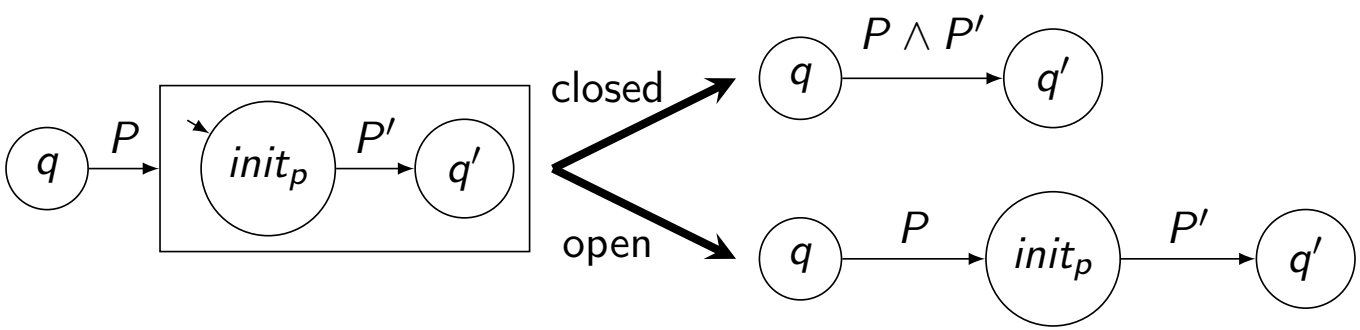
Composition Rules : Transitions (2/3)

3. Transitions from scope to pattern :
   - left-closed interval :

$$\frac{q \xrightarrow{P} cs \in T_s,\ q \in Q_s,\ init_p \xrightarrow{P'} q' \in T_p}{q \xrightarrow{P \wedge P'} q' \in T}$$

   - left-open interval (variant) :

$$\frac{q \xrightarrow{P} cs \in T_s,\ q \in Q_s}{q \xrightarrow{P} init_p \in T}$$



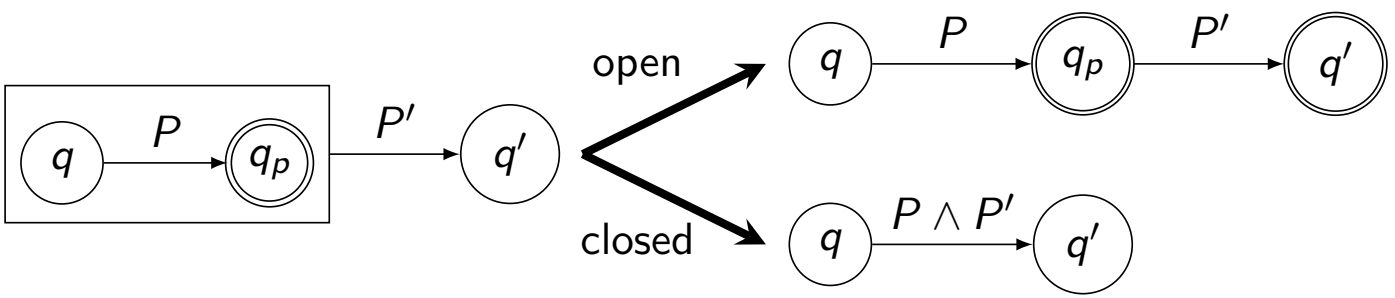
Composition Rules : Transitions (3/3)

4. Transitions from pattern to scope :
   - right-open interval :

$$\frac{cs \xrightarrow{P} q' \in T_s,\ q' \in Q_s,\ q \in F_p}{q \xrightarrow{P} q' \in T}$$

   - right-closed interval (variant) :

$$\frac{q \xrightarrow{P} q_p \in T_p,\ q_p \in F_p,\ cs \xrightarrow{P'} q' \in T_s,\ q' \in Q_s}{q \xrightarrow{P \wedge P'} q' \in T}$$



Example :

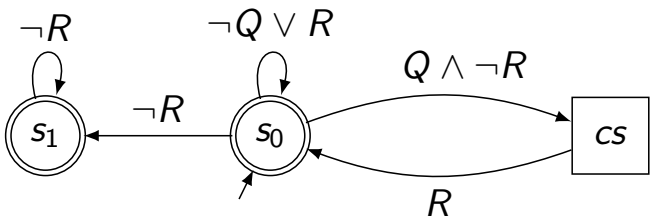
while a user is connected (between Login and Logout), each user request (Req) must be responded (Resp)

Pattern	:	Resp *responds to* Req	(Response)
Scope	:	*between* [Login *and* Logout [	(Between)

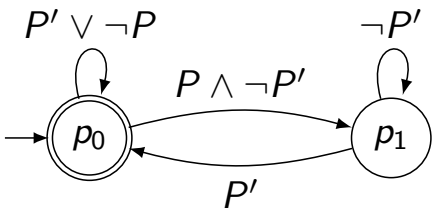
Example :

while a user is connected (between Login and Logout), each user request (Req) must be responded (Resp)

Pattern : Resp *responds to* Req (Response)
Scope : *between* [Login *and* Logout [(Between)



between Q and R

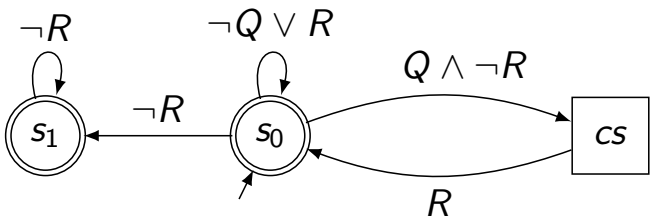


P responds to P'

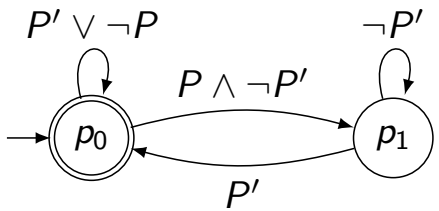
Example :

while a user is connected (between Login and Logout), each user request (Req) must be responded (Resp)

Pattern : Resp *responds to* Req (Response)
Scope : *between* [Login *and* Logout [(Between)



between Q and R



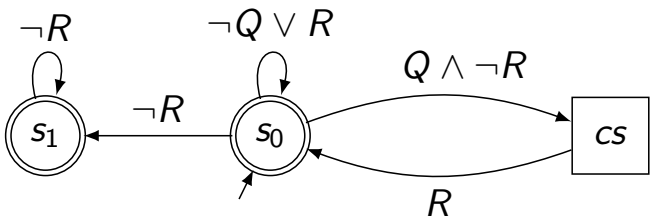
P responds to P'

P responds to P' between [Q and R [

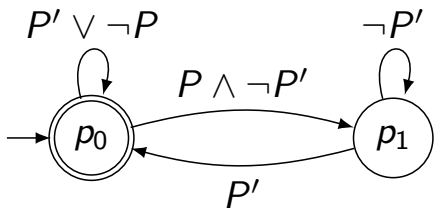
Example :

while a user is connected (between Login and Logout), each user request (Req) must be responded (Resp)

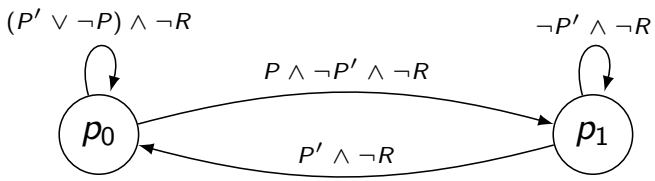
Pattern : Resp *responds to* Req (Response)
Scope : *between* [Login *and* Logout [(Between)



between Q and R



P responds to P'

