Specification, Design and Verification

Kais Klai and Walid Gaaloul

Objectives

1. Object Oriented Design
   - To describe the activities in the object-oriented design process
   - To introduce various UML models that can be used to describe an object-oriented design
   - To show how to use OCL to guarantee the models' constraints

Objectives

1. Object Oriented Design
   - To describe the activities in the object-oriented design process
   - To introduce various UML models that can be used to describe an object-oriented design
   - To show how to use OCL to guarantee the models' constraints
2. Formal Modeling and Verification
   - How to model a concurrent system (using Petri nets)
   - How to express behavioral properties (LTL)
   - How to check a property on a system

Objectives

1. Object Oriented Design
   - To describe the activities in the object-oriented design process
   - To introduce various UML models that can be used to describe an object-oriented design
   - To show how to use OCL to guarantee the models' constraints
2. Formal Modeling and Verification
   - How to model a concurrent system (using Petri nets)
   - How to express behavioral properties (LTL)
   - How to check a property on a system
3. Test
   - Test of Object Oriented applications
   - Unit, Integration and Validation Test

Organisation

- 14h lecture (CM)
- 10h30 Tutorials (TP)
- 10h30 Tutorials (Project)
- Evaluation :
    - 1 exam (DE) (66.66%)
    - a project (33.33%)

Formal Specification and Verification of Concurrent Systems

Kais Klai

Maître de Conférences, LIPN
Université Paris 13 Sorbonne Paris Cité

Outline

Outline


System
Property

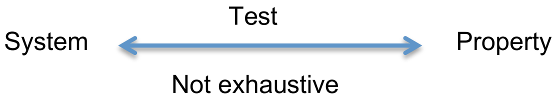
Some Properties

- Reachability: A certain situation can be reached
  x may be zero, each instruction can be executed
- Invariant: Each state respects some good property
  x is never equal to zero, an array never overflows
- Safety: Something bad can never happen
  I access the file if I enter the correct PIN
- Liveness: Something good can always happen
  the program terminate, the message will eventually arrive to the destination, the program always returns to the initial state
- Fairness: Something good happens infinitely often
  If a process asks to enter to a critical section infinitely often, it will access it infinitely often
- ...

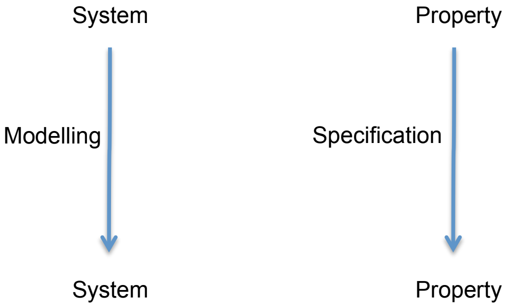
Context

Context

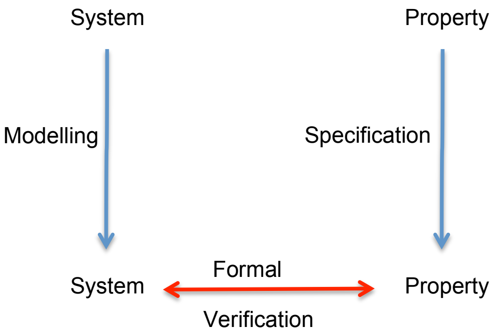
Context



Context


System
Property
Modelling
Specification
System
Property

Context


System
Property
Modelling
Specification
System
Formal
Verification
Property

Formal Verification

1. Theorem Proving
   - Logical description of the system
   - Prove properties by deduction
   - Not fully automatic

Formal Verification

1. Theorem Proving
    - Logical description of the system
    - Prove properties by deduction
    - Not fully automatic
2. Model Checking
    - Exhaustive verification
    - Fully automatic
    - Counter-examples

Example: Mutual Exclusion Algorithm

Global variables: req_P and req_Q

Process P

1. $req_P \leftarrow 1$
2. wait($req_Q = 0$)
3. Critical Section
4. $req_P \leftarrow 0$

Process Q

1. $req_Q \leftarrow 1$
2. wait($req_P = 0$)
3. Critical Section
4. $req_Q \leftarrow 0$

Initial state: $req_P = req_Q = 0$

Example: Mutual Exclusion Algorithm

Global variables: req_P and req_Q

Process P

1. $req_P \leftarrow 1$
2. `wait(`$req_Q = 0$`)`
3. `Critical Section`
4. $req_P \leftarrow 0$

Process Q

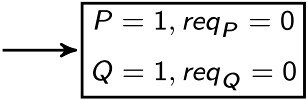
1. $req_Q \leftarrow 1$
2. `wait(`$req_P = 0$`)`
3. `Critical Section`
4. $req_Q \leftarrow 0$

Initial state: $req_P = req_Q = 0$

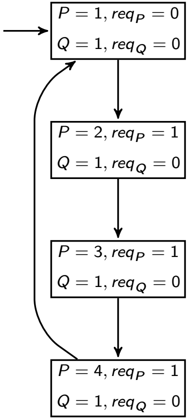
Properties to be checked:

1. Mutual exclusion
2. Fairness
3. Order

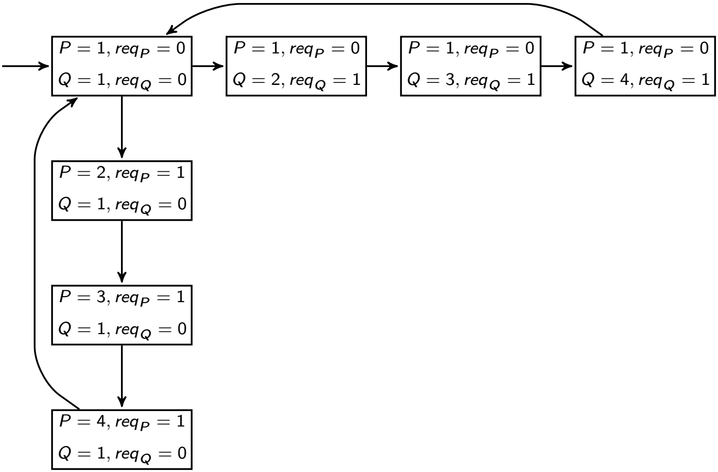
Example: Reachability State Space



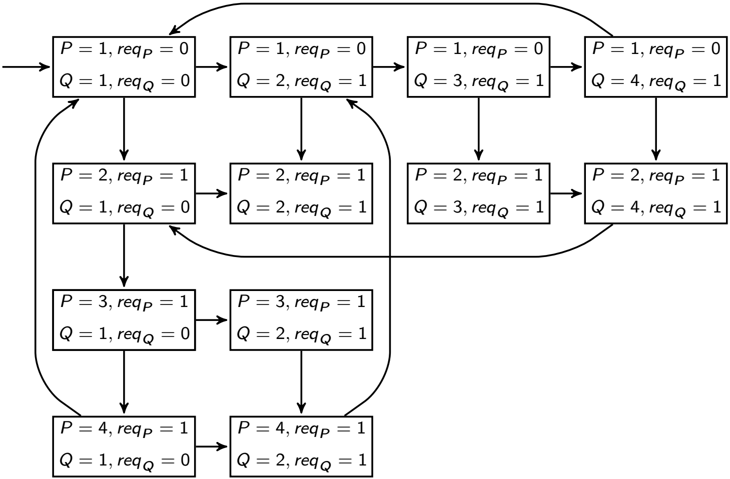
Example: Reachability State Space



Example: Reachability State Space



Example: Reachability State Space



Property 1: Mutual Exclusion

Property 1: Mutual Exclusion

We never have $P = 3 \wedge Q = 3$

Property 1: Mutual Exclusion

We never have $P = 3 \wedge Q = 3$

That's true

Property 1: Mutual Exclusion

We never have $P = 3 \wedge Q = 3$

That's true

To check this property we browse the set of reachable states. We need reachable states only, not the transitions between states.

Property 2: Fairness

Property 2: Fairness

Each path starting at a state where $P = 2$ traverses a state where $P = 3$, and the same for Q

Property 2: Fairness

Each path starting at a state where $P = 2$ traverses a state where $P = 3$, and the same for Q

That's false: State $(P = 2,\ req_P = 1,\ Q = 2,\ , req_Q = 1)$ has no successor

Property 2: Fairness

Each path starting at a state where $P = 2$ traverses a state where $P = 3$, and the same for Q

That's false: State $(P = 2,\ req_P = 1,\ Q = 2,\ , req_Q = 1)$ has no successor

To check this property we browse the reachability graph (having the reachable states only is not sufficient).

Property 3: Order

Property 3: Order

Each path starting at a state where $P = 2 \wedge Q = 1$ do not visit a state satisfying $Q = 3$ before visiting a state where $P = 3$ (+ a symmetric property for Q).

Property 3: Order

Each path starting at a state where $P = 2 \wedge Q = 1$ do not visit a state satisfying $Q = 3$ before visiting a state where $P = 3$ (+ a symmetric property for Q).

That's false: Starting from $(P = 2,\ req_P = 1,\ Q = 1,\ req_Q = 0)$, there exists a path where $P = 3$ is never satisfied.

Property 3: Order

Each path starting at a state where $P = 2 \wedge Q = 1$ do not visit a state satisfying $Q = 3$ before visiting a state where $P = 3$ (+ a symmetric property for Q).

That's false: Starting from $(P = 2,\ req_P = 1,\ Q = 1,\ req_Q = 0)$, there exists a path where $P = 3$ is never satisfied.

To check this property we browse the reachability graph (having the reachable states only is not sufficient).

Outline

Model checking of finite state systems

Principle

1. Design the system with a model $\mathcal{M}$ and design a property φ
2. $\mathcal{M} \models \varphi$? if no, a counter-example σ
3. Analyse the result:
    - If yes, **OK**
    - If no, refine $\mathcal{M}$ using σ and go to (1).

Model checking of finite state systems

Principle

1. Design the system with a model $\mathcal{M}$ and design a property φ
2. $\mathcal{M} \models \varphi$? if no, a counter-example σ
3. Analyse the result:
    - If yes, **OK**
    - If no, refine $\mathcal{M}$ using σ and go to (1).

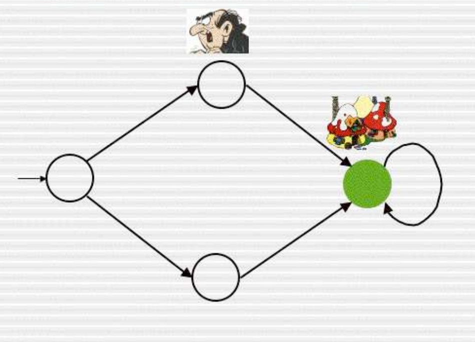
Approach

- State space traversal (Labeled Transition System)

Example

Gargamel !!!

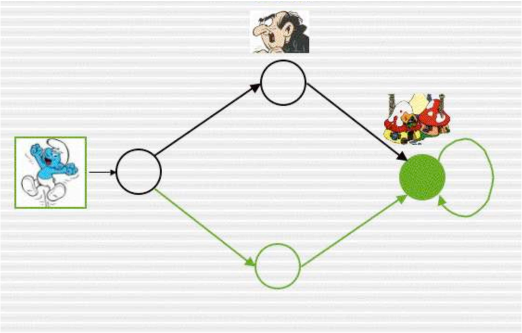
Is there any safe path?



Example

Gargamel !!!

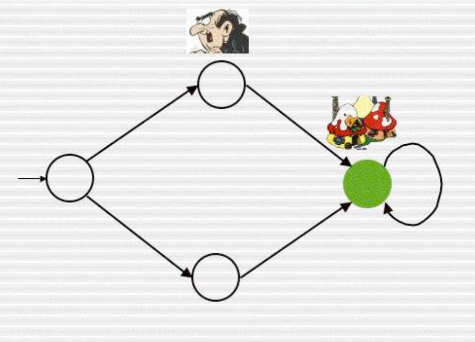
YES



Example

Gargamel !!!

