Petri Nets Tutorial, from Symmetric Nets to Symmetric Nets with Bags (session 1)

Souheib Baarir, Fabrice Kordon, Laure Petrucci

Souheib.Baarir@lrde.epita.fr
Fabrice.Kordon@lip6.fr
Laure.Petrucci@lipn.univ-paris13.fr

LRDE, Epita
LIP6, Université Pierre & Marie Curie
LIPN, Université Paris 13

June 23th, 2015

Thanks

Thanks for their support to...

project ImpRo/ANR-2010-BLAN-0317
project NEOPPOD (FEDER Île-de-France/System@tic-free software)
Inria for an engineer
LIP6/Move, LIPN, LDRE, LSV

and of course...

All the developers of ***CosyVerif*** (tools and platform)

Outline

- Introduction
  - Petri net classes.
  - Symmetric Nets (SN).
- Syntax and semantics of SN
  - Syntax.
  - Semantics (Firing rule).
  - Reachability Graph construction.
- SN and the verification of distributed systems
  - Atomic propositions.
  - Reachability properties.
  - Temporal properties.

Introduction

First of all...

You know about Place/Transition Petri nets:

- about their structure
- about their marking
- about their reachability graph

First of all...

You know about Place/Transition Petri nets:

- about their structure
- about their marking
- about their reachability graph

Nice notation to model and analyse distributed systems. . .

. . . but . . .

. . . complex to handle

- need for parametrisation
- need to represent data
- need for a compact and easy-to-read notation

First of all...

You know about Place/Transition Petri nets:

- about their structure
- about their marking
- about their reachability graph

Nice notation to model and analyse distributed systems. . .

. . . but . . .

. . . complex to handle

- need for parametrisation
- need to represent data
- need for a compact and easy-to-read notation

Let's have a deeper look on this now

Petri net classes.[1]

- Level 1: PNs characterised by places which can **represent boolean values**,
  - i.e. a place is marked by at most *one unstructured token*,
  - Condition/Event (C/E) Systems, Elementary Net (EN) Systems, 1-safe Systems.

[1] Made by *Monika Trompedeller* in 1995 (based on a survey by *L. Bernardinello and F. De Cindio* from 1992)

Petri net classes.[1]

- Level 1: PNs characterised by places which can **represent boolean values**,
  - i.e. a place is marked by at most *one unstructured token*,
  - Condition/Event (C/E) Systems, Elementary Net (EN) Systems, 1-safe Systems.
- Level 2: PNs characterised by places which can **represent integer values**,
  - i.e. a place is marked by *a number of unstructured tokens*,
  - Place/Transition (P/T) Nets,. . .

[1] Made by *Monika Trompedeller* in 1995 (based on a survey by *L. Bernardinello and F. De Cindio* from 1992)

Petri net classes.[1]

- Level 1: PNs characterised by places which can **represent boolean values**,
  - i.e. a place is marked by at most *one unstructured token*,
  - Condition/Event (C/E) Systems, Elementary Net (EN) Systems, 1-safe Systems.
- Level 2: PNs characterised by places which can **represent integer values**,
  - i.e. a place is marked by *a number of unstructured tokens*,
  - Place/Transition (P/T) Nets,...
- Level 3: PNs characterised by places which can **represent high-level values**,
  - i.e. a place is marked by *a multiset of structured tokens*.
  - Coloured Petri Nets, Algebraic Petri Nets, Symmetric Nets (a.k.a. Well-Formed Nets),...

[1] Made by *Monika Trompedeller* in 1995 (based on a survey by *L. Bernardinello and F. De Cindio* from 1992)

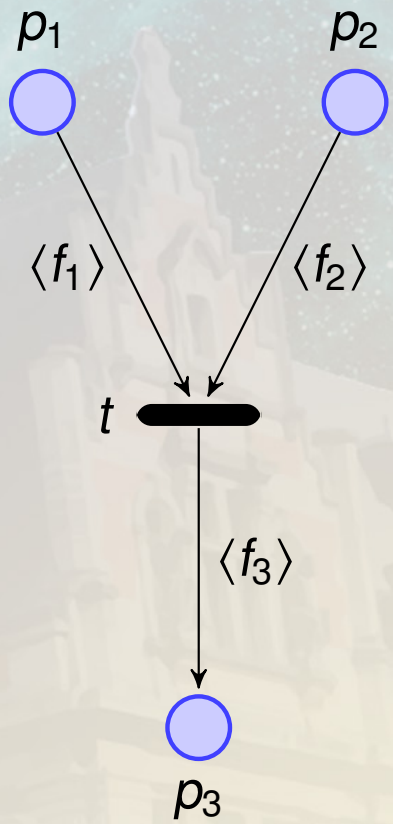
Petri net classes.[1]

- Level 1: PNs characterised by places which can **represent boolean values**,
  - i.e. a place is marked by at most *one unstructured token*,
  - Condition/Event (C/E) Systems, Elementary Net (EN) Systems, 1-safe Systems.
- Level 2: PNs characterised by places which can **represent integer values**,
  - i.e. a place is marked by *a number of unstructured tokens*,
  - Place/Transition (P/T) Nets,...
- Level 3: PNs characterised by places which can **represent high-level values**,
  - i.e. a place is marked by *a multiset of structured tokens*.
  - Coloured Petri Nets, Algebraic Petri Nets, Symmetric Nets (a.k.a. Well-Formed Nets),...

[1] Made by *Monika Trompedeller* in 1995 (based on a survey by *L. Bernardinello and F. De Cindio* from 1992)

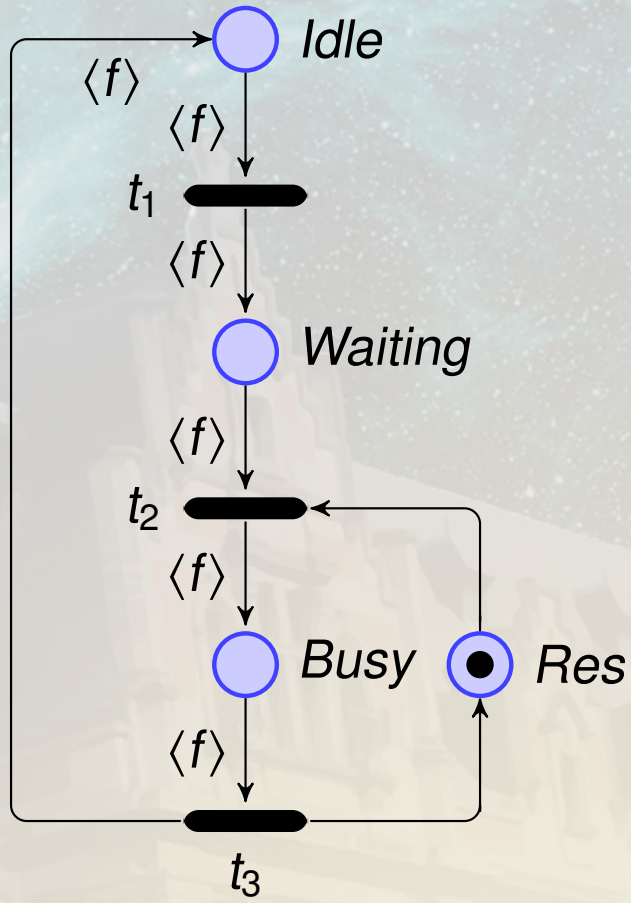
Symmetric Nets (SN): an informal definition

- Each place p is characterised by a colour domain C(p).
- A token of p is an element of C(p).
- Each transition t is characterised by a colour domain C(t).
- The colour domain of a transition characterises the signature of the transition.
- The colour functions on arcs determine the instances of tokens that are consumed and produced during the firing of a transition.



Symmetric Nets (SN): an example

- Processes of class $Cl = \{p_1, ..., p_n\}$, in mutual exclusion on a untyped resource.
- A process is either in an Idle state, or in a Waiting state, or in a Busy state.
- To move from the Waiting state to the Busy state, a process needs the resource.



C(Idle) = C(Waiting) = C(Busy) = Cl
C(Res) = $\{\epsilon\}$
C(t_1) = C(t_2) = C(t_3) = Cl

$f : Cl \rightarrow Cl$
$M_0(Idle) = Cl.All$

Symmetric Nets (SN): another example

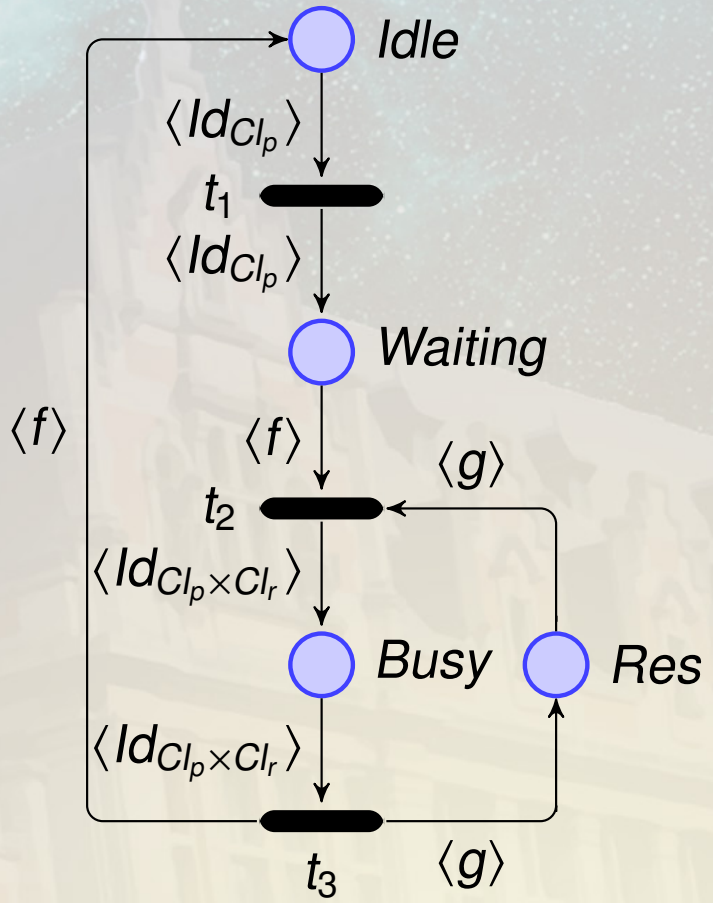
- n_1 processes of class $Cl_p = \{p_1, ..., p_{n_1}\}$, in mutual exclusion on n_2 resources of class $Cl_r = \{r_1, ..., r_{n_2}\}$.
- To move from Waiting to Busy, a process p_i needs a resource r_j.

