Computational approaches to analyze complex dynamic systems: model-checking and its applications.

Part 1: Model-checking of discrete event systems

Morgan Magnin
morgan.magnin@irccyn.ec-nantes.fr
www.morganmagnin.net

NII - *Inoue Laboratory*
École Centrale de Nantes - IRCCyN - *MeForBio team*

Lecture Series - Lecture 1 - 2013/03/21

Overview

Vitæ

2004	Engineer diploma in "Computer Science"
2004	Master thesis in "Automatics and Applied Computer Science"
2007	PhD thesis in "Automatics and Applied Computer Science"École Centrale de Nantes / Université de Nantes $\Rightarrow$ **model-checking**, **dynamic systems**, **expressivity between models**
Since 2008:	Associate professor in Computer Science at École Centrale de Nantes / MeForBio research team at IRCCyN $\Rightarrow$ **bioinformatics**, **large-scale analysis**, **parameters inference**
Since 2010:	E-Learning Project Manager at École Centrale de Nantes

A few words about ICT for Education

Information and Communication Technologies Applied to Education

- An emerging **cross-disciplinary** research topic in our institution
- Several papers and projects about:
  - Skills **assessment** (e.g. Script Concordance Tests)
  - Integration and impact of mobiles devices in the learning environment (adaptation, motivation, ...)
  - **Massive Online Open Courses** (1st French MOOC, ITyPA: 1.400 students)
- **International** collaborations:
  - 2008-2010: HP
  - 2008-2011: OpenOffice.org Education project
  - Since 2009: University of Toronto (co-supervision with Karen Reid of the MarkUs software)
- Co-advisor of 1 PhD Student: Augmented learning in science and engineering

A few words about ICT for Education

E-learning group (ECN)

Morgan MAGNIN
Ass. prof. (CS) & project manager

Francisco CHINESTA
Professor (mechanics)

Guillaume MOREAU
Professor (CS)

Christine EVAIN
Associate professor (English)

Anne-Celine GROLLEAU
Research engineer

Simon CAROLAN
1st year PhD student

Main research activities: Bio-informatics

MeForBio (IRCCyN team, ECN): Formal Methods for Bioinformatics

Research axes

- Models: automata, Petri nets, process algebra ($\rightarrow$ process hitting)
- Extended with: time (**chronometry** vs **chronology**) and/or parameters
- Analysis techniques: model-checking, control, abstraction, parameters inference
- Applied (and/or designed) to biology, e.g. biological regulatory networks

Main research activities: Bio-Informatics

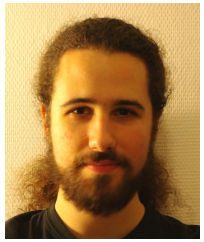
MeForBio (IRCCyN team, ECN): Formal Methods for Bioinformatics

Olivier ROUX
Professor & team leader

Morgan MAGNIN
Associate professor

Carito GUZIOLOWSKI
Associate professor

Julien GRAS
Research engineer



Maxime FOLSCHETTE
2nd year PhD student

Courtney CHANCELLOR
1st year PhD student

Overview

Introduction (1/2)

Real-time system

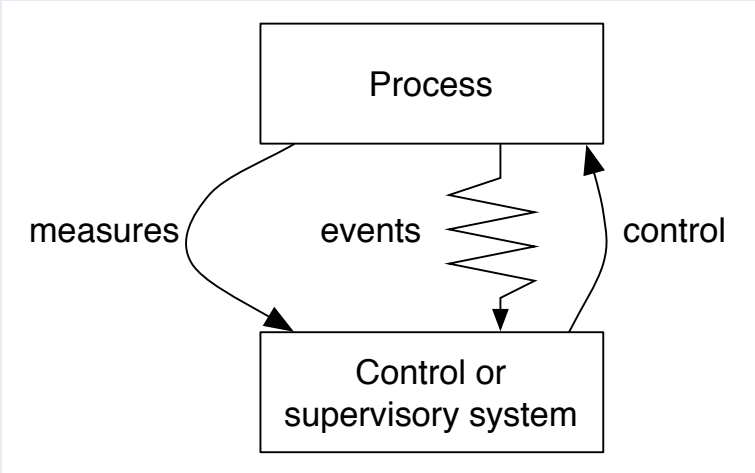


Figure: Analyzed system: { process + control system }

Introduction (1/2)

Real-time system

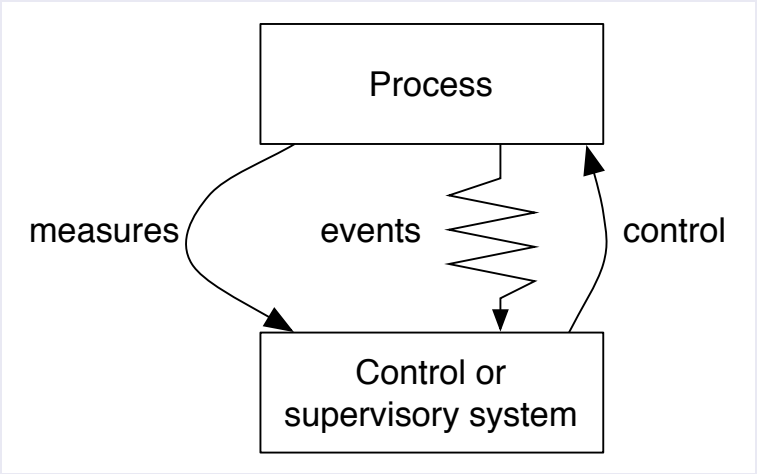


Figure: Analyzed system: { process + control system }

Formal verification of *real-time* systems

- Embedded systems (cars, planes, ...)
- Logistics (systems engineering)
- **Systems biology** (gene regulatory networks)

But still, **why** do we want to perform formal verification?

But still, **why** do we want to perform formal verification?

Figure: Ariane 5 - Flight 501

But still, **why** do we want to perform formal verification?

The two most well-known computer bugs *(at least, for the model-checking community)*

- Crash of the european Ariane 5 rocket in 1996
  - **Exploded 39 seconds** after takeoff
  - Cost 370 millions US$
  - **Overflow** in a conversion of a 64 bit floating point number into a 16 bit integer data conversion from a 64-bit floating
  - Software from Ariane 4 was re-used for Ariane 5 **without re-testing**
  - `http://en.wikipedia.org/wiki/Ariane_5_Flight_501`
- Pentium FDIV bug, 1994.
  - Flaw in the **division algorithm**
  - Cost an average 500 million US$ and image loss for Intel
  - `http://en.wikipedia.org/wiki/Pentium_FDIV_bug`

But still, **why** do we want to perform formal verification?

The two most well-known computer bugs *(at least, for the model-checking community)*

- Crash of the european Ariane 5 rocket in 1996
  - **Exploded 39 seconds** after takeoff
  - Cost 370 millions US$
  - **Overflow** in a conversion of a 64 bit floating point number into a 16 bit integer data conversion from a 64-bit floating
  - Software from Ariane 4 was re-used for Ariane 5 **without re-testing**
  - `http://en.wikipedia.org/wiki/Ariane_5_Flight_501`
- Pentium FDIV bug, 1994.
  - Flaw in the **division algorithm**
  - Cost an average 500 million US$ and image loss for Intel
  - `http://en.wikipedia.org/wiki/Pentium_FDIV_bug`

But still, **why** do we want to perform formal verification?

Emergence of **model-checking**

- To automatically find **bugs** and prove **correctness** of *models*
- Edmund Clarke, E. Allen Emerson and Joseph Sifakis: Türing Award in 2007

Motivations

Objective: formal verification of properties

- Model the system S :
  - Discrete models: finite state automata, Petri nets, ... $\Rightarrow$ **Lecture 1**
  - Timed models:
    - timed extensions of finite state automata: timed/hybrid automata $\Rightarrow$ Lecture 2
    - timed extensions of Petri nets: time/stopwatch Petri nets $\Rightarrow$ Lecture 3
- Formalize the specification φ :
  - Observers
  - Temporal logics: LTL, CTL, ITL, ... $\Rightarrow$ **Lecture 1**
  - Timed extensions of temporals logics: TCTL, ... $\Rightarrow$ Lectures 2 & 3
- Does $S \models \varphi$?

Model-checking algorithms

$\Rightarrow$ State space exploration

Motivations

Objective: formal verification of properties

- Model the system S :
  - Discrete models: finite state automata, Petri nets, ... $\Rightarrow$ **Lecture 1**
  - Timed models:
    - timed extensions of finite state automata: timed/hybrid automata $\Rightarrow$ Lecture 2
    - timed extensions of Petri nets: time/stopwatch Petri nets $\Rightarrow$ Lecture 3
- Formalize the specification φ :
  - Observers
  - Temporal logics: LTL, CTL, ITL, ... $\Rightarrow$ **Lecture 1**
  - Timed extensions of temporals logics: TCTL, ... $\Rightarrow$ Lectures 2 & 3
- Does $S \models \varphi$?

Model-checking algorithms

$\Rightarrow$ State space exploration

Some major issues

Need for modeling tasks with suspending/resuming features

Expressivity/**Decidability** compromise to discuss ⇒ Lectures 2 & 3

State space combinatorial explosion

- Need for symbolic approaches ⇒ Lectures 2 & 3
- Need for new models and abstracted algorithms ⇒ Lecture 4

Some major issues

Need for modeling tasks with suspending/resuming features

Expressivity/**Decidability** compromise to discuss ⇒ Lectures 2 & 3

State space combinatorial explosion

- Need for symbolic approaches ⇒ Lectures 2 & 3
- Need for new models and abstracted algorithms ⇒ Lecture 4

Overview

How to compare (and choose between) different models?

Theoretical criteria

- **Decidability** results w.r.t. common problems (e.g. state reachability, liveness), *but* ...
- Respective **expressivity**

Pragmatical criteria

- Ease to **understand** and **build** models
- What are the **key features** (e.g. modularity, concurrence, chronology) the model has to capture?
- Existence of **software tools** to design and analyze these models
- **History**

Why Petri nets?

Motivation

- Mathematical **and** graphic formalism
- Easy representation **concurrence and parallelism**
- **Structural** properties (P-invariants, T-invariants, ...)
- **Dynamical** properties (liveness, boundedness, reachability, ...)
- Mature tools : Snoopy, ginSIM, Roméo, etc.

Petri nets: a wide range of models

- Discrete [SBSW07]
- Continuous [KH08]
- Hybrid [MDNM00]
- Stochastic [GP98]: the firing of a transition relates to a probability function, which corresponds to the chemical sensitization depending on the concentration
- Colored [GKP10] : tokens are differentiated
- Time and stopwatches: [MMR09]

Petri net - Introduction

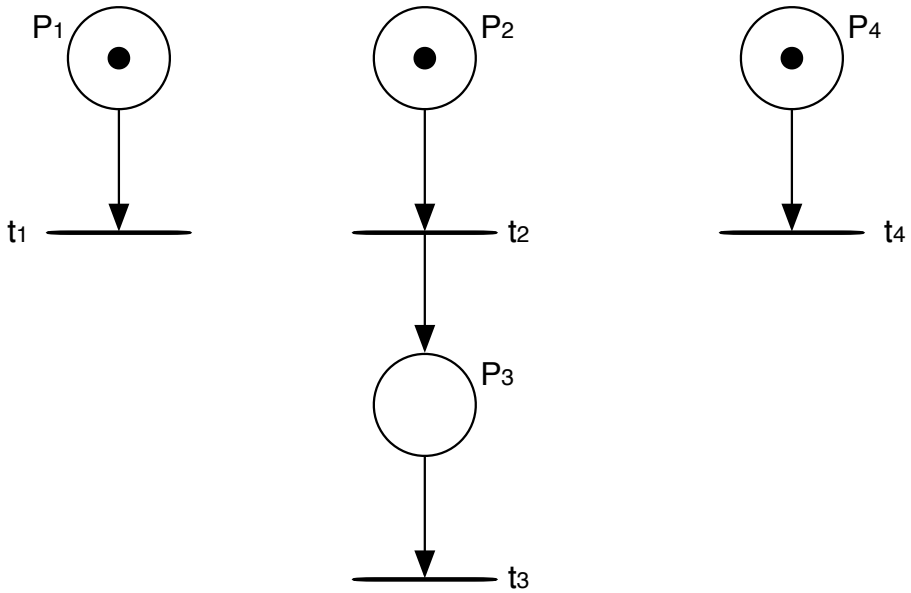


Figure: A Petri net

$$\{P_1, P_2, P_4\}$$

Petri net - Introduction

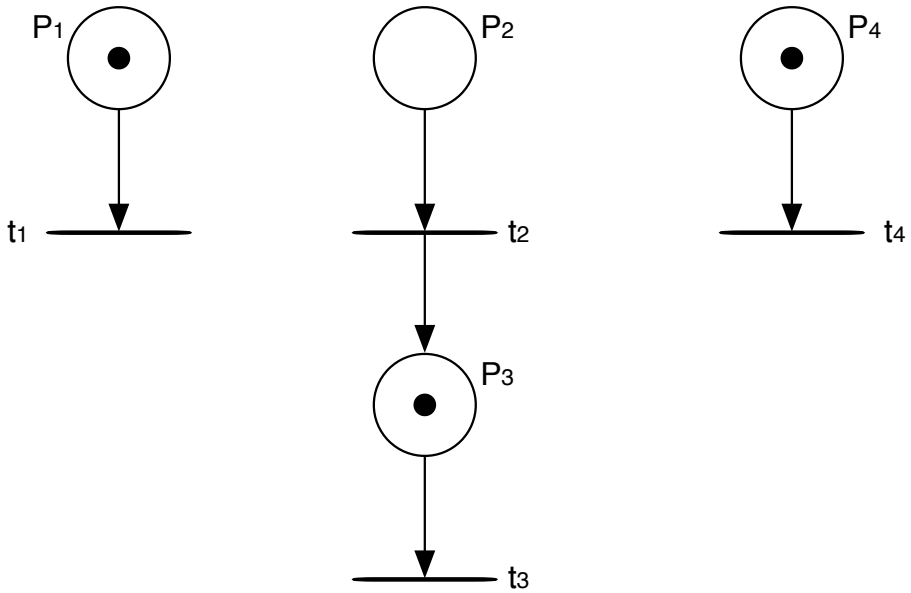


Figure: A Petri net

$$\{P_1, P_2, P_4\} \xrightarrow{t_2} \{P_1, P_3, P_4\} \xrightarrow{t_1} \dots$$

