Journée de la Fédération Charles Hermite

Titre : **Approches à base de systèmes à événements discrets (SED) pour l'ordonnancement de tâches manufacturières ou informatiques**

Organisateurs de la journée : Alexis Aubry (CRAN), Marie Duflot-Kremer (LORIA) et Pascale Marangé (CRAN)

Date : 8 juin 2017

Contexte :
L'ordonnancement consiste à affecter des opérations à des ressources et à définir leurs dates de début et de fin tout en respectant des contraintes propres au système considéré.
Dans un contexte manufacturier, derrière cette définition générale se cache une grande variété de problèmes dépendant notamment du type d'atelier considéré (flow-shop, job-shop, open-shop…) et du contexte opérationnel considéré (prédictif, proactif, réactif...).
Dans un contexte informatique, on peut penser à des applications comme l'ordonnancement des tâches sur une grille de calcul par ou plus généralement l'allocation de ressources à des processus informatiques.

Face à ces problèmes, la théorie de l'ordonnancement, et plus particulièrement la recherche opérationnelle, a proposé plusieurs méthodes de résolution telles que la programmation mathématique ou les métaheuristiques.

Cependant, que ce soit dans le domaine manufacturier ou le domaine informatique, l'environnement est souvent considéré comme stable et certain sans considérer par exemple la dynamique du système ou encore les perturbations inhérentes à tout système. Or, dans le domaine manufacturier, l'état de la machine est intrinsèquement évolutif (en marche, en panne, en mode dégradé, …) et les temps ou les quantités ne sont pas connus précisément (incertitudes sur les durées opératoires, nombre de produits, occurrence d'une panne…). Dans le domaine informatique, les incertitudes sont plutôt liées à la transmission des messages, ou les temps de traitement dûs à la charge de la machine sur laquelle va tourner une tâche.

Dans ce contexte, il est important de pouvoir exploiter les degrés de flexibilité offerts par les systèmes afin de proposer soit des ordonnancements flexibles ou des ré-ordonnancements par réaction. Et la communauté des SED a justement introduit des concepts pour modéliser des systèmes incluant des aspects temps réel ou aléatoire qui offrent donc une flexibilité de modélisation et d'analyse appréciable.

Dernièrement, un axe de recherche s'est mis en place dans le groupe de travail SED du GDR MACS (61ème section CNU) pour apporter de nouvelles propositions de résolution des problèmes d'ordonnancements par les approches SED. Les objectifs de cet axe sont d'une part de comparer les approches SED aux approches classiques et d'autre part d'apporter des éléments de réponse aux problèmes d'ordonnancements réactifs au niveau modélisation et résolution.

Objectifs de la journée :

L'objectif de cette journée est double. Elle permettra à la fois de réunir les communautés informatique et automatique autour de leur intérêt commun pour l'utilisation des outils des SED pour l'ordonnancement de tâches, mais également de voir l'applicabilité de méthodes issues de recherches informatiques à la réalisation d'ordonnancements réactifs.

Cette journée permettra notamment de présenter les travaux des deux communautés autour de ces problèmes afin d'aborder les verrous scientifiques traités, de présenter les approches de modélisation et de résolution privilégiées et également les applications originales. La journée sera divisée en 2 parties : le matin sera consacré aux approches théoriques de modélisation et leurs applications et l'après-midi sur les outils support.

Programme de la journée :

10h00 – 10h30 Accueil des participants/ café

10h30 – 10h45 Ouverture de la journée avec présentation des objectifs
Jean-François Pétin (CRAN – Université de Lorraine) et **Stephan Merz** (LORIA – Université de Lorraine)

10h45 – 11h15 Ordonnancement des ateliers flexibles de production avec les Automates Temporisés
Pascale Marangé et **Alexis Aubry** (CRAN – Université de Lorraine)

11h15 – 11h45 Model-Checking probabiliste pour l'ordonnancement
Marie Duflot-Kremer (LORIA – Université de Lorraine)

12h00 – 12h30 Bilan sur les problèmes d'ordonnancement difficiles à résoudre par les approches classiques
Laurent Houssin (LAAS – Université de Toulouse)

12h30 – 14h00 Repas

14h00 – 14h45 Outil PRISM
Dave Parker (University of Birmingham)

14h45 – 15h30 Outil ROMEO
Didier Lime (IRCCyN – École Centrale de Nantes)

15h30 – 16h00 Discussions

Contacts :

Alexis AUBRY	alexis.aubry@univ-lorraine.fr
Marie DUFLOT-KREMER	marie.duflot-kremer@univ-lorraine.fr
Pascale MARANGE	pascale.marange@univ-lorraine.fr

Inscription auprès des contacts avant le 30 mai 2017

ORDONNANCEMENT D'ATELIERS FLEXIBLES DE PRODUCTION AVEC LES AUTOMATES TEMPORISÉS

A. AUBRY, S. HIMMICHE, P. MARANGÉ, J-F. PÉTIN

AM2I

INS2I
INSIS

UMR 7039

UNIVERSITÉ DE LORRAINE

CONTEXTE ET OBJECTIFS

- L'**ordonnancement** est généralement réalisé de manière **prévisionnelle** en considérant un **environnement certain et stable.**
- **Problème :** pour prendre en compte la **forte variabilité des produits** et des aléas, il est nécessaire de compléter cet ordonnancement par **des ré-ordonnancements réactifs.**
- Dans ce contexte, nos premiers travaux ont proposé de définir un **ordonnancement admissible** pour un **atelier de type Job-shop** en utilisant une **méthode par recherche d'atteignabilité** (Marangé et *al.*, 2011).
- L'objectif de nos travaux est de
  - **définir des modèles génériques**
  - montrer que l'utilisation de ces **patrons** permet facilement de **modéliser différents types d'atelier**
  - **évaluer la robustesse** des ordonnancements trouvés

ÉTAT DE L'ART

Approches classiques

- De nombreux papiers traitent de différents types de problèmes et proposent des approches de résolutions
- Peu traitent du problème de modélisation et sa généricité

Approche de résolution	*Obtention de la solution*		*Modélisation du problème*		
	Optimalité	Efficience	Complexité de modélisation	Généricité	Evolutivité
MILP	+	-	-	-	-
Méta-heuristique	-	+	-	-	-
Heuristique dédiée	-	+	-	-	-

Approches SED

- Travaux dans le domaine SED : (Subbias & Engell 2010, Panek 2006, Behrmann 2005) ont montré l'intérêt d'utiliser des approches SED pour résoudre des problèmes d'ordonnancement
- Avantage de modélisation : graphique, modulaire, représentation par états, représentation de la dynamique
- Problèmes traités : jobshop

PLAN

- Formalisation du problème
- Approche de génération d'un ordonnancement possible
- Outil logiciel d'ordonnancement basé sur les automates temporisés
- Conclusion - Perspectives

PLAN

- **Formalisation du problème**
- Approche de génération d'un ordonnancement possible
- Outil logiciel d'ordonnancement basé sur les automates temporisés
- Conclusion - Perspectives

FORMALISATION DU PROBLÈME (1/2)

- **Type d'atelier : Job-shop / Open-shop / Flow-shop flexible ou non** (une opération est exécutable sur plusieurs machines)

- **Paramètres** :
  - J : l'ensemble des jobs à réaliser
  - O : l'ensemble des opérations pouvant être réalisées dans l'atelier
    - o_{jk} : la $k^{ème}$ opération du job
  - M : l'ensemble des machines
    - m_{jk} : ensemble des machines qualifiées pour l'opération o_{jk}
    - d_{jkm} : durée de l'opération o_{jk} sur la machine m
- **Variables de décision** :
  - X_{jkm} : Affectation des opérations aux machines
  - y_{jk} : date de début de l'opération o_{jk}
  - c_{jk} : date de fin de l'opération o_{jk}

FORMALISATION DU PROBLÈME (2/2)

- **Contraintes** :
  - C1 : Capacité d'une machine : une machine m exécute une seule opération à la fois
  - C2 : Qualification : une opération ne peut être exécutée que sur une machine étant qualifiée
  - C3 : Non-préemption d'une opération : une opération commencée ne peut être interrompue et reprise plus tard
  - C4 : Non splitting d'une opération : une opération n'est allouée qu'à une seule machine
  - C5 : Précédence d'opération :
    - Cas du job-shop / flow-shop : l'opération o_{jk1} doit être terminé avant le début de l'exécution de l'opération o_{jk2}
    - Cas de l'open-shop : pas d'ordre d'exécution entre les opérations mais les opérations ne peuvent pas s'exécuter en même temps

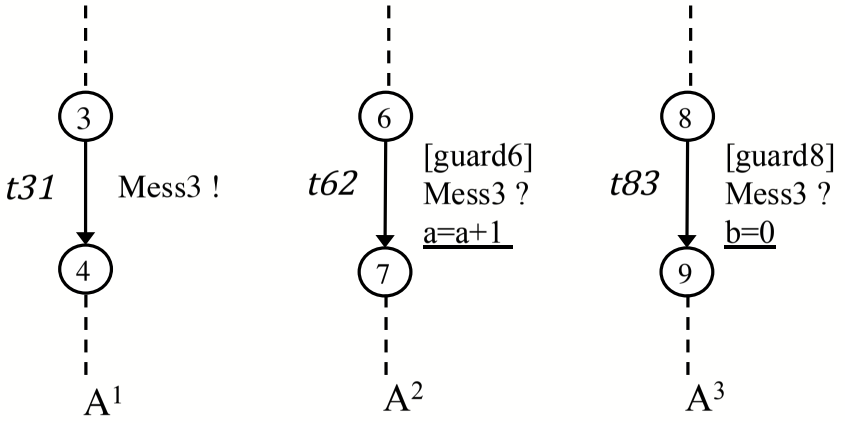
PLAN

- Formalisation du problème
- **Approche de génération d'un ordonnancement possible**
- Outil logiciel d'ordonnancement basé sur les automates temporisés
- Conclusion - Perspectives

MODÉLISATION PAR AUTOMATES TEMPORISÉS

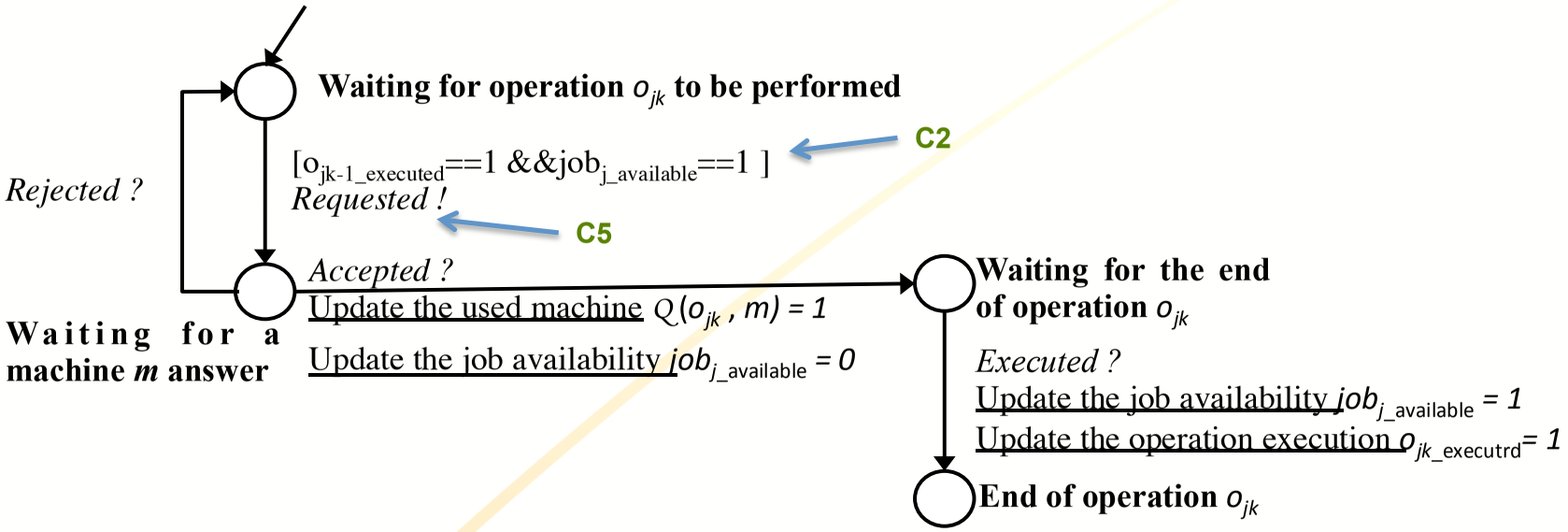
Définition des Automates temporisés

- Définition : A = (S, V, C, L, T, Sm, s_0, v_0)
  - S : ensemble des locaiités
  - V: ensemble des variables
  - C: ensemble des horloges
  - L : ensemble des messages
  - T : ensemble des transitions
  - Sm : ensemble des états marqués
  - s0 : état initiale
  - v0 : valeur initiale des variables



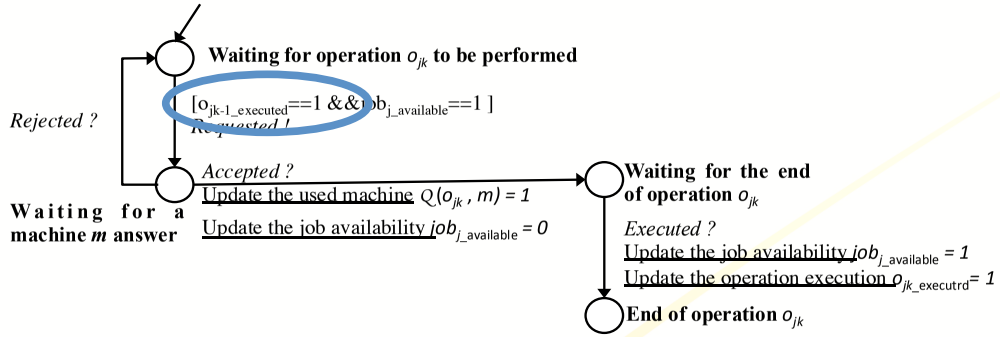
MODÉLISATION PAR AUTOMATES TEMPORISÉS

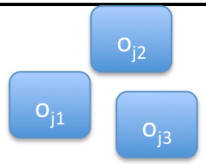
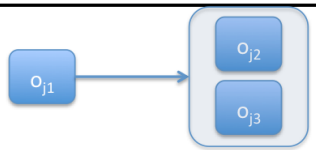
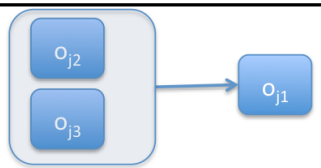
- Évolution du modèle job vers un modèle opération



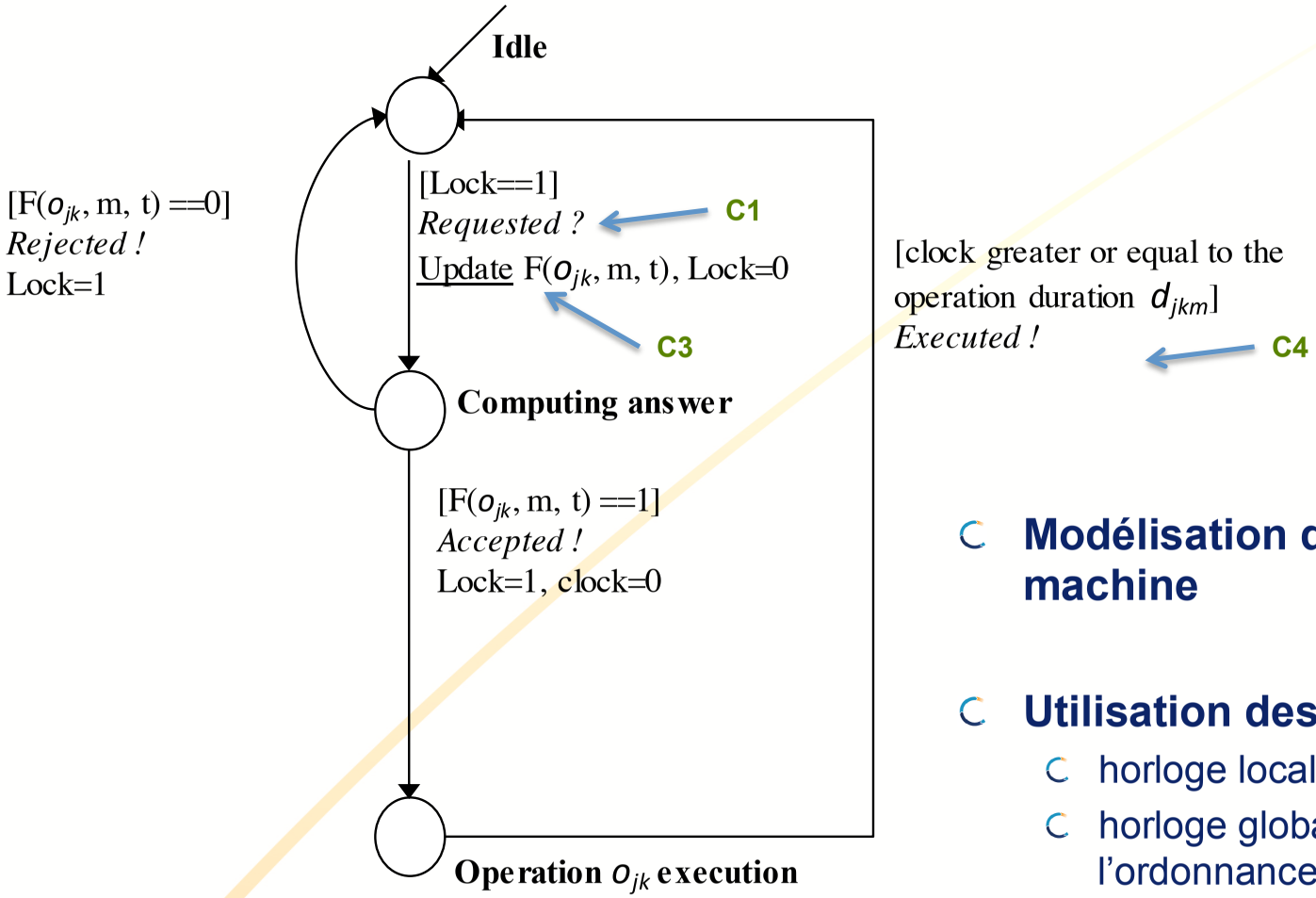
- Ne pas représenter la gamme, mais les opérations
- Règles de précédence sont représentées par la garde
- Mise à jour de l'état de l'opération

MODÉLISATION PAR AUTOMATES TEMPORISÉS



Atelier en Job-shop ou Flow-shop (cas avec precedence entre operations)			
	Precedence constraint o_{j1}	Precedence constraint o_{j2}	Precedence constraint o_{j3}
	∅	[oj1_executed==1]	[oj2_executed==1]
Atelier en Open-shop (cas sans précédence entre opérations)			
	Precedence constraint o_{j1}	Precedence constraint o_{j2}	Precedence constraint o_{j3}
	∅	∅	∅
Atelier hybride 1			
	Precedence constraint o_{j1}	Precedence constraint o_{j2}	Precedence constraint o_{j3}
	∅	[oj1_executed==1]	[oj1_executed==1]
Atelier hybride 2			
	Precedence constraint o_{j1}	Precedence constraint o_{j2}	Precedence constraint o_{j3}
	[oj2_executed==1 ∧ oj3_executed==1]	∅	∅

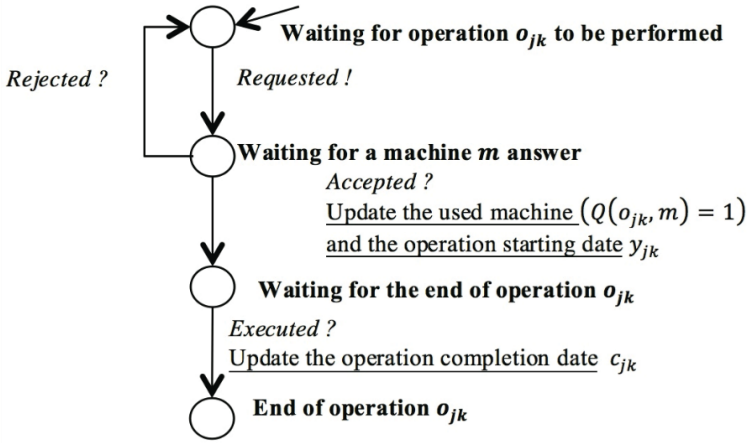
MODÉLISATION PAR AUTOMATES TEMPORISÉS

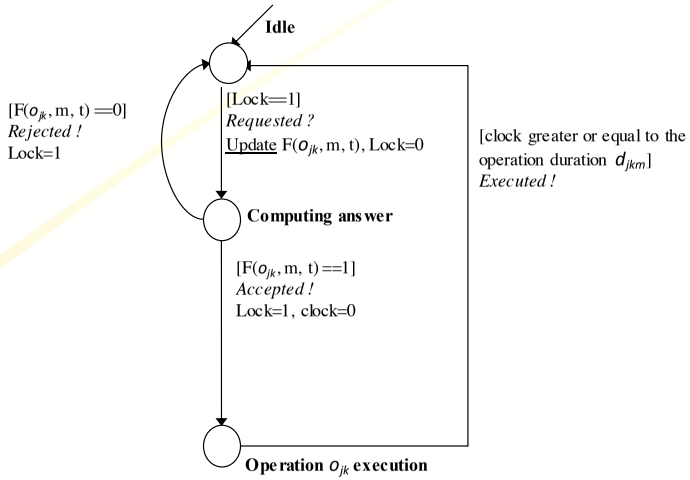


- **Modélisation des états de la machine**
- **Utilisation des horloges**
  - horloge locale à chaque machine
  - horloge globale pour l'ordonnancement

MODÉLISATION PAR AUTOMATES TEMPORISÉS

Approches d'obtention d'un ordonnancement





- Le modèle de job représente la gamme logique à exécuter
- Le modèle de machine représente les capacités, la disponibilité de celle-ci, ainsi que ses états
- Synchronisation entre les modèles de la machine et d'opération est soutenu par un mécanisme d'appel / réponse (Alur and Dill , 1994 ; Lemattre et *al.*, 2011) .

OBTENTION D'UN ORDONNANCEMENT

- **Approche**
  - Obtention d'un ordonnancement faisable par recherche d'atteignabilité
    - Propriété à vérifier : **EF (End_of_Production)**
    - End_of_Production est un état où tous les jobs sont exécutés complétement (Finished product)
  - Possibilité de trouver une solution plus proche de la solution optimale
- **L'affectation est définie dans le dernier état exploré par le model-checker**
- **Les dates de début et de fin se retrouvent dans la trace**

Algorithm 1 Towards optimal scheduling

Require: set of automata $NA = \alpha^1, \alpha^2, ..., \alpha^J, \beta^1, \beta^2, ..., \beta^M$ for J jobs and M machines

Ensure: allocation function $\boldsymbol{F}$, set of starting dates $\mathcal{Y}_j$, optimal completion time $OptimalTime$,

1: /* **Initialisation** */
2: $P \leftarrow EF(End\_Of\_Production)$
3: /* **Iterative reachability analysis:** */
4: **while** $NA \models P$ (P holds on NA) **do**
4: Find some trace such as $NA \models P$ and compute total completion time tct
4: $OptimalTime \leftarrow tct$
4: $P \leftarrow EF(End\_Of\_Production \wedge (tct < OptimalTime))$
5: **end while**

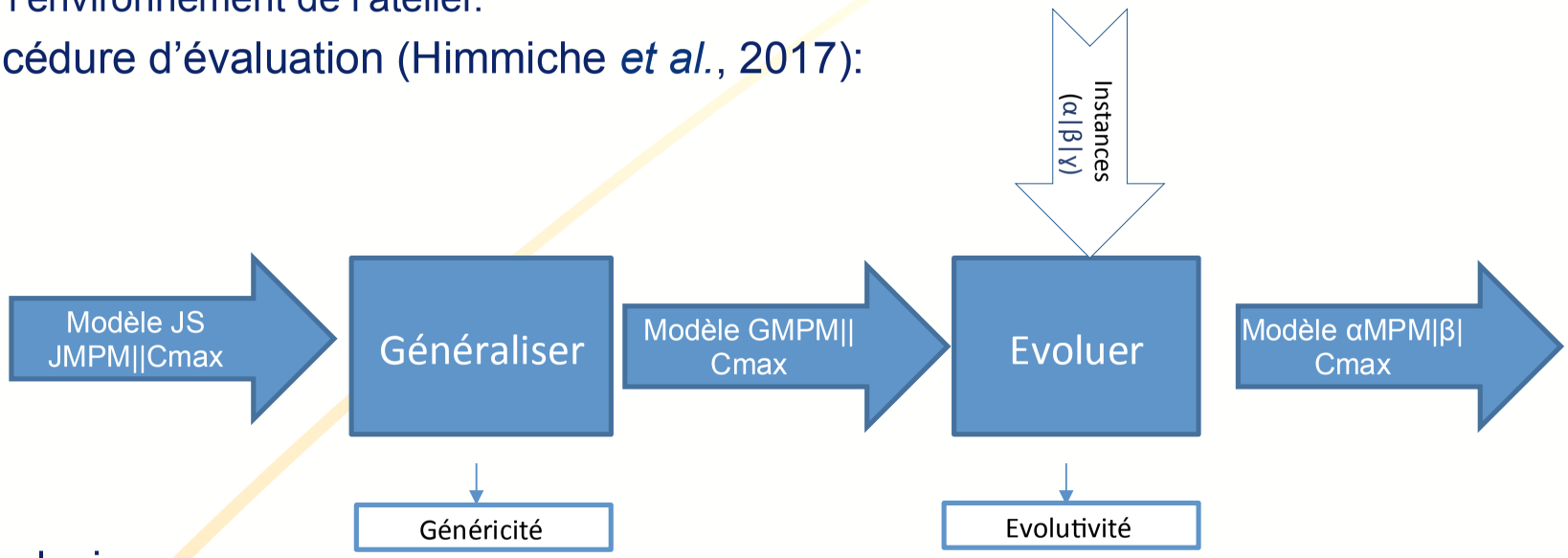
EVALUATION DE LA QUALITÉ DE LA SOLUTION TROUVÉE

- **Temps d'exécution** : cohérent par rapport aux travaux précédents => temps d'exécution compatible avec ordonnancement en ligne

- **Optimalité** : solution admissible uniquement mais lors d'un ré-ordonnancement, l'admissibilité peut être suffisante si la durée de l'ordonnancement reste raisonnable

EVALUATION DE LA MODÉLISATION

- **Pouvoir de modélisation (comparaison avec MILP) :**
  - Complexité : Effort à fournir pour modifier le modèle.
  - Généricité : Capacité d'un modèle à être applicable à tous les types d'atelier
  - Évolutivité : Capacité d'évolution des modèles par rapport aux éléments de l'environnement de l'atelier.
- Procédure d'évaluation (Himmiche *et al.*, 2017):



- Conclusions :
  - Avoir un modèle générique en MILP coûte plus cher
  - Modèles Automates Temporisés plus faciles à faire évoluer pour intégrer des nouvelles contraintes

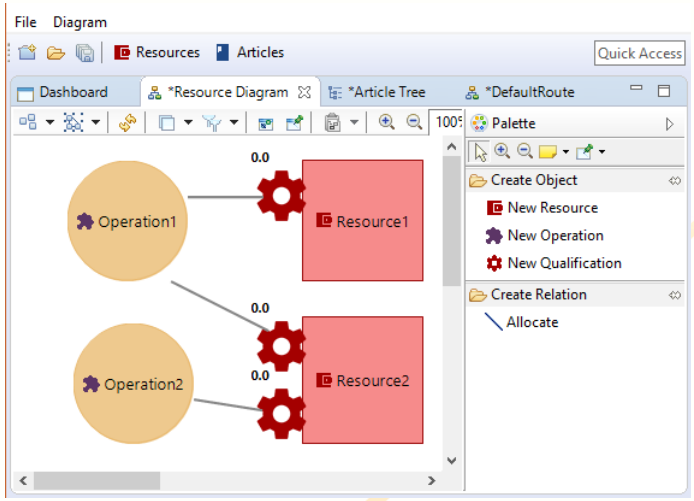
PLAN

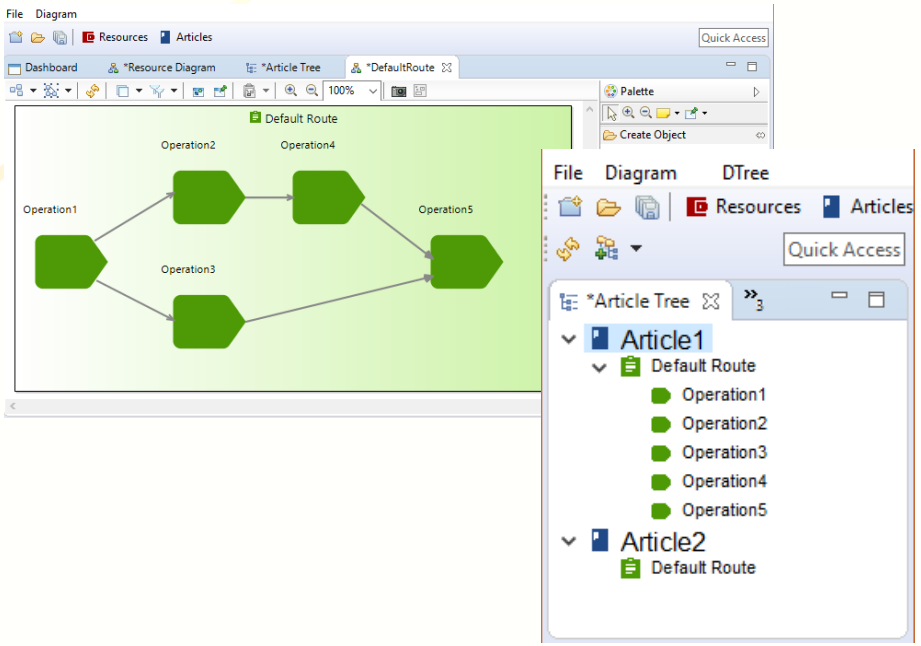
- Formalisation du problème
- Approche de génération d'un ordonnancement possible
- **Outil logiciel d'ordonnancement basé sur les automates temporisés**
- Conclusion - Perspectives

INSTANCIATION DES MODÈLES

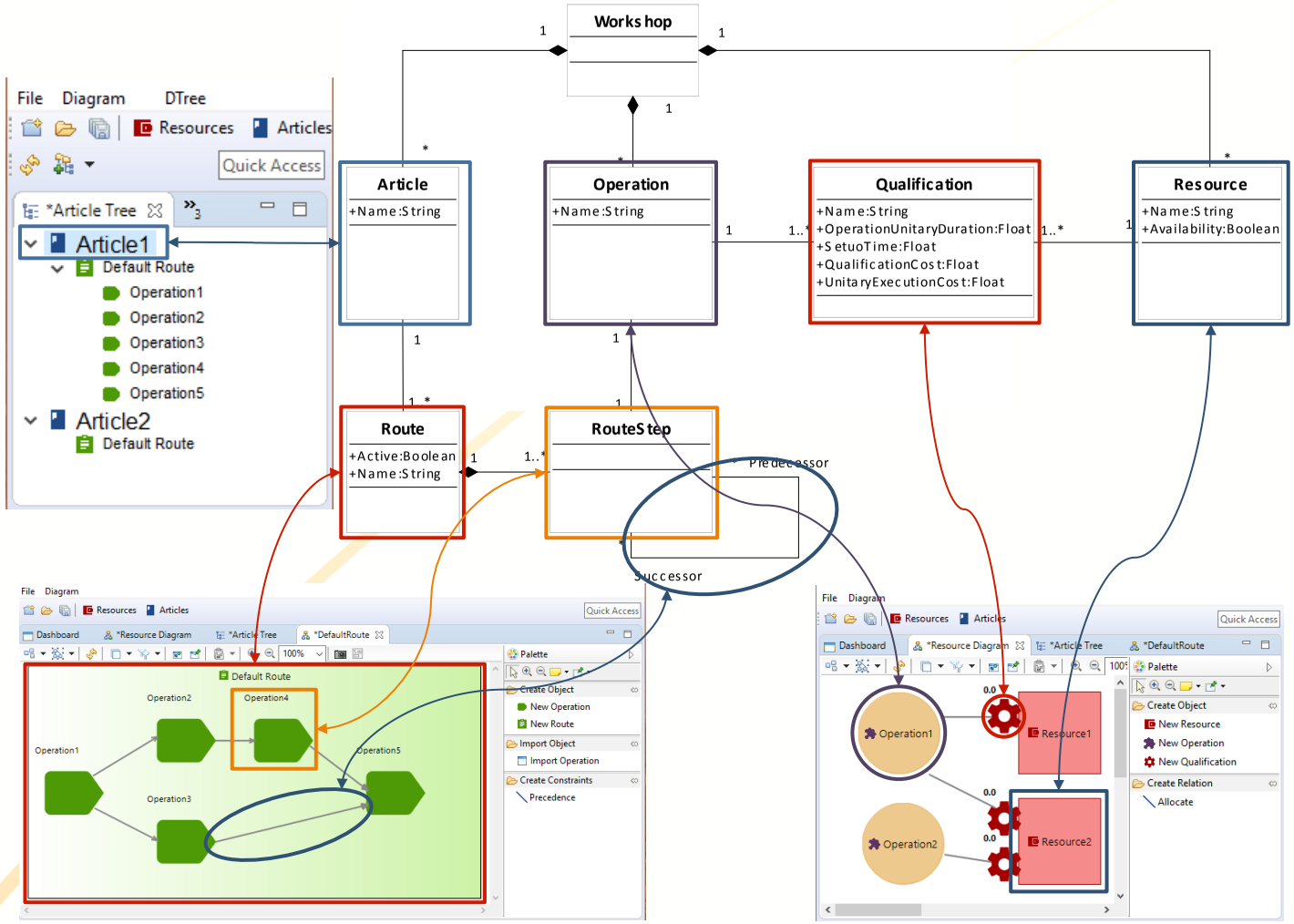
- **Pour que l'approche soit pleinement opérationnelle, un décideur doit pouvoir l'utiliser sans prérequis particulier sur les outils SED.**
- **Un prototype a été développé. Il est supporté par :**
  - i) une IHM qui permet à l'utilisateur de rentrer les informations de l'atelier, de générer automatiquement les automates liés
  - ii) une base de données qui permet de sauvegarder les informations de l'atelier rentrées par l'utilisateur,
  - iii) un générateur de modèles
  - iv) un model-checker (UppAal) pour calculer un ordonnancement admissible et de visualiser cet ordonnancement dans un diagramme de Gantt
- **Logiciel développé par Rémi Pannequin dans le cadre d'un projet CRAN**
  - logiciel opérationnel en cours de tests
  - recherche d'un cas test avec une entreprise

INTERFACE DU LOGICIEL TABS (TIMED-AUTOMATA BASED SCHEDULER)

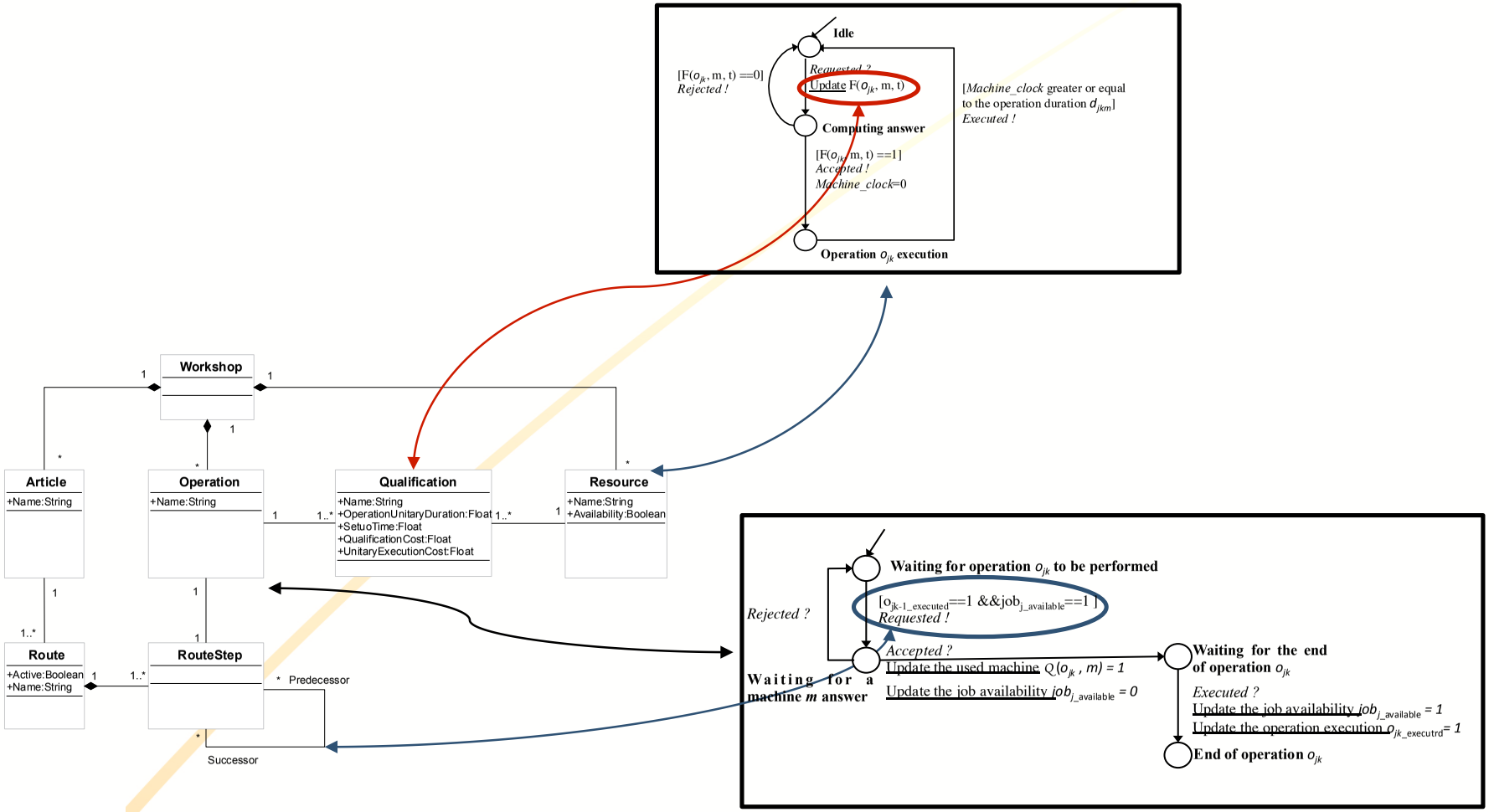




PASSAGE IHM – BASE DE DONNÉES



PASSAGE BASE DE DONNÉES – MODÈLES AUTOMATES



GÉNÉRATION DU GANTT

PLAN

- Formalisation du problème
- Approche de génération d'un ordonnancement possible
- Outil logiciel d'ordonnancement basé sur les automates temporisés
- **Conclusion - Perspectives**

CONCLUSIONS

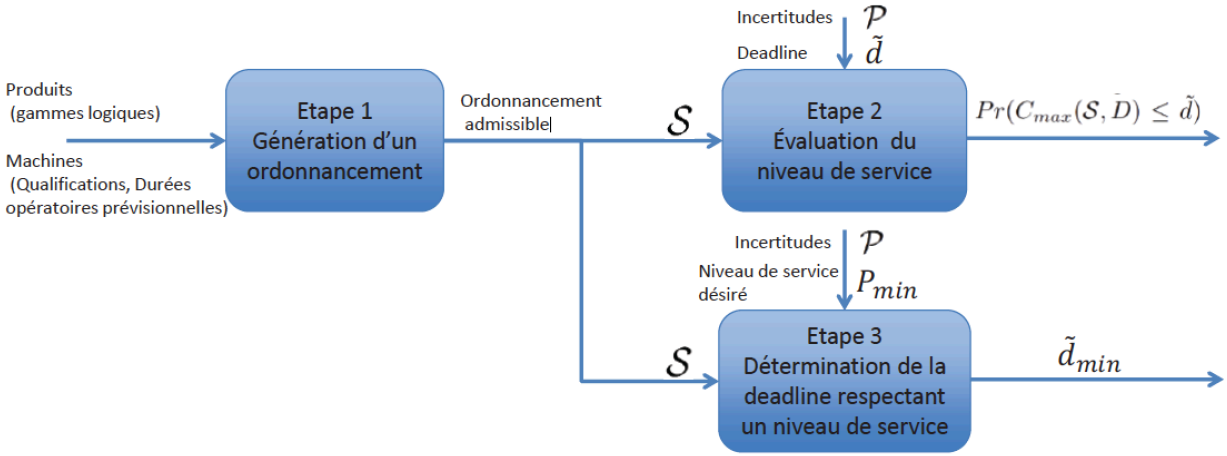
- **Modèles SED => solution prometteuse pour l'ordonnancement d'ateliers flexibles :**
  - Intrinsèquement capables de modéliser la dynamique des systèmes
  - Avec des méthodes associées capables de trouver des ordonnancements
  - Pouvoir de modélisation supérieur aux approches classiques
- **Prototype logiciel en cours de finalisation (sans incertitude)**
  - Preuve de concept

PERSPECTIVES

- **Finaliser l'évaluation de l'approche par rapport aux méthodes classiques**
- **Tester le logiciel sur un exemple industriel**
- **Prise en compte des perturbations :**

PERSPECTIVES

- **Prise en compte des perturbations :**
  - Aléas : L'occurrence d'un évènement non prévu ,
  - Incertitudes : difficulté de connaître de façon certaines les grandeurs factuelles d'un problème.
    - Incertitudes sur les données : durées d'exécutions, deadlines,…
    - Incertitudes sur les états : Etats de panne,…
- **Proposition d'une méthodologie d'évaluation de la robustesse (Master Sara Himmiche)**



- **Modélisation via des automates temporisés probabilistes et évaluation par model-checking**

Model-Checking statistique pour l'ordonnancement

d'après des travaux en collaboration avec
P. Ballarini, H. Djafri, S. Haddad et N. Pekergin

8 juin 2017

remaining slides in unsteadily English

The approach

Modelling FMS

Probabilistic verification/evaluation

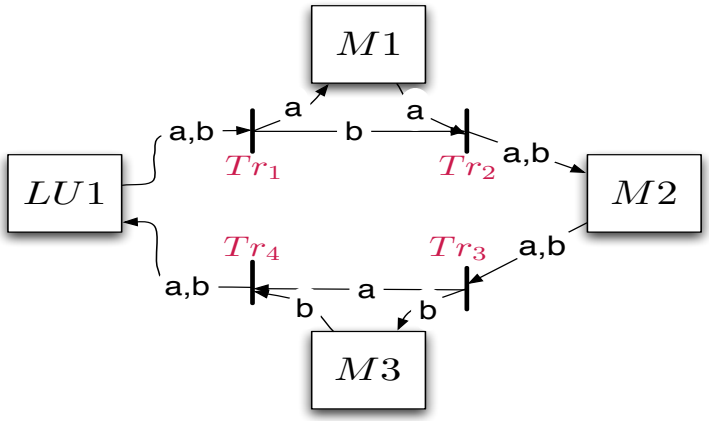
Conclusion

Flexible Manufacturing systems

- a set of basic components
  - machines
  - transportation unit
  - ....