C(Idle) = C(Waiting) = Cl_p
C(Res) = Cl_r
C(Busy) = $Cl_p \times Cl_r$

C(t_1) = Cl_p
C(t_2) = C(t_3) = $Cl_p \times Cl_r$

$f : Cl_p \times Cl_r \rightarrow Cl_p$
$g : Cl_p \times Cl_r \rightarrow Cl_r$

$M_0(Idle) = Cl_p.All$; $M_0(Res) = Cl_r.All$



Conclusion

At this stage:

- you have an idea on Coloured Nets in general . . .
- . . . and Symmetric Nets in particular

Conclusion

At this stage:

- you have an idea on Coloured Nets in general . . .
- . . . and Symmetric Nets in particular

Let's go for a more precise semantics (next sequence)

Syntax and semantics of SN

Introduction

You now:

- have an idea on Coloured Nets in general . . .
- . . . and Symmetric Nets in particular

Introduction

You now:

- have an idea on Coloured Nets in general . . .
- . . . and Symmetric Nets in particular

Let's go for a more precise semantics

Recall: multisets (Bags)

- Let $\mathtt{A}$ be a non empty finite set.
- A bag $\mathtt{a}$ on $\mathtt{A}$ is a function:

$$\begin{aligned} \mathtt{a} : \mathtt{A} &\to \mathbb{N} \\ \mathtt{x} &\to \mathtt{a(x)} \end{aligned}$$

  $\mathtt{a(x)}$ denotes the number of occurrences of x in a.
- We note: $a = \sum_{x \in A} a(x).x$
- $\mathtt{Bag(A)}$ denotes the set of multisets on $\mathtt{A}$.
- Consider the following functions:

$$\begin{aligned} f &: Bag(C_1) \to Bag(C_2) \\ g &: Bag(C_1') \to Bag(C_2') \\ h &: Bag(C) \to Bag(C_1) \end{aligned}$$

  then

$$\begin{aligned} \langle f, g \rangle : Bag(C_1) \times Bag(C_1') &\to Bag(C_2) \times Bag(C_2') \\ (x, y) &\to \langle f(x), g(y) \rangle \end{aligned}$$

  and

$$\begin{aligned} f \circ h : Bag(C) &\to Bag(C_2) \\ x &\to f(h(x)) \end{aligned}$$

SN definition: syntax

- **A Symmetric Net** is a tuple $\langle P, T, C, W^-, W^+, M_0 \rangle$ where:
  - P is the set of places, T is the set transitions ($P \cap T = \emptyset$, $P \cup T \neq \emptyset$).
  - C defines for each place and transition a colour domain.
  - W^- (= Pre) (resp. W^+ = Post), indexed on $P \times T$, is backward (resp. forward) incidence matrix of the net.
  - $W^-(p,t)$ and $W^+(p,t)$ are linear colour functions defined from $C(t)$ to $Bag(C(p))$.
  - M_0 is the initial marking of the net such that $M_0(p) \in Bag(C(p))$.
  - Transitions may be guarded by functions: $C(t) \to \{0,1\}$.
  - Colour domains are generally cartesian products.

SN definition: semantics

- Let $N = \langle P, T, C, W^-, W^+, M_0 \rangle$ be a SN:
  - A **marking** M of N is a vector: $M(p) \in Bag(C(p))$.
  - A transition t is **enabled** for an instance $c_t \in C(t)$ and a marking M *iff*:
    - ⋆ either t is not guarded, or the guard evaluates to true (for c_t), and
    - ⋆ $\forall p \in P, M(p) \geq W^-(p,t)(c_t)$
  - M', the marking reached after the firing of t for an instance c_t, from the marking M is defined by:

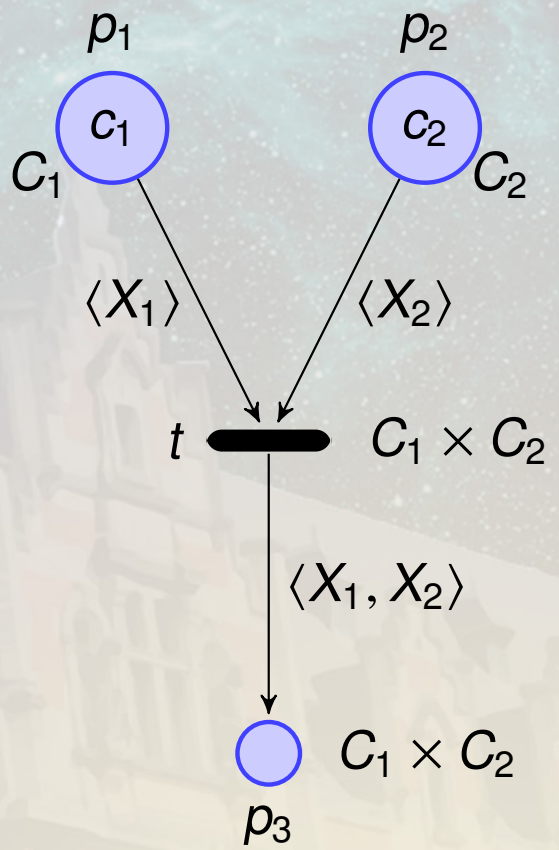
$$\forall p \in P, M'(p) = M(p) - W^-(p,t)(c_t) + W^+(p,t)(c_t)$$

We note:

$$M[(t,c_t)\rangle M' \text{ or } M \xrightarrow{(t,c_t)} M'$$

Example of firing in SN (1/2)

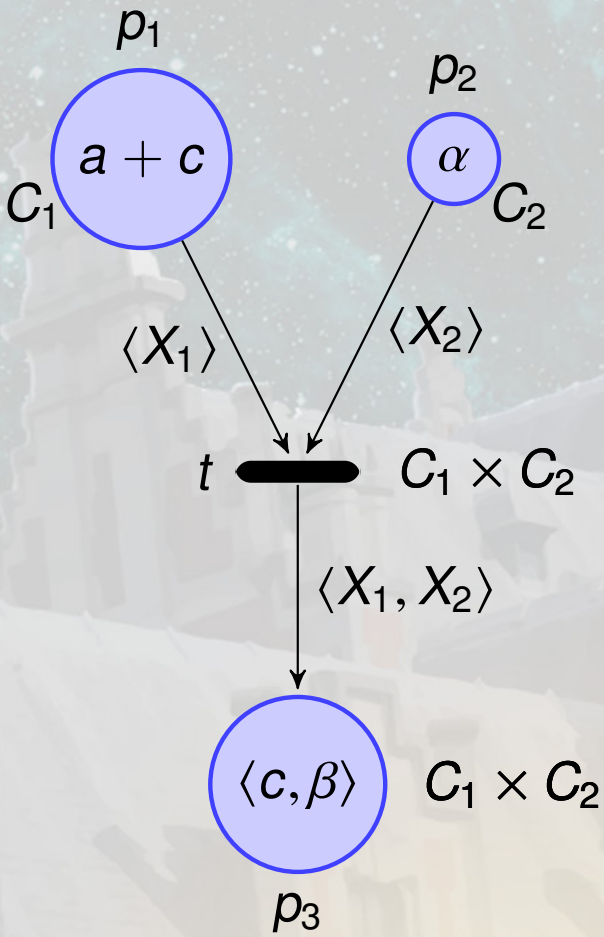
- Let $c_1 \in C1, c_2 \in C_2$ and $m_0 = p_1(c_1) + p_2(c_2)$
- t is **enabled** for (c_1, c_2) *iff*:
  1. p_1 is marked by a token of colour $\langle X_1 \rangle$
  2. p_2 is marked by a token of colour $\langle X_2 \rangle$
- If t is **fired** for (c_1, c_2) then:
  1. A token of colour $\langle X_1 \rangle$ is removed from p_1
  2. A token of colour $\langle X_2 \rangle$ is removed from p_2
  3. A token of colour $\langle c_1, c_2 \rangle$ is produced in $p_3 : \langle X_1, X_2 \rangle(c_1, c_2) = \langle c_1, c_2 \rangle$



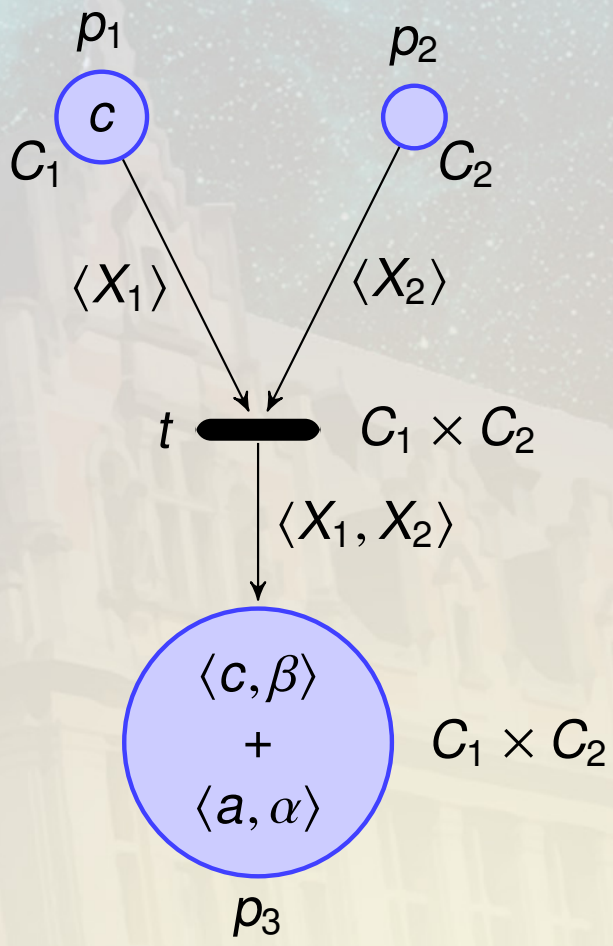
$$X_i : C_1 \times C_2 \to C_i$$
$$X_i(c_1, c_2) = c_i$$

Example of firing in SN (2/2)

$C_1 = \{a, b, c\} \quad C_2 = \{\alpha, \beta\}$



$(t, \langle a, \alpha \rangle)$



$m = p_1(a + c) + p_2(\alpha) + p_3(\langle c, \beta \rangle)$

$m' = p_1(c) + p_3(\langle c, \beta \rangle + \langle a, \alpha \rangle)$

Basic colour functions

- Let:
  - $C = \prod_{i=1}^{n} \prod_{j=1}^{e_i} C_i$, and
  - $c = \langle c_1^1, c_1^2, ..., c_1^{e_1}, ..., c_n^1, c_n^2, ..., c_n^{e_n} \rangle \in C$

Basic colour functions

A colour domain constructed on top of a Cartesian product of colour classes, in which C_i appears e_i times.

- Let:
  - $C = \prod_{i=1}^{n} \prod_{j=1}^{e_i} C_i$, and
  - $c = \langle c_1^1, c_1^2, ..., c_1^{e_1}, ..., c_n^1, c_n^2, ..., c_n^{e_n} \rangle \in C$

Basic colour functions

- Let:
  - $C = \prod_{i=1}^{n} \prod_{j=1}^{e_i} C_i$, and
  - $c = \langle c_1^1, c_1^2, ..., c_1^{e_1}, ..., c_n^1, c_n^2, ..., c_n^{e_n} \rangle \in C$
- **Identity/Projection**:
  - noted by a variable $\langle X \rangle$, Y, or $\langle X_1 \rangle$, or X_1^1, or p, q,...
  - $X_i^j(c) = c_i^j$ (e.g. $q(\langle p, q, r \rangle) = q$).