Petri nets - Introduction

Definition

- A set of places
- A set of transitions
- A backward incidence function
- A forward incidence function,
- *An initial state*

Petri nets - Introduction

Basics

- Non-deterministic execution
- Multiple tokens make PN be suitable to modeling the concurrent behavior of distributed systems

Petri nets - Properties

Some structural properties

- **T-invariant** : sequence of transitions which returns in the same state / marking
- **P-invariant** : marking invariant (e.g. $q_i M(p_i) + q_j M(p_j) + q_k M(p_k) = c$ for any state of the net)

Petri nets - Introduction

Some dynamic properties

- **Liveness**: in every state, every transition will eventually be fired
- **Deadlock**: state in which no more transition can be fired
- Marking **reachability** (given an initial state)

Petri net - Liveness

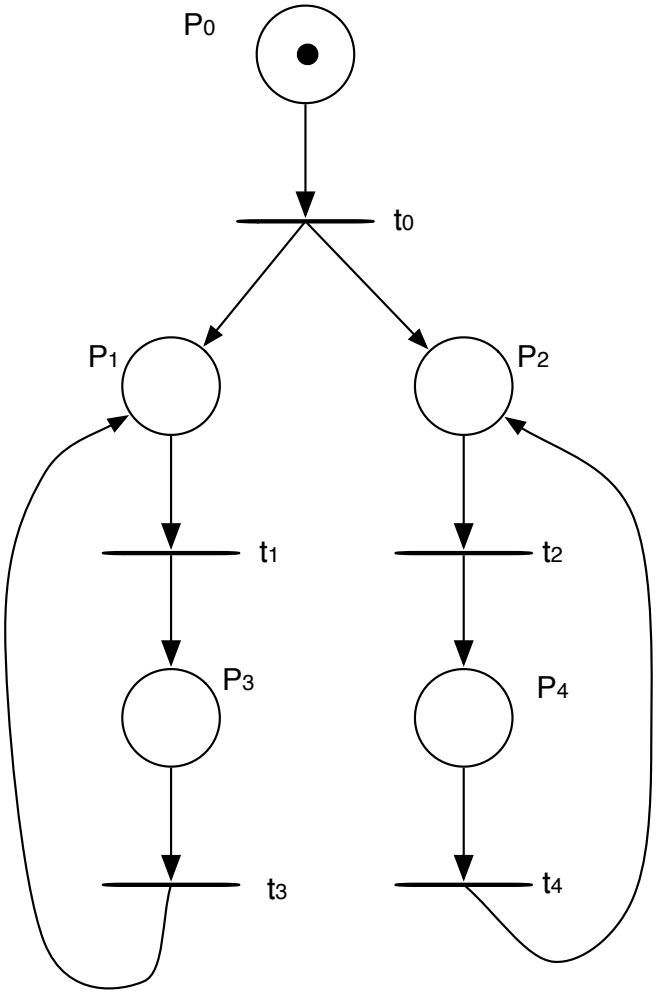


Figure: Is it a live Petri net?

Petri net - Liveness

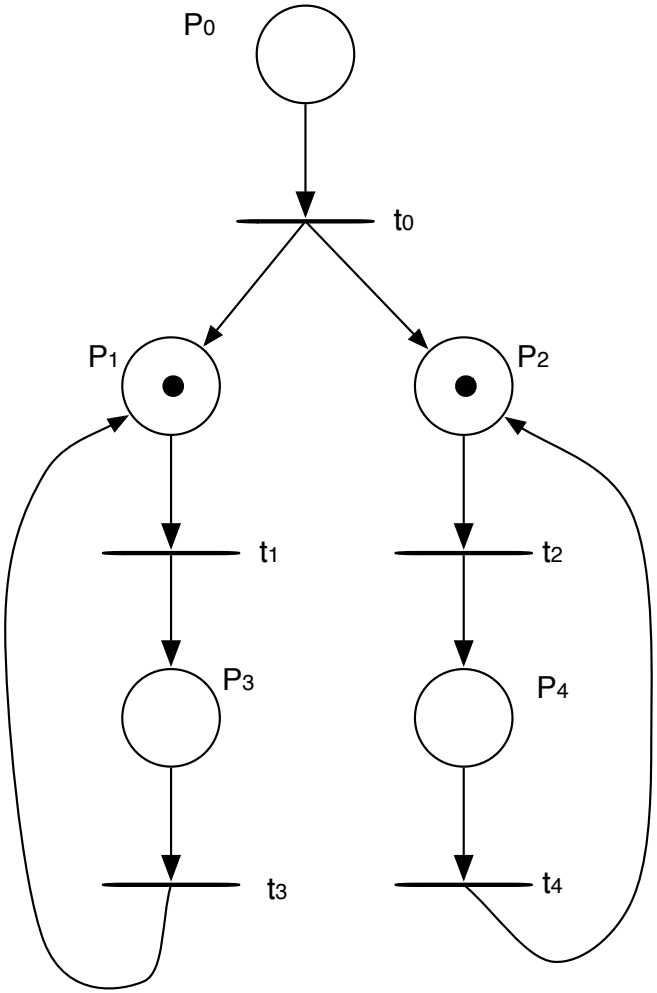


Figure: Is it a live Petri net?

Petri net - Liveness

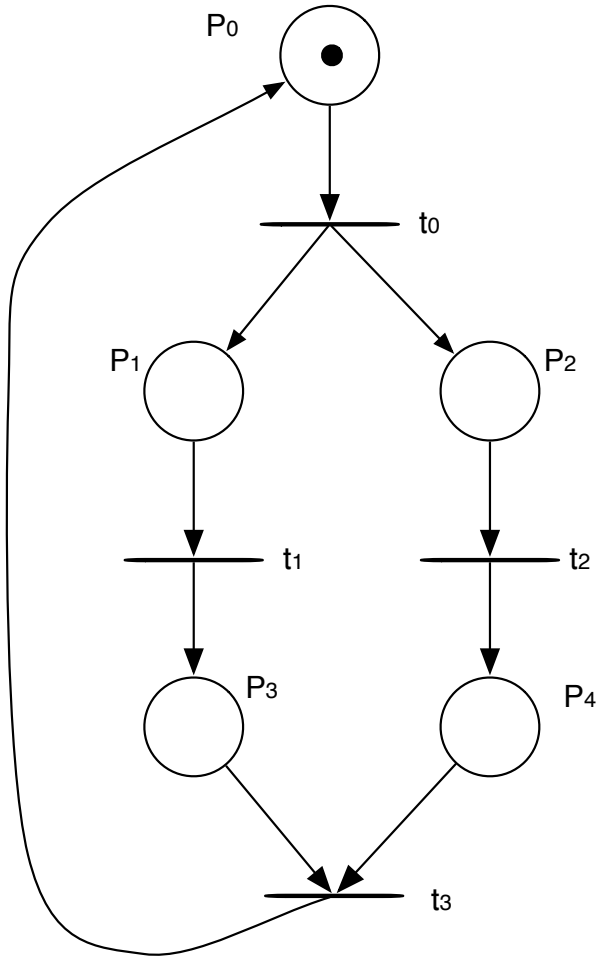


Figure: Is it a live Petri net?

Petri net - Liveness

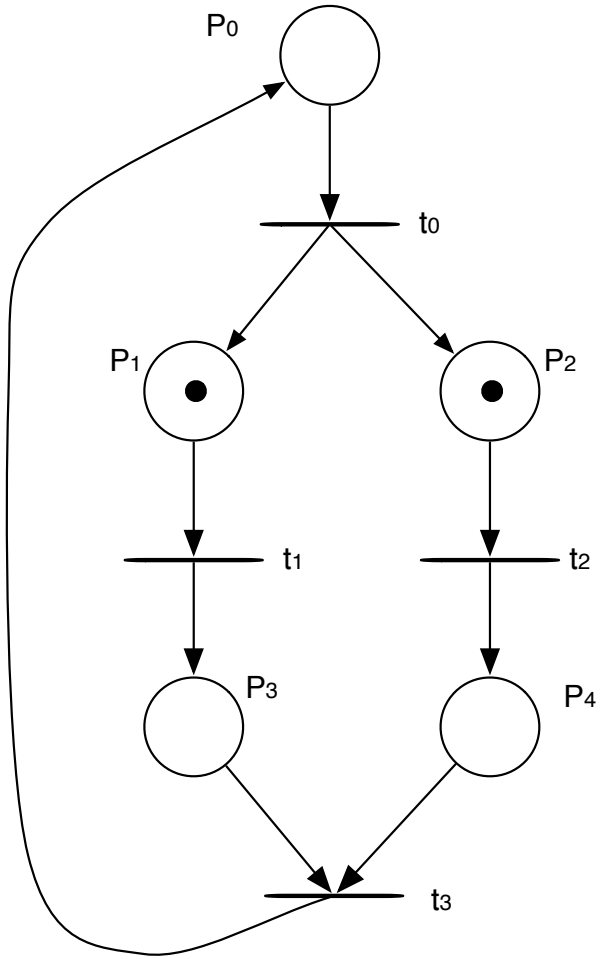


Figure: Is it a live Petri net?

Petri net - Liveness

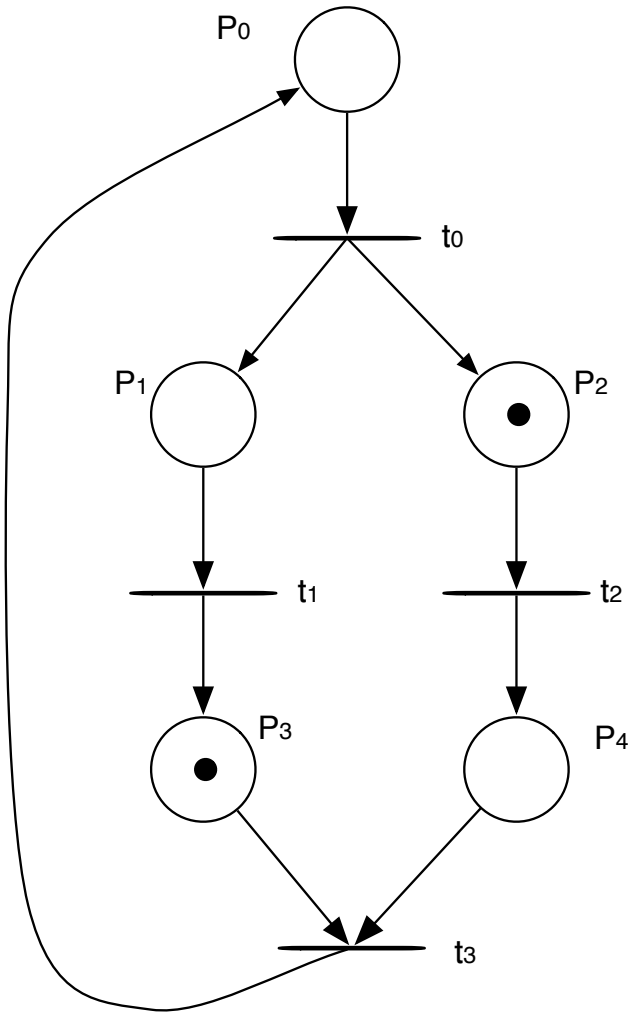


Figure: Is it a live Petri net?

Petri net - Liveness

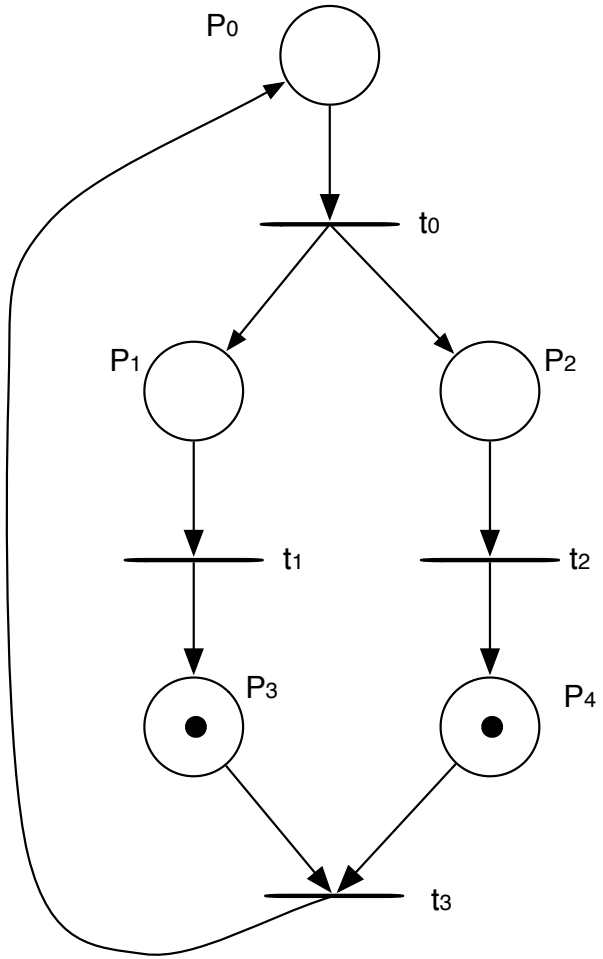


Figure: Is it a live Petri net?