Flexible Manufacturing systems

- a set of basic components
  - machines
  - transportation unit
  - ....
- composition of these different elements
- comparing different architectures/solutions



Principle of the approach

Build the FMS model :

- In an intuitive way
- with appropriate granularity

Principle of the approach

Build the FMS model :

- In an intuitive way
- with appropriate granularity

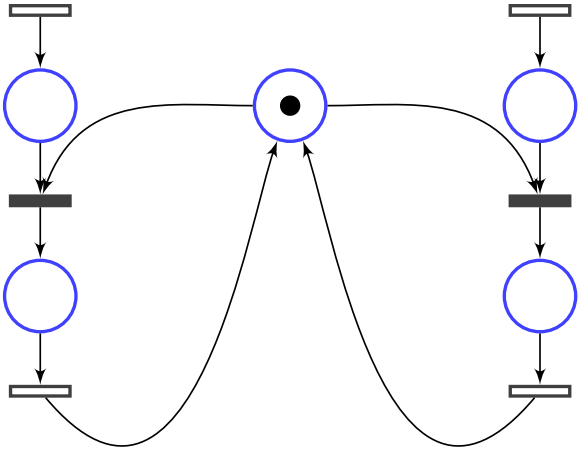
Four steps :

- choose a type of element (transport, machine, loading unit)
- fix parameters
- combine components by binding variable names

Petri Nets

Model to describe our systems

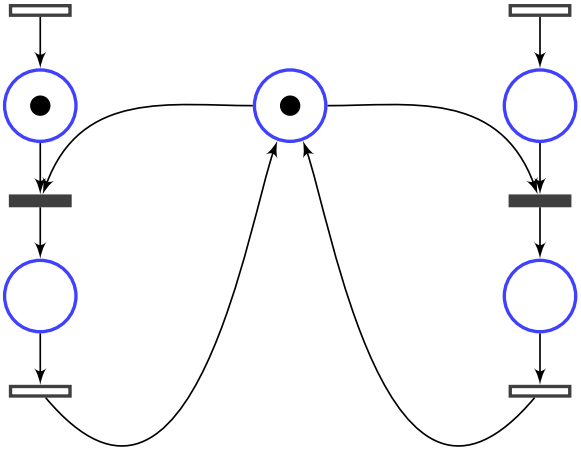
1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems



Petri Nets

Model to describe our systems

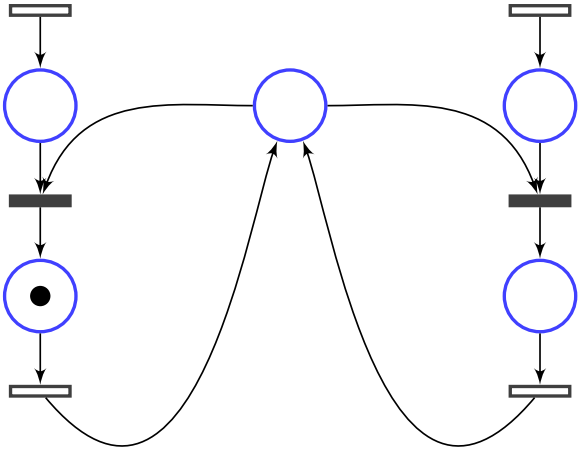
1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems



Petri Nets

Model to describe our systems

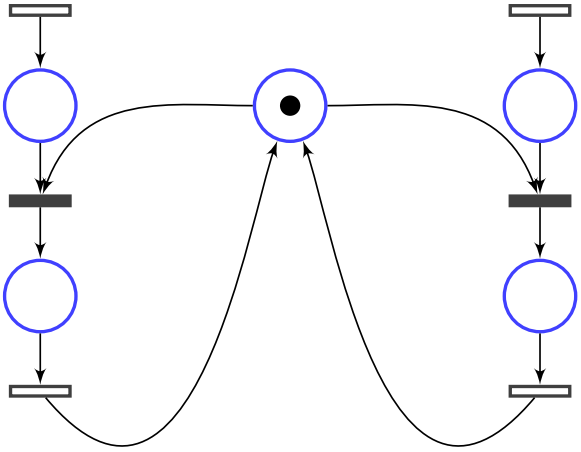
1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems



Petri Nets

Model to describe our systems

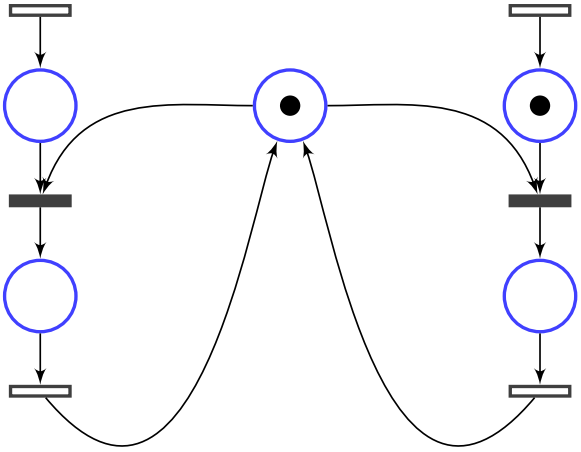
1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems



Petri Nets

Model to describe our systems

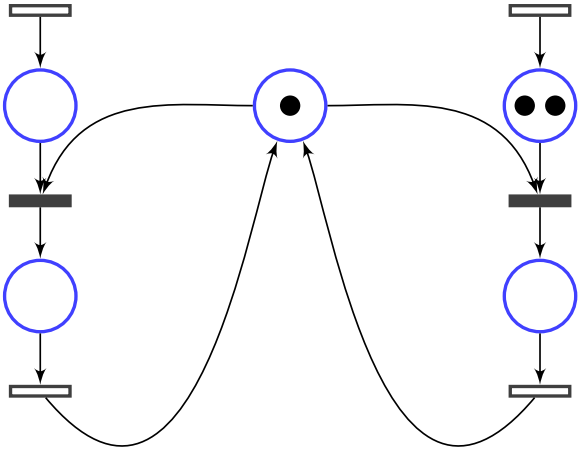
1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems



Petri Nets

Model to describe our systems

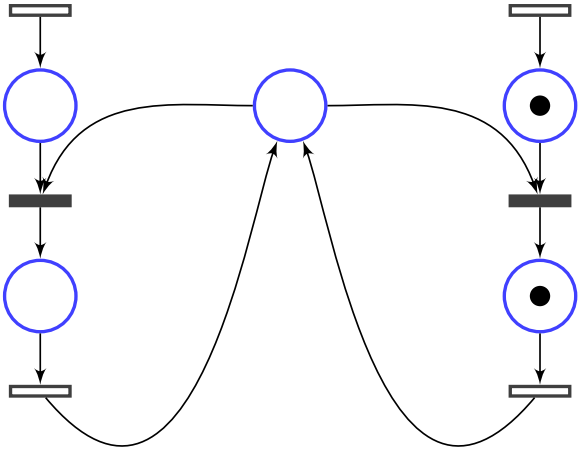
1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems



Petri Nets

Model to describe our systems

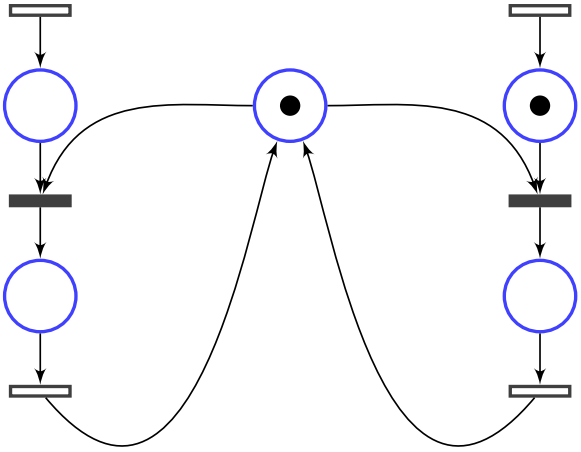
1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems



Petri Nets

Model to describe our systems

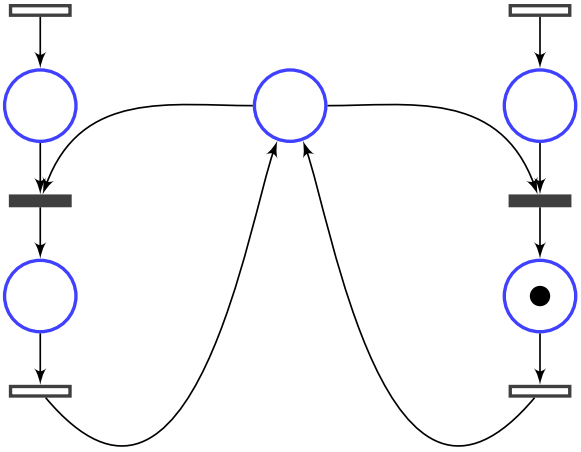
1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems



Petri Nets

Model to describe our systems

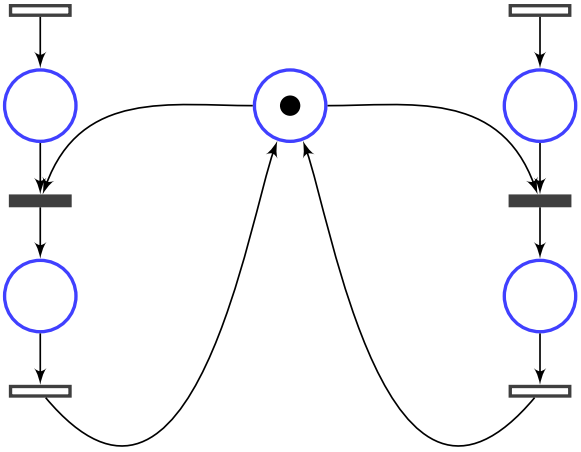
1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems



Petri Nets

Model to describe our systems

1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems



Example : machine unit

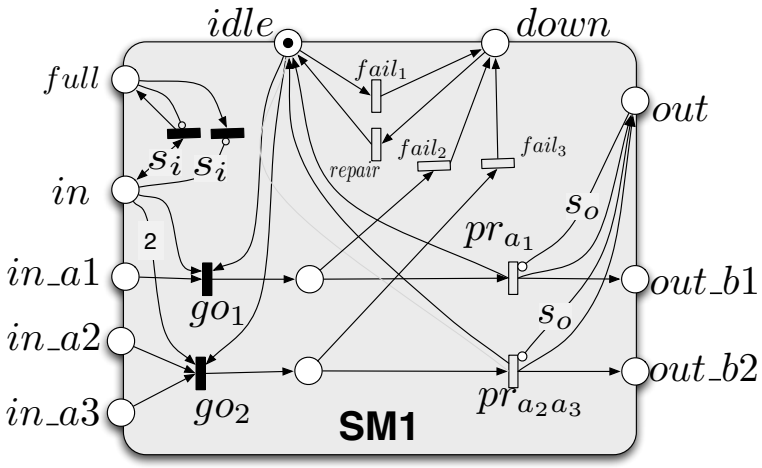
Need to specify :

- type of material needed
- buffers dimension
- processing times
- failure/repair times

Example : machine unit

Need to specify :

- type of material needed
- buffers dimension
- processing times
- failure/repair times

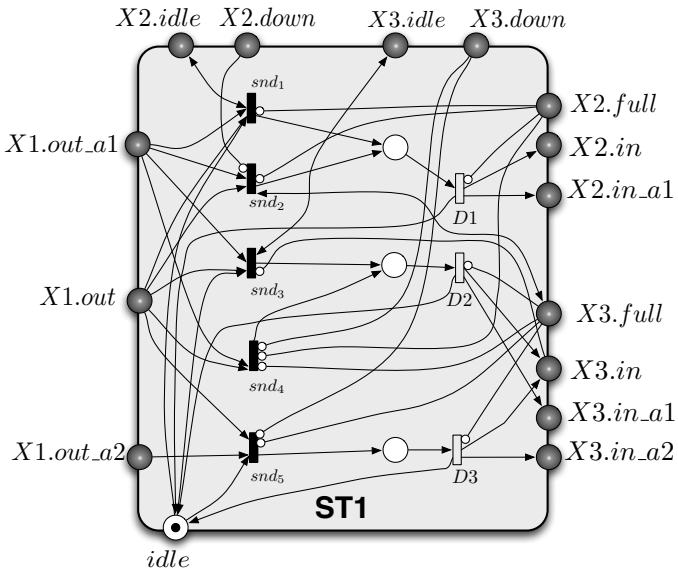


Transport unit

Need to specify :

- what can go where
- transportation policy
- delivery time

+ appropriate parameters depending on the nature of the unit

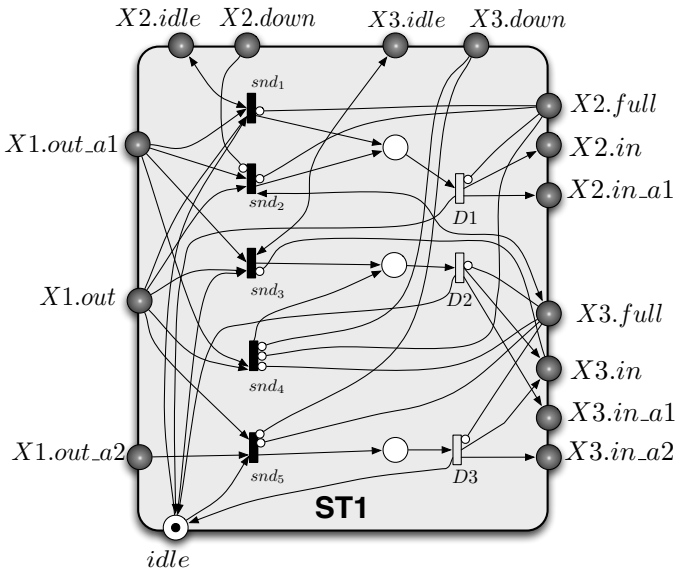


Transport unit

Need to specify :

- what can go where
- transportation policy
- delivery time

+ appropriate parameters depending on the nature of the unit



*a*2 *elements go to* *X*3.
if *X*3 *idle and not full, deliver* *a*1 *to* *X*3,
else if *X*2 *idle and not full, deliver* *a*1 *to* *X*2,
else if *X*3 *up and not full, deliver* *a*1 *to* *X*3,
else if *X*2 *up and not full, deliver* *a*1 *to* *X*2.

Probabilistic aspects

Probabilities required to model :

- uncertainty of the time to process a product,
- possible failure of machines,
- time until repairing the machine,

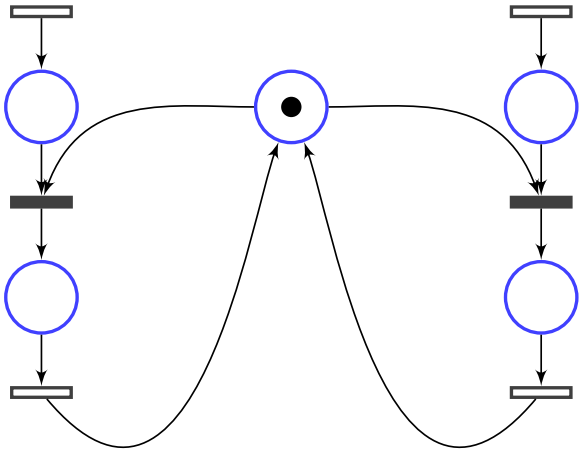
$\rightsquigarrow$ we need to enrich the model with probabilities

Petri Nets

Model to describe our systems

1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems

We need :



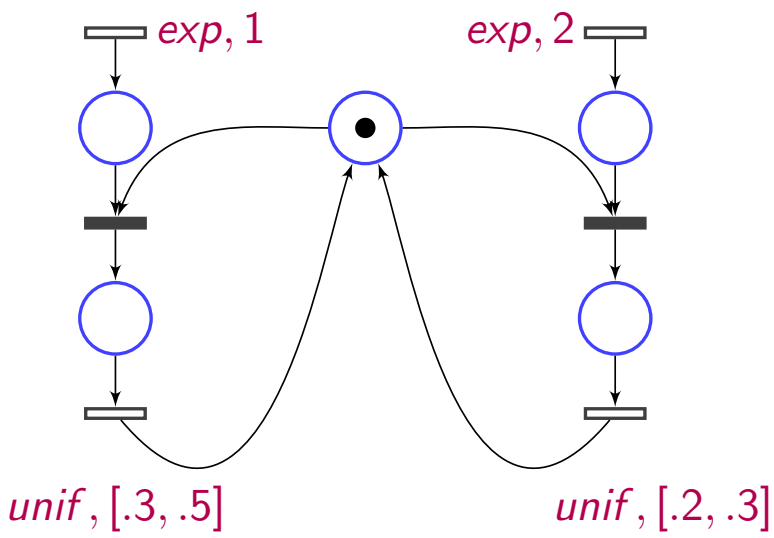
Stochastic Petri Nets

Model to describe our systems

1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems

We need :

- probabilistic distributions (exponential and others)



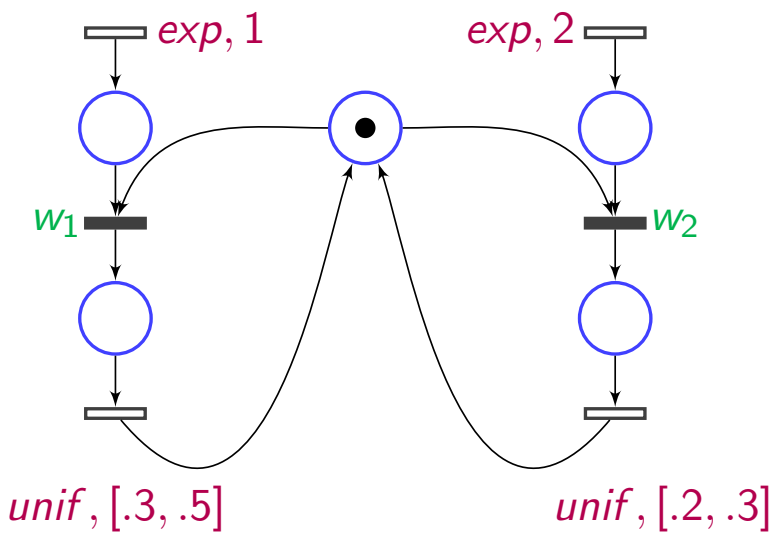
Stochastic Petri Nets

Model to describe our systems

1. places with tokens
2. local transitions consuming/producing tokens
3. good to model distributed systems

We need :

- probabilistic distributions (exponential and others)
- priorities / weights



Dedicated properties/measures

Classical properties to evaluate a particular scheduling :

- makespan (if finite number of objects)
- probability to meet a deadline

Dedicated properties/measures

Classical properties to evaluate a particular scheduling :

- makespan (if finite number of objects)
- probability to meet a deadline

... but also :

- mean time to first failure
- difference of occupancy time between two machines
- mean occupancy of a buffer during rush times/failure
- expected waiting time for k products

Dedicated properties/measures

Classical properties to evaluate a particular scheduling :

- makespan (if finite number of objects)
- probability to meet a deadline

... but also :

- mean time to first failure
- difference of occupancy time between two machines
- mean occupancy of a buffer during rush times/failure
- expected waiting time for k products

Model checking probabilistic systems

Two main approaches :

Numerical	Statistical
graph algorithms + matrix calculus	sampling paths and resorting to statistical results

Model checking probabilistic systems

Two main approaches :

Numerical	Statistical
graph algorithms + matrix calculus	sampling paths and resorting to statistical results
better accuracy	uncertainty due to the statistical approach

Model checking probabilistic systems

Two main approaches :

Numerical	Statistical
graph algorithms + matrix calculus	sampling paths and resorting to statistical results
better accuracy	uncertainty due to the statistical approach
decidable classes of systems	freedom ! !

Model checking probabilistic systems

Two main approaches :

Numerical	Statistical
graph algorithms + matrix calculus	sampling paths and resorting to statistical results
better accuracy	uncertainty due to the statistical approach
decidable classes of systems	freedom ! !
unbounded paths OK	finite (relatively short) paths

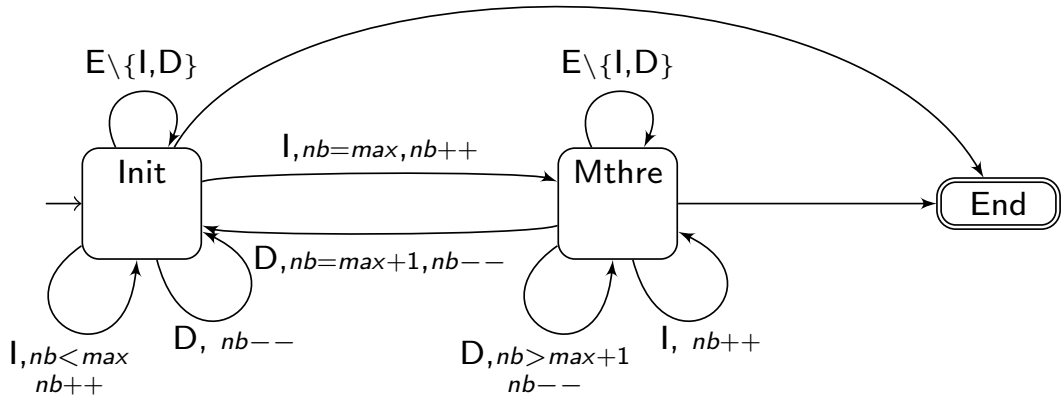
Model checking probabilistic systems

Two main approaches :

Numerical	Statistical
graph algorithms + matrix calculus	sampling paths and resorting to statistical results
better accuracy	uncertainty due to the statistical approach
decidable classes of systems	freedom ! !
unbounded paths OK	finite (relatively short) paths
combinatorial explosion	no problem with big systems

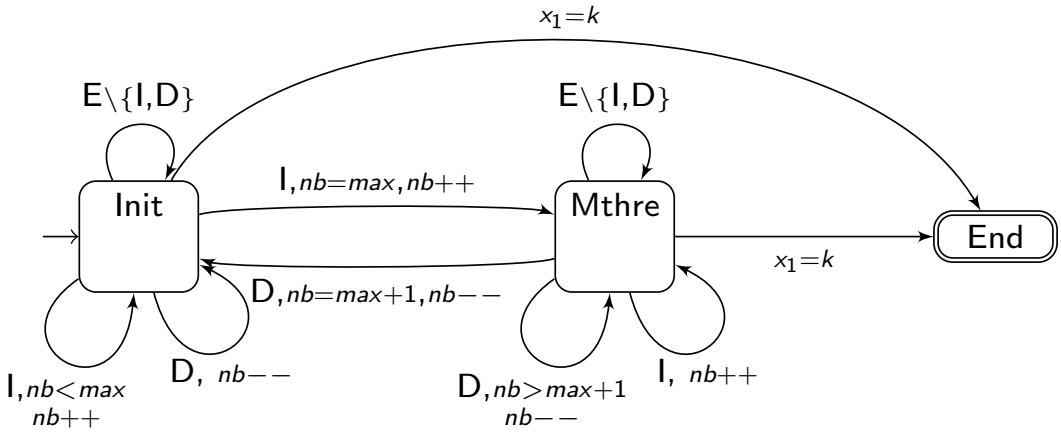
Expressing complex properties

Automaton



Expressing complex properties

Timed Automaton

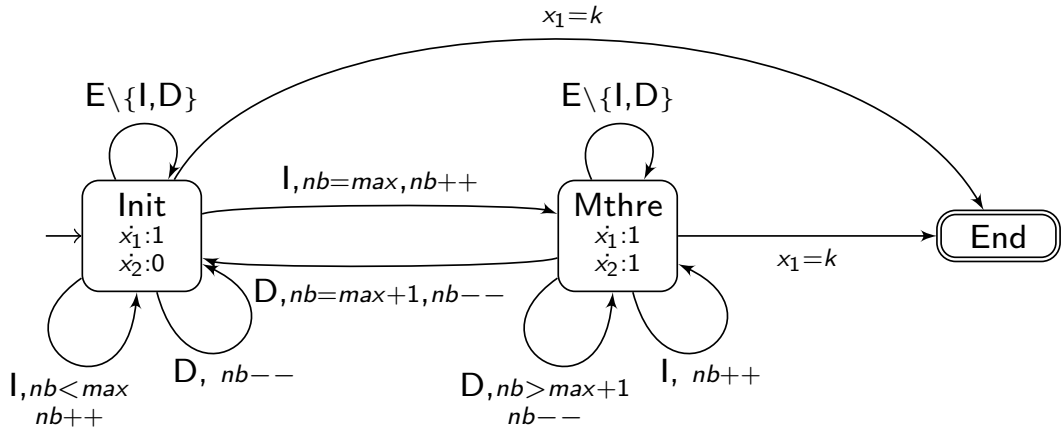


The linear hybrid automaton :

- precisely select chosen executions
- maintain relevant information along the path

Expressing complex properties

Linear Hybrid Automaton



The linear hybrid automaton :

- precisely select chosen executions
- maintain relevant information along the path

COSMOS, a prototype tool for statistical verification

- system : a Petri net
- property/indicator : a linear hybrid automaton

COSMOS, a prototype tool for statistical verification

- system : a Petri net
- property/indicator : a linear hybrid automaton
- synchronise the two (and reject non synchronising paths),
- statistically verify probabilistic properties/Evaluate chosen criteria

Comparison with UppAal SMC

UppAal SMC	COSMOS
user friendly	textual description of properties

Comparison with UppAal SMC

UppAal SMC	COSMOS
user friendly	textual description of properties
well established tool	prototype

Comparison with UppAal SMC

UppAal SMC	COSMOS
user friendly	textual description of properties
well established tool	prototype
exponential & uniform distributions	various others

Comparison with UppAal SMC

UppAal SMC	COSMOS
user friendly	textual description of properties
well established tool	prototype
exponential & uniform distributions	various others
clock rates 0 or 1	constants or system dependant

Comparison with UppAal SMC

UppAal SMC	COSMOS
user friendly	textual description of properties
well established tool	prototype
exponential & uniform distributions	various others
clock rates 0 or 1	constants or system dependant
set clock value to a constant	linear expression over variables

Comparison with UppAal SMC

UppAal SMC	COSMOS
user friendly	textual description of properties
well established tool	prototype
exponential & uniform distributions	various others
clock rates 0 or 1	constants or system dependant
set clock value to a constant	linear expression over variables

comparison with PRISM ?
Let's see after Dave's talk :o)

What else ?

- applying statistical model checking to other domains :
  - biological systems
  - real size applications (through SimGrid)

What else ?

- applying statistical model checking to other domains :
  - biological systems
  - real size applications (through SimGrid)
- an experiment on checking conflict freedom for multi threaded programs (something to talk about over lunch ?)

What else ?

- applying statistical model checking to other domains :
  - biological systems
  - real size applications (through SimGrid)
- an experiment on checking conflict freedom for multi threaded programs (something to talk about over lunch ?)
- starting collaboration with Pascale & Alexis
  - so far in UppAal (SMC)
  - next in COSMOS, or PRISM ?

The time varying cyclic scheduling problem

L. Houssin, I. Hamaz, S. Cafieri

LAAS-CNRS

June 8, 2017

Basic cyclic scheduling problem

Schedule elementary tasks $\mathcal{T} = \{1, ..., n\}$ **infinitely repeated**.

- Each task i has a processing time p_i

Basic cyclic scheduling problem

Schedule elementary tasks $\mathcal{T} = \{1, ..., n\}$ **infinitely repeated**.

- Each task i has a processing time p_i
- $< i, k >$ is the $k-th$ occurrence of i such that $k \in \mathbb{N}$

Basic cyclic scheduling problem

Schedule elementary tasks $\mathcal{T} = \{1, ..., n\}$ **infinitely repeated**.

- Each task i has a processing time p_i
- $< i, k >$ is the $k-th$ occurrence of i such that $k \in \mathbb{N}$
- $t(i, k)$ the start time of $< i, k >$

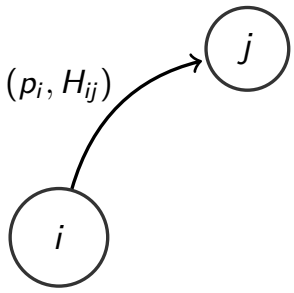
$$\forall i \in \mathcal{T}, \forall k \in \mathbb{N}: \quad t(i, k+1) = \alpha + t(i, k)$$

where α is the cycle time.

Uniform constraints

- Precedence constraint between i and j:

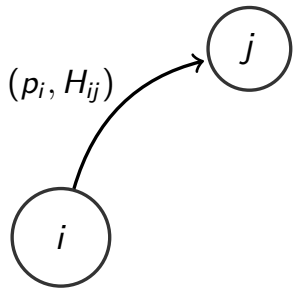
$$\forall i, j \in \mathcal{T}, \forall k \in \mathbb{N}: \quad t(i,k) + p_i \leq t(j, k + H_{ij}).$$



Uniform constraints

- Precedence constraint between i and j:

$$\forall i, j \in \mathcal{T}, \forall k \in \mathbb{N}: \quad t(i,k) + p_i \leq t(j, k + H_{ij}).$$



- Non reentrance constraint:

$$\forall i \in \mathcal{T}, \forall k \in \mathbb{N}: \quad t(i,k) + p_i \leq t(i, k+1).$$

Solving a cyclic scheduling problem

Goal

Minimize the cycle time α.

- find a cyclic schedule w where all $t(i,k)$ minimize α.

Solving a cyclic scheduling problem

Goal

Minimize the cycle time α.

- find a cyclic schedule w where all $t(i,k)$ minimize α.
- Starting times of occurrences of a task i are all connected:

$$\forall i \in \mathcal{T}, \forall k \in \mathbb{N}: \quad t(i,k) = t(i,0) + \alpha k.$$

Solving a cyclic scheduling problem

Cyclic schedule property

A cyclic schedule w is totally defined by

- a set $S_w = \{t(i,0) \in \mathbb{R}^+ \mid i \in \mathcal{T}\}$
- a cycle time α

Solving a cyclic scheduling problem

Cycle time

The minimum cycle time of the system is given by the maximum mean cycle of the graph that is defined by

$$\alpha = \max_{c \in \mathcal{C}} \rho(c)$$

where

$$\rho(c) = \frac{\sum_{(i,j) \in c} L_{ij}}{\sum_{(i,j) \in c} H_{ij}}$$

and $\mathcal{C}$ is the set of all circuits in G.

The cyclic job shop problem

The Cyclic Job Shop Problem (CJSP) is a cyclic scheduling problem characterized by:

- a set of r jobs $\mathcal{J} = \{J_1, ..., J_r\}$

The cyclic job shop problem

The Cyclic Job Shop Problem (CJSP) is a cyclic scheduling problem characterized by:

- a set of r jobs $\mathcal{J} = \{J_1, ..., J_r\}$
- a job J_s involves a set of elementary tasks $\mathcal{T}_{J_s} \subset \mathcal{T}$

The cyclic job shop problem

The Cyclic Job Shop Problem (CJSP) is a cyclic scheduling problem characterized by:

- a set of r jobs $\mathcal{J} = \{J_1, ..., J_r\}$
- a job J_s involves a set of elementary tasks $\mathcal{T}_{J_s} \subset \mathcal{T}$
- a limited number of machines $m < n$

The cyclic job shop problem

The Cyclic Job Shop Problem (CJSP) is a cyclic scheduling problem characterized by:

- a set of r jobs $\mathcal{J} = \{J_1, ..., J_r\}$
- a job J_s involves a set of elementary tasks $\mathcal{T}_{J_s} \subset \mathcal{T}$
- a limited number of machines $m < n$
- Elementary tasks are connected with *uniform* constraints and **disjunctive constraints**.

Disjunctive constraints

Resource constraints leads to

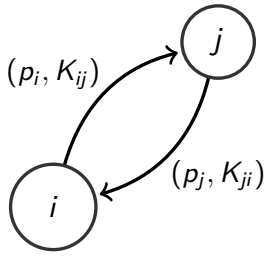
$\forall i, j \in \mathcal{T},\ \forall k, l \in \mathbb{N}$ such that $M(i) = M(j),\ i \neq j$ *and* $k \neq l$:

$$t(i,k) \leq t(j,l) \Rightarrow t(i,k) + p_i \leq t(j,l).$$

Occurrence shift property

For all machine M_s, if $i \in T_{M_s}$ and $j \in T_{M_s}$ then

$$K_{ij} + K_{ji} = 1$$



Scheduling with Work-In-Process $W > 1$

- In a schedule such that $W > 1$, W works-in-process are allowed.

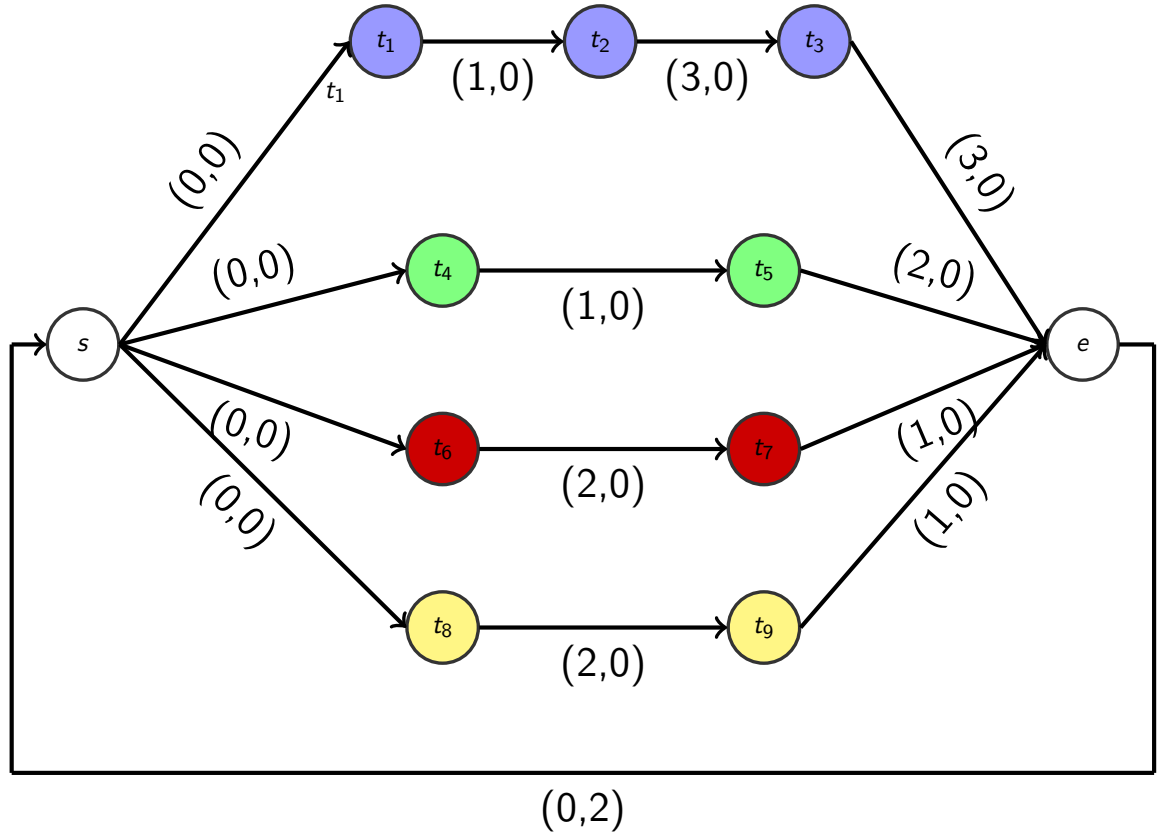
Example

CJSP Example

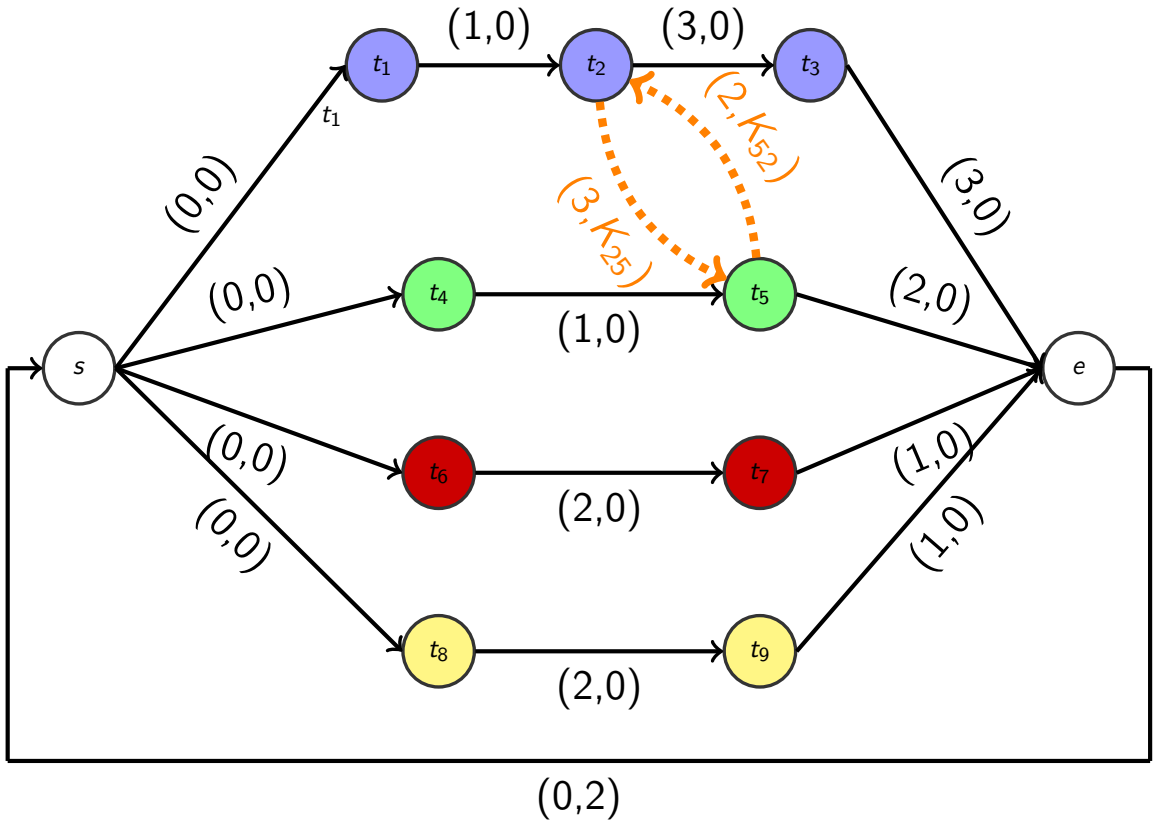
- Data:
  - 9 tasks
  - 4 Jobs
  - 3 machines

Job	J_1			J_2		J_3		J_4	
Task	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
Time	1	3	3	1	2	2	1	2	1
Machine	M_1	M_2	M_3	M_3	M_2	M_1	M_3	M_1	M_3

Associated graph of the CJSP (uniform constraints)

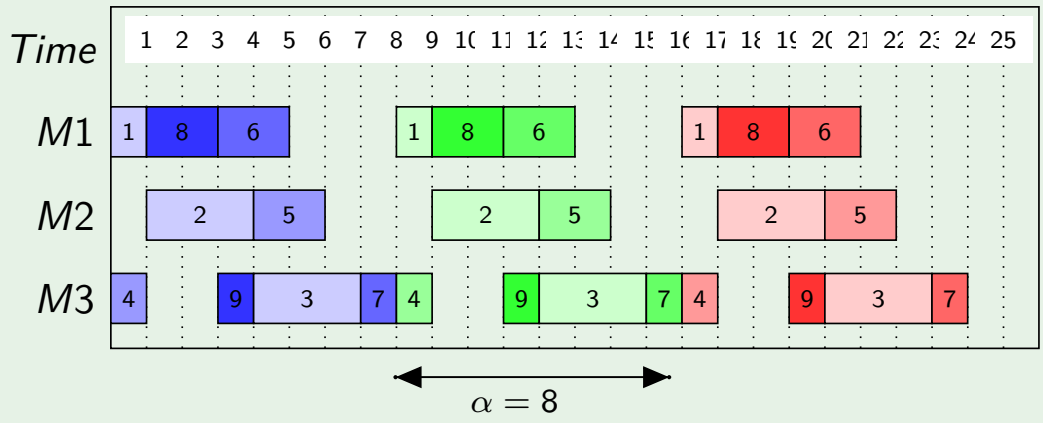


Associated graph of the CJSP (uniform constraints and 2 disjunctive constraints)



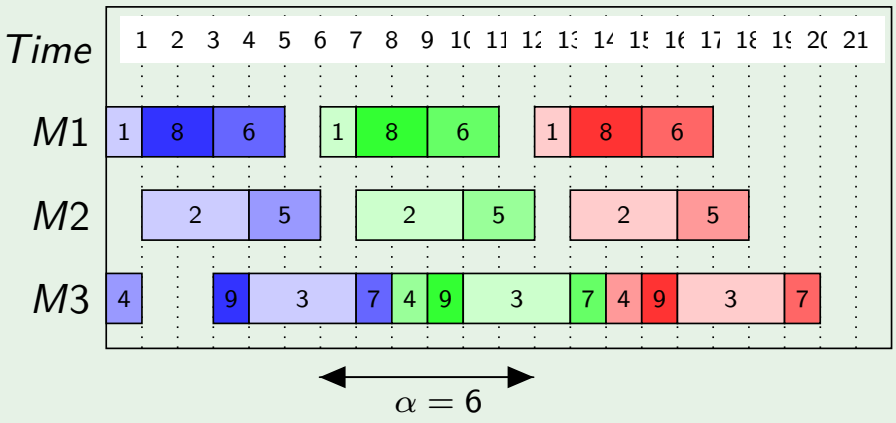
Example - cyclic schedule s.t. $WIP = 1$

Example of a 1-cyclic schedule such that $WIP = 1$



Example - cyclic schedule s.t. $WIP = 2$

Example of a 1-cyclic schedule such that $WIP = 2$



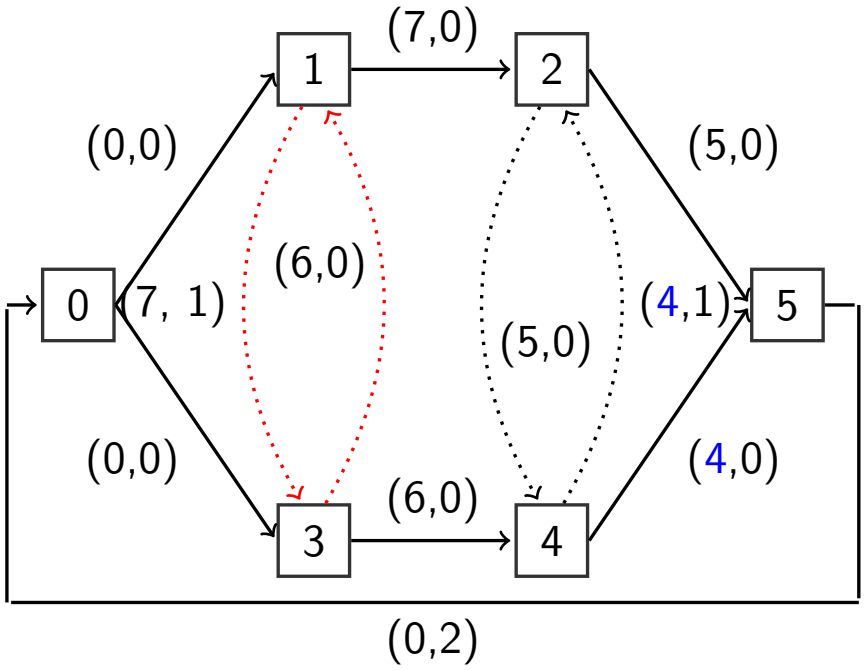
Sensitivity analysis of a cyclic schedule

How does the schedule behave when a subset of tasks has their processing time increasing?

