Spécification & vérification formelle de systèmes temps-réels (Automates)

- Spécification et vérification
- Rappels
- Enjeux du temps réel
- Modèles comportementaux : les automates
- Expression des exigences sous forme logique

Spécification & vérification de modèles

Spécification et modèle

- **Spécification formelle : description abstraite du système reposant sur un formalisme mathématique**

- **Modèle de comportements :**
  Description de la dynamique du système
  - Variation de l'état du système
  - Enchainement d'événements (instant ou changement d'état clés du système).
- **Un modèle ==**
  - Définit l'espace d'états du système et/ou ses événements
  - Définit **ses interactions** avec son **environnement** via la description de ses **comportements possibles**

Exigences et Vérification

- **Objectif :**
  Décrire l'ensemble des comportements souhaités
  - Par un modèle de comportement
  - Par langage exprimant un besoin sans avoir à décrire comment le satisfaire !!!
- **Une exigence == définit une contrainte sur l'ensemble des états que le système peut ou doit occuper (idem pour leur variation)**
- **Vérification : comparaison d'un modèle décrivant les comportements possibles, et d'une exigence décrivant ceux qui sont souhaités pour s'assurer que le possible correspond au souhaité.**

Exemple sur l'ordonnancement

Le modèle d'ordonnancement Liu et Layland

- Modèle du système :
  - ensemble fini de tâches devant partager le processeur
  - tâches préemptibles en temps nul
  - Tâches périodiques Indépendantes munies de priorités
  - De pire temps d'exécution borné connu et correct
  - Au plus une tâche s'exécute à un même instant t...

- Exigences :
  - Respect des échéances par rapport aux dates d'activations
  - Pas d'inversions de priorités

- **Vérification** : test de faisabilité de l'ordonnancement conditions nécessaires / suffisantes.

Rappels

Rappels élémentaires en formalisation

- **Construction inductive d'un ensemble repose sur :**
  - Un ensemble B « d'exemples » d'éléments de X (les cas simples)
  - Un ensemble de *règles* permettant de *construir*e des éléments de X à partir d'un ensemble d'éléments contenus dans X
- **Traduction mathématique :**
  - B = sous ensemble de X
  - Règle = fonction de $X^n \rightarrow X$
- **Exemples : les entiers pairs, et les expression arithmétiques**

Rappel sur les langages

- Soit ∑ un ensemble de symboles
- ∑* désigne l'ensemble des séquences de symboles contenus dans ∑.
- L'opérateur '.' permet de concaténer 2 séquences
- *Un ensemble L* inclus dans ∑* est un langage (séquence = mot => ensemble de mots = langage)
- '.' peut concaténer des langages

- Exemple : A= {+,*,1,2,3,4,5,6,7,8,9,0,(,),}
  Que représente A* ?

Exemple expressions arithmétiques et entiers pairs

- **Entiers Pairs :**
  - B= {0}
  - ∏ = { f} avec f : x → x+2 pour x entier.
- **Expressions arithmétiques EA:**

  B = {1,2,3,4,5,6,7,8,9}.{1,2,3,4,5,6,7,8,9,0}*

  ∏ :
  - Addition : EA x EA → EA
    a, b → a+b

    Mul : EA x EA → EA
    a b → a*b
  - Parenth : EA → (EA)
    a → (a)

Les concepts abstraits pour modéliser une exécution

- La séquence : crée un ordre total permettant de décrire la progression de l'exécution pas à pas

- Les ordres partiels : permet de décrire la notion de parallélisme et causalité ≠ écoulement du temps

Problème n° 2

- **Un lot de tâches RT dépendantes (sémaphores)**
  - Modèles décrivant les séquences d'appels à P() et V() (ou post et wait)
  - Définition de l'exécution par une alternance d'actions de synchronisation et de phases de calcul
  - Identification d'états dits de section critique dans chaque modèle.
  - Modèle de fonctionnement du sémaphore
- **Exigences :**
  - Temps de réponse inférieur à la deadline
  - Pas d'accès multiples aux sections critiques