Basic colour functions

- Let:
  - $C = \prod_{i=1}^{n} \prod_{j=1}^{e_i} C_i$, and
  - $c = \langle c_1^1, c_1^2, ..., c_1^{e_1}, ..., c_n^1, c_n^2, ..., c_n^{e_n} \rangle \in C$
- **Identity/Projection**:
  - noted by a variable $\langle X \rangle$, Y, or $\langle X_1 \rangle$, or X_1^1, or p, q,...
  - $X_i^j(c) = c_i^j$ (e.g. $q(\langle p, q, r \rangle) = q$).
- **Successor (on a circularly ordered C_i)**:
  - noted $X_i{+}{+}$ or $(X_i \oplus 1)$ or $X_i!$
  - $X_i^j{+}{+}(c) = successor(c_i^j)$.

Basic colour functions

- Let:
  - $C = \prod_{i=1}^{n} \prod_{j=1}^{e_i} C_i$, and
  - $c = \langle c_1^1, c_1^2, ..., c_1^{e_1}, ..., c_n^1, c_n^2, ..., c_n^{e_n} \rangle \in C$
- **Identity/Projection**:
  - noted by a variable $\langle X \rangle$, Y or $\langle X_i \rangle$ or X^1 or p or q
  - $X_i^j(c) = c_i^j$ (e.g. $q(\langle p, q, r \rangle) = q$).
- **Successor (on a circularly ordered C_i)**:
  - noted $X_i{+}{+}$ or $(X_i \oplus 1)$ or $X_i!$
  - $X_i^j{+}{+}(c) = successor(c_i^j)$.

The successor relation is defined by the enumeration order of elements in class C_i.

Basic colour functions

- Let:
  - $C = \prod_{i=1}^{n} \prod_{j=1}^{e_i} C_i$, and
  - $c = \langle c_1^1, c_1^2, ..., c_1^{e_1}, ..., c_n^1, c_n^2, ..., c_n^{e_n} \rangle \in C$
- **Identity/Projection**:
  - noted by a variable $\langle X \rangle$, Y, or $\langle X_1 \rangle$, or X_1^1, or p, q,...
  - $X_i^j(c) = c_i^j$ (e.g. $q(\langle p, q, r \rangle) = q$).
- **Successor (on a circularly ordered C_i)**:
  - noted $X_i{+}{+}$ or $(X_i \oplus 1)$ or $X_i!$
  - $X_i^j{+}{+}(c) = successor(c_i^j)$.
- **Diffusion / Synchronisation (on C_i)**
  - noted $C_i.All$ or S_{C_i}
  - $C_i.All(c) = \sum_{x \in C_i} x$

Conclusion

At this stage:

- you know the formal syntax of Symmetric Nets
- the enabling conditions
- their firing rule

Conclusion

At this stage:

- you know the formal syntax of Symmetric Nets
- the enabling conditions
- their firing rule

Let's go for a detailed example (next sequence)

Modelling with Symmetric Nets

Introduction

You now know about Symmetric Nets:

- the formal syntax
- the enabling conditions
- their firing rule

Introduction

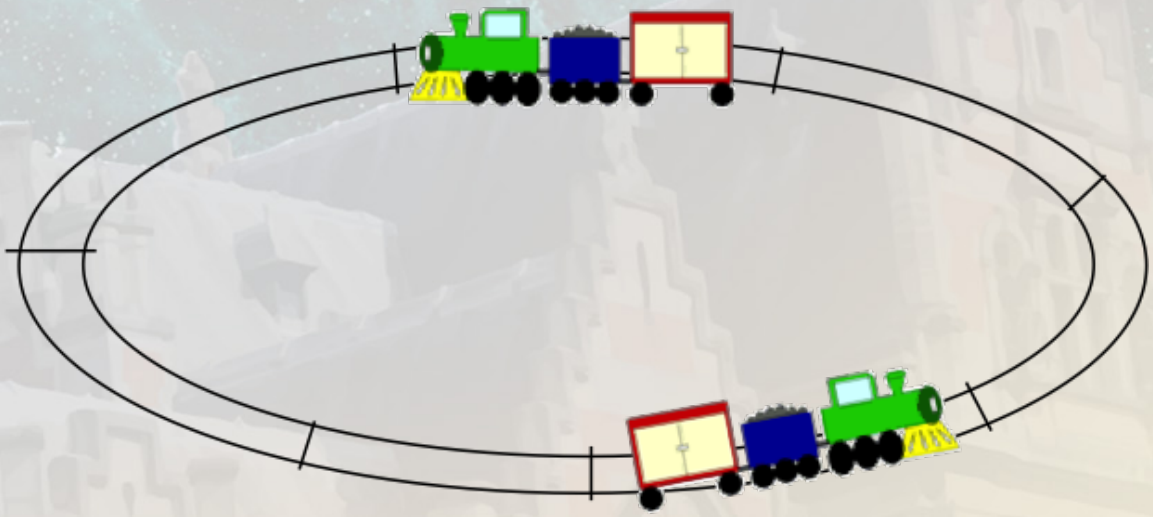
You now know about Symmetric Nets:

- the formal syntax
- the enabling conditions
- their firing rule

Let's go for a detailed example

Modelling example: the Trains Problem (1/3)

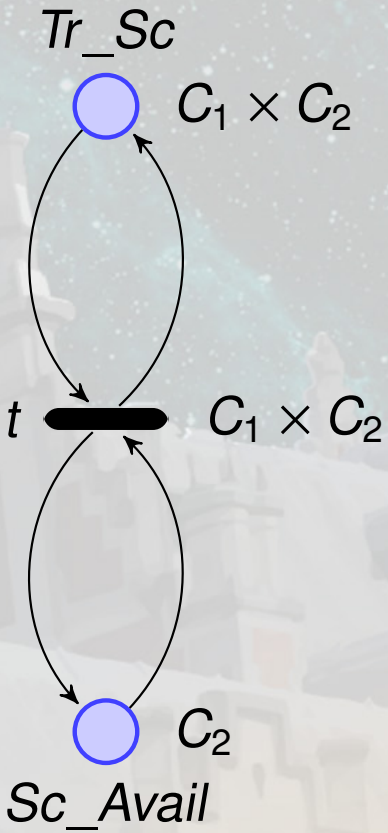
- n_1 trains distributed on a circular track, decomposed into n_2 sections.
- For security reasons, a train can enter a section only if this section and the next one are free.



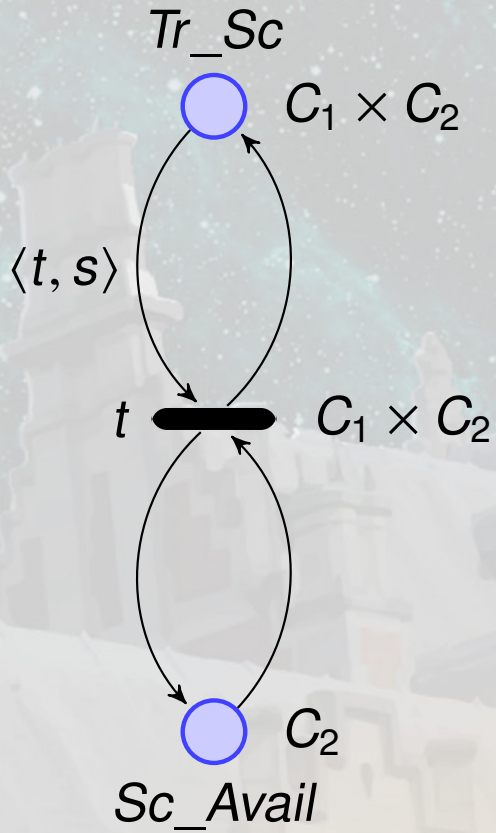
Modelling example: the Trains Problem (2/3)

- **Colour Domains:**
  - $C_1 = \{tr_1, \ldots, tr_{n_1}\}$
  - $C_2 = \{sc_1, \ldots, sc_{n_2}\}$
- **The architecture:**
  - The system state is given by a set of associations ⟨train nb , section nb⟩.
    → *place Tr_Sc*
  - A free section is a resource that allows for a train to move.
    → *place Sc_Avail*
  - A transition representing the progress of a train.

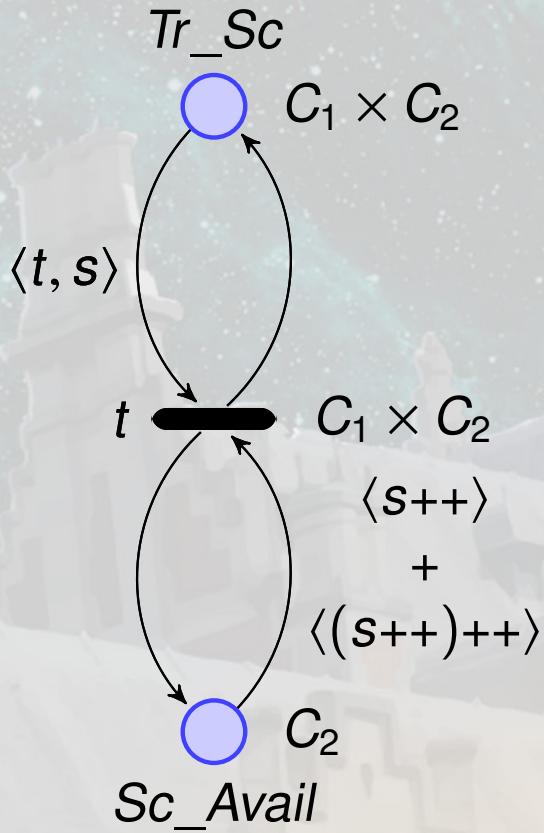
Modelling example: the Trains Problem (3/3)



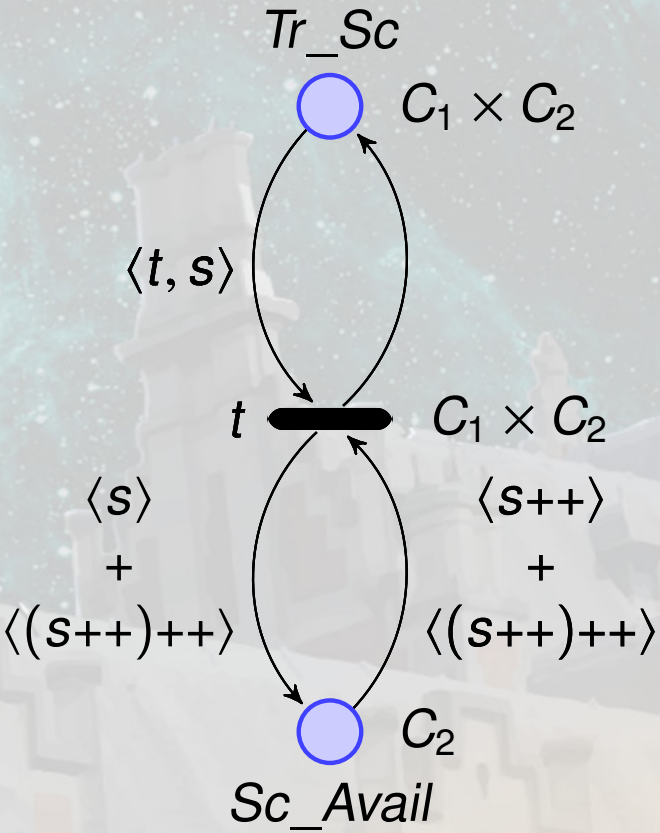
Modelling example: the Trains Problem (3/3)