Sensitivity analysis of a cyclic schedule

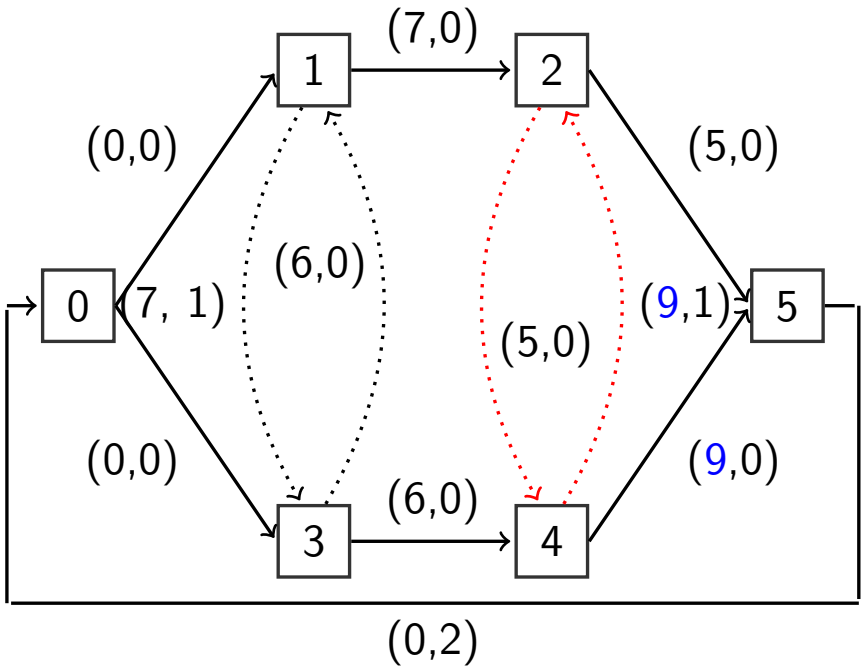
- A subset of tasks has variable processing times.
- $p_i \in [\underline{p_i}, \overline{p_i}]$ or $\overline{p_i} = \underline{p_i} + \delta_i$.
- What is the influence on the cycle time?

The critical circuit: a moving element



Critical circuit: $3 - 1 - 3$, $\alpha = 13$

The critical circuit: a moving element



Circuit critique : $2 - 4 - 2$, $\alpha = 14$

The critical circuit: a moving element

A circuit $\mathcal{C}$ is characterized by

- The sum of the edge's length of the circuit: $L_{\mathcal{C}}$
- The sum of the edge's height of the circuit: $H_{\mathcal{C}}$
- The own cycle time of the circuit $\alpha_{\mathcal{C}} = \frac{L_{\mathcal{C}}}{H_{\mathcal{C}}}$

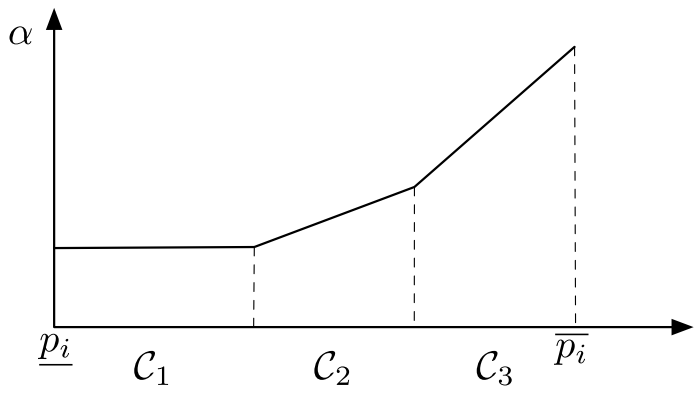
The cycle time of the schedule is the greatest cycle time of all circuits in the graph.

The critical circuit: a moving element

The cycle time of a circuit $\mathcal{C}$ including a time varying task i is moving as follows:

$$\alpha_{\mathcal{C}}(\delta_i) = \frac{L_{\mathcal{C}}}{H_{\mathcal{C}}} + \frac{\delta_i}{H_{\mathcal{C}}}$$

For a given schedule, the critical circuit can move according to the duration time of the task i.
Several circuits could be concerned.



Finding circuits in a graph

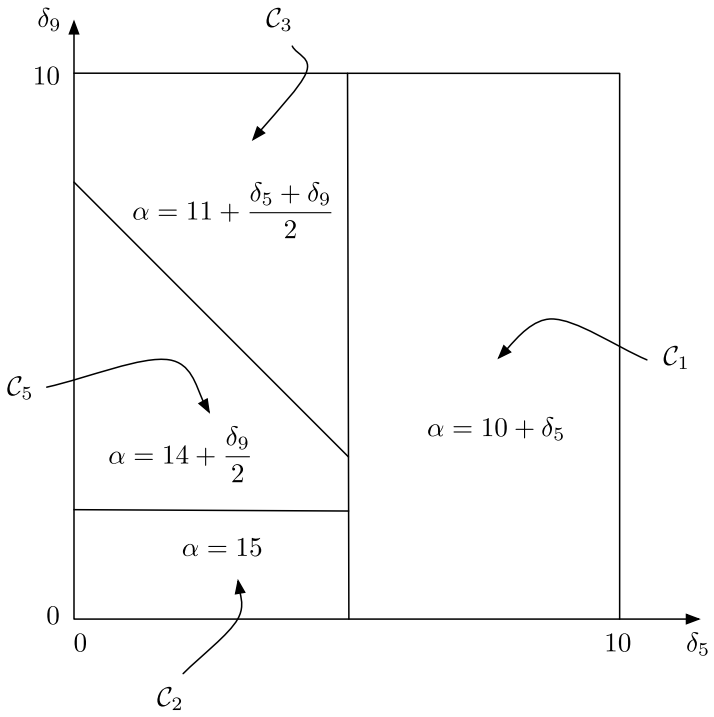
Although there is an exponential number of simple circuits in the graph, several algorithms can find them.

1. Tarjan
2. Johnson

Example

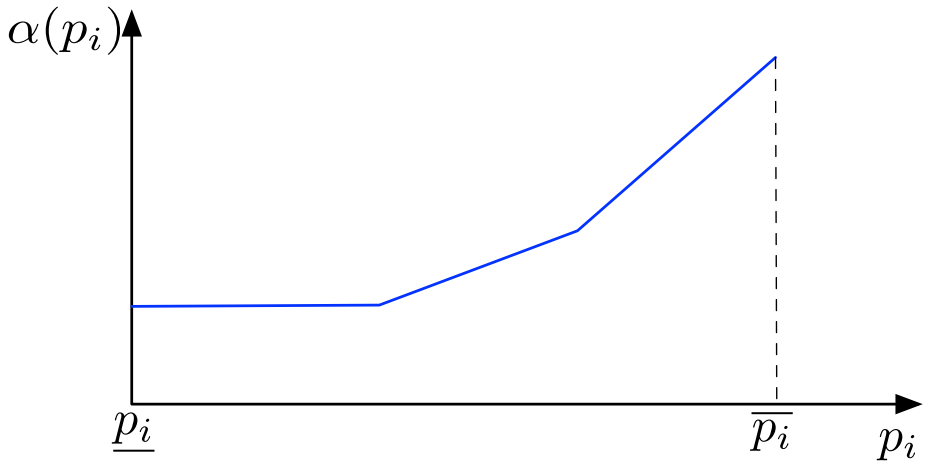
Tasks 5 et 9 have varying processing times: $\delta_5 \in [0, 10]$ and $\delta_9 \in [0, 10]$. Circuits implied with respect to the variation.

Circuit	Length	Height	Circuit
$\mathcal{C}_1$	10	1	1-2-5-8-10-1
$\mathcal{C}_2$	15	1	3-8-11-3
$\mathcal{C}_3$	22	2	2-5-9-2
$\mathcal{C}_4$	26	2	2-6-2
$\mathcal{C}_5$	28	2	3-7-9-3


δ_9
$\mathcal{C}_3$
10
$\alpha = 11 + \frac{\delta_5 + \delta_9}{2}$
$\mathcal{C}_1$
$\mathcal{C}_5$
$\alpha = 14 + \frac{\delta_9}{2}$
$\alpha = 10 + \delta_5$
$\alpha = 15$
0
0
10
δ_5
$\mathcal{C}_2$

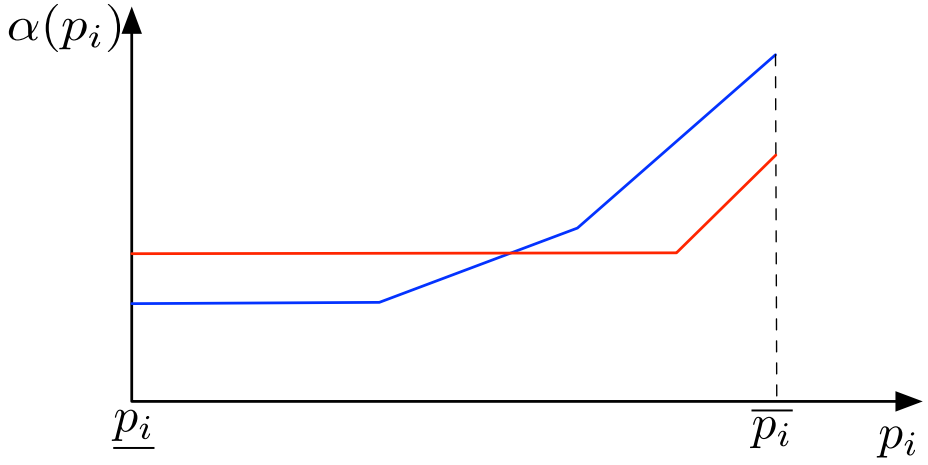
Evaluation of a schedule

Let us consider $p_i = \underline{p_i}$. The optimal schedule is $S_{\underline{p_i}}$.



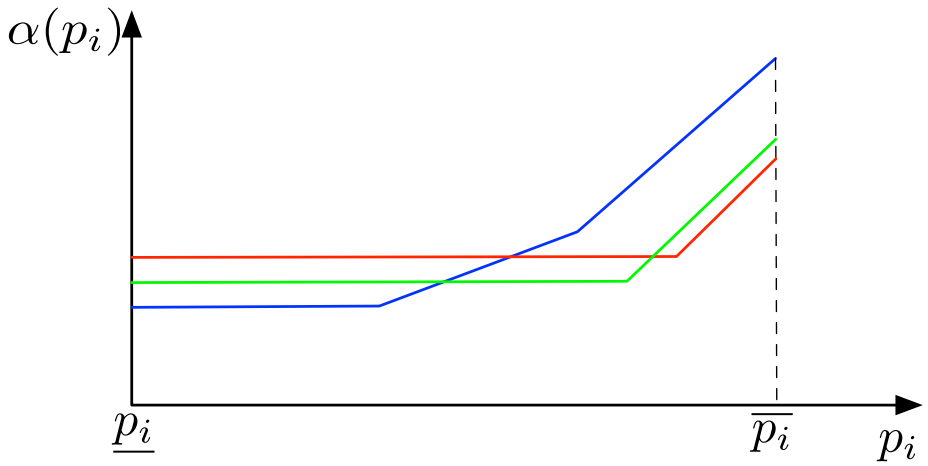
Evaluation of a schedule

Let us consider $p_i = \overline{p_i}$. The optimal schedule is $S_{\overline{p_i}}$.



Evaluation of a schedule

Other schedules could be of interest.

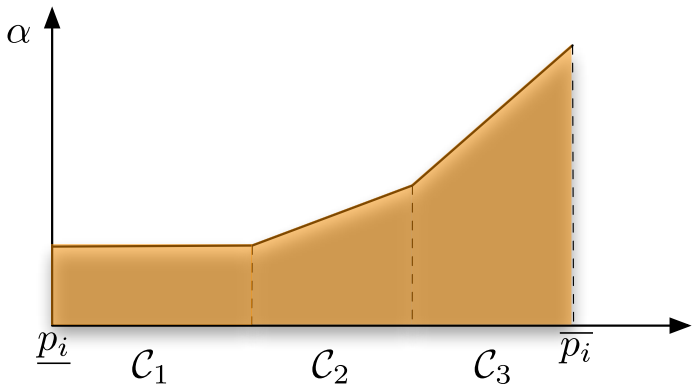


Evaluation of a schedule

How to decide? Which criterion should we consider?

Evaluation of a schedule

How to decide? Which criterion should we consider?



Minimize the volume of the polytope.

$$min \quad V(S) = \int_{\underline{p_i}}^{\overline{p_i}} \alpha(p_i) dp_i.$$

Evaluation of a schedule

How to compute the volume of the polytope?

.

Evaluation of a schedule

How to compute the volume of the polytope?
Library VINCI

.

Evaluation of a schedule

How to compute the volume of the polytope?
Library VINCI
Just need the faces: $Ax \leq b$.

- The objective is to find the schedule that produces the smallest polytope. We can use a branch and bound procedure to solve the problem.

- The objective is to find the schedule that produces the smallest polytope. We can use a branch and bound procedure to solve the problem.
- We branch on the occurrence shifts. When an occurrence shift is fixed, the disjunction is solved.

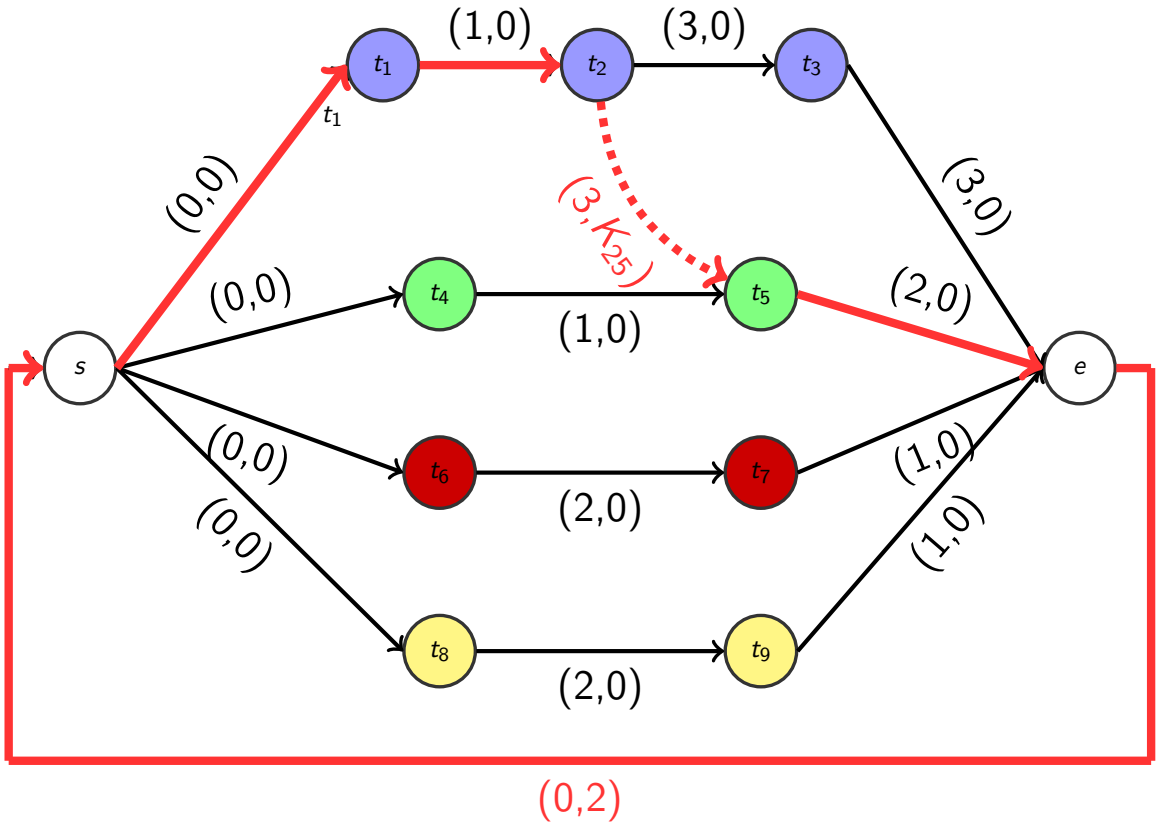
The branch and bound scheme

- A schedule is consistent if there is no circuit with non-positive height.

The branch and bound scheme

- A schedule is consistent if there is no circuit with non-positive height.
- Each disjunction must not compromise the consistency of the schedule.

The branch and bound scheme



We can deduce bounds on the occurrence shifts.

- each occurrence shift K_{ij} has a lower bound that guarantees the consistence of the graph:

$$K_{ij}^{-} = 1 - \min\{H(\mu) \mid \mu \text{ path from } j \text{ to } i \text{ in } G\}.$$

We can deduce bounds on the occurrence shifts.

- each occurrence shift K_{ij} has a lower bound that guarantees the consistence of the graph:

$$K_{ij}^{-} = 1 - \min\{H(\mu) \mid \mu \text{ path from } j \text{ to } i \text{ in } G\}.$$

- since $K_{ij} + K_{ji} = 1$, we can easily deduce an upper bound:

$$K_{ij}^{-} \leq K_{ij} \leq 1 - K_{ji}^{-}$$

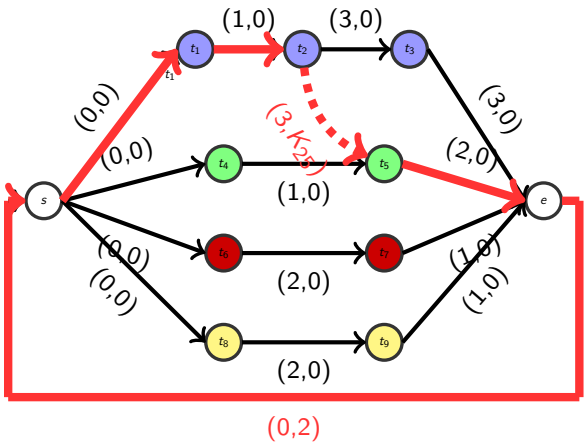
Example

Lower bound K_{25}

New circuit $C = (t_5, e, s, t_1, t_2,)$ such that
$H(C) = 0 + 0 + K_{25} + 0 + 2.$

since $H(C) \geq 1$.

We deduce the following lower bound: $K_{25}^{-} = -1$



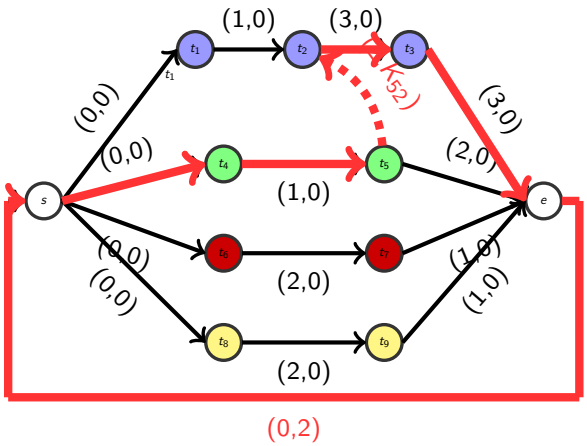
Bounds on occurrence shifts: Example

Lower bound K_{52}

New circuit
$C' = (t_2, t_3, e, s, t_4, t_5)$
such that
$H(C') = 0 + 0 + K_{52} + 0 + 0 + 2.$

since $H(C') \geq 1$

We have the following lower bound: $K_{52}^{-} = -1$



Bounds on event shifts: Example

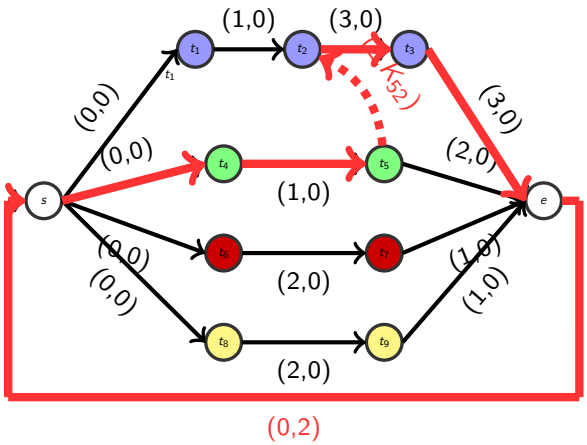
Bounds on K_{25} and K_{52}

Since $K_{ij}^{-} \leq K_{ij} \leq 1 - K_{ji}^{-}$
we have

$$-1 \leq K_{25} \leq 2$$

and

$$-1 \leq K_{52} \leq 2$$



Example

CJSP Example

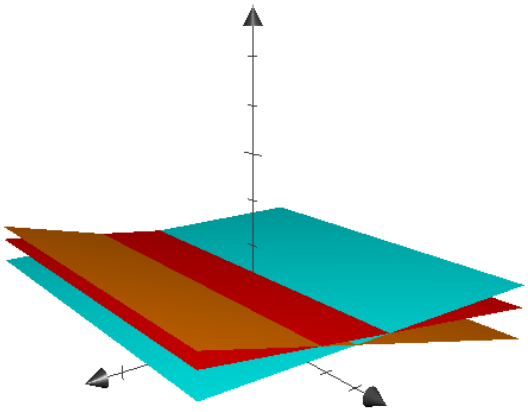
- Data:
  - 8 tasks
  - 3 Jobs
  - 2 machines

Job	J_1	J_2				J_3		
Task	t_6	t_1	t_5	t_7	t_8	t_2	t_3	t_4
Time	3	9	8	10	3	1	1	4
Machine	M_2	M_1	M_2	M_2	M_1	M_2	M_2	M_2

  - 2 tasks with varying processing time: $p_3 \in \{1, 10\}$ and $p_5 \in \{8, 17\}$

Example

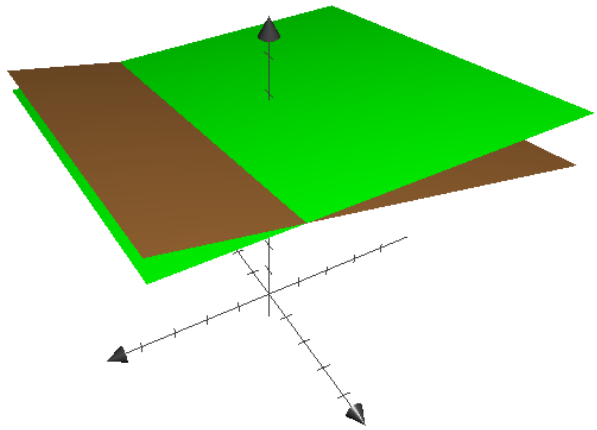
Optimal schedule for $p_3 = 1$ and $p_5 = 8$.



$V(S_{\underline{p}}) = 3159$,
mean value of α with respect to the variation: 39.

Example

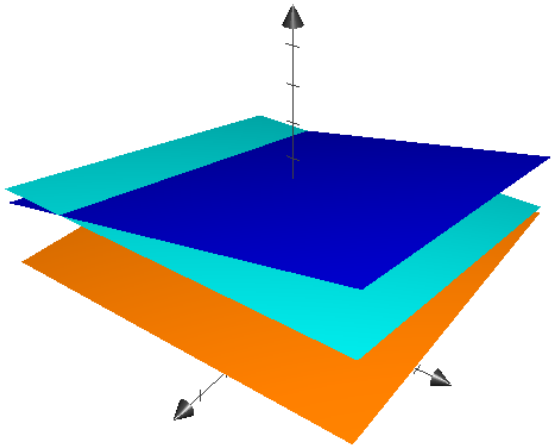
Optimal schedule for $p_3 = 10$ and $p_5 = 17$.



$V(S_{\overline{p}}) = 3136.5$,
mean value of α with respect to the variation: 38.72

Example

Optimal schedule for $p_3 \in \{1, 10\}$ and $p_5 \in \{8, 17\}$.



$V(S_{opt}) = 2956$,
mean value of α with respect to the variation: 36.5

Example

```
 Optimum Found = 1
First Feasible Solution = 3136.5
Best = 2956.5
Associate cycle time = 36.5
Number Visited Nodes = 31
Number Feasible Nodes = 6
Elapsed Time = 1.647
```

Perspectives

Perspectives

- Improve the search of circuits.

Perspectives

- Improve the search of circuits.
- Consider other criteria.

Perspectives

- Improve the search of circuits.
- Consider other criteria.
  - Maximize the number of scenarii where the cycle time is lower than a given value.

Perspectives

- Improve the search of circuits.
- Consider other criteria.
  - Maximize the number of scenarii where the cycle time is lower than a given value.
  - Maximize the range of δ around a nominal value of a vector p where the cycle time remains unchanged.

Part II: Robust optimization applied to the basic cyclic scheduling problem

Robust optimization

Optimization under uncertainties

- Stochastic optimization
- Robust optimization

Robust optimization

Deterministic LP

$$\begin{aligned} \min \quad & Cx \\ s.t. \quad & Ax \geq b \\ & x \in X \end{aligned}$$

Robust optimization

Some parameters belong to an uncertainty set Ξ. We aim to optimize some criteria over the uncertainty set.

Robust optimization

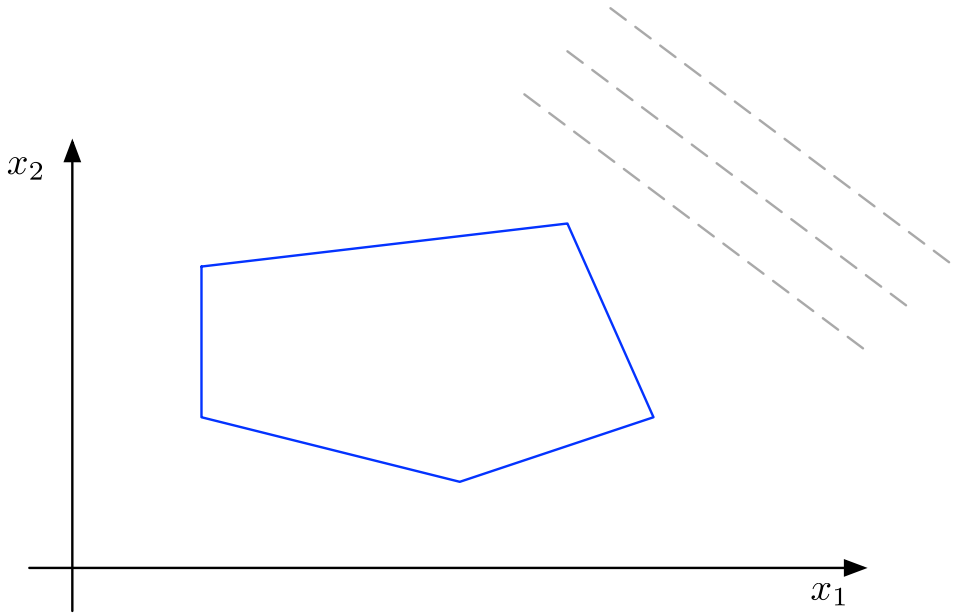
Robust optimization

Some parameters belong to an uncertainty set Ξ. We aim to optimize some criteria over the uncertainty set.

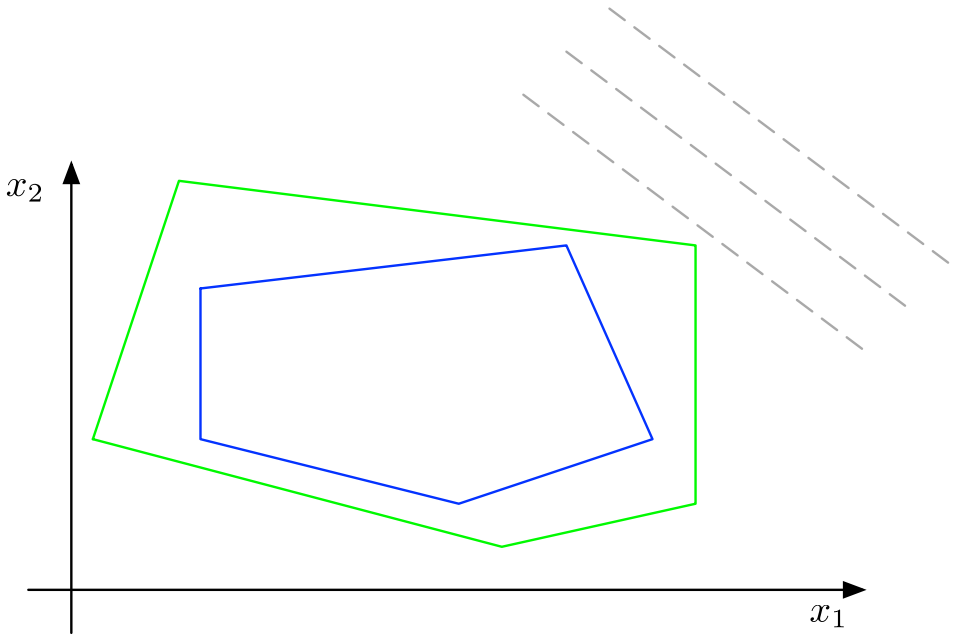
$$a_{ij} = \overline{a}_{ij} + \widehat{a}_{ij} z_{ij}$$

$$z_{ij} \in [-1, 1]$$

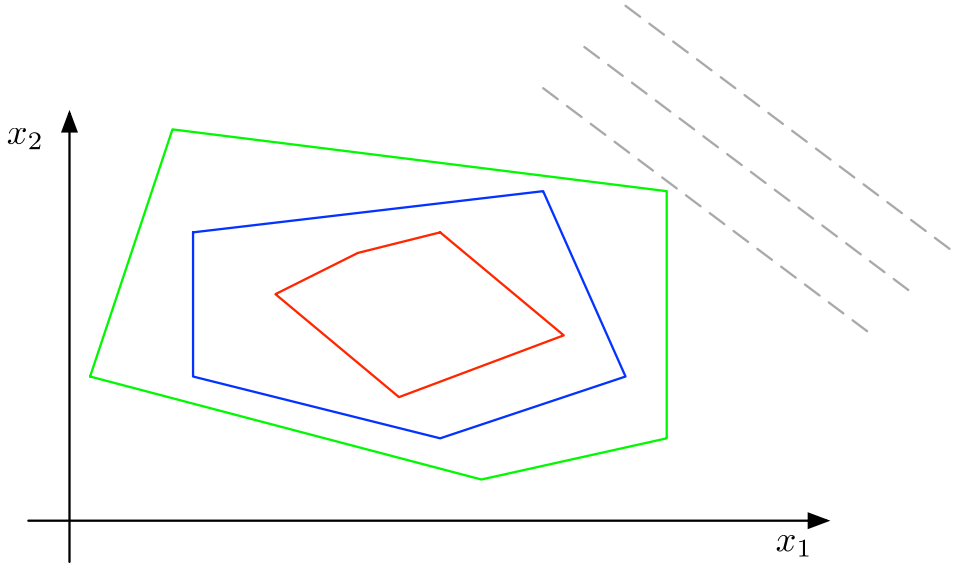
Robust optimization



Robust optimization



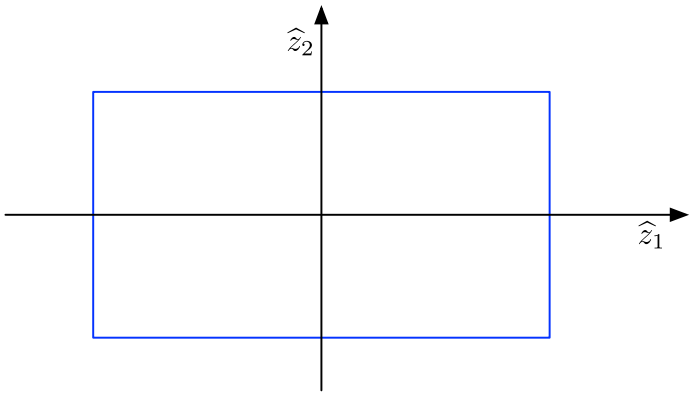
Robust optimization



Robust optimization

Robust optimization

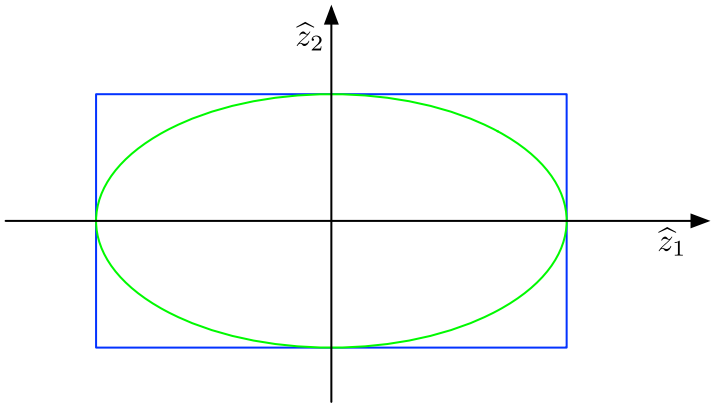
- Conservative approach (Soyster 1973)



Robust optimization

Robust optimization

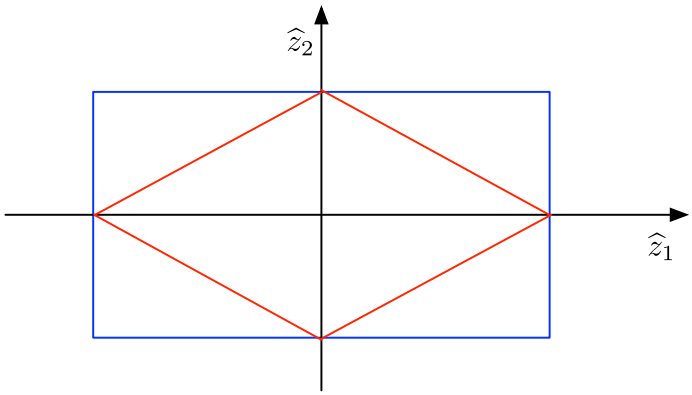
- Conservative approach (Soyster 1973)
- Ellipsoidal approach (Ben Tal & Nimerovski 1999-2000)



Robust optimization

Robust optimization

- Conservative approach (Soyster 1973)
- Ellipsoidal approach (Ben Tal & Nimerovski 1999-2000)
- Bertsimas and Sim approach (Berstimas & Sim 2004)



Two-stage robust optimization

Two-stage robust optimization

Two groups of decision variables are considered:

- First stage : All decisions are taken before knowing the uncertainty.
- Second stage : delayed until the uncertainty has been revealed.

$$x \longrightarrow \xi \longrightarrow y(\xi)$$

Two-stage robust optimization

Two-stage robust optimization

Two groups of decision variables are considered:

- First stage : All decisions are taken before knowing the uncertainty.
- Second stage : delayed until the uncertainty has been revealed.

$$x \longrightarrow \xi \longrightarrow y(\xi)$$

two-stage LP (Determinist)

$$\begin{aligned} \min \quad & cx + qy \\ s.t. \quad & Tx + Wy = h \\ & x \in X \\ & y \geq 0 \end{aligned}$$

Two-stage robust optimization

Two-stage robust optimization

Two groups of decision variables are considered:

- First stage : All decisions are taken before knowing the uncertainty.
- Second stage : delayed until the uncertainty has been revealed.

$$x \longrightarrow \xi \longrightarrow y(\xi)$$

two-stage LP (Determinist)

$$\begin{aligned} \min \quad & cx + qy \\ s.t. \quad & Tx + Wy = h \\ & x \in X \\ & y \geq 0 \end{aligned}$$

Two-stage robust LP

$$\begin{aligned} \min \quad & \max_{\xi \in \Xi} \{cx + q(\xi)y(\xi)\} & \\ s.t. \quad & T(\xi)x + W(\xi)y(\xi) = h(\xi) & \forall \xi \in \Xi \\ & x \in X & \\ & y(\xi) \geq 0 & \forall \xi \in \Xi \end{aligned}$$

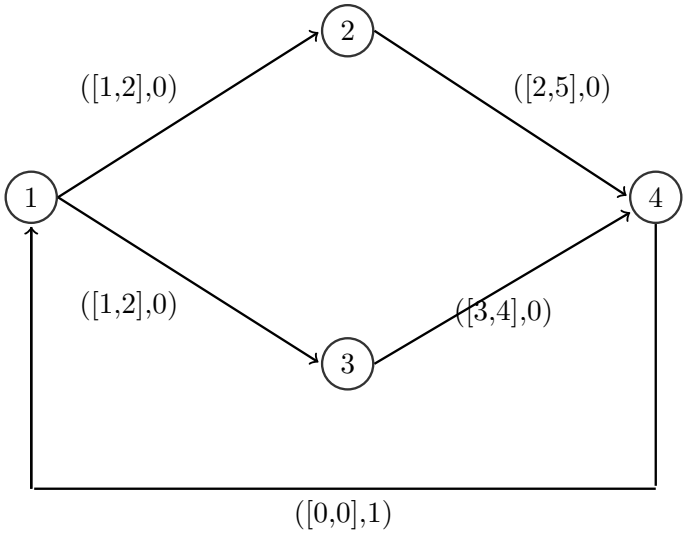
Uncertainty set

- The processing time p_i of each task i belongs to the interval $[\bar{p}_i, \bar{p}_i + \hat{p}_i]$

$$p_i(\xi) = \bar{p}_i + \xi_i \hat{p}_i, \forall \xi \in \Xi, i \in T$$

$$\text{with } \Xi = \left\{ (\xi_i)_{1 \leq i \leq T} \middle| \sum_{i=1}^{T} \xi_i \leq \Gamma, \xi_i \in \{0, 1\} \right\}$$

Associated graph for example 1



- The determinist cycle time value is $\alpha = 4$
- The robust cycle time (with $\Gamma = 1$) value is $\alpha = 6$

Mathemathical formulation (as two-stage model)

- The BCSP can be considered as a two stage optimization problem

$$\begin{aligned} \min \quad & \alpha \\ s.t. \quad & t_j(\xi) - t_i(\xi) + \alpha.H_{ij} \geq p_i(\xi) \quad \forall (i,j) \in E, \xi \in \Xi \end{aligned}$$

Decision variables

- α : first stage decision variable (before knowing the realization of the uncertainty)
- $(t)_{1 \leq i \leq \mathcal{T}}$: second stage decision variables (after the uncertainty realization)

Separation problem

- For a given cycle time $\bar{\alpha}$, determine whether the precedence constraints can be satisfied for each realization ξ or not.

$$t_j(\xi) - t_i(\xi) \geq p_i(\xi) - \bar{\alpha}.H_{ij} \quad \forall (i,j) \in E, \xi \in \Xi$$

Separation problem

- According to Farkas Lemma, a given cycle time α is feasible if and only if the optimal solution of the following optimization problem is non-positive

$$
\begin{aligned}
\max \quad & \sum_{e \in E} (p_e(\xi) - H_e \bar{\alpha}) u_e \\
s.t. \quad & \xi \in \Xi \\
& \sum_{e \in \sigma^-(v)} u_e - \sum_{e \in \sigma^+(v)} u_e = 0 \quad \forall v \in V \\
& u_e \geq 0 \qquad\qquad\qquad \forall e \in E
\end{aligned}
$$

Separation problem

- According to Farkas Lemma, a given cycle time α is feasible if and only if the optimal solution of the following optimization problem is non-positive

$$
\begin{aligned}
\max \quad & \sum_{e \in E} (p_e(\xi) - H_e \bar{\alpha}) u_e \\
s.t. \quad & \xi \in \Xi \\
& \sum_{e \in \sigma^-(v)} u_e - \sum_{e \in \sigma^+(v)} u_e = 0 \quad \forall v \in V \\
& u_e \geq 0 \qquad \forall e \in E
\end{aligned}
$$

- The solution of the problem is a circulation flow.

