CTL Properties

Introduction

Now you know:

- how to build a Reachability Graph
- how it can be used for system analysis
- LTL logics to express properties

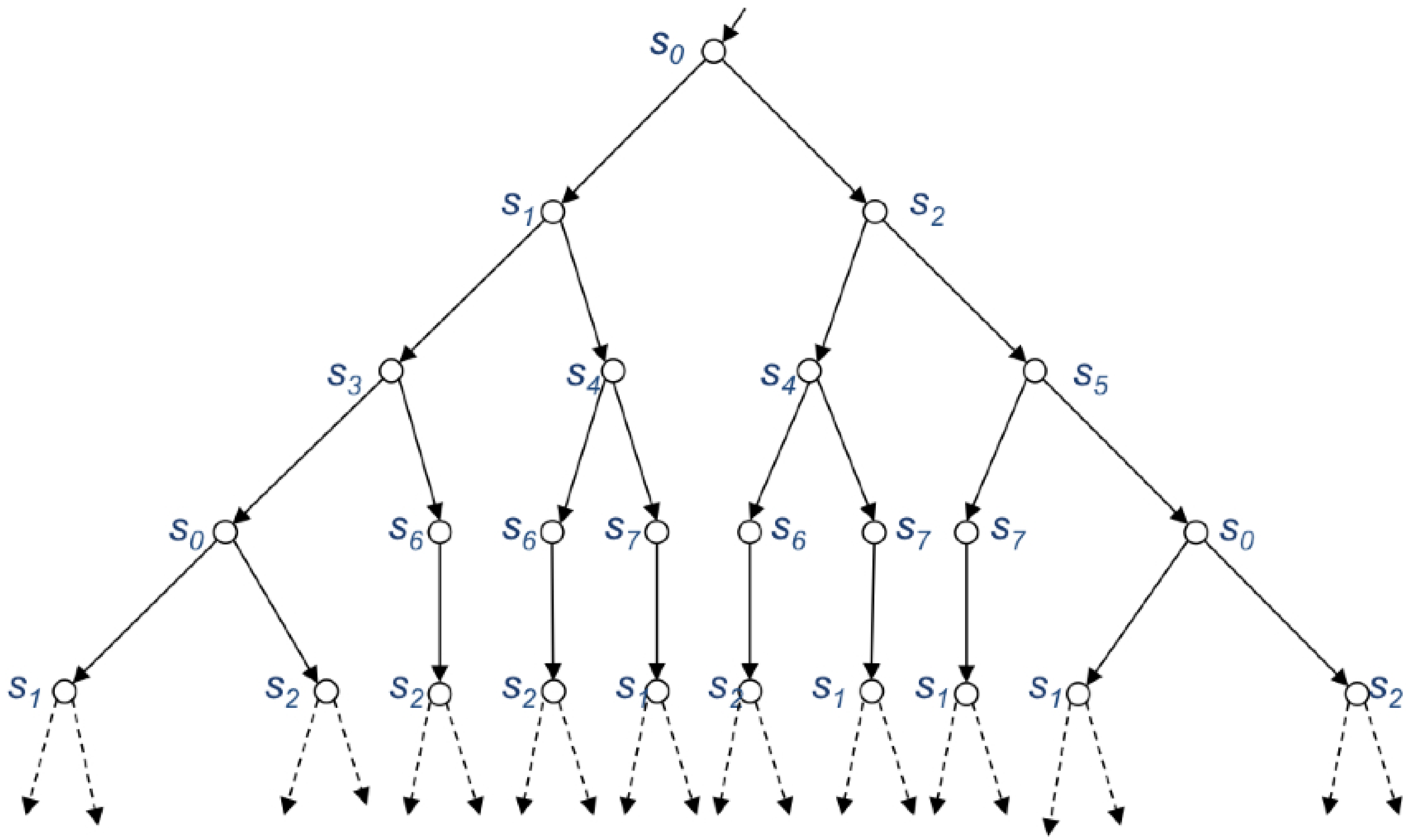
Introduction

Now you know:

- how to build a Reachability Graph
- how it can be used for system analysis
- LTL logics to express properties

Let's see CTL logics to express additional properties

The branching time semantics



Logics to express branching time properties

- CTL = Computational Tree Logic.
- Syntax: let *AP* be a set of atomic propositions.
  - $a \in AP$ is an CTL formula.
  - If ϕ_1 and ϕ_2 are CTL formulae then so are
    $\neg\phi_1$ $\phi_1 \wedge \phi_2$ $EX\ \phi_1$ $EG\ \phi_1$ $E\phi_1\ U\ \phi_2$
    where X stands for “next”, G for “globally”, E for “exists” and U for “until”.

Logics to express branching time properties

- CTL = Computational Tree Logic.
- Syntax: let AP be a set of atomic propositions.
  - $a \in AP$ is an CTL formula.
  - If ϕ_1 and ϕ_2 are CTL formulae then so are
    $\neg\phi_1$ $\quad$ $\phi_1 \wedge \phi_2$ $\quad$ EX ϕ_1 $\quad$ EG ϕ_1 $\quad$ Eϕ_1 U ϕ_2
    where X stands for "next", G for "globally", E for "exists" and U for "until".
- Let $K = \langle S, I, \rightarrow, s_0 \rangle$ be a Kripke structure. To each CTL formula ϕ, we associate a set $S_k(\phi) \subseteq S$ of states, s.t.:

$$
\begin{aligned}
s \in S_k(a) &\Leftrightarrow a \in I(s) \\
s \in S_k(\neg\phi) &\Leftrightarrow s \notin S_k(\phi) \\
s \in S_k(\phi_1 \wedge \phi_2) &\Leftrightarrow s \in S_k(\phi_1) \cap S_k(\phi_2) \\
s \in S_k(\mathrm{EX}\phi) &\Leftrightarrow \exists s' : s \rightarrow s' \wedge s' \in S_k(\phi) \\
s \in S_k(\mathrm{EG}\phi) &\Leftrightarrow \exists \text{ a run } \tau \text{ of K s.t. } \tau(0) = s \wedge \forall\, i \geq 0, \tau(i) \in S_k(\phi) \\
s \in S_k(\mathrm{E}\phi_1 \,\mathrm{U}\, \phi_2) &\Leftrightarrow \exists \text{ a run } \tau \text{ of K s.t. } \tau(0) = s \wedge \exists\, i, \tau(i) \in S_k(\phi_2) \wedge \\
&\qquad \forall k < i, \tau(k) \in S_k(\phi_1)
\end{aligned}
$$