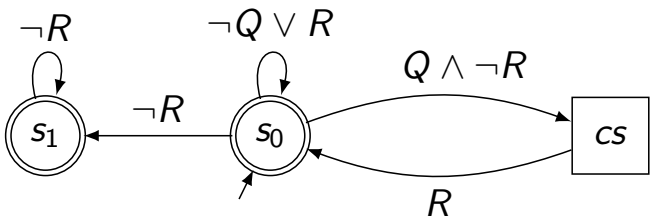


P responds to P' between [Q and R [

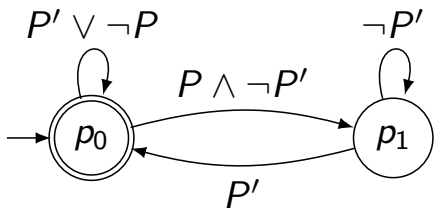
Example :

while a user is connected (between Login and Logout), each user request (Req) must be responded (Resp)

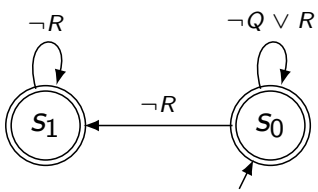
Pattern : Resp *responds to* Req (Response)
Scope : *between* [Login *and* Logout [(Between)

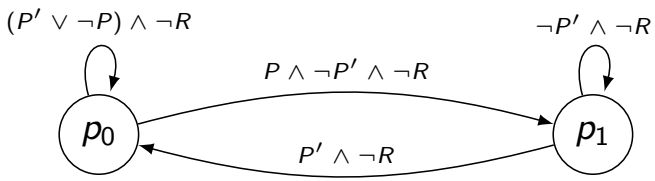


between Q and R



P responds to P'



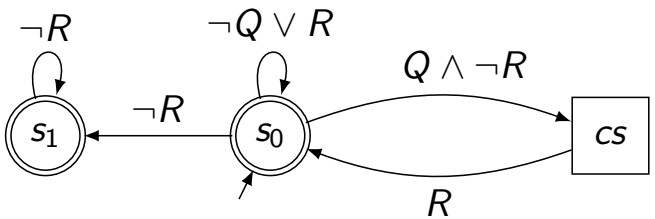


P responds to P' between [Q and R [

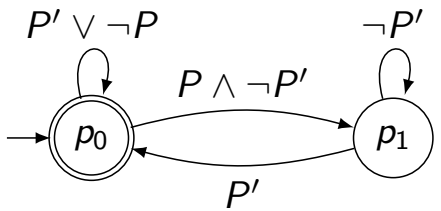
Example :

while a user is connected (between Login and Logout), each user request (Req) must be responded (Resp)

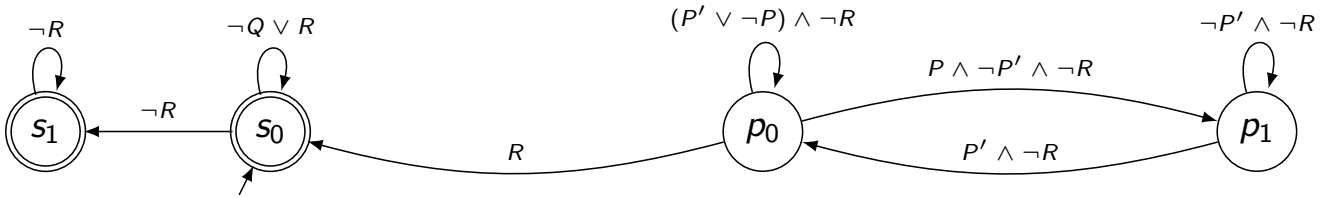
Pattern : Resp *responds to* Req (Response)
Scope : *between* [Login *and* Logout [(Between)



between Q and R



P responds to P'

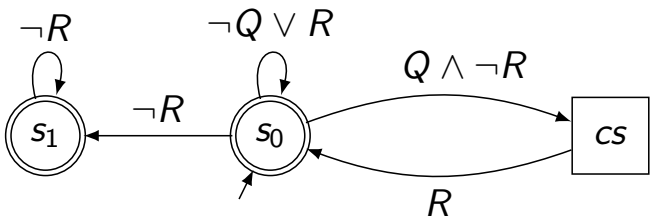


P responds to P' between [Q and R [

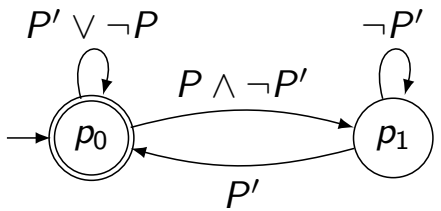
Example :

while a user is connected (between Login and Logout), each user request (Req) must be responded (Resp)

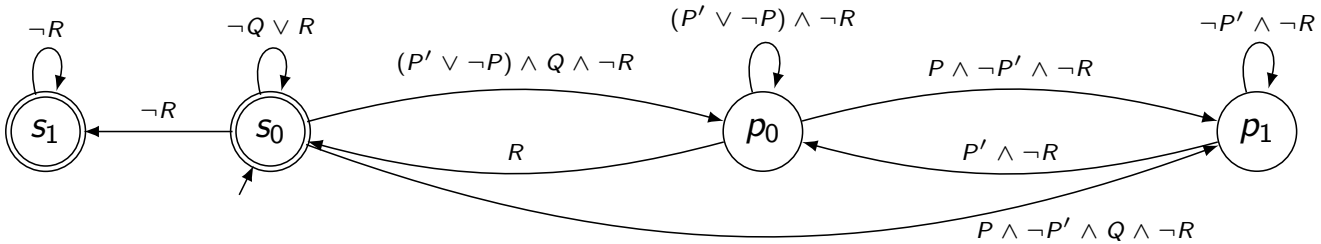
Pattern : Resp *responds to* Req (Response)
Scope : *between* [Login *and* Logout [(Between)



between Q and R



P responds to P'



P responds to P' between [Q and R [

Composition vs Translation

- Translational Semantics by Dwyer et al. :
  - n patterns $\times$ m scopes $\simeq$ **n** $\times$ **m** combinations to translate
  - finite support for generic patterns $k \in 1..N$

Composition vs Translation

- Translational Semantics by Dwyer et al. :
  - n patterns $\times$ m scopes $\simeq$ **n** $\times$ **m** combinations to translate
  - finite support for generic patterns $k \in 1..N$
- Compositional semantics :
  - n patterns $\times$ m scopes $\simeq$ **n** $+$ **m** automata to define
  - native support for generic patterns $k \in \mathbb{N}$

Composition vs Translation

- Translational Semantics by Dwyer et al. :
  - n patterns $\times$ m scopes $\simeq$ **n** $\times$ **m** combinations to translate
  - finite support for generic patterns $k \in 1..N$
- Compositional semantics :
  - n patterns $\times$ m scopes $\simeq$ **n** $+$ **m** automata to define
  - native support for generic patterns $k \in \mathbb{N}$

Composition **2 : 0** **Translation**

- Compositional Semantics brings Genericity and Scalability

Composition vs Translation

- Translational Semantics by Dwyer et al. :
  - n patterns $\times$ m scopes $\simeq$ **n** $\times$ **m** combinations to translate
  - finite support for generic patterns $k \in 1..N$
- Compositional semantics :
  - n patterns $\times$ m scopes $\simeq$ **n** $+$ **m** automata to define
  - native support for generic patterns $k \in \mathbb{N}$

Composition **2 : 0** **Translation**

- Compositional Semantics brings Genericity and Scalability
- what about Homogeneity ?