Separation problem

Property

A circulation u can be represented as a cycle flow along at most m directed cycles.

- We denote $A_e(\xi) = p_e(\xi) - H_e\bar{\alpha}$ be the amplitude of the arc e.
- Let $G_{\bar{\alpha}}(E, A)$ the graph where each arc e has $A_e(\xi)$ as value.
  The separation problem is feasible $\iff$ the graph $G_{\bar{\alpha}}$ has no circuit with positive amplitude for all $\xi \in \Xi$.

Separation problem

The cycle time α is feasible if and only if $G_{\bar{\alpha}}(E, A)$ has no circuit with positive cycle.

The cycle time α is feasible if and only if $G_{\bar{\alpha}}(E, -A)$ has no circuit with negative cycle.

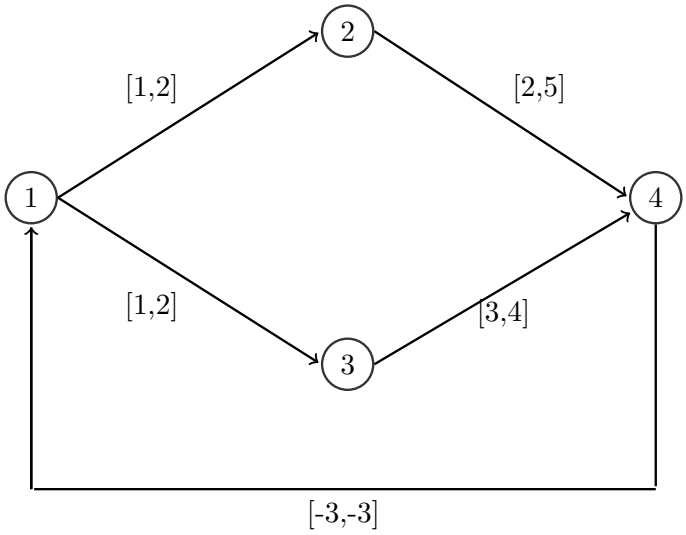
Separation problem

The cycle time α is feasible if and only if $G_{\bar{\alpha}}(E, A)$ has no circuit with positive cycle.

The cycle time α is feasible if and only if $G_{\bar{\alpha}}(E, -A)$ has no circuit with negative cycle.

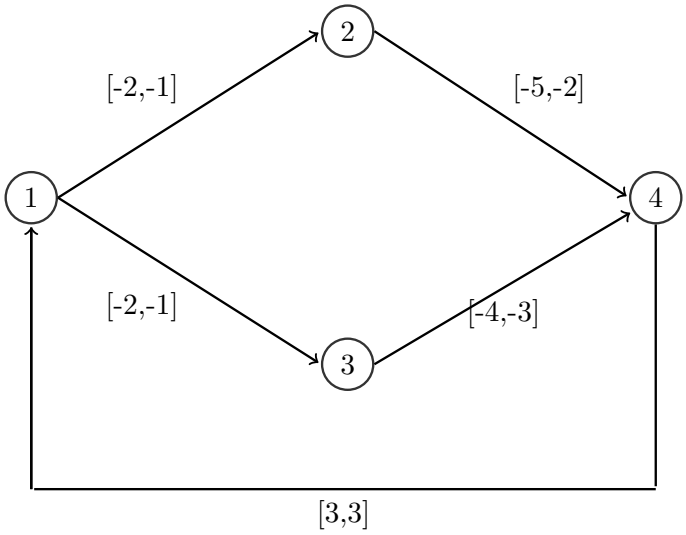
- The bellman-Ford algorithm with cycle detection can be extended to compute to shortest path with at least Γ deviations, and detect a negative cycle if it exists.

Associated graph for example 1



$\bar{\alpha} = 3$

Associated graph for example 1



$\bar{\alpha} = 3$

Separation problem

Definition

$d_i[\gamma]$: The value of the shortest path from a source s to i with uncertainty budget γ.

- Dynamic program recursion:

$$\forall j \in \mathcal{T} \backslash S, \forall \gamma = 0..\Gamma :$$

$$d_j[\gamma] = \min_{i \in pred(j)} \min\{d_i[\gamma - 1] + \bar{c}_{ij} + \hat{c}_{ij}, d_i[\gamma] + \bar{c}_{ij}\}$$

Separation problem

Algorithm 1 Bellman-Ford's algorithm

Input: A graph $G(V, E, c)$.
Output: Return "True" if negative cycle exists, "False" else

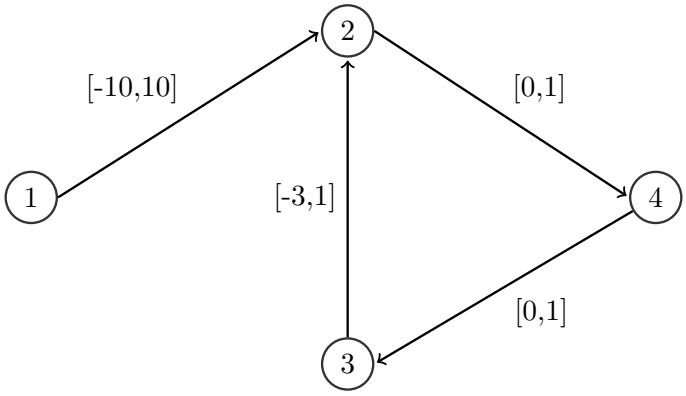
```
Negative = False‘
for all γ = 0 .. Γ do
    d_s[γ] = 0
for all j ∈ V\s do
    d_j[γ] = ∞
for all j = 1..|V| − 1 do
    for all γ = 0 .. Γ do
        for all (i, j) ∈ E do
            Scan(i,j,γ )
            if d_s[γ] < 0 then
                Negative = True
            end if
return Negative
```

Negative cycle detection algorithm

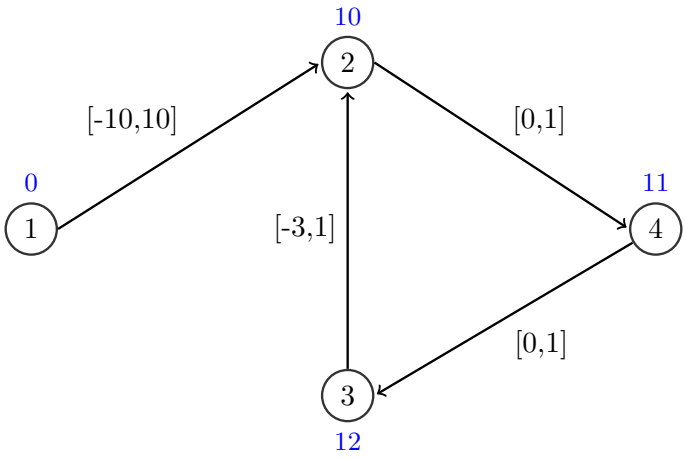
Algorithm 2 scan (i,j,γ)

$d_j[\gamma] = \min_{i \in pred(j)} \min\{d_i[\gamma - 1] + \bar{c}_{ij} + \hat{c}_{ij}, d_i[\gamma] + \bar{c}_{ij}\}$
$Pred_j[\gamma] = \text{argmin}_{i \in pred(j)} \min\{d_i[\gamma - 1] + \bar{c}_{ij} + \hat{c}_{ij}, d_i[\gamma] + \bar{c}_{ij}\}$
if $d_j[\gamma] = d_i[\gamma - 1] + \bar{c}_{i,j} + \hat{c}_{i,j}$ **then**
    $\xi_j[\gamma] = 1$
end if

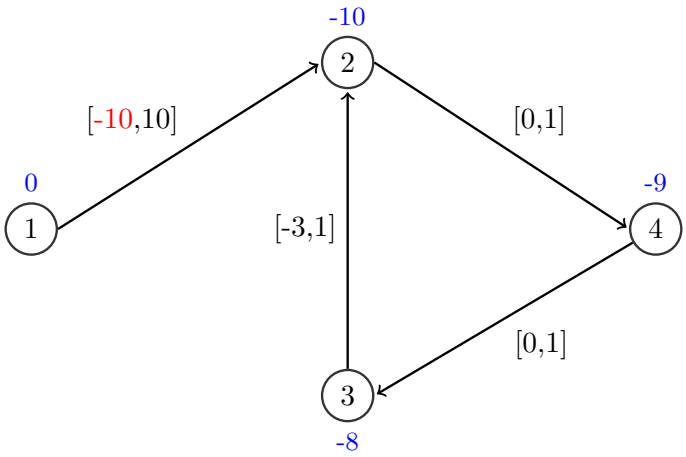
Example



Example



Example



Negative cycle detection algorithm (complexity)

Remark

The algorithm detects only cycles containing the source node $S \Rightarrow$ the algorithm must be performed with each node as a source

- Complexity $\mathcal{O}(n^2 m\Gamma)$.

Algorithm

Algorithm 3 RBCSP

1: Start with a lower bound $\bar{\alpha}$.
2: Run the negative cycle detection algorithm with $G_{\bar{\alpha}}(E, -A)$.
3: If a negative cycle c is detected
Then improve the cycle time c ($\bar{\alpha} = \rho(c)$).
return to **step 2**
Else the cycle time $\bar{\alpha}$ is optimal.

Complexity

- The algorithm run in $O(n^2.m.\Gamma.N)$ (N : The number of simple cycles of the graph).

Remark

A binary search algorithm can adapted to resolve the robust BCSP and this algorithm run in $O(n^2.m.\Gamma.lg(n.P_{max}.H_{max}))$

Conclusion

- We proposed an algorithm to resolve the robust basic cyclic scheduling problem

Conclusion

- We proposed an algorithm to resolve the robust basic cyclic scheduling problem
- Perform numerical experementation.

Conclusion

- We proposed an algorithm to resolve the robust basic cyclic scheduling problem
- Perform numerical experementation.
- Improve the negative circuit dectection (other strategies?)

Conclusion

- Perform numerical experementation.
- Improve the negative circuit dectection (other strategies?)
- Consider extensions of the problem such as the cyclic scheduling problem with resource constraints (Jobshop problem).

Model-checking Real-time Systems with Roméo

Journée FCH – Approches à base de SED pour l'ordonnancement de tâches manufacturières ou informatiques

Didier Lime

Based on work with Étienne André, Hanifa Boucheneb, Aleksandra Jovanović, and Olivier H. Roux, Charlotte Seidner, Louis-Marie Traonouez

École Centrale de Nantes – LS2N

Nancy, June 8th, 2017

A Scheduling Problem (adapted from [BFSV04])

- Three real-time tasks τ_1, τ_2 and τ_3:
  - τ_1 is periodic with period 50 and execution time $C_1 \in [10, 20]$;
  - τ_2 is sporadic with min. delay 100 and execution time $C_2 \in [18, 28]$;
  - τ_3 is periodic with period 150 and execution time $C_3 \in [20, 28]$;
- Scheduling policy: priority with $\tau_1 > \tau_2 > \tau_3$ (non preemptive);
- Is it **schedulable**?
  Each task always has at most one instance running

A Scheduling Problem (adapted from [BFSV04])

- Three real-time tasks τ_1, τ_2 and τ_3:
  - τ_1 is periodic with period 50 and execution time $C_1 \in [10, 20]$;
  - τ_2 is sporadic with min. delay 100 and execution time $C_2 \in [18, 28]$;
  - τ_3 is periodic with period 150 and execution time $C_3 \in [20, 28]$;
- Scheduling policy: priority with $\tau_1 > \tau_2 > \tau_3$ (non preemptive);
- Is it **schedulable**?
  Each task always has at most one instance running
- It is! (we will see that later)

A Scheduling Problem (adapted from [BFSV04])

- Three real-time tasks τ_1, τ_2 and τ_3:
  - τ_1 is periodic with period 50 and execution time $C_1 \in [10, 20]$;
  - τ_2 is sporadic with min. delay 100 and execution time $C_2 \in [18, 28]$;
  - τ_3 is periodic with period 150 and execution time $C_3 \in [20, 28]$;
- Scheduling policy: priority with $\tau_1 > \tau_2 > \tau_3$ (non preemptive);
- Is it **schedulable**?
  Each task always has at most one instance running
- It is! (we will see that later)
- Do it for:
  - Complex interactions $\rightarrow$ **formal models**;
  - Timing uncertainty $\rightarrow$ **timed models**;
  - preemptive scheduling $\rightarrow$ **stopwatches**;
  - Design uncertainty $\rightarrow$ **parameters**;
  - Soft real-time constraints, or energy constraints $\rightarrow$ **cost optimisation**.

Roméo

- Roméo is a tool for the verification of Time Petri Nets;
- Developed since 2001 by Olivier H. Roux and Didier Lime;
- Written in C++ (engine, ∼24K loc) and Tcl/Tk (GUI, ∼18K loc);
- Distributed under the terms of the CeCILL **open source** license;
- Supports discrete variables, stopwatches, costs, parameters.
- Available at:

`http://romeo.rts-software.org`

Roméo: Some Success Stories

- Analysis of resilience properties in oscillatory **biological** systems [AMI16];
- Environment requirements for an aerial **video tracking** system (with Thales Research) [PRH+16];
- **Operational scenarios** modelling in the DGA OMOTESC project (with Sodius Nantes, Charlotte Seidner's Ph. D.) [Sei09].

Outline

Introduction

Time Petri Nets

Stopwatch Petri Nets

Timing Parameters

Minimal Cost Reachability

Conclusion

Outline

Introduction

Time Petri Nets

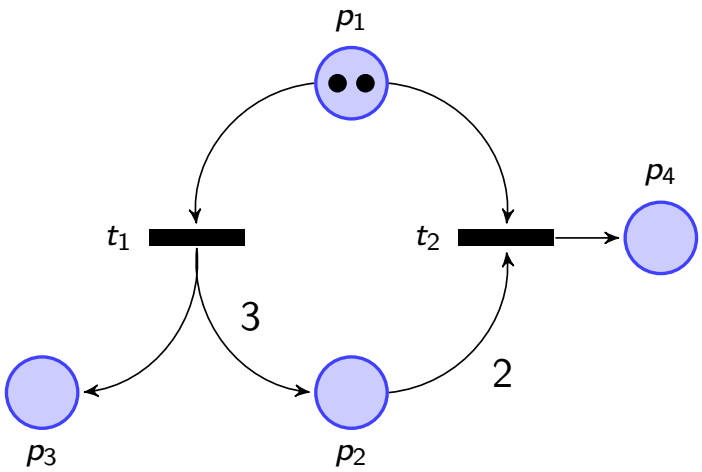
Stopwatch Petri Nets

Timing Parameters

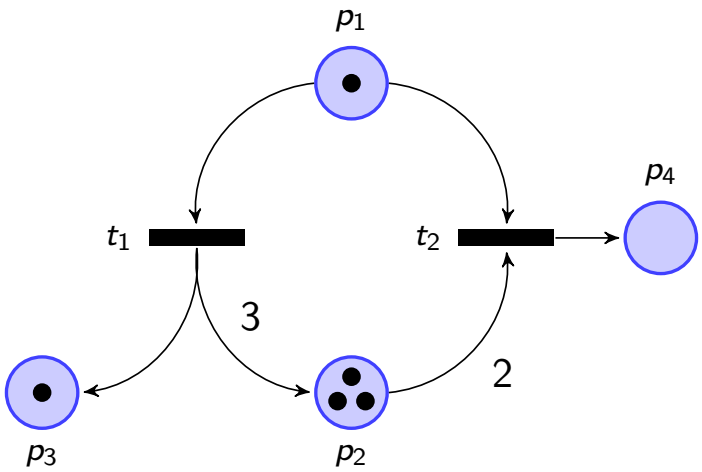
Minimal Cost Reachability

Conclusion

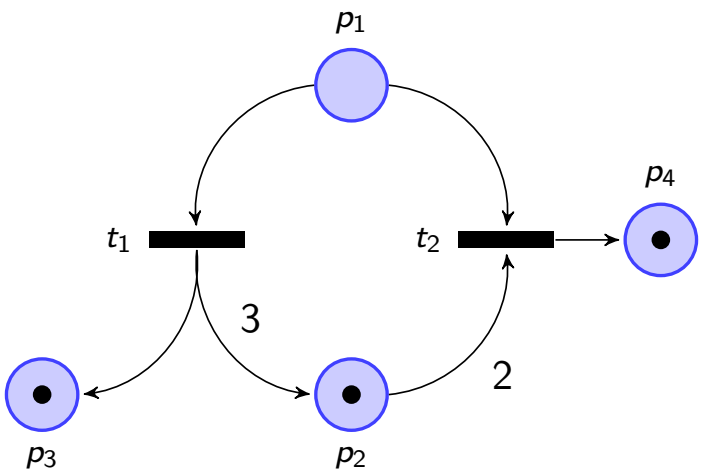
Petri Nets



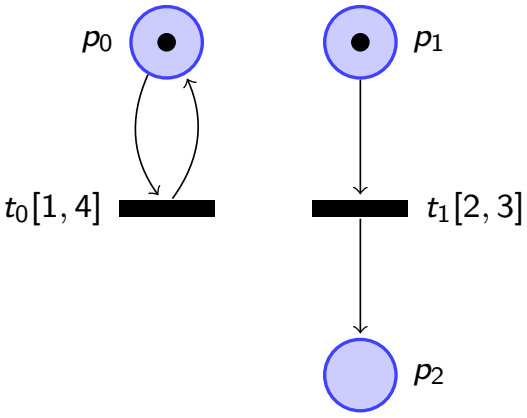
Petri Nets



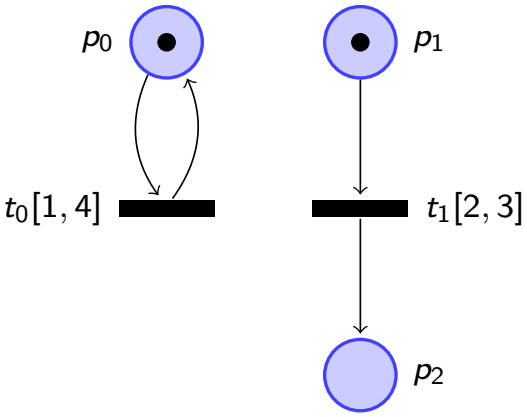
Petri Nets



Time Petri Nets

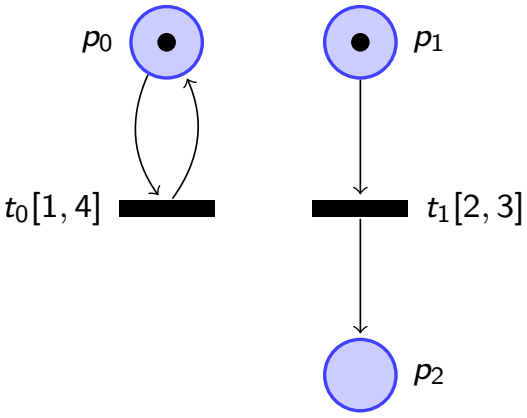


Time Petri Nets



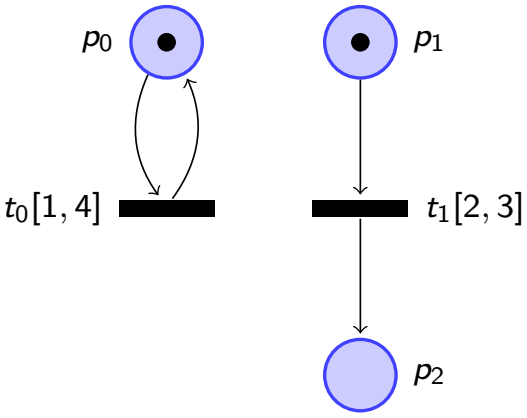
$t_0 \in [1,4]$
$t_1 \in [2,3]$

Time Petri Nets



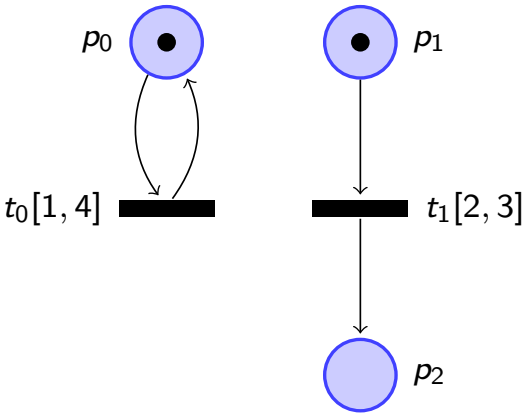
$$\begin{array}{l} t_0 \in [1,4] \\ t_1 \in [2,3] \end{array} \xrightarrow{1.1} \begin{array}{l} [0, 2.9] \\ [0.9,\ 1.9] \end{array}$$

Time Petri Nets



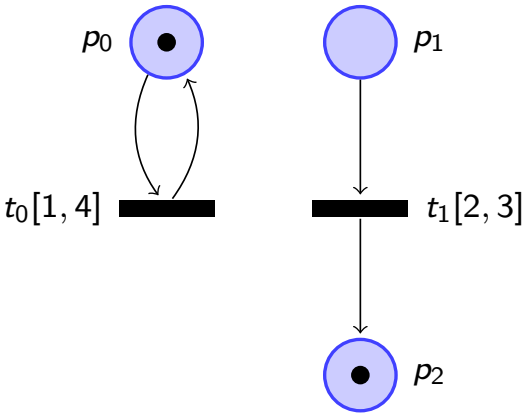
$$\begin{matrix} t_0 \in [1,4] \\ t_1 \in [2,3] \end{matrix} \xrightarrow{1.1} \begin{matrix} [0, 2.9] \\ [0.9,\ 1.9] \end{matrix} \xrightarrow{t_0} \begin{matrix} [1,4] \\ [0.9,\ 1.9] \end{matrix}$$

Time Petri Nets



$$\begin{matrix} t_0 \in [1,4] \\ t_1 \in [2,3] \end{matrix} \xrightarrow{1.1} \begin{matrix} [0,2.9] \\ [0.9,\ 1.9] \end{matrix} \xrightarrow{t_0} \begin{matrix} [1,4] \\ [0.9,\ 1.9] \end{matrix} \xrightarrow{1.9} \begin{matrix} [0,2.1] \\ [0,\ 0] \end{matrix}$$

Time Petri Nets

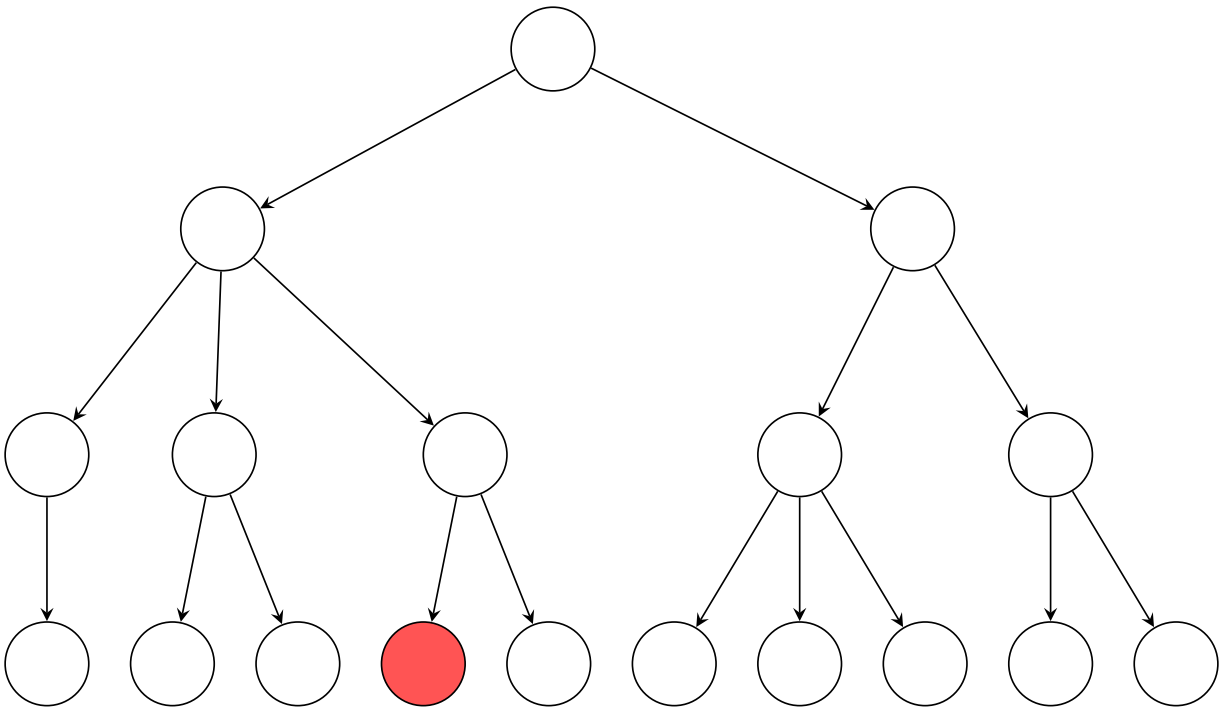


$$\begin{matrix} t_0 \in [1,4] \\ t_1 \in [2,3] \end{matrix} \xrightarrow{1.1} \begin{matrix} [0,2.9] \\ [0.9,\ 1.9] \end{matrix} \xrightarrow{t_0} \begin{matrix} [1,4] \\ [0.9,\ 1.9] \end{matrix} \xrightarrow{1.9} \begin{matrix} [0,2.1] \\ [0,\ 0] \end{matrix} \xrightarrow{t_1} [0,\ 2.1]$$

Basic Properties

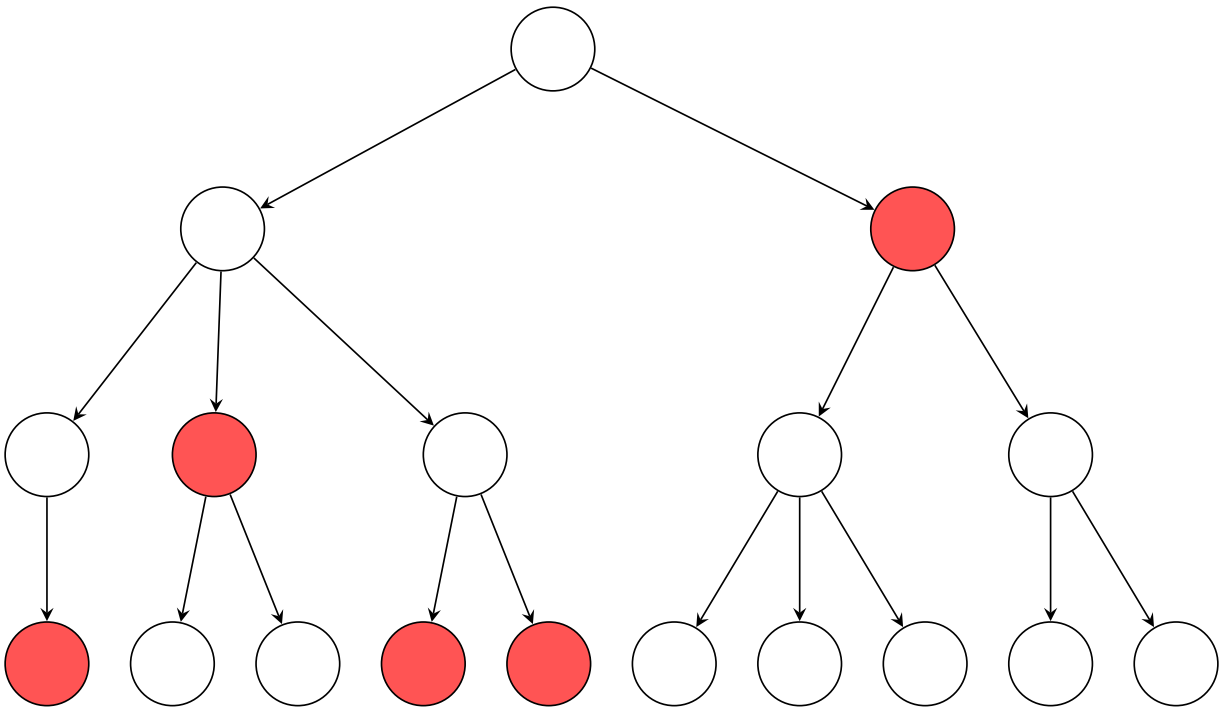
- The **non-nested** fragment of TCTL + (bounded) **response**;
- **Marking** properties are either:
  - linear constraints on the marking: $p_1 + 2 * p_2 > 4$
  - a boundedness property: `bounded(1)`
  - a deadlock property: `deadlock`
- **Temporal** properties (ϕ, ψ are marking properties):
  - $\mathtt{E}\,\phi\,\mathtt{U}\,[3,4]\,\psi$: there is a path on which ψ eventually holds in 3 to 4 t.u. and ϕ holds in the meantime;
  - $\mathtt{A}\,\phi\,\mathtt{U}\,[3,4]\,\psi$: on all paths ψ eventually holds in 3 to 4 t.u. and ϕ holds in the meantime;
  - $\phi --> [0,5]\psi$: whenever ϕ holds, on all subsequent paths ψ holds within 5 t.u.
- Classic **shorthands**:
  - $\mathtt{EF}\,[3,4]\,\psi = \mathtt{E}\,\mathtt{true}\,\mathtt{U}\,[3,4]\,\psi$: reachability;
  - $\mathtt{AF}\,[3,4]\,\psi = \mathtt{A}\,\mathtt{true}\,\mathtt{U}\,[3,4]\,\psi$: inevitability;
  - $\mathtt{EG}\,\psi = \neg\mathtt{AF}\,(\neg\psi)$: preservability;
  - $\mathtt{AG}\,\psi = \neg\mathtt{EF}\,(\neg\psi)$: safety.

Basic properties



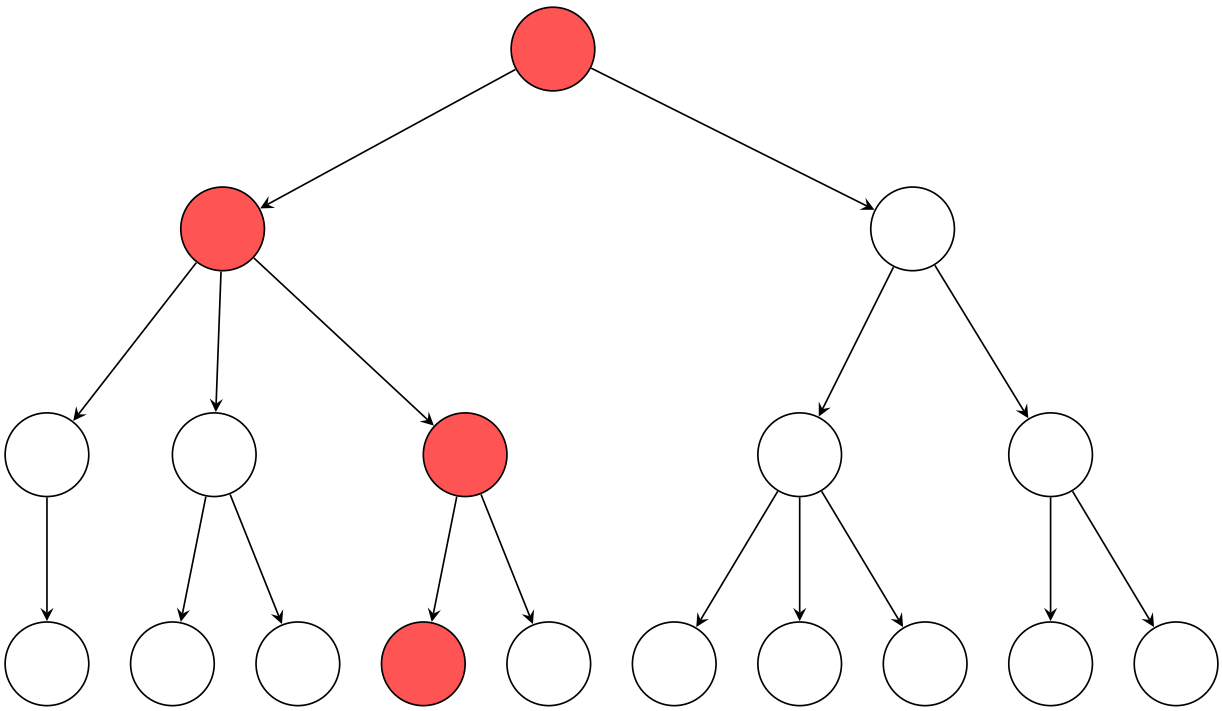
EFφ

Basic properties



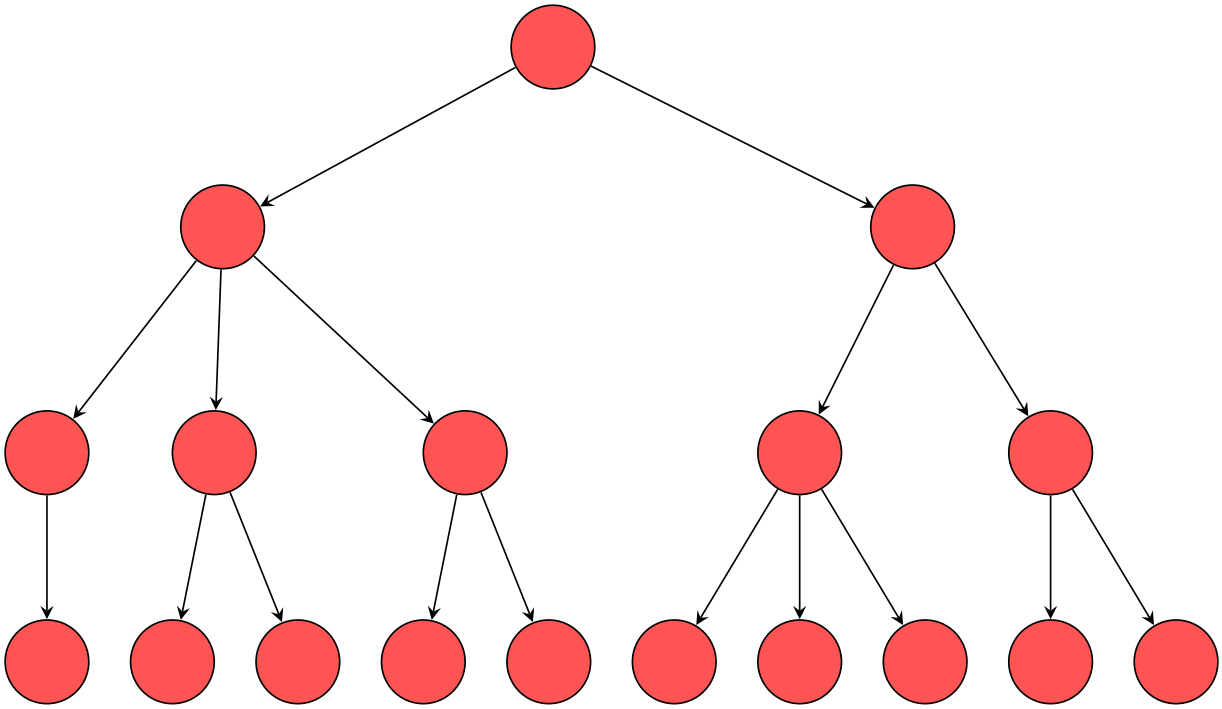
AFφ

Basic properties



EGφ

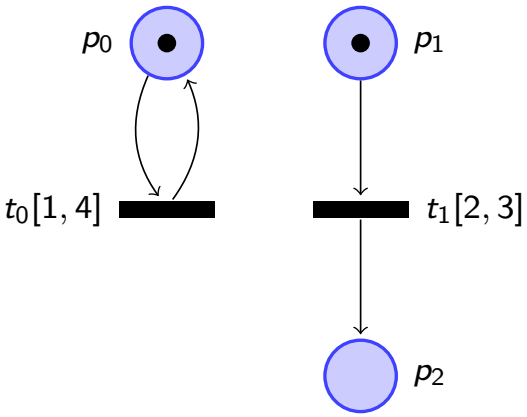
Basic properties



AGφ

State Classes [BD91]

- There is an uncountable number of states even in **bounded** TPNs;
- $\Rightarrow$ group all states obtained by the same sequence of transition firing;



Initially:

$$\begin{cases} 1 \leq t_0 \leq 4 \\ 2 \leq t_1 \leq 3 \end{cases}$$

Fire t_0:

$$\begin{cases} 1 \leq t_0 \leq 4 \\ 2 \leq t_1 \leq 3 \\ \mathbf{t_0 \leq t_1} \end{cases}$$

New times to fire:

$$\begin{cases} 1 \leq t_0 \leq 4 \\ 2 \leq \mathbf{t_1' + t_0} \leq 3 \\ t_0 \leq \mathbf{t_1' + t_0} \end{cases}$$

Disabled (incl. t_0):

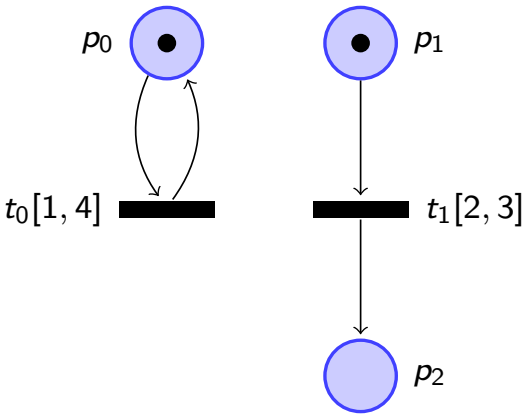
$$\{ \ \mathbf{0} \leq t_1' \leq \mathbf{2}$$

Newly enabled:

$$\begin{cases} \mathbf{1 \leq t_0 \leq 4} \\ 0 \leq t_1 \leq 2 \end{cases}$$

State Classes [BD91]

- There is an uncountable number of states even in **bounded** TPNs;
- $\Rightarrow$ group all states obtained by the same sequence of transition firing;



Initially:

$$\begin{cases} 1 \leq t_0 \leq 4 \\ 2 \leq t_1 \leq 3 \end{cases}$$

Fire t_0:

$$\begin{cases} 1 \leq t_0 \leq 4 \\ 2 \leq t_1 \leq 3 \\ \mathbf{t_0 \leq t_1} \end{cases}$$

New times to fire:

$$\begin{cases} 1 \leq t_0 \leq 4 \\ 2 \leq \mathbf{t_1' + t_0} \leq 3 \\ t_0 \leq \mathbf{t_1' + t_0} \end{cases}$$

Disabled (incl. t_0):

$$\{ \ \mathbf{0} \leq t_1' \leq \mathbf{2}$$

Newly enabled:

$$\begin{cases} \mathbf{1 \leq t_0 \leq 4} \\ 0 \leq t_1 \leq 2 \end{cases}$$

Class firing domains are zones (DBMs).