Logics to express branching time properties

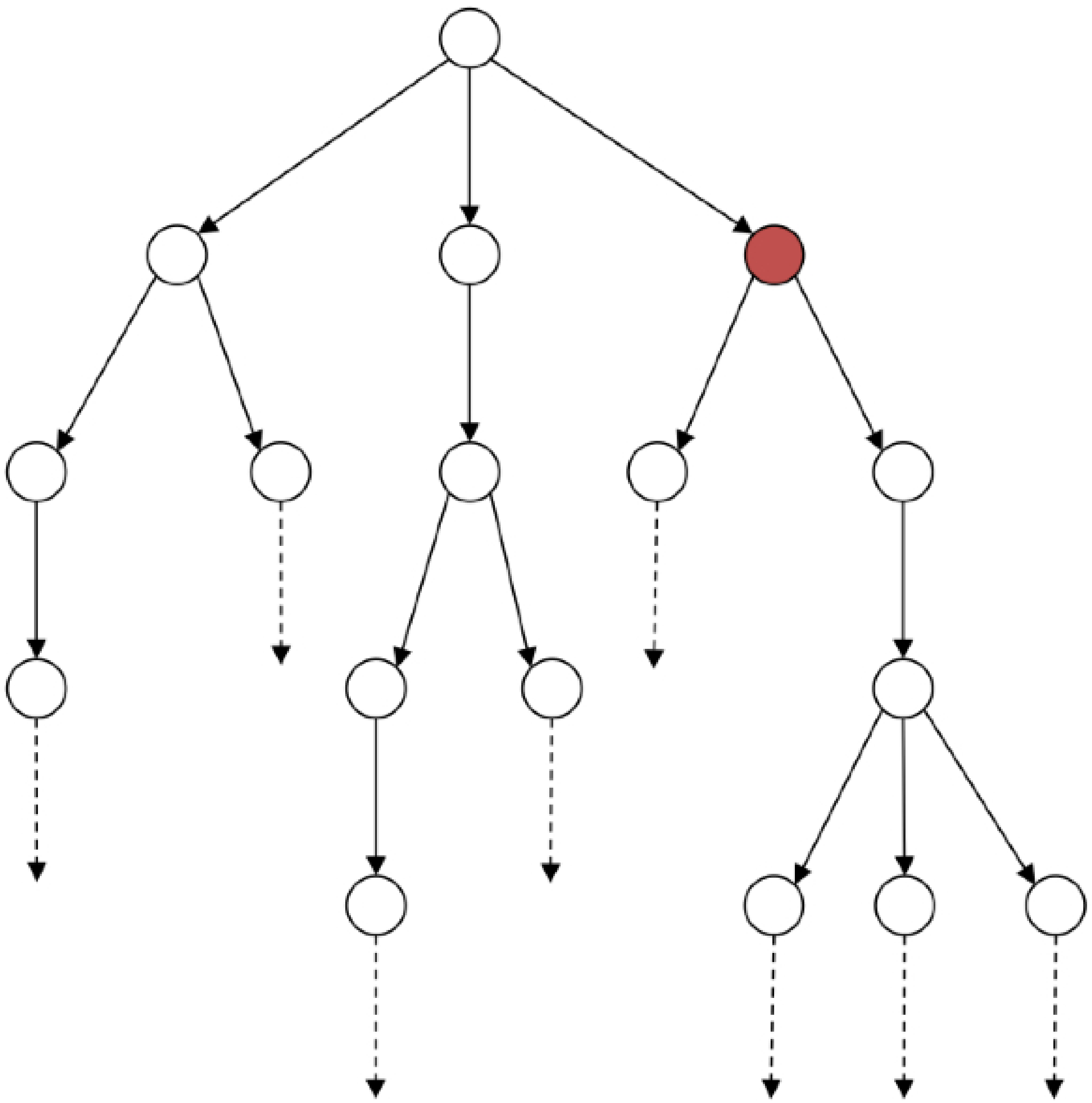
- CTL = Computational Tree Logic.
- Syntax: let AP be a set of atomic propositions.
  - $a \in AP$ is an CTL formula.
  - If ϕ_1 and ϕ_2 are CTL formulae then so are
    $\neg\phi_1$ $\qquad \phi_1 \wedge \phi_2$ $\qquad \mathrm{EX}\,\phi_1$ $\qquad \mathrm{EG}\,\phi_1$ $\qquad \mathrm{E}\phi_1\,\mathrm{U}\,\phi_2$
    where X stands for "next", G for "globally", E for "exists" and U for "until".
- Let $K = \langle S, l, \rightarrow, s_0 \rangle$ be a Kripke structure. To each CTL formula ϕ, we associate a set $S_k(\phi) \subseteq S$ of states, s.t.:

$$
\begin{aligned}
s \in S_k(a) &\Leftrightarrow a \in l(s)\\
s \in S_k(\neg\phi) &\Leftrightarrow s \notin S_k(\phi)\\
s \in S_k(\phi_1 \wedge \phi_2) &\Leftrightarrow s \in S_k(\phi_1) \cap S_k(\phi_2)\\
s \in S_k(\mathrm{EX}\phi) &\Leftrightarrow \exists s' : s \rightarrow s' \wedge s' \in S_k(\phi)\\
s \in S_k(\mathrm{EG}\phi) &\Leftrightarrow \exists \text{ a run } \tau \text{ of K s.t. } \tau(0) = s \wedge \forall\, i \geq 0, \tau(i) \in S_k(\phi)\\
s \in S_k(\mathrm{E}\phi_1\,\mathrm{U}\,\phi_2) &\Leftrightarrow \exists \text{ a run } \tau \text{ of K s.t. } \tau(0) = s \wedge \exists\, i, \tau(i) \in S_k(\phi_2) \wedge\\
&\qquad \forall k < i, \tau(k) \in S_k(\phi_1)
\end{aligned}
$$

- K satisfies a CTL formula ϕ iff $s_0 \in S_k(\phi)$

Illustration of the CTL semantics (1/8)