Les systèmes à transition et les traces

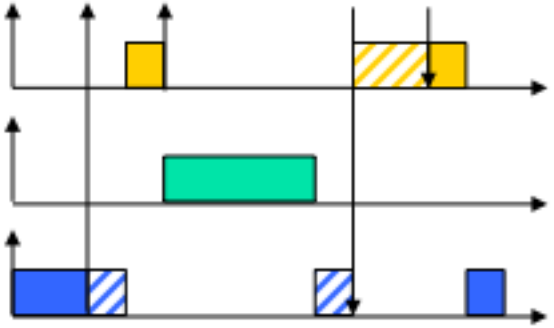
- **Une structure de graphe pour représenter un ensemble d'exécutions**
  - S = ensemble d'états
  - E = (SxS) ensemble de transitions
  - *L : E → ∑, fonction d'étiquetage des transitions*
- **Une exécution == un chemin ~ séquence :**

  $(e_n)_{n \text{ in } \mathbb{N}}$ ⩥ deux visions: séquences finies/infinies
- **Types d'exigences et méthodes de vérification**
  - **Etre sur d'éviter des transitions ou états redoutés**
  - **Etre sur d'atteindre des états ou franchir des transitions**
  - **Invariant d'état / de topologie**

Enjeux spécifiques aux Systèmes embarqués temps réels

Les contraintes

- **Causalité vs positionnement temporel :**
  - Ordre sur les estampilles temporelles
  - Ordre causal



- **Contraintes quantitatives (temps, ressources, précision)**
  - Deadlines
  - Borner les appels récursifs
  - Borner les taux de défaillance

Tests de faisabilité, et quoi d'autre ?

- **Vérification de la faisabilité d'un ordonnancement**
  - Méthode analytique directe (Liu et Layland)
  - Génération de traces d'exécution par simulation
  - Analyse de modèles à transitions
- **Les autres objectifs vérifications**
  - Temps blocage borné dans le temps
  - Absence de violation de priorités dans un lot de tâche TR
  - Respect de contraintes de précédence induites par les échéances et dates d'activation

Tests de faisabilité, et quoi d'autre ?

- **Vérification de la faisabilité d'un ordonnancement**
  - Méthode analytique directe (Liu et Layland)
  - Génération de traces d'exécution par simulation
  - Vérification de modèles à transitions (UPPAAL / Times
- **Les autres vérifications**
  - Temps blocage borné dans le temps
  - Absence de violation de priorités dans un lot de tâche TR
  - Respect de contraintes de précédence induites par les échéances et dates d'activation

Automates et modélisation du comportement

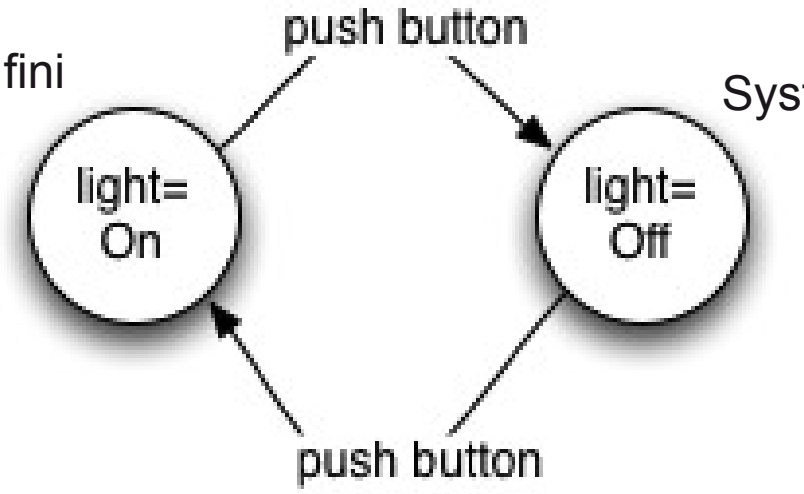
Automates finis
vision événement vs transitions

- **Définition d'un automate fini : A= (Q0,Q,∑,Δ,F)**
  - Q est un ensemble d'états, contenant e0, l'état dit initial, et un sous-ensemble F d'états dit finaux
  - ∑ est l'ensemble des événements qui seront associés aux transitions de l'automates
  - Δ est l'ensemble des transitions franchissables dans l'automate, une transition est de la forme (q,e,p) avec q et p dans Q et e dans ∑.
- franchissement d'une transition

  Si l'automate est dans l'état q, alors toute transition de la forme (q,e,p) dans Δ peut être franchie, l'état de l'automate devient alors q'.

L'automate : arc et nœuds étiquettés

Espace d'état fini
+
transitions
finies...