Homogeneity

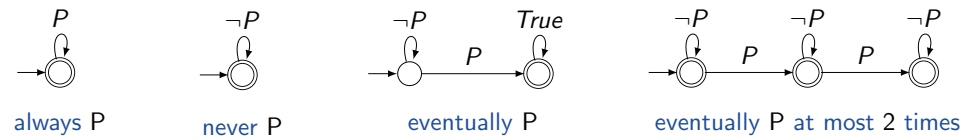
	globally	before R	after Q	between Q and R	after Q unless R
always P	$\Box P$	$\Diamond R \Rightarrow (P \text{ U } R)$	$\Box(Q \Rightarrow \Box P)$	$\Box((Q \wedge \neg R \wedge \Diamond R) \Rightarrow (P \text{ U } R))$	$\Box((Q \wedge \neg R) \Rightarrow (P \text{ W } R))$
never P	$\Box\neg P$	LTL	LTL	LTL	LTL
eventually P	$\Diamond P$	LTL	LTL	LTL	LTL
eventually P at most 2	$(\neg P \text{ W } (P \text{ W } (\neg P \text{ W } (P \text{ W } \Box\neg P))))$	LTL	LTL	LTL	LTL
P precedes P′	$\neg P' \text{ W } P$	LTL	LTL	LTL	LTL
(P_1, P_2) precedes P′	$\Diamond P' \Rightarrow (\neg P' \text{ U } (P_1 \wedge \neg P' \wedge \bigcirc(\neg P' \text{ U } P_2)))$	LTL	LTL	LTL	LTL
P precedes (P_1', P_2')	$(\Diamond(P_1' \wedge \bigcirc\Diamond P_2')) \Rightarrow ((\neg P_1') \text{ U } P))$	LTL	LTL	LTL	LTL
P′ responds to P	$\Box(P \Rightarrow \Diamond P')$	LTL	LTL	LTL	LTL
(P_1', P_2') responds to P	$\Box(P \Rightarrow \Diamond(P_1' \wedge \bigcirc\Diamond P_2'))$	LTL	LTL	LTL	LTL
P′ responds to (P_1, P_2)	$\Box(P_1 \wedge \bigcirc\Diamond P_2 \Rightarrow \bigcirc(\Diamond(P_2 \wedge \Diamond P')))$	LTL	LTL	LTL	LTL

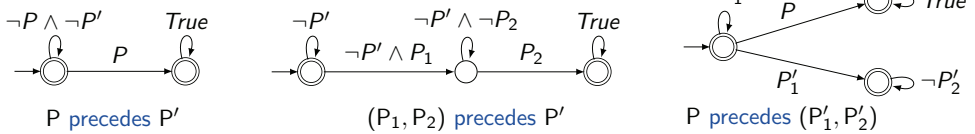
Approach :

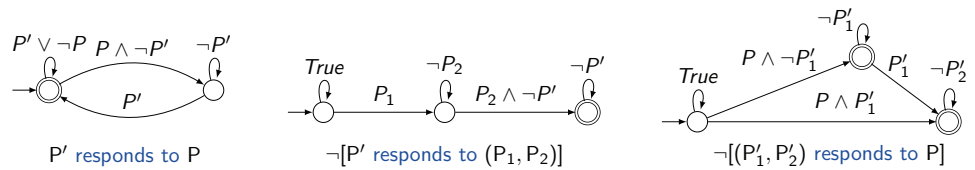
1. Collect the 50 LTL formulas given by Dwyer et al.

Homogeneity

	globally
always P	$\Box P$
never P	$\Box\neg P$
eventually P	$\Diamond P$
eventually P at most 2	$(\neg P \text{ W } (P \text{ W } (\neg P \text{ W } (P \text{ W } \Box\neg P))))$
P precedes P′	$\neg P' \text{ W } P$
(P_1, P_2) precedes P′	$\Diamond P' \Rightarrow (\neg P' \text{ U } (P_1 \wedge \neg P' \wedge \bigcirc(\neg P' \text{ U } P_2)))$
P precedes (P'_1, P'_2)	$(\Diamond(P'_1 \wedge \bigcirc\Diamond P'_2)) \Rightarrow ((\neg P'_1) \text{ U } P))$
P′ responds to P	$\Box(P \Rightarrow \Diamond P')$
(P'_1, P'_2) responds to P	$\Box(P \Rightarrow \Diamond(P'_1 \wedge \bigcirc\Diamond P'_2))$
P′ responds to (P_1, P_2)	$\Box(P_1 \wedge \bigcirc\Diamond P_2 \Rightarrow \bigcirc(\Diamond(P_2 \wedge \Diamond P')))$





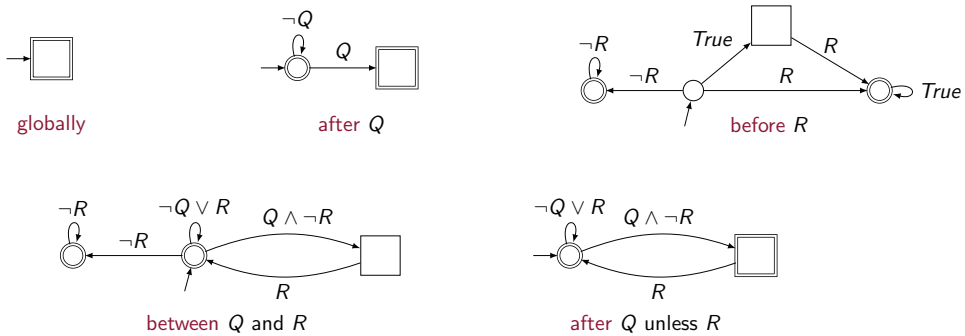


Approach :

1. Collect the 50 LTL formulas given by Dwyer et al.
2. Generate the Patterns automata (LTL2BA)

Homogeneity

	globally	before R	after Q	between Q and R	after Q unless R
always P	$\Box P$	$\Diamond R \Rightarrow (P \text{ U } R)$	$\Box(Q \Rightarrow \Box P)$	$\Box((Q \wedge \neg R \wedge \Diamond R) \Rightarrow (P \text{ U } R))$	$\Box((Q \wedge \neg R) \Rightarrow (P \text{ W } R))$



Approach :

1. Collect the 50 LTL formulas given by Dwyer et al.
2. Generate the Patterns automata (LTL2BA)
3. Generate the Scopes automata (LTL2BA)

Homogeneity

	globally	before R	after Q	between Q and R	after Q unless R
always P	$\Box P$	$\Diamond R \Rightarrow (P \text{ U } R)$	$\Box(Q \Rightarrow \Box P)$	$\Box((Q \wedge \neg R \wedge \Diamond R) \Rightarrow (P \text{ U } R))$	$\Box((Q \wedge \neg R) \Rightarrow (P \text{ W } R))$
never P	$\Box\neg P$	Composition	Composition	Composition	Composition
eventually P	$\Diamond P$	Composition	Composition	Composition	Composition
eventually P at most 2	$(\neg P \text{ W } (P \text{ W } (\neg P \text{ W } (P \text{ W } \Box\neg P))))$	Composition	Composition	Composition	Composition
P precedes P′	$\neg P' \text{ W } P$	Composition	Composition	Composition	Composition
(P_1, P_2) precedes P′	$\Diamond P' \Rightarrow (\neg P' \text{ U } (P_1 \wedge \neg P' \wedge \bigcirc(\neg P' \text{ U } P_2)))$	Composition	Composition	Composition	Composition
P precedes (P'_1, P'_2)	$(\Diamond(P'_1 \wedge \bigcirc\Diamond P'_2)) \Rightarrow ((\neg P'_1) \text{ U } P))$	Composition	Composition	Composition	Composition
P′responds to P	$\Box(P \Rightarrow \Diamond P')$	Composition	Composition	Composition	Composition
(P'_1, P'_2) responds to P	$\Box(P \Rightarrow \Diamond(P'_1 \wedge \bigcirc\Diamond P'_2))$	Composition	Composition	Composition	Composition
P′ responds to (P_1, P_2)	$\Box(P_1 \wedge \bigcirc\Diamond P_2 \Rightarrow \bigcirc(\Diamond(P_2 \wedge \Diamond P')))$	Composition	Composition	Composition	Composition

Approach :

1. Collect the 50 LTL formulas given by Dwyer et al.
2. Generate the Patterns automata (LTL2BA)
3. Generate the Scopes automata (LTL2BA)
4. Apply the composition to generate the automata of remaining combinations

