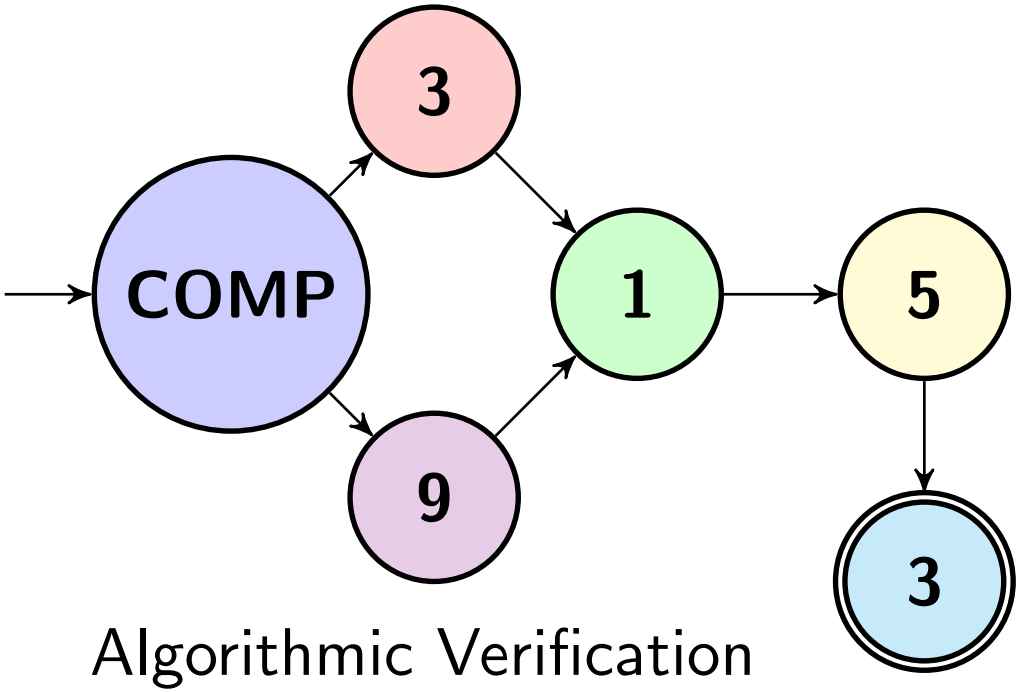


Algorithmic Verification

Temporal Logics

Dr. Liam O'Connor
CSE, UNSW (for now)
Term 1 2020

Reachability

Define $\mathrm{Reach}(A, q) \subseteq Q$ as the set of states reachable in A from q.

Define $\mathrm{Reach}(A) \equiv \mathrm{Reach}(A, q_0)$.

Reachability

Define $\mathrm{Reach}(A, q) \subseteq Q$ as the set of states reachable in A from q.

Define $\mathrm{Reach}(A) \equiv \mathrm{Reach}(A, q_0)$.

Exercise

Describe the algorithm for computing $\mathrm{Reach}(A)$.

Reachability

Define $\text{Reach}(A, q) \subseteq Q$ as the set of states reachable in A from q.

Define $\text{Reach}(A) \equiv \text{Reach}(A, q_0)$.

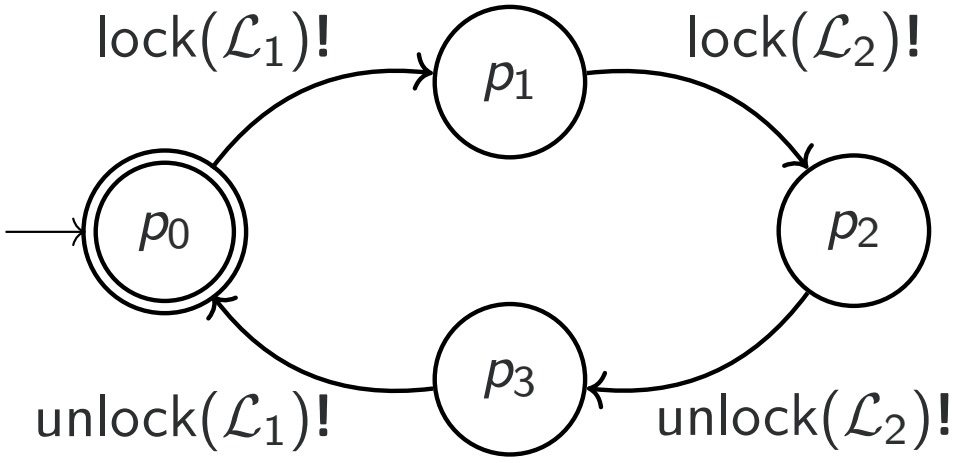
Exercise

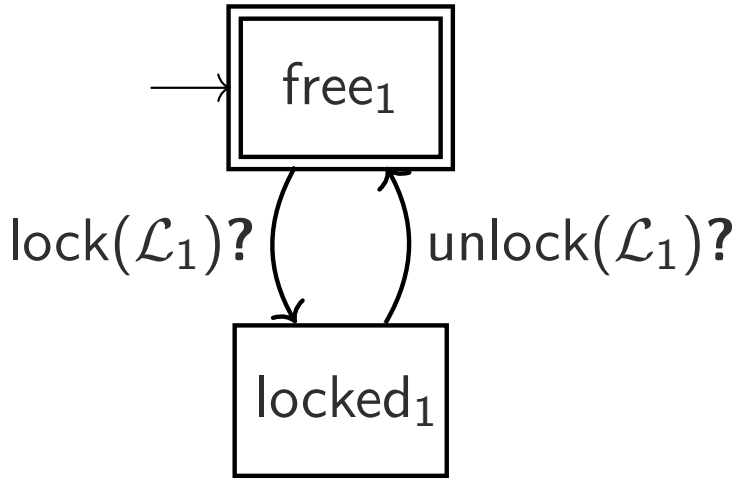
Describe the algorithm for computing $\text{Reach}(A)$.

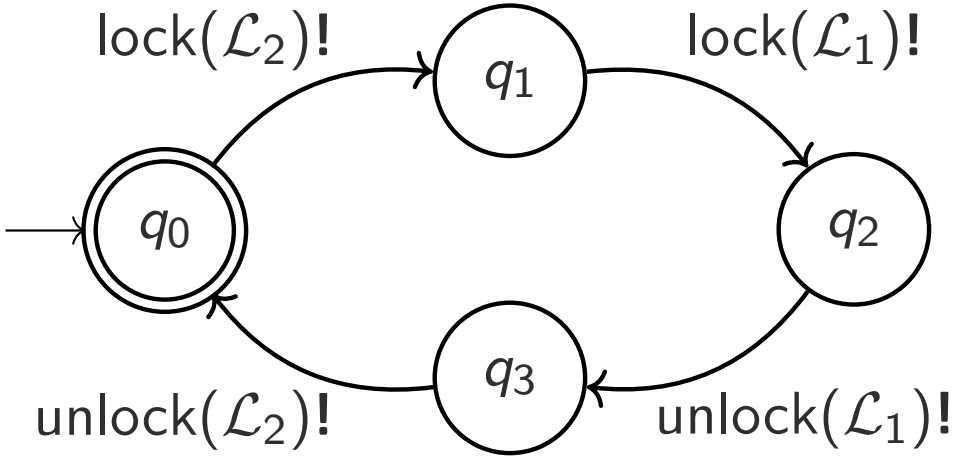
Deadlock or a stuck state is a state $q \in Q$ which has no outgoing transitions i.e $\forall a.\ \delta(q, a) = \emptyset$.

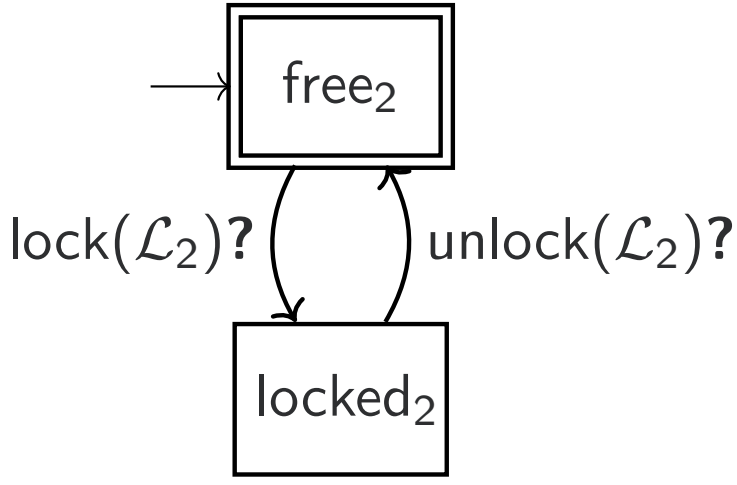
Deadlock Example

Assuming unicast synchronisation:





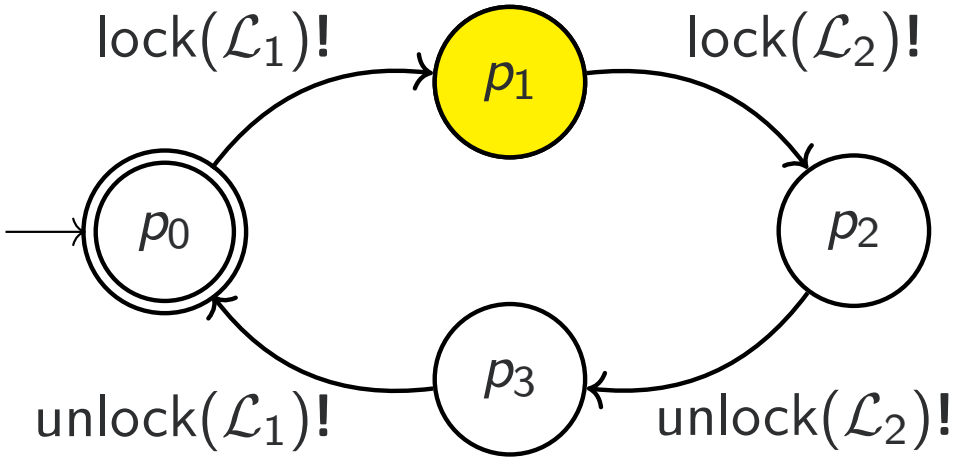


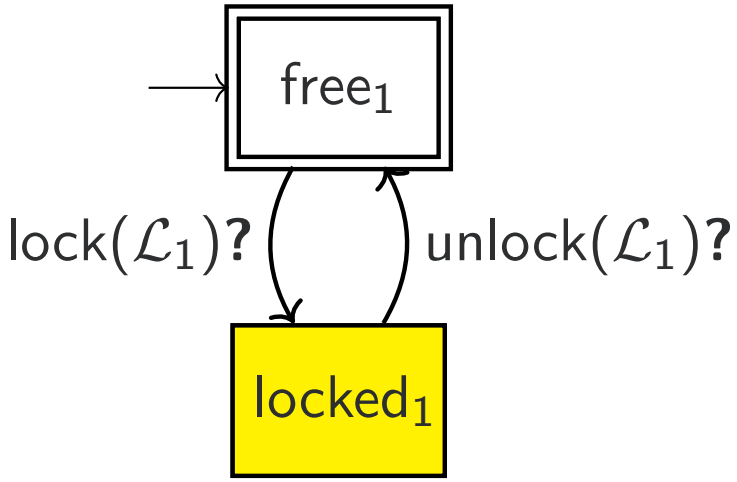


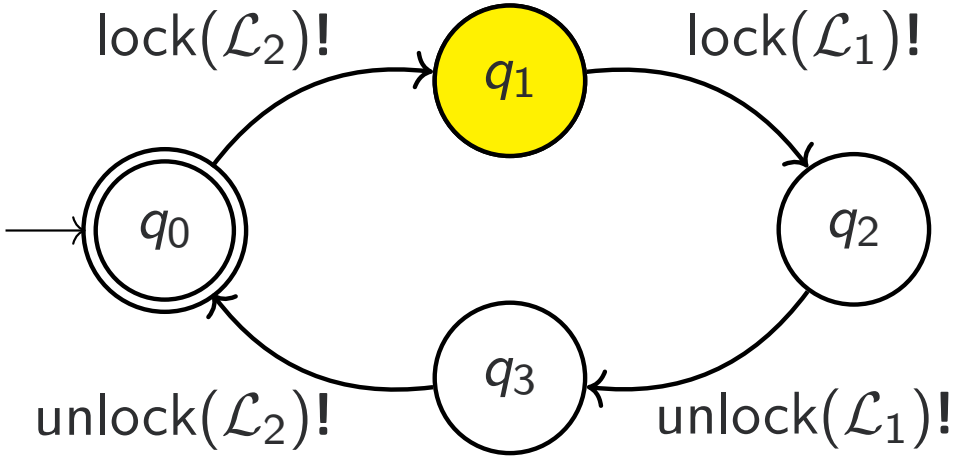
Exercise: What is an algorithm to detect deadlock?