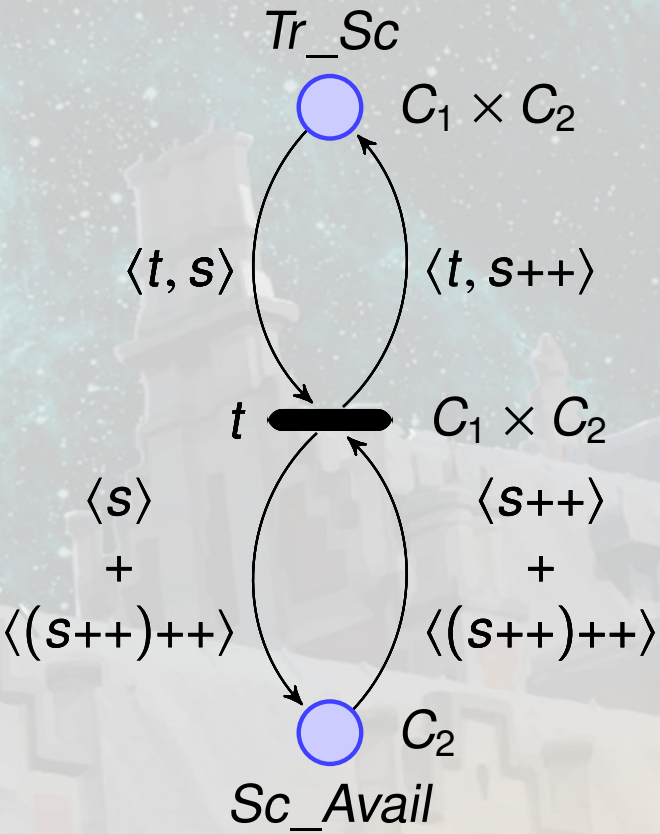
Modelling example: the Trains Problem (3/3)



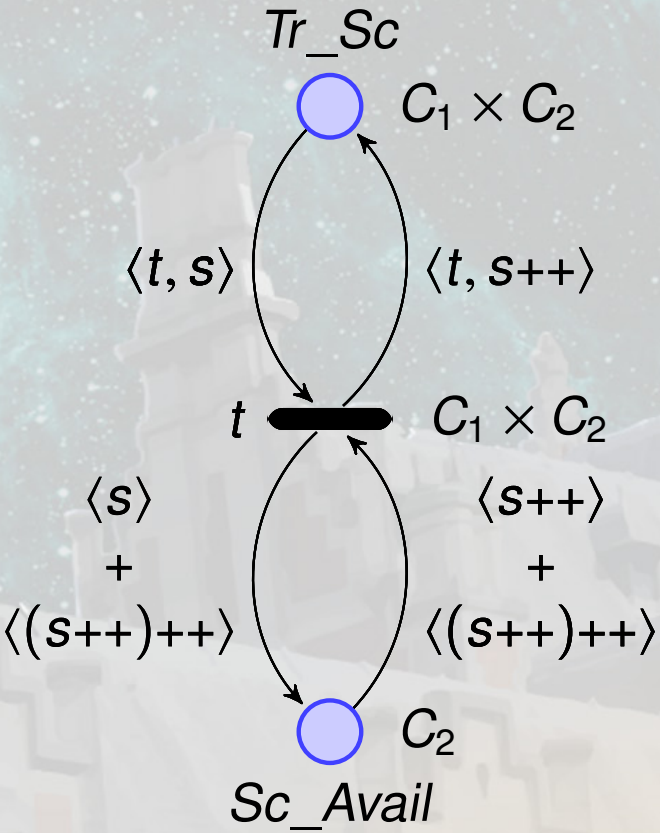
Modelling example: the Trains Problem (3/3)

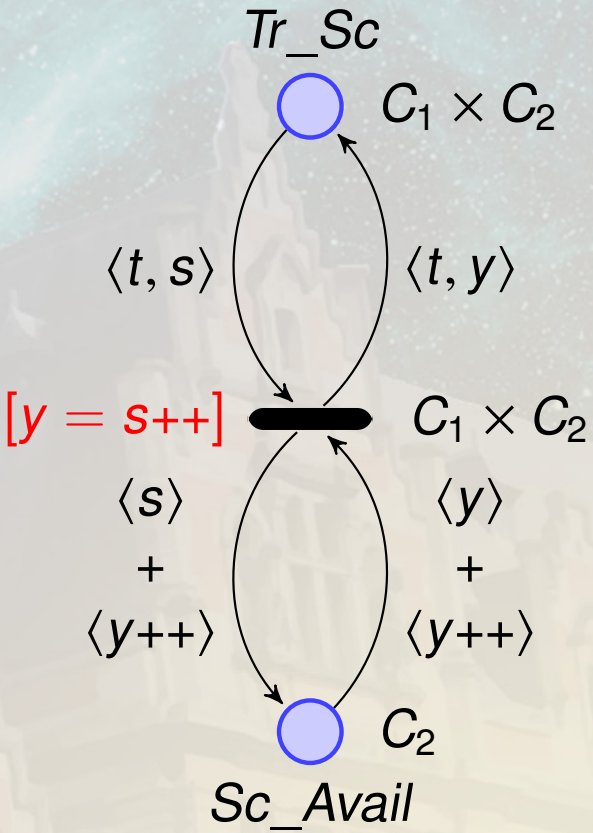


Modelling example: the Trains Problem (3/3)



Modelling example: the Trains Problem (3/3)





Conclusion

At this stage:

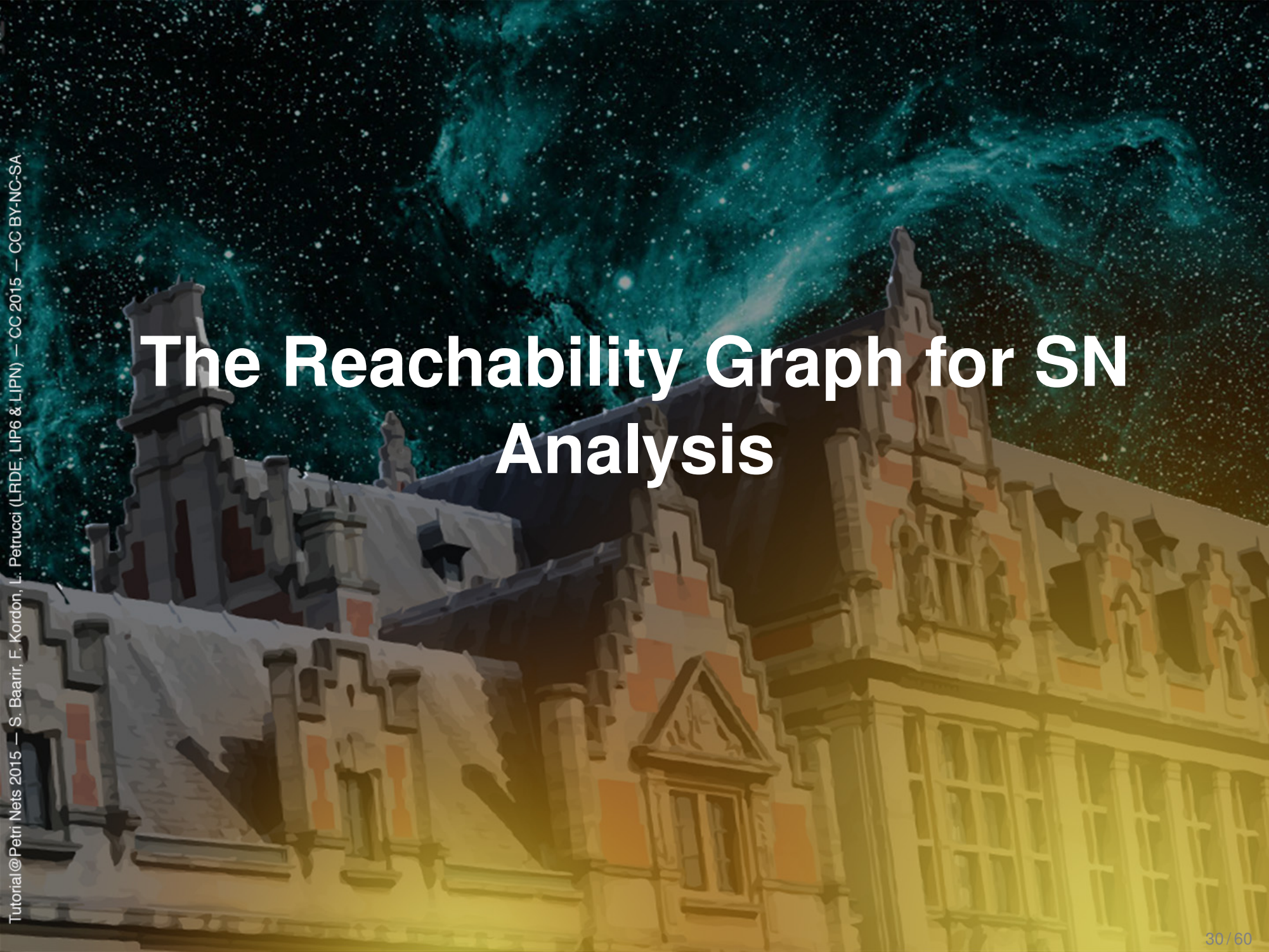
- you know how to model using Symmetric Nets
- you have seen a comprehensive detailed example

Conclusion

At this stage:

- you know how to model using Symmetric Nets
- you have seen a comprehensive detailed example

Let's see the Reachability Graph for analysis (next sequence)



The Reachability Graph for SN Analysis

Introduction

You have seen:

- how to model using Symmetric Nets
- a comprehensive detailed example

Introduction

You have seen:

- how to model using Symmetric Nets
- a comprehensive detailed example

Let's see the Reachability Graph for analysis

Analysis of SNs

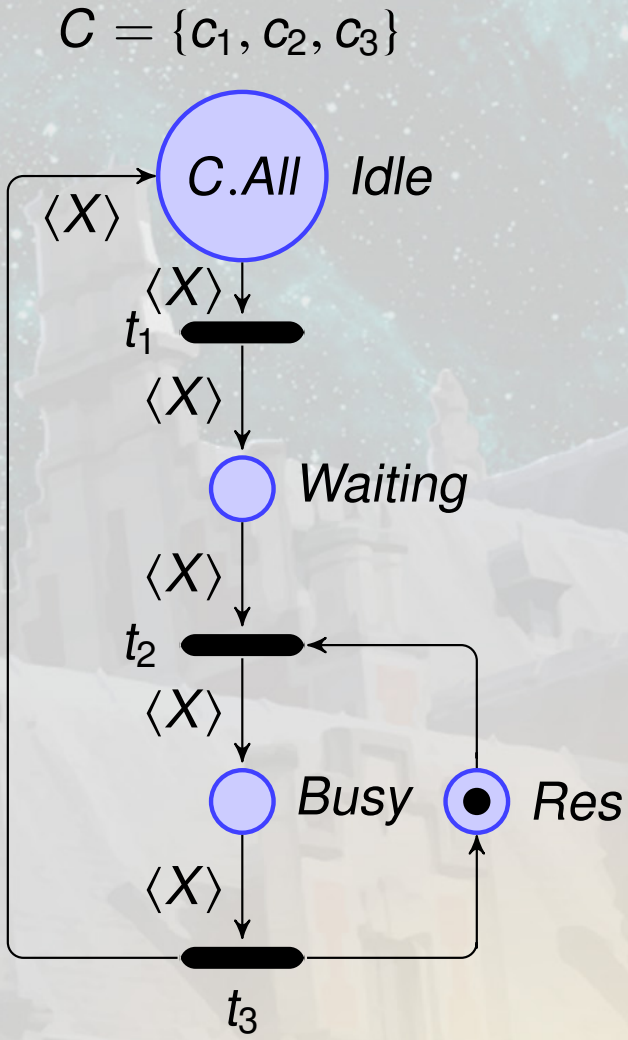
- Does a model conform to the specification?
- Possibility of answers thanks to:
  - Linear invariants.
  - The reduction theory.
  - The construction of the reachability graph (when the system is finite!)
- Try to take benefits from the structure of the model induced by the colour functions.