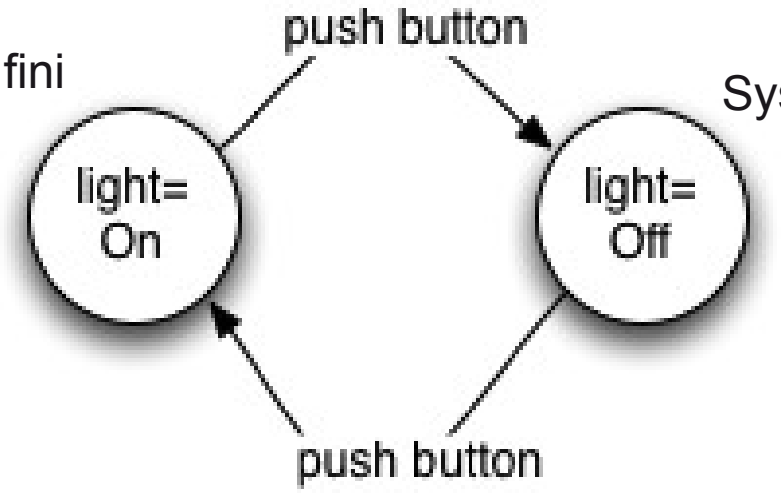
Système à transition ~ automate

Les cas « utiles » :

- Espace d'état infini ou très complexe (temps + état logique), état de piles
- Abstraction du TS en regroupant les états et les transitions
- Définition de règles de transitions paramétrées

Vers des automates plus riches...

Espace d'état fini
+
transitions
finies...



Système à transition ~ automate

Exemple d'exécution
On.Off.On.Off.On …...
Ou Push.On.Push.Off

Espace d'état == ensemble des nœuds
Alphabet des événements=étiquettes sur les arcs

Deux visions en opposition du « temps »

- **Idée : on veut pouvoir mesurer différents type de durées (longueur d'un intervalle ou somme de longueurs d'intervalles)=> fonction du temps**
- **Temps physique – horloges**
  - Fonction presque tout le temps croissante (== sauf en quelques points isolés== remises à zéro)
  - Permet de mesurer le temps séparant deux événements
- **Chronomètres**
  - Fonction définie par morceau sur des intervalles I dans Int, telle que sur chaque I, la fonction est soit croissante, soit constante.
  - Mesure du temps passé dans certains états.

Des automates finis vers les automates étendus

- **Extensions de l'espace d'état**
  - Un nœud de l'automate doit représenter plusieurs états

  => Introduction de variables (entiers, booléens ..)
- **Extension de l'étiquetage des transitions**
  - 1 arc + étiquette pour un grands nombre de transitions d'état % contexte d'exécution
  - Pouvoir n'autoriser les transition que pour un sous ensemble des état d'un nœud => notion de garde == contrainte
  - Exprimer la variation des états sous forme d'affectation

Comment exprimer des contraintes simples ?

- Réponse : utiliser des expressions logiques (formules)
- Le cas simple : modélisé 'et' 'ou', 'négation de' sur des variable booléennes => logique propositionnelle
- But définir des **expressions booléennes** contenant des **variables** telle que l'expression sera évaluée par substitution des variables par des constantes booléennes

Logique propositionnelle

- **Soit Ap un ensemble de variables booléennes**
- **Et Soit ¬, ∧, et ∨ des fonctions dites booléennes**
- **L'ensemble des formules propositionnelle Prop :**
  - B= AP U {true, false}
  - ∏ = {**¬, ∧, ∨**} **les opérateurs tels que si F1 et F2 sont deux formules différentes :**

    - ¬F1 est une formule

    - F1∧F2 est une formule

    - F1 ∨F2 est une formule aussi

Interprétation et valuation

- Une valuation est une fonction de AP dans {true , false }
- Principe de substitution : F[Y/x] est la formule telle que toute occurrence de x est remplacé par Y.
- L'évaluation d'une formule pour une valuation donnée consiste à substituer chaque variable par son image via la valuation.