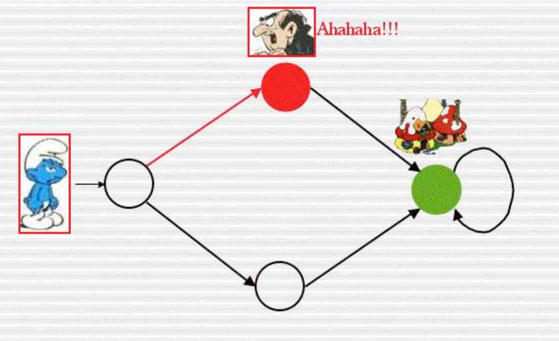
Are all the paths safe?



Example

Gargamel !!!

NO



Formal Specifications

1. The System

   Systems are formally expressed using:

   - State Machines
   - Automata
   - Petri Nets

Formal Specifications

1. The System

   Systems are formally expressed using:

   - State Machines
   - Automata
   - Petri Nets

2. The properties

   Properties are formally expressed using temporal logics

   - Linear Temporal Logic (LTL)
   - Tree Computational Logic (CTL)
   - CTL*

Formal Specifications

1. The System
   Systems are formally expressed using:
   - State Machines
   - Automata
   - Petri Nets
2. The properties
   Properties are formally expressed using temporal logics
   - Linear Temporal Logic (LTL)
   - Tree Computational Logic (CTL)
   - CTL*

Advantages:

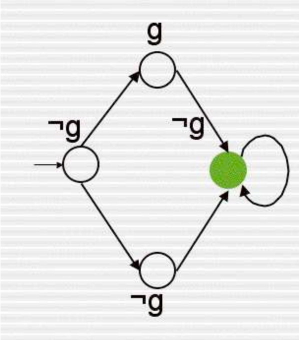
- unambiguous
- generic
- allows for automatic verification

Example

Let's be serious 5 minutes

Example

- *A*: for all paths
- *E*: there exists a path
- *G*: always
- *g*: *Gargamel*
- $\neg$: negation



The formula $EG\ \neg g$ is satisfied by the model

Model Checking

Ingredients

- $\mathcal{M} =$ The behavior of the System
- $\varphi =$ a temporal formula
- $MC = \mathcal{M} \models \varphi$?

Model Checking

Ingredients

- $\mathcal{M}$ = The behavior of the System
- φ = a temporal formula
- $MC = \mathcal{M} \models \varphi$?

Advantages

- During specification/design time
- Automatic
- Global w.r.t. Test
- Efficient (in some fields)

Model Checking

Ingredients

- $\mathcal{M}$ = The behavior of the System
- φ = a temporal formula
- $MC = \mathcal{M} \models \varphi$?

Advantages

- During specification/design time
- Automatic
- Global w.r.t. Test
- Efficient (in some fields)

Drawbacks

- Finite LTSs
- Requires formal expertise
- State space explosion problem

State space explosion problem

Reduction Techniques

- On-the-fly construction
  - Stop the exploration as soon as a counter-example is found

State space explosion problem

Reduction Techniques

- On-the-fly construction
  - Stop the exploration as soon as a counter-example is found
- Partial order reduction
  - Exploits the independence between actions

State space explosion problem

Reduction Techniques

- On-the-fly construction
  - Stop the exploration as soon as a counter-example is found
- Partial order reduction
  - Exploits the independence between actions
- Stuttering equivalence
  - stutter-invariant formula
    - $a\bar{b}.a\bar{b}.a\bar{b}.ab.ab.ab\ldots$
    - $a\bar{b}.a\bar{b}.a\bar{b}.a\bar{b}.ab.ab.ab.\ldots$

State space explosion problem

Reduction Techniques

- On-the-fly construction
  - Stop the exploration as soon as a counter-example is found
- Partial order reduction
  - Exploits the independence between actions
- Stuttering equivalence
  - stutter-invariant formula
    - $a\bar{b}.a\bar{b}.a\bar{b}.ab.ab.ab\ldots$
    - $a\bar{b}.a\bar{b}.a\bar{b}.a\bar{b}.ab.ab.ab.\ldots$
- Modularity

State space explosion problem

Reduction Techniques

- On-the-fly construction
  - Stop the exploration as soon as a counter-example is found
- Partial order reduction
  - Exploits the independence between actions
- Stuttering equivalence
  - stutter-invariant formula
    - $a\bar{b}.a\bar{b}.a\bar{b}.ab.ab.ab\ldots$
    - $a\bar{b}.a\bar{b}.a\bar{b}.a\bar{b}.ab.ab.ab.\ldots$
- Modularity
- Symbolic representations (e.g., BDDs)
- ...

State space explosion problem

Reduction Techniques

- On-the-fly construction ✓
  - Stop the exploration as soon as a counter-example is found
- Partial order reduction
  - Exploits the independence between actions
- Stuttering equivalence ✓
  - stutter-invariant formula
    - $a\bar{b}.a\bar{b}.a\bar{b}.ab.ab.ab\ldots$
    - $a\bar{b}.a\bar{b}.a\bar{b}.a\bar{b}.ab.ab.ab.\ldots$
- Modularity ✓
- Symbolic representations (e.g., BDDs) ✓
- ...

Outline

State Machines

Syntactical Representation of a System

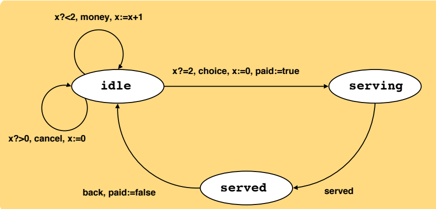
S=(C,V,A,T)

- C: Control States
- V: Variables
- A: Actions on V
- T: Transitions

Syntactical Representation of a System

S=(C,V,A,T)

- C: Control States
- V: Variables
- A: Actions on V
- T: Transitions



Labeled Transition System (LTS)

LTS = Semantics of the system

S=(Q,T,$\rightarrow$)

- Q: set of states (control state,valeriable's values)
- T: set of transitions
- $\rightarrow \subseteq Q \times T \times Q$: the transition relation
- we can add an initial state I

Labeled Transition System (LTS)

LTS = Semantics of the system

S=(Q,T,$\rightarrow$)

- Q: set of states (control state,valeriable's values)
- T: set of transitions
- $\rightarrow \subseteq Q \times T \times Q$: the transition relation
- we can add an initial state I

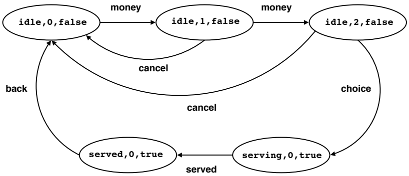
Q represents the possible states of the system
a transition t can be executed at state a leading to state q' is $(q, t, q') \in \rightarrow$ (denoted by $q \xrightarrow{t} q'$)

Labeled Transition System (LTS)

LTS = Semantics of the system

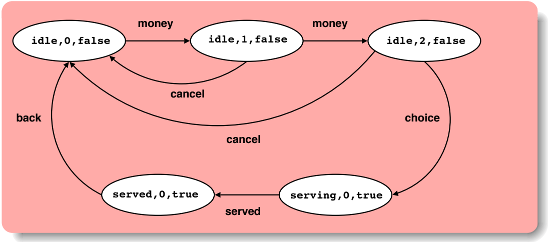
S=(Q,T,$\rightarrow$)

- Q: set of states (control state,valeriable's values)
- T: set of transitions
- $\rightarrow \subseteq Q \times T \times Q$: the transition relation
- we can add an initial state I



Q represents the possible states of the system
a transition t can be executed at state a leading to state q' is $(q, t, q') \in \rightarrow$ (denoted by $q \xrightarrow{t} q'$)

Executions of the system

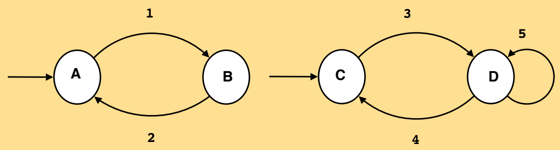


- $(i, 0, f) \xrightarrow{money} (i, 1, f) \xrightarrow{money} (i, 2, f) \xrightarrow{choice} (sg, 0, t) \ldots$
- money, money, choice, served, back, money

$L(S)$ = Language of S= The set of executions of S

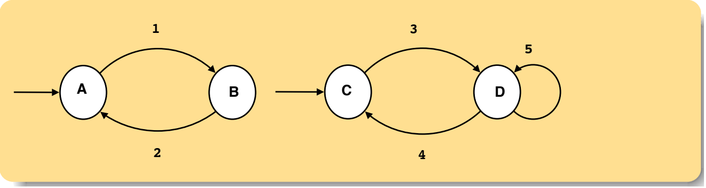
Concurrency

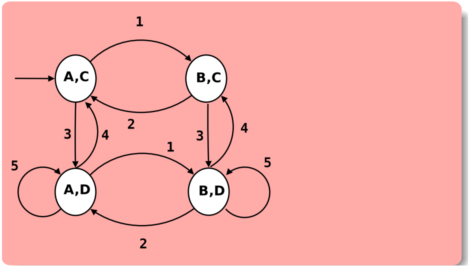
Asynchronous product



Concurrency

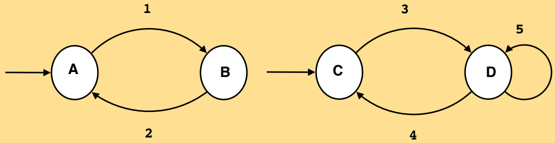
Asynchronous product



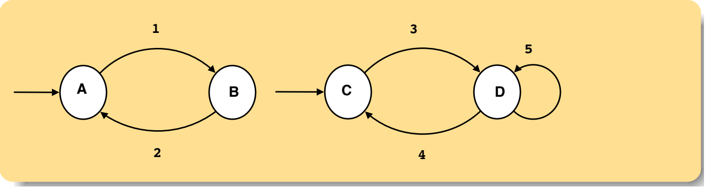


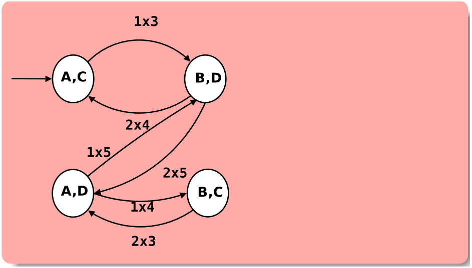
Concurrency

Synchronous product

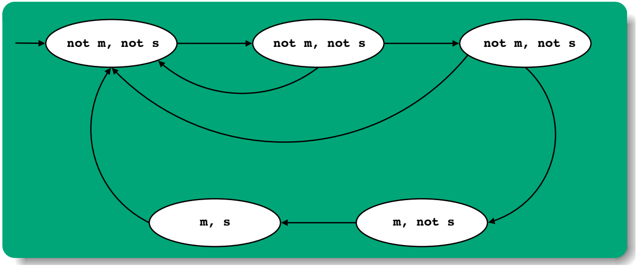


Synchronous product

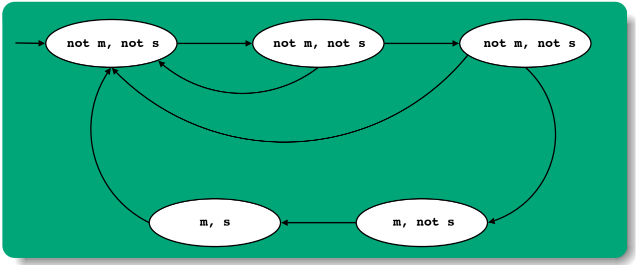


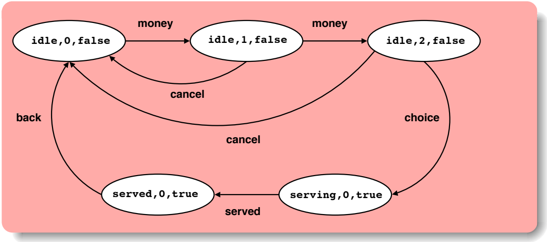


Kripke structure



Kripke structure





Exercice: The Lift Example

The lift controller system (for 3 floors) is defined by:

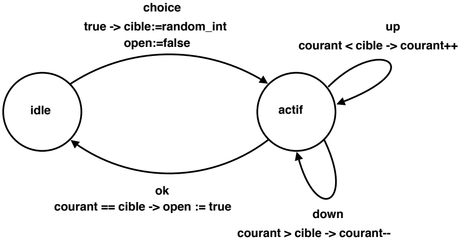
1. the controller saves in memory the current and the target floors.
2. in active mode, when the target floor is reached, the doors are opened and the controller switches to the idle mode.
3. in active, when the target floor is greater than the current one, the controller raises the lift.
4. in active, when the target floor is lower than the current one, the controller lowers the lift.
5. in the idle mode, it may be that someone enters the lift and choose a new target floor. The elevator then closes the doors and becomes active.
6. initially, the elevator is at floor 0 and in the idle mode.