$EX\ p$



$EG\ p$

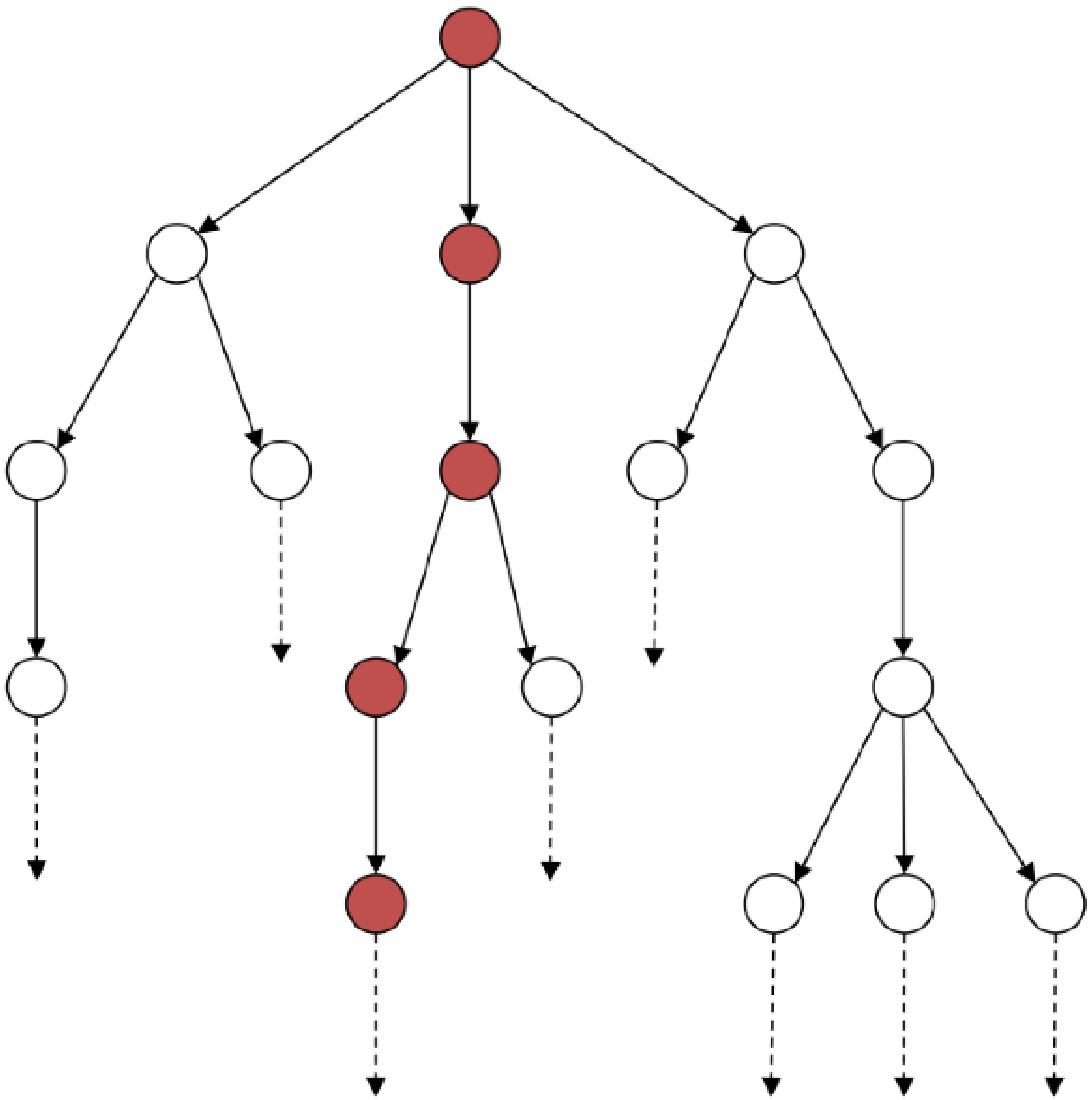