Petri net - Liveness

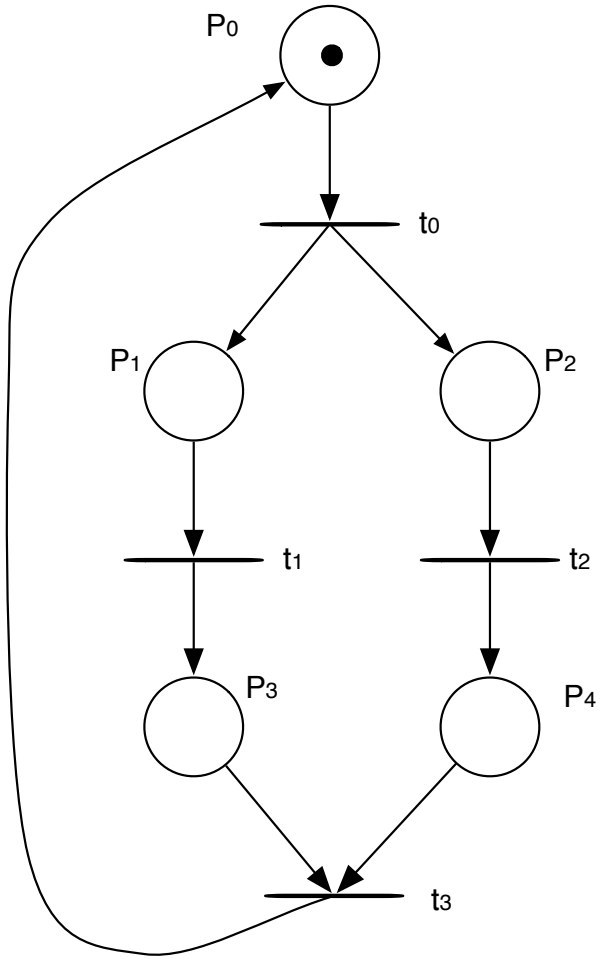


Figure: Is it a live Petri net?

Petri net - Deadlock

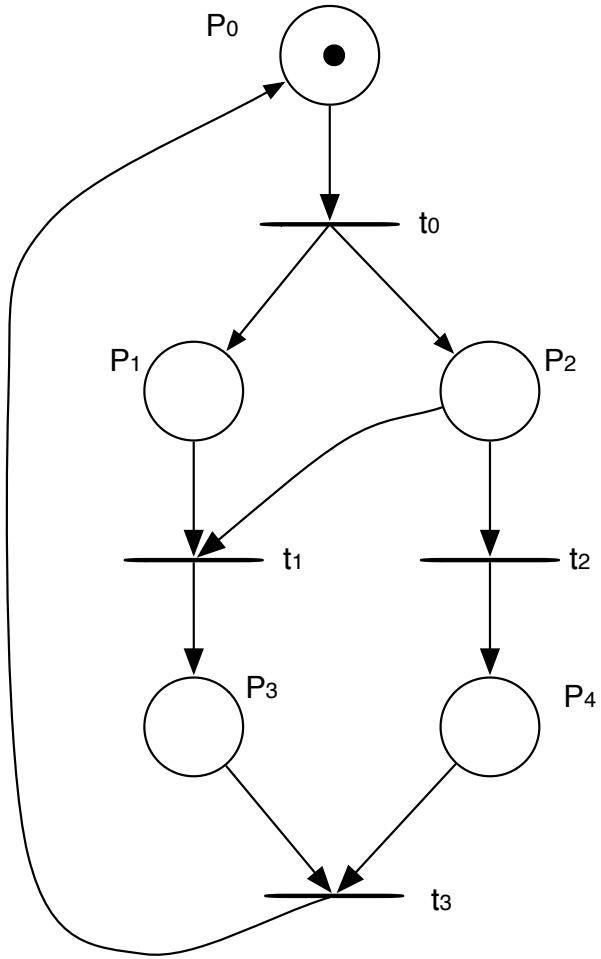


Figure: Will this Petri net lead to a deadlock?

Petri net - Deadlock

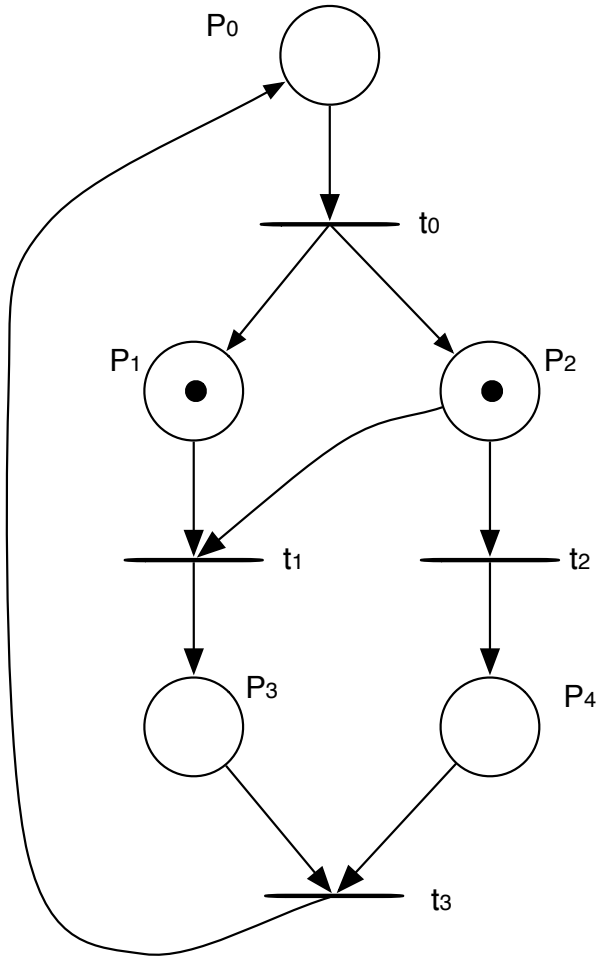


Figure: Will this Petri net lead to a deadlock?

Petri net - Deadlock

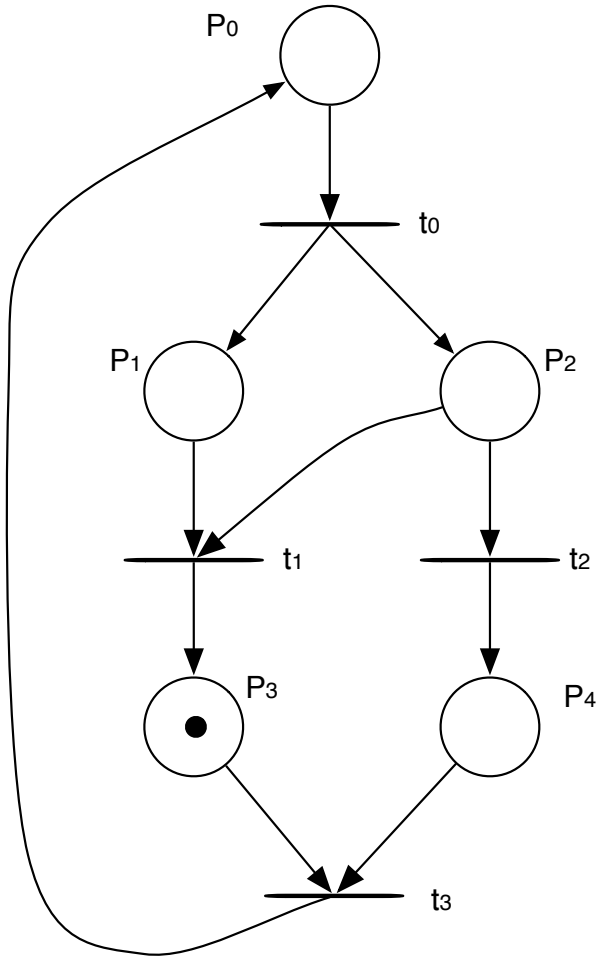


Figure: Will this Petri net lead to a deadlock?

Petri nets - Applications

- Production systems (factories)
- Logistic deployment systems
- Embedded systems
- Video games (A.I. modeling)
- And, of course, biology!

Overview

Short insight on the meaning of Petri net components

Intuitive meaning

- Marking of a place: **presence/absence** or **quantity** of a component
- Arc : **precedence** or **succession**
- Transition : **event** andor **transformation**
- Weight : necessary **quantity**, consumed and/or produced

Petri nets for modeling biochemical networks

Principle of qualitative modeling

- Places : reactants, products, enzymes
- Transitions : reactions, catalysis
- Weight on the arcs: stoichiometry

Application to the modeling of biochemical reactions

$$2NAD^+ + 2H_2O \rightarrow 2NADH + 2H^+ + O_2$$

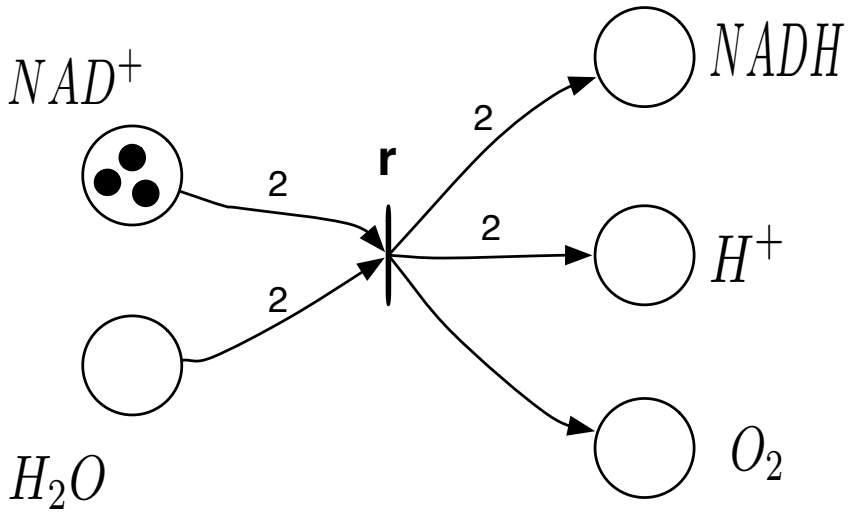


Figure: An example of a very simple translation from a biochemical reaction

Petri nets properties for the modeling of biochemical networks

Structural properties

- Incidence matrix: stoichiometry matrix
- P-invariants: conservation relations
- T-invariants: elementary flux modes

Dynamic properties

- Liveness: components are sufficient to trigger reactions
- Deadlock: stable state

Additional modeling features

Critical issues

- How to **test** the number of tokens in a place (e.g. concentration level of a protein) without decrementing it?
- How to model an action that takes place only **below** a given threshold (e.g. concentration level of a protein)?
- How to reset the number of tokens in a place (e.g. reset of a system) whatever was its previous concentration?