Questions

1. Design the system using a state machine (formal definition and the figure).
2. Define and draw the corresponding transition system.

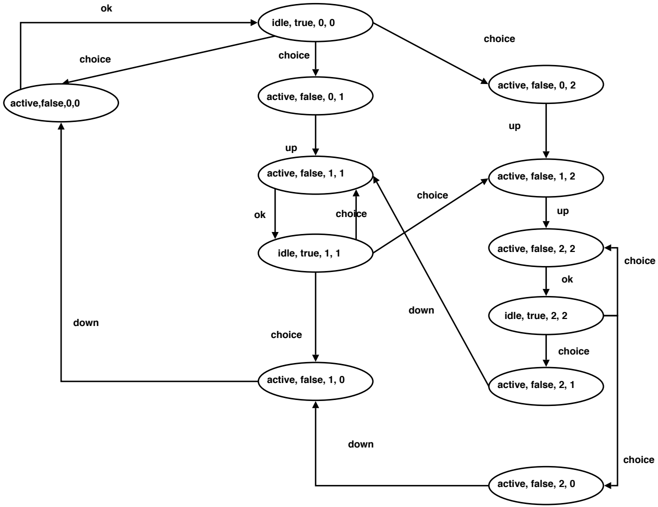
State Machine

- V= `courant: int[0...2]`, cible: int[0...2], open: bool
- `random.in` $\in$ `[0...2]`



The Lift Example

Labeled Transition System



Outline

Plan

4. Formal Specifications
   - Petri nets
   - Coverability Graph
   - Linear Temporal Logic (LTL)

Petri Nets [Petri 73]

Syntax

Definition

A Petri net is 5-tuple $N = \langle P, T, F, W, m_0 \rangle$ where:

- P is a finite set of places (cercles) and T a finite set of transitions (squares) with $(P \cup T) \neq \emptyset$ and $P \cap T = \emptyset$,
- A flow relation $F \subseteq (P \times T) \cup (T \times P)$,
- $W : F \rightarrow \mathbb{N}^+$ assigns a weight (> 0)to any arc.
- An initial marking m_0 where a marking m is a mapping $m : P \rightarrow \mathbb{N}$.

Petri Nets [Petri 73]

Syntax

Definition

A Petri net is 5-tuple $N = \langle P, T, F, W, m_0 \rangle$ where:

- P is a finite set of places (cercles) and T a finite set of transitions (squares) with $(P \cup T) \neq \emptyset$ and $P \cap T = \emptyset$,
- A flow relation $F \subseteq (P \times T) \cup (T \times P)$,
- $W : F \rightarrow \mathbb{N}^+$ assigns a weight (> 0)to any arc.
- An initial marking m_0 where a marking m is a mapping $m : P \rightarrow \mathbb{N}$.

Incidence matrix C: $\forall (p, t) \in P \times T : C(p, t) = W(t, p) - W(p, t)$

Petri Nets [Petri 73]

Syntax

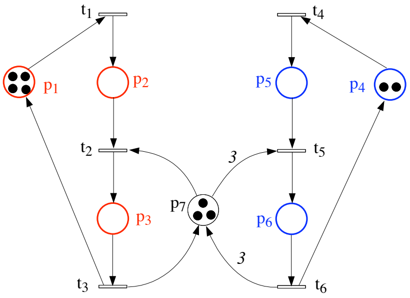
Definition

A Petri net is 5-tuple $N = \langle P, T, F, W, m_0 \rangle$ where:

- P is a finite set of places (cercles) and T a finite set of transitions (squares) with $(P \cup T) \neq \emptyset$ and $P \cap T = \emptyset$,
- A flow relation $F \subseteq (P \times T) \cup (T \times P)$,
- $W : F \rightarrow \mathbb{N}^+$ assigns a weight (> 0)to any arc.
- An initial marking m_0 where a marking m is a mapping $m : P \rightarrow \mathbb{N}$.

Incidence matrix C: $\forall (p, t) \in P \times T : C(p, t) = W(t, p) - W(p, t)$
Notation: $C(p, t) = Post(t, p) - Pre(t, p))$

Petri Nets: an example



$$
\begin{bmatrix}
-1 & 0 & 1 & 0 & 0 & 0 \\
1 & -1 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 1 \\
0 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 1 & -1 \\
0 & -1 & 1 & 0 & -3 & 3
\end{bmatrix}
$$

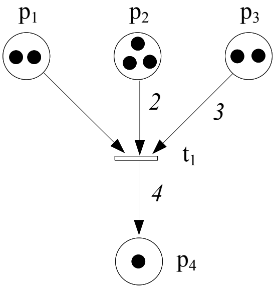
Petri Nets: Semantics

- Fireability of a transition

Petri Nets: Semantics

- Fireability of a transition
  - t is fireable at a marking m iff $\forall p,\ W(p,t) \leq m(p)$

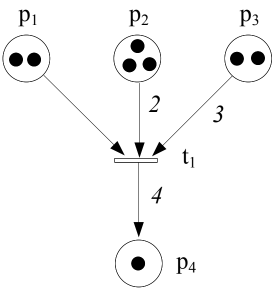
- Fireability of a transition
  - t is fireable at a marking m iff $\forall p,\ W(p,t) \leq m(p)$



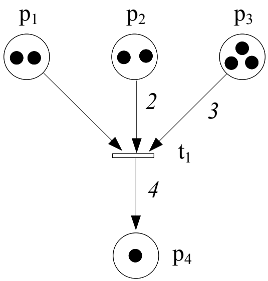
not firable

Petri Nets: Semantics

- Fireability of a transition
  - t is fireable at a marking m iff $\forall p,\ W(p,t) \leq m(p)$



not firable



firable

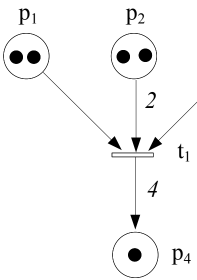
Petri Nets: Semantics

- Firing a transition

Petri Nets: Semantics

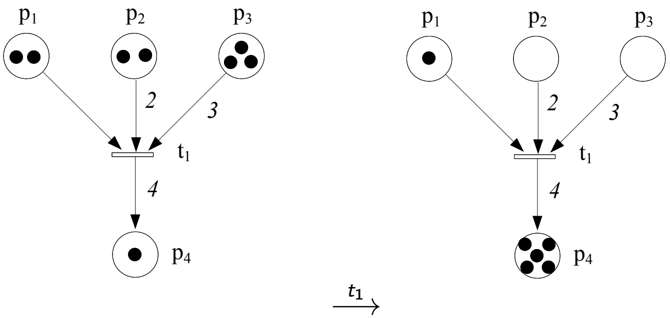
- Firing a transition
  - The firing of a (fireable) transition t from a marking m leads to $m' = m - W(p,t) + W(t,p)$

- Firing a transition
  - The firing of a (fireable) transition t from a marking m leads to $m' = m - W(p,t) + W(t,p)$



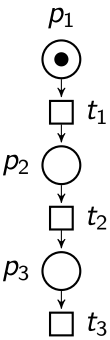
Petri Nets: Semantics

- Firing a transition
  - The firing of a (fireable) transition t from a marking m leads to $m' = m - W(p,t) + W(t,p)$



Petri Nets: Expression Power

- Causality

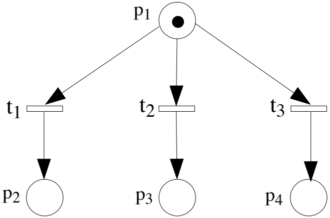


Petri Nets: Expression Power

- Conflict/Choice

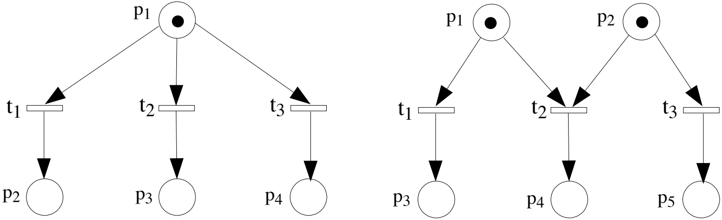
Petri Nets: Expression Power

- Conflict/Choice



Petri Nets: Expression Power

- Conflict/Choice

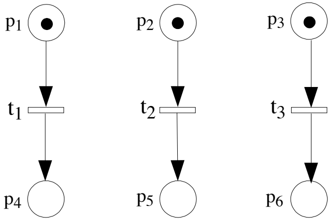


Petri Nets: Expression Power

- Parallelism

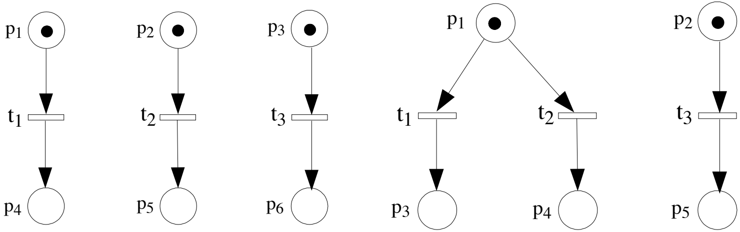
Petri Nets: Expression Power

- Parallelism



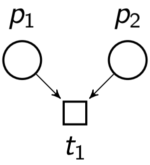
Petri Nets: Expression Power

- Parallelism

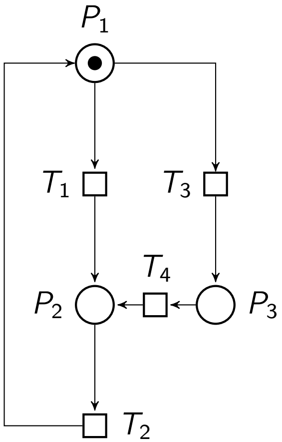


Petri Nets: Expression Power

- Synchronization

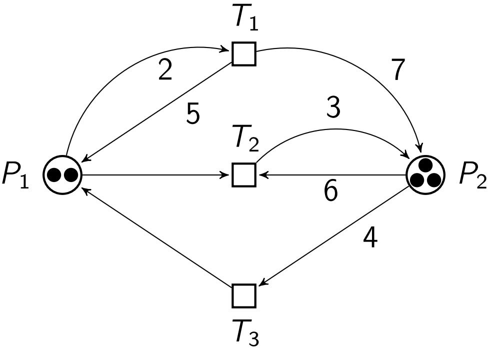


Petri Nets: exercice 1



1. Give the Pre, Post and the incidence matrices of this Petri net.
2. Which are the fireable transitions from the initial marking?

Petri Nets: exercice 2



1. Is T_1 fireable from the initial marking? If yes, which is the reachable marking?
2. Give the incidence matrix of this Petri net.
3. Check formally the fireability of the transition T_1. If T_1 is fireable, then compute the reachable marking formally.

Petri Nets: Semantics (Cont.)

- $\sigma = t_1 \ldots t_n \in T^*$ is fireable at m_0 (denoted by $m_0 \xrightarrow{\sigma}$ iff $\exists m_1 \ldots m_n$ s.t. $m_0 \xrightarrow{t_1} m_1 \longrightarrow \ldots \xrightarrow{t_n} m_n$

Petri Nets: Semantics (Cont.)