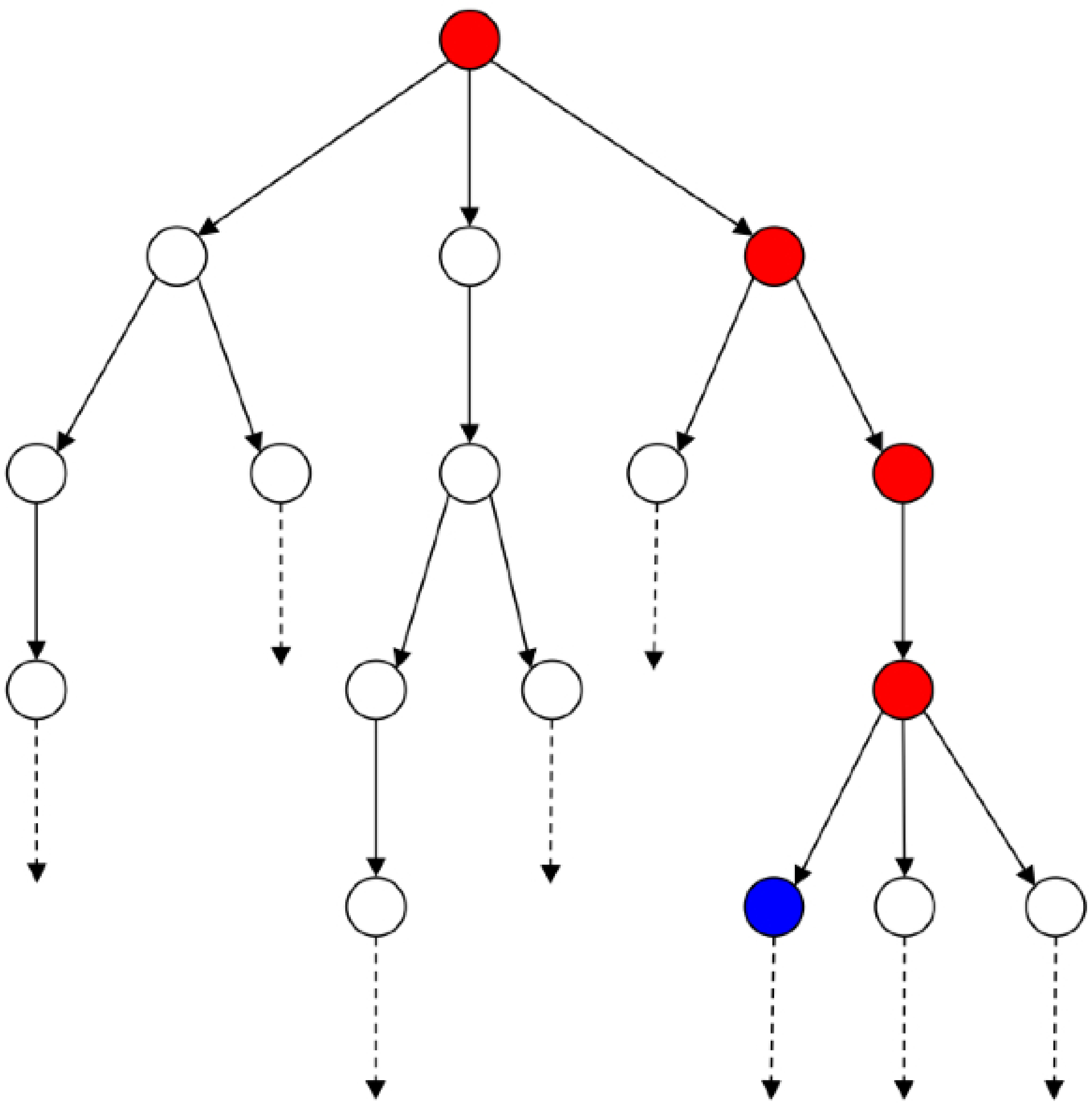
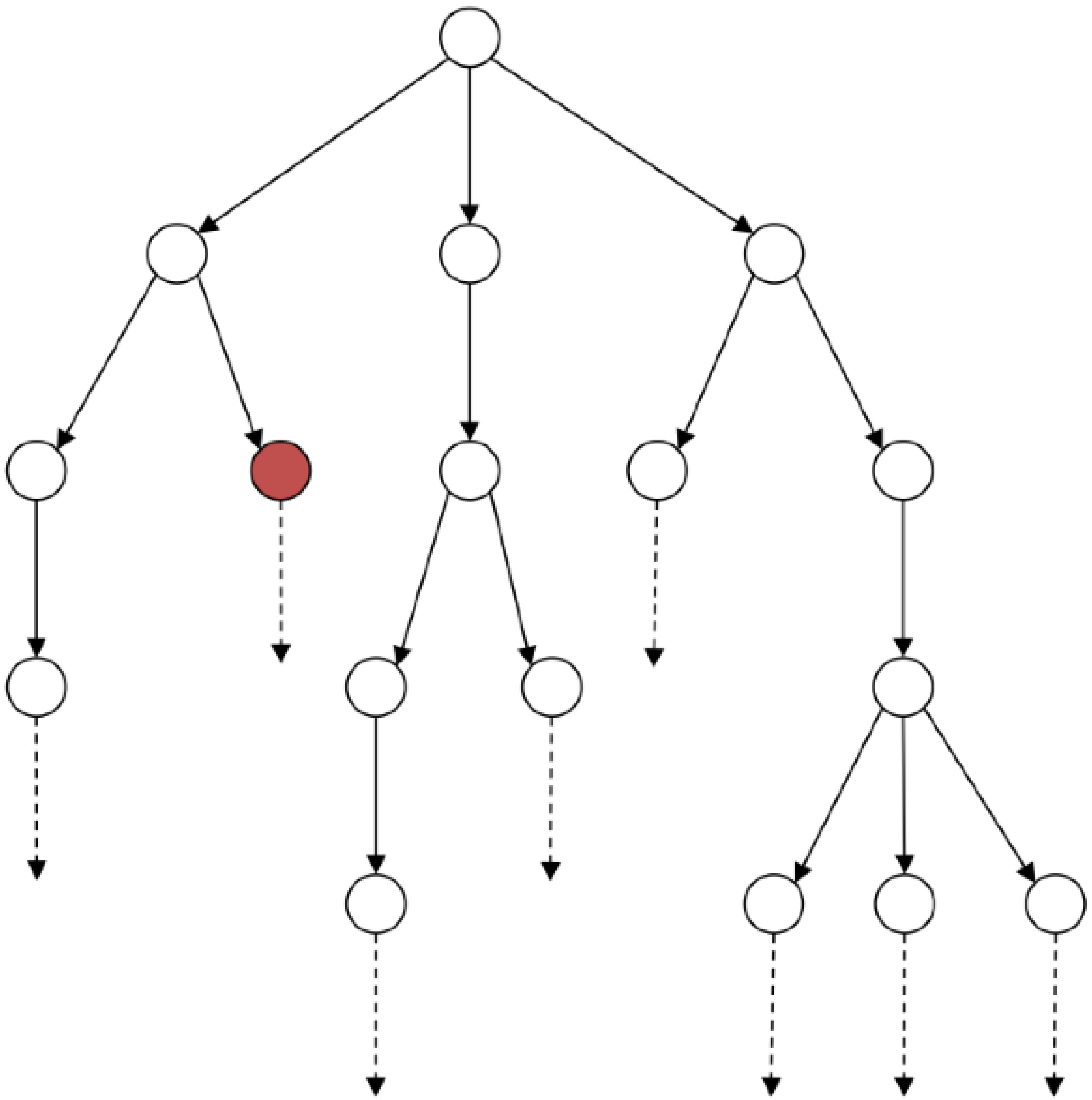