Homogeneity

	globally	before R	after Q	between Q and R	after Q unless R
always P	$\Box P$	$\Diamond R \Rightarrow (P \text{ U } R)$	$\Box(Q \Rightarrow \Box P)$	$\Box((Q \wedge \neg R \wedge \Diamond R) \Rightarrow (P \text{ U } R))$	$\Box((Q \wedge \neg R) \Rightarrow (P \text{ W } R))$
never P	$\Box\neg P$	$\equiv$	$\equiv$	$\equiv$	$\equiv$
eventually P	$\Diamond P$	$\subset$ (1)	$\equiv$	$\subset$ (2)	$\subset$ (2)
eventually P at most 2	$(\neg P \text{ W } (P \text{ W } (\neg P \text{ W } (P \text{ W } \Box\neg P))))$	$\equiv$	$\equiv$	$\equiv$	$\equiv$
P precedes P′	$\neg P' \text{ W } P$	$\equiv$	$\supset$ (3)	$\subset$ (2)	$\subset$ (2)
(P_1, P_2) precedes P′	$\Diamond P' \Rightarrow (\neg P' \text{ U } (P_1 \wedge \neg P' \wedge \bigcirc(\neg P' \text{ U } P_2)))$	$\equiv$	$\equiv$	$\subset$ (2)	$\subset$ (2)
P precedes (P'_1, P'_2)	$(\Diamond(P'_1 \wedge \bigcirc\Diamond P'_2)) \Rightarrow ((\neg P'_1) \text{ U } P))$	$\equiv$	$\equiv$	$\subset$ (2)	$\subset$ (2)
P′ responds to P	$\Box(P \Rightarrow \Diamond P')$	$\equiv$	$\equiv$	$\equiv$	$\equiv$
(P'_1, P'_2) responds to P	$\Box(P \Rightarrow \Diamond(P'_1 \wedge \bigcirc\Diamond P'_2))$	$\supset$ (4)	$\equiv$	$\supset$ (4)	$\supset$ (4)
P′ responds to (P_1, P_2)	$\Box(P_1 \wedge \bigcirc\Diamond P_2 \Rightarrow \bigcirc(\Diamond(P_2 \wedge \Diamond P')))$	$\neq$ (5)	$\equiv$	$\neq$ (5)	$\neq$ (5)

Approach :

1. Collect the 50 LTL formulas given by Dwyer et al.
2. Generate the Patterns automata (LTL2BA)
3. Generate the Scopes automata (LTL2BA)
4. Apply the composition to generate the automata of remaining combinations
5. Compare ! (Translation $\equiv, \subset, \supset$ Composition)

Homogeneity : mismatching cases

Mismatching cases	**Dwyer et al.'s Formula**		**Composition Formula**
(1) eventually P before R	$\neg R \ \mathsf{W} \ (P \wedge \neg R)$	$\subset$	
(2) P precedes P′ / P eventually between Q and R / after Q unless R	$\Box((Q \wedge \neg R) \Rightarrow \dots)$	$\subset$	
(3) P precedes P′ after Q	$\Box\neg Q \vee \Diamond(Q \wedge (\neg P' \ \mathsf{W} \ P))$	$\supset$	
(4) $(\mathrm{P}'_1, \mathrm{P}'_2)$ responds to P before R/ . . .	$\dots(P \Rightarrow (\neg R \ \mathsf{U} \ (P'_1 \wedge \neg R \wedge \bigcirc(\neg R \ \mathsf{U} \ P'_2)))) \dots$	$\supset$	
(5) P′ responds to $(\mathrm{P}_1, \mathrm{P}_2)$ before R/ . . .	$\dots(P_1 \wedge \bigcirc(\neg R \ \mathsf{U} \ P_2) \Rightarrow \bigcirc(\neg R \ \mathsf{U} \ (P_2 \wedge \Diamond P'))) \dots$	$\neq$	

Homogeneity : mismatching cases

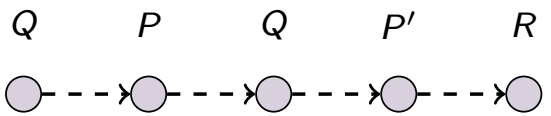
Mismatching cases	**Dwyer et al.'s Formula**		**Composition Formula**
(1) eventually P before R	$\neg R \ \mathsf{W}\ (P \wedge \neg R)$	$\subset$	$\underline{R \vee} \ \neg R \ \mathsf{W}\ (P \wedge \neg R)$
(2) P precedes P′ / P eventually between Q and R / after Q unless R	$\Box((Q \wedge \neg R) \Rightarrow \dots)$	$\subset$	
(3) P precedes P′ after Q	$\Box\neg Q \vee \Diamond(Q \wedge (\neg P' \ \mathsf{W}\ P))$	$\supset$	
(4) (P'_1, P'_2) responds to P before R/ . . .	$\dots(P \Rightarrow (\neg R \ \mathsf{U}\ (P'_1 \wedge \neg R \wedge \bigcirc(\neg R \ \mathsf{U}\ P'_2)))) \dots$	$\supset$	
(5) P′ responds to (P_1, P_2) before R/ . . .	$\dots(P_1 \wedge \bigcirc(\neg R \ \mathsf{U}\ P_2) \Rightarrow \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge \Diamond P'))) \dots$	$\neq$	

1. if R holds immediately (right-open interval) !

Homogeneity : mismatching cases

Mismatching cases	**Dwyer et al.'s Formula**		**Composition Formula**
(1) eventually P before R	$\neg R \ \mathsf{W} \ (P \wedge \neg R)$	$\subset$	$R \vee \neg R \ \mathsf{W} \ (P \wedge \neg R)$
(2) P precedes P′ / P eventually between Q and R / after Q unless R	$\Box((Q \wedge \neg R) \Rightarrow \dots)$	$\subset$	$\Box((Q \wedge \ominus (\neg Q \ \mathsf{S} \ R) \wedge \neg R) \Rightarrow \dots)$
(3) P precedes P′ after Q	$\Box\neg Q \vee \Diamond(Q \wedge (\neg P' \ \mathsf{W} \ P))$	$\supset$	
(4) (P'_1, P'_2) responds to P before R/ ...	$\dots(P \Rightarrow (\neg R \ \mathsf{U} \ (P'_1 \wedge \neg R \wedge \bigcirc (\neg R \ \mathsf{U} \ P'_2)))) \dots$	$\supset$	
(5) P′ responds to (P_1, P_2) before R/ ...	$\dots(P_1 \wedge \bigcirc(\neg R \ \mathsf{U} \ P_2) \Rightarrow \bigcirc(\neg R \ \mathsf{U} \ (P_2 \wedge \Diamond P'))) \dots$	$\neq$	

1. if R holds immediately (right-open interval) !
2. Dwyer et al. consider all occurrences of Q (not the 1^{st}) : stronger constraint

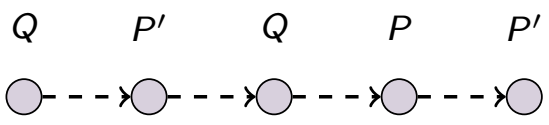


Translation ✗ Composition ✓

Homogeneity : mismatching cases

Mismatching cases	**Dwyer et al.'s Formula**		**Composition Formula**
(1) eventually P before R	$\neg R \ \mathsf{W}\ (P \wedge \neg R)$	$\subset$	$R \vee \neg R \ \mathsf{W}\ (P \wedge \neg R)$
(2) P precedes P′ / P eventually between Q and R / after Q unless R	$\Box((Q \wedge \neg R) \Rightarrow \dots)$	$\subset$	$\Box((Q \wedge \ominus(\neg Q \ \mathsf{S}\ R) \wedge \neg R) \Rightarrow \dots)$
(3) P precedes P′ after Q	$\Box\neg Q \vee \Diamond(Q \wedge (\neg P' \ \mathsf{W}\ P))$	$\supset$	$\Box\neg Q \vee \Diamond(Q \wedge \ominus\boxminus\neg Q \wedge (\neg P' \ \mathsf{W}\ P))$
(4) (P'_1, P'_2) responds to P before R/ ...	$\dots(P \Rightarrow (\neg R \ \mathsf{U}\ (P'_1 \wedge \neg R \wedge \bigcirc(\neg R \ \mathsf{U}\ P'_2)))) \dots$	$\supset$	
(5) P′ responds to (P_1, P_2) before R/ ...	$\dots(P_1 \wedge \bigcirc(\neg R \ \mathsf{U}\ P_2) \Rightarrow \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge \Diamond P'))) \dots$	$\neq$	