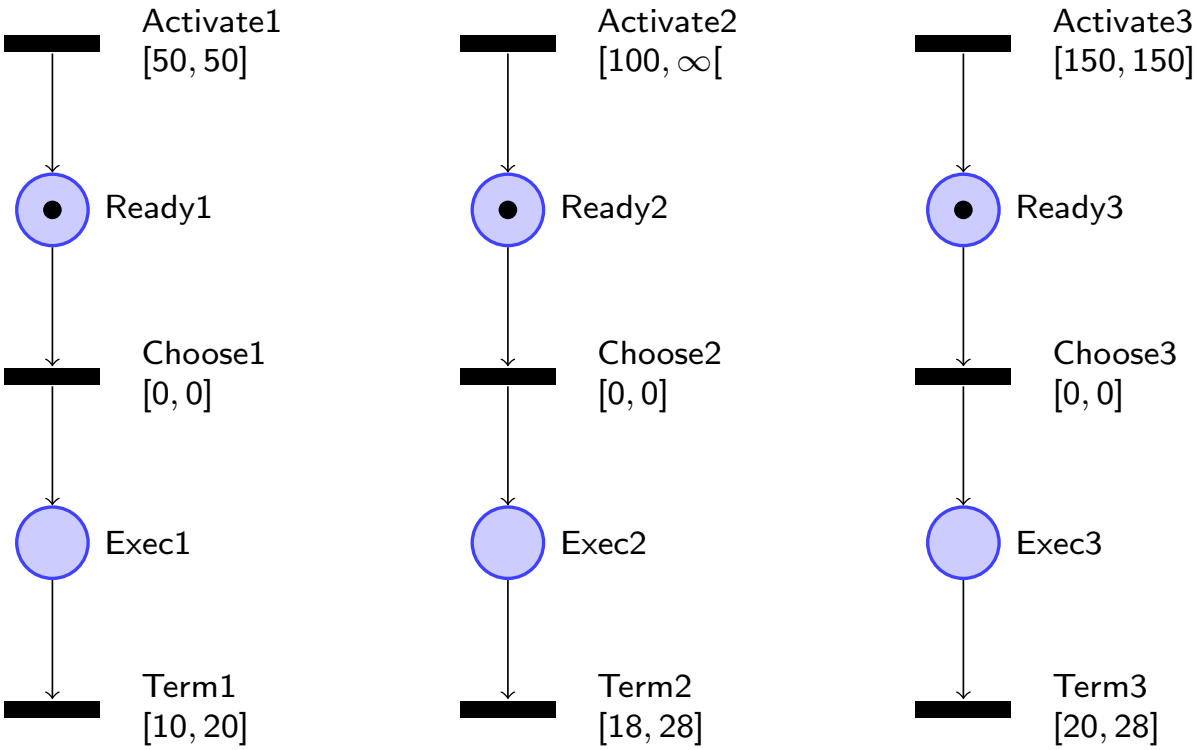
Basic Algorithms

$$\mathrm{EF}_G(S,M) = \begin{cases} \texttt{true} & \text{if } m \in G \\ \texttt{false} & \text{if } S \in M \\ \bigvee_{\substack{t \in T \\ S' = \mathrm{Next}(S,t)}} \mathrm{EF}_G\left(S', M \cup \{S\}\right) & \text{otherwise} \end{cases}$$

$$\mathrm{AF}_G(S,M) = \begin{cases} \texttt{true} & \text{if } S = (m,D) \text{ and } m \in G \\ \texttt{false} & \text{if } S \in M \\ \texttt{false} & \text{if } S \text{ has no successor} \\ \bigwedge_{\substack{t \in T \\ S' = \mathrm{Next}(S,t)}} \mathrm{AF}_G\left(S', M \cup \{S\}\right) & \text{otherwise} \end{cases}$$

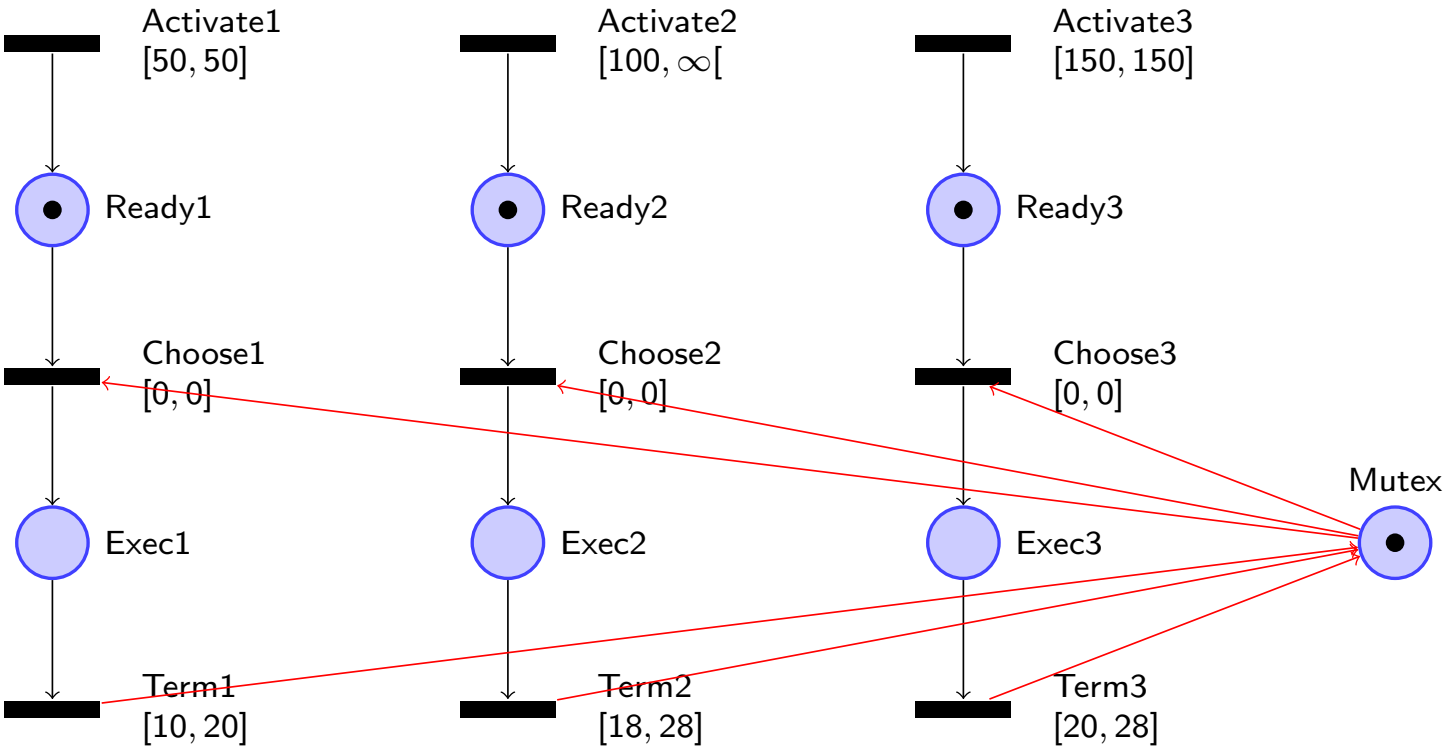
- G a set of markings to reach;
- M is a list of visited state classes;
- $\mathrm{Next}(S,t)$ computes the state class successor of S by transition t;
- If the net is **bounded** termination is guaranteed.

Example: A Scheduling Problem



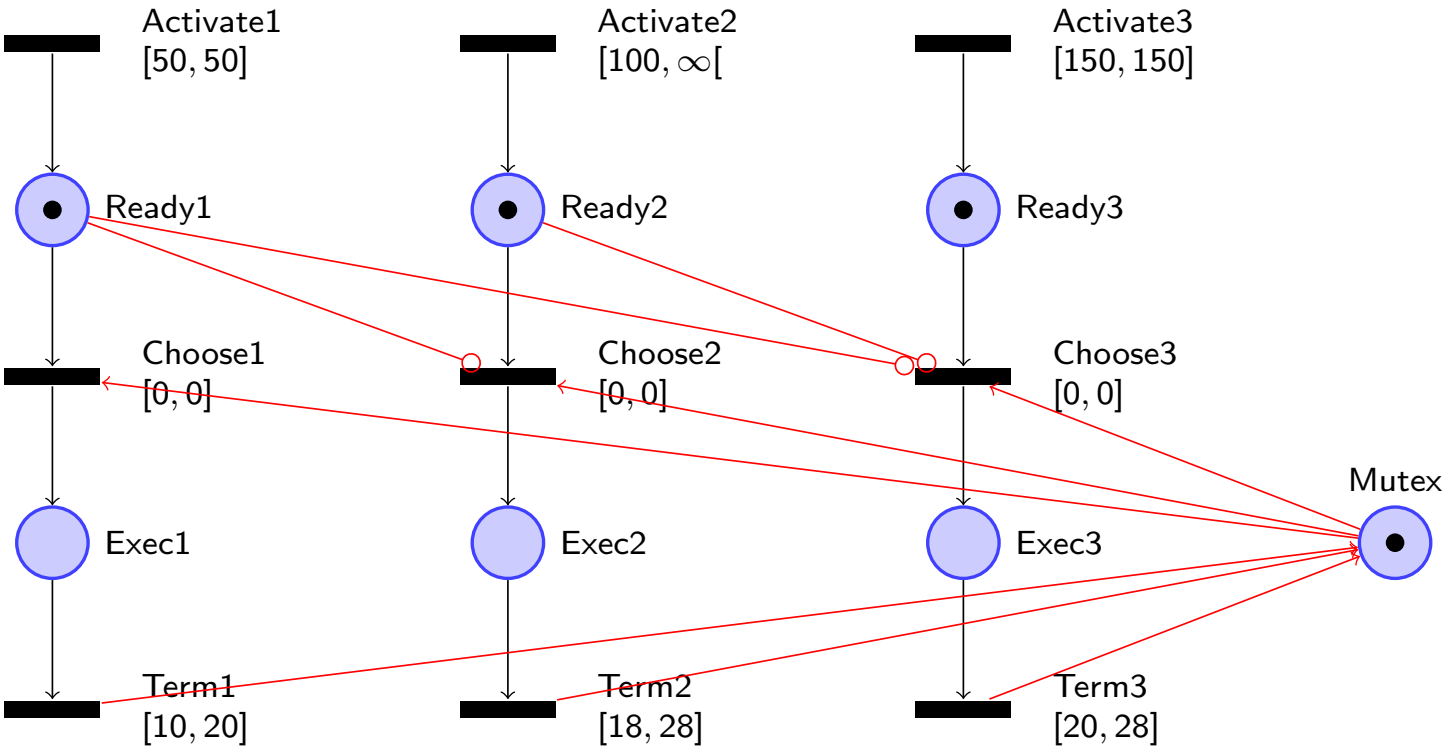
Schedulability: `AG bounded(1)`

Example: A Scheduling Problem



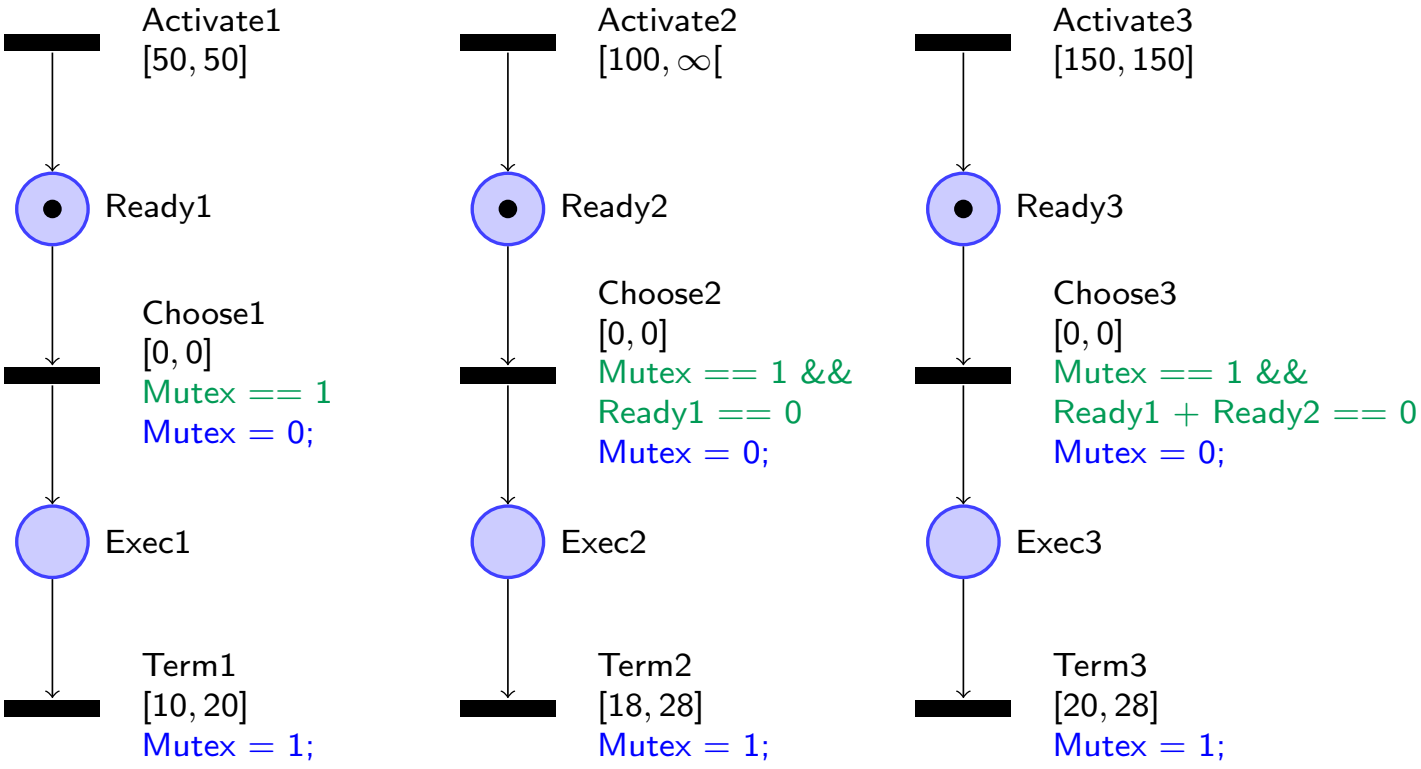
Schedulability: `AG bounded(1)`

Example: A Scheduling Problem



Schedulability: `AG bounded(1)`

Example: A Scheduling Problem – with Variables



`Mutex` is here an integer variable intialised to 1.

Outline

Introduction

Time Petri Nets

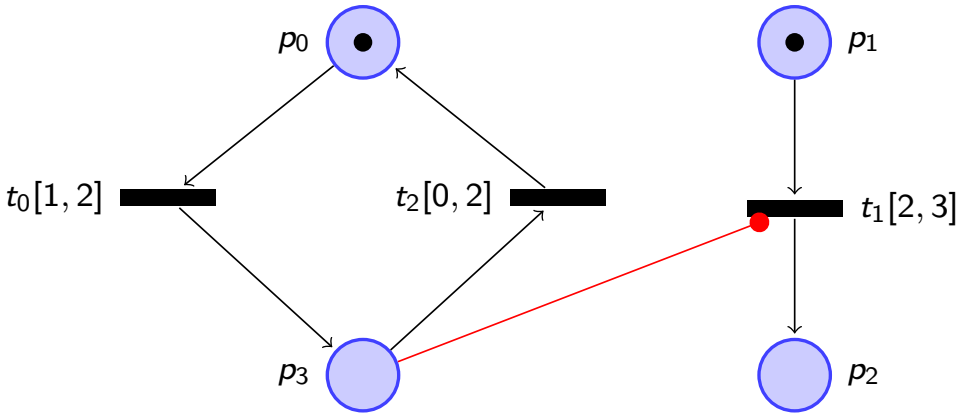
Stopwatch Petri Nets

Timing Parameters

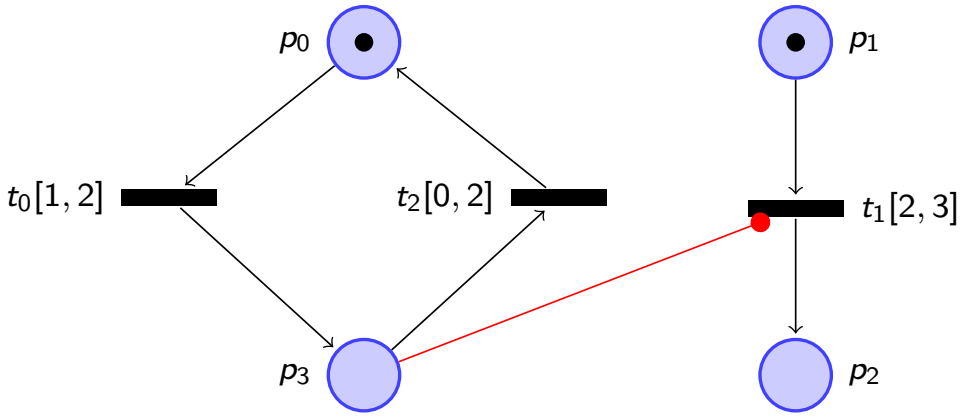
Minimal Cost Reachability

Conclusion

Stopwatch Petri Nets (Time Inhibitor Arcs)

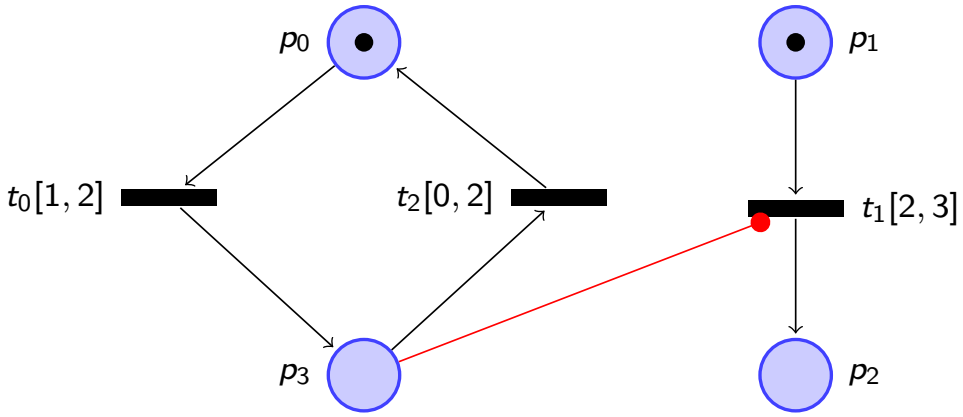


Stopwatch Petri Nets (Time Inhibitor Arcs)



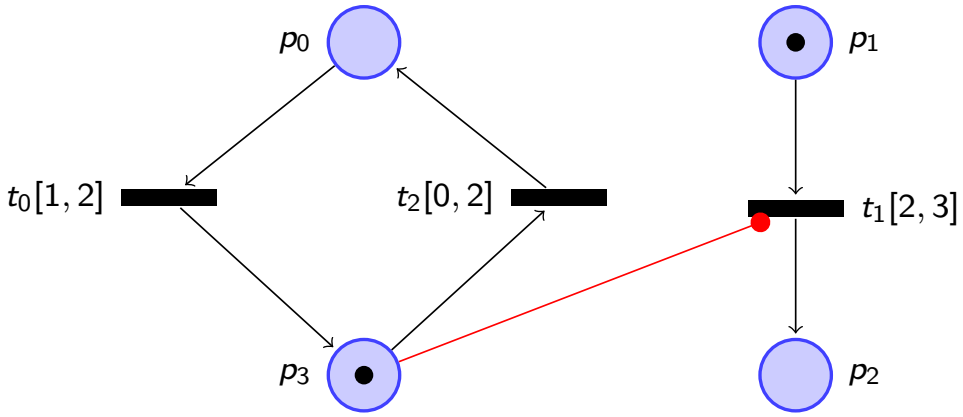
$t_0 \in [1,2]$
$t_1 \in [2,3]$

Stopwatch Petri Nets (Time Inhibitor Arcs)



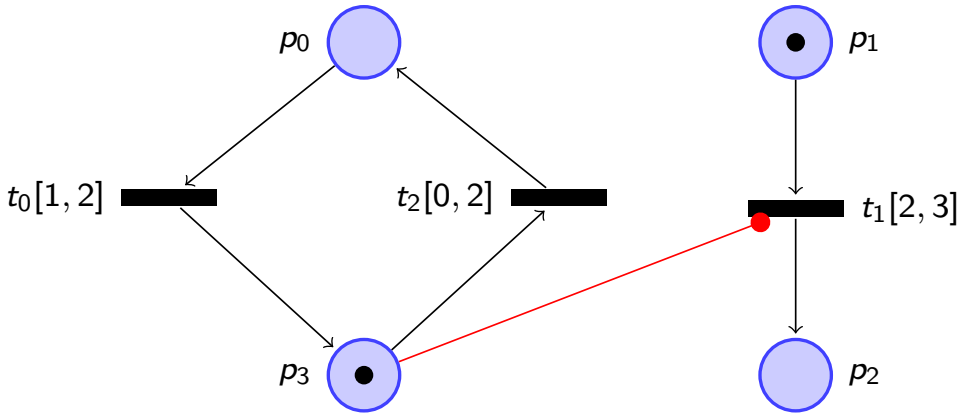
$$\begin{array}{l} t_0 \in [1,2] \\ t_1 \in [2,3] \end{array} \xrightarrow{1.1} \begin{array}{l} [0,0.9] \\ [0.9,\ 1.9] \end{array}$$

Stopwatch Petri Nets (Time Inhibitor Arcs)



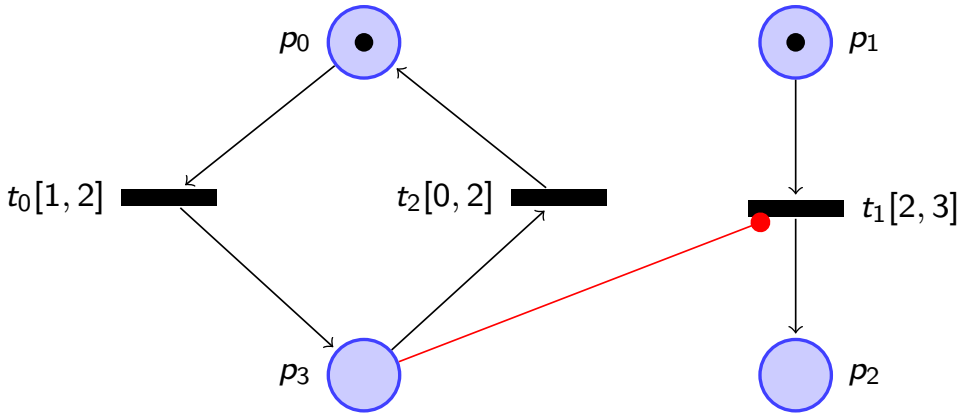
$$\begin{matrix} t_0 \in [1,2] \\ t_1 \in [2,3] \end{matrix} \xrightarrow{1.1} \begin{matrix} [0,0.9] \\ [0.9,\ 1.9] \end{matrix} \xrightarrow{t_0} \begin{matrix} t_2 \in [0,2] \\ t_1 \in [0.9,\ 1.9] \end{matrix}$$

Stopwatch Petri Nets (Time Inhibitor Arcs)



$$\begin{matrix} t_0 \in [1,2] \\ t_1 \in [2,3] \end{matrix} \xrightarrow{1.1} \begin{matrix} [0,0.9] \\ [0.9,\ 1.9] \end{matrix} \xrightarrow{t_0} \begin{matrix} t_2 \in [0,2] \\ t_1 \in [0.9,\ 1.9] \end{matrix} \xrightarrow{1.9} \begin{matrix} [0,0.1] \\ \mathbf{[0.9,1.9]} \end{matrix}$$

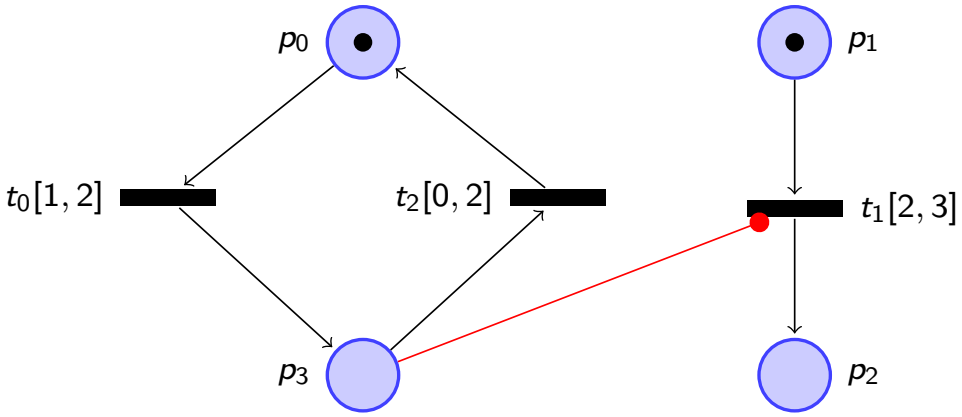
Stopwatch Petri Nets (Time Inhibitor Arcs)



$$\begin{array}{l} t_0 \in [1,2] \\ t_1 \in [2,3] \end{array} \xrightarrow{1.1} \begin{array}{l} [0,0.9] \\ [0.9,\ 1.9] \end{array} \xrightarrow{t_0} \begin{array}{l} t_2 \in [0,2] \\ t_1 \in [0.9,\ 1.9] \end{array} \xrightarrow{1.9} \begin{array}{l} [0,0.1] \\ \mathbf{[0.9,1.9]} \end{array}$$

$$\xrightarrow{t_2} \begin{array}{l} t_0 \in [1,2] \\ t_1 \in [0.9,\ 1.9] \end{array}$$

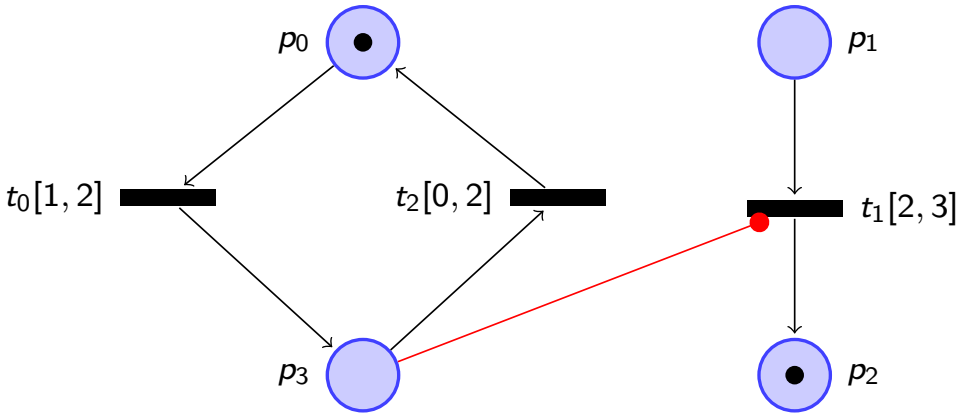
Stopwatch Petri Nets (Time Inhibitor Arcs)



$$\begin{array}{l} t_0 \in [1,2] \\ t_1 \in [2,3] \end{array} \xrightarrow{1.1} \begin{array}{l} [0,0.9] \\ [0.9,\ 1.9] \end{array} \xrightarrow{t_0} \begin{array}{l} t_2 \in [0,2] \\ t_1 \in [0.9,\ 1.9] \end{array} \xrightarrow{1.9} \begin{array}{l} [0,0.1] \\ \mathbf{[0.9,1.9]} \end{array}$$

$$\xrightarrow{t_2} \begin{array}{l} t_0 \in [1,2] \\ t_1 \in [0.9,\ 1.9] \end{array} \xrightarrow{1.9} \begin{array}{l} [0,0.1] \\ [0,\ 0] \end{array}$$

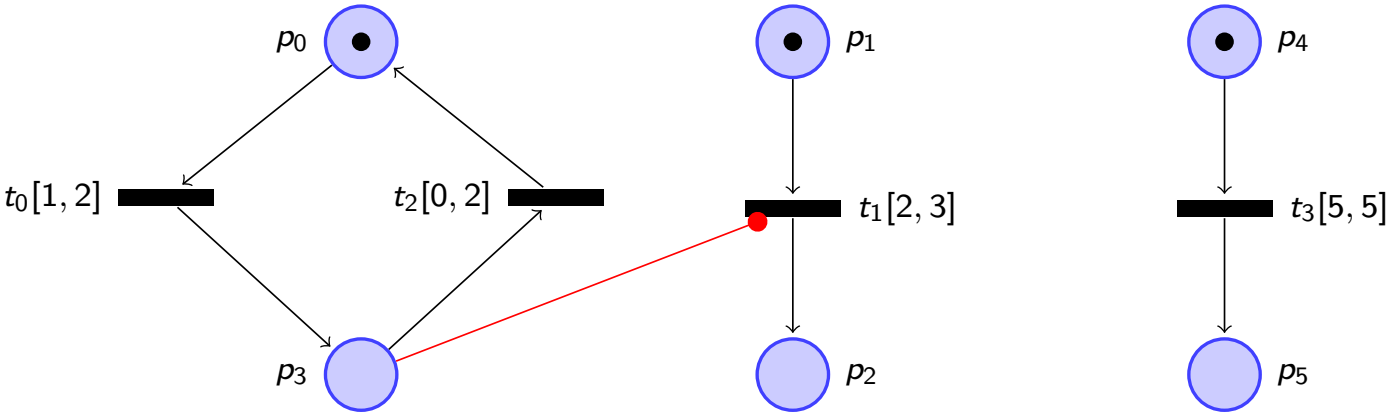
Stopwatch Petri Nets (Time Inhibitor Arcs)



$$\begin{array}{l} t_0 \in [1,2] \\ t_1 \in [2,3] \end{array} \xrightarrow{1.1} \begin{array}{l} [0,0.9] \\ [0.9,\ 1.9] \end{array} \xrightarrow{t_0} \begin{array}{l} t_2 \in [0,2] \\ t_1 \in [0.9,\ 1.9] \end{array} \xrightarrow{1.9} \begin{array}{l} [0,0.1] \\ \mathbf{[0.9,1.9]} \end{array}$$

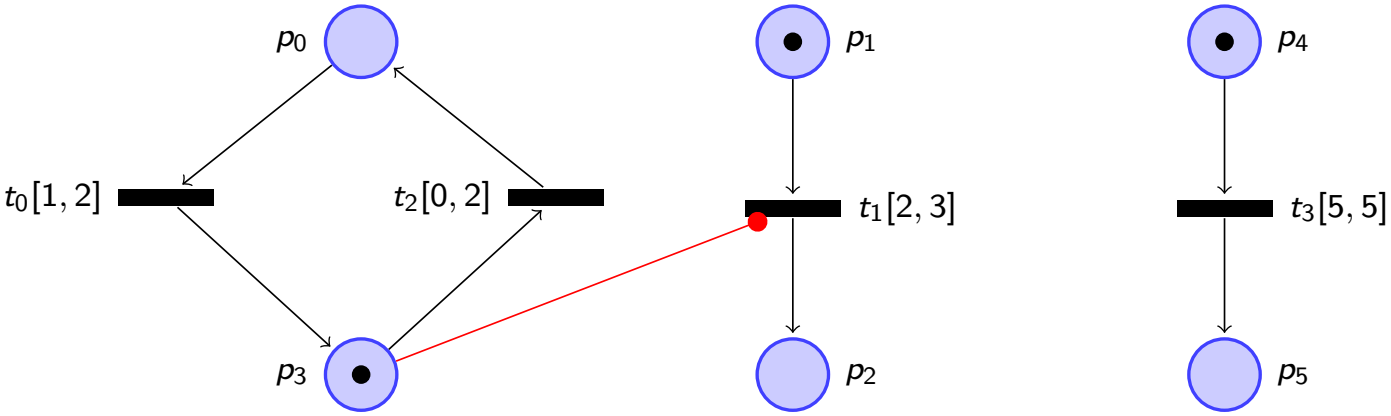
$$\xrightarrow{t_2} \begin{array}{l} t_0 \in [1,2] \\ t_1 \in [0.9,\ 1.9] \end{array} \xrightarrow{1.9} \begin{array}{l} [0,0.1] \\ [0,\ 0] \end{array} \xrightarrow{t_1} [0,0.1]$$

Stopwatch State Classes



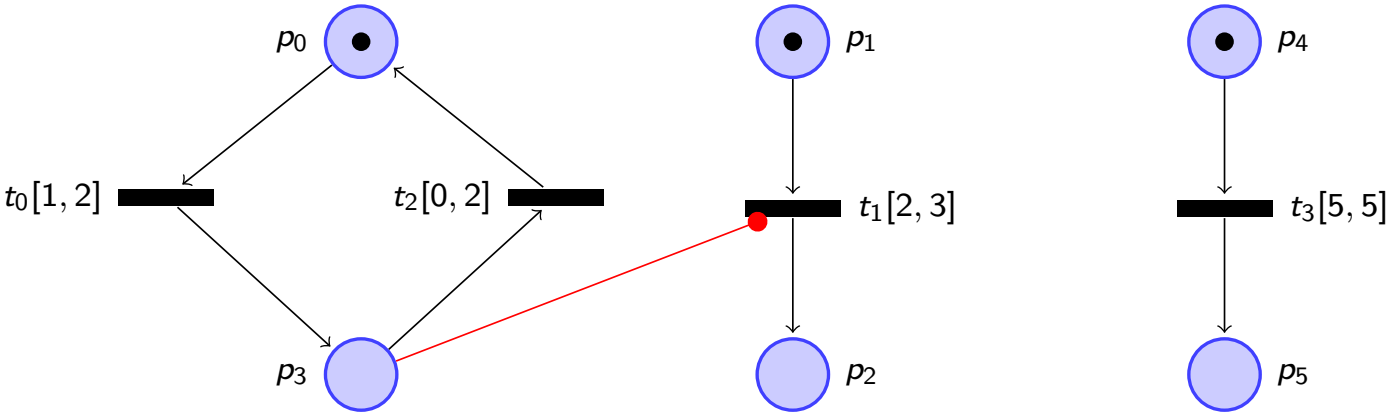
$$1 \leq t_0 \leq 2$$
$$2 \leq t_1 \leq 3$$
$$5 \leq t_3 \leq 5$$

Stopwatch State Classes



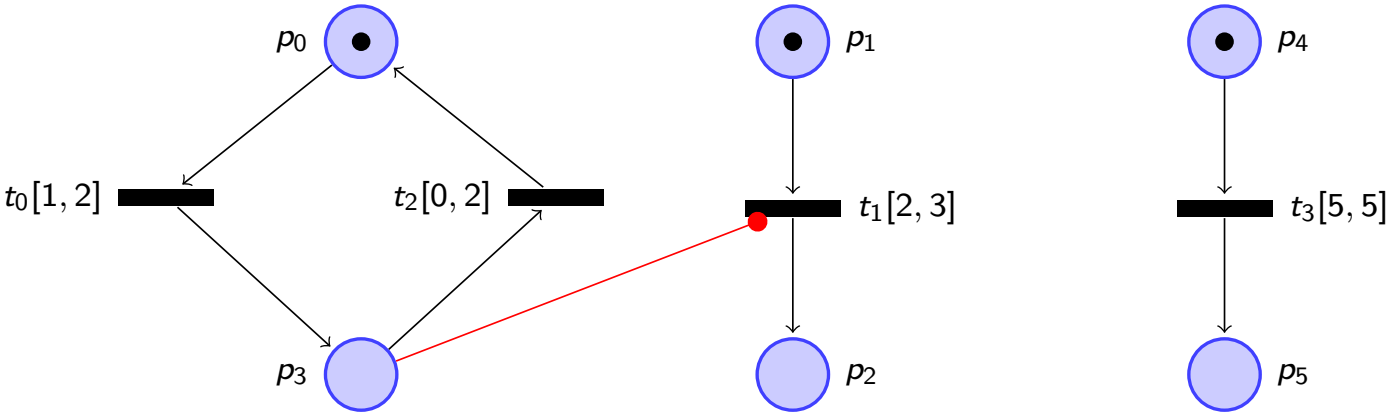
$$\begin{array}{l} 1 \leq t_0 \leq 2 \\ 2 \leq t_1 \leq 3 \\ 5 \leq t_3 \leq 5 \end{array} \xrightarrow{t_0} \begin{array}{l} 0 \leq t_2 \leq 2 \\ 0 \leq t_1 \leq 2 \\ 3 \leq t_3 \leq 4 \\ -3 \leq t_3 - t_4 \leq -2 \end{array}$$

Stopwatch State Classes



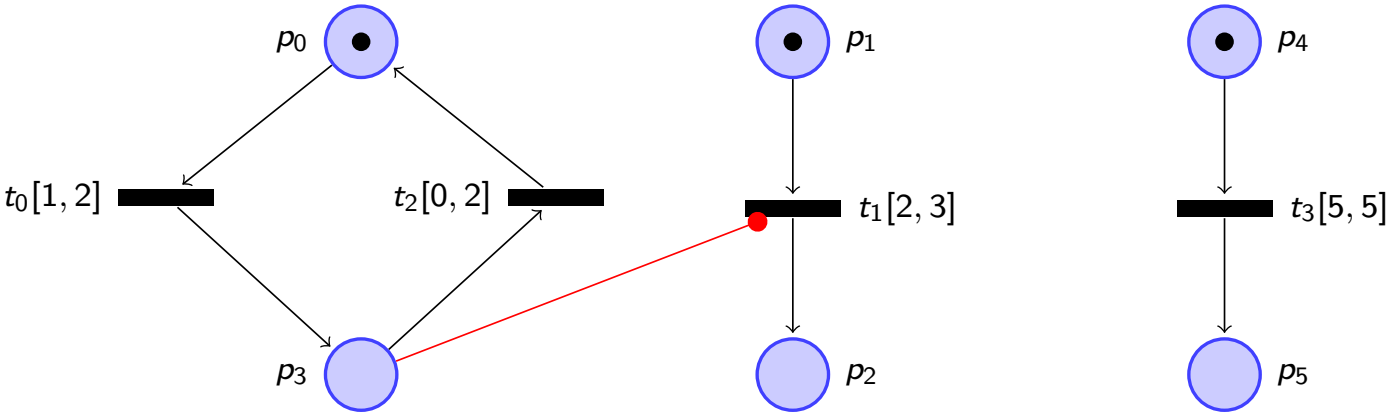
$$\begin{array}{l} 1 \le t_0 \le 2 \\ 2 \le t_1 \le 3 \\ 5 \le t_3 \le 5 \end{array} \xrightarrow{t_0} \begin{array}{l} 0 \le t_2 \le 2 \\ 0 \le t_1 \le 2 \\ 3 \le t_3 \le 4 \\ -3 \le t_3 - t_4 \le -2 \end{array} \xrightarrow{t_2} \begin{array}{l} 0 \le t_2 \le 2 \\ 0 \le t_1 \le 2 \\ 3 \le t_3 \le 4 \\ -3 \le t_1 - t_3 \le -2 \\ \mathbf{t_2 \le t_3} \end{array}$$