Illustration of the CTL semantics (3/8)

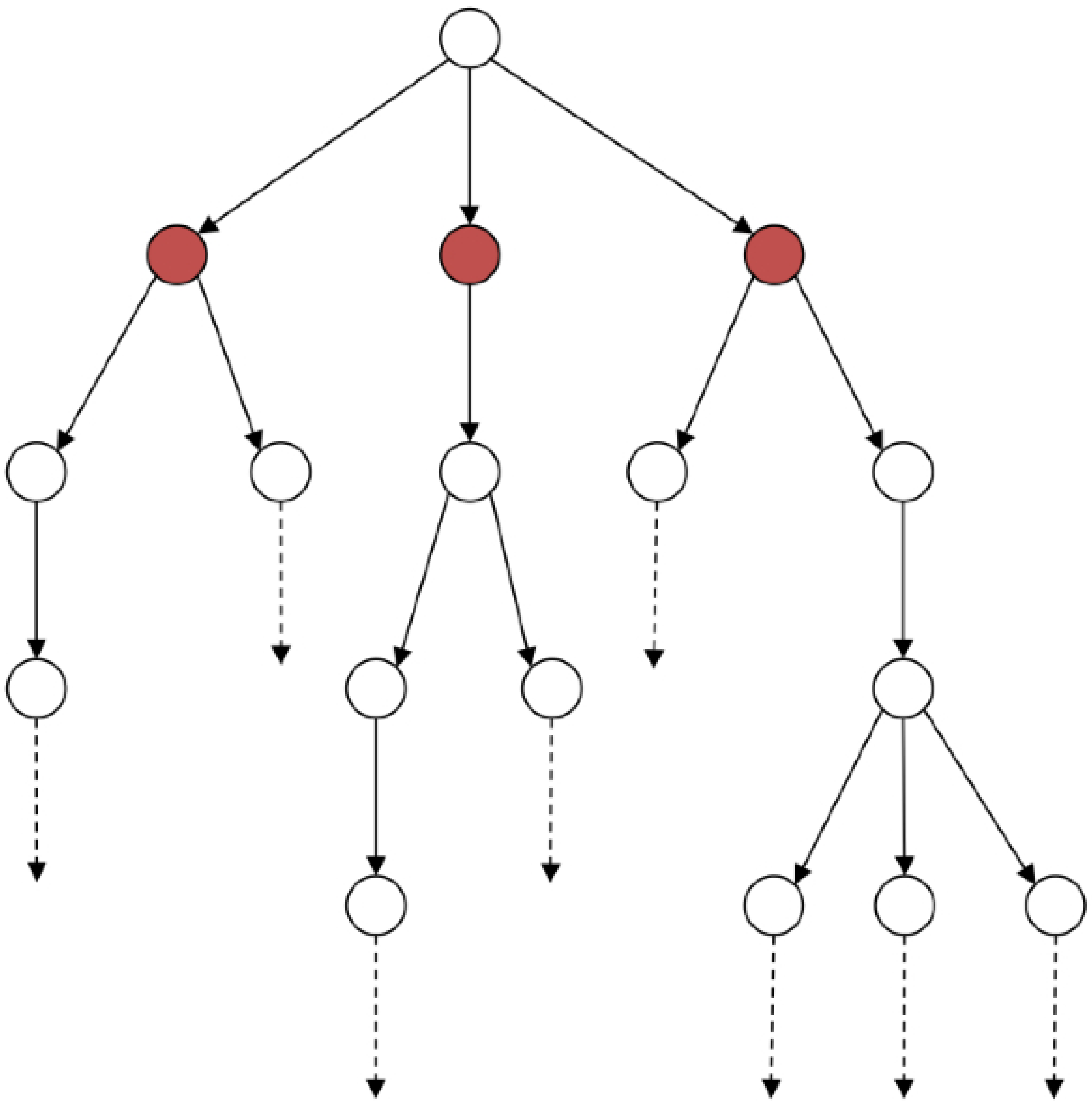
$E\, q\, U\, p$



$$EF\ p = (\mathrm{E}\ true\ U\ p)$$



$$AX\ p = \neg\ EX\ \neg\ p$$