1. if R holds immediately (right-open interval) !
2. Dwyer et al. consider all occurrences of Q (not the 1^{st}) : stronger constraint
3. Dwyer et al. consider some occurrence of Q (not the 1^{st}) : weaker constraint



Translation $\surd$ Composition X

Homogeneity : mismatching cases

Mismatching cases	**Dwyer et al.'s Formula**		**Composition Formula**
(1) eventually P before R	$\neg R \ \mathsf{W}\ (P \wedge \neg R)$	$\subset$	$R \vee \neg R \ \mathsf{W}\ (P \wedge \neg R)$
(2) P precedes P′ / P eventually between Q and R / after Q unless R	$\Box((Q \wedge \neg R) \Rightarrow \dots)$	$\subset$	$\Box((Q \wedge \ominus(\neg Q \ \mathsf{S}\ R) \wedge \neg R) \Rightarrow \dots)$
(3) P precedes P′ after Q	$\Box\neg Q \vee \Diamond(Q \wedge (\neg P' \ \mathsf{W}\ P))$	$\supset$	$\Box\neg Q \vee \Diamond(Q \wedge \ominus\boxminus\neg Q \wedge (\neg P' \ \mathsf{W}\ P))$
(4) (P'_1, P'_2) responds to P before R/ . . .	$\dots(P \Rightarrow (\neg R \ \mathsf{U}\ (P'_1 \wedge \neg R \wedge \bigcirc(\neg R \ \mathsf{U}\ P'_2)))) \dots$	$\supset$	$\dots(P \Rightarrow (\neg R \ \mathsf{U}\ (P'_1 \wedge \neg R \wedge \bigcirc(\neg R \ \mathsf{U}\ (P'_2 \wedge \neg R))))) \dots$
(5) P′ responds to (P_1, P_2) before R/ . . .	$\dots(P_1 \wedge \bigcirc(\neg R \ \mathsf{U}\ P_2) \Rightarrow \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge \Diamond P'))) \dots$	$\neq$	$\dots(P_1 \wedge \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge \neg R)) \Rightarrow \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge (\neg R \ \mathsf{U}\ (P' \wedge \neg R))))) \dots$

1. if R holds immediately (right-open interval) !
2. Dwyer et al. consider all occurrences of Q (not the 1^{st}) : stronger constraint
3. Dwyer et al. consider some occurrence of Q (not the 1^{st}) : weaker constraint
4. right-open interval !
5. right-open interval !

Homogeneity : mismatching cases

Mismatching cases	**Dwyer et al.'s Formula**		**Composition Formula**
(1) eventually P before R	$\neg R \ \mathsf{W}\ (P \wedge \neg R)$	$\subset$	$R \vee \neg R \ \mathsf{W}\ (P \wedge \neg R)$
(2) P precedes P′ / P eventually between Q and R / after Q unless R	$\Box((Q \wedge \neg R) \Rightarrow \dots)$	$\subset$	$\Box((Q \wedge \ominus (\neg Q \ \mathsf{S}\ R) \wedge \neg R) \Rightarrow \dots)$
(3) P precedes P′ after Q	$\Box\neg Q \vee \Diamond(Q \wedge (\neg P' \ \mathsf{W}\ P))$	$\supset$	$\Box\neg Q \vee \Diamond(Q \wedge \ominus \boxminus \neg Q \wedge (\neg P' \ \mathsf{W}\ P))$
(4) (P'_1, P'_2) responds to P before R/ . . .	$\dots(P \Rightarrow (\neg R \ \mathsf{U}\ (P'_1 \wedge \neg R \wedge \bigcirc(\neg R \ \mathsf{U}\ P'_2)))) \dots$	$\supset$	$\dots(P \Rightarrow (\neg R \ \mathsf{U}\ (P'_1 \wedge \neg R \wedge \bigcirc(\neg R \ \mathsf{U}\ (P'_2 \wedge \neg R))))) \dots$
(5) P′ responds to (P_1, P_2) before R/ . . .	$\dots(P_1 \wedge \bigcirc(\neg R \ \mathsf{U}\ P_2) \Rightarrow \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge \Diamond P'))) \dots$	$\neq$	$\dots(P_1 \wedge \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge \neg R)) \Rightarrow \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge (\neg R \ \mathsf{U}\ (P' \wedge \neg R))))) \dots$

Composition 3 : 0 Translation

- Compositional Semantics reveals inconsistencies within Translational Semantics

Homogeneity : mismatching cases

Mismatching cases	**Dwyer et al.'s Formula**		**Composition Formula**
(1) eventually P before R	$\neg R \ \mathsf{W}\ (P \wedge \neg R)$	$\subset$	$R \vee \neg R \ \mathsf{W}\ (P \wedge \neg R)$
(2) P precedes P′ / P eventually between Q and R / after Q unless R	$\Box((Q \wedge \neg R) \Rightarrow \dots)$	$\subset$	$\Box((Q \wedge \ominus(\neg Q \ \mathsf{S}\ R) \wedge \neg R) \Rightarrow \dots)$
(3) P precedes P′ after Q	$\Box\neg Q \vee \Diamond(Q \wedge (\neg P' \ \mathsf{W}\ P))$	$\supset$	$\Box\neg Q \vee \Diamond(Q \wedge \ominus\boxminus\neg Q \wedge (\neg P' \ \mathsf{W}\ P))$
(4) (P'_1, P'_2) responds to P before R/ . . .	$\dots(P \Rightarrow (\neg R \ \mathsf{U}\ (P'_1 \wedge \neg R \wedge \bigcirc(\neg R \ \mathsf{U}\ P'_2)))) \dots$	$\supset$	$\dots(P \Rightarrow (\neg R \ \mathsf{U}\ (P'_1 \wedge \neg R \wedge \bigcirc(\neg R \ \mathsf{U}\ (P'_2 \wedge \neg R))))) \dots$
(5) P′ responds to (P_1, P_2) before R/ . . .	$\dots(P_1 \wedge \bigcirc(\neg R \ \mathsf{U}\ P_2) \Rightarrow \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge \Diamond P'))) \dots$	$\neq$	$\dots(P_1 \wedge \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge \neg R)) \Rightarrow \bigcirc(\neg R \ \mathsf{U}\ (P_2 \wedge (\neg R \ \mathsf{U}\ (P' \wedge \neg R))))) \dots$

Composition 3 : 0 Translation

- Compositional Semantics reveals inconsistencies within Translational Semantics

Composition 4 : 0 Translation

- Composition is linear where $LTL \rightarrow BA$ is PSPACE-complete !

Furthermore

- Dwyer et al. combine 1 pattern and 1 scope

Furthermore

- Dwyer et al. combine 1 pattern and 1 scope

Extension

- 1 pattern and successive applications of k scopes !