$\rightarrow$ Introduction of new arcs

Petri net - Enriching the expressivity of the model

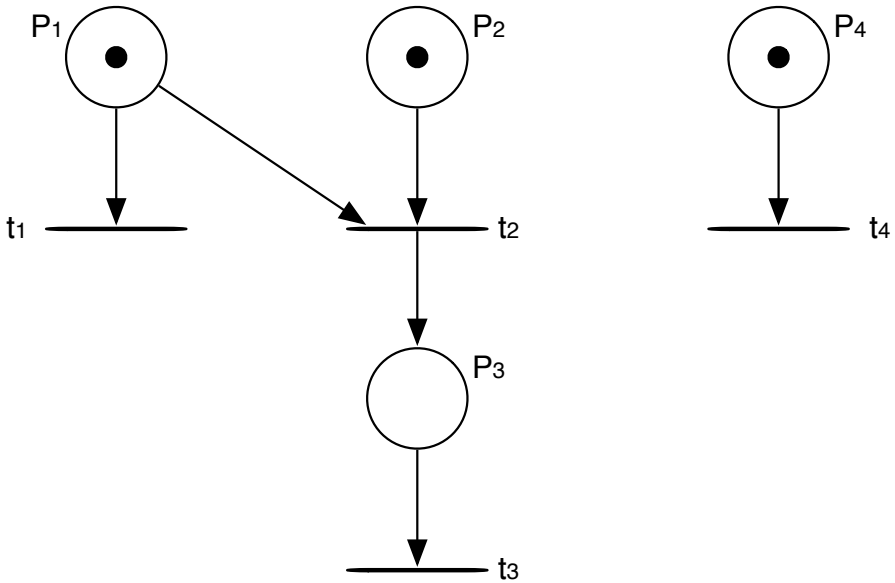


Figure: An other Petri net

$$\{P_1, P_2, P_4\}$$

Petri net - Enriching the expressivity of the model

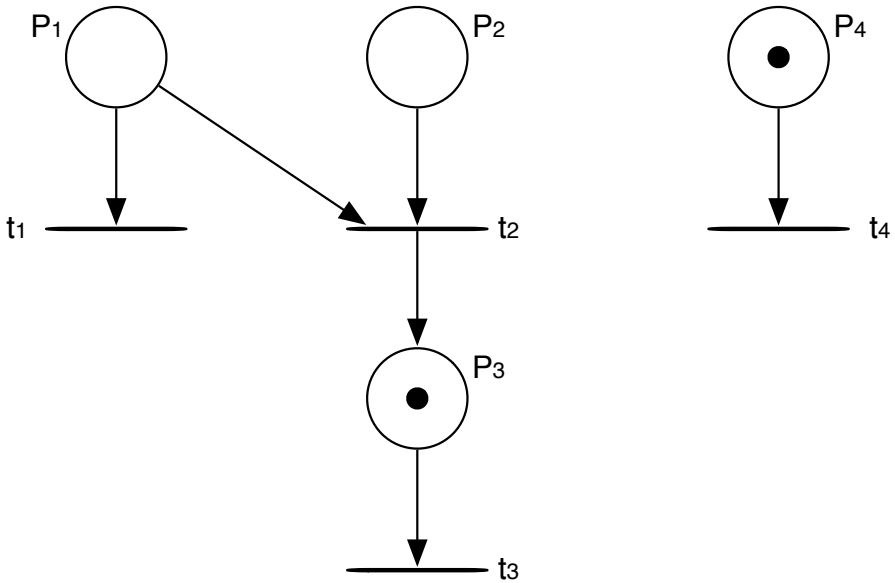


Figure: An other Petri net

$$\{P_1, P_2, P_4\} \xrightarrow{t_2} \{P_3, P_4\} \dots$$

Petri net with reset arcs - Introduction

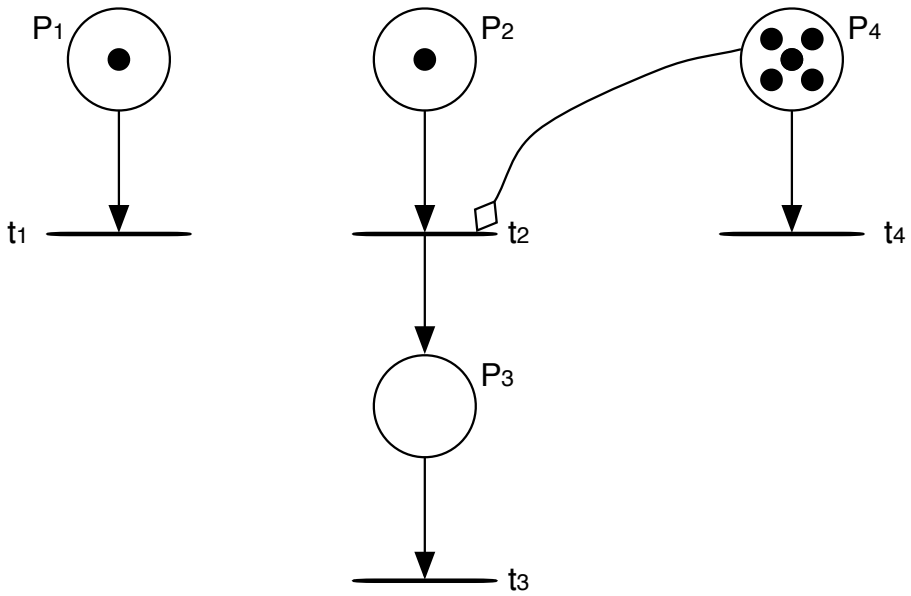


Figure: A Petri net with *reset* arcs

$$\{P_1, P_2, 5 \times P_4\} \xrightarrow{t_2} \dots$$

Petri net with reset arcs - Introduction

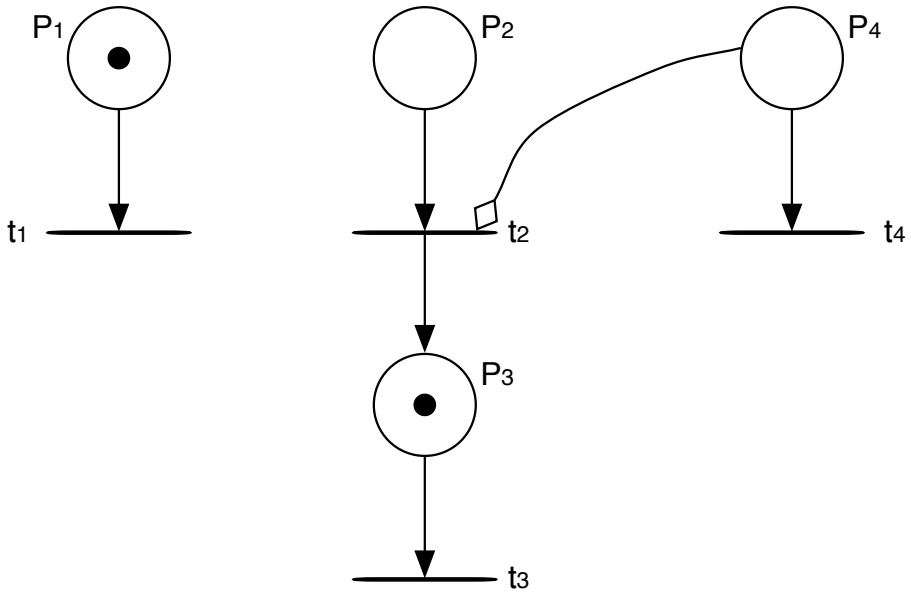


Figure: A Petri net with *reset* arcs

$$\{P_1, P_2, 5 \times P_4\} \xrightarrow{t_2} \{P_1, P_3\} \xrightarrow{t_1} \dots$$

Petri nets with read arcs

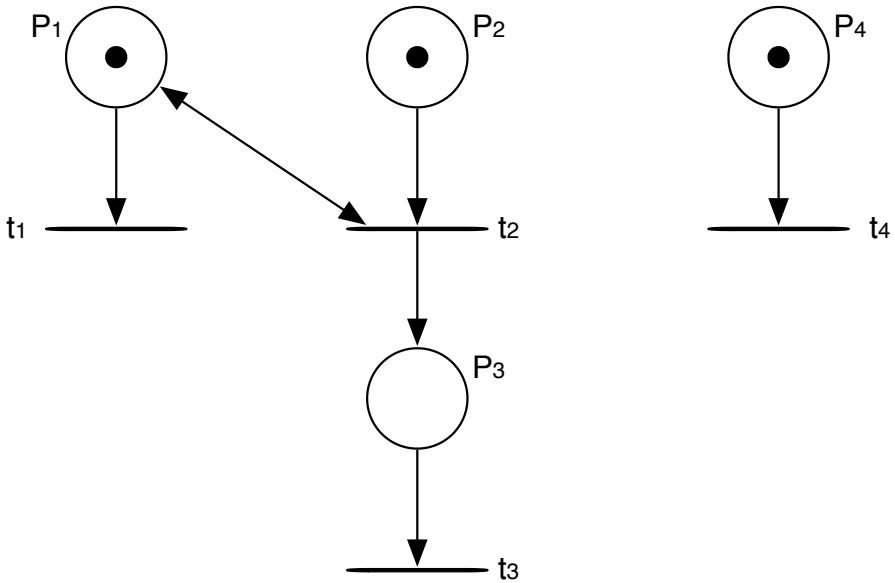


Figure: A Petri net with read arcs

$$\{P_1, P_2, P_4\}$$

Petri nets with read arcs

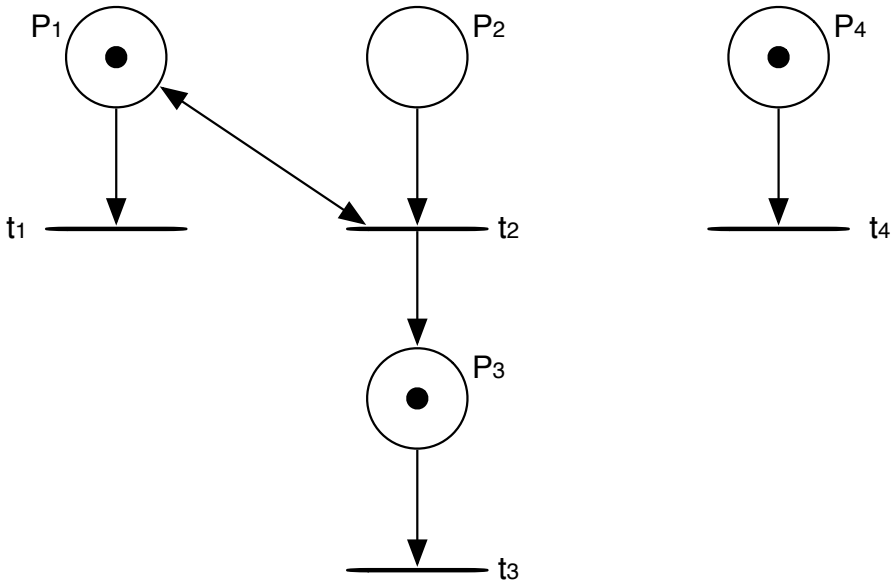


Figure: A Petri net with read arcs

$$\{P_1, P_2, P_4\} \xrightarrow{t_2} \{P_1, P_3, P_4\} \ldots$$

(Logic) Inhibitor Arc Petri nets: OR inhibition

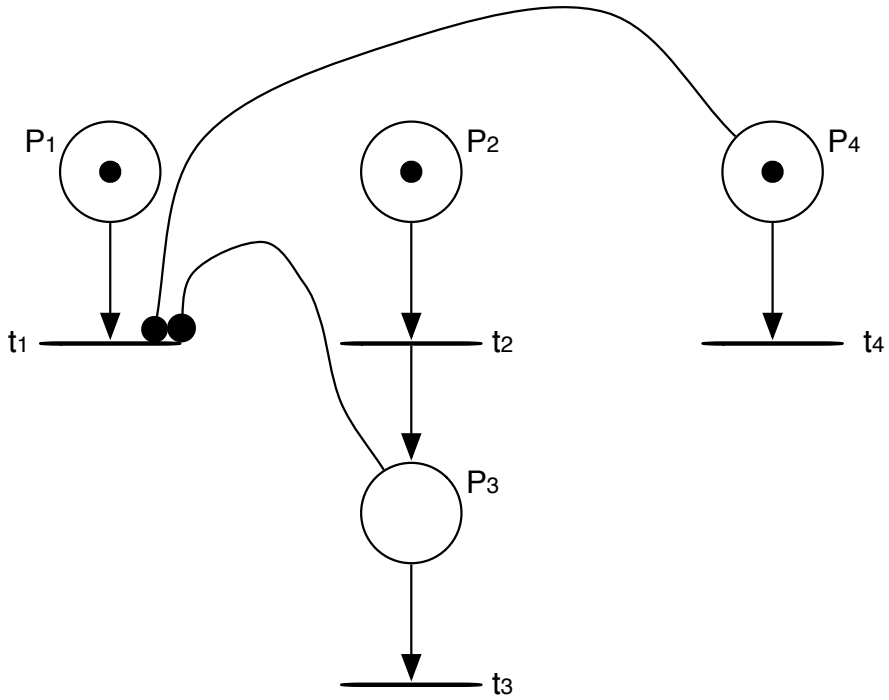


Figure: An inhibitor arc Petri net

$$\{P_1, P_2, P_4\}$$

(Logic) Inhibitor Arc Petri nets: OR inhibition

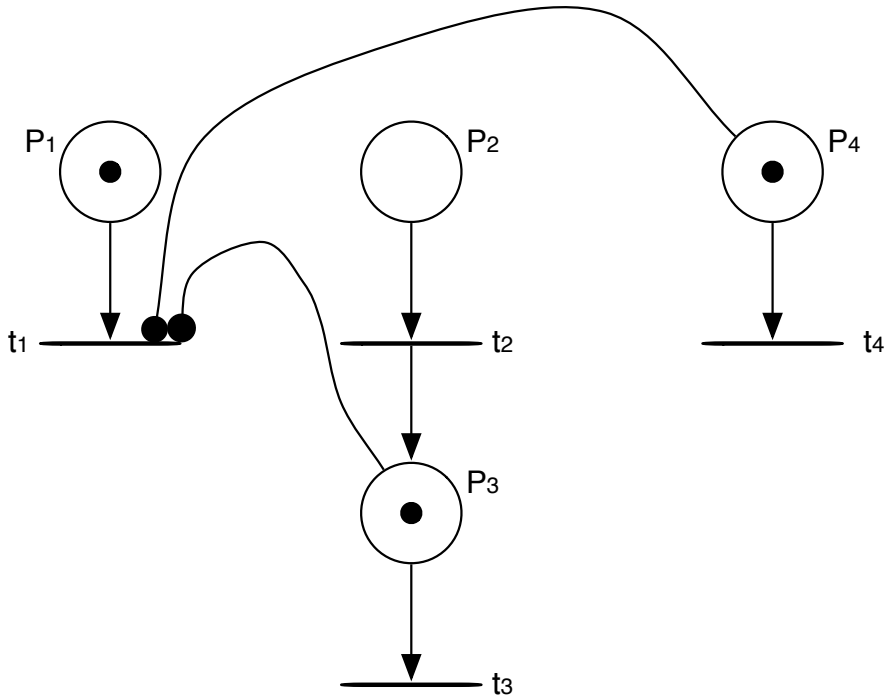


Figure: An inhibitor arc Petri net : t_1 inhibited iff $(M(P_3) \geq 1$ **or** $M(P_4) \geq 1)$