$$AG\ p = \neg\ (E\ true\ U\ \neg p)$$

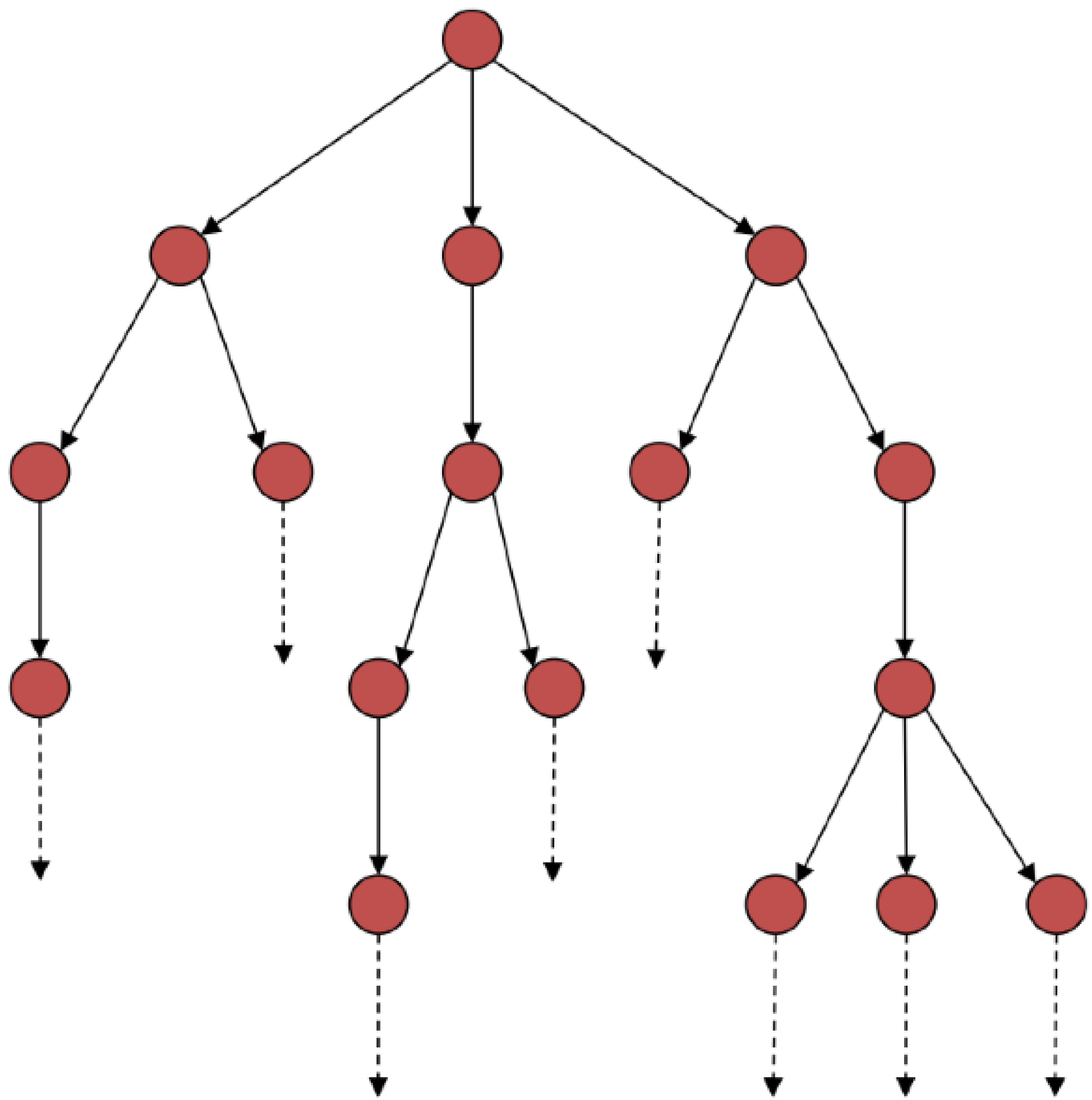
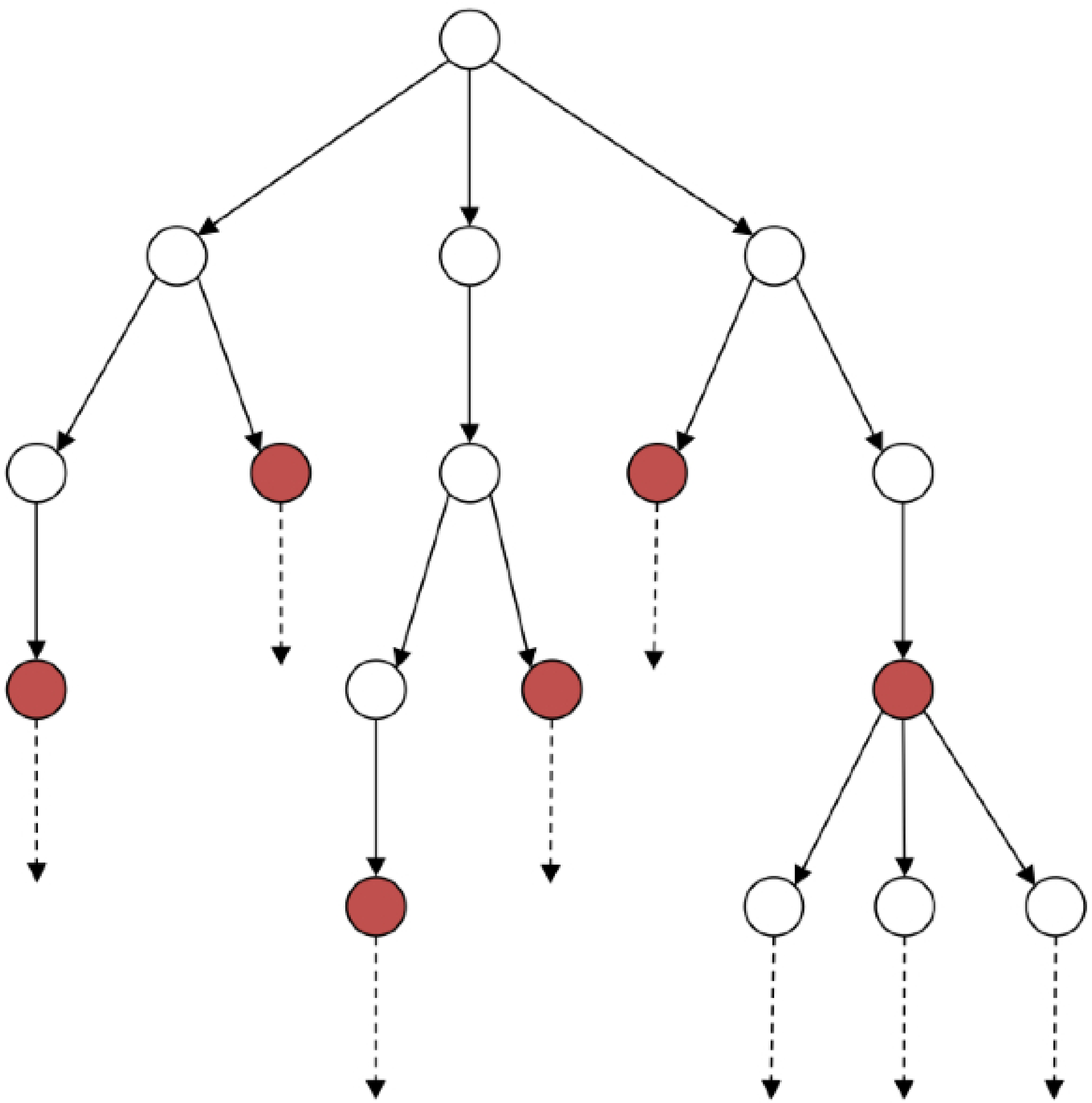
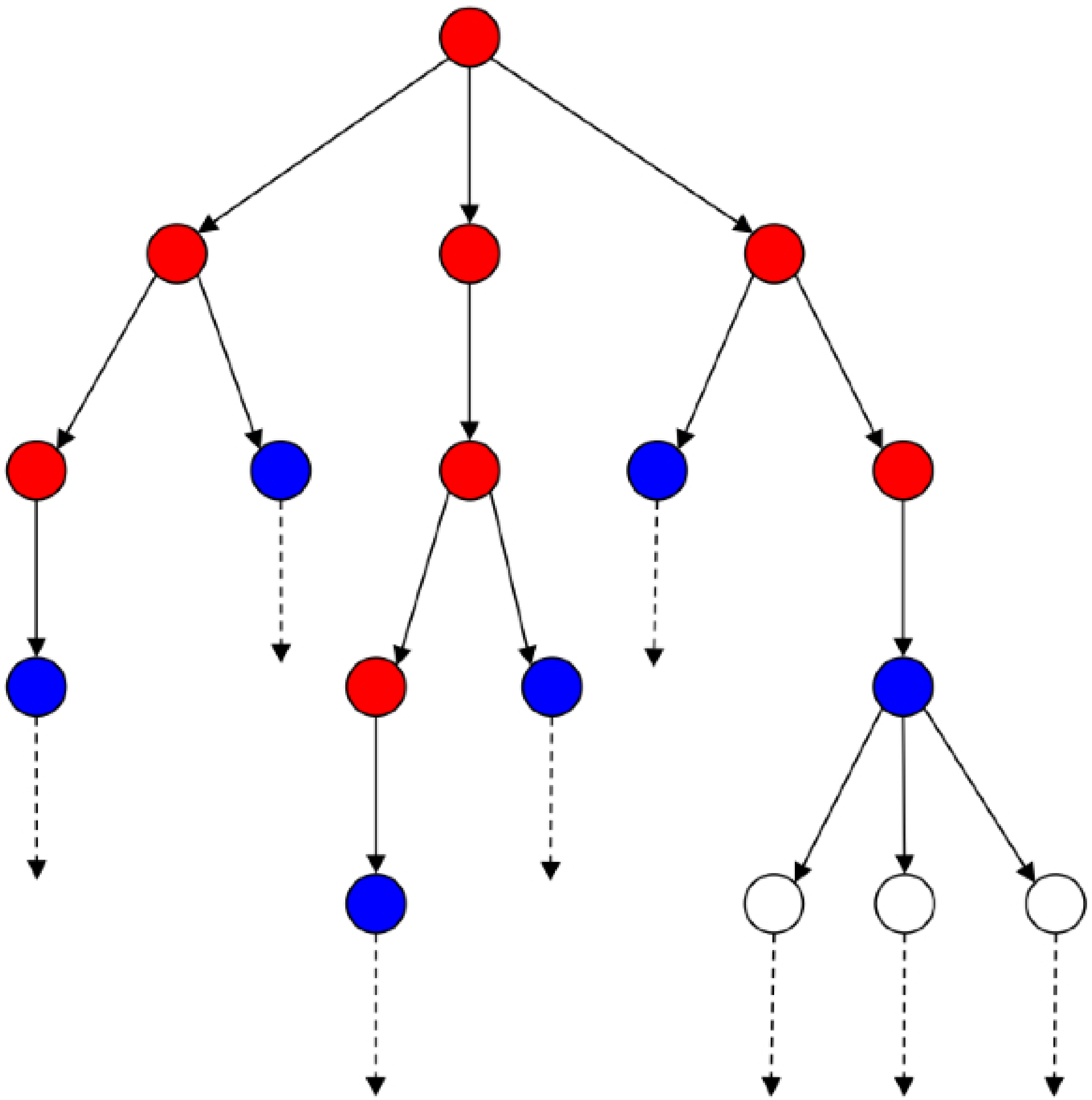