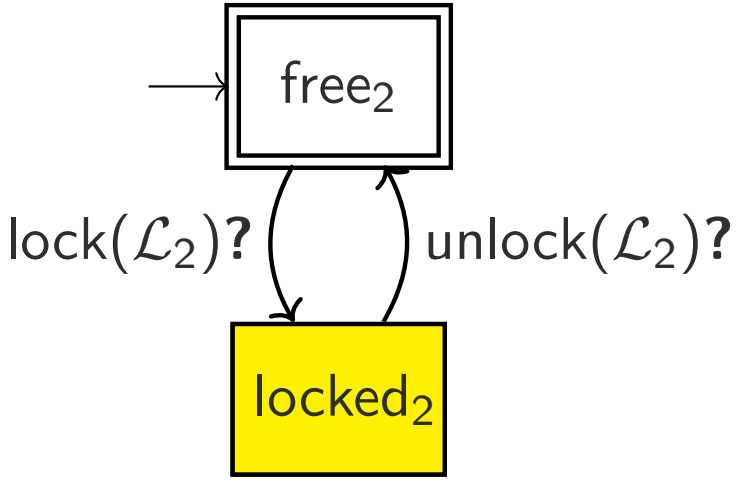
Deadlock Example

Assuming unicast synchronisation:



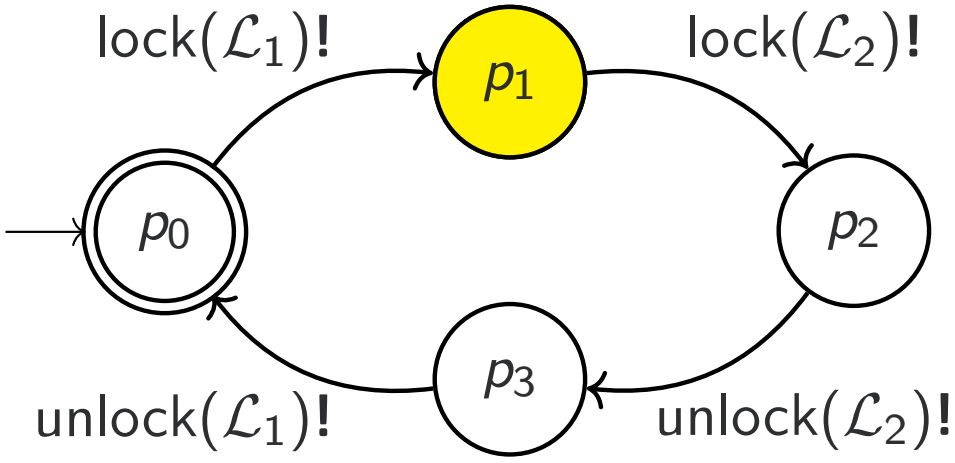


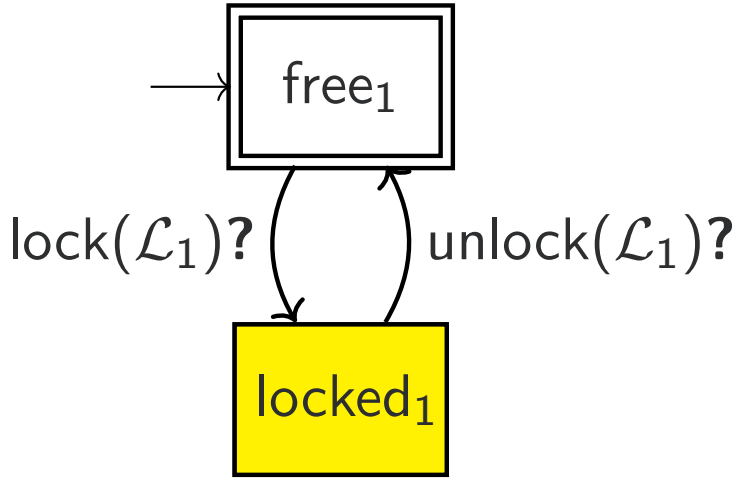


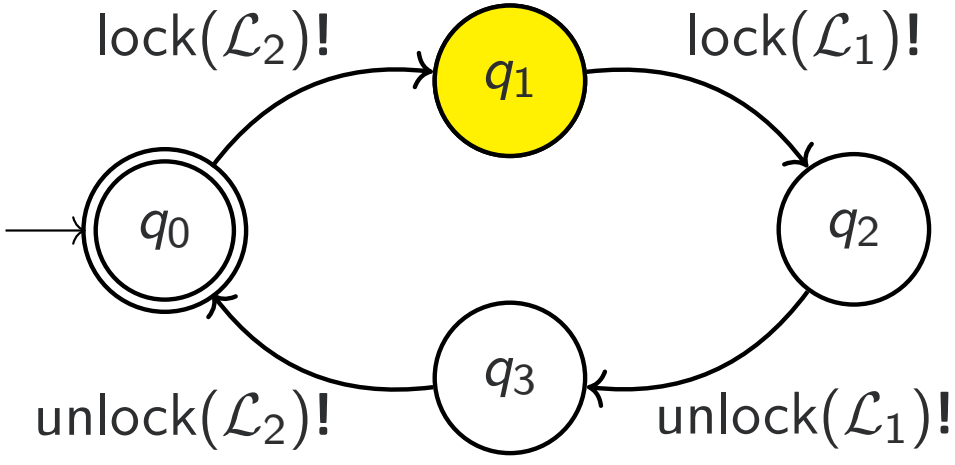


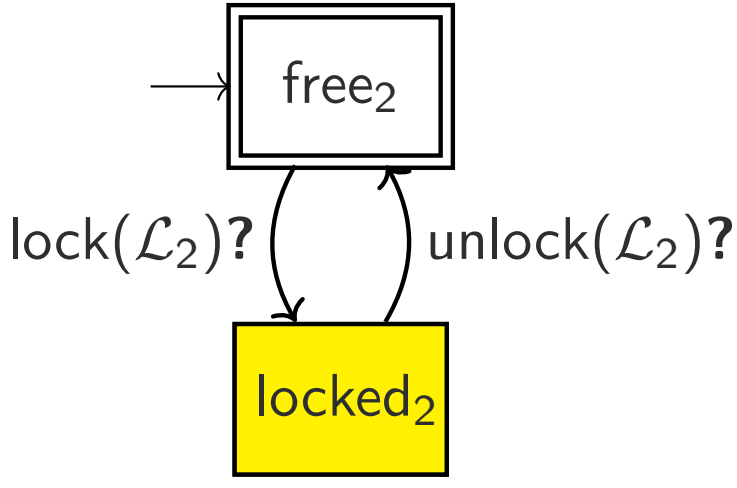
Deadlock Example

Assuming unicast synchronisation:









Exercise: What is an algorithm to detect deadlock?

Safety Properties

A safety property is an assertion that bad things do not happen. In other words, given some set of states $\mathsf{Bad} \subseteq Q$, we want to check that:

$$\mathsf{Bad} \cap \mathsf{Reach}(A) = \emptyset$$

Safety Properties

A safety property is an assertion that bad things do not happen. In other words, given some set of states $\mathrm{Bad} \subseteq Q$, we want to check that:

$$\mathrm{Bad} \cap \mathrm{Reach}(A) = \emptyset$$

Exercise

Give an algorithm to check a safety property.

Observations

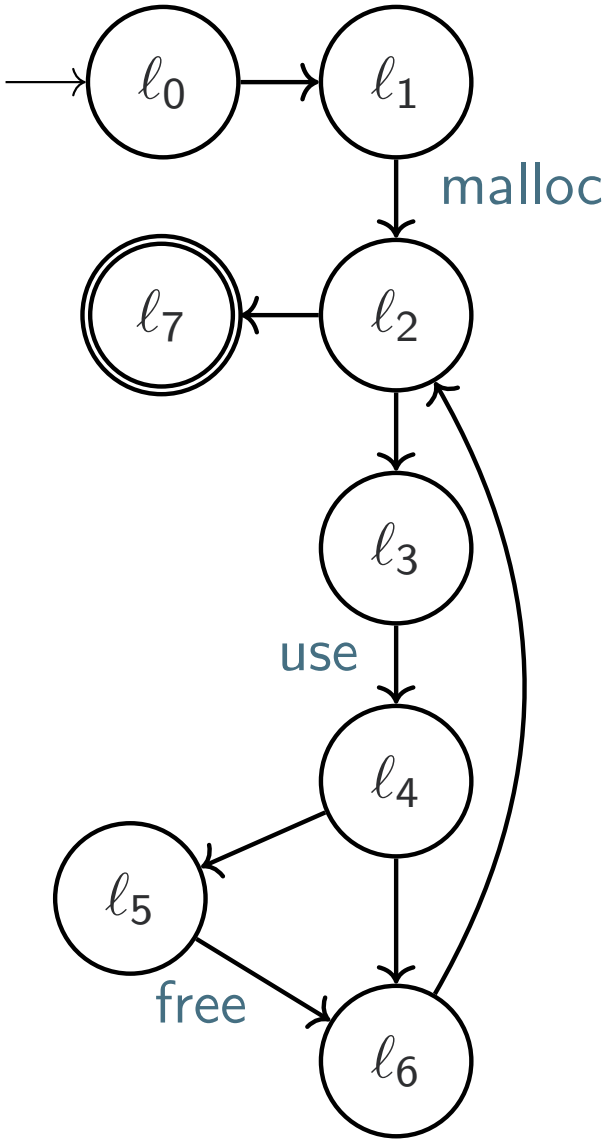
Is use after free a safety property?

```
void foo() {
   int x, a;
   int *p = malloc(sizeof(int));
   for (x = 10; x > 0; x--) {
      a = *p;
      if (x <= 1) {
         free(p);
      }
   }
}
```

Observations

Is use after free a safety property?

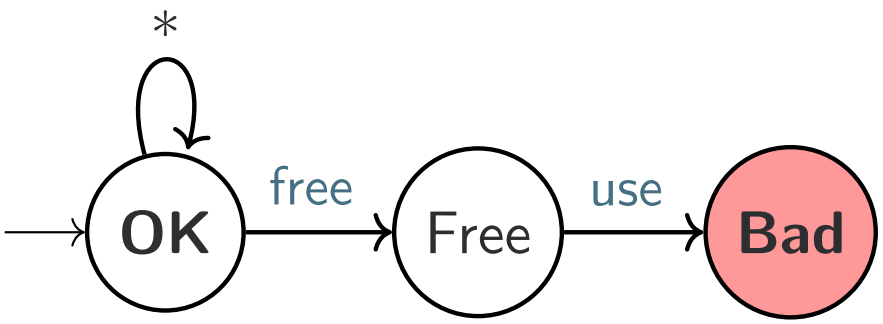
```
void foo() {
   int x, a;
   int *p = malloc(sizeof(int));
   for (x = 10; x > 0; x--) {
      a = *p;
      if (x <= 1) {
         free(p);
      }
   }
}
```

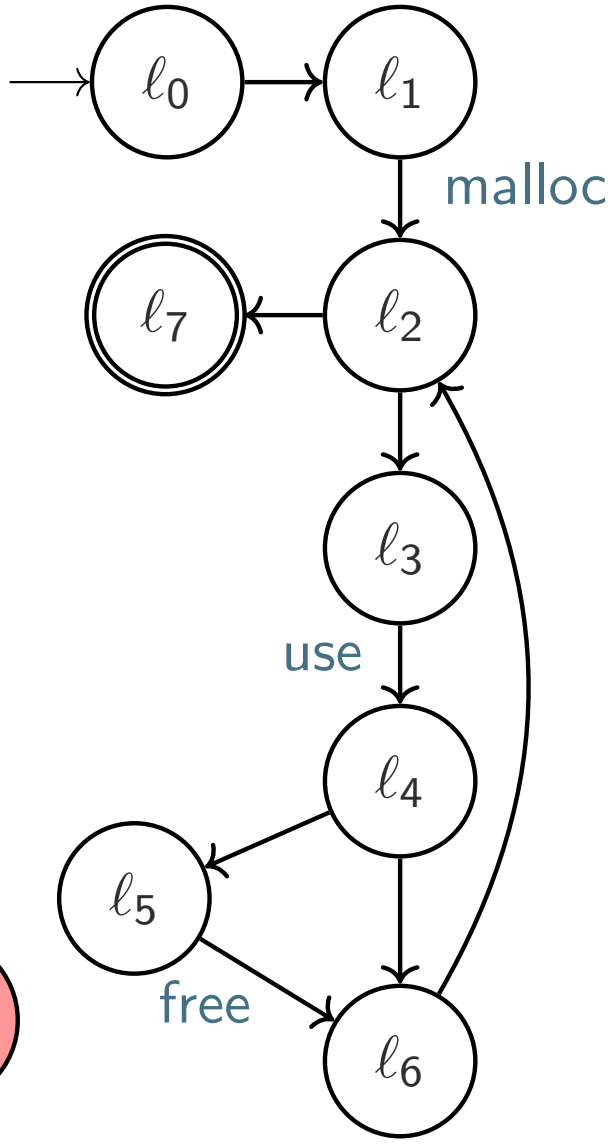


Observations

Is use after free a safety property?

```
void foo() {
   int x, a;
   int *p = malloc(sizeof(int));
   for (x = 10; x > 0; x--) {
      a = *p;
      if (x <= 1) {
         free(p);
      }
   }
}
```