Extension aux autres types : les prédicats

- **La difficulté est que si l'on manipule des types autres qu'entier, on doit toujours finir par les transformer en booléens**
- **Prédicat : fonction transformant un ensemble de n paramètres de types T1, … Tn en une valeur booléenne**
- **Arité du prédicat : nombre de ses paramètres**
- **Logique des prédicat : même règles d'induction mais B contient :**
  - Les variables booléennes
  - Les expressions que l'on peut créer en appliquant des prédicats sur des variables non booléennes

La logique du premier ordre

- **1er ordre = prédicat + quantification**
- **Quantification == contraindre l'ensemble des valeurs d'une variable pour lesquelles il faut évaluer une formule**
  - ∀ : pour toutes les valeurs
  - ∃ : pour au moins une valeur
- **Variable libre** : variable non quantifié
- **Satifiable** : si la formule possède des variables libre => il existe une affectation des ces dernière rendant la formule vrai
- **Valide** : la formule est vrai pour toutes valuations de ses variables libres.

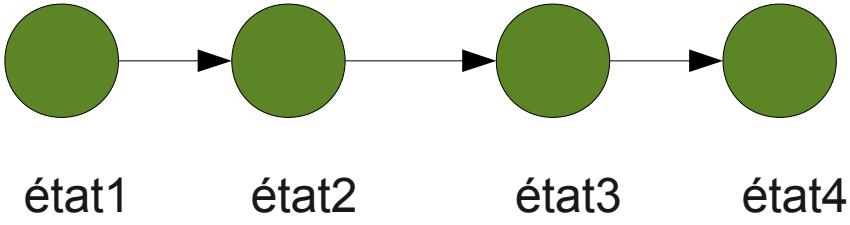
Notion d'automate étendu

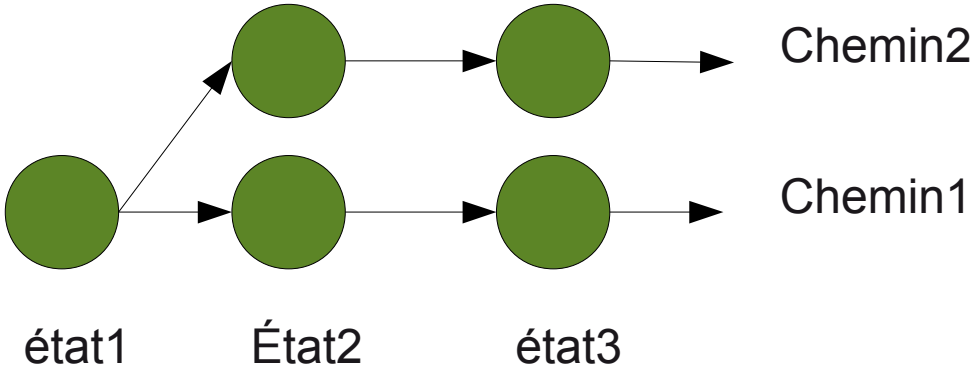
- V : un ensemble de variable
- Q : un ensemble d'état de contrôle
- E : un ensemble d'arcs QxQ
- La : un étiquetage des arcs par des expressions d'affectation (de la forme x:= exp)
  ou exp est une expression arithmétique sur des constantes et variables numériques
- Lg : un étiquetage des arcs par des gardes (formules dans la logique des prédicats/ premier ordre).

Logiques temporelles

Motivation

- **Modéliser l'écoulement continu du temps est difficile**
  => le modèle d'exécution est pas à pas
- **2 visions vont être plébiscités :**





Syntaxe PLTL

- **Définition : les formules de PLTL sont définies par la grammaire suivante :**

  $\Phi,\Psi ::= p \mid q \mid ... \mid true \mid false \mid$ (formules atomiques)

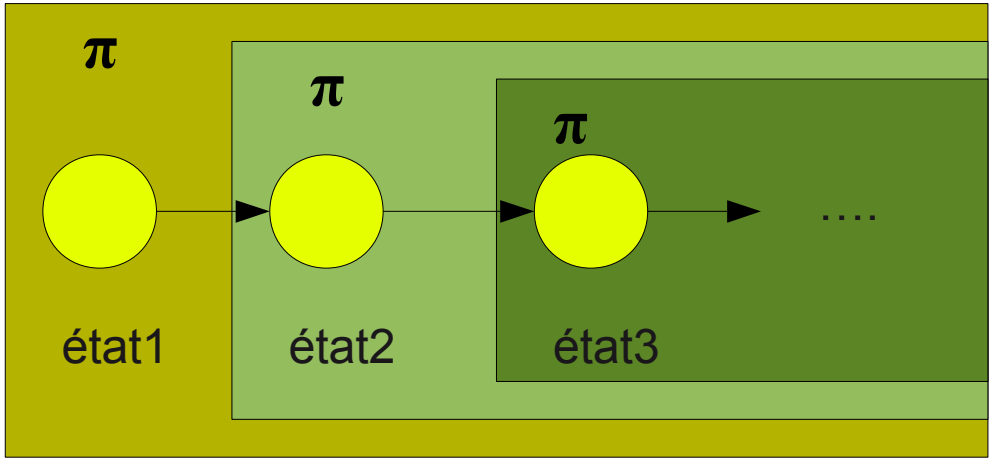
  $\Phi \wedge \Psi \mid \Phi \vee \Psi \mid \Phi \Rightarrow \Psi \mid \neg\Phi \mid$ (connecteurs booléens)