$$\{P_1, P_2, P_4\} \xrightarrow{t_2} \{P_1, P_3, P_4\}$$

(Logic) Inhibitor Arc Petri nets: OR inhibition

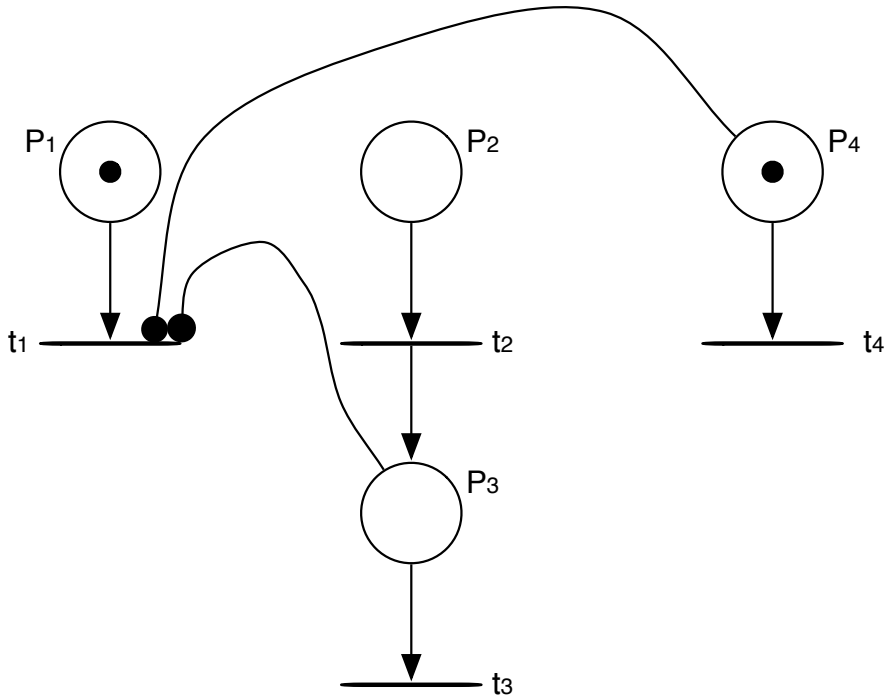


Figure: An inhibitor hyperarc Petri net

$$\{P_1, P_2, P_4\} \xrightarrow{t_2} \{P_1, P_3, P_4\} \xrightarrow{t_3} \{P_1, P_4\} \xrightarrow{t_4} \dots$$

(Logic) Inhibitor Hyperarc Petri nets: AND inhibition

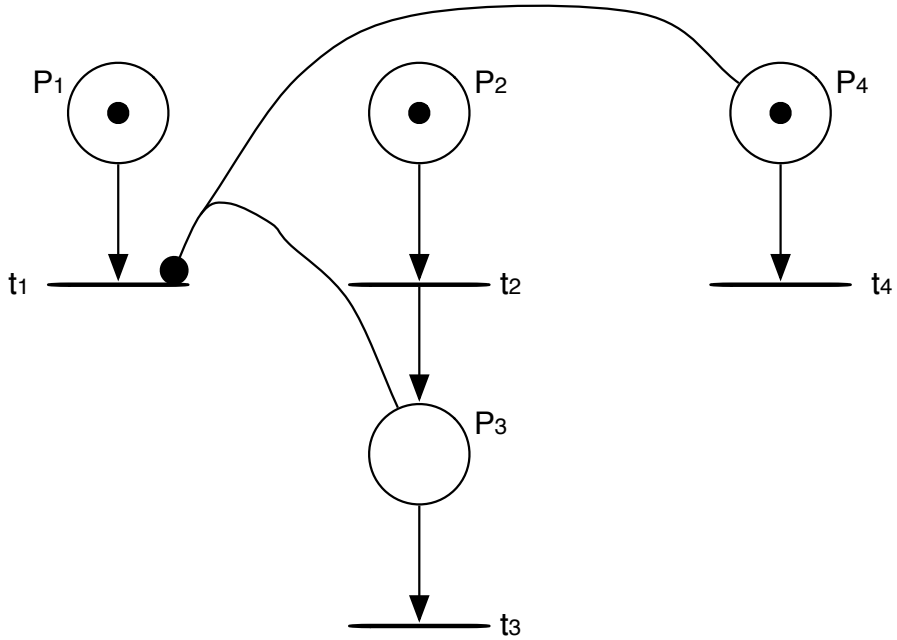


Figure: An inhibitor hyperarc Petri net

$$\{P_1, P_2, P_4\}$$

(Logic) Inhibitor Hyperarc Petri nets: AND inhibition

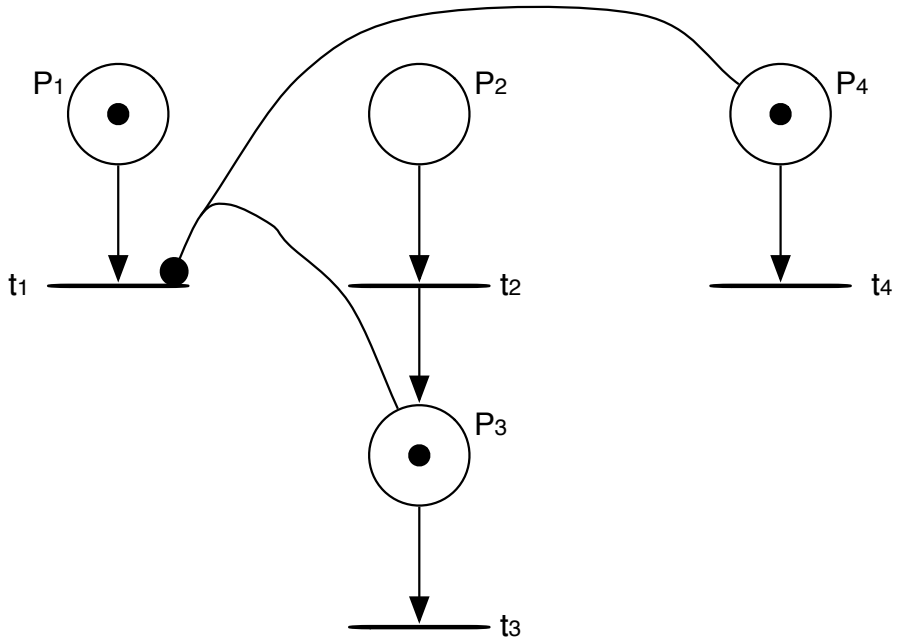


Figure: An inhibitor hyperarc Petri net : t_1 inhibited iff ($M(P_3) \geq 1$ **and** $M(P_4) \geq 1$)

$$\{P_1, P_2, P_4\} \xrightarrow{t_2} \{P_1, P_3, P_4\}$$

(Logic) Inhibitor Hyperarc Petri nets: AND inhibition

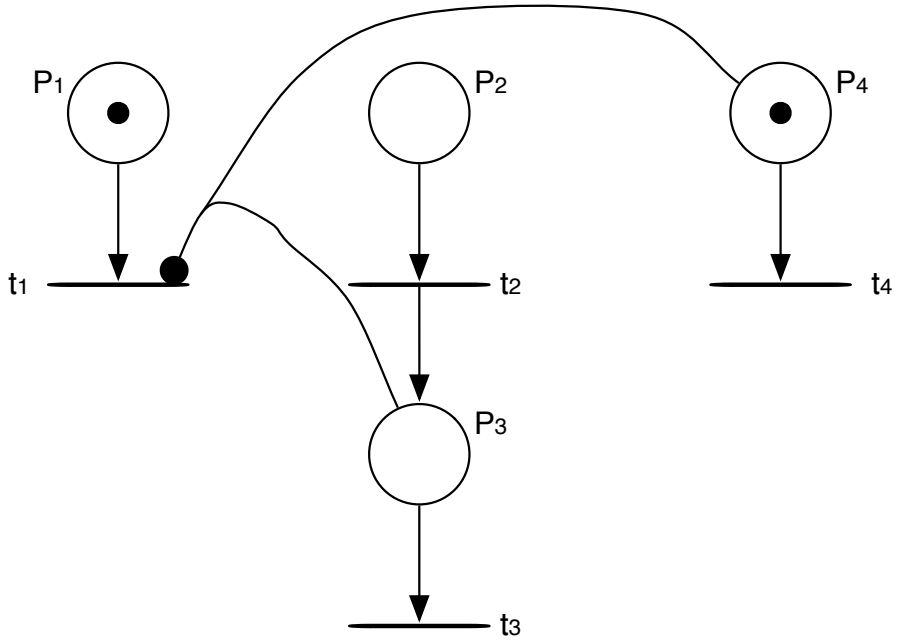


Figure: An inhibitor hyperarc Petri net

$$\{P_1, P_2, P_4\} \xrightarrow{t_2} \{P_1, P_3, P_4\} \xrightarrow{t_3} \{P_1, P_4\} \xrightarrow{t_1} \dots$$

Implementation

How to automatically compute state space of PNs?

Define **efficient data-structures** designed for:

- Storage of states
- Modeling incidence functions

$\rightarrow$ Get inspiration from *Binary Decision Diagrams* (BDDs)

A word about Binary Decision Diagrams (BDDs)

How to get a compact representation of a boolean function?

- **Compressed** representation of sets
- **Directed acyclic graph**
- To make it **canonical** would allow to make equivalence-checking easier

x1	x2	x3	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figure: Truth table for $f(x_1, x_2, x_3) = \bar{x_1}\bar{x_2}\bar{x_3} + x_1x_2 + x_2x_3$ (from WIKIPEDIA)

A word about Binary Decision Diagrams (BDDs)

How to get a compact representation of a boolean function?

- One root node
- Terminal nodes: 0, 1
- Each non-terminal node has two children and is labeled by a variable
- No variable appears more than once along a path

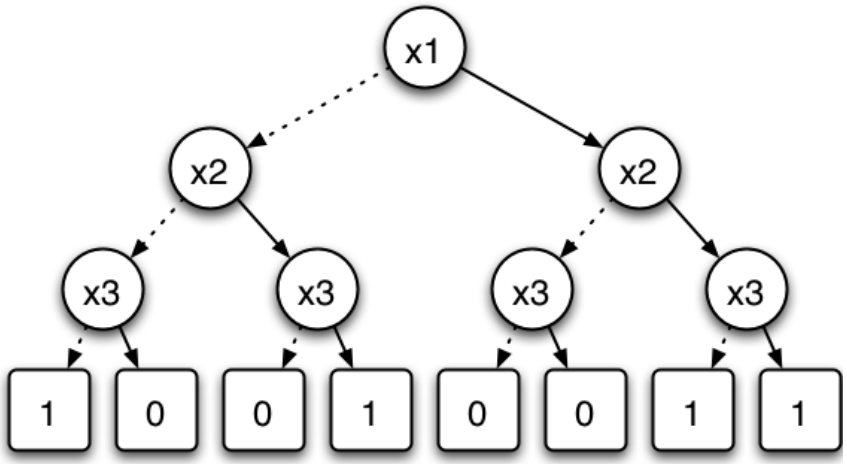


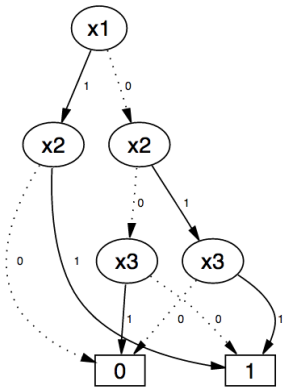
Figure: Binary tree for $f(x_1, x_2, x_3) = \bar{x_1}\bar{x_2}\bar{x_3} + x_1x_2 + x_2x_3$ (from WIKIPEDIA)

A word about Binary Decision Diagrams (BDDs)

How to get a compact and **canonical** representation of a boolean function?

Key ideas formalized by Briant [Bry86] in 1986:

- Variables **ordering**: in all paths, variables appear in the same order
- Reduction :
  - Merge any isomorphic subgraphs
  - Eliminate any node whose two children are isomorphic
- Each non-terminal node has two children and is labeled by a variable



A word about Binary Decision Diagrams (BDDs)

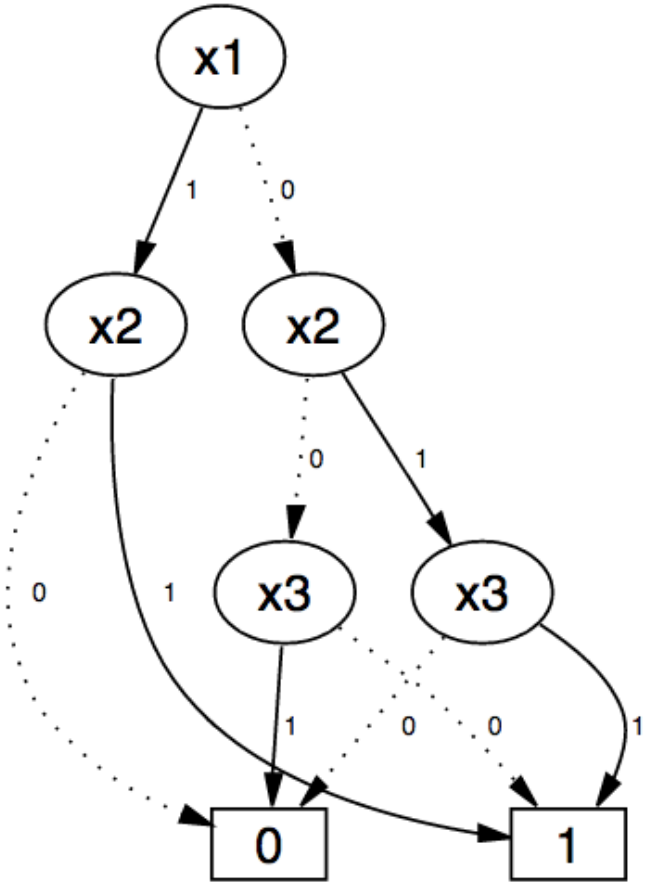


Figure: BDD for $f(x_1, x_2, x_3) = \bar{x_1}\bar{x_2}\bar{x_3} + x_1x_2 + x_2x_3$ (from WIKIPEDIA)

A word about Binary Decision Diagrams (BDDs)

Challenges w.r.t. BDDs

- Size of a BDD **depends** critically on variable ordering
- Finding the *best* variable ordering is NP-hard
- Some efficient **heuristics** still exist!