Illustration of the CTL semantics (7/8)

$$AF\, p = \neg\, EG\, \neg\, p$$

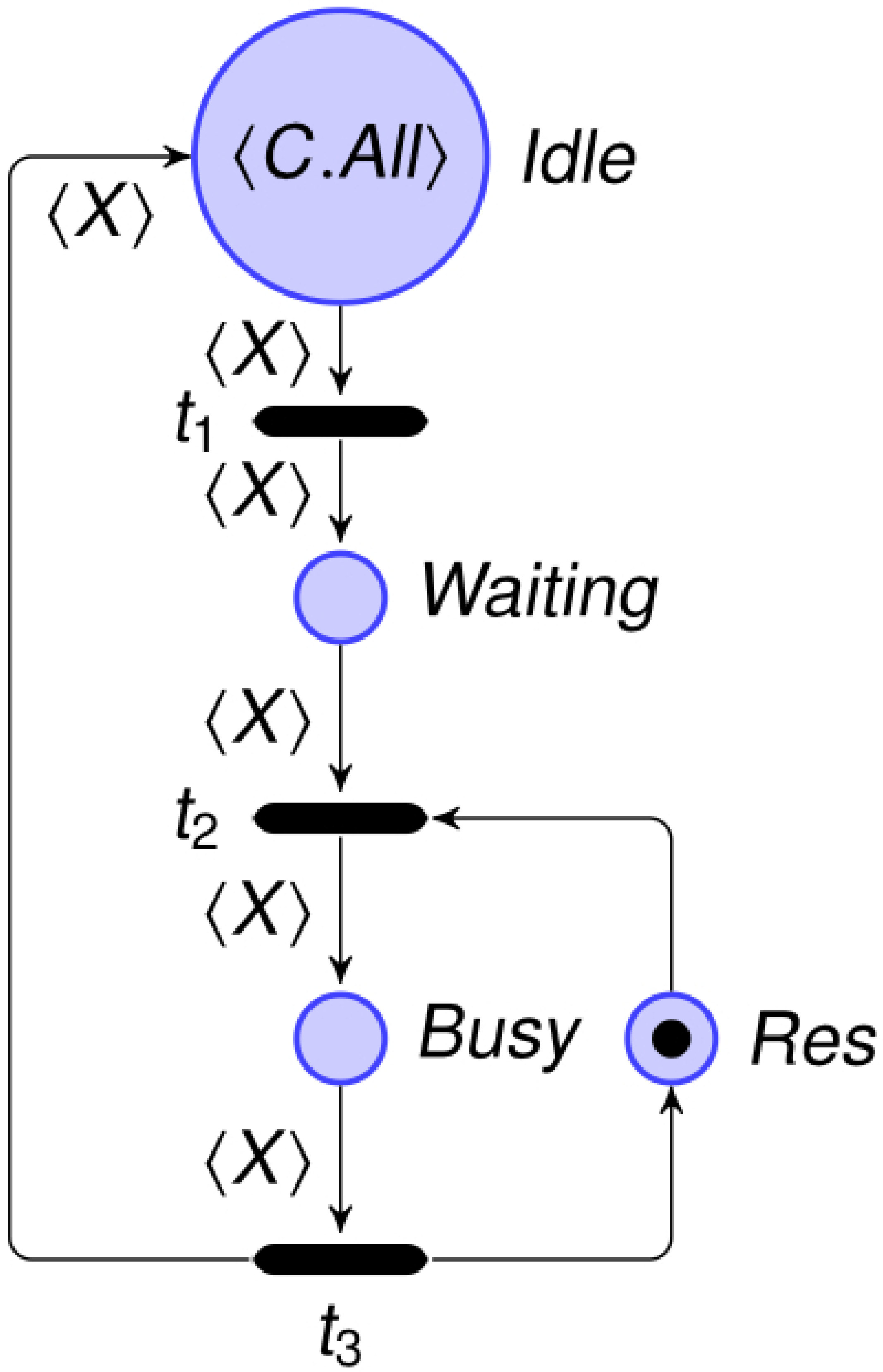


$A\, q\, U\, p$



Examples of CTL formulae

$C = \{c_1, c_2, c_3\}$



- Atomic propositions:
  - $p(c)$, where, $p \in P \setminus \{Res\}$ and $c \in C$
  - $t(c)$, where, $t \in T$ and $c \in C$
- $\mathrm{AG}\neg(Busy(c_1)\ \&\ Busy(c_2))$: it always holds that c_1 and c_2 do not appear together in critical section (place *Busy*).
- $\mathrm{AG}(Waiting(c_3) \Rightarrow \mathrm{AF}(Busy(c_3)))$: whenever c_3 requests to enter its critical section, it will eventually succeed.
- $\mathrm{AG}(\mathrm{EF}(Idle(c_1) \wedge Idle(c_2) \wedge Idle(c_3)))$: whatever is the system state, it has the possibility to return to the initial state.

Conclusion

At this stage, you know:

- Symmetric Nets with their syntax and semantics
- how to build a Reachability Graph
- how it can be used for system analysis
- LTL and CTL logics to express properties

Conclusion

At this stage, you know:

- Symmetric Nets with their syntax and semantics
- how to build a Reachability Graph
- how it can be used for system analysis
- LTL and CTL logics to express properties

Let's put into practice using the CosyVerif platform!