Furthermore

- Dwyer et al. combine 1 pattern and 1 scope

Extension

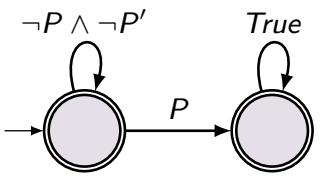
- 1 pattern and successive applications of k scopes !
- thanks to composition ! $Pattern \times Scope^{k \in \mathbb{N}} \xrightarrow{semantics} BA$

Furthermore

- Dwyer et al. combine 1 pattern and 1 scope

Extension

- 1 pattern and successive applications of k scopes !
- thanks to composition ! $Pattern \times Scope^{k \in \mathbb{N}} \xrightarrow{semantics} BA$
- ex : P precedes P'

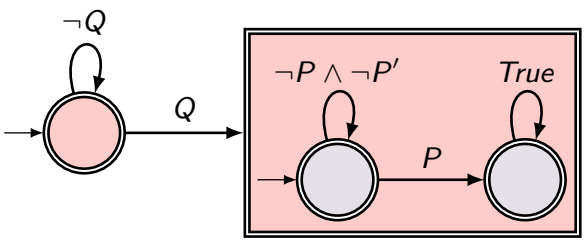


Furthermore

- Dwyer et al. combine 1 pattern and 1 scope

Extension

- 1 pattern and successive applications of k scopes !
- thanks to composition ! $Pattern \times Scope^{k \in \mathbb{N}} \xrightarrow{semantics} BA$
- ex : P precedes P' after Q

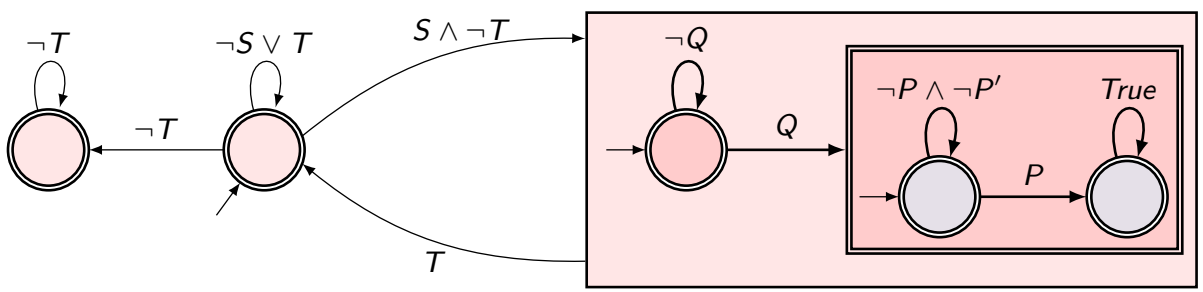


Furthermore

- Dwyer et al. combine 1 pattern and 1 scope

Extension

- 1 pattern and successive applications of k scopes !
- thanks to composition ! $Pattern \times Scope^{k \in \mathbb{N}} \xrightarrow{semantics} BA$
- ex : P precedes P' after Q, between S and T

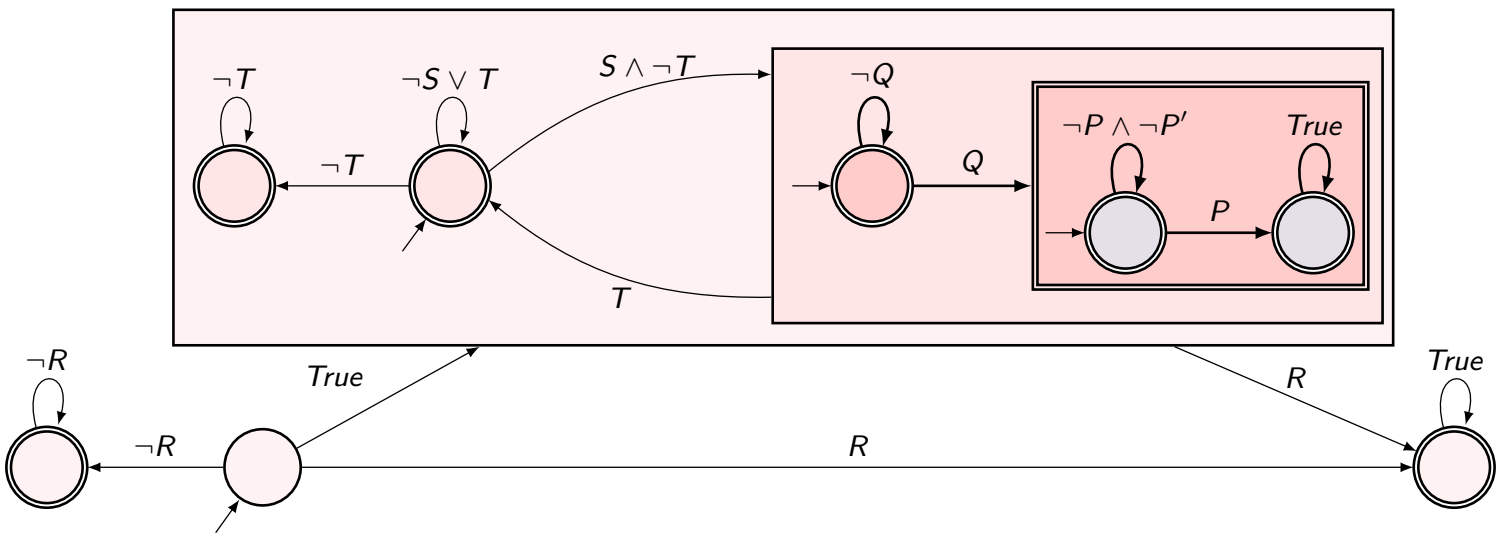


Furthermore

- Dwyer et al. combine 1 pattern and 1 scope

Extension

- 1 pattern and successive applications of k scopes !
- thanks to composition ! $Pattern \times Scope^{k \in \mathbb{N}} \xrightarrow{semantics} BA$
- ex : P precedes P' after Q, between S and T, before R ...



Plan

Test Intention

- Given a test purpose automaton, coverage criteria may be :
  - all states
  - all transitions
  - all k-paths
  - ...

Test Intention

- Given a test purpose automaton, coverage criteria may be :
  - all states
  - all transitions
  - all k-paths
  - ...
- How do we cover predicates over transitions ?
  - C/DC (Condition and Decision Coverage)
  - MCC (Multiple Condition Coverage)

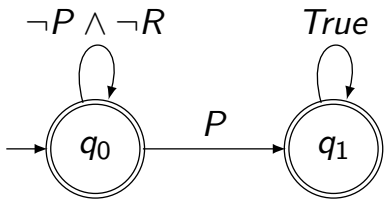
Test Intention

- Given a test purpose automaton, coverage criteria may be :
  - all states
  - all transitions
  - all k-paths
  - ...
- How do we cover predicates over transitions ?
  - C/DC (Condition and Decision Coverage)
  - MCC (Multiple Condition Coverage)

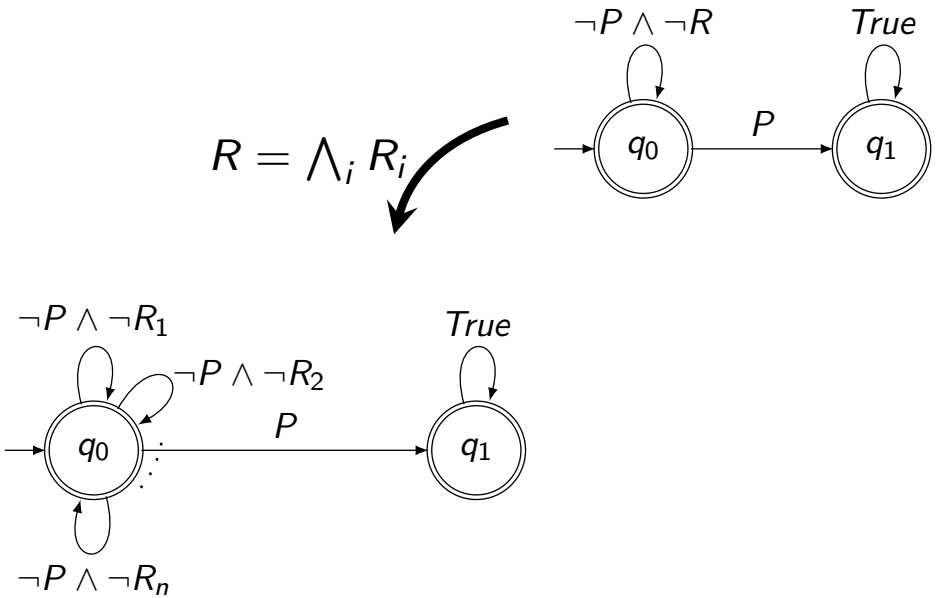
Transformation rules

There is a common practice to transform a test purpose automaton to an **equivalent** test intention automaton