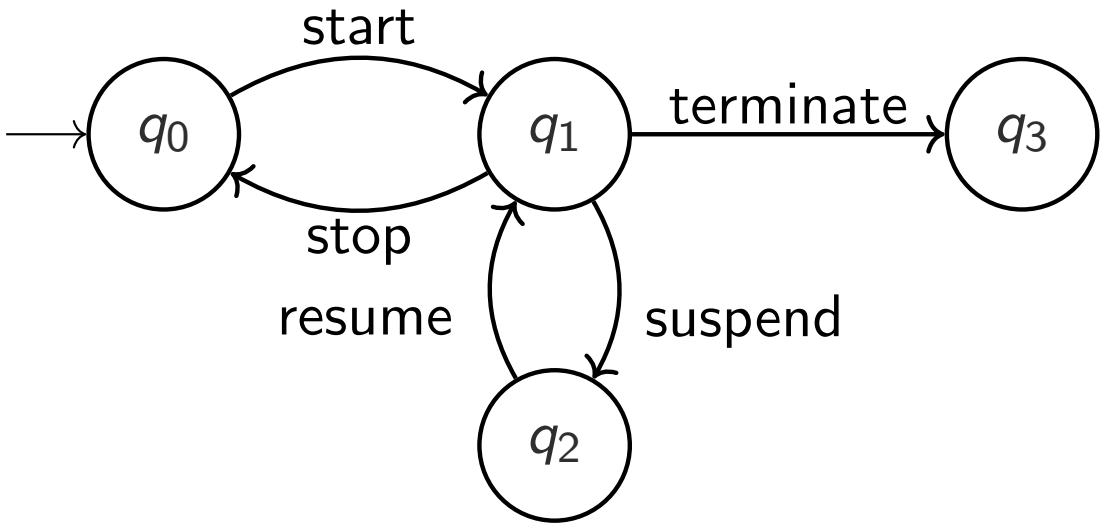




Kripke Structures

Definition

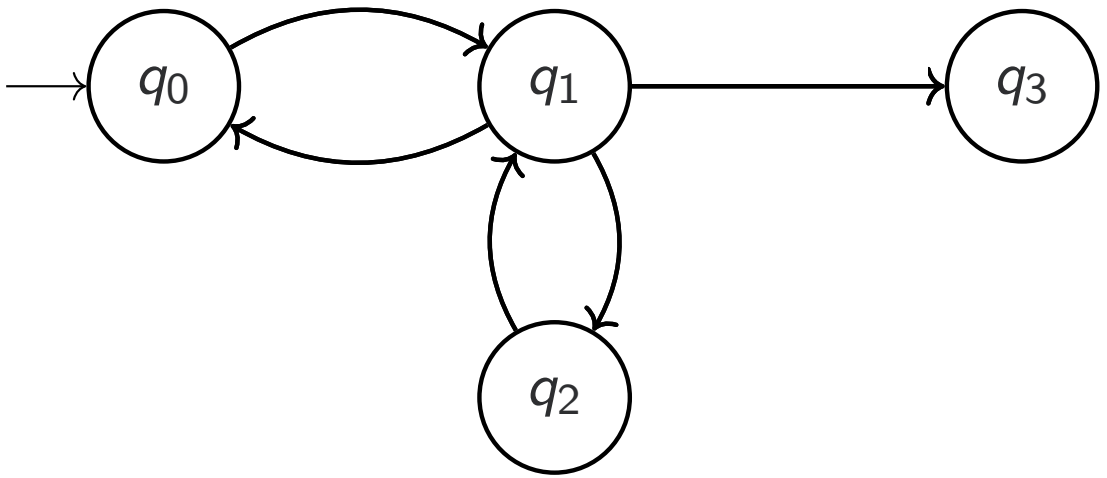
A *labelled automaton* is a FA $(Q, q_0, \Sigma, \delta, F, L)$ with an additional labelling function $L : Q \to 2^{\mathcal{P}}$, where $\mathcal{P}$ is our atomic propositions. A *Kripke structure* is a type of labelled automaton where $|\Sigma| = 1$, $F = Q$. Equivalently, we don't have a notion of actions or final states, and $\delta : Q \to 2^Q$. We also require that for any q, $\delta(q) \neq \emptyset$.



Kripke Structures

Definition

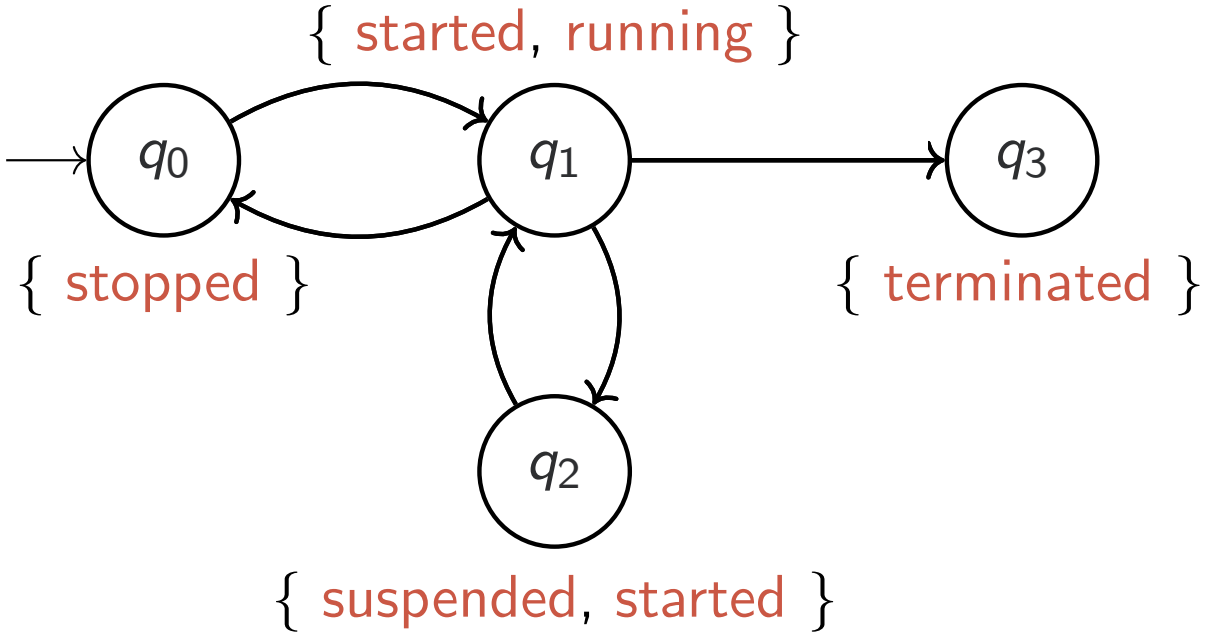
A *labelled automaton* is a FA $(Q, q_0, \Sigma, \delta, F, L)$ with an additional labelling function $L : Q \to 2^{\mathcal{P}}$, where $\mathcal{P}$ is our atomic propositions. A *Kripke structure* is a type of labelled automaton where $|\Sigma| = 1$, $F = Q$. Equivalently, we don't have a notion of actions or final states, and $\delta : Q \to 2^Q$. We also require that for any q, $\delta(q) \neq \emptyset$.



Kripke Structures

Definition

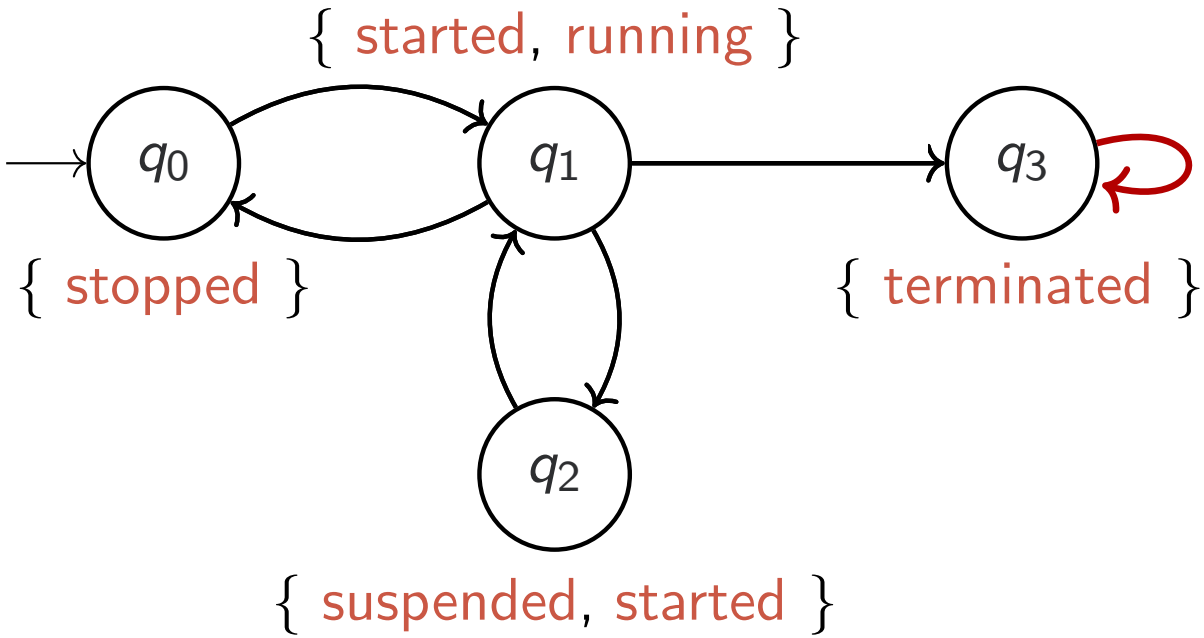
A *labelled automaton* is a FA $(Q, q_0, \Sigma, \delta, F, L)$ with an additional labelling function $L : Q \to 2^{\mathcal{P}}$, where $\mathcal{P}$ is our atomic propositions. A *Kripke structure* is a type of labelled automaton where $|\Sigma| = 1$, $F = Q$. Equivalently, we don't have a notion of actions or final states, and $\delta : Q \to 2^Q$. We also require that for any q, $\delta(q) \neq \emptyset$.



Kripke Structures

Definition

A *labelled automaton* is a FA $(Q, q_0, \Sigma, \delta, F, L)$ with an additional labelling function $L : Q \to 2^{\mathcal{P}}$, where $\mathcal{P}$ is our atomic propositions. A *Kripke structure* is a type of labelled automaton where $|\Sigma| = 1$, $F = Q$. Equivalently, we don't have a notion of actions or final states, and $\delta : Q \to 2^Q$. We also require that for any q, $\delta(q) \neq \emptyset$.



Traces

Definition

A *trace*, also called a *behaviour*, is the sequence of labels corresponding to a run. For Kripke structures it is necessarily infinite in length.

Define $\text{Traces}(A)$ to be all possible infinite traces from q_0 in A.

Definition

A linear time *property* is a set of traces, i.e. a subset of $(2^{\mathcal{P}})^{\omega}$. We say a Kripke structure A *satisfies* a property P iff:

$$\text{Traces}(A) \subseteq P$$

LTL

Linear temporal logic (LTL) is a *logic* designed to describe linear time properties.

Linear temporal logic syntax

We have normal propositional operators:

- $p \in \mathcal{P}$ is an LTL formula.
- If φ, ψ are LTL formulae, then $\varphi \wedge \psi$ is an LTL formula.
- If φ is an LTL formula, $\neg\varphi$ is an LTL formula.

LTL

Linear temporal logic (LTL) is a *logic* designed to describe linear time properties.

Linear temporal logic syntax

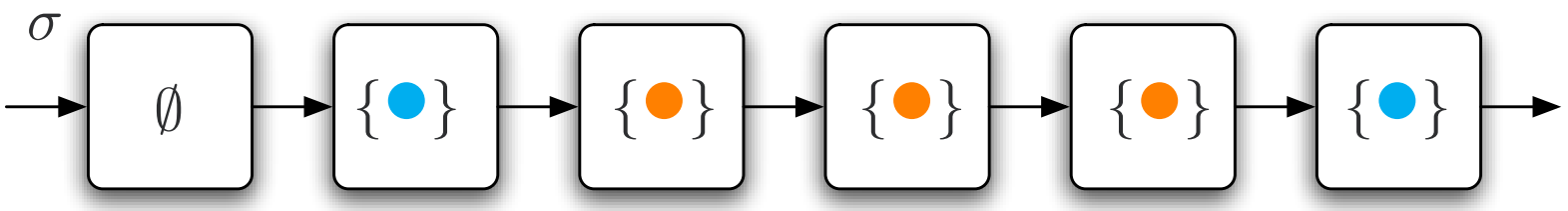
We have normal propositional operators:

- $p \in \mathcal{P}$ is an LTL formula.
- If φ, ψ are LTL formulae, then $\varphi \wedge \psi$ is an LTL formula.
- If φ is an LTL formula, $\neg\varphi$ is an LTL formula.