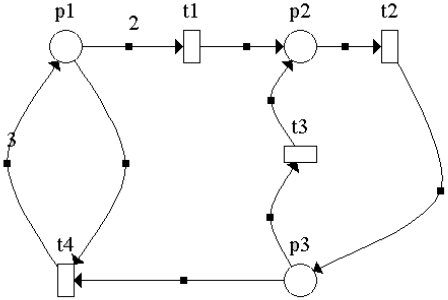
- $\sigma = t_1 \ldots t_n \in T^*$ is fireable at m_0 (denoted by $m_0 \xrightarrow{\sigma}$ iff $\exists m_1 \ldots m_n$ s.t. $m_0 \xrightarrow{t_1} m_1 \longrightarrow \ldots \xrightarrow{t_n} m_n$
- $L(N, m_0) = \{\sigma \in T^* \mid m_0 \xrightarrow{\sigma}\}$

Petri Nets: Semantics (Cont.)

- $\sigma = t_1 \dots t_n \in T^*$ is fireable at m_0 (denoted by $m_0 \xrightarrow{\sigma}$ iff $\exists m_1 \dots m_n$ s.t. $m_0 \xrightarrow{t_1} m_1 \longrightarrow \dots \xrightarrow{t_n} m_n$
- $L(N, m_0) = \{\sigma \in T^* \mid m_0 \xrightarrow{\sigma}\}$
- $R(N, m)$ = the set markings reachable from a marking m of N

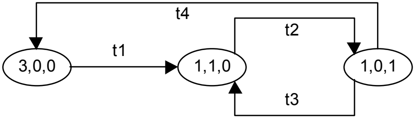
Petri Nets: Semantics (Cont.)

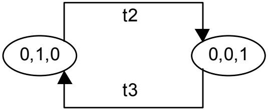
- $\sigma = t_1 \dots t_n \in T^*$ is fireable at m_0 (denoted by $m_0 \xrightarrow{\sigma}$ iff $\exists m_1 \dots m_n$ s.t. $m_0 \xrightarrow{t_1} m_1 \longrightarrow \dots \xrightarrow{t_n} m_n$
- $L(N, m_0) = \{\sigma \in T^* \mid m_0 \xrightarrow{\sigma}\}$
- $R(N, m)$ = the set markings reachable from a marking m of N
- the reachability graph is a LTS $\langle S, A, \rightarrow, s_0 \rangle$ s.t.
    - $S = R(N, m_0)$
    - $A = T$
    - $s_0 = m_0$
    - $(s_1, t, s_2) \in \rightarrow$ iff $s_1 \xrightarrow{t} s_2$



initial marking $(3, 0, 0)$, then $(0, 1, 0)$

Petri Nets: Reachability Graph

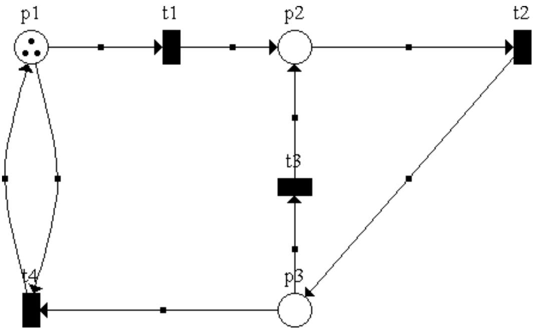




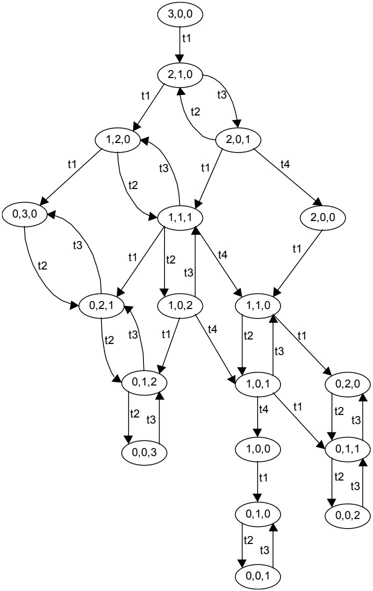
$m_0 = (3, 0, 0)$

$m_0 = (0, 1, 0)$

Petri Nets: Reachability Graph



Petri Nets: Reachability Graph



Example: Mutual Exclusion Algorithm

Global variables: req_P and req_Q

Process P

1. $req_P \leftarrow 1$
2. `wait(`$req_Q = 0$`)`
3. `Critical Section`
4. $req_P \leftarrow 0$

Process Q

1. $req_Q \leftarrow 1$
2. `wait(`$req_P = 0$`)`
3. `Critical Section`
4. $req_Q \leftarrow 0$

Initial state: $req_P = req_Q = 0$

Example: Mutual Exclusion Algorithm

Global variables: req_P and req_Q

Process P

```
1.   reqP ← 1
2.   wait(reqQ = 0)
3.   Critical Section
4.   reqP ← 0
```

Process Q

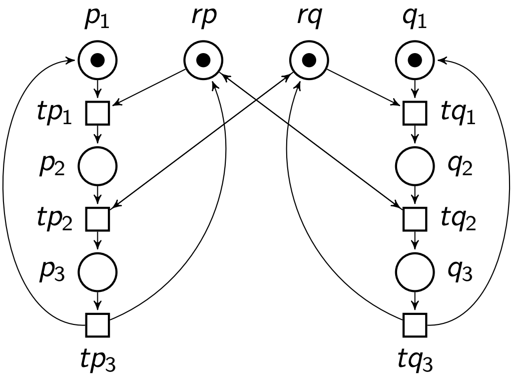
```
1.   reqQ ← 1
2.   wait(reqP = 0)
3.   Critical Section
4.   reqQ ← 0
```

Initial state: $req_P = req_Q = 0$

Properties to be checked:

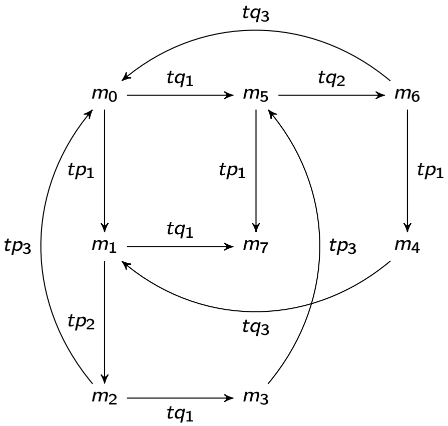
1. Mutual exclusion
2. Fairness
3. Order

Example: Mutual Exclusion Algorithm



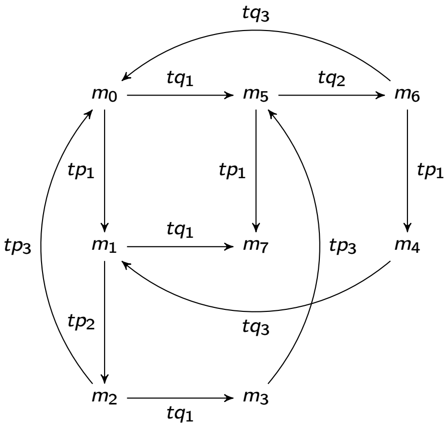
Example: Mutual Exclusion Algorithm

$m_0 = p_1 + rp + rq + q_1$
$m_1 = p_2 + rq + q_1$
$m_2 = p_3 + rq + q_1$
$m_3 = p_3 + q_2$
$m_4 = p_3 + q_2$
$m_5 = p_1 + rp + q_2$
$m_6 = p_1 + rp + q_3$
$m_7 = p_2 + q_2$



Example: Mutual Exclusion Algorithm

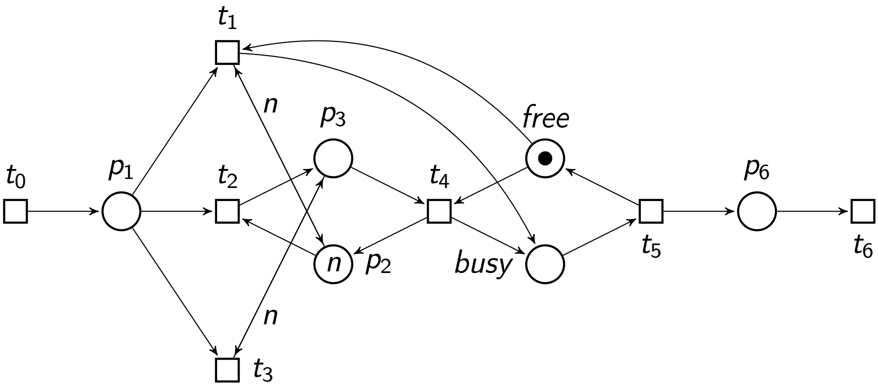
$m_0 = p_1 + rp + rq + q_1$
$m_1 = p_2 + rq + q_1$
$m_2 = p_3 + rq + q_1$
$m_3 = p_3 + q_2$
$m_4 = p_3 + q_2$
$m_5 = p_1 + rp + q_2$
$m_6 = p_1 + rp + q_3$
$m_7 = p_2 + q_2$



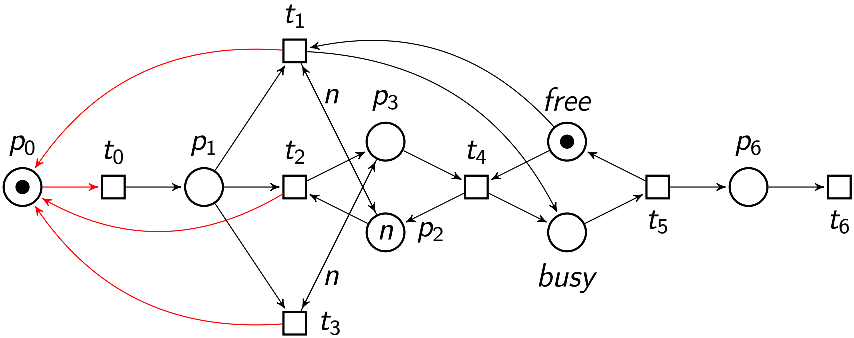
Compare with the previous reachability graph of the mutual exclusion example

Petri Nets modeling a hairdresser

Petri Nets modeling a hairdresser



Petri Nets modeling a hairdresser (Cont.)



Petri Nets: Properties

- A marking m^* is a home state if and only if $\forall m \in R(N, m_0),\ m^* \in R(N, m)$.

Petri Nets: Properties

- A marking m^* is a home state if and only if $\forall m \in R(N, m_0),\ m^* \in R(N, m)$.
- N is reversible iff m_0 is a home state.

Petri Nets: Properties

- A marking m^* is a home state if and only if $\forall m \in R(N, m_0),\ m^* \in R(N, m)$.
- N is reversible iff m_0 is a home state.
- N is bounded iff $\forall p \in P:\ \exists k \in \mathbb{N}$ s.t. $\forall m \in R(N, m_0),\ m(p) \leq k$.

Petri Nets: Properties

- A marking m^* is a home state if and only if $\forall m \in R(N, m_0),\ m^* \in R(N, m)$.
- N is reversible iff m_0 is a home state.
- N is bounded iff $\forall p \in P:\ \exists k \in \mathbb{N}$ s.t. $\forall m \in R(N, m_0),\ m(p) \leq k$.
- N is structurally bounded iff N is bounded for all initial marking m_0.

Petri Nets: Properties

- A marking m^* is a home state if and only if $\forall m \in R(N, m_0),\ m^* \in R(N, m)$.
- N is reversible iff m_0 is a home state.
- N is bounded iff $\forall p \in P:\ \exists k \in \mathbb{N}$ s.t. $\forall m \in R(N, m_0),\ m(p) \leq k$.
- N is structurally bounded iff N is bounded for all initial marking m_0.
- N is quasi-live iff $\forall t \in T:\ \exists M \in R(N, m_0)$ for which t is enabled.

Petri Nets: Properties

- A marking m^* is a home state if and only if $\forall m \in R(N, m_0),\ m^* \in R(N, m)$.
- N is reversible iff m_0 is a home state.
- N is bounded iff $\forall p \in P:\ \exists k \in \mathbb{N}$ s.t. $\forall m \in R(N, m_0),\ m(p) \leq k$.
- N is structurally bounded iff N is bounded for all initial marking m_0.
- N is quasi-live iff $\forall t \in T:\ \exists M \in R(N, m_0)$ for which t is enabled.
- N is deadlock-free (weakly live) iff $\forall M \in R(N, m_0),\ \exists t \in T$ enabled in M.