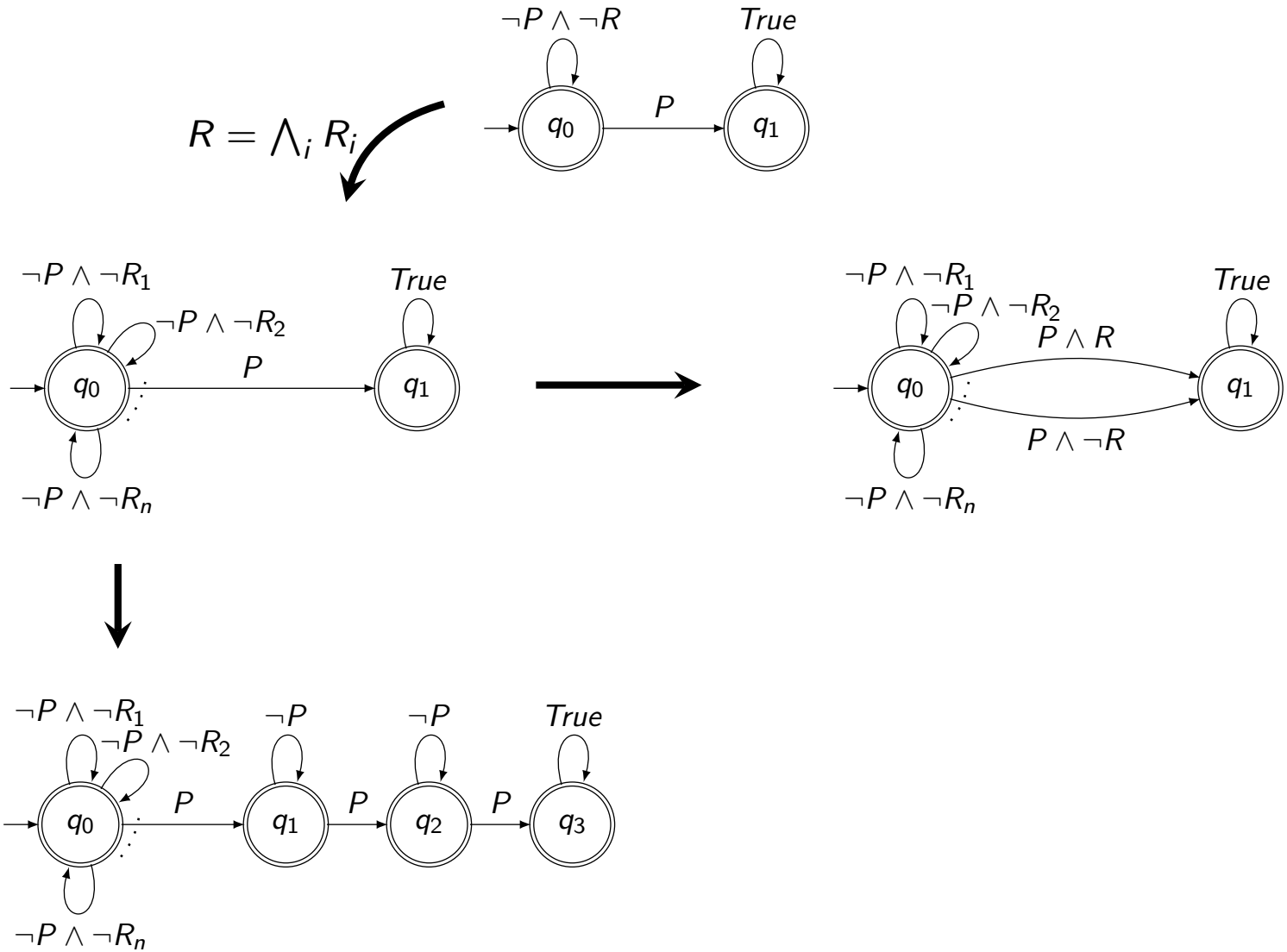
Example : P precedes R



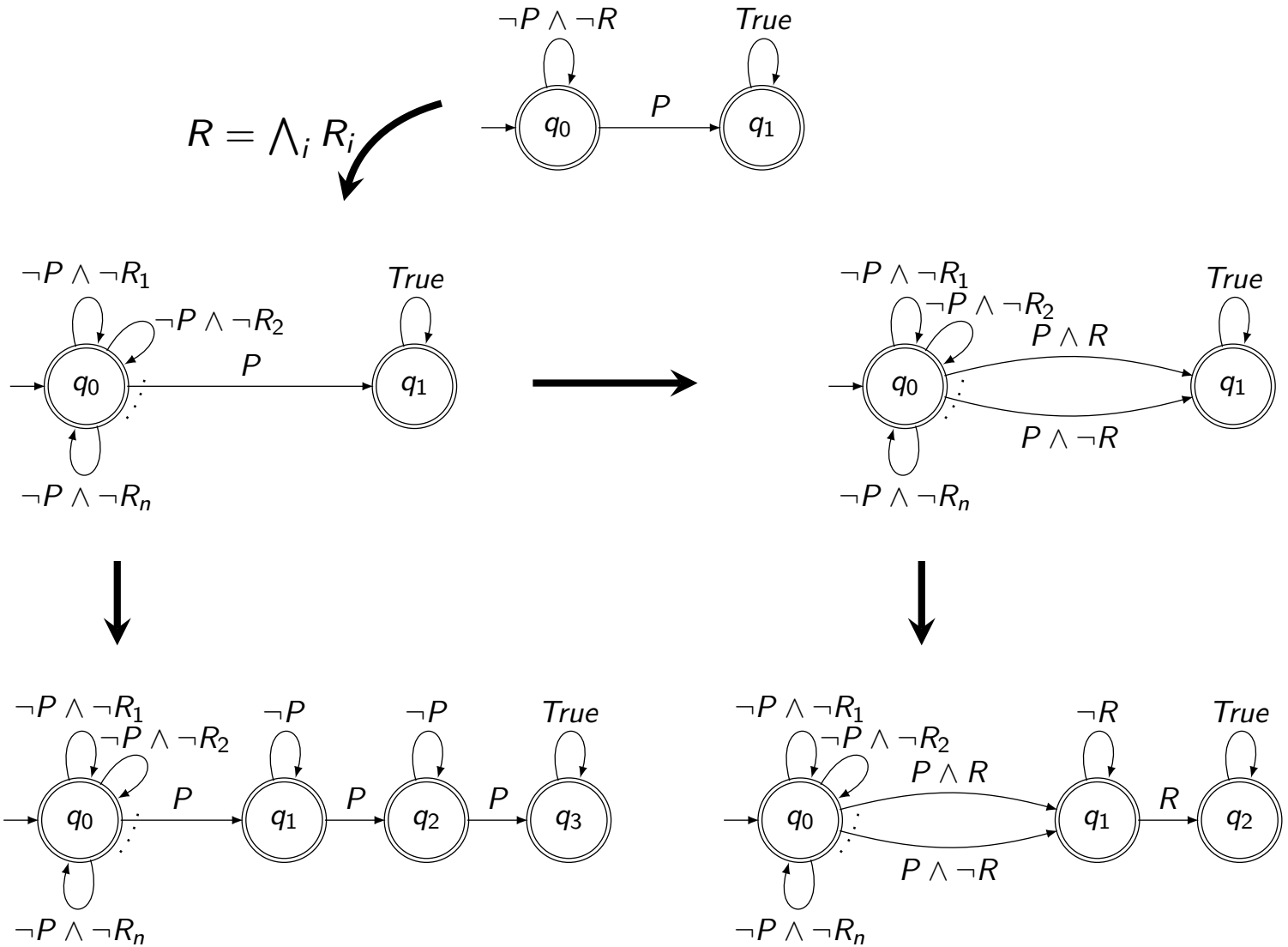
Example : P precedes R



Example : P precedes R



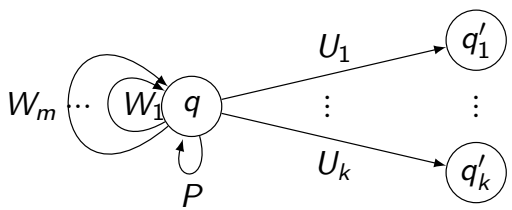
Example : P precedes R



Transformation Rule

Transformation Definition

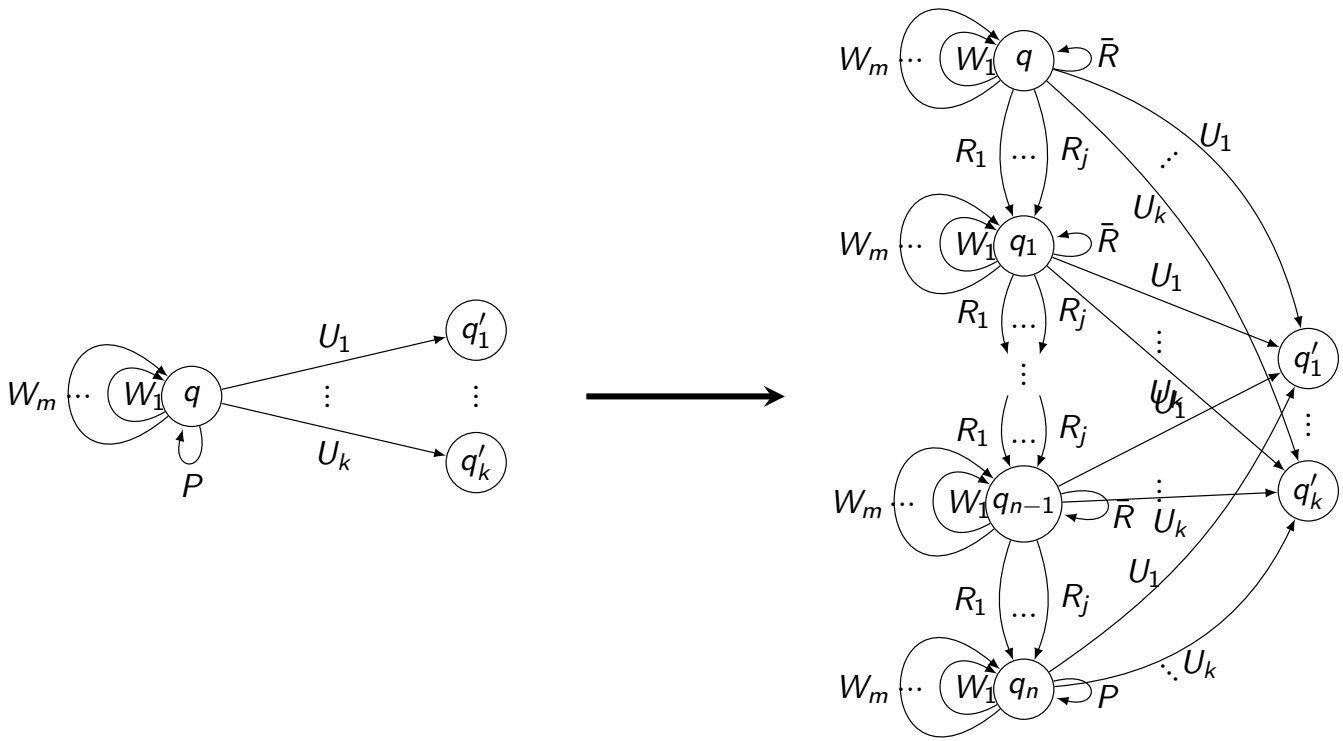
- test a reflexive transition $\xrightarrow{P}$
- predicate coverage : test all predicate cases $P = (\overbrace{(\bigvee_i R_i)}^{\mathcal{R}} \vee \bar{R})$
- paths coverage : unfold $\mathcal{R}$ between 0 and n times



Transformation Rule

Transformation Definition

- test a reflexive transition $\xrightarrow{P}$
- predicate coverage : test all predicate cases $P = (\overbrace{(\bigvee_i R_i)}^{\mathcal{R}} \vee \bar{R})$
- paths coverage : unfold $\mathcal{R}$ between 0 and n times



Transformation Realization

1. Define/implement a transformation rule over automata

Transformation Realization

1. Define/implement a transformation rule over automata
2. After transformation, thanks to composition, we can apply new scopes !

Transformation Realization

1. Define/implement a transformation rule over automata
2. After transformation, thanks to composition, we can apply new scopes !

Theorem : Composition Consistency

Let $\mathsf{p}_i \stackrel{\text{def}}{=} (Q_{\mathsf{p}_i}, init_{\mathsf{p}_i}, F_{\mathsf{p}_i}, \mathcal{P}, T_{\mathsf{p}_i}),\ i \in \{1,2\}$ be two automata and $\mathsf{s} \stackrel{\text{def}}{=} (Q_{\mathsf{s}}, init_{\mathsf{s}}, F_{\mathsf{s}}, \mathcal{P}, T_{\mathsf{s}})$ be any scope automaton. Then, we have :