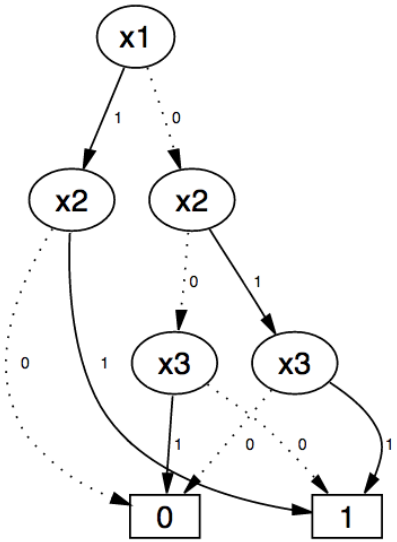


Figure: BDD for $f(x_1, x_2, x_3) = \bar{x_1}\bar{x_2}\bar{x_3} + x_1x_2 + x_2x_3$ (from WIKIPEDIA)

Application of BDD to Petri nets

Illustration of the implementation in the Markg software

- Computes the state space of untimed Petri nets (with **read hyperarcs** and **marking values affectation**)
- Based on data-structures inspired by the *Binary Decision Diagrams* (BDDs) :
  - Vector Decision Diagrams (VDDs) : storage of states;
  - Arithmetic Expression Decision Diagrams (AEDDs) : model incidence functions;
- Deals with potentially infinite markings
- Tackles state space with more than 10^{14} states and 10^{15} transitions
- Download: `http://markg.rts-software.org`

Storage of states: Vector decision diagrams

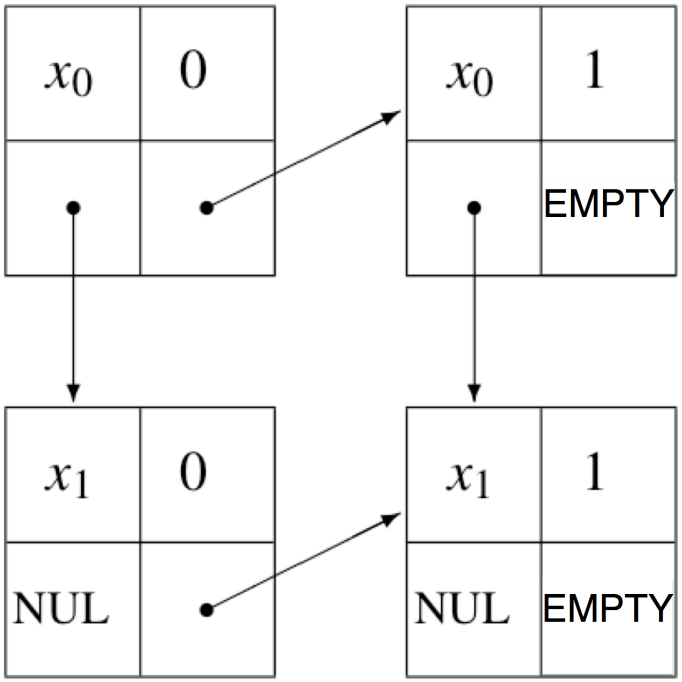


Figure: VDD representing states $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$

Modeling incidence functions: Arithmetic Expression Decision Diagrams

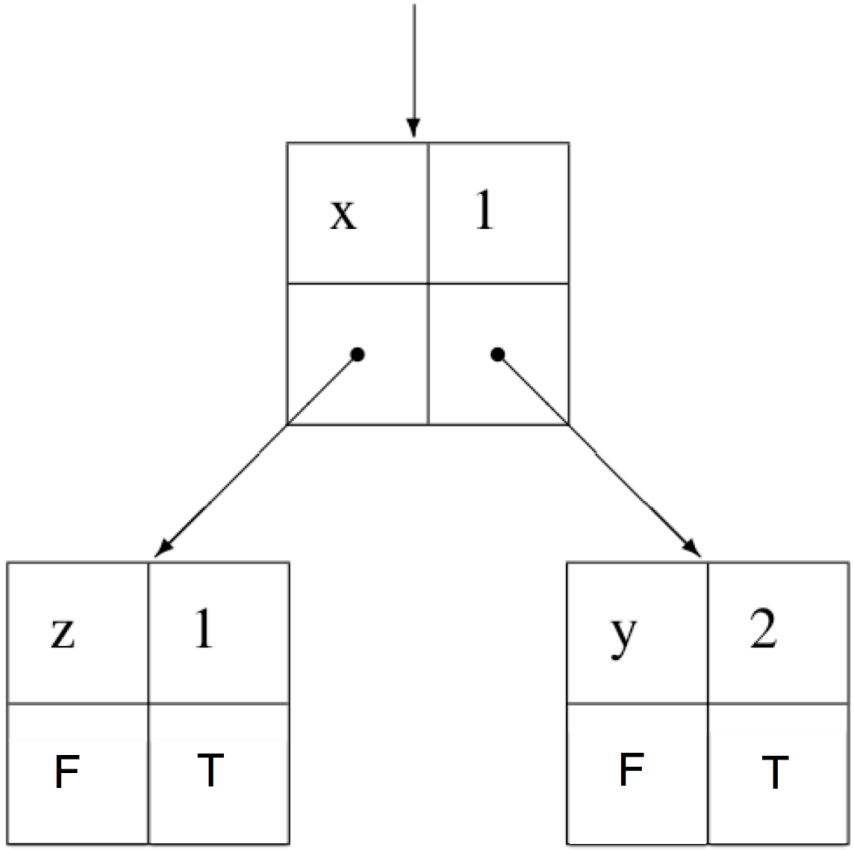


Figure: AEDD modeling $f(x, y, z) = ((x \geq 1) \wedge (y \geq 2) \cup ((x < 1) \wedge (z \geq 1))$

Application of BDD to Petri nets

Up-to-date state of the art in this field

libDDD developed at LIP6, Paris [Mieg et al.]

- C++ library for manipulation of decision diagrams
- Support for Petri nets and some of their extensions
- Available at `http://move.lip6.fr/software/DDD/`

Models classification

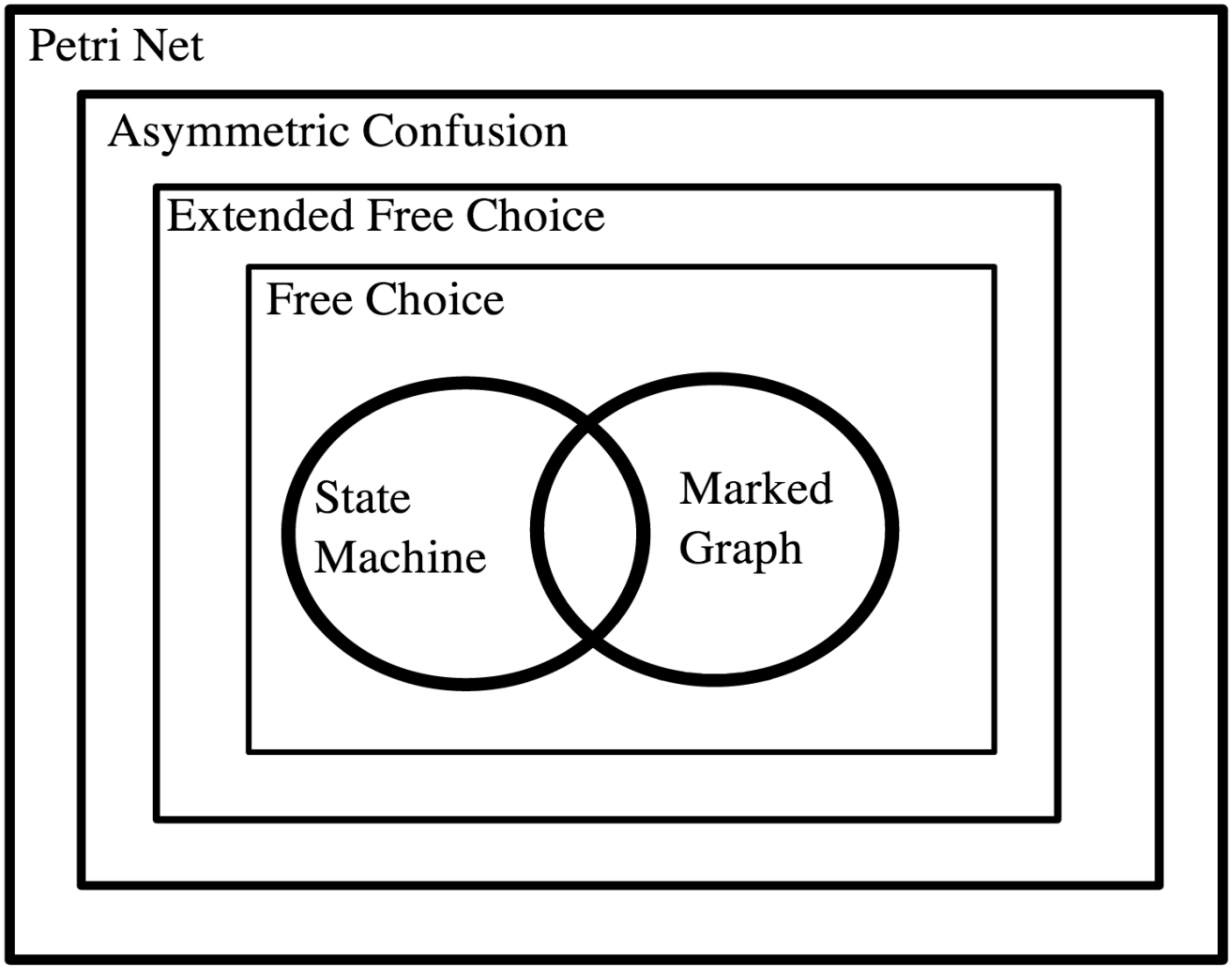


Figure: Various classes of models

Overview

Finite state machine - Introduction

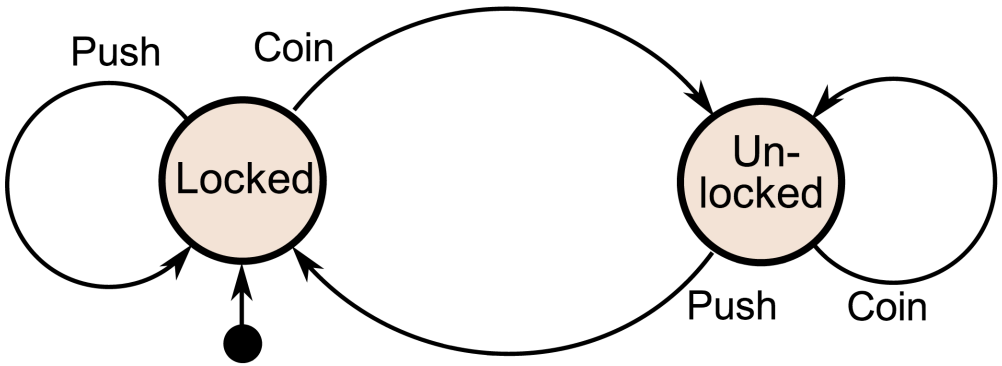


Figure: Example of a FSM modeling a turnstile (taken from `Wikipedia`)

Finite state machine - Introduction

Definition

- A finite number of states
- A set of transitions triggered by events or conditions
- *An initial state*

Finite state machine - Introduction

Basics

- The machine is in only one state at a time
- FSM are just a part of **automata theory**.
- **Powerset construction**: any nondeterministic finite automaton (NFA) can be turned into a deterministic finite automaton (DFA) which recognizes the same formal language.