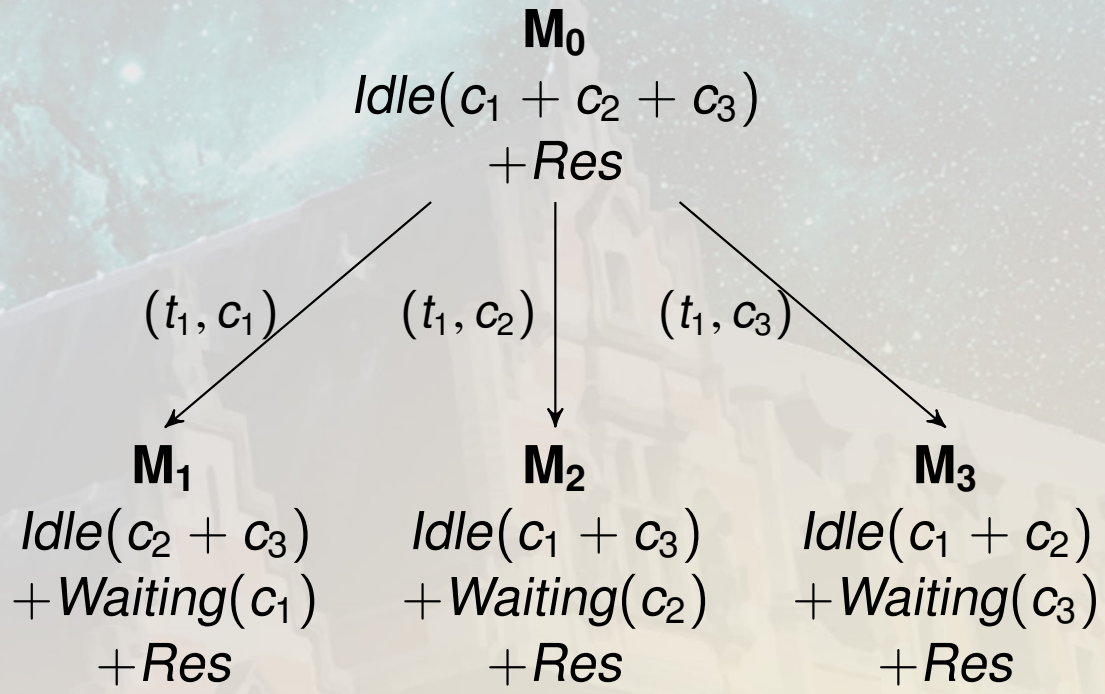
Reachability Graph (RG) Construction Algorithm

RG_Construction($N = \langle P, T, C, W^-, W^+, M_0 \rangle$)

$RG.Q = \{M_0\}$; $RG.\delta = \emptyset$;
$RG.q_0 = M_0$; $States = \{M_o\}$:
While (States $\neq \emptyset$) {
  s = pick a state in States ;
  States = States \ $\{s\}$;
  for each $t \in T, c \in C(t)$ {
    if $(s[(t,c)\rangle)$ {
      $s[(t,c)\rangle ns$;
      if $(ns \notin RG.Q)$ {
        $RG.Q = RG.Q \cup \{ns\}$;
        $States = States \cup \{ns\}$;
      }
      $RG.\delta = RG.\delta \cup \{(s, ns)\}$;
      $RG.\lambda(s, ns) = (t, c)$;
    }
  }
}
return RG;

Example of RG construction





More advanced verification of finite systems

To operate such verification, we need:

1. A representation of the behaviour of the system.

More advanced verification of finite systems

To operate such verification, we need:

1. A representation of the behaviour of the system.
   → **Kripke structure**: in our case, the RG.

More advanced verification of finite systems

To operate such verification, we need:

1. A representation of the behaviour of the system.
   → **Kripke structure**: in our case, the RG.
   → **State-based propositions:** proposition $p(c)$, where $c \in C(p)$, is true in a state of the RG *iff* place p is marked by colour c.

More advanced verification of finite systems

To operate such verification, we need:

1. A representation of the behaviour of the system.
   → **Kripke structure**: in our case, the RG.
   → **State-based propositions:** proposition $p(c)$, where $c \in C(p)$, is true in a state of the RG *iff* place p is marked by colour c.
   → **Event-based propositions:** proposition $t(c)$, where $c \in C(t)$, is true in a state of the RG *iff* transition t is enabled for colour c.

More advanced verification of finite systems

To operate such verification, we need:

1. A representation of the behaviour of the system.
   → **Kripke structure**: in our case, the RG.
   → **State-based propositions:** proposition $p(c)$, where $c \in C(p)$, is true in a state of the RG *iff* place p is marked by colour c.
   → **Event-based propositions:** proposition $t(c)$, where $c \in C(t)$, is true in a state of the RG *iff* transition t is enabled for colour c.

2. A representation of the specification (property) to be checked.

More advanced verification of finite systems

To operate such verification, we need:

1. A representation of the behaviour of the system.
   → **Kripke structure**: in our case, the RG.
   → **State-based propositions:** proposition $p(c)$, where $c \in C(p)$, is true in a state of the RG *iff* place p is marked by colour c.
   → **Event-based propositions:** proposition $t(c)$, where $c \in C(t)$, is true in a state of the RG *iff* transition t is enabled for colour c.

2. A representation of the specification (property) to be checked.
   → **Reachability properties:** e.g. for all states of RG, $p(c) \wedge !p(c_1)$; for all states of RG, there exists $t \in T, c \in C(t), t(c)$.

More advanced verification of finite systems

To operate such verification, we need:

1. A representation of the behaviour of the system.
   → **Kripke structure**: in our case, the RG.
   → **State-based propositions:** proposition $p(c)$, where $c \in C(p)$, is true in a state of the RG *iff* place p is marked by colour c.
   → **Event-based propositions:** proposition $t(c)$, where $c \in C(t)$, is true in a state of the RG *iff* transition t is enabled for colour c.

2. A representation of the specification (property) to be checked.
   → **Reachability properties:** e.g. for all states of RG, $p(c) \wedge !p(c_1)$; for all states of RG, there exists $t \in T, c \in C(t), t(c)$.
   → **Temporal properties:** two particular semantics are generally admitted:
   - the linear time semantics (an execution of the system is an infinite path in the Kripke structure);
   - the branching time semantics (the execution of the system is represented by the underlying infinite tree).

Conclusion

At this stage:

- you know how to build a Reachability Graph
- you have seen how it can be used for system analysis

Conclusion

At this stage:

- you know how to build a Reachability Graph
- you have seen how it can be used for system analysis

Let's see two logics to express properties (next two sequences)

LTL Properties

Introduction

Now you know:

- how to build a Reachability Graph
- how it can be used for system analysis

Introduction

Now you know:

- how to build a Reachability Graph
- how it can be used for system analysis

Let's see LTL logics to express properties

The linear time semantics

Logics to express linear time properties

- LTL = Linear-time Temporal Logic
- Syntax: let AP be a set of atomic propositions.
  - $a \in AP$ is an LTL formula.
  - If ϕ_1 and ϕ_2 are LTL formulae then so are
    $\neg\phi_1 \qquad \phi_1 \wedge \phi_2 \qquad \mathsf{X}\,\phi_2 \qquad \phi_1 \,\mathsf{U}\, \phi_2$
    where X stands for “next” and U for “until”
  - Two usual shortcuts: $F\phi \equiv \mathit{true}\,\mathsf{U}\,\phi$ and $G\phi \equiv \neg F \neg\phi$ where F stand for “future” and G for “globally”.

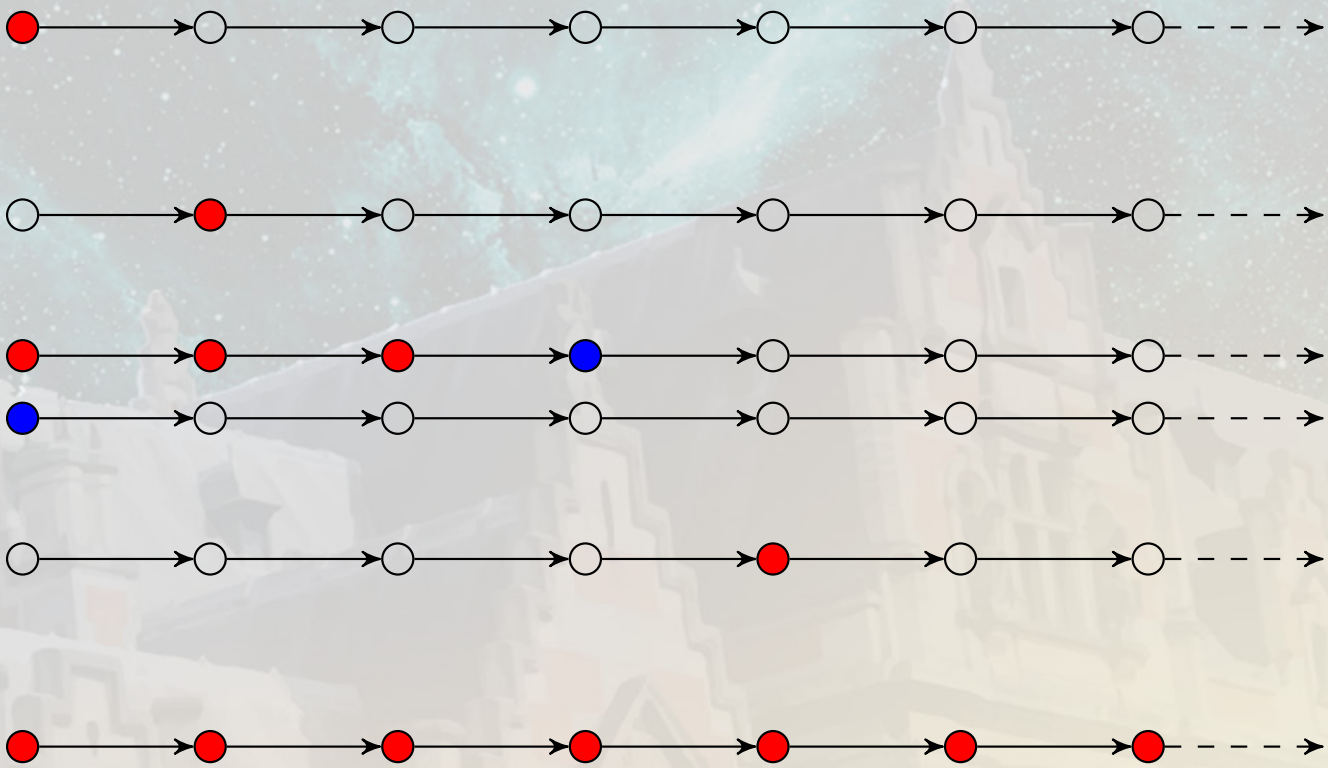
Logics to express linear time properties