Stopwatch State Classes



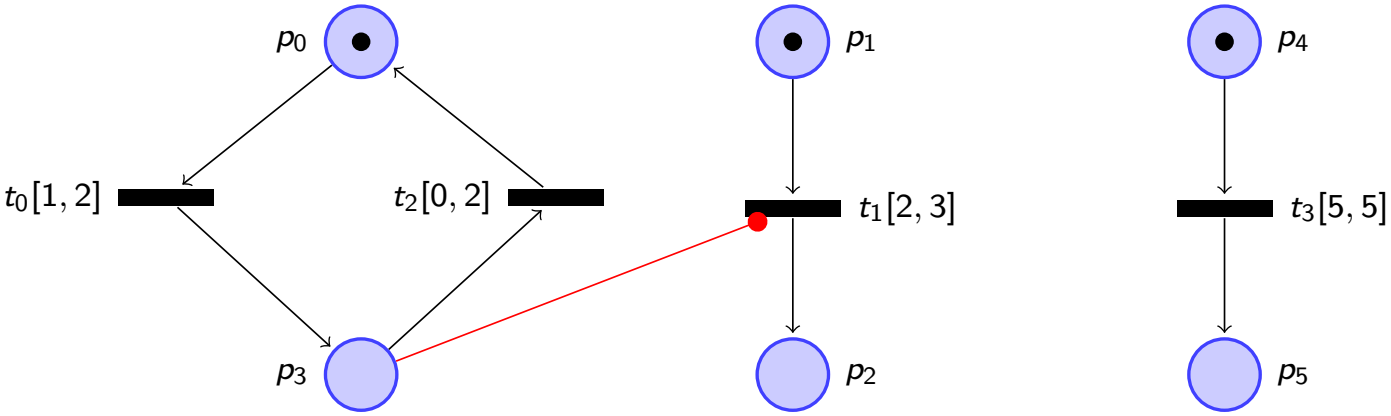
$$\begin{array}{l} 1 \le t_0 \le 2 \\ 2 \le t_1 \le 3 \\ 5 \le t_3 \le 5 \end{array} \xrightarrow{t_0} \begin{array}{l} 0 \le t_2 \le 2 \\ 0 \le t_1 \le 2 \\ 3 \le t_3 \le 4 \\ -3 \le t_3 - t_4 \le -2 \end{array} \xrightarrow{t_2} \begin{array}{l} 0 \le t_2 \le 2 \\ 0 \le \mathbf{t_1} \le 2 \\ 3 \le t_3' + \mathbf{t_2} \le 4 \\ -3 \le t_1 - t_3 \le -2 \\ t_2 \le t_3' + \mathbf{t_2} \end{array}$$

Stopwatch State Classes



$$\begin{array}{l} 1 \leq t_0 \leq 2 \\ 2 \leq t_1 \leq 3 \\ 5 \leq t_3 \leq 5 \end{array} \xrightarrow{t_0} \begin{array}{l} 0 \leq t_2 \leq 2 \\ 0 \leq t_1 \leq 2 \\ 3 \leq t_3 \leq 4 \\ -3 \leq t_3 - t_4 \leq -2 \end{array} \xrightarrow{t_2} \begin{array}{l} 1 \leq t_0 \leq 2 \\ 0 \leq t_1 \leq 2 \\ 1 \leq t_3 \leq 4 \\ -3 \leq t_1 - t_3 \leq 0 \end{array}$$

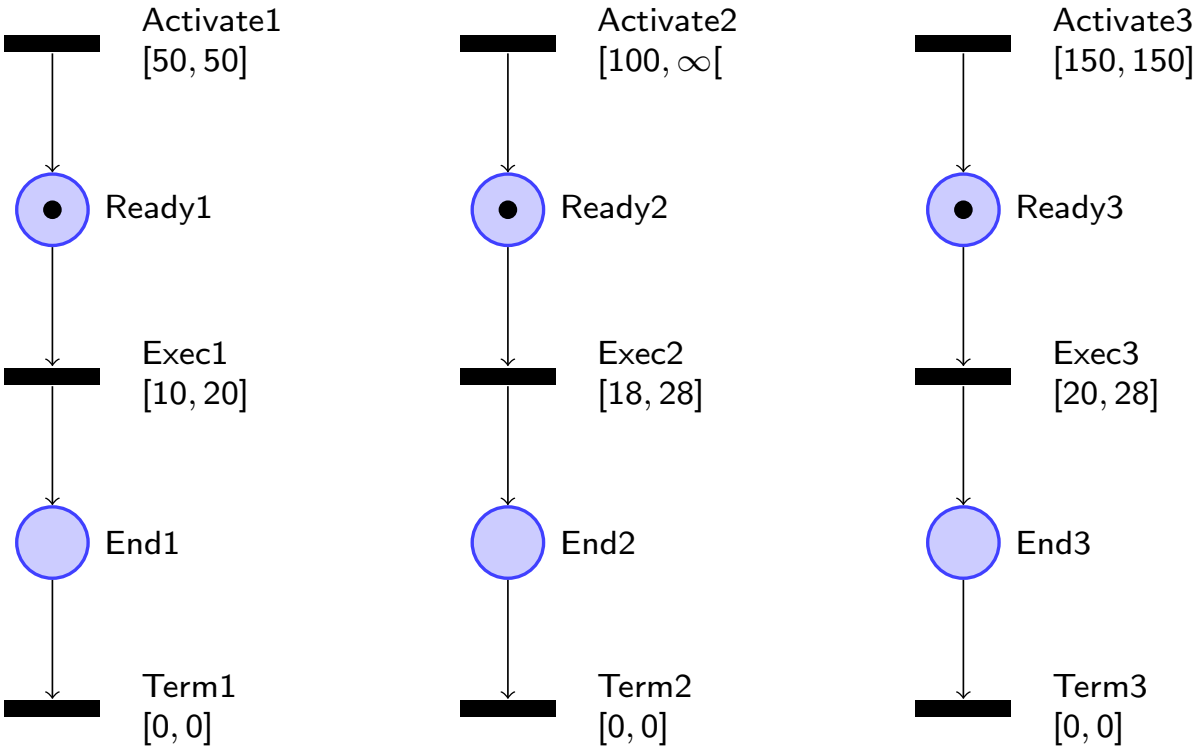
Stopwatch State Classes



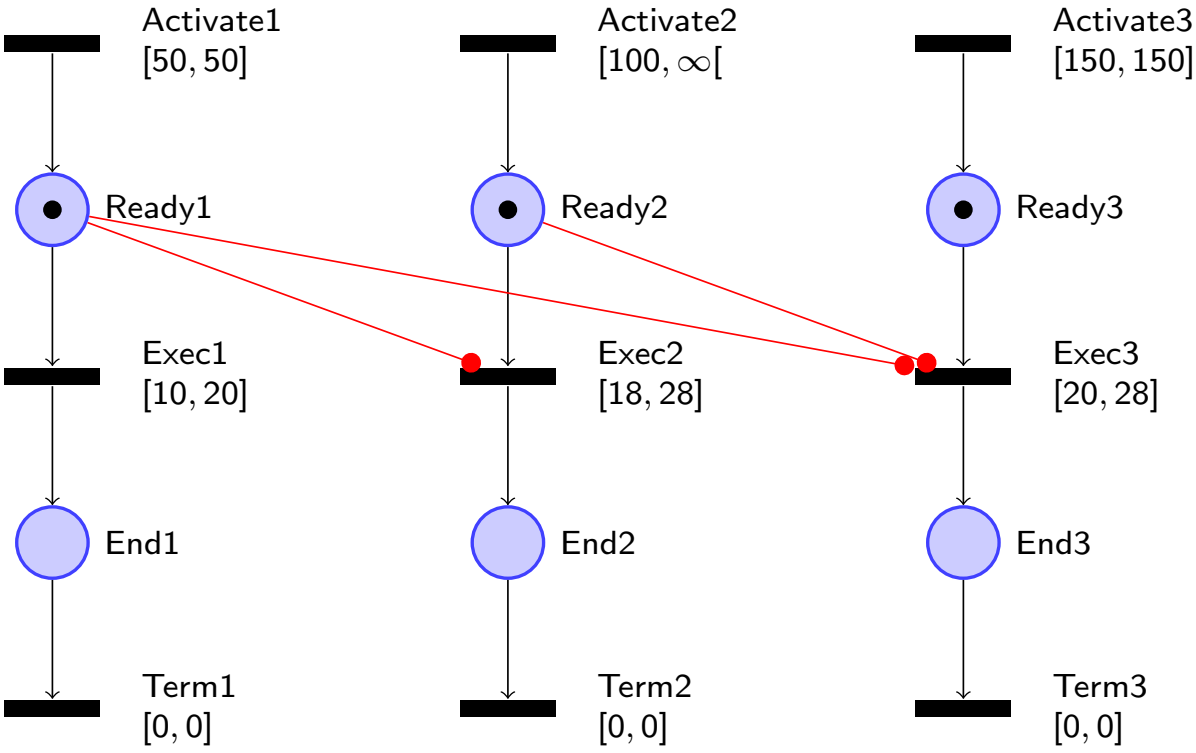
$$\begin{array}{l} 1 \le t_0 \le 2 \\ 2 \le t_1 \le 3 \\ 5 \le t_3 \le 5 \end{array} \xrightarrow{t_0} \begin{array}{l} 0 \le t_2 \le 2 \\ 0 \le t_1 \le 2 \\ 3 \le t_3 \le 4 \\ -3 \le t_3 - t_4 \le -2 \end{array} \xrightarrow{t_2} \begin{array}{l} 1 \le t_0 \le 2 \\ 0 \le t_1 \le 2 \\ 1 \le t_3 \le 4 \\ -3 \le t_1 - t_3 \le 0 \end{array}$$

Class firing domains can be arbitrary polyhedra.

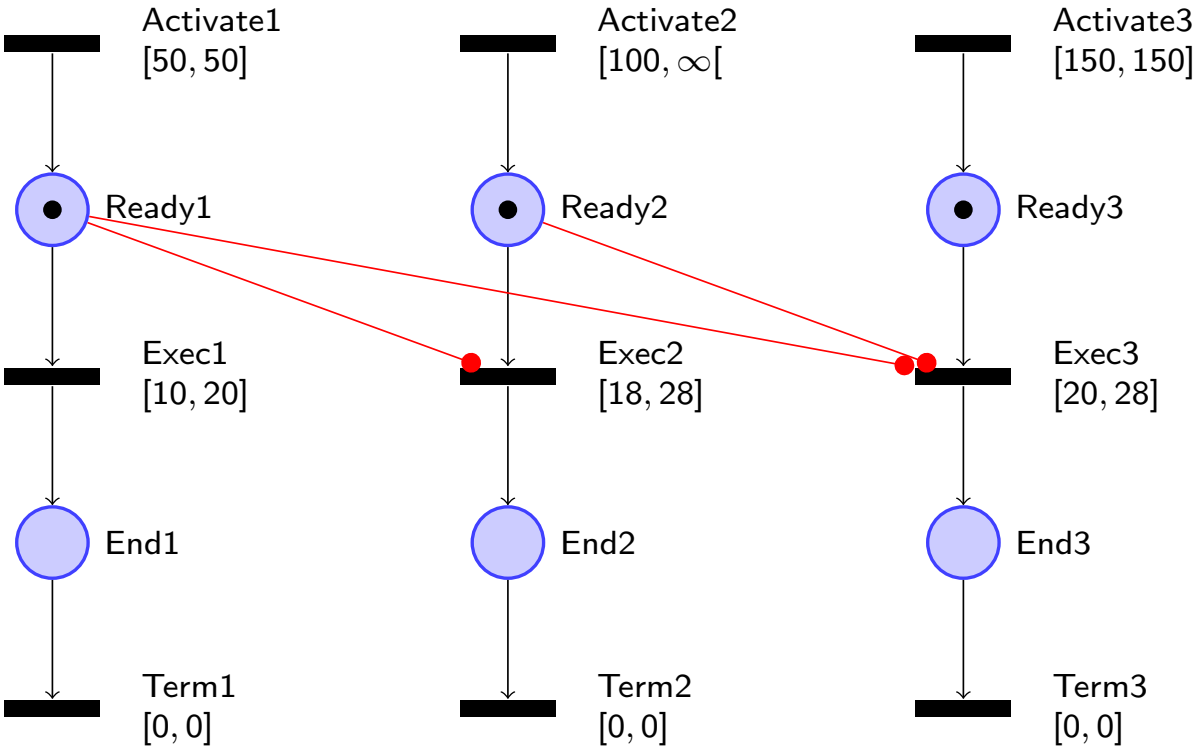
Example: A Preemptive Scheduling Problem



Example: A Preemptive Scheduling Problem



Example: A Preemptive Scheduling Problem



WCRT of τ_3 less than 96: `(Ready3 > 0) -> [0,96] (End3 > 0)`

Outline

Introduction

Time Petri Nets

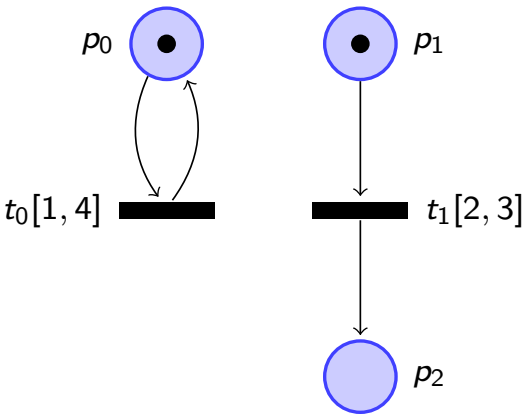
Stopwatch Petri Nets

Timing Parameters

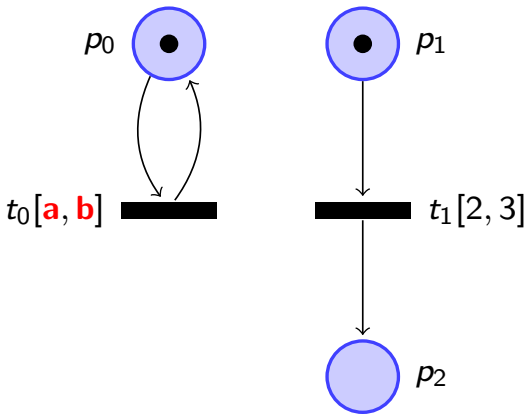
Minimal Cost Reachability

Conclusion

Parametric Time Petri Nets



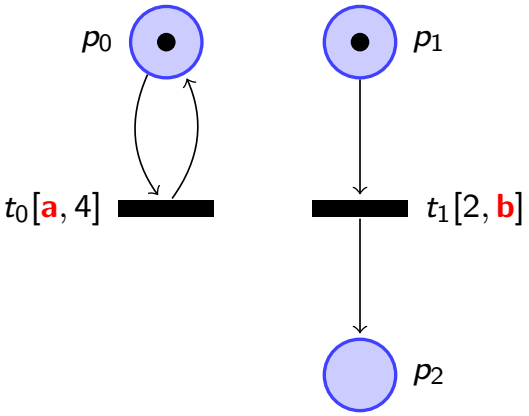
Parametric Time Petri Nets



Parametric Properties

- Given a TCTL property φ, what are the **values of the parameters** such that φ holds?
- Every (non-trivial) related decision problem is **undecidable**.

Parametric State Classes [TLR09]



Initially:

$$\begin{cases} \mathbf{a} \leq t_0 \leq 4 \\ 2 \leq t_1 \leq \mathbf{b} \end{cases}$$

Fire t_0:

$$\begin{cases} \mathbf{a} \leq t_0 \leq 4 \\ 2 \leq t_1 \leq \mathbf{b} \\ \mathbf{t_0} \leq \mathbf{t_1} \\ (\mathbf{a} \leq \mathbf{b}) \end{cases}$$

New times to fire:

$$\begin{cases} \mathbf{a} \leq t_0 \leq 4 \\ 2 \leq \mathbf{t_1'} + \mathbf{t_0} \leq \mathbf{b} \\ t_0 \leq \mathbf{t_1'} + \mathbf{t_0} \end{cases}$$

Disabled (incl. t_0):

$$\{ \ \mathbf{0} \leq t_1' \leq \mathbf{b} - \mathbf{a}$$

Newly enabled:

$$\begin{cases} \mathbf{a} \leq \mathbf{t_0} \leq \mathbf{4} \\ 0 \leq t_1 \leq \mathbf{b} - \mathbf{a} \end{cases}$$

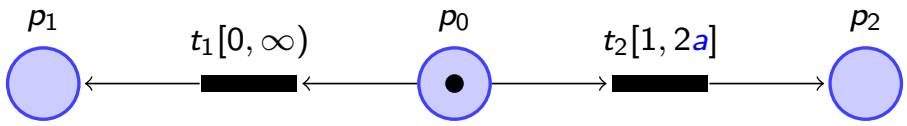
Class firing domains are parametric zones.

Parametric Algorithms

$$\mathrm{EF}_G(S,M) = \begin{cases} \mathbf{D}_{\mathbb{P}} & \text{if } m \in G \\ \emptyset & \text{if } S \in M \\ \bigcup_{\substack{t \in T \\ S'=\mathrm{Next}(S,t)}} \mathrm{EF}_G\left(S', M \cup \{S\}\right) & \text{otherwise.} \end{cases}$$

$$\mathrm{AF}_G(S,M) = \begin{cases} D_{\mathbb{P}} & \text{if } m \in G \\ \emptyset & \text{if } S \in M \\ \emptyset & \text{if } S \text{ has no succ.} \\ \left(\bigcap_{\substack{t \in T \\ S'=\mathrm{Next}(S,t)}} \left(\mathrm{AF}_G\left(S', M \cup \{S\}\right) \cup (\mathbb{Q}^{\mathbf{P}} \setminus \mathbf{S'}_{\mathbb{P}})\right) \right) & \text{otherwise} \end{cases}$$

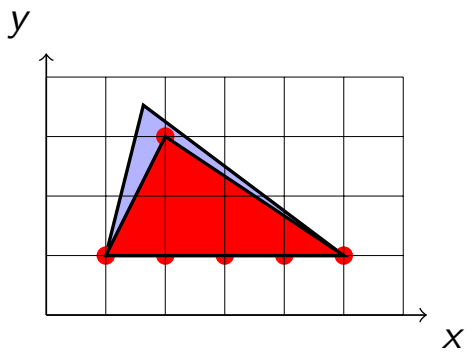
AF: Cutting for More



- Put a token in p_1: no constraint
- Put a token in p_2: $a \geq \frac{1}{2}$
- Ensuring both paths are possible (for AF ($p_1 > 0$ or $p_2 > 0$)): $a \geq \frac{1}{2}$
- Or we can **cut** t_2 **and** p_2 **off** with $a < \frac{1}{2}$ and the property is satisfied with no further constraint
- Finally, AF ($p_1 > 0$ or $p_2 > 0$) is satisfied for all values of a.

Symbolic Synthesis for Bounded Integers

- EF-emptiness is **undecidable** for **integer** parameters [AHV93];
- It is **undecidable** for **bounded rational** parameters [Mil00];
- It is **PSPACE-complete** for **bounded integer** parameters [JLR15].
  - **non-deterministically guess** a parameter valuation and store it (polynomial storage size);
  - instantiate the PTA or PTPN and solve the problem (PSPACE);
  - PSPACE = NPSPACE (Savitch's theorem).
- Synthesis can be done **symbolically**, using **integer hulls**:



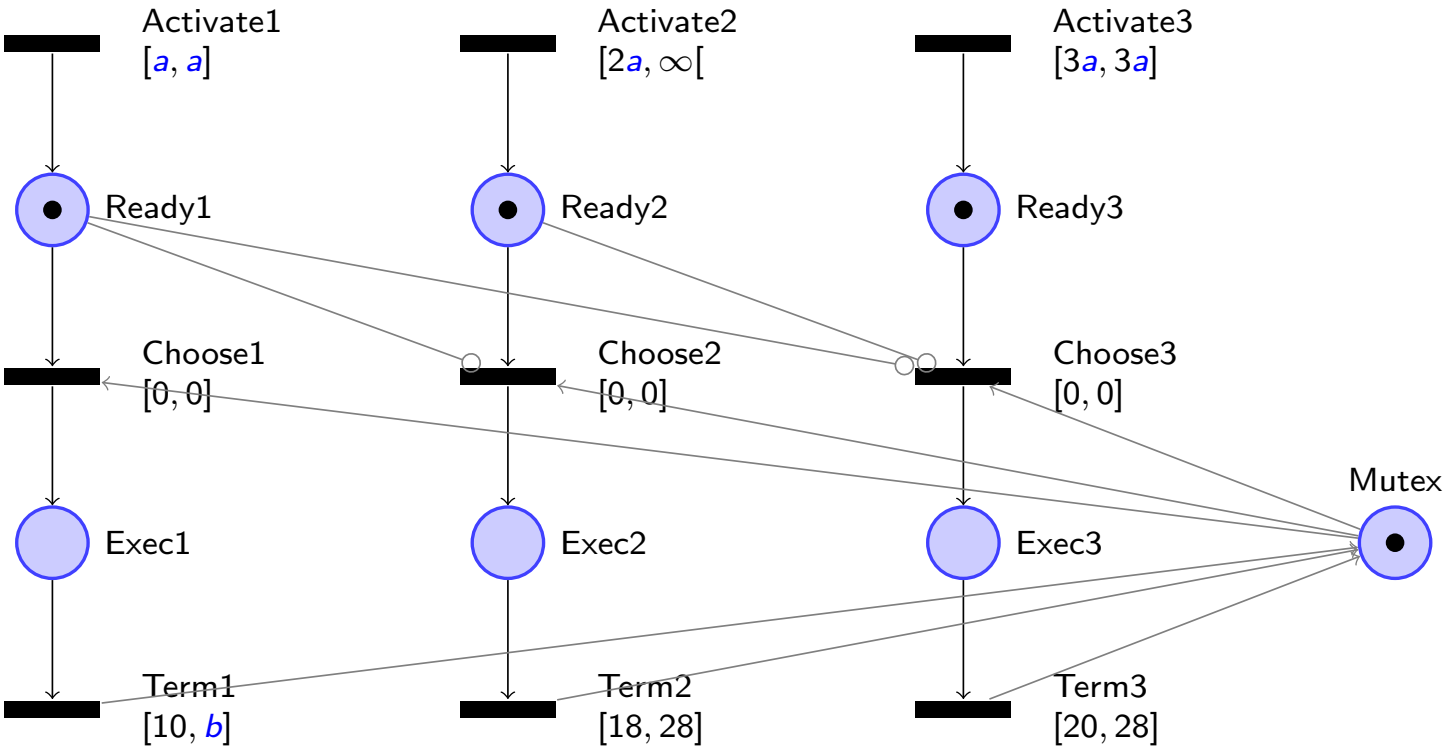
Symbolic Synthesis for Bounded Integer Parameters

- IEF computes polyhedra containing **exactly** the "good" **integer** parameter valuations:

$$\mathrm{IEF}_G(S,M) = \begin{cases} D_{\mathbb{P}} & \text{if } m \in G \\ \emptyset & \text{if } S \in M \\ \bigcup_{\substack{t \in T \\ S' = \mathrm{IH}(\mathrm{Next}(S,t))}} \mathrm{IEF}_G\big(S', M \cup \{S\}\big) & \text{otherwise.} \end{cases}$$

- It is **guaranteed to terminate** when the parameters are **bounded**;
- AF can be modified similarly;
- **Warning**: the result of IAF is not **dense** [ALR15]
  But we can modify IAF to make it so

Example: A Parametric Scheduling Problem



Schedulability: `AG bounded(1)`

Outline

Introduction

Time Petri Nets

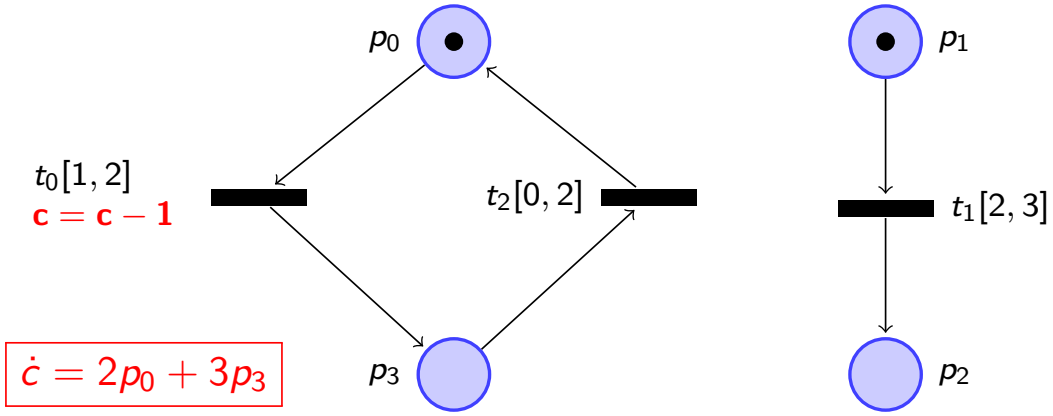
Stopwatch Petri Nets

Timing Parameters

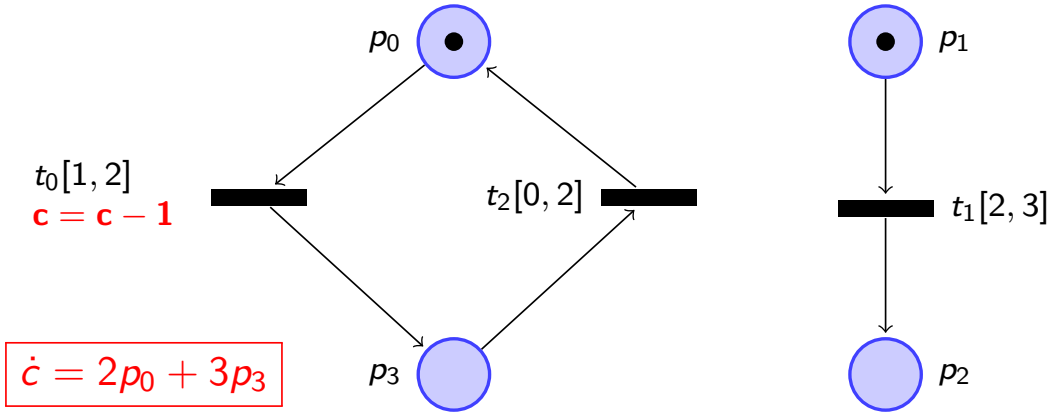
Minimal Cost Reachability

Conclusion

Cost Time Petri Nets

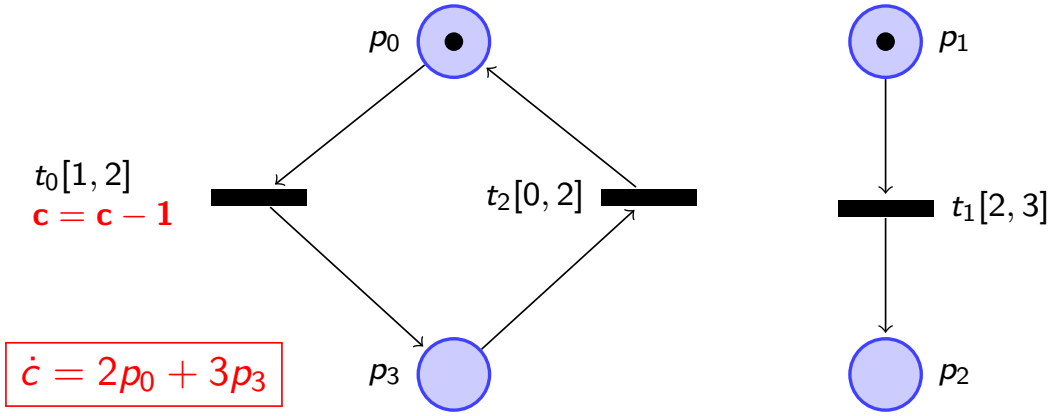


Cost Time Petri Nets



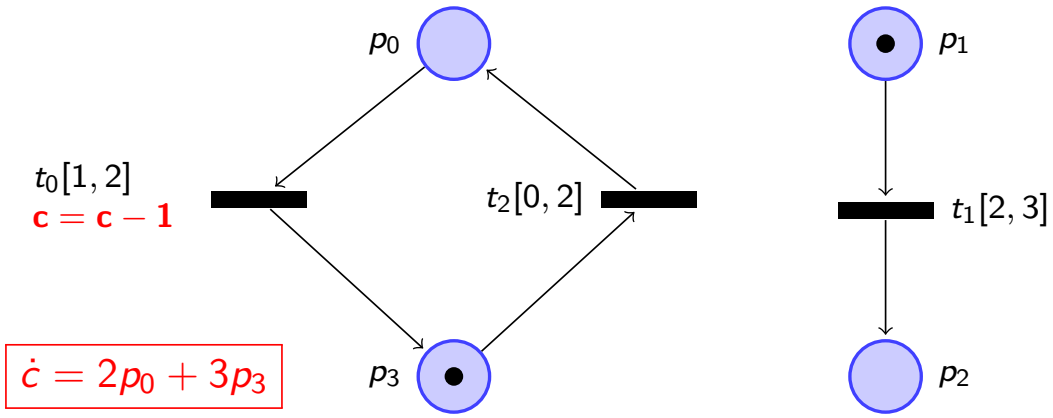
$t_0 \in [1,2]$
$t_1 \in [2,3]$
$\mathbf{c} = \mathbf{0}$

Cost Time Petri Nets



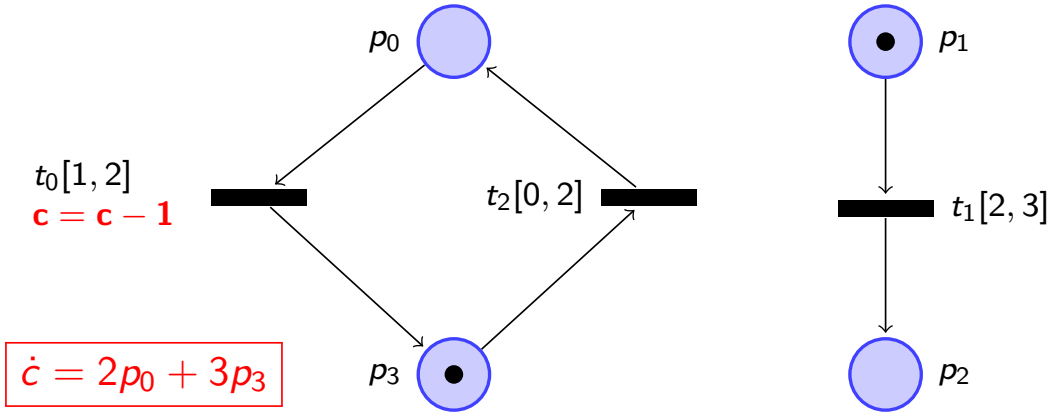
$$\begin{array}{l} t_0 \in [1,2] \\ t_1 \in [2,3] \\ \mathbf{c = 0} \end{array} \xrightarrow{1.1} \begin{array}{l} [0, 0.9] \\ [0.9,\ 1.9] \\ \mathbf{c = 2.2} \end{array}$$

Cost Time Petri Nets



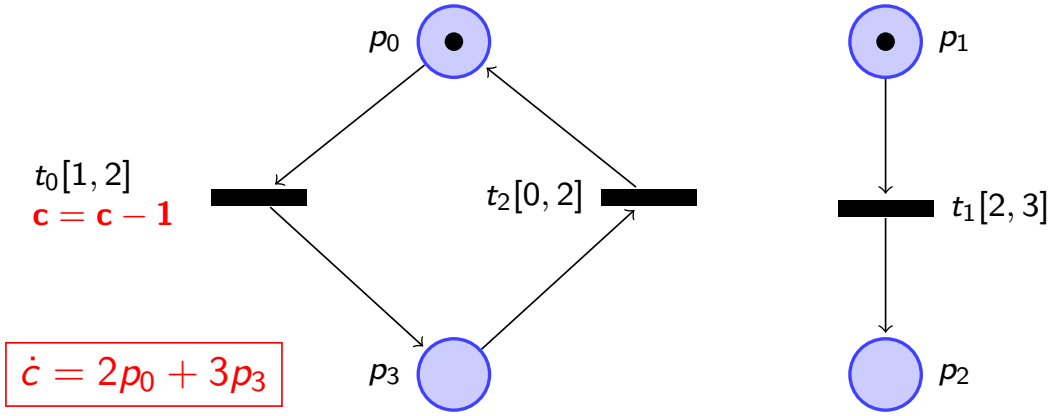
$$\begin{array}{l} t_0 \in [1,2] \\ t_1 \in [2,3] \\ \mathbf{c = 0} \end{array} \xrightarrow{1.1} \begin{array}{l} [0, 0.9] \\ [0.9,\ 1.9] \\ \mathbf{c = 2.2} \end{array} \xrightarrow{t_0} \begin{array}{l} t_2 \in [0,2] \\ t_1 \in [0.9,\ 1.9] \\ \mathbf{c = 1.2} \end{array}$$

Cost Time Petri Nets



$$\begin{array}{l} t_0 \in [1,2] \\ t_1 \in [2,3] \\ \mathbf{c = 0} \end{array} \xrightarrow{1.1} \begin{array}{l} [0,0.9] \\ [0.9,\ 1.9] \\ \mathbf{c = 2.2} \end{array} \xrightarrow{t_0} \begin{array}{l} t_2 \in [0,2] \\ t_1 \in [0.9,\ 1.9] \\ \mathbf{c = 1.2} \end{array} \xrightarrow{1.9} \begin{array}{l} [0,0.1] \\ [0.9,1.9] \\ \mathbf{c = 6.9} \end{array}$$

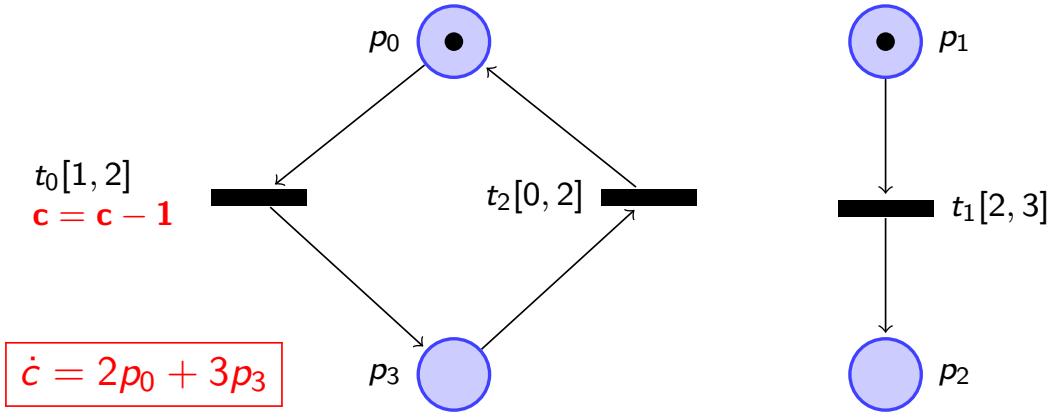
Cost Time Petri Nets



$$\begin{array}{l} t_0 \in [1,2] \\ t_1 \in [2,3] \\ \mathbf{c = 0} \end{array} \xrightarrow{1.1} \begin{array}{l} [0,0.9] \\ [0.9,\ 1.9] \\ \mathbf{c = 2.2} \end{array} \xrightarrow{t_0} \begin{array}{l} t_2 \in [0,2] \\ t_1 \in [0.9,\ 1.9] \\ \mathbf{c = 1.2} \end{array} \xrightarrow{1.9} \begin{array}{l} [0,0.1] \\ [0.9,1.9] \\ \mathbf{c = 6.9} \end{array}$$

$$\xrightarrow{t_2} \begin{array}{l} t_0 \in [1,2] \\ t_1 \in [0.9,\ 1.9] \\ \mathbf{c = 6.9} \end{array}$$

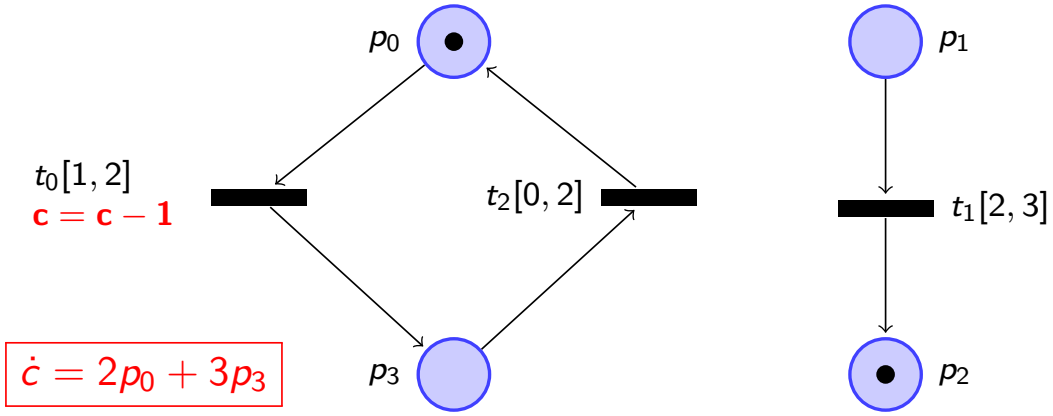
Cost Time Petri Nets



$$\begin{matrix} t_0 \in [1,2] \\ t_1 \in [2,3] \\ \mathbf{c = 0} \end{matrix} \xrightarrow{1.1} \begin{matrix} [0,0.9] \\ [0.9,\ 1.9] \\ \mathbf{c = 2.2} \end{matrix} \xrightarrow{t_0} \begin{matrix} t_2 \in [0,2] \\ t_1 \in [0.9,\ 1.9] \\ \mathbf{c = 1.2} \end{matrix} \xrightarrow{1.9} \begin{matrix} [0,0.1] \\ [0.9,1.9] \\ \mathbf{c = 6.9} \end{matrix}$$

$$\xrightarrow{t_2} \begin{matrix} t_0 \in [1,2] \\ t_1 \in [0.9,\ 1.9] \\ \mathbf{c = 6.9} \end{matrix} \xrightarrow{1.9} \begin{matrix} [0,0.1] \\ [0,\ 0] \\ \mathbf{c = 10.7} \end{matrix}$$

Cost Time Petri Nets



$$\begin{array}{l} t_0 \in [1,2] \\ t_1 \in [2,3] \\ \mathbf{c = 0} \end{array} \xrightarrow{1.1} \begin{array}{l} [0,0.9] \\ [0.9,\ 1.9] \\ \mathbf{c = 2.2} \end{array} \xrightarrow{t_0} \begin{array}{l} t_2 \in [0,2] \\ t_1 \in [0.9,\ 1.9] \\ \mathbf{c = 1.2} \end{array} \xrightarrow{1.9} \begin{array}{l} [0,0.1] \\ [0.9,1.9] \\ \mathbf{c = 6.9} \end{array}$$

$$\xrightarrow{t_2} \begin{array}{l} t_0 \in [1,2] \\ t_1 \in [0.9,\ 1.9] \\ \mathbf{c = 6.9} \end{array} \xrightarrow{1.9} \begin{array}{l} [0,0.1] \\ [0,\ 0] \\ \mathbf{c = 10.7} \end{array} \xrightarrow{t_1} \begin{array}{l} [0,0.1] \\ \\ \mathbf{c = 10.7} \end{array}$$

Cost State Classes

- Cost state classes can be computed as if the cost were a **hybrid** variable (much as for stopwatches);
- They can also be split into finite unions of **simple** cost state classes: $(\text{DBM}(\vec{\theta}), c \geq \ell(\vec{\theta}))$;
- Mincost algorithm is a trivial variation of EF;
- Termination is guaranteed (currently, in the implementation, only if there are no negative cost loops).

Example: A Soft Real-time Scheduling Problem

See demo.

Outline

Introduction

Time Petri Nets

Stopwatch Petri Nets

Timing Parameters

Minimal Cost Reachability

Conclusion

Conclusion

- Buy Roméo **now**!
  - Roméo allows for a wide range of analyses on Time Petri Nets (extended with variables);
  - The additional combined availability of costs, parameters, and stopwatches make it unique;
  - It is constantly evolving as a prototype but has good performance and not too many bugs.
- **Next** evolutions and uses:
  - Add timed control, à la Uppaal-Tiga, but with state classes;
  - Add lazy abstraction based algorithms [JL16];
  - Model the multicore version of Trampoline RTOS [TBFR17]

References I


Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi.
Parametric real-time reasoning.
In *STOC*, pages 592–601. ACM, 1993.


Étienne André, Didier Lime, and Olivier H. Roux.
Integer-complete synthesis for bounded parametric timed automata.
In *RP*, volume 9058 of *Lecture Notes in Computer Science*. Springer, 2015.


Alexander Andreychenko, Morgan Magnin, and Katsumi Inoue.
Analyzing resilience properties in oscillatory biological systems using parametric model checking.
Biosystems, 149:50 – 58, 2016.
Selected papers from the Computational Methods in Systems Biology 2015 conference.


B. Berthomieu and M. Diaz.
Modeling and verification of time dependent systems using time Petri nets.
IEEE Trans. on Soft. Eng., 17(3):259–273, 1991.

References II


G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario.
Timed state space analysis of real-time preemptive systems.
IEEE Trans. on Soft. Eng., 30(2):97–111, February 2004.


Loïg Jezequel and Didier Lime.
Lazy reachability analysis in distributed systems.
In Josée Desharnais and Rhada Jagadeesan, editors, *The 27th International Conference on Concurrency Theory (CONCUR 2016)*, LIPIcs, Québec City, Québec, Canada, August 2016. Dagstuhl Publishing.