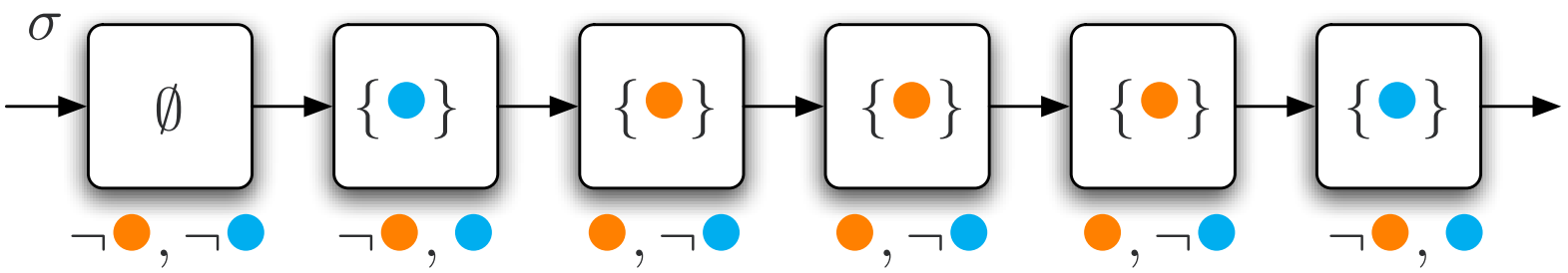
We also have modal or temporal operators:

- If φ is an LTL formula, then $\mathbf{X}\ \varphi$ is an LTL formula.
- If φ, ψ are LTL formulae, then φ **UNTIL** ψ is an LTL formula.