- LTL = Linear-time Temporal Logic
- Syntax: let AP be a set of atomic propositions.
  - $a \in AP$ is an LTL formula.
  - If ϕ_1 and ϕ_2 are LTL formulae then so are
    $\neg\phi_1 \qquad \phi_1 \wedge \phi_2 \qquad \mathsf{X}\,\phi_2 \qquad \phi_1 \,\mathsf{U}\, \phi_2$
    where X stands for "next" and U for "until"
  - Two usual shortcuts: $F\phi \equiv \mathit{true}\,\mathsf{U}\,\phi$ and $G\phi \equiv \neg F \neg \phi$ where F stand for "future" and G for "globally".
- With each LTL formula ϕ, we associate a language $\mathcal{L}(\phi)$ of ω-words over 2^{AP} (i.e. we have $\mathcal{L}(\phi) \subseteq (2^{AP})^\omega$). Let $\sigma \in (2^{AP})^\omega$:

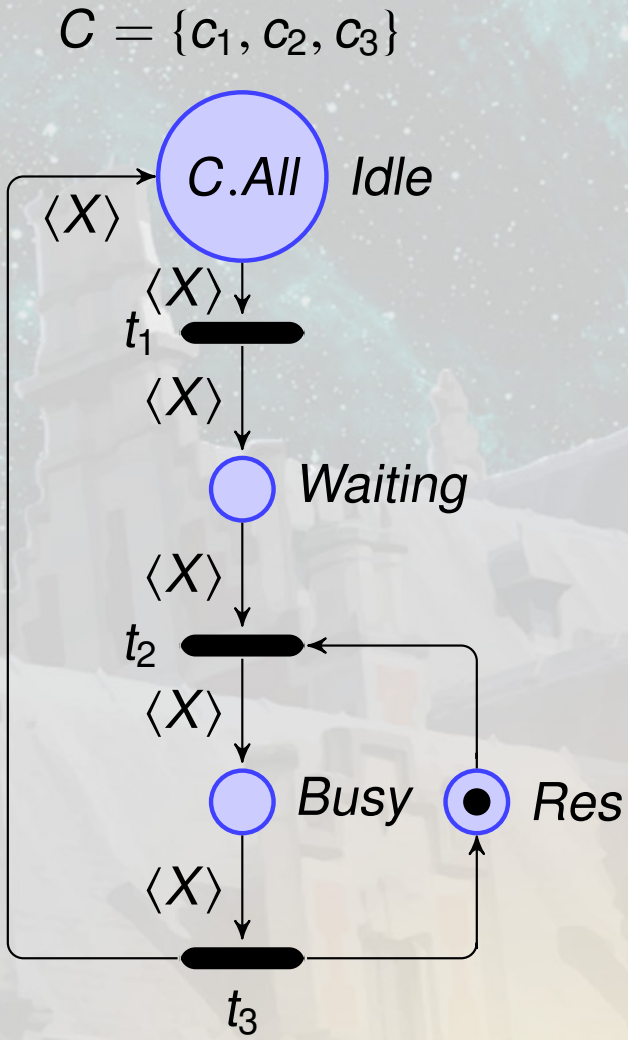
$$
\begin{aligned}
\sigma \in \mathcal{L}(a) &\Leftrightarrow a \in \sigma(0) \\
\sigma \in \mathcal{L}(\neg\phi) &\Leftrightarrow \neg\sigma \in \mathcal{L}(\phi) \\
\sigma \in \mathcal{L}(\phi_1 \wedge \phi_2) &\Leftrightarrow \sigma \in \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) \\
\sigma \in \mathcal{L}(\mathsf{X}\phi) &\Leftrightarrow \sigma(1) \in \mathcal{L}(\phi) \\
\sigma \in \mathcal{L}(\phi_1 \,\mathsf{U}\, \phi_2) &\Leftrightarrow \exists i : \sigma(i) \in \mathcal{L}(\phi_2) \wedge \forall k < i, \sigma(k) \in \mathcal{L}(\phi_1)
\end{aligned}
$$

Illustration of the LTL semantics

- a
- $\mathsf{X}\phi$
- $\phi_1 \,\mathsf{U}\, \phi_2$
- $\mathsf{F}\phi$
- $\mathsf{G}\phi$



Examples of LTL formulae



- Atomic propositions:
  - $p(c)$, where, $p \in P \setminus \{Res\}$ and $c \in C$
  - $t(c)$, where, $t \in T$ and $c \in C$
- $\mathsf{G}\neg(Busy(c_1) \wedge Busy(c_2))$: it always holds that c_1 and c_2 do not appear together in critical section (place *Busy*).
- $\mathsf{G}(Waiting(c_3) \Rightarrow \mathsf{F}(Busy(c_3)))$: whenever c_3 requests to enter its critical section, it will eventually succeed.

Conclusion

At this stage, you know:

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

Conclusion

At this stage, 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 (next sequence)

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 a CTL formula.
  - If ϕ_1 and ϕ_2 are CTL formulae then so are

    $\neg\phi_1$ $\quad$ $\phi_1 \wedge \phi_2$ $\quad$ $\mathrm{EX}\,\phi_1$ $\quad$ $\mathrm{EG}\,\phi_1$ $\quad$ $\mathrm{E}\phi_1\,\mathrm{U}\,\phi_2$

    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 a CTL formula.
  - If ϕ_1 and ϕ_2 are CTL formulae then so are

    $\neg\phi_1$ $\quad$ $\phi_1 \wedge \phi_2$ $\quad$ $\mathrm{EX}\,\phi_1$ $\quad$ $\mathrm{EG}\,\phi_1$ $\quad$ $\mathrm{E}\phi_1\,\mathrm{U}\,\phi_2$

    where X stands for “next”, G for “globally”, E for “exists” and U for “until”.
- Let $K = \langle S, I, \rightarrow, s_0 \rangle$ be a Kripke structure. With each CTL formula ϕ, we associate a set $S_k(\phi) \subseteq S$ of states, such that:

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

Logics to express branching time properties

- CTL = Computational Tree Logic.
- Syntax: let AP be a set of atomic propositions.
  - $a \in AP$ is a CTL formula.
  - If ϕ_1 and ϕ_2 are CTL formulae then so are

    $\neg\phi_1$ $\quad$ $\phi_1 \wedge \phi_2$ $\quad$ $\mathrm{EX}\,\phi_1$ $\quad$ $\mathrm{EG}\,\phi_1$ $\quad$ $\mathrm{E}\phi_1\,\mathrm{U}\,\phi_2$

    where X stands for “next”, G for “globally”, E for “exists” and U for “until”.
- Let $K = \langle S, I, \rightarrow, s_0 \rangle$ be a Kripke structure. With each CTL formula ϕ, we associate a set $S_k(\phi) \subseteq S$ of states, such that:

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

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

Illustration of the CTL semantics (1/8)

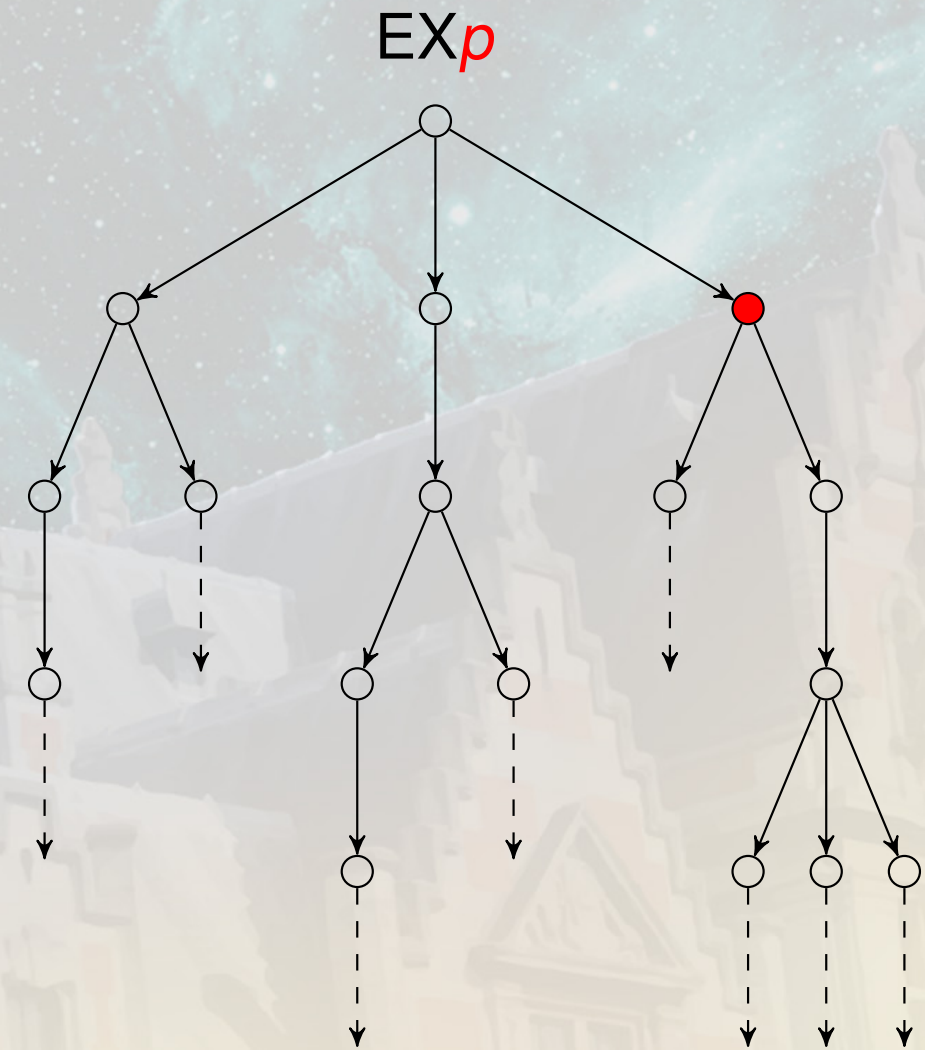


Illustration of the CTL semantics (2/8)