Aleksandra Jovanović, Didier Lime, and Olivier H. Roux.
Integer parameter synthesis for timed automata.
IEEE Transactions on Software Engineering, 41(5):445–461, 2015.


Joseph S. Miller.
Decidability and complexity results for timed automata and semi-linear hybrid automata.
In *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–309. Springer, 2000.

References III

Baptiste Parquier, Laurent Rioux, Rafik Henia, Romain Soulat, Olivier H. Roux, Didier Lime, and Étienne André.
Applying parametric model-checking techniques for reusing real-time critical systems.
In Cyrille Artho and Peter Csaba Ölveczky, editors, *5th International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2016)*, Communications in Computer and Information Science, Tokyo, Japan, November 2016. Springer.

Charlotte Seidner.
Vérification des EFFBDs : Model checking en Ingénierie Système. (EFFBDs Verification: Model checking in Systems Engineering).
PhD thesis, University of Nantes, France, 2009.

Toussaint Tigori, Jean-Luc Bechennec, Sebastien Faucou, and Olivier H. Roux.
Formal model-based synthesis of application specific static RTOS.
ACM Transactions on Embedded Computing Systems, 16(4), 2017.

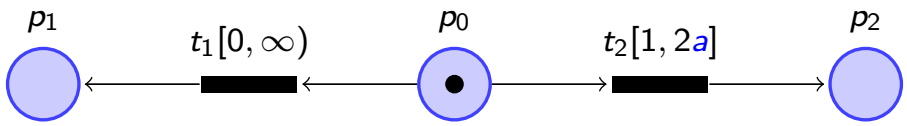
References IV


Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux.
Parametric model-checking of stopwatch Petri nets.
Journal of Universal Computer Science, 15(17):3273–3304, 2009.

Density of the Results of IEF and IAF

- The question:
  - the result of IEF or IAF is a union of convex polyhedra;
  - we know that these sets contain **exactly** the "good" **integer** valuations;
  - but what of the **non-integer** valuations in those polyhedra?
- The short answer:
  - they are all "good" for IEF (but we can do a bit better);
  - they are in general not all "good" for IAF (and we can do a bit better).

The Result of IAF is not Dense



- To ensure AF ($p_1 > 0$), cut t_2 and p_2, i.e., take $a < \frac{1}{2}$;
- When p_2 is marked, $D_2 = \{1 \leq x \wedge 1 \leq 2a\}$, so $\mathsf{IH}(C_2) = \{1 \leq x \wedge 1 \leq a\}$
- So, to cut $(p_2 = 1, \mathsf{IH}(D_2))$, we need $a < 1$.
- $\frac{1}{2} \leq a < 1$ are not "good" valuations.

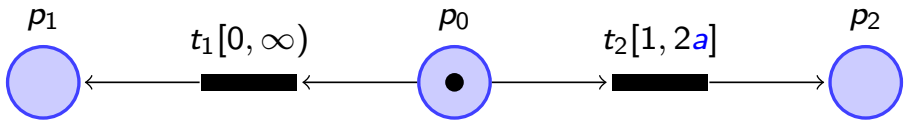
Integer-preserving Dense Underapproximations

- In IAF, we cut off not enough states because $\mathrm{IH}(D) \subseteq D$;
- Solution: use integer hulls only for **convergence** [ALR15]:

$$\mathrm{RIEF}_G(S, M) = \begin{cases} D_{\mathbb{P}} & \text{if } m \in G \\ \emptyset & \text{if } \mathrm{IH}(\mathbf{S}) \in M \\ \bigcup_{\substack{t \in T \\ S' = \mathrm{Next}(S,t)}} \mathrm{EF}_G\left(S', M \cup \{\mathrm{IH}(\mathbf{S})\}\right) & \text{otherwise.} \end{cases}$$

- Similar for AF;
- Gives a "dense" underapproximation containing **at least all integer** valuations.

Dense Integer-preserving Underapproxations



- AF l_1: $a < \frac{1}{2}$ instead of (erroneous) $a < 1$ for IAF
- EF l_2: $a \geq \frac{1}{2}$ instead of $a \geq 1$ for IEF

Scheduling and Strategy Synthesis for Probabilistic Real-Time Systems with PRISM

Dave Parker

University of Birmingham

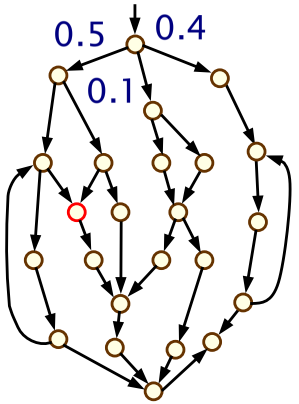
Journée FCH, Nancy, June 2017

Overview

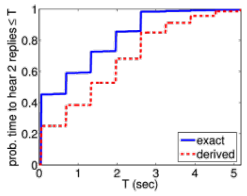
- Probabilistic model checking
  - key ideas: pros and cons
  - tool support: PRISM
  - Markov decision processes + strategy synthesis
  - real-time systems (probabilistic timed automata)
  - example: task graph scheduling under uncertainty
  - more complex property specifications
  - example: robot navigation planning
- Recent/new directions for PRISM
  - multi-objective model checking
  - stochastic games
  - partial observability
  - permissive strategy synthesis

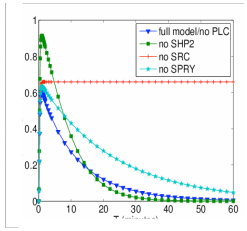
Probabilistic model checking

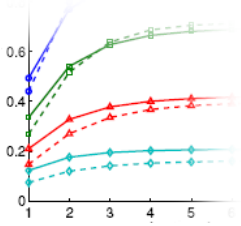
- Construction and analysis of probabilistic models
  - state-transition systems labelled with probabilities (e.g. Markov chains, Markov decision processes)
  - from a description in a high-level modelling language

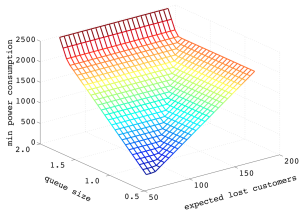


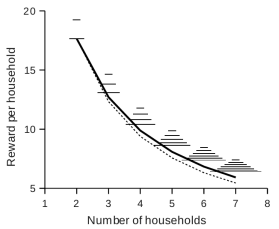
- Properties expressed in temporal logic, e.g. PCTL:
  - trigger → $P_{\geq 0.999}$ [$F^{\leq 20}$ deploy]
  - "the probability of the airbag deploying within 20ms of being triggered is at at least 0.999"
  - properties checked against models using exhaustive search and numerical computation









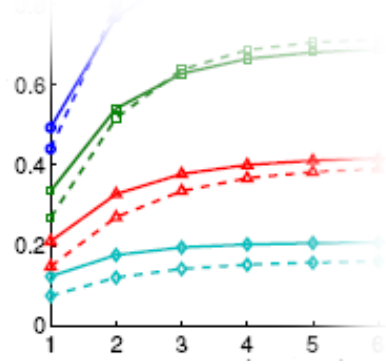


Probabilistic model checking

- Many types of probabilistic models supported
- Wide range of quantitative properties, expressible in temporal logic (probabilities, timing, costs, rewards, …)
- Often focus on numerical results (probabilities etc.)
  - analyse trends, look for system flaws, anomalies

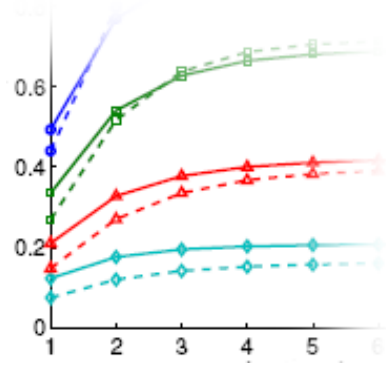
- $P_{\leq 0.1}$ [F *fail*] – "the probability of a failure occurring is at most 0.1"

- $P_{=?}$ [F *fail*] – "what is the probability of a failure occurring?"



Probabilistic model checking

- Many types of probabilistic models supported
- Wide range of quantitative properties, expressible in temporal logic (probabilities, timing, costs, rewards, …)
- Often focus on numerical results (probabilities etc.)
  - analyse trends, look for system flaws, anomalies
- Provides "exact" numerical results/guarantees
  - compared to, for example, simulation (e.g. statistical model checking)
  - also exhaustive analysis (state space, nondeterminism)
  - the trade-off? accuracy vs. scalability

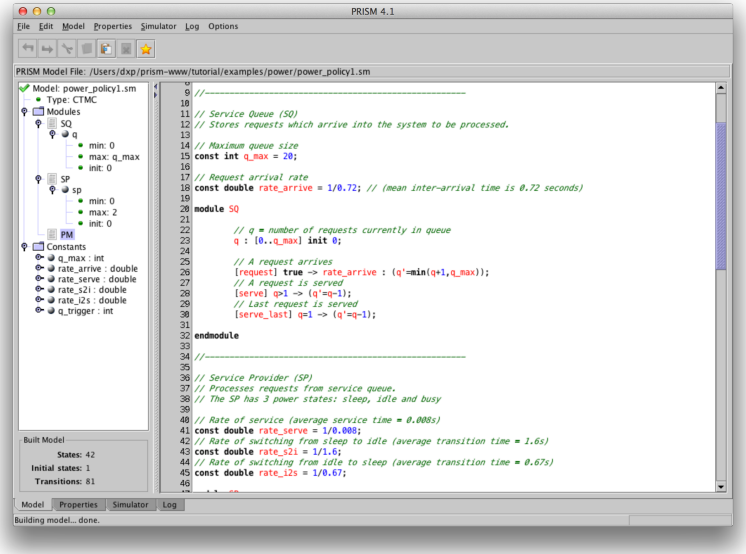


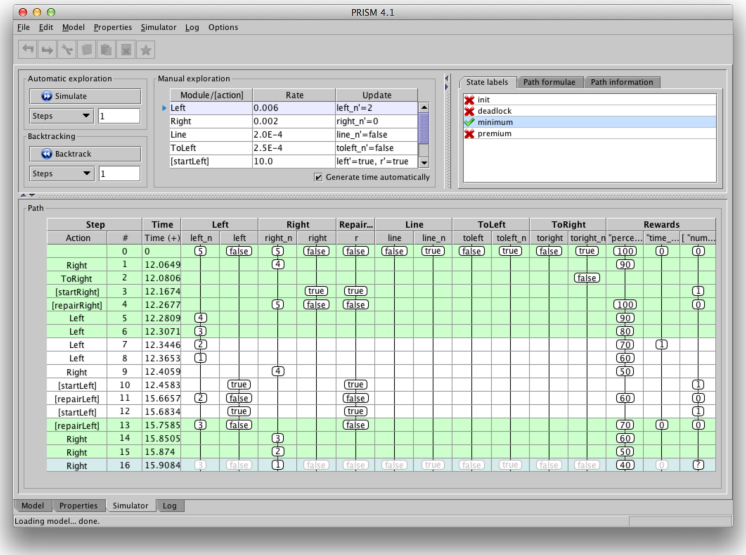
- Fully automated, tools available, widely applicable
  - network/communication protocols, security, biology, robotics & planning, power management, scheduling, …

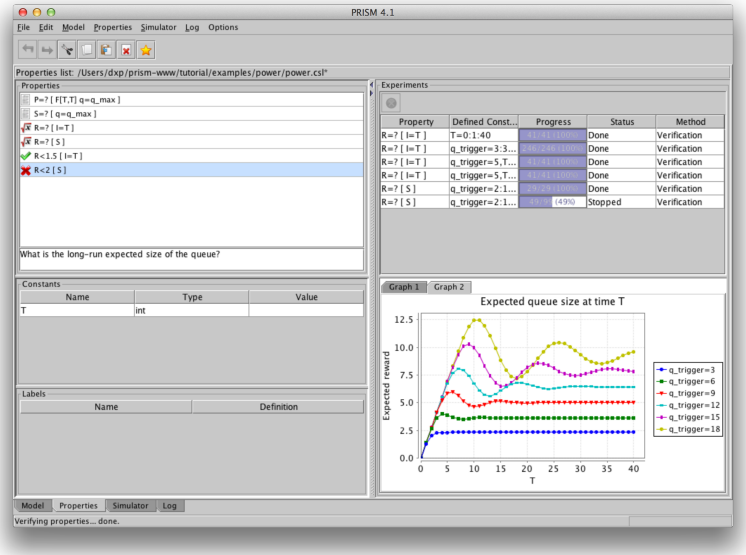
Tool support: PRISM

- PRISM: Probabilistic symbolic model checker [CAV'11]
  - developed at Birmingham/Oxford University, since 1999
  - free, open source software (GPL), runs on all major OSs
- Support for:
  - models: DTMCs, CTMCs, MDPs, PTAs, SMGs, …
  - properties: PCTL, CSL, LTL, PCTL*, rPATL, costs/rewards, …
- Features:
  - simple but flexible high-level modelling language
  - multiple efficient model checking engines (e.g. symbolic)
  - user interface: editors, simulator, experiments, graph plotting
  - recent/new: strategy synthesis, multi-objective model checking, stochastic games, parametric models, …
- See: http://www.prismmodelchecker.org/

The PRISM tool





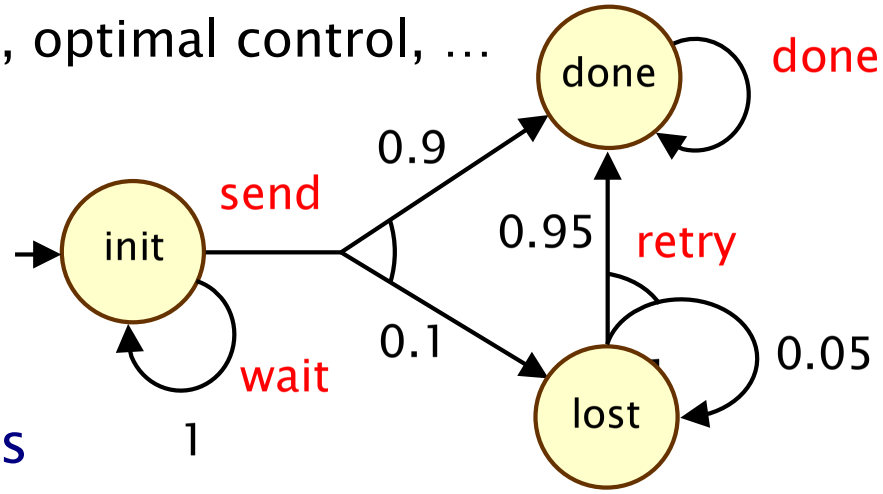


Overview

- Probabilistic model checking
  - key idea: pros and cons
  - tool support: PRISM
  - Markov decision processes + strategy synthesis
  - real-time systems (probabilistic timed automata)
  - example: task graph scheduling under uncertainty
  - more complex property specifications
  - example: robot navigation planning
- Recent/new directions for PRISM
  - multi-objective model checking
  - stochastic games
  - partial observability
  - permissive strategy synthesis

Markov decision processes

- Markov decision processes (MDPs)
  - widely used also in: AI, planning, optimal control, …
- Strategies (or: policies, schedulers, adversaries, …)
  - resolve actions based on history
- Verification vs. strategy synthesis



  - verification: check system satisfies a correctness specification
    - e.g. “the probability of a failure is always < 0.01”
  - strategy synthesis: build a "correct-by-construction" system directly from a specification of correctness
    - e.g. “does there exist a controller (strategy) for which the probability of a failure occurring is < 0.01?”
    - or, optimise: “how to minimise the probability of a failure?”

Probabilistic real-time systems

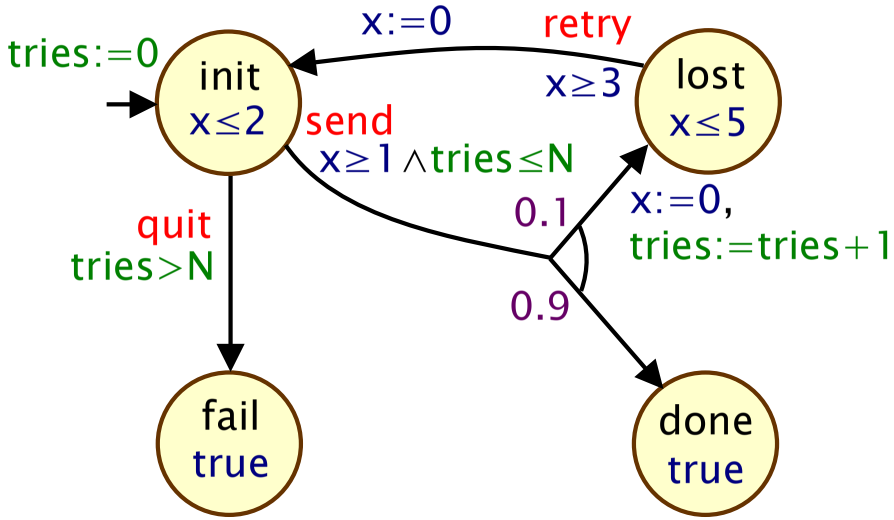
- Probabilistic timed automata (PTAs)
  - Markov decision processes (MDPs) + real-valued clocks
  - or: timed automata + discrete probabilistic choice
  - model probabilistic, nondeterministic and timed behaviour
- PTA example:
  - message transmission over a faulty channel

States
- locations + data variables

Transitions
- guards and action labels

Real-valued clocks
- state invariants, guards, resets

Probability
- discrete probabilistic choice



Modelling PTAs in PRISM

- PRISM modelling language
  - textual language, based on guarded commands

```
pta
const int N;
module transmitter
    s : [0..3] init 0;
    tries : [0..N+1] init 0;
    x : clock;
    invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant
    [send] s=0 & tries≤N & x≥1
          → 0.9 : (s'=3)
          + 0.1 : (s'=1) & (tries'=tries+1) & (x'=0);
    [retry] s=1 & x≥3 → (s' =0) & (x' =0);
    [quit]  s=0 & tries>N → (s' =2);
endmodule
rewards "energy" (s=0) : 2.5; endrewards
```

Modelling PTAs in PRISM

- PRISM modelling language
  - textual language, based on guarded commands

```
pta
const int N;
module transmitter
   s : [0..3] init 0;
   tries : [0..N+1] init 0;
   x : clock;
   invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant
   [send] s=0 & tries≤N & x≥1
         → 0.9 : (s'=3)
         + 0.1 : (s'=1) & (tries'=tries+1) & (x'=0);
   [retry] s=1 & x≥3 → (s' =0) & (x' =0);
   [quit]  s=0 & tries>N → (s' =2);
endmodule
rewards "energy" (s=0) : 2.5; endrewards
```

Basic ingredients:

- modules
- variables
- commands

Modelling PTAs in PRISM

- PRISM modelling language
  - textual language, based on guarded commands

```
pta
const int N;
module transmitter
    s : [0..3] init 0;
    tries : [0..N+1] init 0;
    x : clock;
    invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant
    [send] s=0 & tries≤N & x≥1
          → 0.9 : (s'=3)
          + 0.1 : (s'=1) & (tries'=tries+1) & (x'=0);
    [retry] s=1 & x≥3 → (s' =0) & (x' =0);
    [quit]  s=0 & tries>N → (s' =2);
endmodule
rewards "energy" (s=0) : 2.5; endrewards
```

Basic ingredients:

- modules
- variables
- commands

For PTAs:

- clocks
- invariants
- guards/resets

Modelling PTAs in PRISM

- PRISM modelling language
  - textual language, based on guarded commands

```
pta
const int N;
module transmitter
    s : [0..3] init 0;
    tries : [0..N+1] init 0;
    x : clock;
    invariant (s=0 ⇒ x≤2) & (s=1 ⇒ x≤5) endinvariant
    [send] s=0 & tries≤N & x≥1
          → 0.9 : (s'=3)
          + 0.1 : (s'=1) & (tries'=tries+1) & (x'=0);
    [retry] s=1 & x≥3 → (s' =0) & (x' =0);
    [quit]  s=0 & tries>N → (s' =2);
endmodule
rewards "energy" (s=0) : 2.5; endrewards
```

Basic ingredients:

- modules
- variables
- commands

For PTAs:

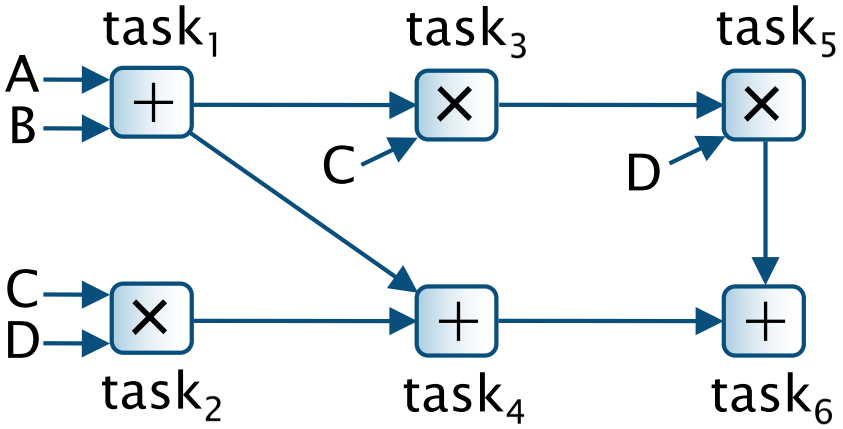
- clocks
- invariants
- guards/resets

Also:

- rewards (i.e. costs, prices)
- parallel composition

Example: Task-graph scheduling

- Use probabilistic model checking of PTAs to solve scheduling problems, e.g. for a task-graph
  - task-graph = tasks to complete + dependencies/ordering
  - for ex.: real-time scheduling, embedded systems controllers
- Simple example: [adapted from BFLM11]
  - evaluate expression: D×(C×(A+B))+((A+B)+(C×D))
  - with subterms evaluated on one of two processors, P_1 or P_2

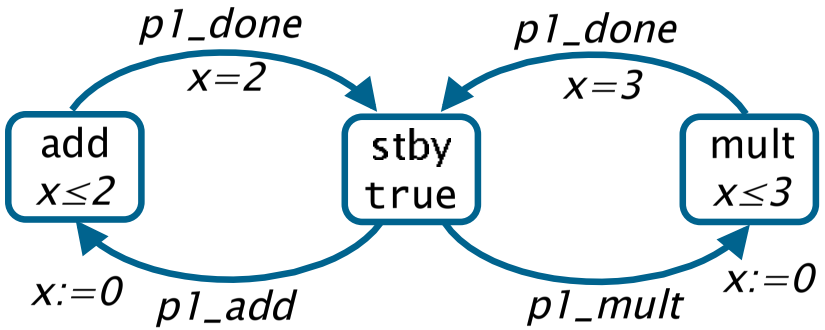


	P_1	P_2
+	2 picoseconds	5 picoseconds
×	3 picoseconds	7 picoseconds
idle	10 Watts	20 Watts
active	90 Watts	30 Watts

Example: Task-graph scheduling

- Task-graph scheduling
  - modelled as (non-probabilistic) timed automata
  - aim to find optimal (time, energy usage, etc.) schedulers
  - PTAs allow us to reason about uncertain delays + failures
  - optimal scheduler derived from optimal strategy

- PTA model
  - parallel composition of 3 PTAs: one scheduler, two processors
  - for example, processor P_1, with local clock x:



Locations also labelled with costs/rewards for time/energy usage

Example: Task-graph scheduling

- Property specification:
  - $R^{time}_{min=?}$ [F complete] – minimise (expected) time
  - $R^{energy}_{min=?}$ [F complete] – minimise (expected) energy usage
- Model check with PRISM (digital clocks)
  - and extract optimal strategy/scheduler
- Time optimal (12 picoseconds)

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	task1			task3				task5				task4		task6						
P_2	task2																			

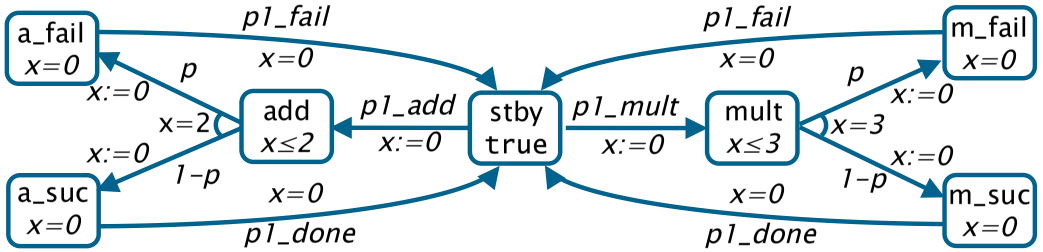
- Energy optimal (1.32 nanojoules)

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	task1			task3				task4												
P_2	task2								task5							task6				

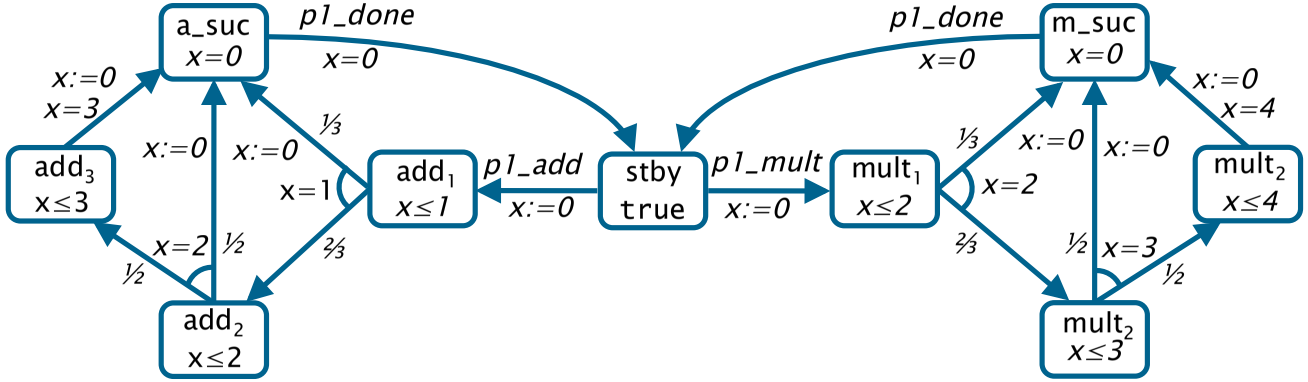
- No probabilities yet…

Adding probabilities

- Faulty processors
  - add third processor P_3: faster, but may fail to execute task



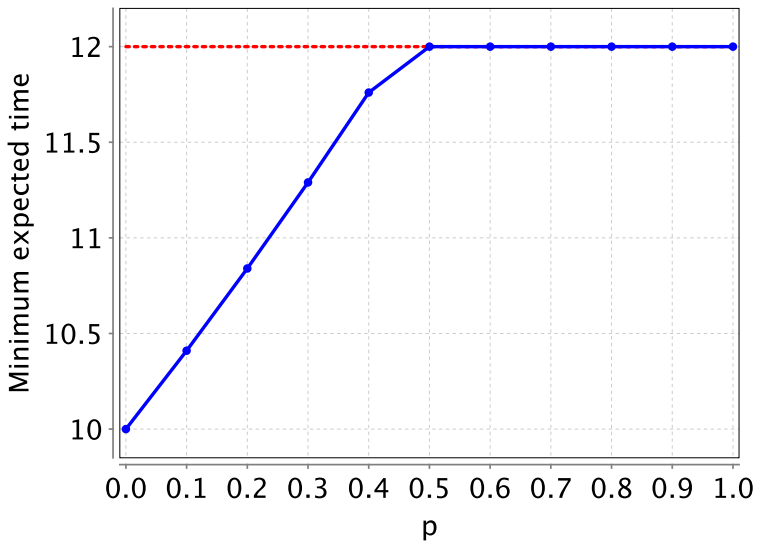
- Probabilistic task execution times
  - simple example: (deterministic) delay of 3 in processor P_1 replaced by distribution: ⅓:2, ⅓:3, ⅓:4



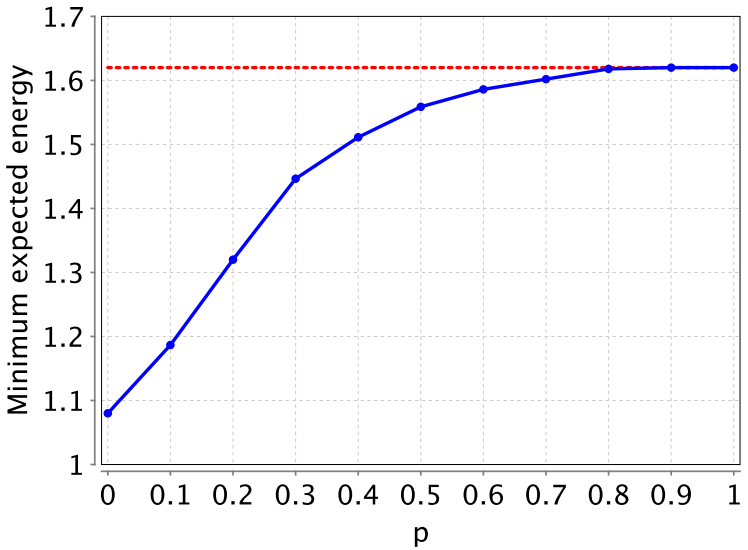
Results (with faulty processor)

- Compute optimal (time/energy) schedulers
  - (using same properties as before)
- Results (for varying failure rates p of processor P_3):
  - dotted red line shows original results (no failures)
  - conclusion: better performance for low values of failure probability p; no benefit for higher values

Expected time

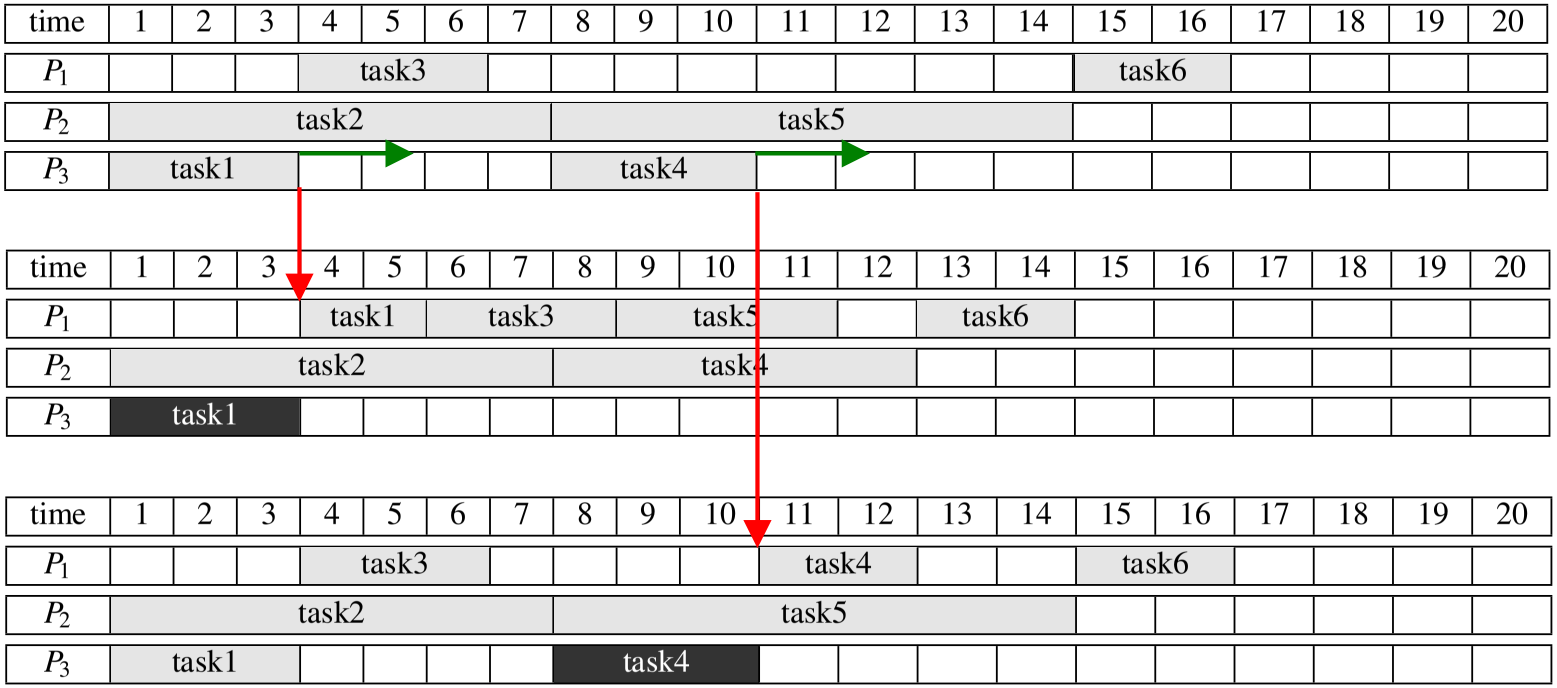


Expected energy usage



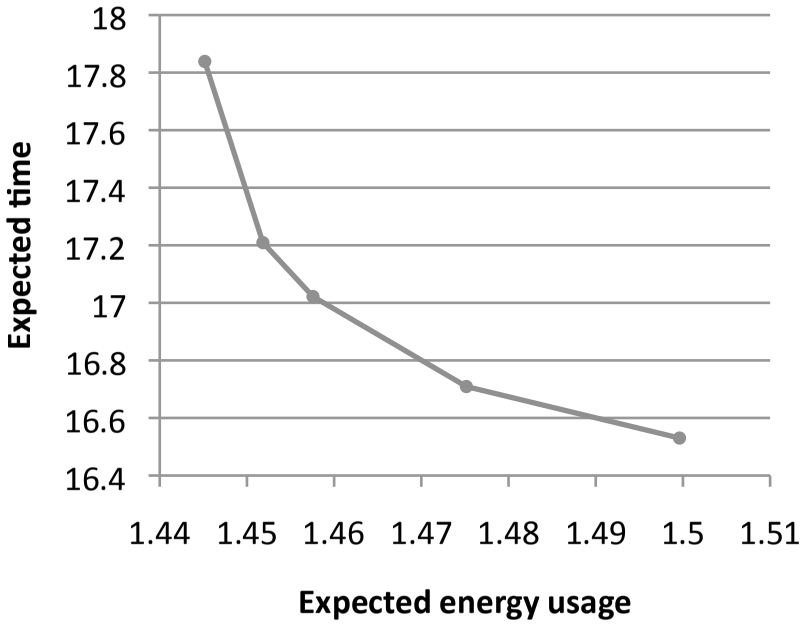
Schedulers (with faulty processor)

- Example (for p=0.5)
  - optimal scheduler to minimise energy consumption
- Optimal scheduler again obtained from strategy
  - now, behaviour depends on outcome of task execution



Multi-objective properties

- Multi-objective controller synthesis
  - (on MDP generated via digital clocks approach)
  - explore trade-off between time/energy usage
- Properties
  - e.g. minimise expected time, subject to bound on energy
  - or: Pareto curve for two objectives: time/energy
  - NB: both may generate randomised schedulers

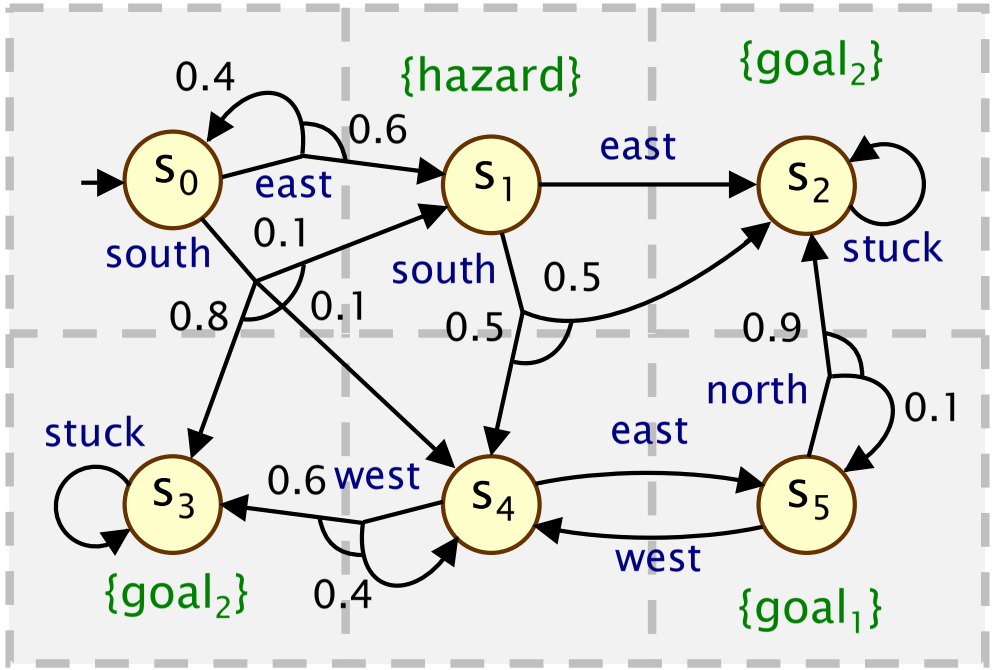


Overview

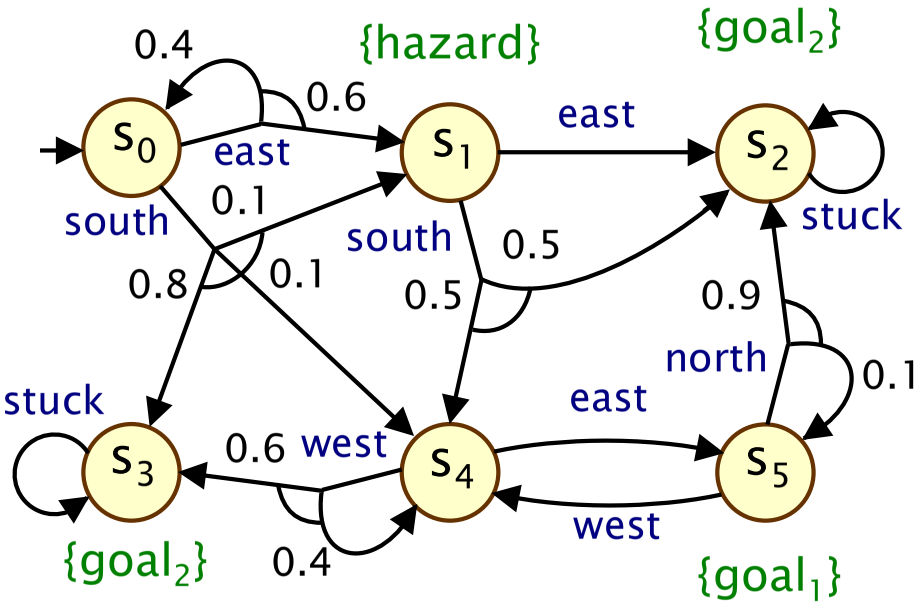
- Probabilistic model checking
  - key idea: pros and cons
  - tool support: PRISM
  - Markov decision processes + strategy synthesis
  - real-time systems (probabilistic timed automata)
  - example: task graph scheduling under uncertainty
  - more complex property specifications
  - example: robot navigation planning
- Recent/new directions for PRISM
  - multi-objective model checking
  - stochastic games
  - partial observability
  - permissive strategy synthesis

Example: Robot navigation planning

- Example MDP
  - robot moving through terrain divided in to 3 x 2 grid



Example – Reachability



Verify: $P_{\leq 0.6}$ [F $goal_1$]

or

Synthesise for: $P_{\geq 0.4}$ [F $goal_1$]

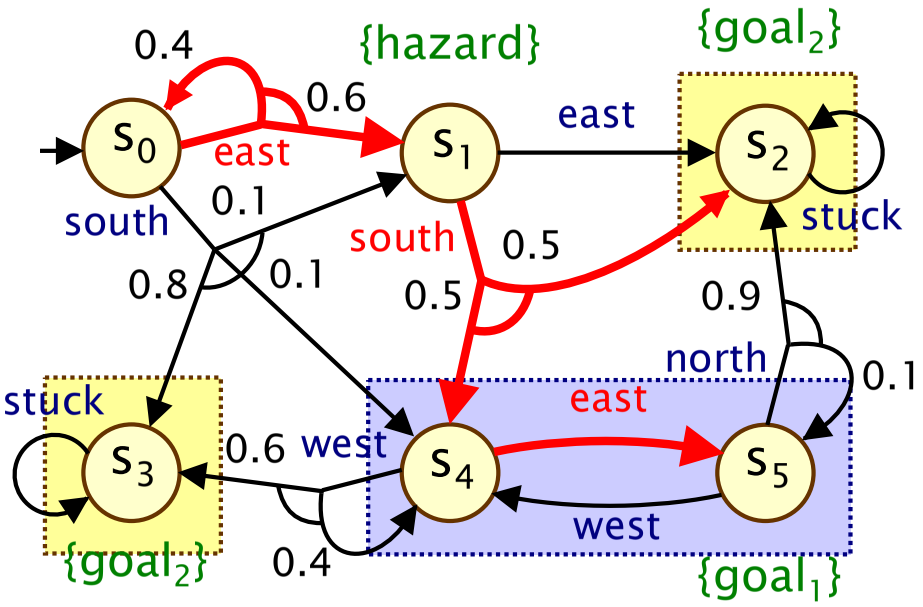
⇓

Compute: $P_{max=?}$ [F $goal_1$]