Petri Nets: Properties

- A marking m^* is a home state if and only if $\forall m \in R(N, m_0),\ m^* \in R(N, m)$.
- N is reversible iff m_0 is a home state.
- N is bounded iff $\forall p \in P :\ \exists k \in \mathbb{N}$ s.t. $\forall m \in R(N, m_0),\ m(p) \leq k$.
- N is structurally bounded iff N is bounded for all initial marking m_0.
- N is quasi-live iff $\forall t \in T :\ \exists M \in R(N, m_0)$ for which t is enabled.
- N is deadlock-free (weakly live) iff $\forall M \in R(N, m_0),\ \exists t \in T$ enabled in M.
- N is live iff $\forall t \in T :\ \forall m \in R(N, m_0) \exists m' \in R(N, m)$ for which t is enabled.

Petri Nets: Properties

- A marking m^* is a home state if and only if $\forall m \in R(N, m_0),\ m^* \in R(N, m)$.
- N is reversible iff m_0 is a home state.
- N is bounded iff $\forall p \in P:\ \exists k \in \mathbb{N}$ s.t. $\forall m \in R(N, m_0),\ m(p) \leq k$.
- N is structurally bounded iff N is bounded for all initial marking m_0.
- N is quasi-live iff $\forall t \in T:\ \exists M \in R(N, m_0)$ for which t is enabled.
- N is deadlock-free (weakly live) iff $\forall M \in R(N, m_0),\ \exists t \in T$ enabled in M.
- N is live iff $\forall t \in T:\ \forall m \in R(N, m_0) \exists m' \in R(N, m)$ for which t is enabled.
- N is structurally live iff $\forall m_0, (N, m_0)$ is live.

Relation Between Properties

- quasi-liveness VS Liveness ??

Relation Between Properties

- quasi-liveness VS Liveness ??
- quasi-liveness VS weak liveness ??

Relation Between Properties

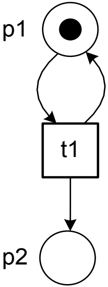
- quasi-liveness VS Liveness ??
- quasi-liveness VS weak liveness ??
- liveness VS weak liveness ??

Relation Between Properties

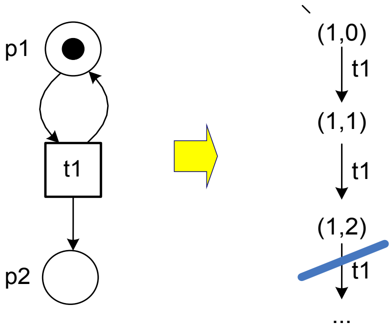
- quasi-liveness VS Liveness ??
- quasi-liveness VS weak liveness ??
- liveness VS weak liveness ??
- m_0 home state and quasi live $\Rightarrow$ live ?? (if yes, proof)

Plan

Problem



Problem



- Notations:
  - new symbol $\omega \notin \mathbb{N}$ s.t.
    - $\omega + n = \omega$
    - $\omega - n = \omega$
    - $\omega > n$
    - $\omega \leq \omega$
  - $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$
  - For $q \in \mathbb{N}_\omega^m$, $q^{-1}(\omega) = \{p \in P \mid q(p) = \omega\}$

Definition (Coverability Tree)

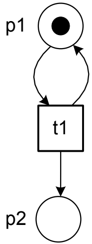
The coverability tree of a marked Petri net $\langle N, m_0 \rangle$ is a tree $\langle S, X \rangle$ where:

- nodes of S are labeled with vectors in $\mathbb{N}_\omega^m$ ($m =|| P ||$)
- edges of X are labeled with transitions in T.

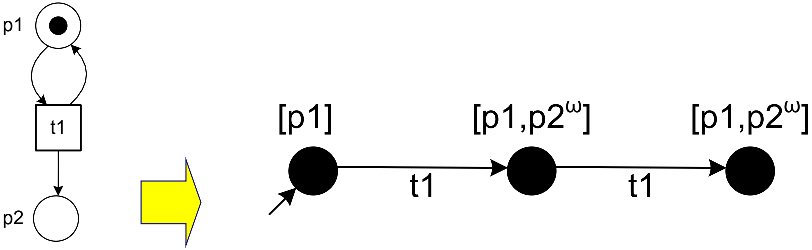
Coverability Tree: Algorithm

1. Label the initial marking m_0 as the root and tag it "new".
2. While "new" markings exists, do the following:
   1. Select a new marking m and remove the "new" tag.
   2. If m is identical to a marking on the path from the root to m, then tag m "old" and go to another new marking.
   3. If no transitions are enabled at m, tag m "dead-end".
   4. While there exist enabled transitions at m, do the following for each enabled transition t at m:
      1. Obtain the marking m' that results from firing t at m.
      2. If, on the path from the root to m, there exists a marking $m'' \neq m'$ such that $m' \geq m''$, then replace $m'(p)$ ω for each p such that $m'(p) > m''(p)$.
      3. Introduce m' as a node, draw an arc with label t from m to m', and tag m' "new".
3. Output the tree

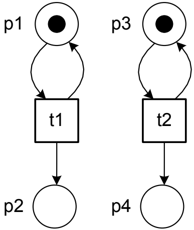
Coverability Tree: Example



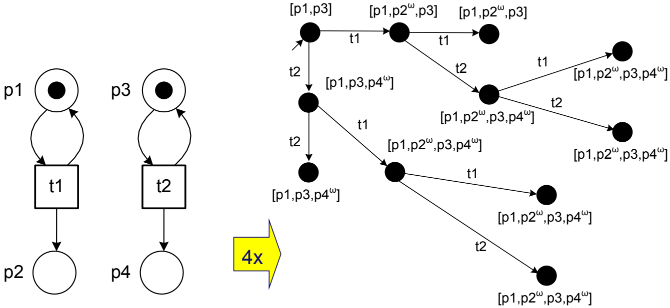
Coverability Tree: Example



Coverability Tree: Another Example



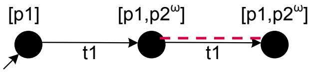
Coverability Tree: Another Example

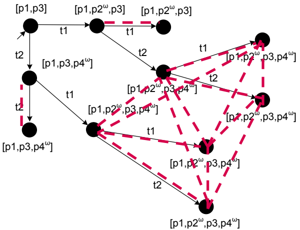


Coverability Graph

Take the coverability tree and merge nodes with identical labels

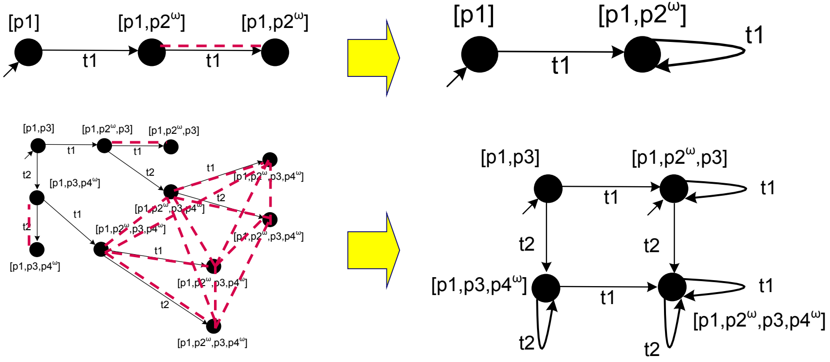
Take the coverability tree and merge nodes with identical labels



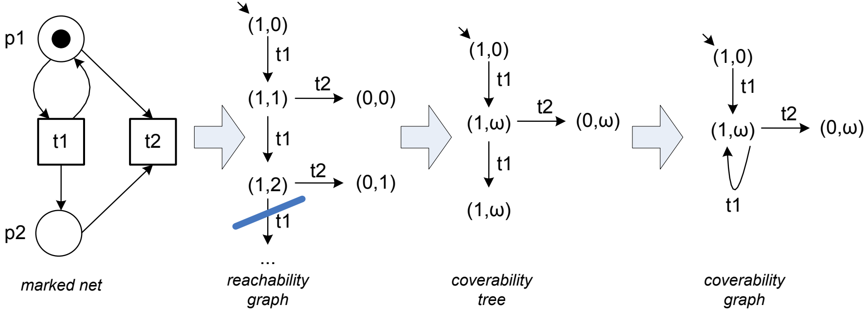


Coverability Graph

Take the coverability tree and merge nodes with identical labels



Coverability Graph: Another Example



Properties

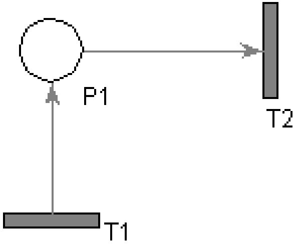
- The coverability tree/graph is always finite.
- The marked Petri net is bounded if and only if the corresponding coverability tree/graph contains only ω-free markings.
- The coverability tree/graph gives an over-approximation.
- Different Petri nets may have the same coverability tree/graph.
- Any firing sequence of the marked Petri net can be matched by a "walk" through the coverability graph.

Limitation: Loss of Information

The reverse is not true!!!!

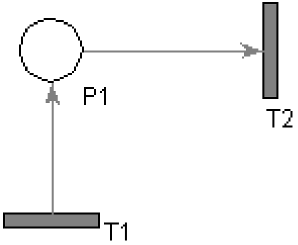
Limitation: Loss of Information

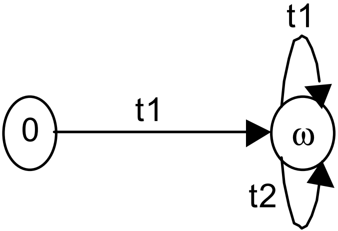
The reverse is not true!!!!



Limitation: Loss of Information

The reverse is not true!!!!



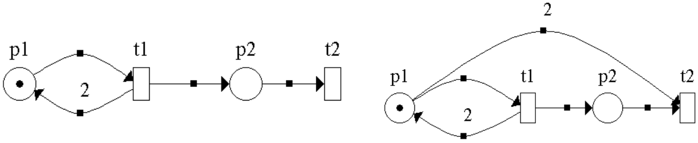


Limitation: Loss of Information

Two nets with the same coverability graph!

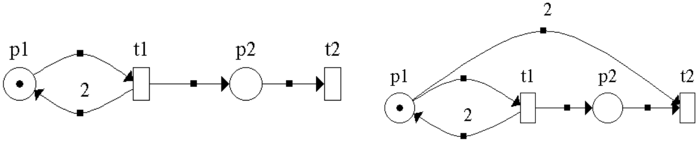
Limitation: Loss of Information

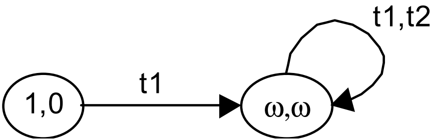
Two nets with the same coverability graph!



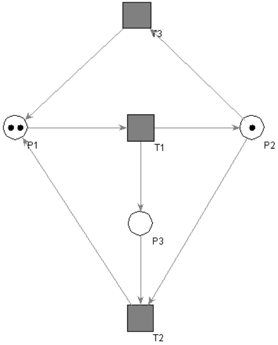
Limitation: Loss of Information

Two nets with the same coverability graph!