$$\left.\begin{array}{rcl} \mathcal{L}(\mathsf{p}_1) & = & \mathcal{L}(\mathsf{p}_2) \\ \mathcal{L}^{\omega}(\mathsf{p}_1) & = & \mathcal{L}^{\omega}(\mathsf{p}_2) \end{array}\right\} \iff \forall \mathsf{s}.\ \mathcal{L}^{\omega}(\mathsf{p}_1 \odot \mathsf{s}) = \mathcal{L}^{\omega}(\mathsf{p}_2 \odot \mathsf{s})$$

Transformation Realization

1. Define/implement a transformation rule over automata
2. After transformation, thanks to composition, we can apply new scopes !

Theorem : Composition Consistency

Let $p_i \stackrel{\text{def}}{=} (Q_{p_i}, init_{p_i}, F_{p_i}, \mathcal{P}, T_{p_i}),\ i \in \{1,2\}$ be two automata and $s \stackrel{\text{def}}{=} (Q_s, init_s, F_s, \mathcal{P}, T_s)$ be any scope automaton. Then, we have :

$$\left.\begin{array}{rcl} \mathcal{L}(p_1) & = & \mathcal{L}(p_2) \\ \mathcal{L}^\omega(p_1) & = & \mathcal{L}^\omega(p_2) \end{array}\right\} \iff \forall s.\ \mathcal{L}^\omega(p_1 \odot s) = \mathcal{L}^\omega(p_2 \odot s)$$

3. Prove transformation consistency !

Thank you

any questions ?

References

Dwyer, M., Avrunin, G. S., and Corbett, J. C. (1999).
Patterns in property specifications for finite-state verification.
In Proceedings of the 21st International Conference on Software Engineering, pages 411–420.

Taha, S., Julliand, J., Dadeau, F., Castillos, K. C., and Kanso, B. (2015).
A compositional automata-based semantics and preserving transformation rules for testing property patterns.
Formal Asp. Comput., 27(4) :641–664.