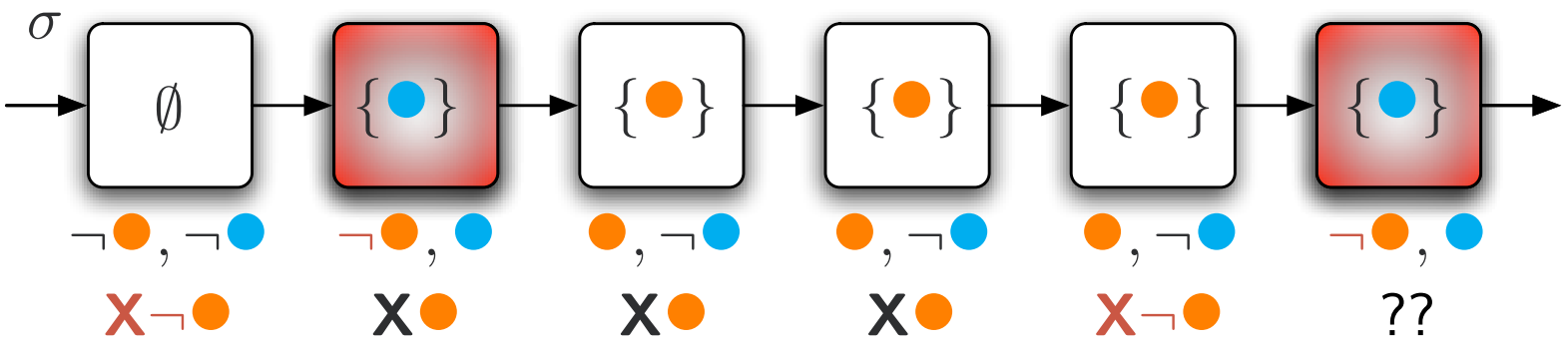
LTL Semantics in Pictures



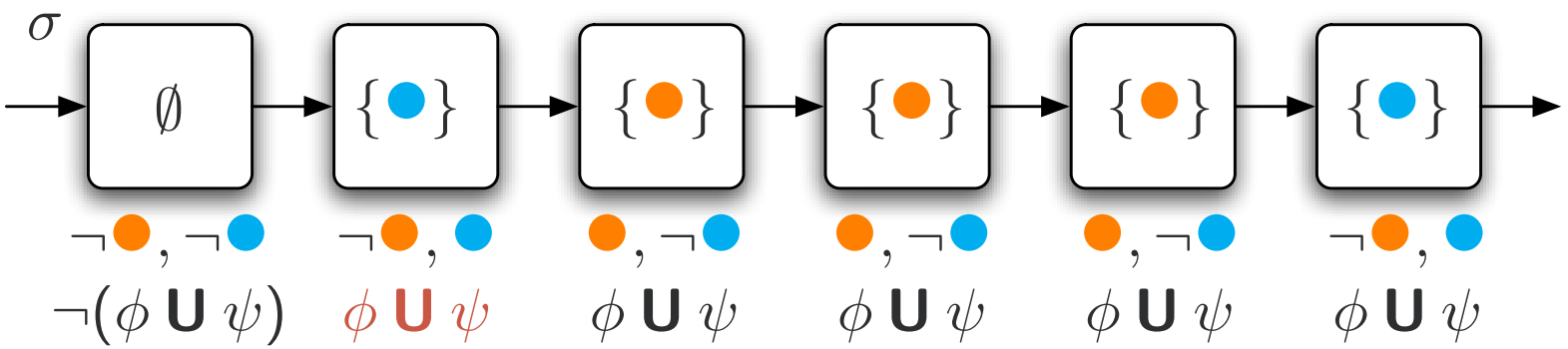
LTL Semantics in Pictures



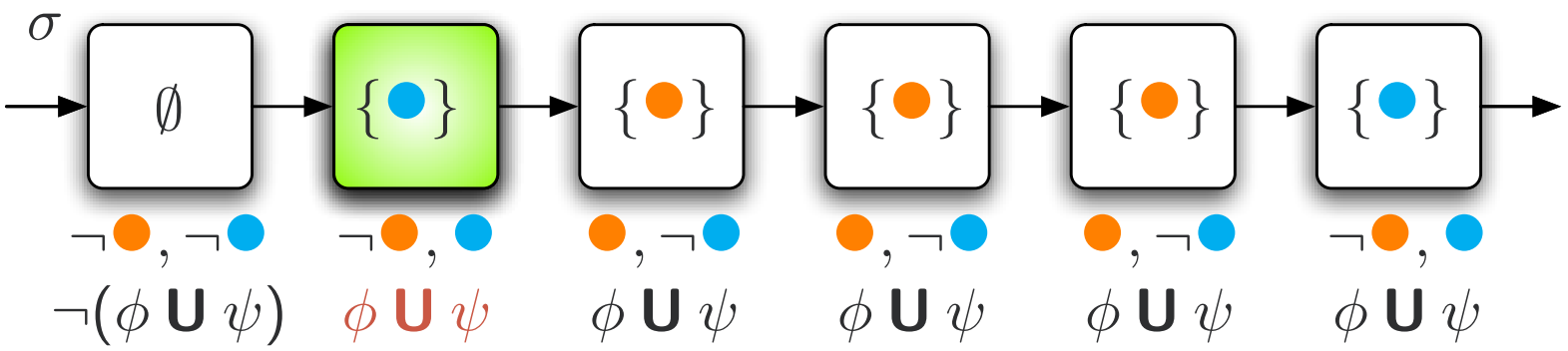
LTL Semantics in Pictures



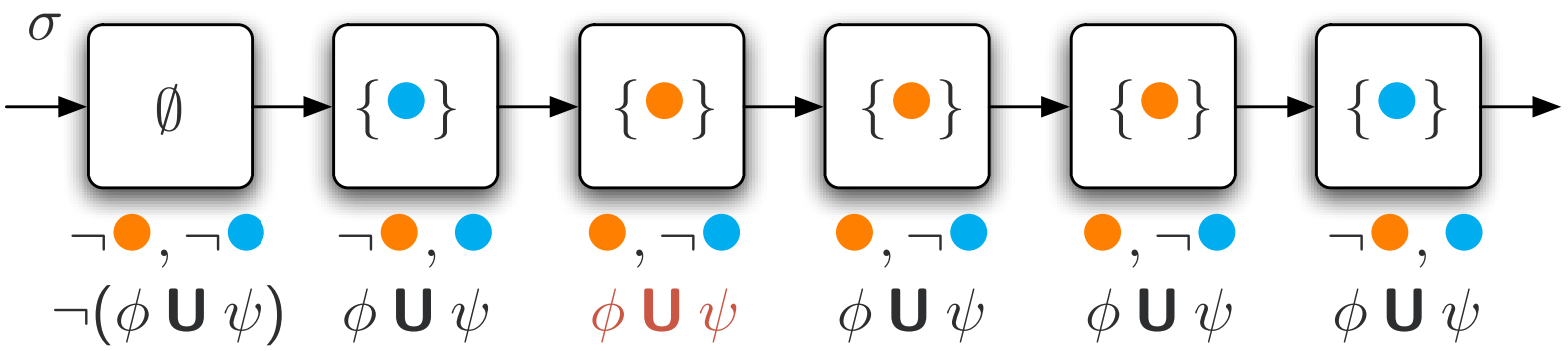
LTL Semantics in Pictures



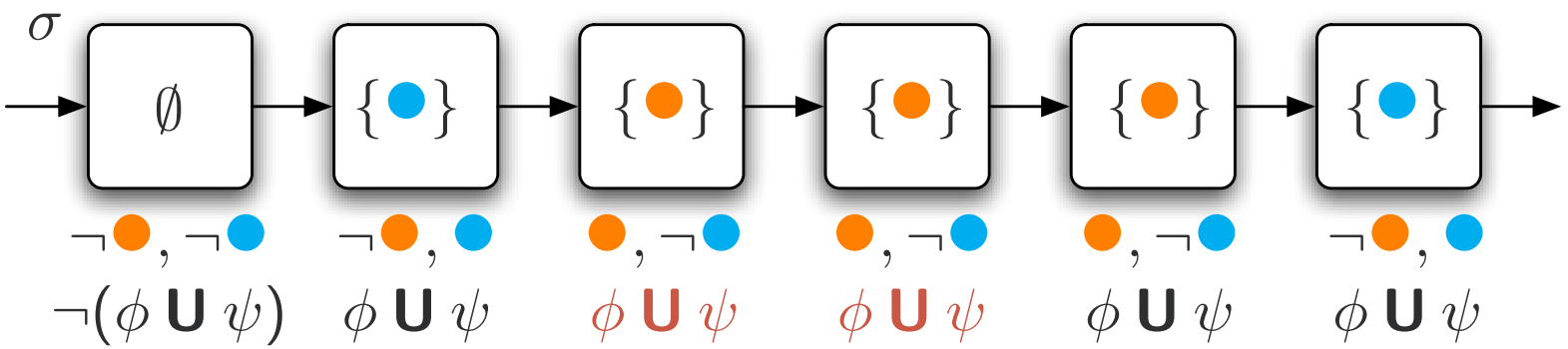
LTL Semantics in Pictures



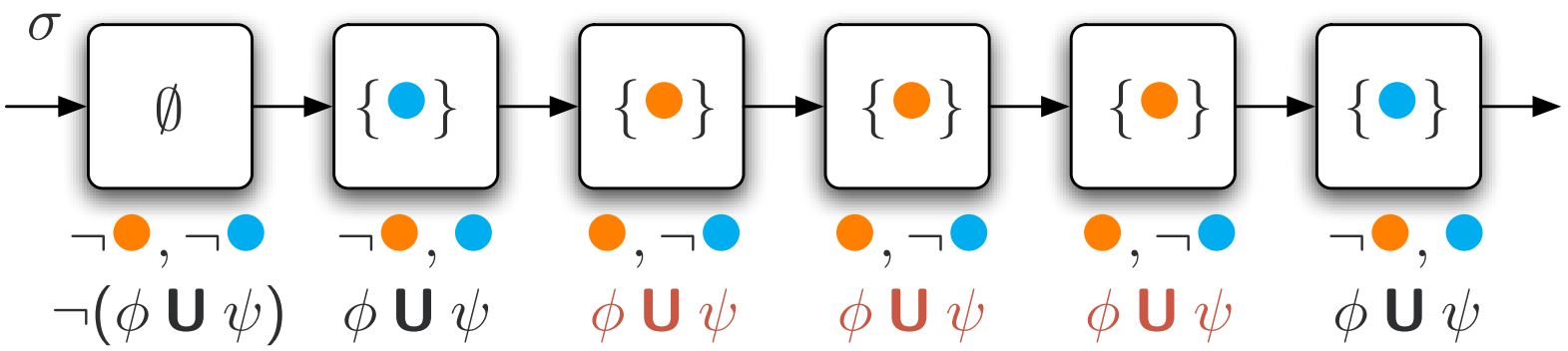
LTL Semantics in Pictures



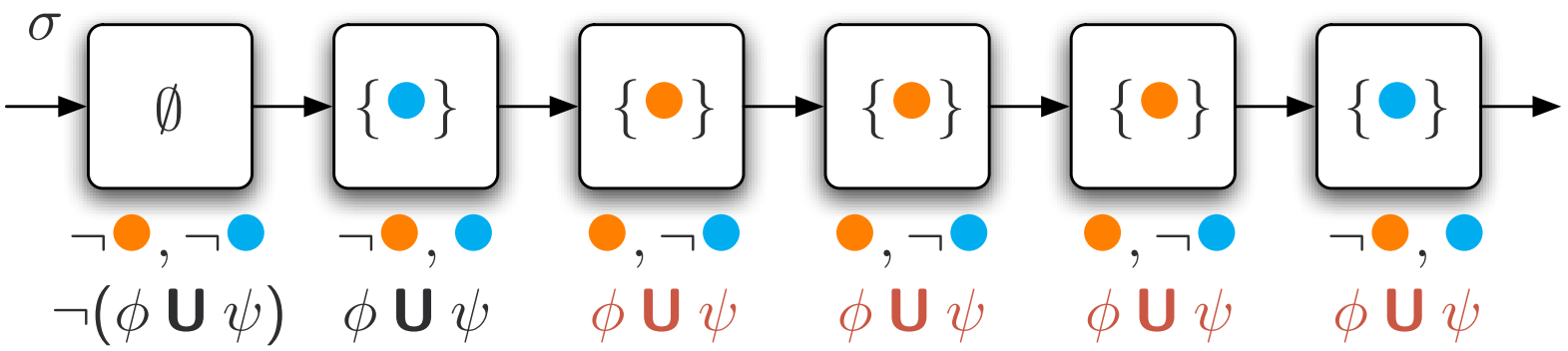
LTL Semantics in Pictures



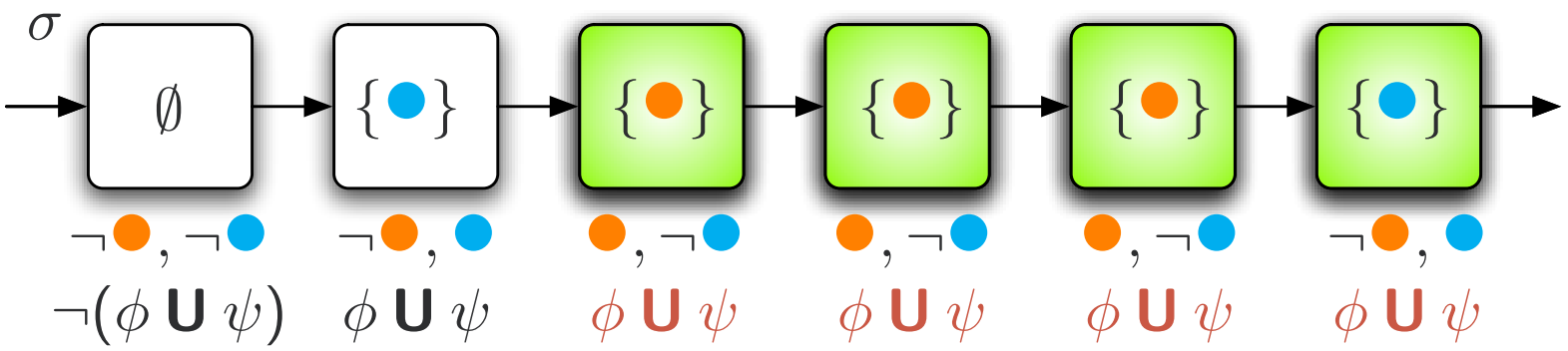
LTL Semantics in Pictures



LTL Semantics in Pictures



LTL Semantics in Pictures



LTL Semantics

Let $\sigma = \sigma_0\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\ldots$ be a trace. Then define notation:

- $\sigma|_0 = \sigma$
- $\sigma|_1 = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\ldots$
- $\sigma|_{n+1} = (\sigma|_1)|_n$

LTL Semantics

Let $\sigma = \sigma_0\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\ldots$ be a trace. Then define notation:

- $\sigma|_0 = \sigma$
- $\sigma|_1 = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\ldots$
- $\sigma|_{n+1} = (\sigma|_1)|_n$

Semantics

The models of LTL are traces. For atomic propositions, we just look at the first state:

$$\sigma \models p \quad \Leftrightarrow \quad p \in \sigma_0$$

LTL Semantics

Let $\sigma = \sigma_0\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\ldots$ be a trace. Then define notation:

- $\sigma|_0 = \sigma$
- $\sigma|_1 = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\ldots$
- $\sigma|_{n+1} = (\sigma|_1)|_n$

Semantics

The models of LTL are traces. For atomic propositions, we just look at the first state:

$$\begin{aligned}
\sigma &\models p && \Leftrightarrow && p \in \sigma_0 \\
\sigma &\models \varphi \wedge \psi && \Leftrightarrow && \sigma \models \varphi \text{ and } \sigma \models \psi \\
\sigma &\models \neg\varphi && \Leftrightarrow && \sigma \not\models \varphi
\end{aligned}$$

LTL Semantics

Let $\sigma = \sigma_0\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\ldots$ be a trace. Then define notation:

- $\sigma|_0 = \sigma$
- $\sigma|_1 = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\ldots$
- $\sigma|_{n+1} = (\sigma|_1)|_n$

Semantics

The models of LTL are traces. For atomic propositions, we just look at the first state:

$$
\begin{array}{lcl}
\sigma \models p & \Leftrightarrow & p \in \sigma_0 \\
\sigma \models \varphi \wedge \psi & \Leftrightarrow & \sigma \models \varphi \text{ and } \sigma \models \psi \\
\sigma \models \neg\varphi & \Leftrightarrow & \sigma \not\models \varphi \\
\sigma \models \mathbf{X}\,\varphi & \Leftrightarrow & \sigma|_1 \models \varphi
\end{array}
$$

LTL Semantics

Let $\sigma = \sigma_0\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\ldots$ be a trace. Then define notation:

- $\sigma|_0 = \sigma$
- $\sigma|_1 = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\ldots$
- $\sigma|_{n+1} = (\sigma|_1)|_n$

Semantics

The models of LTL are traces. For atomic propositions, we just look at the first state:

$$
\begin{array}{lcl}
\sigma \models p & \Leftrightarrow & p \in \sigma_0 \\
\sigma \models \varphi \wedge \psi & \Leftrightarrow & \sigma \models \varphi \text{ and } \sigma \models \psi \\
\sigma \models \neg\varphi & \Leftrightarrow & \sigma \not\models \varphi \\
\sigma \models \mathbf{X}\ \varphi & \Leftrightarrow & \sigma|_1 \models \varphi \\
\sigma \models \varphi\ \mathbf{UNTIL}\ \psi & \Leftrightarrow & \text{There exists an } i \text{ such that } \sigma|_i \models \psi \\
 & & \text{and for all } j < i,\ \sigma|_j \models \varphi
\end{array}
$$

We say $A \models \varphi$ iff $\forall\sigma \in \text{Traces}(A).\ \sigma \models \varphi$.

Derived Operators

The operator **F** φ ("finally" or "eventually") says that φ will be true at some point.

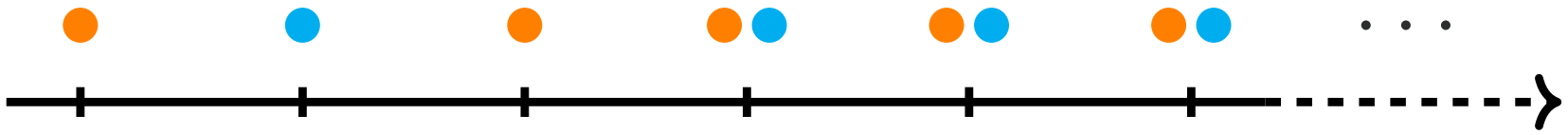
The operator **G** φ ("globally" or "always") says that φ is always true.

Exercise

- Give the semantics of **F** and **G**.
- Define **F** and **G** in terms of other operators.

More Exercises

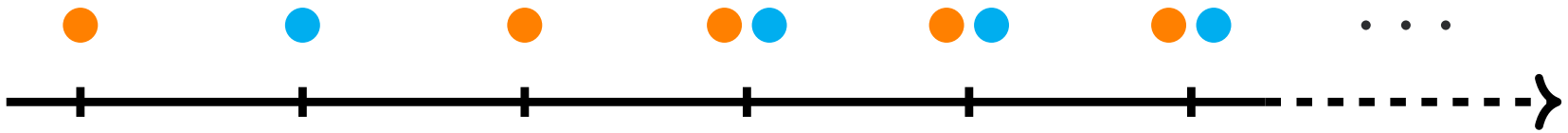
Let ρ be this trace:



$\rho \models$ ●?

More Exercises

Let ρ be this trace:

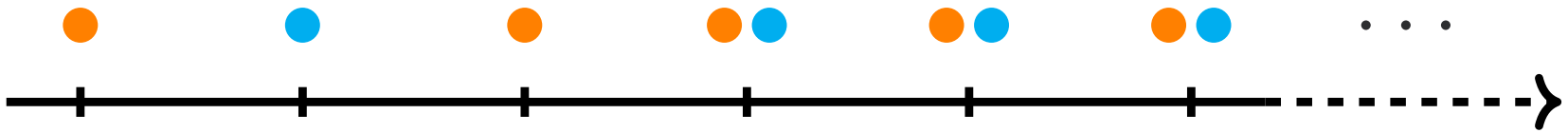


$\rho \models$ ●?

$\rho \models$ ●?

More Exercises

Let ρ be this trace:



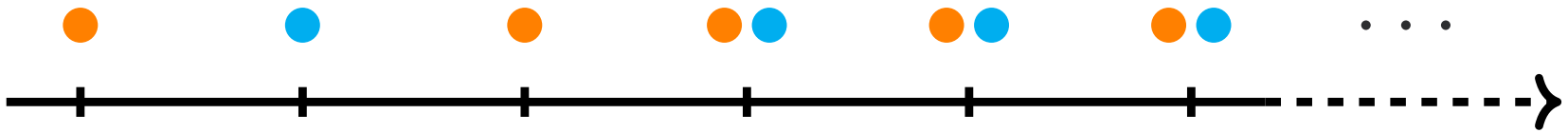
$\rho \models$ ●?

$\rho \models$ ●?

$\rho \models$ **X** ●?

More Exercises

Let ρ be this trace:



$\rho \models$ ●?

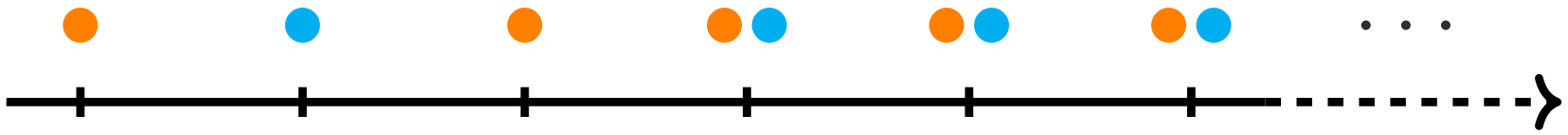
$\rho \models$ ●?

$\rho \models$ **X** ●?

$\rho \models$ **F** ●?

More Exercises

Let ρ be this trace:



$\rho \models$ ●?

$\rho \models$ ●?

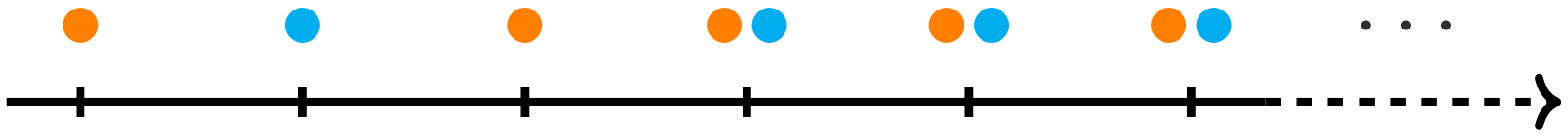
$\rho \models$ **X** ●?

$\rho \models$ **F** ●?

$\rho|_3 \models$ **F** (● $\wedge \neg$ ●)?

More Exercises

Let ρ be this trace:



$\rho \models$ ●?

$\rho \models$ ●?

$\rho \models$ **X** ●?

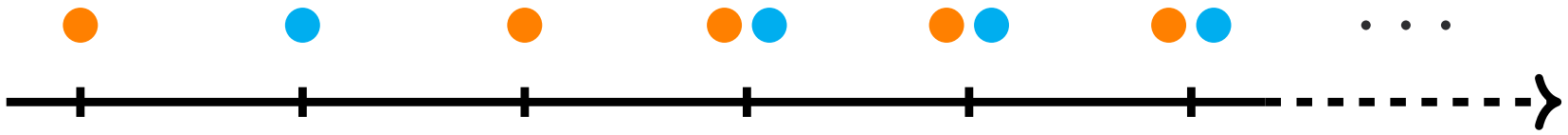
$\rho \models$ **F** ●?

$\rho|_3 \models$ **F** (● $\wedge \neg$ ●)?

$\rho \models$ **FG** (● $\wedge$ ●)?

More Exercises

Let ρ be this trace:



$\rho \models$ ●?

$\rho \models$ ●?

$\rho \models$ **X** ●?

$\rho \models$ **F** ●?