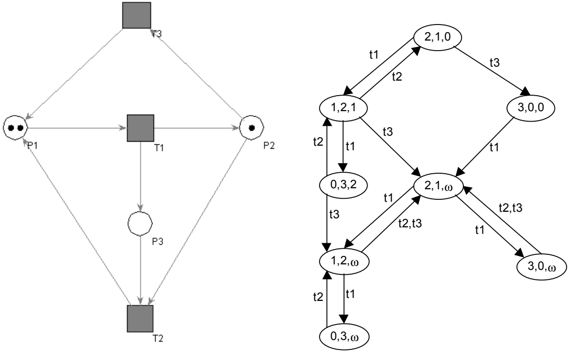




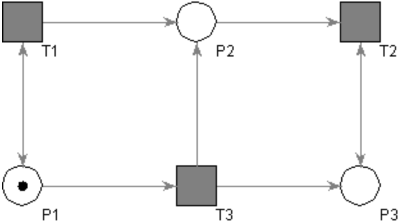
Coverability Graph: Exercice



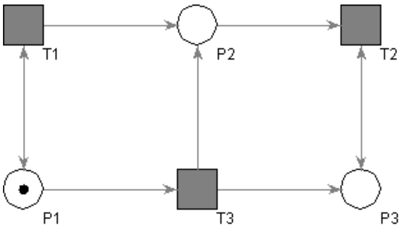
Coverability Graph: Exercice

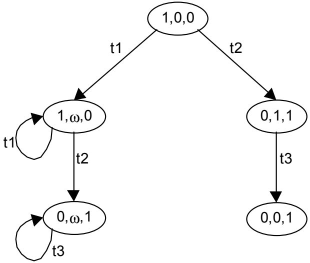


Coverability Graph: Another Exercice



Coverability Graph: Another Exercice





Plan

4. Formal Specifications
   - Petri nets
   - Coverability Graph
   - Linear Temporal Logic (LTL)

Temporal Logics

Two kinds of temporal operators

- sequence of expected events along one path
- e.g. **U**, **X**, **G**, **F**

Temporal Logics

Two kinds of temporal operators

- sequence of expected events along one path
- e.g. **U**, **X**, **G**, **F**

Insufficient: are all/some paths starting from a given state satsfy some property?

path quantifiers

- quantify paths starting from a state and satisfying a property
- e.g. **A**, **E**

Linear Temporal Logic (LTL)

Syntax

AP: a set of atomic propositions

$\varphi ::= \mathit{true} \mid$	logical constant true
$p \mid$	atomic proposition
$\neg\varphi \mid$	negation
$\varphi \wedge \varphi \mid$	and
$X\varphi \mid$	next time
$\varphi U \varphi$	Until

Linear Temporal Logic (LTL)

Syntax

AP: a set of atomic propositions

$\varphi ::= true \mid$	logical constant true
$p \mid$	atomic proposition
$\neg\varphi \mid$	negation
$\varphi \wedge \varphi \mid$	and
$X\varphi \mid$	next time
$\varphi U \varphi$	Until

Abreviations :

Linear Temporal Logic (LTL)

Syntax

AP: a set of atomic propositions

- $\varphi ::= true \mid$ logical constant true
  $p \mid$ atomic proposition
  $\neg\varphi \mid$ negation
  $\varphi \wedge \varphi \mid$ and
  $X\varphi \mid$ next time
  $\varphi U \varphi$ Until

Abreviations :

- $\varphi_1 \implies \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$

Linear Temporal Logic (LTL)

Syntax

AP: a set of atomic propositions

- $\varphi ::= true \mid$ logical constant true
  $p \mid$ atomic proposition
  $\neg\varphi \mid$ negation
  $\varphi \wedge \varphi \mid$ and
  $X\varphi \mid$ next time
  $\varphi U \varphi$ Until

Abreviations :

- $\varphi_1 \implies \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
- $F\varphi$: now or sometimes in the future
  - $F\varphi \equiv true\ U\varphi$

Linear Temporal Logic (LTL)

Syntax

AP: a set of atomic propositions

- $\varphi ::= true \mid$ logical constant true
  - $p \mid$ atomic proposition
  - $\neg\varphi \mid$ negation
  - $\varphi \wedge \varphi \mid$ and
  - $X\varphi \mid$ next time
  - $\varphi U \varphi$ Until

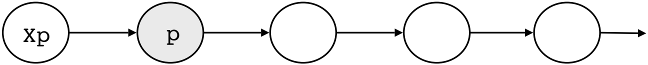
Abreviations :

- $\varphi_1 \implies \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
- $F\varphi$: now or sometimes in the future
  - $F\varphi \equiv true\ U\varphi$
- $G\varphi$: now and always in the future
  - $G\varphi \equiv \neg F \neg\varphi$

LTL: Semantics

Express sequence of events along a path

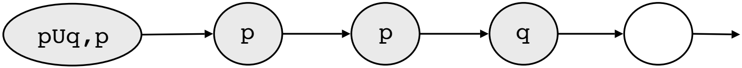
Operator **X** "next"



Temporal connectors

Express sequence of events along a path

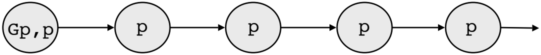
Operator **U** "p true until q true"



Temporal connectors

Express sequence of events along a path

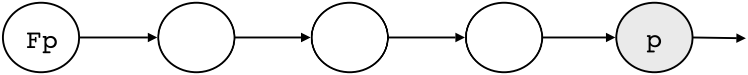
Operator **G** "always in the future"



Temporal connectors

Express sequence of events along a path

Operator **F** "eventually in the future"



LTL: Semantics

LTL is interpreted on infinite paths of a Kripke structure K.

$\pi = s_0 \longrightarrow s_1 \longrightarrow \ldots$

- $\pi \models p$ iff $p \in L(s_0)$
- $\pi \models \varphi_1 \wedge \varphi_2$ iff $\pi \models \varphi_1$ and $\pi \models \varphi_2$
- $\pi \models \neg\varphi$ iff not $\pi \models \varphi$
- $\pi \models X\varphi$ iff $\pi^1 \models \varphi$ (π^i = suffix of π starting at s_i)
- $\pi \models \varphi_1 U \varphi_2$ iff $\exists i \geq 0$ s.t. $\pi^i \models \varphi_2$ and $\forall 0 \leq j < i \wedge \pi^j \models \varphi_1$

LTL: Semantics

LTL is interpreted on infinite paths of a Kripke structure K.

$\pi = s_0 \longrightarrow s_1 \longrightarrow \dots$

- $\pi \models p$ iff $p \in L(s_0)$
- $\pi \models \varphi_1 \wedge \varphi_2$ iff $\pi \models \varphi_1$ and $\pi \models \varphi_2$
- $\pi \models \neg\varphi$ iff not $\pi \models \varphi$
- $\pi \models X\varphi$ iff $\pi^1 \models \varphi$ (π^i = suffix of π starting at s_i)
- $\pi \models \varphi_1 U \varphi_2$ iff $\exists i \geq 0$ s.t. $\pi^i \models \varphi_2$ and $\forall 0 \leq j < i \wedge \pi^j \models \varphi_1$

$$K \models \varphi \Leftrightarrow \forall \text{ path } \pi \text{ of } K,\ \pi \models \varphi$$

LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur
2. p is always true
3. p occurs infinitely often
4. p and q are never true simultaneously
5. After an occurrence of p there will be at least one occurrence of q
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2.
7. Before each occurrence of p, there is at least one occurrence of q.
8. Between each couple of occurrence of p there is at least one occurrence of q.
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup.
10. If the machine accepts a card, it does not accept the other before having ejected the first card.

LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur (Fp)
2. p is always true
3. p occurs infinitely often
4. p and q are never true simultaneously
5. After an occurrence of p there will be at least one occurrence of q
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2.
7. Before each occurrence of p, there is at least one occurrence of q.
8. Between each couple of occurrence of p there is at least one occurrence of q.
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup.
10. If the machine accepts a card, it does not accept the other before having ejected the first card.

LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur (Fp)
2. p is always true (Gp)
3. p occurs infinitely often
4. p and q are never true simultaneously
5. After an occurrence of p there will be at least one occurrence of q
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2.
7. Before each occurrence of p, there is at least one occurrence of q.
8. Between each couple of occurrence of p there is at least one occurrence of q.
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup.
10. If the machine accepts a card, it does not accept the other before having ejected the first card.

LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur (Fp)
2. p is always true (Gp)
3. p occurs infinitely often(GFp)
4. p and q are never true simultaneously
5. After an occurrence of p there will be at least one occurrence of q
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2.
7. Before each occurrence of p, there is at least one occurrence of q.
8. Between each couple of occurrence of p there is at least one occurrence of q.
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup.
10. If the machine accepts a card, it does not accept the other before having ejected the first card.

LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur (Fp)
2. p is always true (Gp)
3. p occurs infinitely often(GFp)
4. p and q are never true simultaneously $(\neg F(p \wedge q)$ ou encore $G\neg(p \wedge q))$
5. After an occurrence of p there will be at least one occurrence of q
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2.
7. Before each occurrence of p, there is at least one occurrence of q.
8. Between each couple of occurrence of p there is at least one occurrence of q.
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup.
10. If the machine accepts a card, it does not accept the other before having ejected the first card.

LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur (Fp)
2. p is always true (Gp)
3. p occurs infinitely often(GFp)
4. p and q are never true simultaneously $(\neg F(p \wedge q)$ ou encore $G\neg(p \wedge q))$
5. After an occurrence of p there will be at least one occurrence of q $(G(p \implies F\,q))$
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2.
7. Before each occurrence of p, there is at least one occurrence of q.
8. Between each couple of occurrence of p there is at least one occurrence of q.
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup.
10. If the machine accepts a card, it does not accept the other before having ejected the first card.

LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur (Fp)
2. p is always true (Gp)
3. p occurs infinitely often(GFp)
4. p and q are never true simultaneously $(\neg F(p \wedge q)$ ou encore $G\neg(p \wedge q))$
5. After an occurrence of p there will be at least one occurrence of q $(G(p \implies F\,q))$
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2. $((GFp_1 \wedge GFP_2) \implies G(q_1 \implies Fq_2))$
7. Before each occurrence of p, there is at least one occurrence of q.
8. Between each couple of occurrence of p there is at least one occurrence of q.
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup.
10. If the machine accepts a card, it does not accept the other before having ejected the first card.

LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur (Fp)
2. p is always true (Gp)
3. p occurs infinitely often(GFp)
4. p and q are never true simultaneously ($\neg F(p \wedge q)$ ou encore $G\neg(p \wedge q)$)
5. After an occurrence of p there will be at least one occurrence of q ($G(p \implies F\,q)$)
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2. ($(GFp_1 \wedge GFP_2) \implies G(q_1 \implies Fq_2)$)
7. Before each occurrence of p, there is at least one occurrence of q. ($G(\neg p) \vee (\neg p)Uq$)
8. Between each couple of occurrence of p there is at least one occurrence of q.
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup.
10. If the machine accepts a card, it does not accept the other before having ejected the first card.

LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur (Fp)
2. p is always true (Gp)
3. p occurs infinitely often(GFp)
4. p and q are never true simultaneously ($\neg F(p \wedge q)$ ou encore $G\neg(p \wedge q)$)
5. After an occurrence of p there will be at least one occurrence of q ($G(p \implies F\,q)$)
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2. ($(GFp_1 \wedge GFP_2) \implies G(q_1 \implies Fq_2)$)
7. Before each occurrence of p, there is at least one occurrence of q. ($G(\neg p) \vee (\neg p)Uq$)
8. Between each couple of occurrence of p there is at least one occurrence of q. ($G(p \implies X\,(G\neg p \vee Fp \wedge ((\neg p)Uq)))$)
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup.
10. If the machine accepts a card, it does not accept the other before having ejected the first card.

LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur (Fp)
2. p is always true (Gp)
3. p occurs infinitely often(GFp)
4. p and q are never true simultaneously $(\neg F(p \wedge q)$ ou encore $G\neg(p \wedge q))$
5. After an occurrence of p there will be at least one occurrence of q $(G(p \implies F\,q))$
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2. $((GFp_1 \wedge GFP_2) \implies G(q_1 \implies Fq_2))$
7. Before each occurrence of p, there is at least one occurrence of q. $(G(\neg p) \vee (\neg p)Uq)$
8. Between each couple of occurrence of p there is at least one occurrence of q. $(G(p \implies X\,(G\neg p \vee Fp \wedge ((\neg p)Uq))))$
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup. $(G(pay \implies (\neg orderUremove)))$
10. If the machine accepts a card, it does not accept the other before having ejected the first card.