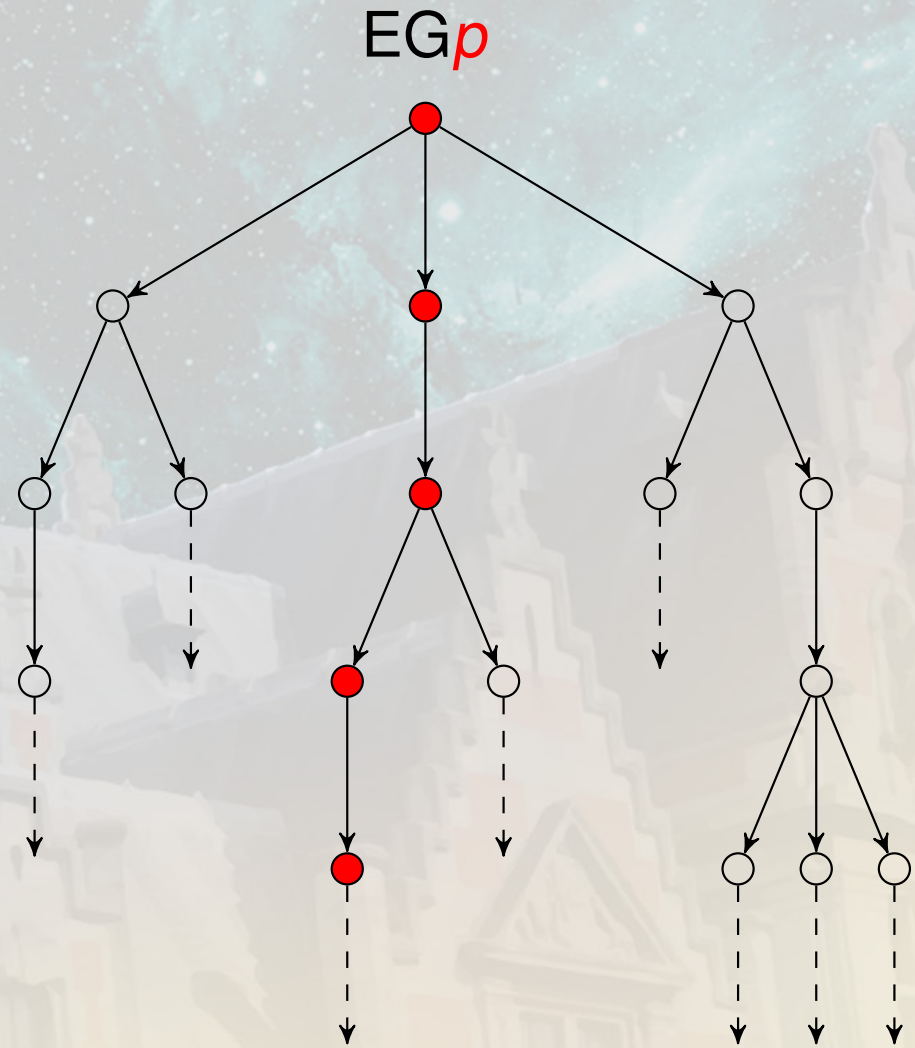


Illustration of the CTL semantics (3/8)

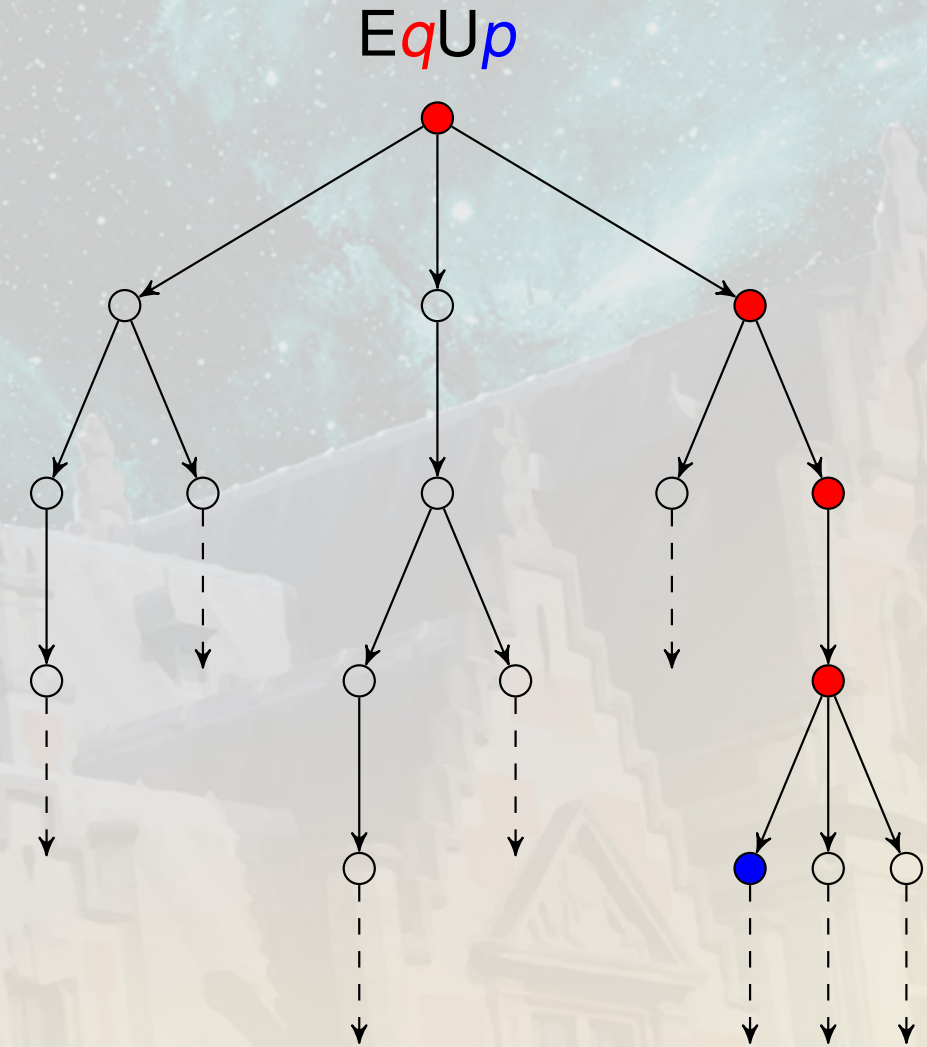


Illustration of the CTL semantics (4/8)

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

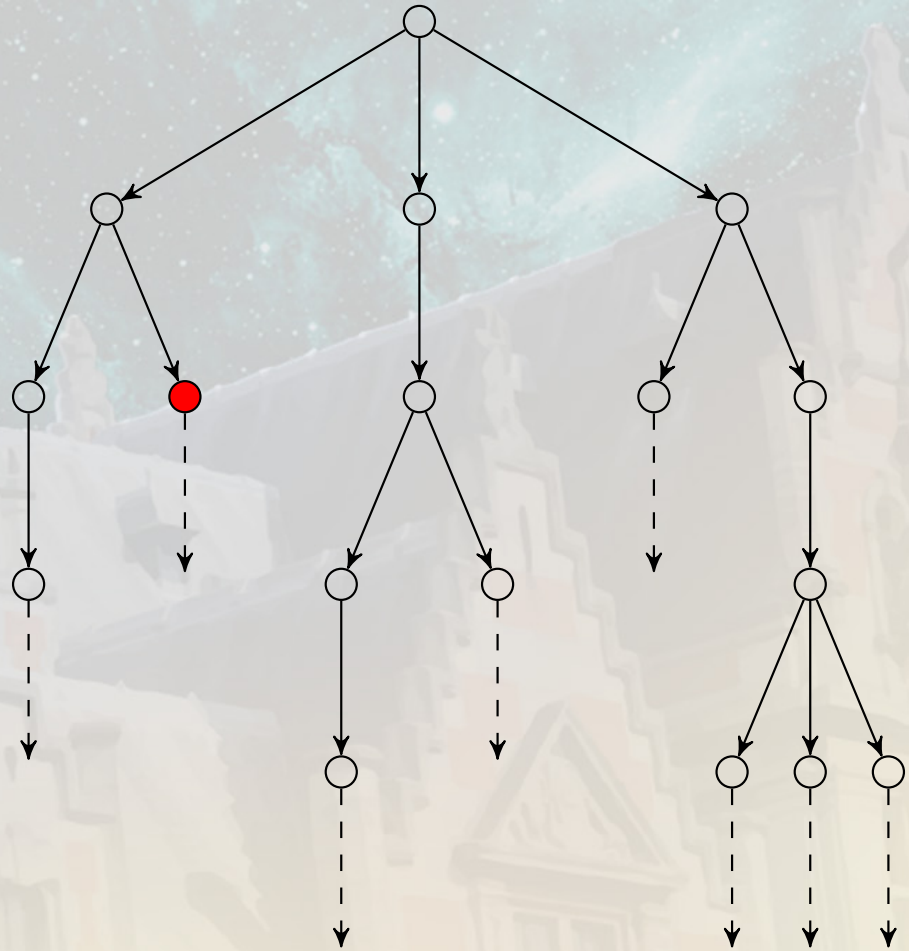


Illustration of the CTL semantics (5/8)

$$\mathrm{AX}p = \neg\mathrm{EX}\neg p$$

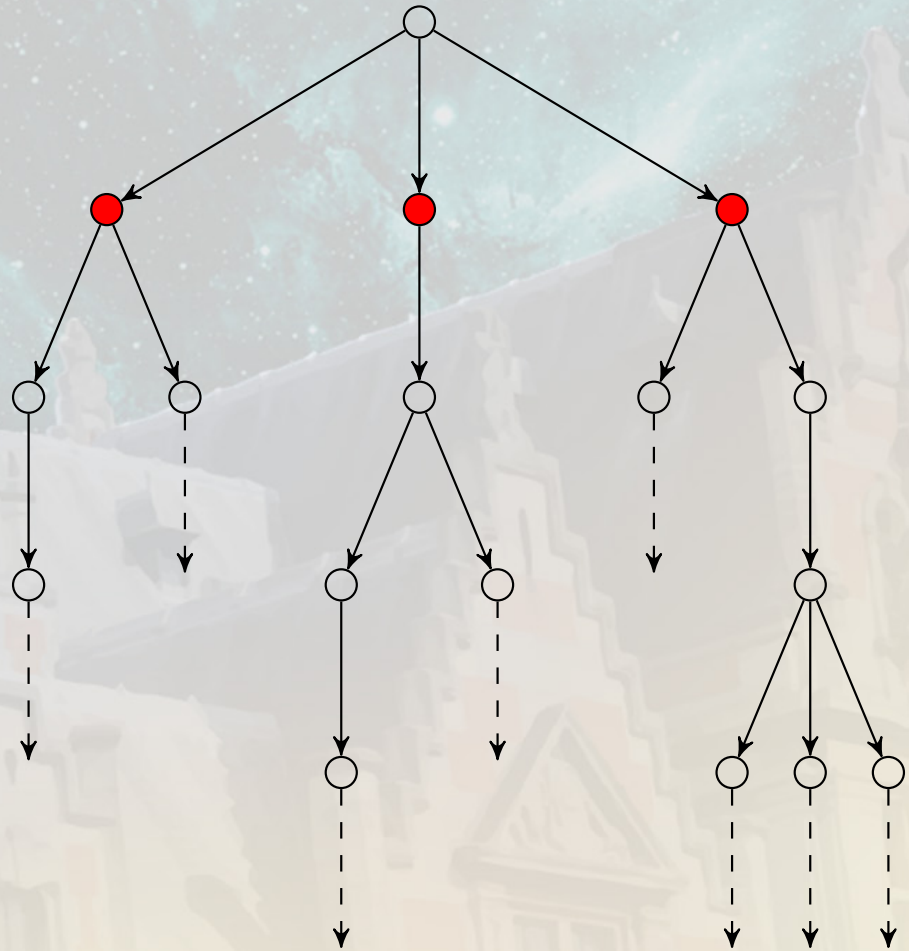


Illustration of the CTL semantics (6/8)