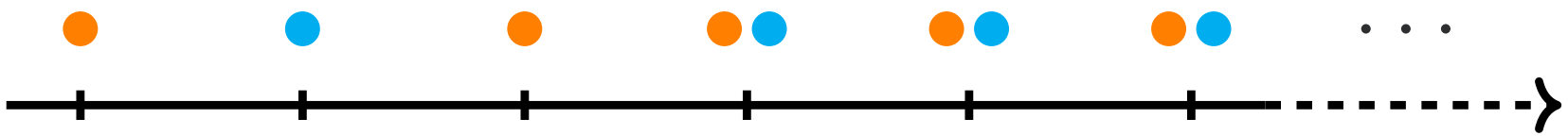
$\rho|_3 \models$ **F** (● $\wedge \neg$ ●)?

$\rho \models$ **FG** (● $\wedge$ ●)?

$\rho \models$ **G** (● **UNTIL** ●)?

More Exercises

Let ρ be this trace:

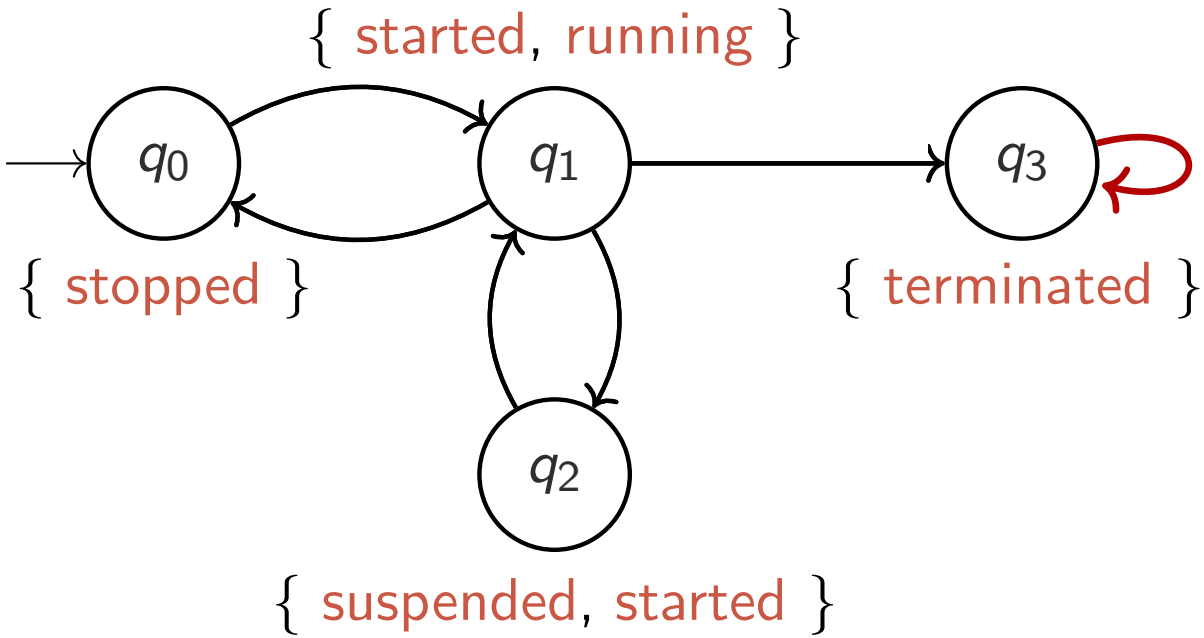


$\rho \models$ ●?
$\rho \models$ ●?
$\rho \models$ **X** ●?
$\rho \models$ **F** ●?
$\rho|_3 \models$ **F** (● $\wedge \neg$ ●)?
$\rho \models$ **FG** (● $\wedge$ ●)?
$\rho \models$ **G** (● **UNTIL** ●)?

More Derived Operators

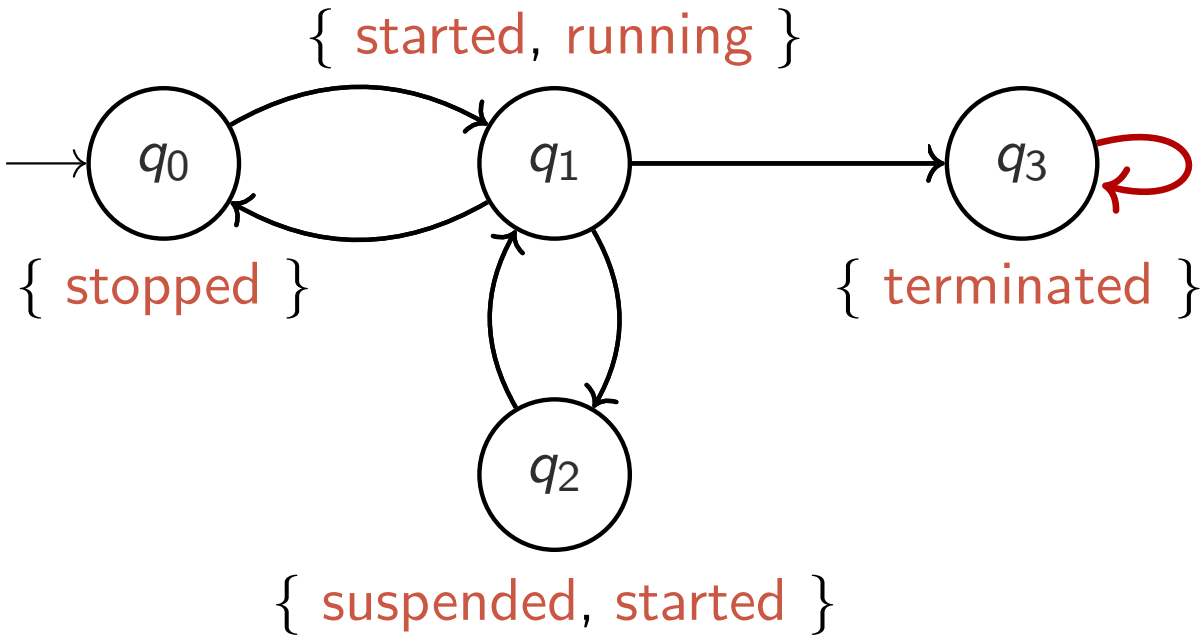
- Define "Infinitely Often" in LTL.
- Define "Almost Globally" in LTL (always true from some point onwards).

Possible Futures



We can see that it is always possible for a run to move to the terminated state. How do we express this in LTL?

Possible Futures



We can see that it is always possible for a run to move to the terminated state. How do we express this in LTL? **We can't!** — it is a *branching time* property.

Branching Time

Definition

The *computation tree* of a Kripke structure A, written $\mathrm{Tree}(A)$, is an infinite tree of Kripke structure states, where q_0 is the root and a state q' is a child of q if $q' \in \delta(q)$.

Branching Time

Definition

The *computation tree* of a Kripke structure A, written $\mathrm{Tree}(A)$, is an infinite tree of Kripke structure states, where q_0 is the root and a state q' is a child of q if $q' \in \delta(q)$.
A *path* $t_1 t_2 t_3 \ldots$ is a (infinite) sequence of computation trees such that t_{n+1} is the child of t_n. Define $\mathrm{Paths}(t)$ to be the set of all paths starting at t.

Branching Time

Definition

The *computation tree* of a Kripke structure A, written $\mathrm{Tree}(A)$, is an infinite tree of Kripke structure states, where q_0 is the root and a state q' is a child of q if $q' \in \delta(q)$.
A *path* $t_1 t_2 t_3 \ldots$ is a (infinite) sequence of computation trees such that t_{n+1} is the child of t_n. Define $\mathrm{Paths}(t)$ to be the set of all paths starting at t.

Exercise

Draw the CT for the process example.

CTL* Syntax

Definition

We define two types of formulae, state formulae and path formulae, named based on their models.

A state formula (SF) is defined as follows:

- All $p \in \mathcal{P}$ are SFs.
- Given SFs P and Q, $\neg P$ is a SF and $P \wedge Q$ is a SF.

CTL* Syntax

Definition

We define two types of formulae, state formulae and path formulae, named based on their models.

A state formula (SF) is defined as follows:

- All $p \in \mathcal{P}$ are SFs.
- Given SFs P and Q, $\neg P$ is a SF and $P \wedge Q$ is a SF.
- Given a PF φ, $\mathbf{E}\varphi$ and $\mathbf{A}\varphi$ are SFs.

CTL* Syntax

Definition

We define two types of formulae, state formulae and path formulae, named based on their models.

A state formula (SF) is defined as follows:

- All $p \in \mathcal{P}$ are SFs.
- Given SFs P and Q, $\neg P$ is a SF and $P \wedge Q$ is a SF.
- Given a PF φ, $\mathbf{E}\varphi$ and $\mathbf{A}\varphi$ are SFs.

A path formula (PF) is defined much like LTL:

- If P is a SF, then P is a PF.

CTL* Syntax

Definition

We define two types of formulae, state formulae and path formulae, named based on their models.

A state formula (SF) is defined as follows:

- All $p \in \mathcal{P}$ are SFs.
- Given SFs P and Q, $\neg P$ is a SF and $P \wedge Q$ is a SF.
- Given a PF φ, $\mathbf{E}\varphi$ and $\mathbf{A}\varphi$ are SFs.

A path formula (PF) is defined much like LTL:

- If P is a SF, then P is a PF.
- Given PFs φ and ψ, $\neg\varphi$ is a PF and $\varphi \wedge \psi$ is a PF.

CTL* Syntax

Definition

We define two types of formulae, state formulae and path formulae, named based on their models.

A state formula (SF) is defined as follows:

- All $p \in \mathcal{P}$ are SFs.
- Given SFs P and Q, $\neg P$ is a SF and $P \wedge Q$ is a SF.
- Given a PF φ, $\mathbf{E}\varphi$ and $\mathbf{A}\varphi$ are SFs.

A path formula (PF) is defined much like LTL:

- If P is a SF, then P is a PF.
- Given PFs φ and ψ, $\neg\varphi$ is a PF and $\varphi \wedge \psi$ is a PF.
- Given a PF φ then $\mathbf{X}\varphi$ is a PF.
- Given PFs φ and ψ, φ **UNTIL** ψ is a PF.

Initially, we start with state formulae (SFs).

CTL* Semantics

State Semantics

$$t \models p \quad \Leftrightarrow \quad p \in L(t_{\text{root}})$$

CTL* Semantics

State Semantics

$$
\begin{array}{lcl}
t \models p & \Leftrightarrow & p \in L(t_{\text{root}}) \\
t \models P \wedge Q & \Leftrightarrow & t \models P \text{ and } t \models Q \\
t \models \neg P & \Leftrightarrow & t \not\models P
\end{array}
$$

CTL* Semantics

State Semantics

$$
\begin{aligned}
t \models p &\quad\Leftrightarrow\quad p \in L(t_{\text{root}}) \\
t \models P \wedge Q &\quad\Leftrightarrow\quad t \models P \text{ and } t \models Q \\
t \models \neg P &\quad\Leftrightarrow\quad t \not\models P \\
t \models \mathbf{E}\,\varphi &\quad\Leftrightarrow\quad \exists \rho \in \text{Paths}(t).\ \rho \models \varphi
\end{aligned}
$$

CTL* Semantics

State Semantics

$$
\begin{array}{lcl}
t \models p & \Leftrightarrow & p \in L(t_{\text{root}}) \\
t \models P \wedge Q & \Leftrightarrow & t \models P \text{ and } t \models Q \\
t \models \neg P & \Leftrightarrow & t \not\models P \\
t \models \mathbf{E}\,\varphi & \Leftrightarrow & \exists \rho \in \text{Paths}(t).\ \rho \models \varphi \\
t \models \mathbf{A}\,\varphi & \Leftrightarrow & \forall \rho \in \text{Paths}(t).\ \rho \models \varphi
\end{array}
$$

CTL* Semantics

State Semantics

$$
\begin{array}{lcl}
t \models p & \Leftrightarrow & p \in L(t_{\text{root}}) \\
t \models P \wedge Q & \Leftrightarrow & t \models P \text{ and } t \models Q \\
t \models \neg P & \Leftrightarrow & t \not\models P \\
t \models \mathbf{E}\,\varphi & \Leftrightarrow & \exists \rho \in \text{Paths}(t).\ \rho \models \varphi \\
t \models \mathbf{A}\,\varphi & \Leftrightarrow & \forall \rho \in \text{Paths}(t).\ \rho \models \varphi
\end{array}
$$

Path Semantics

$$
\rho \models P \quad \Leftrightarrow \quad \rho_0 \models P
$$

CTL* Semantics

State Semantics

$$
\begin{array}{lcl}
t \models p & \Leftrightarrow & p \in L(t_{\text{root}}) \\
t \models P \wedge Q & \Leftrightarrow & t \models P \text{ and } t \models Q \\
t \models \neg P & \Leftrightarrow & t \not\models P \\
t \models \mathbf{E}\ \varphi & \Leftrightarrow & \exists \rho \in \text{Paths}(t).\ \rho \models \varphi \\
t \models \mathbf{A}\ \varphi & \Leftrightarrow & \forall \rho \in \text{Paths}(t).\ \rho \models \varphi
\end{array}
$$

Path Semantics

$$
\begin{array}{lcl}
\rho \models P & \Leftrightarrow & \rho_0 \models P \\
\rho \models \varphi \wedge \psi & \Leftrightarrow & \rho \models \varphi \text{ and } \rho \models \psi \\
\rho \models \neg \varphi & \Leftrightarrow & \rho \not\models \varphi
\end{array}
$$