Automata: a wide family

Variations

- **Input**
  - Finite input: accepts only finite words
  - Infinite input: accepts infinite words
- **States**
  - Finite states automaton
  - Infinite states automaton
  - Pushdown automaton
- **Transition function**
  - Deterministic
  - Nondeterministic

Automata: classification

Classification

Automaton = finite representation of a formal language that may be an infinite set

Automaton	**Recognizable language**
Nondeterministic/Deterministic Finite state machine (FSM)	regular languages
Pushdown automaton (PDA)	context-free languages
Turing machine	recursively enumerable languages
Deterministic Büchi automaton	ω-limit languages
Nondeterministic Büchi automaton	ω-regular languages

Overview

Computation paths vs computation tree

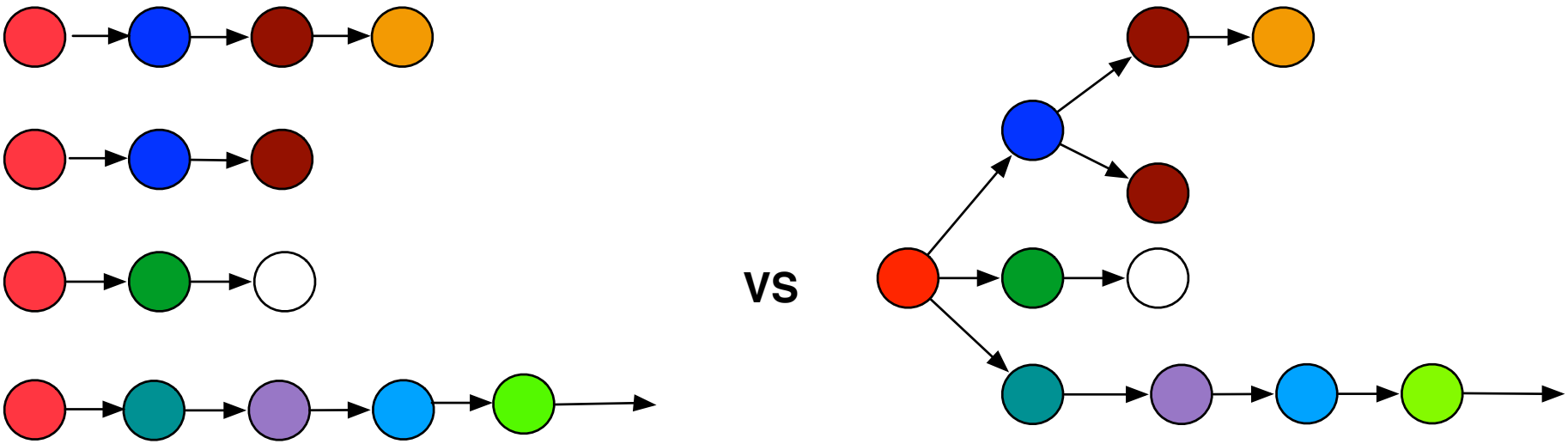


Figure: Execution can be seen as a set of execution paths or as an execution tree

Linear Temporal Logic

Principle

- First proposed for the **formal verification of computer programs** by A. Pnueli in 1977
- Sometimes called propositional temporal logic (PTL)
- LTL is the set of formulae φ_s s.t.:

$$\varphi_1, \varphi_2 := \textbf{true} \mid ap \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid X\varphi_2 \mid \varphi_1 U \varphi_2$$

  with ap an atomic proposition.
- The above definition allows the definition of G and F operators, $\Longrightarrow$ operator, ...:
  - $F\varphi \Leftrightarrow \textbf{true} U \varphi$
  - $G\varphi \Leftrightarrow \neg F \neg \varphi$

Model-checking of LTL properties

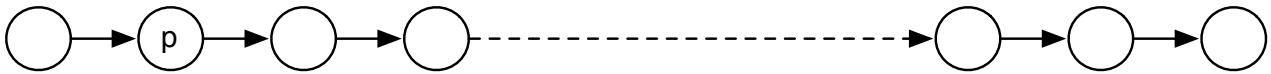


Figure: Xp

Model-checking of LTL properties

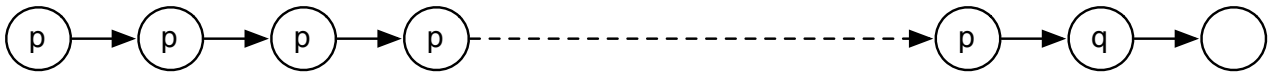


Figure: pUq

Model-checking of LTL properties

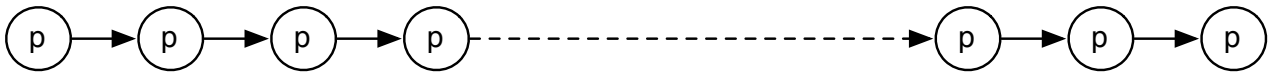


Figure: Gp

Model-checking of LTL properties

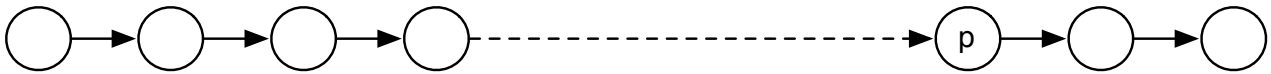
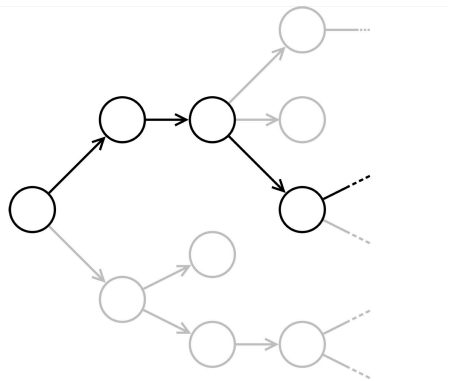


Figure: Fp

Model-checking of LTL properties

Principle

- Express desired properties using LTL operators and actually check if the model satisfies these properties
- Well adapted to:
  - **safety** properties **on a path** (i.e. something bad never happens, stated through $\mathsf{G}(\neg\varphi)$ operator)
  - **liveness** properties **on a path** (i.e. something good keeps happening, stated through $\mathsf{GF}\psi$ or $\mathsf{G}(\varphi \implies \mathsf{F}\psi)$
- But LTL does not allow to express properties on every path



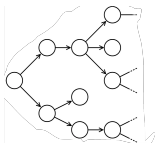
Computation Tree Logic

Principle

- **Branching-time logic**
- Defined by E. M. Clarke and E. A. Emerson in 1981
- CTL is the set of formulae s.t.:

$$\varphi_1, \varphi_2 := \textbf{true} \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid EX\varphi_1 \mid AX\varphi_1 \mid \varphi_1 EU \varphi_2 \mid \varphi_1 AU \varphi_2$$

  with *ap* is an atomic proposition
- Note that E stands for "*along at least one path*"(there Exists one path)
- The above formal definition allows the definition of additional operations, like: AG, AF, ... (A stands for "along All paths")