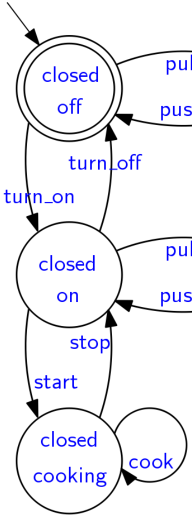
LTL: Exercice

Ecrire les formules LTL formalisant les propriétés suivantes :

1. One day, p will occur (Fp)
2. p is always true (Gp)
3. p occurs infinitely often(GFp)
4. p and q are never true simultaneously $(\neg F(p \wedge q)$ ou encore $G\neg(p \wedge q))$
5. After an occurrence of p there will be at least one occurrence of q $(G(p \implies F\, q))$
6. If p_1 occurs infinitely often and p_2 occurs inifnitely often, then each occurrence of q_1 is followed by an occurrence of q_2. $((GFp_1 \wedge GFP_2) \implies G(q_1 \implies Fq_2))$
7. Before each occurrence of p, there is at least one occurrence of q. $(G(\neg p) \vee (\neg p)Uq)$
8. Between each couple of occurrence of p there is at least one occurrence of q. $(G(p \implies X\,(G\neg p \vee Fp \wedge ((\neg p)Uq))))$
9. No other coffee orders are accepted between the payment of the amount due and the removal of the cup. $(G(pay \implies (\neg orderUremove)))$
10. If the machine accepts a card, it does not accept the other before having ejected the first card. $(G(accept \implies X(\neg accept\ U\ eject)))$

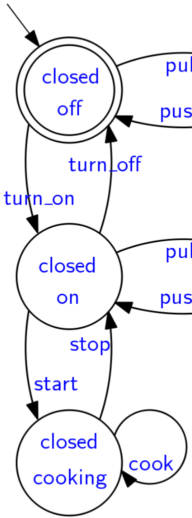
Does the property holds? counterexample?

$G(start \implies F\ stop)$



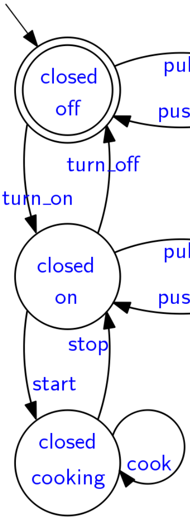
Does the property holds? counterexample?

$G(start \implies F\ stop)$ ✓



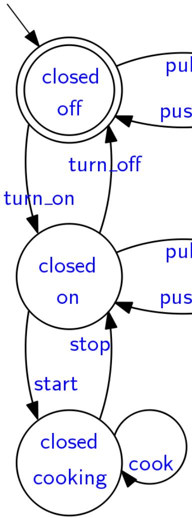
Does the property holds? counterexample?

G F turn_off



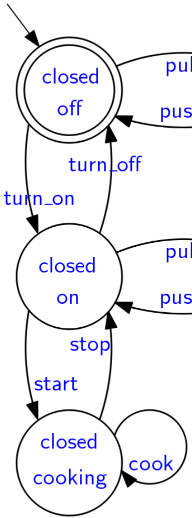
Does the property holds? counterexample?

G F turn_off ✓ (*pull push*)$^\omega$



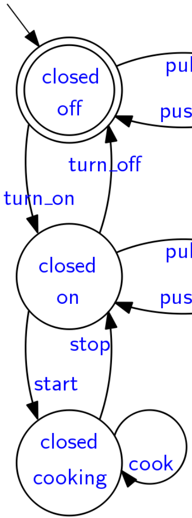
Does the property holds? counterexample?

$G\ F\ (turn\_off \vee push)$



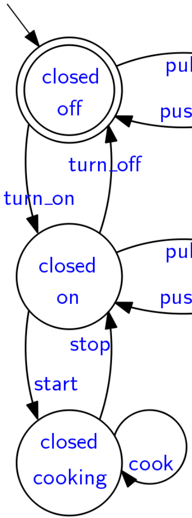
Does the property holds? counterexample?

$G\ F\ (turn\_off \vee push)$ ✓



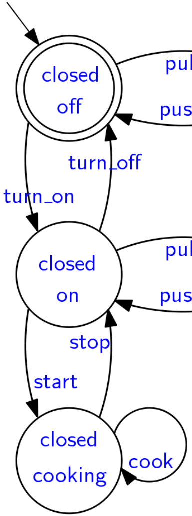
Does the property holds? counterexample?

$G\ False \vee F(turn\_off \vee push)$



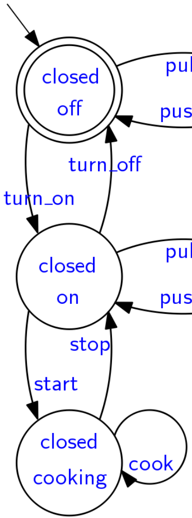
Does the property holds? counterexample?

$G\ False \vee F(turn\_off \vee push)$ ✓



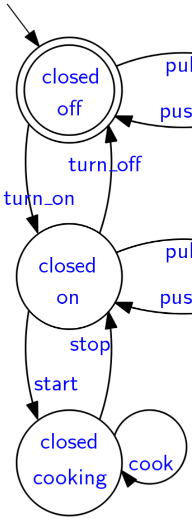
Does the property holds? counterexample?

$G(start \implies (cook\ U\ F\ turn\_off))$



Does the property holds? counterexample?

$G(start \implies (cook\ U\ F\ turn\_off))$ ✓



Outline

Plan

Büchi Automata

Definition

A Büchi automaton is 6-tuple $A = \langle \Sigma, Q, Q_0, F, \delta \rangle$ where:

- Σ is a finite alphabet
- Q is a finite set of state
- Q_0 is a set of initial states
- F is a set of accepting states
- $\delta \subseteq Q \times 2^{\Sigma} \times Q$.

Büchi Automata

Definition

A Büchi automaton is 6-tuple $A = \langle \Sigma, Q, Q_0, F, \delta \rangle$ where:

- Σ is a finite alphabet
- Q is a finite set of state
- Q_0 is a set of initial states
- F is a set of accepting states
- $\delta \subseteq Q \times 2^{\Sigma} \times Q$.

- An infinite run is accepted by A iff it goes through states of F infinitely often

Büchi Automata

Definition

A Büchi automaton is 6-tuple $A = \langle \Sigma, Q, Q_0, F, \delta \rangle$ where:

- Σ is a finite alphabet
- Q is a finite set of state
- Q_0 is a set of initial states
- F is a set of accepting states
- $\delta \subseteq Q \times 2^{\Sigma} \times Q$.

- An infinite run is accepted by A iff it goes through states of F infinitely often
- For any LTL formula φ there exists a Büchi automaton Q_φ s.t. $L(A_\varphi) = L(\varphi)$

Büchi Automata

Definition

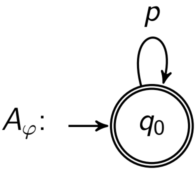
A Büchi automaton is 6-tuple $A = \langle \Sigma, Q, Q_0, F, \delta \rangle$ where:

- Σ is a finite alphabet
- Q is a finite set of state
- Q_0 is a set of initial states
- F is a set of accepting states
- $\delta \subseteq Q \times 2^{\Sigma} \times Q$.

- An infinite run is accepted by A iff it goes through states of F infinitely often
- For any LTL formula φ there exists a Büchi automaton Q_φ s.t. $L(A_\varphi) = L(\varphi)$
- Generalized Büchi Automata (State/Transition-Based)

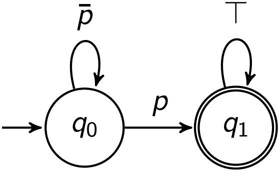
From LTL to Büchi automata

$\varphi = Gp$



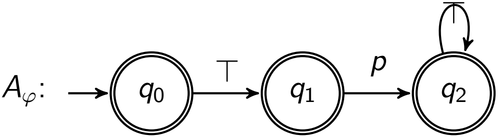
$\varphi = Fp$

A_φ:



From LTL to Büchi automata

$\varphi = Xp$



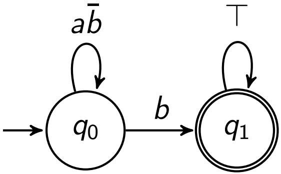
From LTL to Büchi automata

$\varphi = a \, \mathbf{U} \, b$

From LTL to Büchi automata

$\varphi = a \ \mathbf{U} \ b$

A_φ:



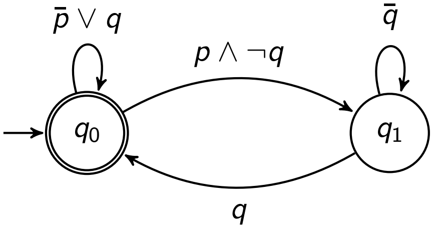
From LTL to Büchi automata

$$\varphi = G(p \implies F\, q)$$

From LTL to Büchi automata

$\varphi = G(p \implies F\, q)$

A_φ:



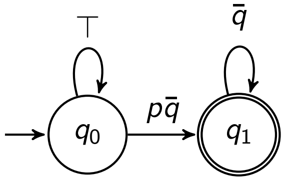
From LTL to Büchi automata

$\varphi = \neg G(p \implies F\ q)$

From LTL to Büchi automata

$\varphi = \neg G(p \implies F\ q)$

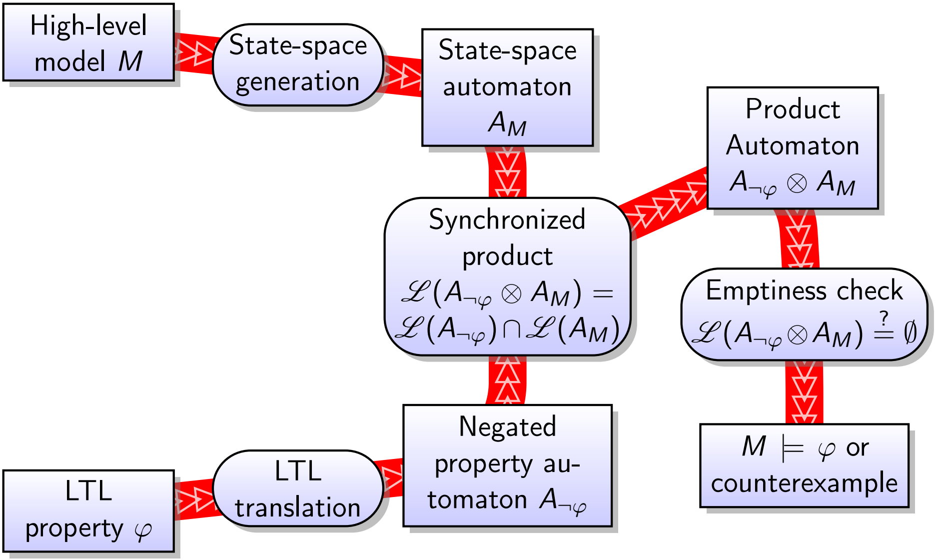
A_φ:



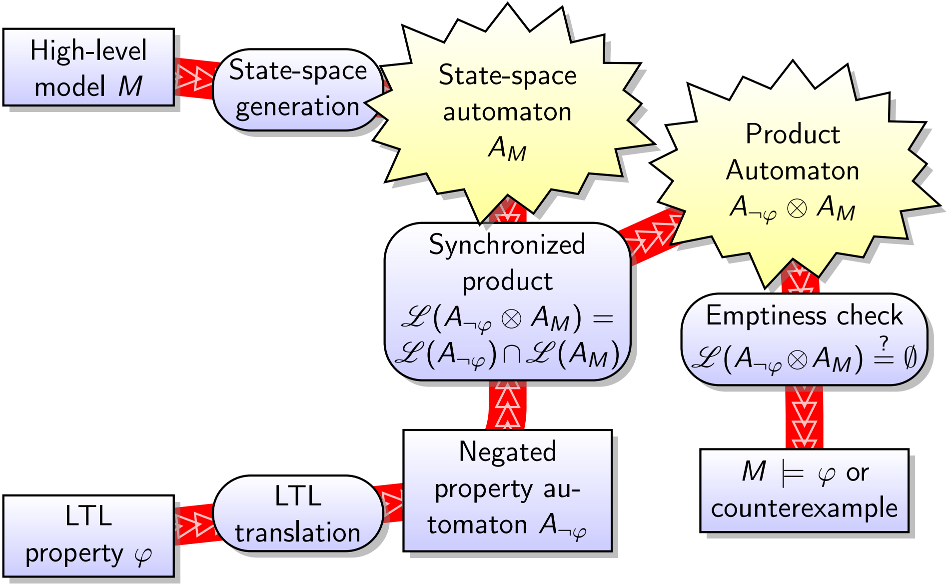
Plan

5. LTL Model Checking
   - Büchi Automata
   - Automata-Theoretic Explicit LTL Model Checking

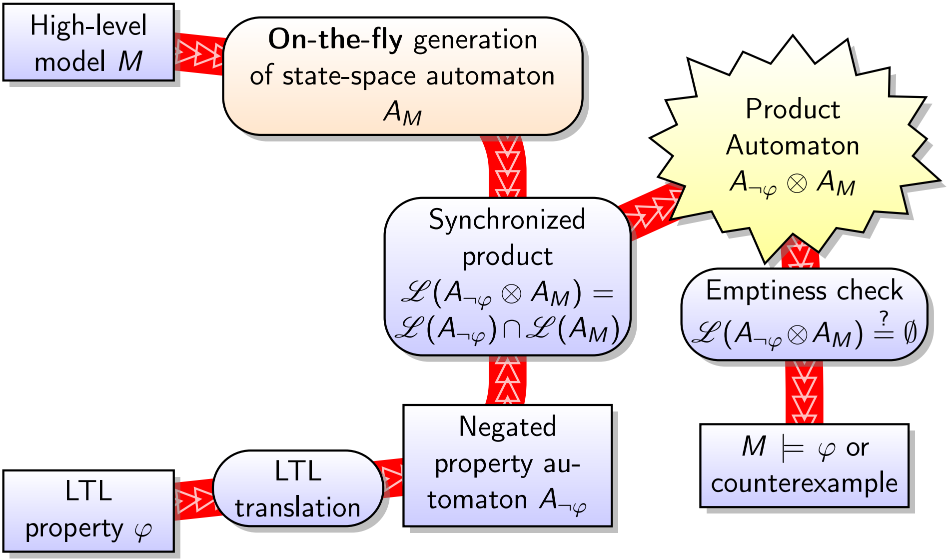
Automata-Theoretic Explicit LTL Model Checking



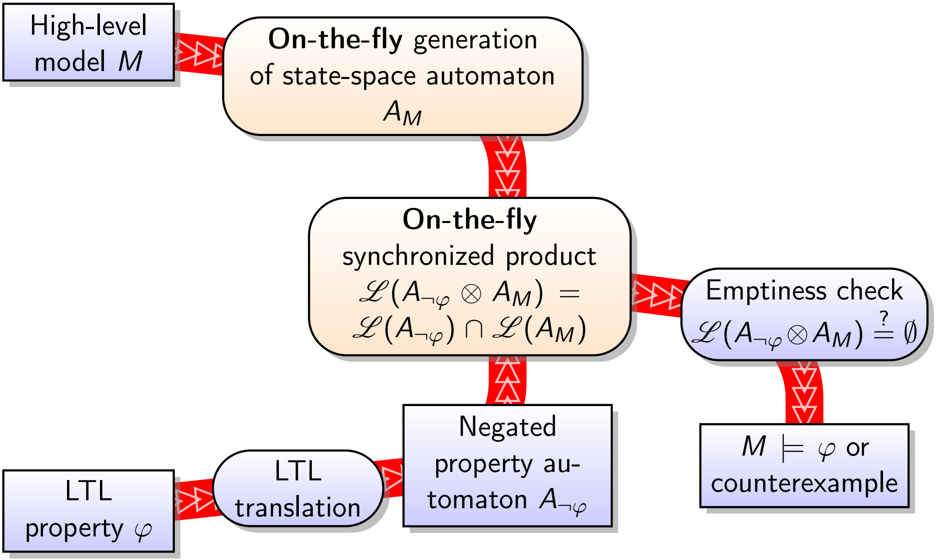
Automata-Theoretic Explicit LTL Model Checking



Automata-Theoretic Explicit LTL Model Checking

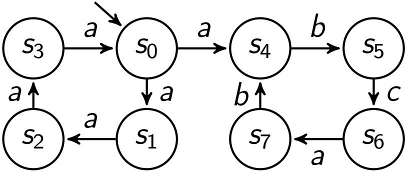


Automata-Theoretic Explicit LTL Model Checking

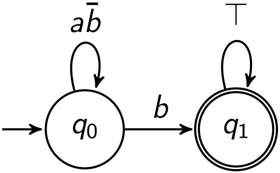


LTS × Büchi Automaton

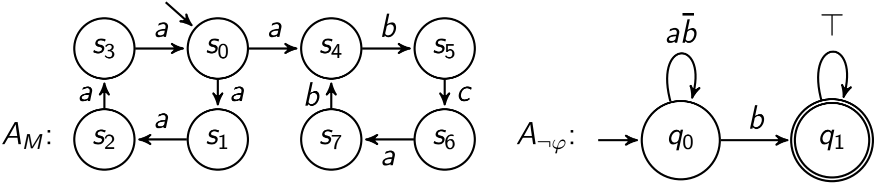
A_M:

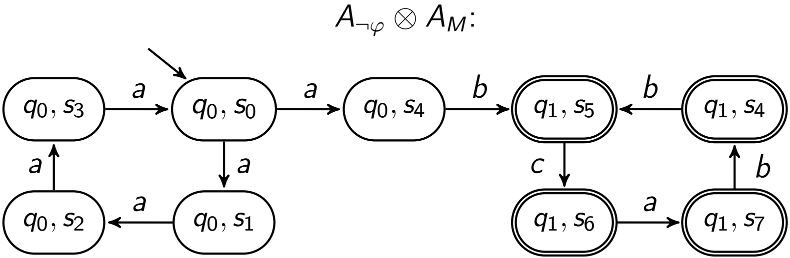


$A_{\neg\varphi}$:

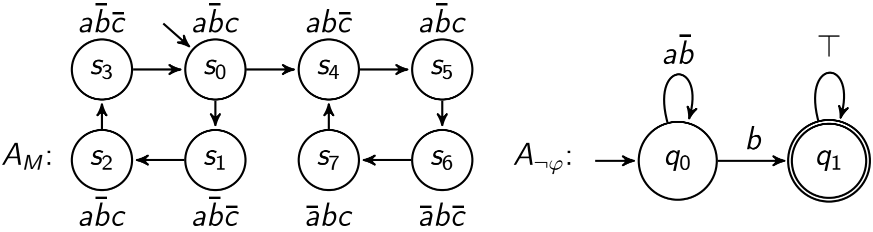


LTS × Büchi Automaton

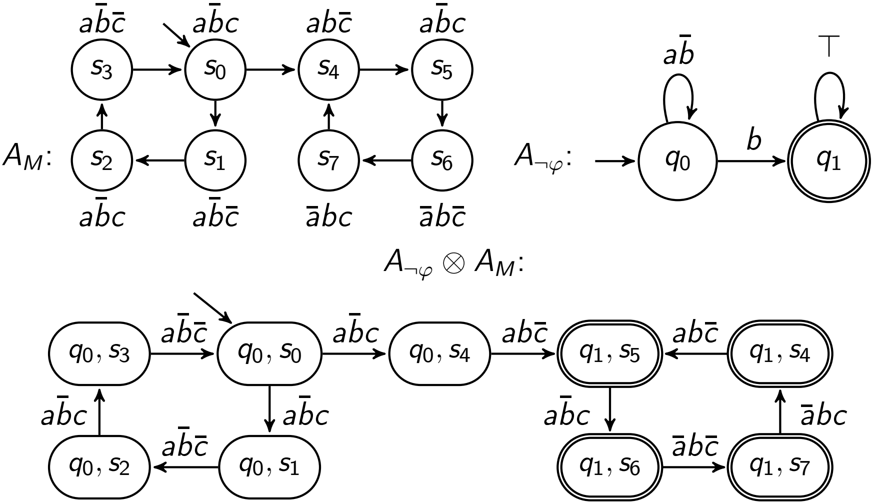




Kripke Srutcure × Büchi Automaton

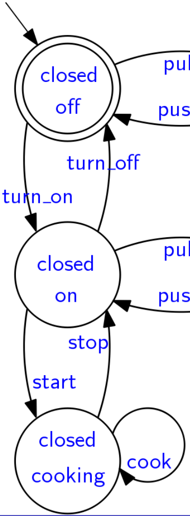


Kripke Srutcure × Büchi Automaton



LTS × Büchi Automaton: Exercice

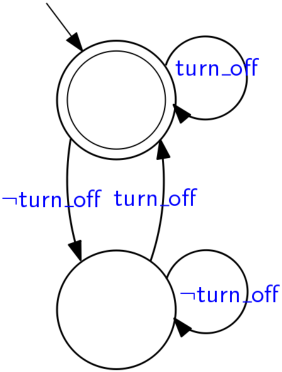
Let us demonstrate by model checking that *G F turn_off* is not satisfied



LTS × Büchi Automaton: Exercic

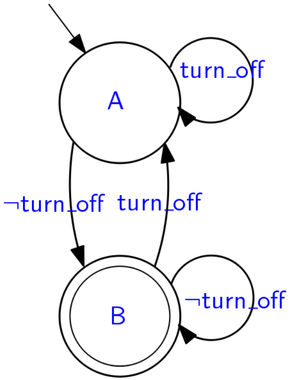
- Build a Büchi automaton with the same language as $\neg(G\ F\ turn\_off)$.
- Let us start from the unnegated formula: $G\ F\ turn\_off$

G F turn_off

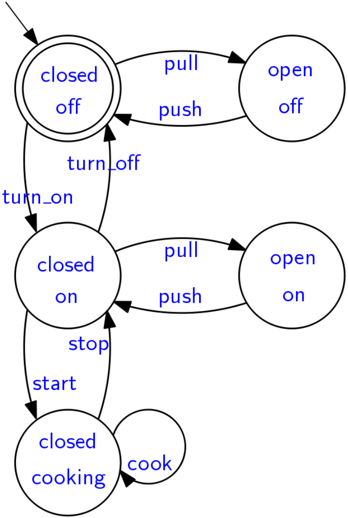


LTS × Büchi Automaton: Exercic

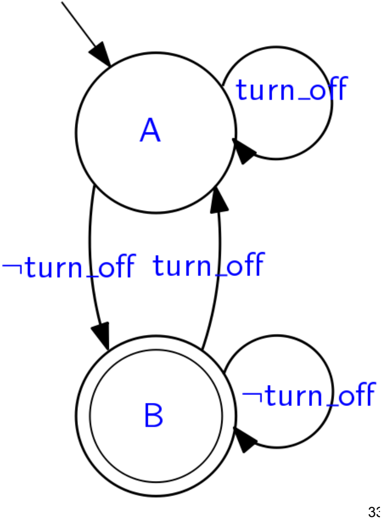
$\neg(G\ F\ turn\_off)$



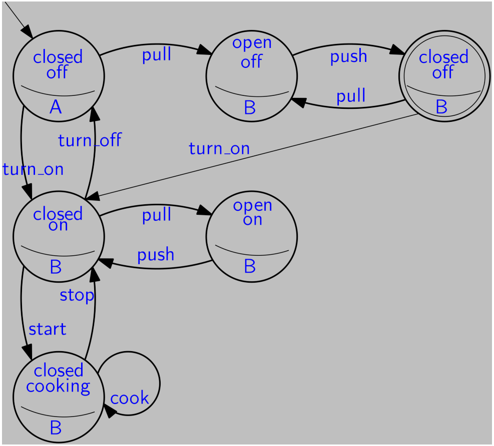
LTS × Büchi Automaton



×



LTS × Büchi Automaton



Kripke Structure × Büchi Automaton: Exercice

Express (with LTL) and Check the three properties of the mutual exclusion Petri net model

$m_0 = p_1 + rp + rq + q_1$
$m_1 = p_2 + rq + q_1$
$m_2 = p_3 + rq + q_1$
$m_3 = p_3 + q_2$
$m_4 = p_3 + q_2$
$m_5 = p_1 + rp + q_2$
$m_6 = p_1 + rp + q_3$
$m_7 = p_2 + q_2$