CTL* Semantics

State Semantics

$$
\begin{array}{lcl}
t \models p & \Leftrightarrow & p \in L(t_{\text{root}}) \\
t \models P \wedge Q & \Leftrightarrow & t \models P \text{ and } t \models Q \\
t \models \neg P & \Leftrightarrow & t \not\models P \\
t \models \mathbf{E}\ \varphi & \Leftrightarrow & \exists \rho \in \text{Paths}(t).\ \rho \models \varphi \\
t \models \mathbf{A}\ \varphi & \Leftrightarrow & \forall \rho \in \text{Paths}(t).\ \rho \models \varphi
\end{array}
$$

Path Semantics

$$
\begin{array}{lcl}
\rho \models P & \Leftrightarrow & \rho_0 \models P \\
\rho \models \varphi \wedge \psi & \Leftrightarrow & \rho \models \varphi \text{ and } \rho \models \psi \\
\rho \models \neg \varphi & \Leftrightarrow & \rho \not\models \varphi \\
\rho \models \mathbf{X}\ \varphi & \Leftrightarrow & \rho|_1 \models \varphi
\end{array}
$$

CTL* Semantics

State Semantics

$$
\begin{array}{lcl}
t \models p & \Leftrightarrow & p \in L(t_{\text{root}}) \\
t \models P \wedge Q & \Leftrightarrow & t \models P \text{ and } t \models Q \\
t \models \neg P & \Leftrightarrow & t \not\models P \\
t \models \mathbf{E}\ \varphi & \Leftrightarrow & \exists \rho \in \text{Paths}(t).\ \rho \models \varphi \\
t \models \mathbf{A}\ \varphi & \Leftrightarrow & \forall \rho \in \text{Paths}(t).\ \rho \models \varphi
\end{array}
$$

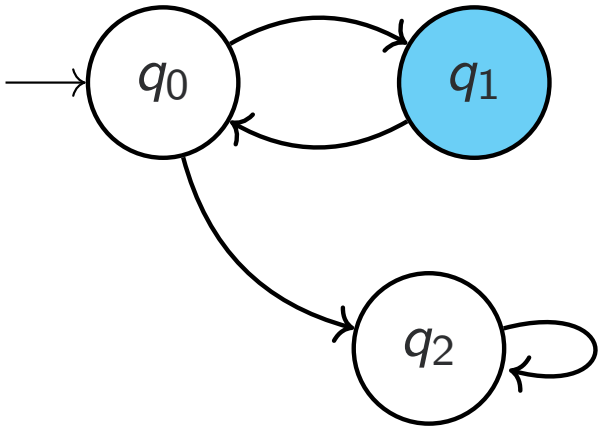
Path Semantics

$$
\begin{array}{lcl}
\rho \models P & \Leftrightarrow & \rho_0 \models P \\
\rho \models \varphi \wedge \psi & \Leftrightarrow & \rho \models \varphi \text{ and } \rho \models \psi \\
\rho \models \neg\varphi & \Leftrightarrow & \rho \not\models \varphi \\
\rho \models \mathbf{X}\ \varphi & \Leftrightarrow & \rho|_1 \models \varphi \\
\rho \models \varphi\ \mathbf{UNTIL}\ \psi & \Leftrightarrow & \text{There exists an } i \text{ such that } \rho|_i \models \psi \\
 & & \text{and for all } j < i,\ \rho|_j \models \varphi
\end{array}
$$

CTL* Examples

We say a Kripke structure A satisfies a CTL* property P, that is, $A \models P$ iff $\mathrm{Tree}(A) \models P$

Given this automaton A:

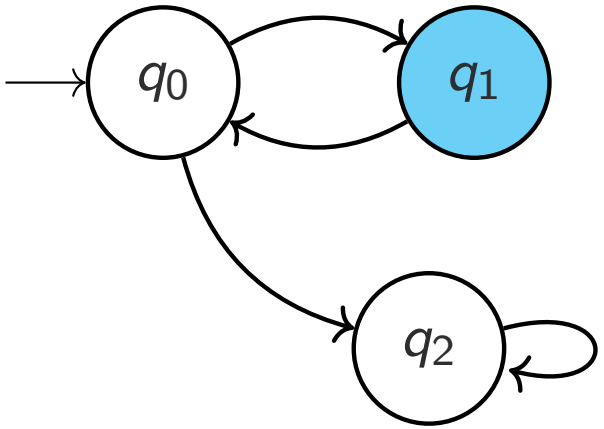


- $A \models \mathbf{E}\,\mathbf{G}\,\mathbf{F}$ ●?

CTL* Examples

We say a Kripke structure A satisfies a CTL* property P, that is, $A \models P$ iff $\mathrm{Tree}(A) \models P$

Given this automaton A:

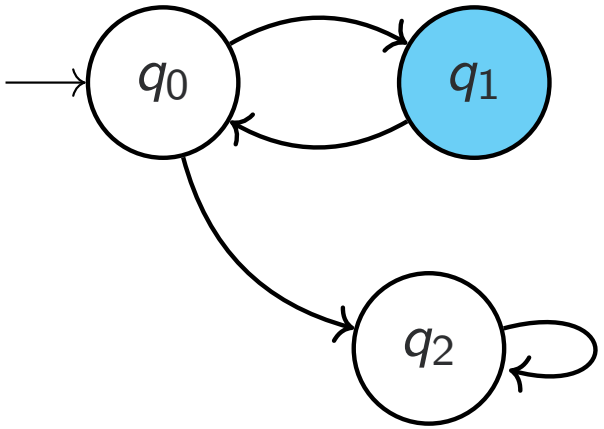


- $A \models \mathbf{E}\,\mathbf{G}\,\mathbf{F}$ ●?
- $A \models \mathbf{A}\,\mathbf{G}\,\mathbf{F}$ ●?

CTL* Examples

We say a Kripke structure A satisfies a CTL* property P, that is, $A \models P$ iff $\mathrm{Tree}(A) \models P$

Given this automaton A:

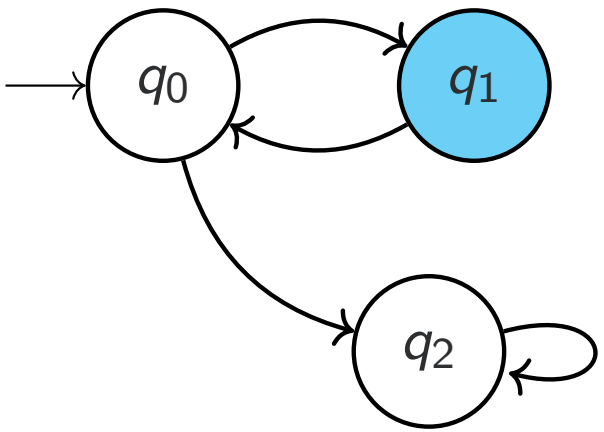


- $A \models$ **E G F** ●?
- $A \models$ **A G F** ●?
- $A \models$ **A F** ●?

CTL* Examples

We say a Kripke structure A satisfies a CTL* property P, that is, $A \models P$ iff $\mathrm{Tree}(A) \models P$

Given this automaton A:



- $A \models \mathbf{E}\ \mathbf{G}\ \mathbf{F}$ ●?
- $A \models \mathbf{A}\ \mathbf{G}\ \mathbf{F}$ ●?
- $A \models \mathbf{A}\ \mathbf{F}$ ●?
- $A \models \mathbf{A}\ \mathbf{E}\ \mathbf{F}$ ●?

Simplifying

CTL* is very expressive but very complicated.

It's also extremely hard to model check, which we'll get to later.

CTL* to CTL

Keep state formulae the same:

- All $p \in \mathcal{P}$ are SFs.
- Given SFs P and Q, $\neg P$ is a SF and $P \wedge Q$ is a SF.
- Given a PF φ, $\mathbf{E}\varphi$ and $\mathbf{A}\varphi$ are SFs.

But we force path formulae to go straight back to state formulae immediately with a temporal operator:

- Given a SF P then $\mathbf{X}P$ is a PF.
- Given SFs P and Q, P **UNTIL** Q is a PF.

Examples

Which of the following CTL* formulae are CTL formulae?

- a **UNTIL** (b **UNTIL** c)
- **A** (a **UNTIL** c)
- **X X** a
- **X A** a
- **A** (a **UNTIL** (b **UNTIL** c))
- **A E** (a **UNTIL** b)
- **E X** a
- **X E** a

Non-mutual CTL Syntax

Simpler CTL Syntax

A CTL formula is defined as follows:

- All $p \in \mathcal{P}$ are formulae.
- Given formulae P and Q, $\neg P$ is a formula and $P \wedge Q$ is a formula.
- Given a formula P, **EX** P and **AX** P are formulae.
- Given formulae P and Q, **E**(P **UNTIL** Q) and **A**(P **UNTIL** Q) are formulae.

Simpler CTL Semantics

Semantics are as with CTL*, but can be defined more directly:

Semantics

$$t \models p \quad \Leftrightarrow \quad p \in L(t_{\text{root}})$$

Simpler CTL Semantics

Semantics are as with CTL*, but can be defined more directly:

Semantics

$$
\begin{aligned}
t \models p &\quad\Leftrightarrow\quad p \in L(t_{\text{root}}) \\
t \models P \wedge Q &\quad\Leftrightarrow\quad t \models P \text{ and } t \models Q \\
t \models \neg P &\quad\Leftrightarrow\quad t \not\models P
\end{aligned}
$$

Simpler CTL Semantics

Semantics are as with CTL*, but can be defined more directly:

Semantics

$$
\begin{array}{lcl}
t \models p & \Leftrightarrow & p \in L(t_{\text{root}}) \\
t \models P \wedge Q & \Leftrightarrow & t \models P \text{ and } t \models Q \\
t \models \neg P & \Leftrightarrow & t \not\models P \\
t \models \mathbf{EX}\ P & \Leftrightarrow & \exists \rho \in \text{Paths}(t).\ \rho_1 \models P
\end{array}
$$

Simpler CTL Semantics

Semantics are as with CTL*, but can be defined more directly:

Semantics

$$
\begin{array}{lcl}
t \models p & \Leftrightarrow & p \in L(t_{\text{root}}) \\
t \models P \wedge Q & \Leftrightarrow & t \models P \text{ and } t \models Q \\
t \models \neg P & \Leftrightarrow & t \not\models P \\
t \models \mathbf{EX}\ P & \Leftrightarrow & \exists \rho \in \text{Paths}(t).\ \rho_1 \models P \\
t \models \mathbf{AX}\ P & \Leftrightarrow & \forall \rho \in \text{Paths}(t).\ \rho_1 \models P
\end{array}
$$

Simpler CTL Semantics

Semantics are as with CTL*, but can be defined more directly:

Semantics

$$
\begin{array}{lcl}
t \models p & \Leftrightarrow & p \in L(t_{\text{root}}) \\
t \models P \wedge Q & \Leftrightarrow & t \models P \text{ and } t \models Q \\
t \models \neg P & \Leftrightarrow & t \not\models P \\
t \models \mathbf{EX}\ P & \Leftrightarrow & \exists \rho \in \text{Paths}(t).\ \rho_1 \models P \\
t \models \mathbf{AX}\ P & \Leftrightarrow & \forall \rho \in \text{Paths}(t).\ \rho_1 \models P \\
t \models \mathbf{A}(P\ \mathbf{UNTIL}\ Q) & \Leftrightarrow & \forall \rho \in \text{Paths}(t), \text{ there } \exists \text{ an } i \text{ such that:} \\
 & & \quad \rho_i \models Q \text{ and } \forall j < i.\ \rho_j \models P
\end{array}
$$

Simpler CTL Semantics

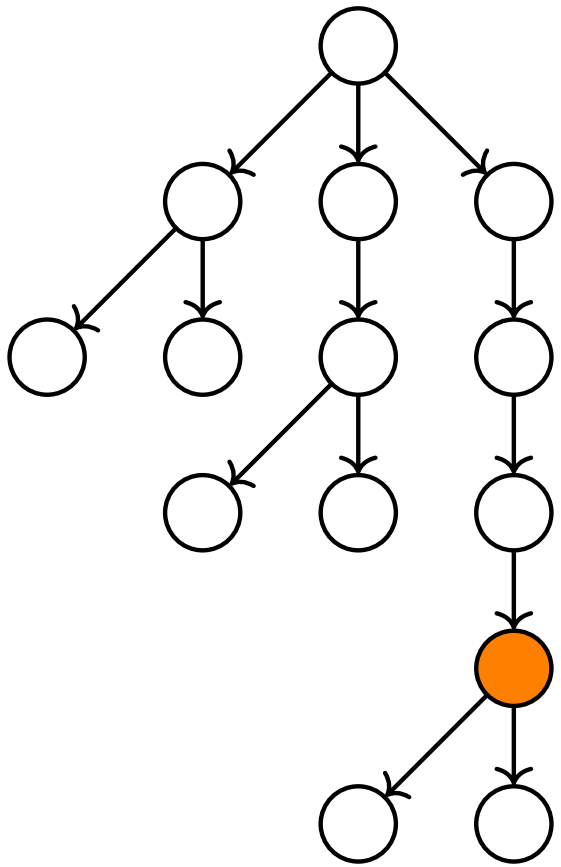
Semantics are as with CTL*, but can be defined more directly:

Semantics

$$
\begin{array}{lcl}
t \models p & \Leftrightarrow & p \in L(t_{\text{root}}) \\
t \models P \wedge Q & \Leftrightarrow & t \models P \text{ and } t \models Q \\
t \models \neg P & \Leftrightarrow & t \not\models P \\
t \models \mathbf{EX}\ P & \Leftrightarrow & \exists \rho \in \text{Paths}(t).\ \rho_1 \models P \\
t \models \mathbf{AX}\ P & \Leftrightarrow & \forall \rho \in \text{Paths}(t).\ \rho_1 \models P \\
t \models \mathbf{A}(P\ \mathbf{UNTIL}\ Q) & \Leftrightarrow & \forall \rho \in \text{Paths}(t), \text{ there } \exists \text{ an } i \text{ such that:} \\
 & & \quad \rho_i \models Q \text{ and } \forall j < i.\ \rho_j \models P \\
t \models \mathbf{E}(P\ \mathbf{UNTIL}\ Q) & \Leftrightarrow & \exists \rho \in \text{Paths}(t) \text{ and an } i \text{ such that:} \\
 & & \quad \rho_i \models Q \text{ and } \forall j < i.\ \rho_j \models P
\end{array}
$$

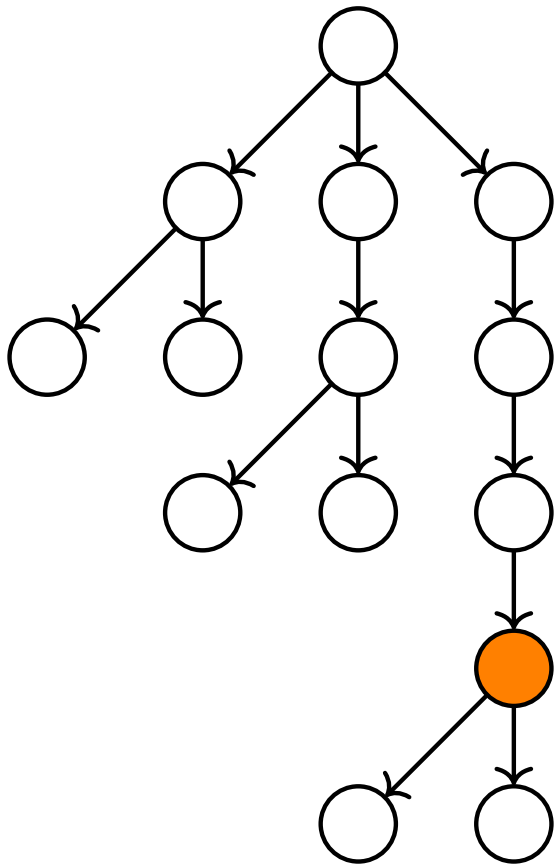
Derived Operators

Define **EF●**:



Derived Operators

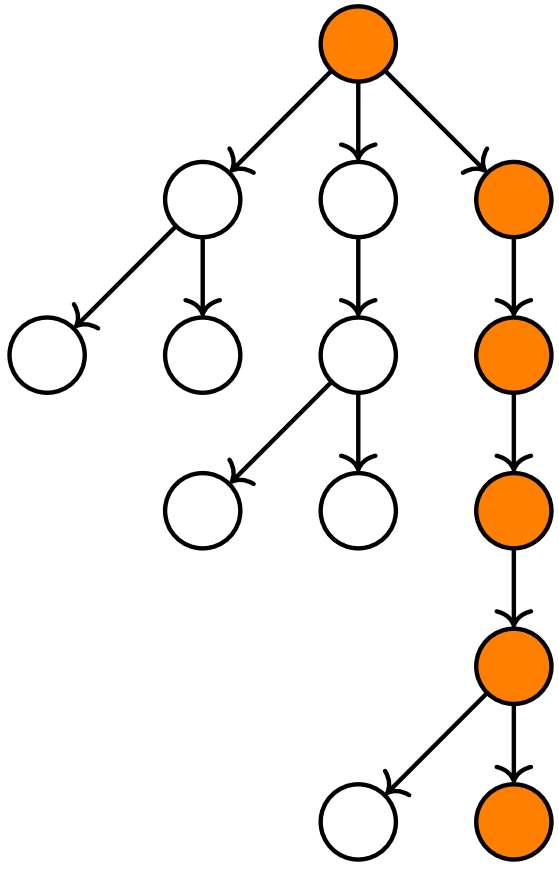
Define **EF**●:



E(True **UNTIL** ●)

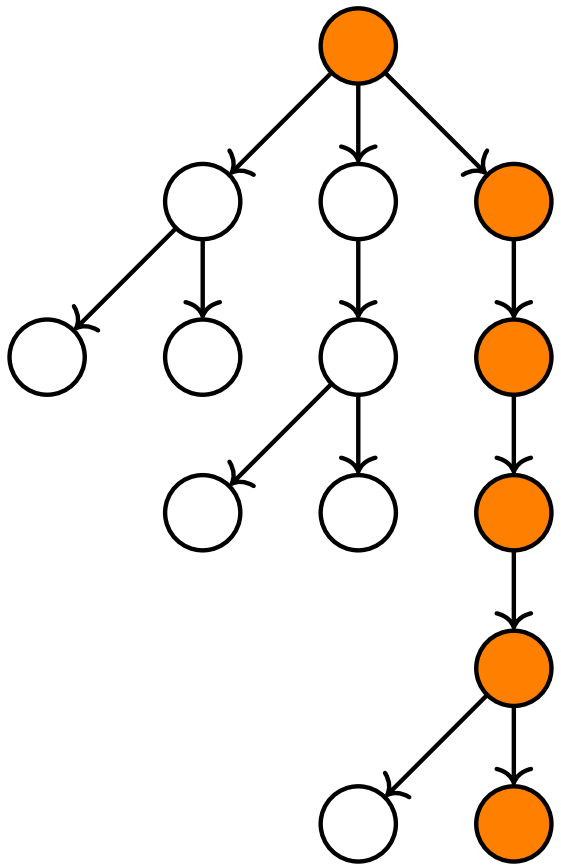
Derived Operators

Define **EG●**:



Derived Operators

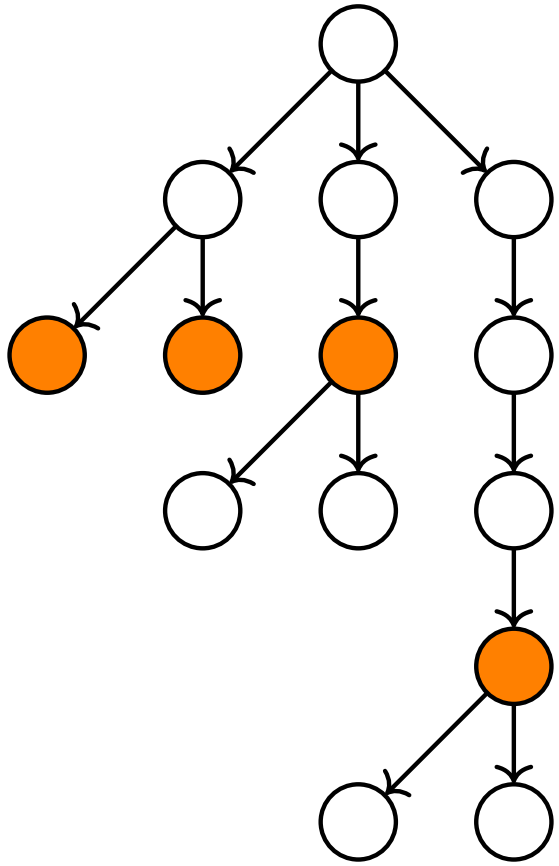
Define **EG**●:



¬**A**(True **UNTIL** ¬●)

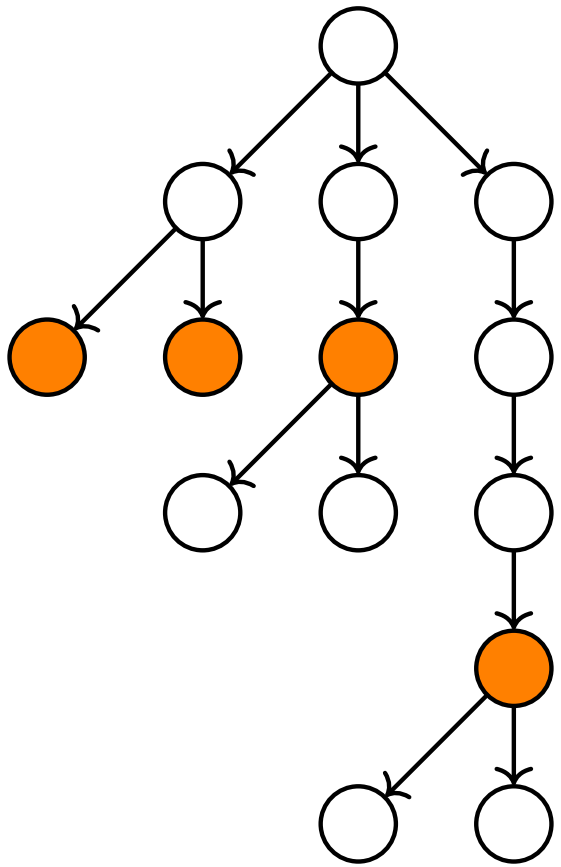
Derived Operators

Define **AF●**:



Derived Operators

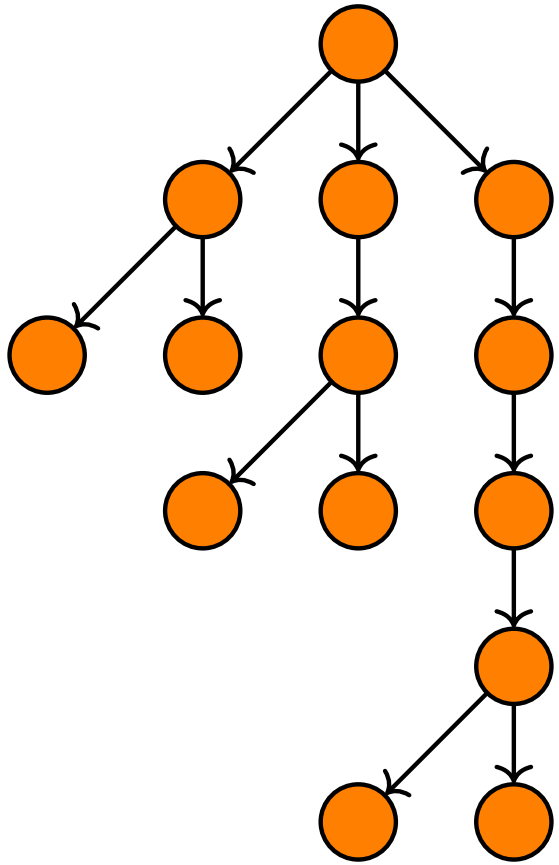
Define **AF**●:



A(True **UNTIL** ●)

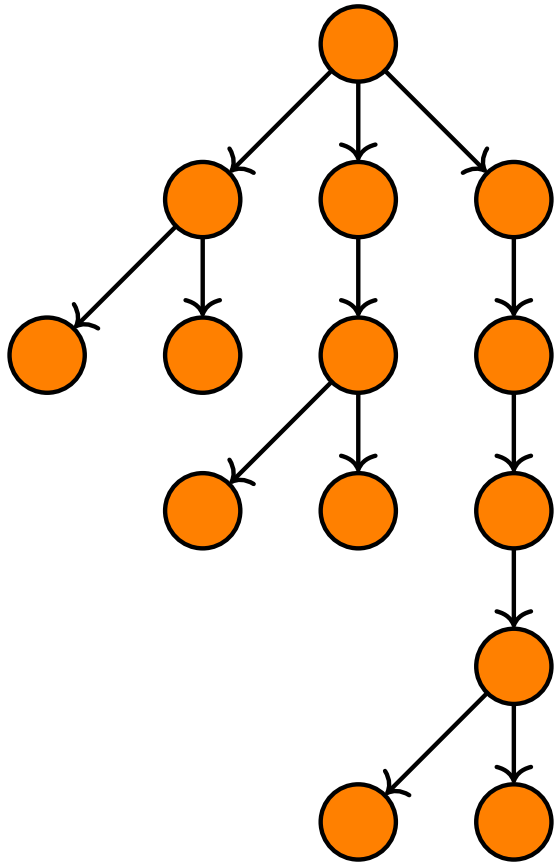
Derived Operators

Define **AG●**:



Derived Operators

Define **AG●**:



¬EF ¬●

Bibliography

- Huth/Ryan: Logic in Computer Science, Section 3.2 and 3.4
- Bayer/Katoen: Principles of Model Checking Sections 5.1 and 6.2