$$\mathsf{AG}p = \neg(\mathsf{E}\mathit{true}\,\mathsf{U}\,\neg p)$$

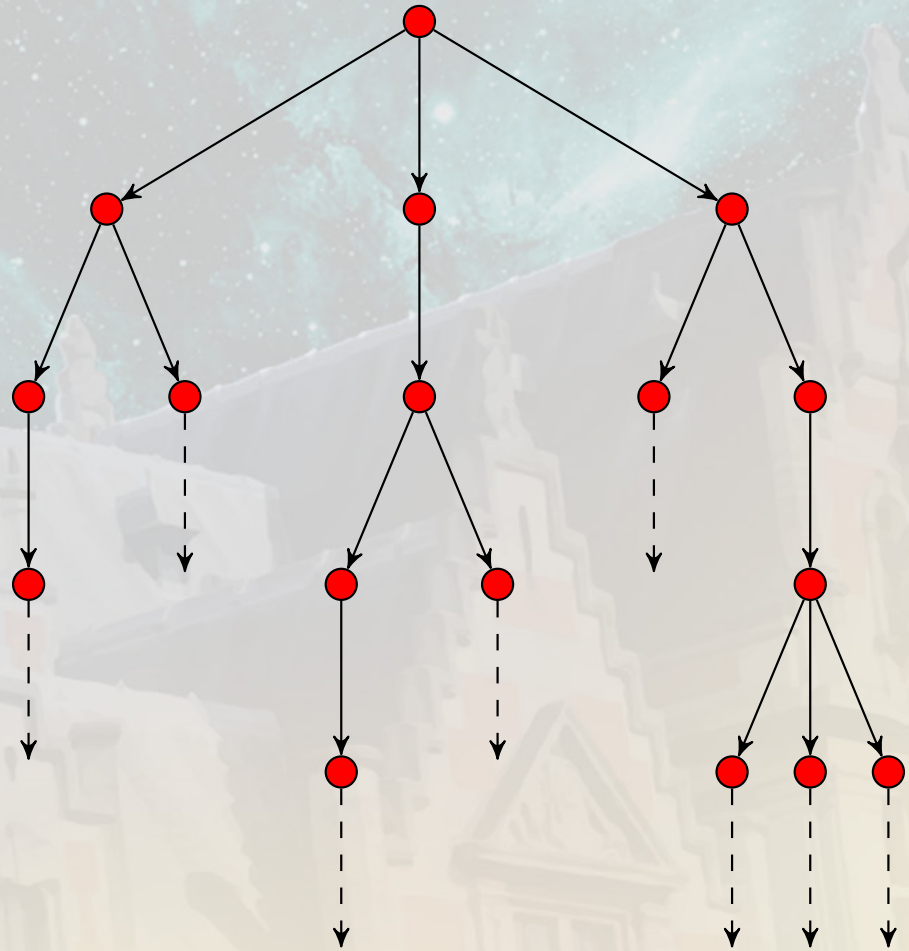


Illustration of the CTL semantics (7/8)

$\mathrm{AF}p = \neg\mathrm{EG}\neg p$

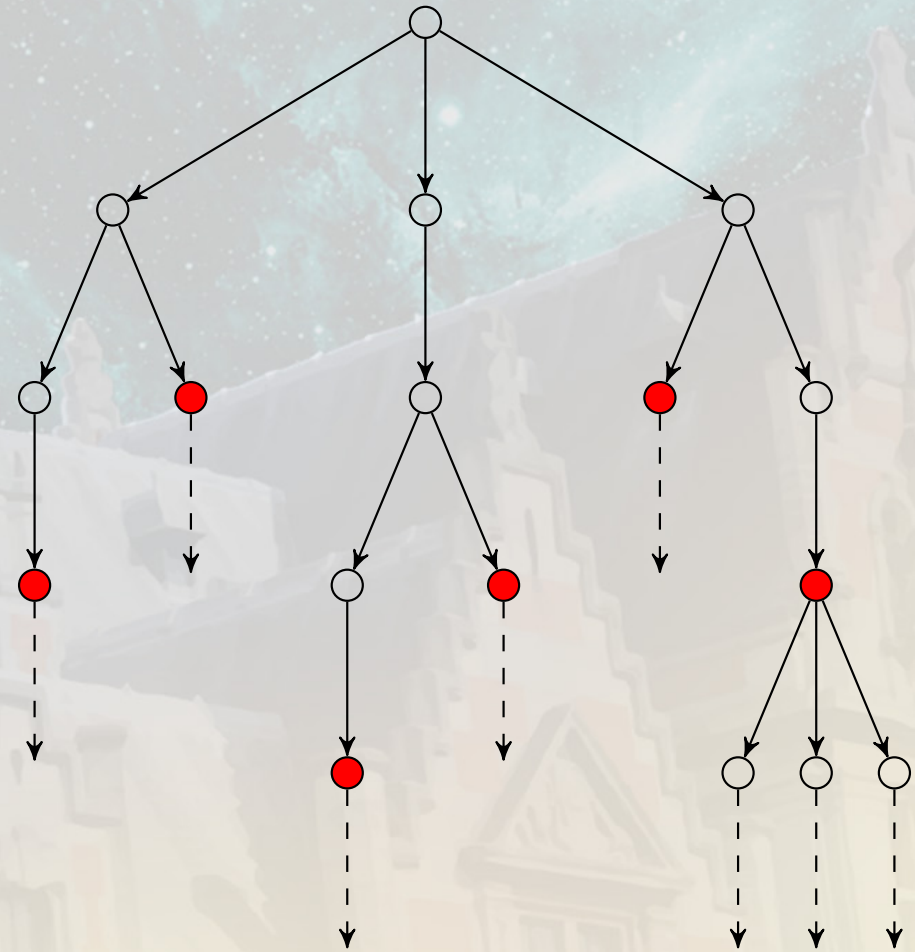
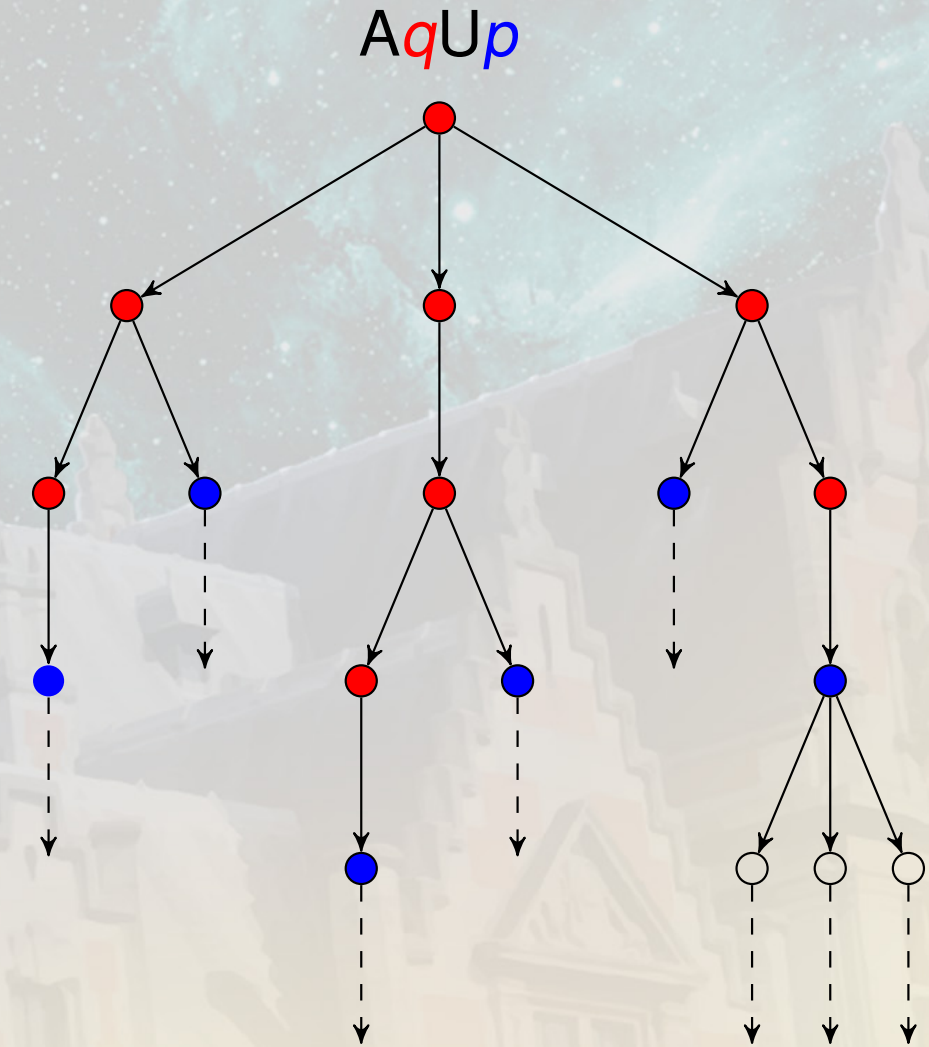
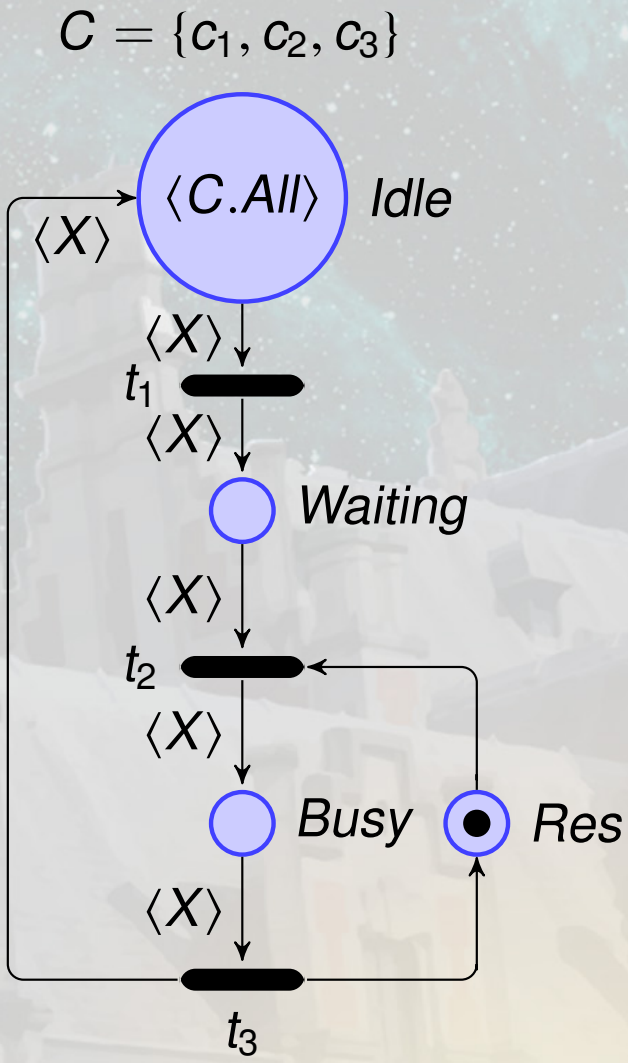


Illustration of the CTL semantics (8/8)



Examples of CTL formulae



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