Model-checking of CTL properties

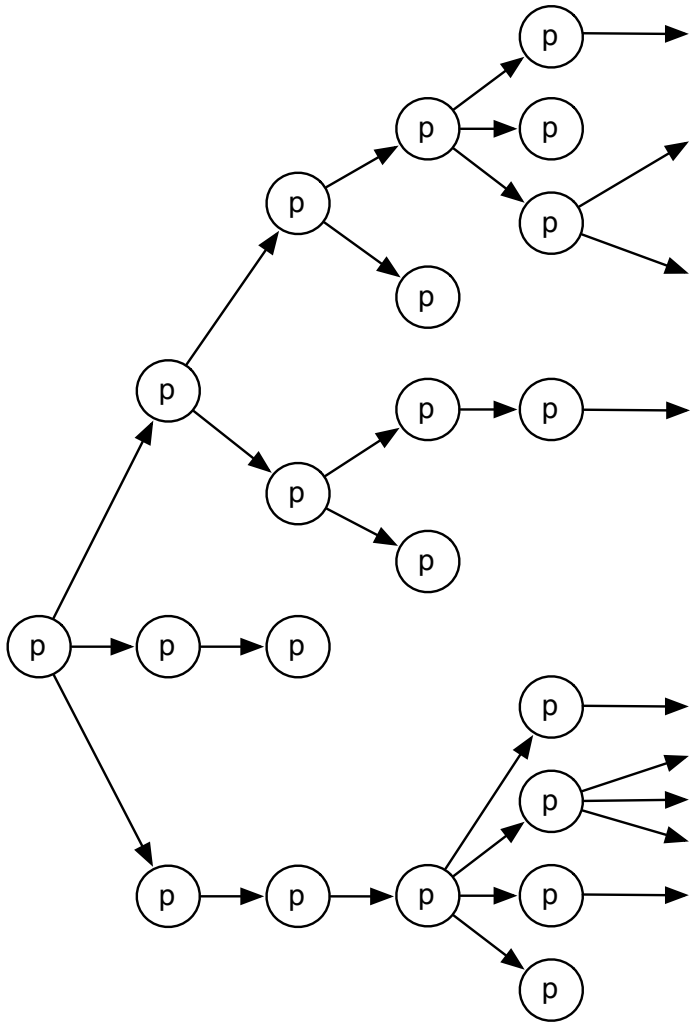


Figure: AGp

Model-checking of CTL properties

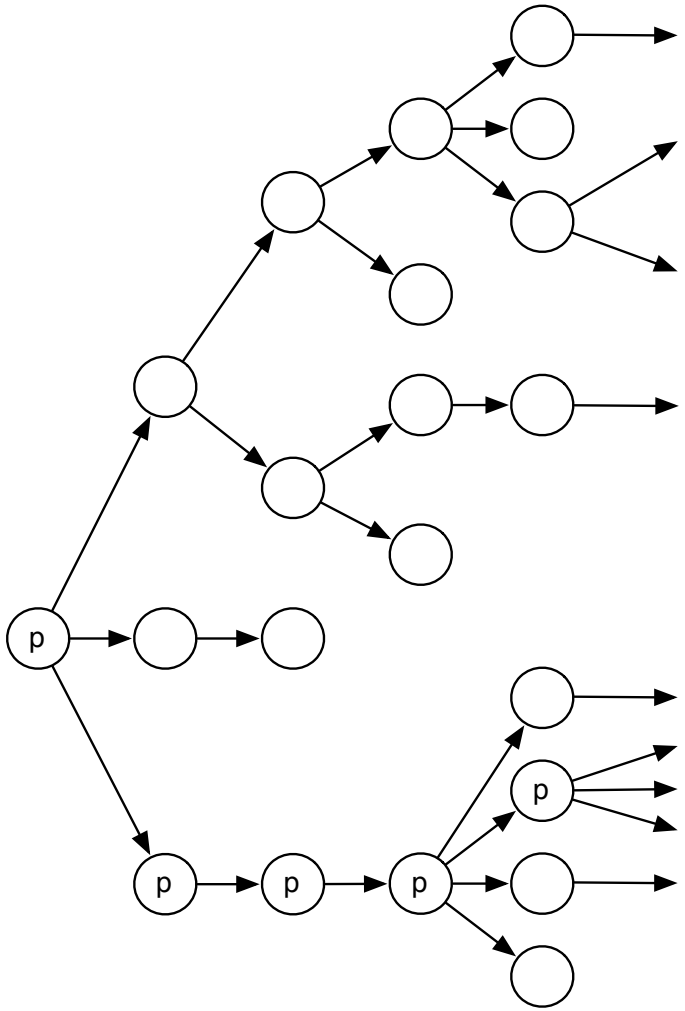


Figure: EGp

Model-checking of CTL properties

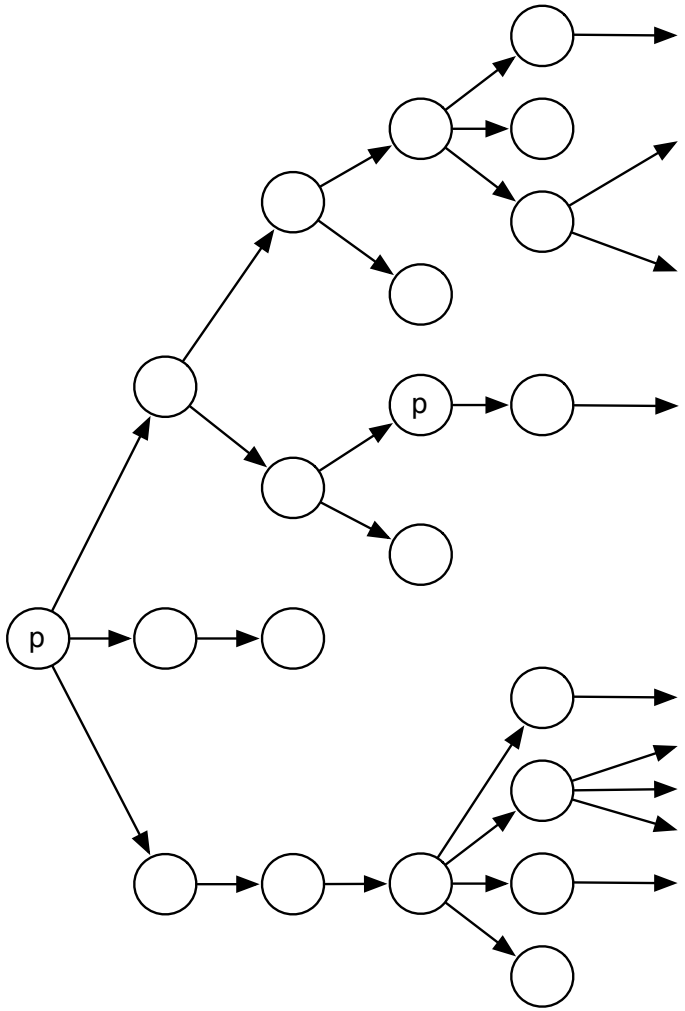


Figure: $EF\,p$

Model-checking of CTL properties

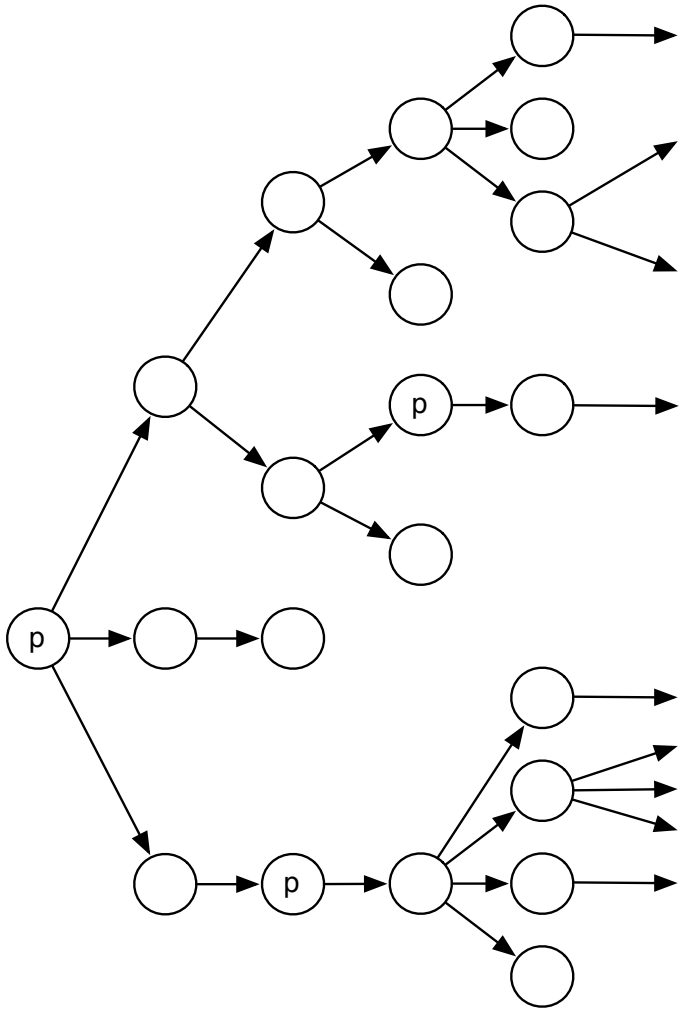


Figure: $EF\,p$

Model-checking of CTL properties

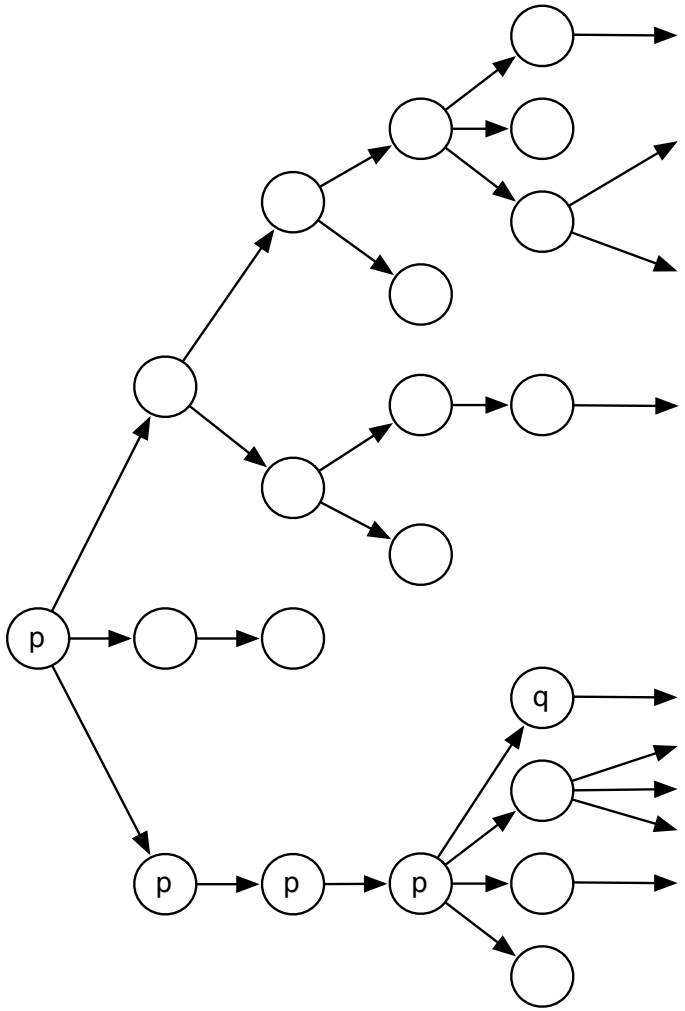


Figure: p*EU*q

Model-checking of CTL properties

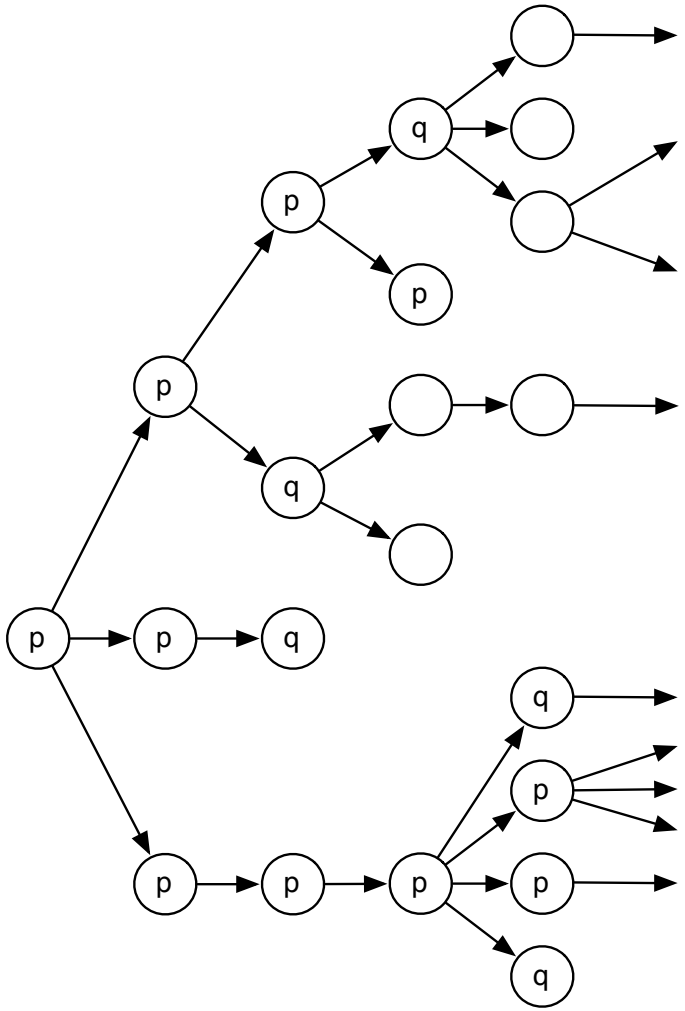


Figure: pAUq

Model-checking of CTL properties

Principle

- Express desired properties using CTL operators and actually check if the model satisfies these properties
- Well adapted to:
  - **safety** properties (e.g. when some initial condition is satisfied, then all possible executions of a program avoid some undesirable condition)
  - **liveness** properties (e.g. when some initial condition is satisfied, then some desirable property keeps happening)

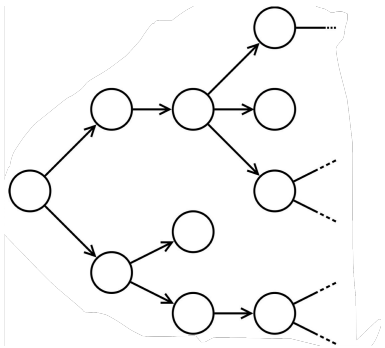


Figure: CTL defines properties on the different paths

LTL vs CTL

Comparison

- Linear time (paths) VS branching time (trees)
- **Incomparable expressive power**:
  - FGp cannot be expressed in CTL
  - $AGEFp$ cannot be expressed in LTL
- LTL may be more intuitive, but seems **too weak**

Computation Tree Logic CTL*

Principle

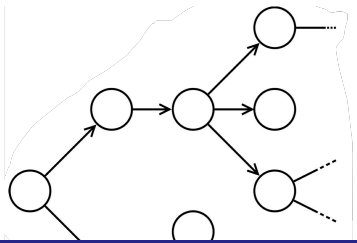
- **Superset** of CTL and LTL
- Combines **path quantifiers** and **temporal operators**
- Defined by E. A. Emerson and J.Y. Halpern in 1986
- CTL* is the set of φ_s formulae s.t.:

$$\varphi_s, \psi_s := \textbf{true} \mid ap \mid \neg\varphi_s \mid \varphi_s \wedge \psi_s \mid A\varphi_p \mid E\varphi_p$$

with

$$\varphi_p, \psi_p := \textbf{true} \mid \varphi_s \mid \varphi_p \wedge \psi_p \mid X\varphi_p \mid \varphi_p U \psi_p$$

- *AGFp* is a CTL* property, not a CTL property.



Computation Tree Logic CTL*

A path quantifier can prefix an assertion composed of arbitrary combinations of the usual linear-time operators.

Path quantifiers

- A : "for every path"
- E : "there exists a path"

Linear-time operators

- Xp : "p holds true next time"
- Fp : "p holds true sometime in the future"
- Gp : "p holds true globally in the future"
- pUq : "p holds true until q holds true"

Model-checking of CTL* properties

Principle

- Express desired properties using CTL* operators and actually check if the model satisfies these properties
- Well adapted to **complex properties**, like "whatever is the state of the system, if an alarm is received, then a signal must be emitted": $AG(alarm \Rightarrow AFsignal)$

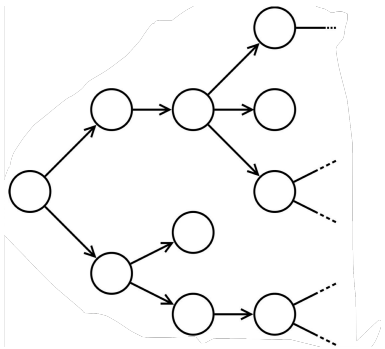


Figure: CTL* defines properties on the different paths

Tips for model-checking CTL* properties

Principles (as summarized by G. Bernot)

- Idea 1: work on the **state graph** instead of the path trees
- Idea 2: check first the **atoms** of φ and then check the connectives of φ with a bottom-up computation strategy
- Idea 3: (computational optimization) group some cases together using **BDD-inspired** techniques

But...

Since the late 90s, progress in BDDs has stalled while SAT solvers performances have increased $\Rightarrow$ some recent papers focus on checking:

- LTL formulae **through SAT solvers** (Biere et al.,1999, 2006)
- CTL formulae **through SAT solvers** (Heymans, 2005)

Model-checking

Summary

- Powerful methodology to:
  - **Verify** a wide range of given properties
  - **Validate** a model
- Application to discrete models, like FSM, Petri nets, ...
- Available automatic tools

Model-checking

Summary

- Powerful methodology to **verify** and/or **validate**
- Application to discrete models, like FSM, Petri nets, ...
- Available automatic tools

How to circumvent state space explosion?

- **Symbolic algorithms** to avoid building the whole state space and regroup states (e.g. BDD-based techniques)
- **Bounded model-checking**: consider only counterexamples of a bounded length $\Rightarrow$ whereas completeness, termination is guaranteed.
- **Abstraction** (e.g. consider only boolean variables)
- **Partial order reduction** (e.g. given a property, if the order between two sequences σ_1 and σ_2 is not meaningful, then it is unnecessary to consider both interleavings $\sigma_1.\sigma_2$ and $\sigma_2.\sigma_1$)

Model-checking

Summary

- Powerful methodology to **verify** and/or **validate**
- Application to discrete models, like FSM, Petri nets, ...
- Available automatic tools

Further work

- Progressive introduction of **time delays**
- Extend the logics expressiveness to tackle properties with **quantitative timing information** $\Rightarrow$ Lecture 2


Randal E. Bryant.
Graph-based algorithms for boolean function manipulation.
IEEE Trans. Computers, 35(8):677–691, 1986.


Jean-Louis Giavitto, Hanna Klaudel, and Franck Pommereau.
Qualitative modelling and analysis of regulations in multi-cellular systems using petri nets and topological collections.
In *4th Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'10)*, number 40 in EPTCS, pages 1–16, 2010.


P. J. Goss and J. Peccoud.
Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets.
Proceedings of the National Academy of Sciences of the United States of America, 95(12):6750–6755, June 1998.


I Koch and M Heiner.
Petri Nets, chapter 7, pages 139–179.
Wiley Book Series on Bioinformatik. John Wiley & Sons, 2008.

Series Eds. Yi Pan, Albert Y. Zomaya.


H Matsuno, A Doi, M Nagasaki, and S Miyano.
Hybrid petri net representation of gene regulatory network.
Pacific Symposium On Biocomputing, 349(338-349):341–352, 2000.


Morgan Magnin, Pierre Molinaro, and Olivier H. Roux.
Expressiveness of Petri nets with stopwatches. Dense-time part.
Fundamenta Informaticae, 97(1-2):111–138, 2009.


L. Jason Steggles, Richard Banks, Oliver Shaw, and Anil Wipat.
Qualitatively modelling and analysing genetic regulatory networks: a petri net approach.
Bioinformatics, 23:2006, 2007.