Optimal strategies:
memoryless and deterministic

Computation:
graph analysis + numerical soln.
(linear programming, value iteration, policy iteration)

Example – Reachability



Optimal strategy:
s_0 : east
s_1 : south
s_2 : –
s_3 : –
s_4 : east
s_5 : –

Verify: $P_{\leq 0.6}$ [F $goal_1$]

or

Synthesise for: $P_{\geq 0.4}$ [F $goal_1$]

⇓

Compute: $P_{max=?}$ [F $goal_1$] = 0.5

Optimal strategies:
memoryless and deterministic

Computation:
graph analysis + numerical soln.
(linear programming, value iteration, policy iteration)

Property specifications

- Linear temporal logic (LTL) – multiple temporal operators
  - e.g. $P_{max=?}$ [(G¬hazard) ∧ (GF $goal_1$)] – "maximum probability of avoiding hazard and visiting $goal_1$ infinitely often?"
  - e.g. $P_{max=?}$ [¬$zone_3$ U ($zone_1$ ∧ F $zone_4$)] – "max. probability of patrolling zones 1 then 4, without passing through 3"
- Costs and rewards (expected, accumulated values)
  - e.g. $R_{max=?}$ [F end] – "what is the worst-case (maximum) expected time for the protocol to complete?"
  - e.g. $R_{min=?}$ [F $goal_2$] – "what is the optimal (minimum) expected number of moves needed to reach $goal_2$?"
- Expected cost/reward to satisfy (co-safe) LTL formula
  - e.g. $R_{min=?}$ [¬$zone_3$ U ($zone_1$ ∧ F $zone_4$)] – "minimise exp. time to patrol zones 1 then 4, without passing through 3"

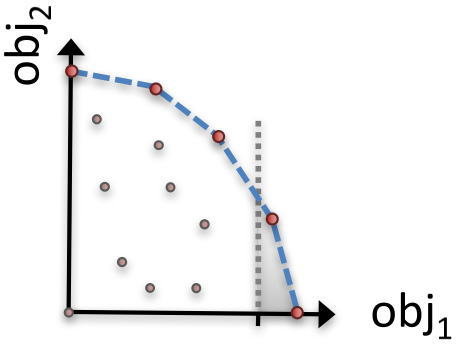
Overview

- Probabilistic model checking
  - key idea: pros and cons
  - tool support: PRISM
  - Markov decision processes + strategy synthesis
  - real-time systems (probabilistic timed automata)
  - example: task graph scheduling under uncertainty
  - more complex property specifications
  - example: robot navigation planning

- Recent/new directions for PRISM
  - multi-objective model checking
  - stochastic games
  - partial observability
  - permissive strategy synthesis

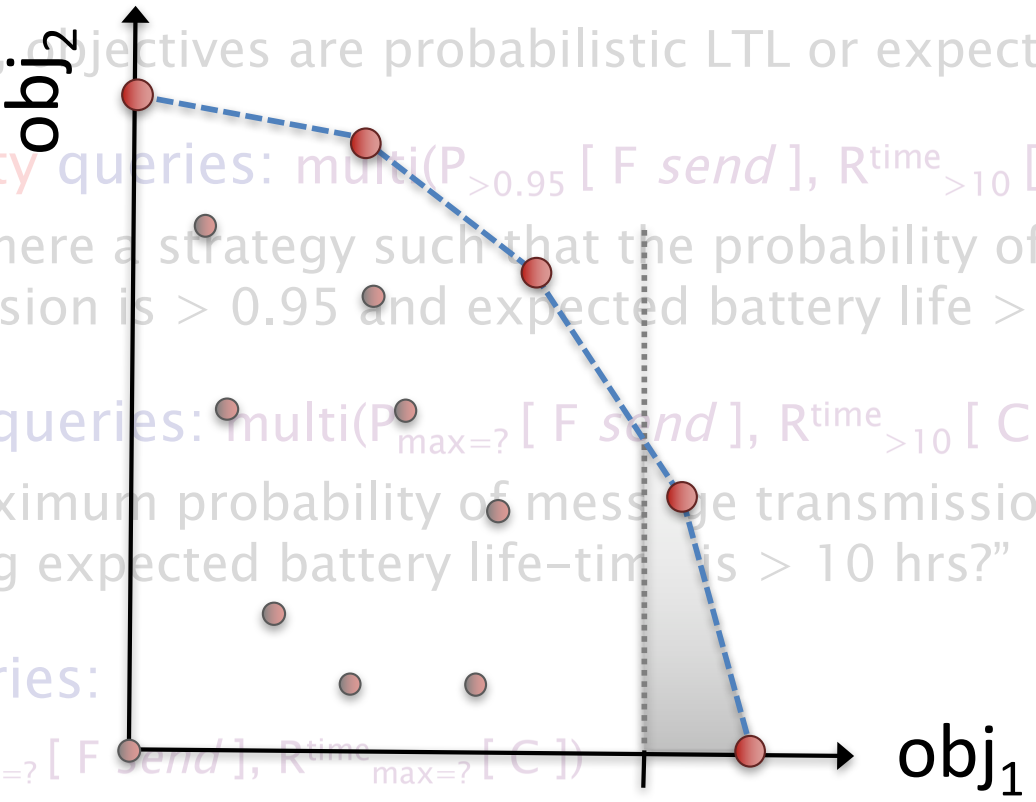
Multi-objective model checking

- Multi-objective probabilistic model checking
  - investigate trade-offs between conflicting objectives
  - in PRISM, objectives are probabilistic LTL or expected rewards
- Achievability queries: multi($P_{>0.95}$ [F *send*], $R^{time}_{>10}$ [C])
  - e.g. “is there a strategy such that the probability of message transmission is > 0.95 and expected battery life > 10 hrs?”
- Numerical queries: multi($P_{max=?}$ [F *send*], $R^{time}_{>10}$ [C])
  - e.g. “maximum probability of message transmission, assuming expected battery life-time is > 10 hrs?”
- Pareto queries:
  - multi($P_{max=?}$ [F *send*], $R^{time}_{max=?}$ [C])
  - e.g. "Pareto curve for maximising probability of transmission and expected battery life-time”

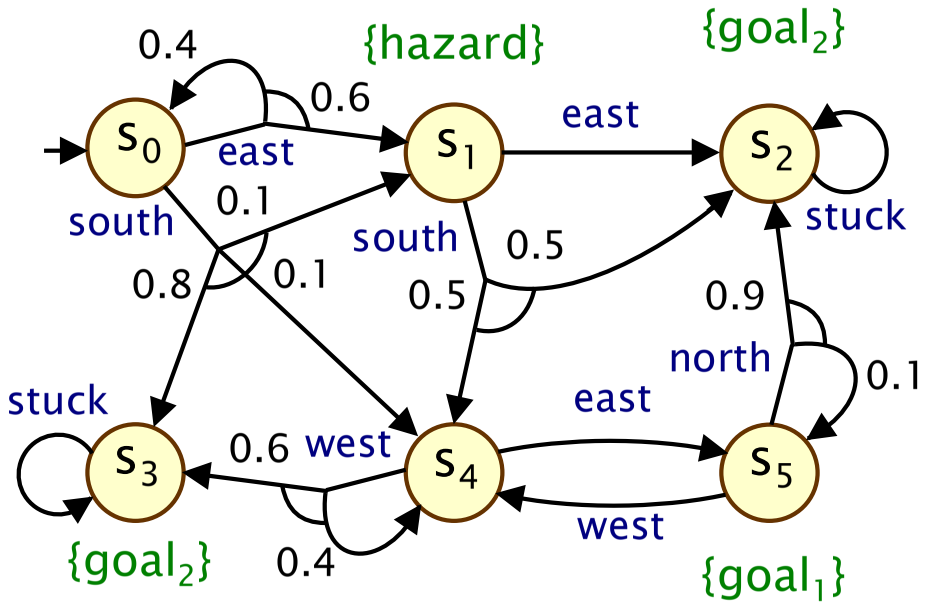


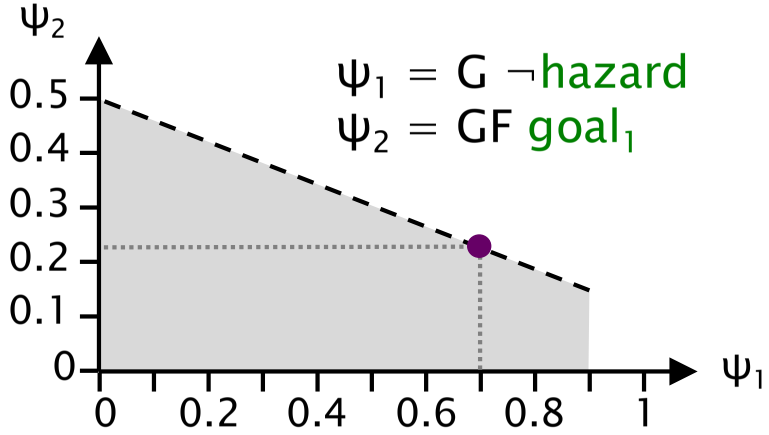
Multi-objective model checking

- Multi-objective probabilistic model checking
  - investigate trade-offs between conflicting objectives
  - in PRISM, objectives are probabilistic LTL or expected rewards
- Achievability queries: multi($P_{>0.95}$ [F *send*], $R^{time}_{>10}$ [C])
  - e.g. "is there a strategy such that the probability of message transmission is > 0.95 and expected battery life > 10 hrs?"
- Numerical queries: multi($P_{max=?}$ [F *send*], $R^{time}_{>10}$ [C])
  - e.g. "maximum probability of message transmission, assuming expected battery life-time is > 10 hrs?"
- Pareto queries:
  - multi($P_{max=?}$ [F *send*], $R^{time}_{max=?}$ [C])
  - e.g. "Pareto curve for maximising probability of transmission and expected battery life-time"



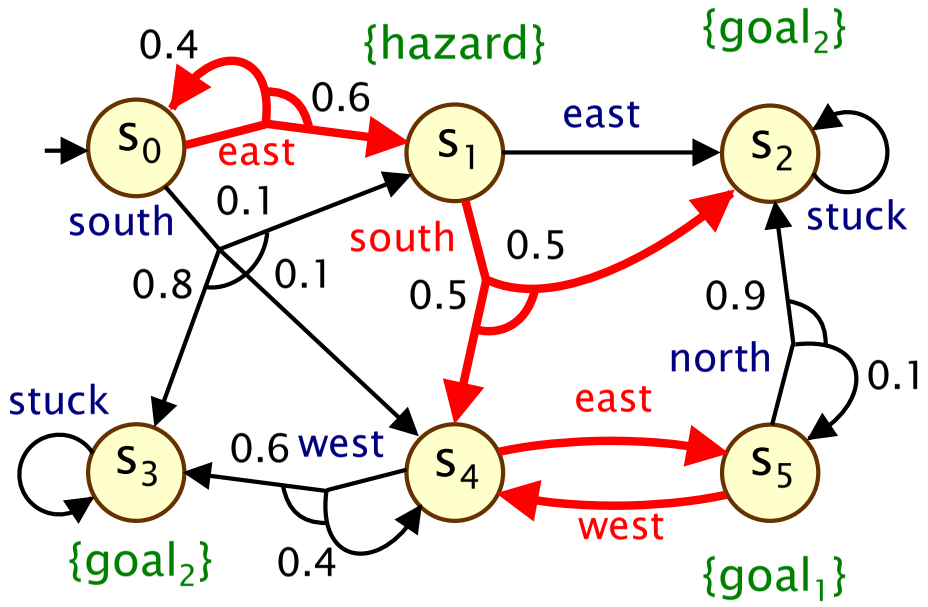
Example – Multi-objective





- Achievability query
  - $P_{\geq 0.7}$ [G ¬hazard] ∧ $P_{\geq 0.2}$ [GF $goal_1$] ? True (achievable)
- Numerical query
  - $P_{max=?}$ [GF $goal_1$] such that $P_{\geq 0.7}$ [G ¬hazard] ? ~0.2278
- Pareto query
  - for $P_{max=?}$ [G ¬hazard] ∧ $P_{max=?}$ [GF $goal_1$] ?

Example – Multi-objective



Strategy 1 (deterministic)

s_0 : east

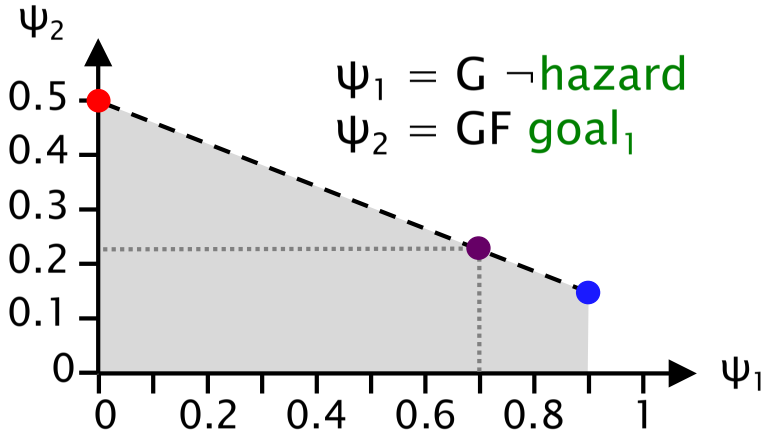
s_1 : south

s_2 : –

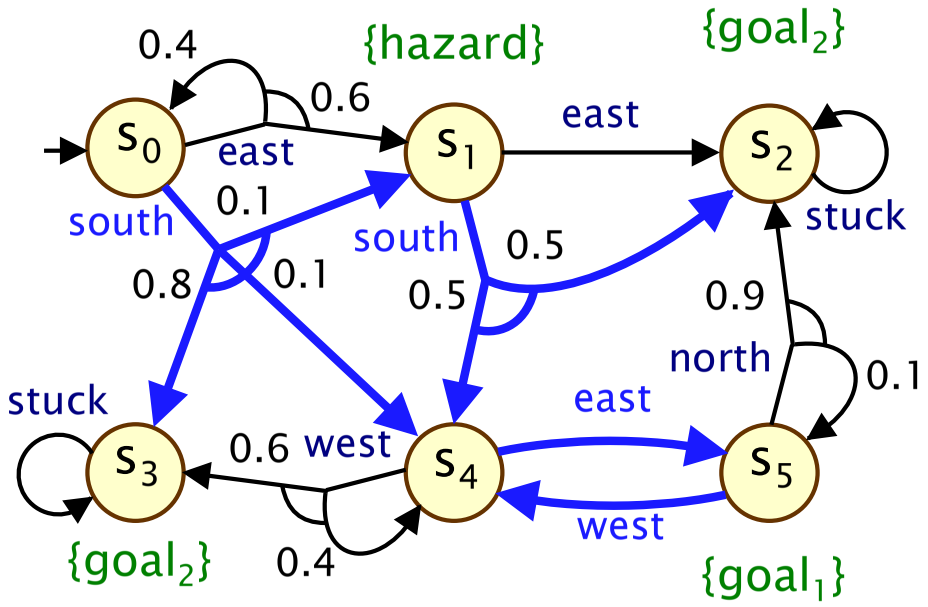
s_3 : –

s_4 : east

s_5 : west



Example – Multi-objective



Strategy 2 (deterministic)

s_0 : south

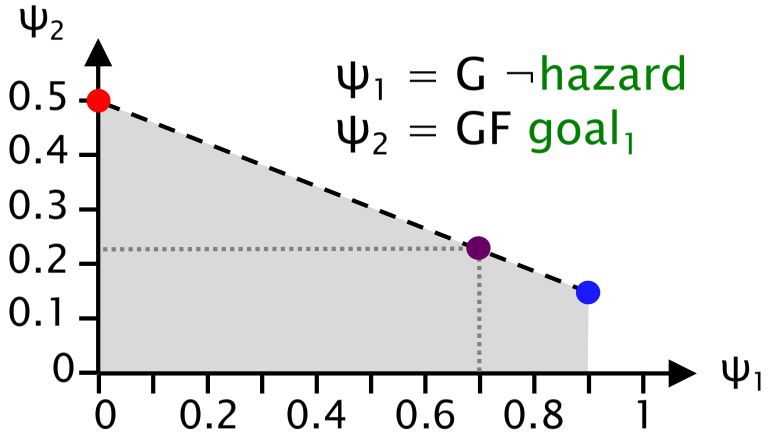
s_1 : south

s_2 : –

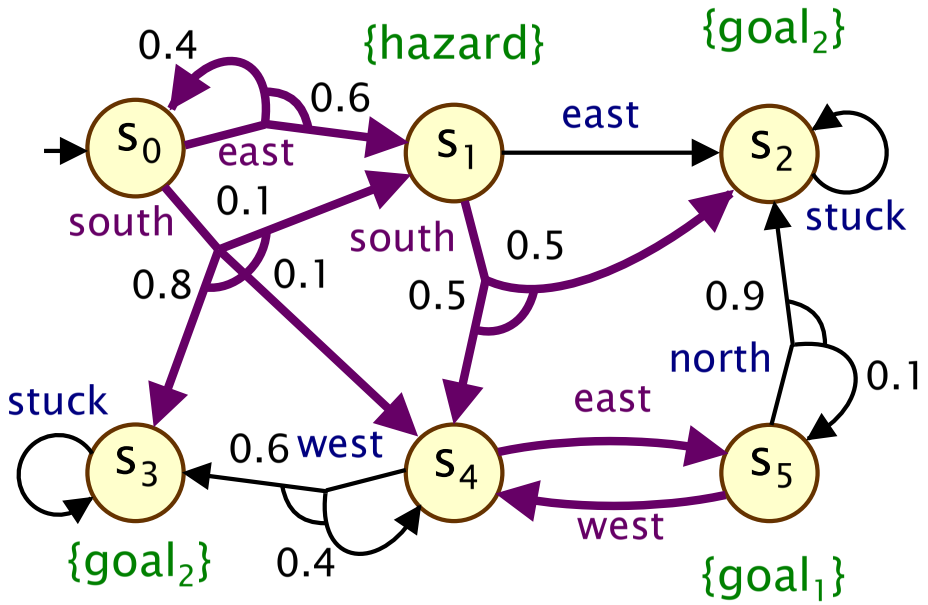
s_3 : –

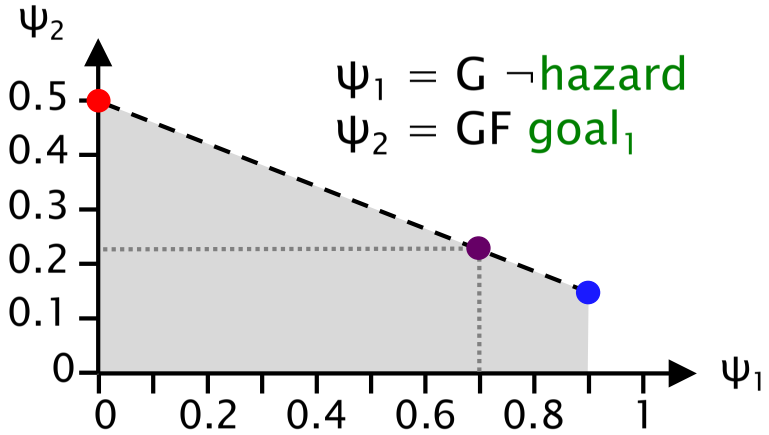
s_4 : east

s_5 : west



Example – Multi-objective





Optimal strategy:
(randomised)

s_0 : 0.3226 : east
0.6774 : south

s_1 : 1.0 : south

s_2 : –

s_3 : –

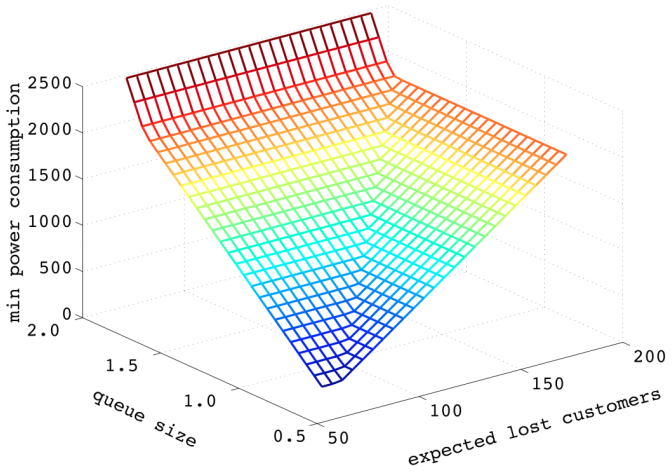
s_4 : 1.0 : east

s_5 : 1.0 : west

Controller synthesis – Applications

- Examples of PRISM-based controller synthesis

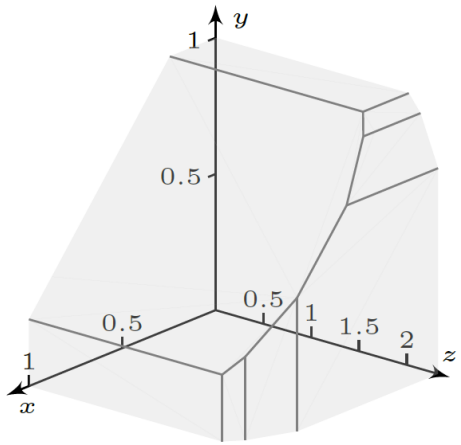
Synthesis of dynamic power management controllers [FKN+11]



Minimise energy consumption, subject to constraints on:
(i) expected job queue size;
(ii) expected number of lost jobs

Motion planning for a service robot using LTL [LPH14b]

Synthesis of team formation strategies [CKPS11, FKP12]



Pareto curve:
x="probability of completing task 1";
y="probability of completing task 2";
z="expected size of successful team"

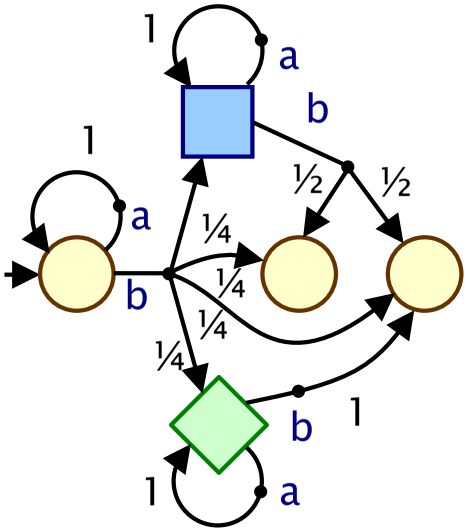
Overview

- Probabilistic model checking
  - key idea: pros and cons
  - tool support: PRISM
  - Markov decision processes + strategy synthesis
  - real-time systems (probabilistic timed automata)
  - example: task graph scheduling under uncertainty
  - more complex property specifications
  - example: robot navigation planning

- Recent/new directions for PRISM
  - multi-objective model checking
  - stochastic games
  - partial observability
  - permissive strategy synthesis

Stochastic multi-player games (SMGs)

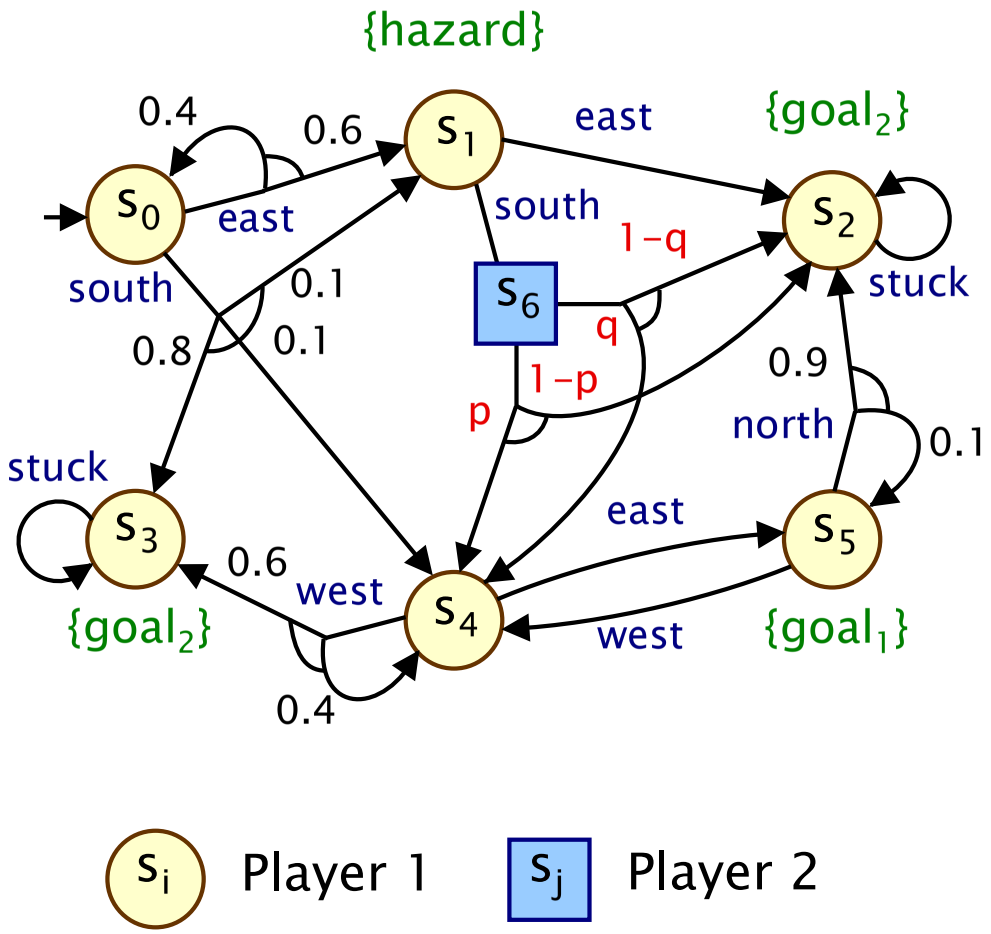
- Stochastic multi-player games
  - players control states; choose actions
  - models competitive/collaborative behaviour
  - applications: security (system vs. attacker), controller synthesis (controller vs. environment), distributed algorithms/protocols, …



- Property specifications: rPATL
  - $\langle\langle\{1,2\}\rangle\rangle\, P_{\geq 0.95}\, [\, F^{\leq 45}\, \mathit{done}\,]$: "can nodes 1,2 collaborate so that the probability of the protocol terminating within 45 seconds is at least 0.95, whatever nodes 3,4 do?"
  - formally: $\langle\langle C\rangle\rangle\psi$: do there exist strategies for players in C such that, for all strategies of other players, property ψ holds?
- Model checking [TACAS'12,FMSD'13]
  - zero sum properties: analysis reduces to 2-player games
  - PRISM-games: www.prismmodelchecker.org/games

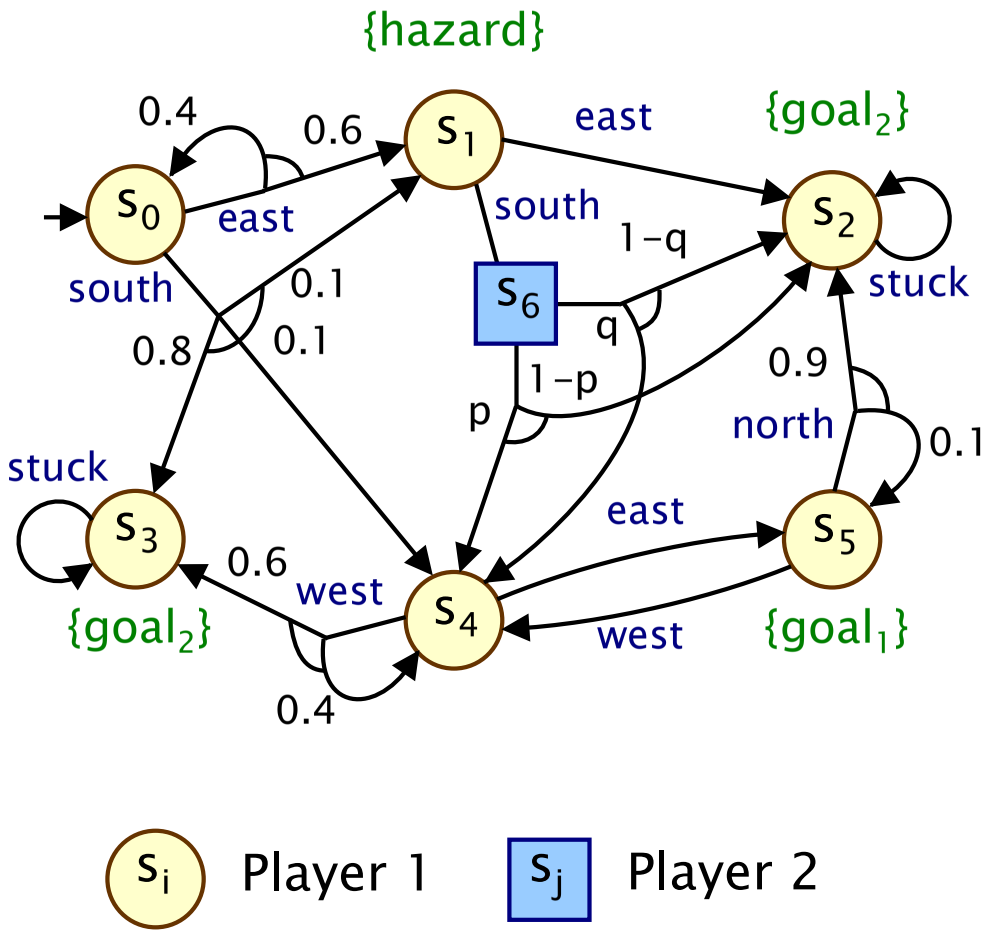
Example – Stochastic games

- Two players: 1 (robot controller), 2 (environment)
  - probability of s_1–south→s_4 is in [p,q] = [0.5–Δ, 0.5+Δ]



Example – Stochastic games

- Two players: 1 (robot controller), 2 (environment)
  - probability of s_1–south→s_4 is in [p,q] = [0.5–Δ, 0.5+Δ]



rPATL: ⟨⟨{1}⟩⟩ $P_{max=?}$ [F $goal_1$]

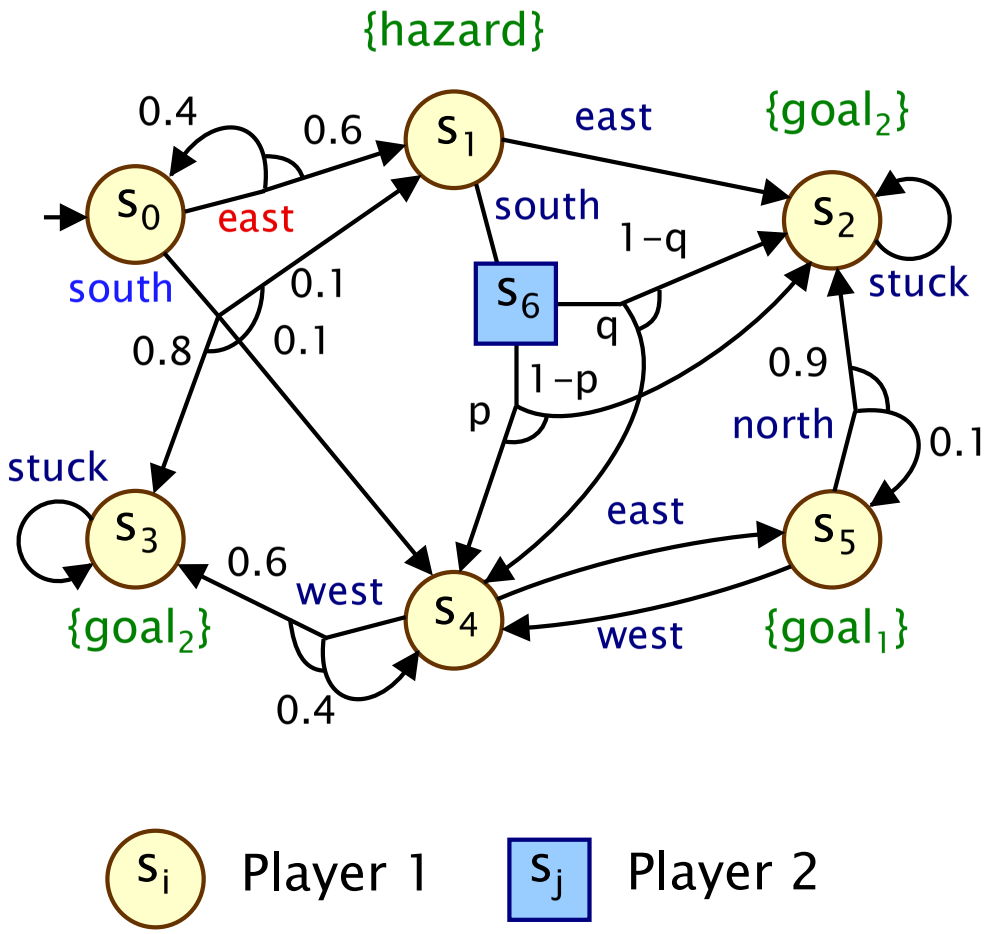
Optimal strategies:
memoryless and deterministic

Computation: graph analysis
& numerical approximation

s_i Player 1 s_j Player 2

Example – Stochastic games

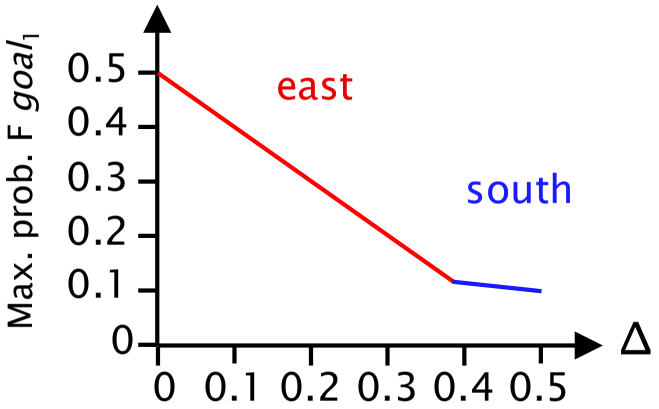
- Two players: 1 (robot controller), 2 (environment)
  - probability of s_1–south→s_4 is in [p,q] = [0.5–Δ, 0.5+Δ]



rPATL: $\langle\langle\{1\}\rangle\rangle\, P_{max=?}[\, F\, goal_1\,]$

Optimal strategies:
memoryless and deterministic

Computation: graph analysis
& numerical approximation



Overview

- Probabilistic model checking
  - key idea: pros and cons
  - tool support: PRISM
  - Markov decision processes + strategy synthesis
  - real-time systems (probabilistic timed automata)
  - example: task graph scheduling under uncertainty
  - more complex property specifications
  - example: robot navigation planning

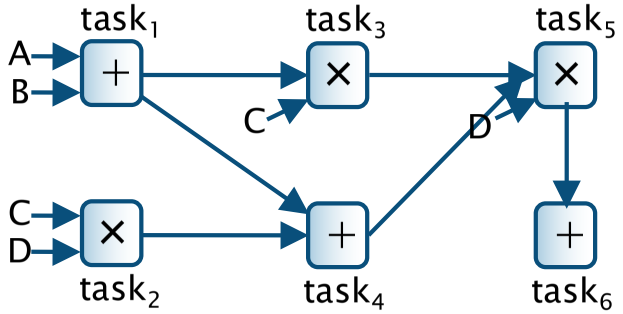
- Recent/new directions for PRISM
  - multi-objective model checking
  - stochastic games
  - partial observability
  - permissive strategy synthesis

Partial observability

- Partial observable Markov decision processes (POMDPs)
  - limit strategies ability to view precise states of the MDP
- Motivation
  - e.g. because robot can only make decisions based on sensors
  - e.g. because scheduler cannot probe state of a channel
- Optimal strategies
  - resolve actions based on observations; maintain belief state about the true state of the MDP
- Implemented in prototype extension of PRISM
  - PRISM model variables declared as observable/hidden
  - computes lower/upper bounds for optimal values and a (possibly sub-optimal) strategy with grid-based approximations
  - real-time extension: partially observable PTAs [FORMATS'15]
  - more details (for POMDPs and POPTAs), plus tool support and case studies, in [RTS'17]

POMDP/POPTA Case studies

- Task graph scheduling
  - processors have different speeds and energy consumption
  - scheduler cannot observe if a process is sleeping or idling
  - synthesize optimal schedulers
    - again, minimising expected execution time or energy usage



- Wireless network scheduling
  - schedule traffic to number of users/channels
  - packets have hard deadlines (packets not sent by their deadline are dropped) and priorities
  - status of channels is not available (unobservable)
  - generate optimal scheduling of packets, maximising priorities and minimising dropped packets
  - demonstrates that idling is sometimes the optimal choice

Overview

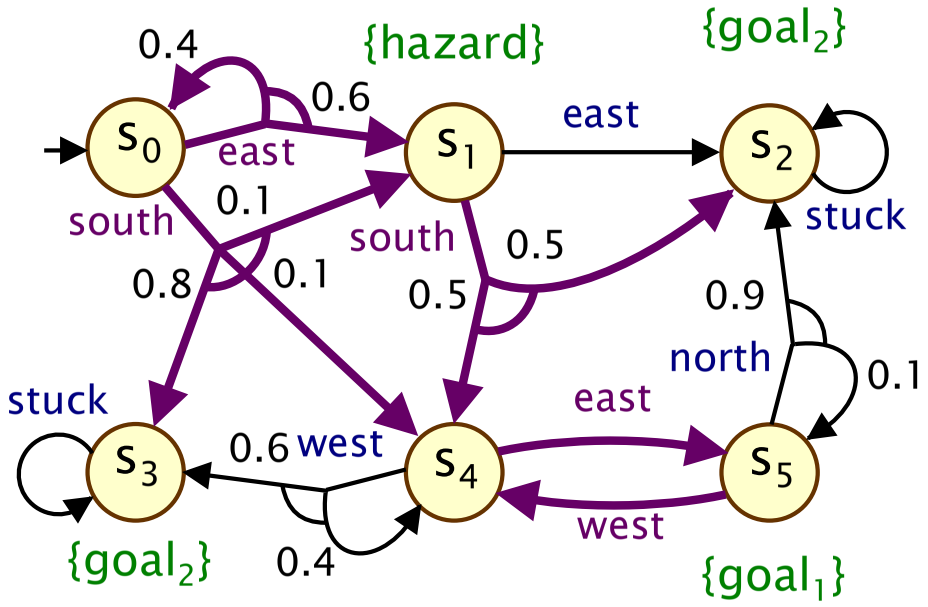
- Probabilistic model checking
  - key idea: pros and cons
  - tool support: PRISM
  - Markov decision processes + strategy synthesis
  - real-time systems (probabilistic timed automata)
  - example: task graph scheduling under uncertainty
  - more complex property specifications
  - example: robot navigation planning

- Recent/new directions for PRISM
  - multi-objective model checking
  - stochastic games
  - partial observability
  - permissive strategy synthesis

Permissive controller synthesis

- Multi-strategy synthesis [TACAS'14]
  - for Markov decision processes and stochastic games
  - choose sets of actions to take in each state
  - controller is free to choose any action at runtime
  - flexible/robust (e.g. actions become unavailable or goals change)

- Example



Multi-strategy:

s_0 : east or south

s_1 : south

s_2 : –

s_3 : –

s_4 : east

s_5 : west

Permissive controller synthesis

- Multi-strategies and temporal logic
  - multi-strategy Θ satisfies a property $P_{>p}$ [F goal] iff any strategy σ that adheres to Θ satisfies $P_{>p}$ [F goal]
- We quantify the permissivity of multi-strategies
  - by assigning penalties to each action in each state
  - a multi-strategy is penalised for every action it blocks
  - static and dynamic (expected) penalty schemes
- Permissive controller synthesis
  - $\exists$ a multi-strategy satisfying $P_{\leq 0.6}$ [F $goal_1$] with penalty $< c$?
  - what is the multi-strategy with optimum permissivity?
  - reduction to mixed-integer LP problems
  - applications: energy management, cloud scheduling, …

Conclusion

- Probabilistic model checking
  - verification vs. strategy synthesis
  - Markov decision processes, probabilistic timed automata, …
  - applicable to scheduling problems under uncertainty
- Recent extensions
  - multi-objective model checking
  - stochastic games
  - partial observability, permissive strategy synthesis
- Challenges & directions
  - controller synthesis methods, e.g. MCTS, UCT, RDTP
  - partial information/observability: greater efficiency
  - scalability, e.g. symbolic methods, abstraction
  - stochastic games: multi-objective, equilibria, richer logics

Thanks for your attention

More info here:

www.prismmodelchecker.org