  $F\Phi \mid G\Phi \mid \Phi U \Psi \mid X\Phi$ (opérateur temporels)

- **Ces formules s'interprètent sur les séquences d'états**
- **Les formules de LTL sont obtenues en considérant des prédicats atomiques à la place des propositions atomiques, et en ajoutant les quantificateurs ∀ et ∃ sur les variables qu'ils manipulent**

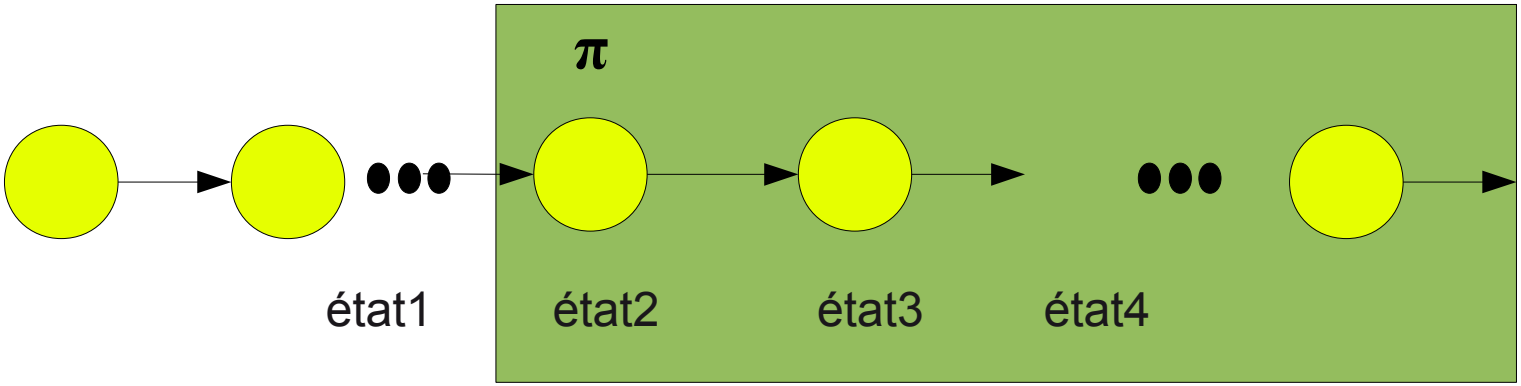
Interprétation pratique

- Soit π une formule PLTL, G π est vrai ssi



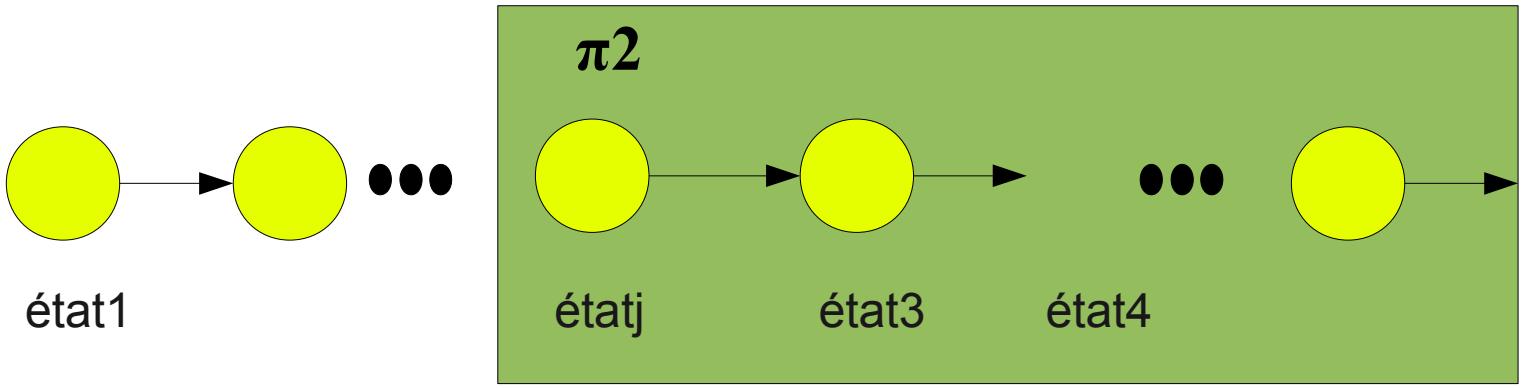
Interprétation pratique

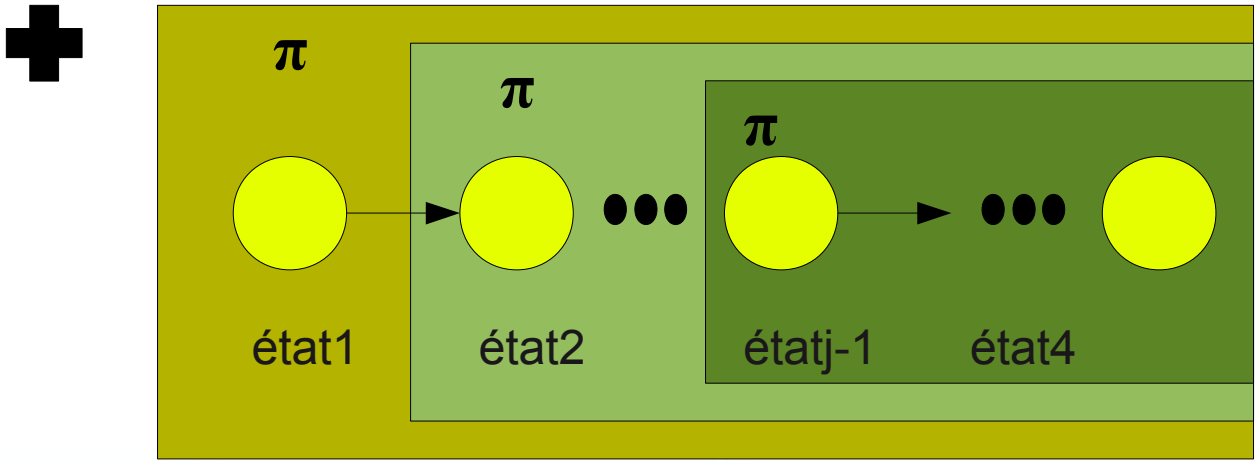
- Soit π une formule PLTL, F π est vrai ssi



Interprétation pratique

Soit π une formule PLTL, π1 U π2 est vrai ssi





Interprétation des opérateurs temporels

- **Supposons que la trace π soit constituée de la séquence s0.s1.s2.....sk**
- **Gp: «p est toujours vérifié dans le futur de l'exécution»**

  pour tout j sj prouve p *Permet de prouve***r Gp**
- **Fp : «p finit par être vrai dans le futur »**

  il existe j≥0, tel que sj prouve p *Permet de prouver* **Fp**
- **Xp : «p est vrai au prochain pas d'exécution »**

  s1 prouve p *Permet de prouver* **Xp**
- **p U q : « p est vrai tant que q ne l'est pas au moins une fois »**

  Il existe j, tel que pour tout k<j, sk ne prouve pas q et sj prouve q, et sk prouve q *Permet de prouver* **que p U q**

Extensions temporisées

- **Un nouveau modèle de traces :**
  - Idée : ajouter une durée physique aux états
    => (Etat, durée)* ou (événement,date)* ou un mix
- **Pour la logique :**
  Ajout de contraintes temporelles sur les opérateurs
  - $G_{<10}$: G s'applique sur l'intervalle relatif [0,10[
  - $G_{>10}$: G s'applique sur l'intervalle]10;+∞[

    ...
- **Pour les systèmes à transitions Observer :**

  Automates temporisés déterministes (dont on peut construire le complémentaire)

Operateurs CTL

- Interpreted on trees
- A π : π has to be satisfied on each path on the execution tree
- E π : there exists one path on which π is satisfied

- Exemples en TP

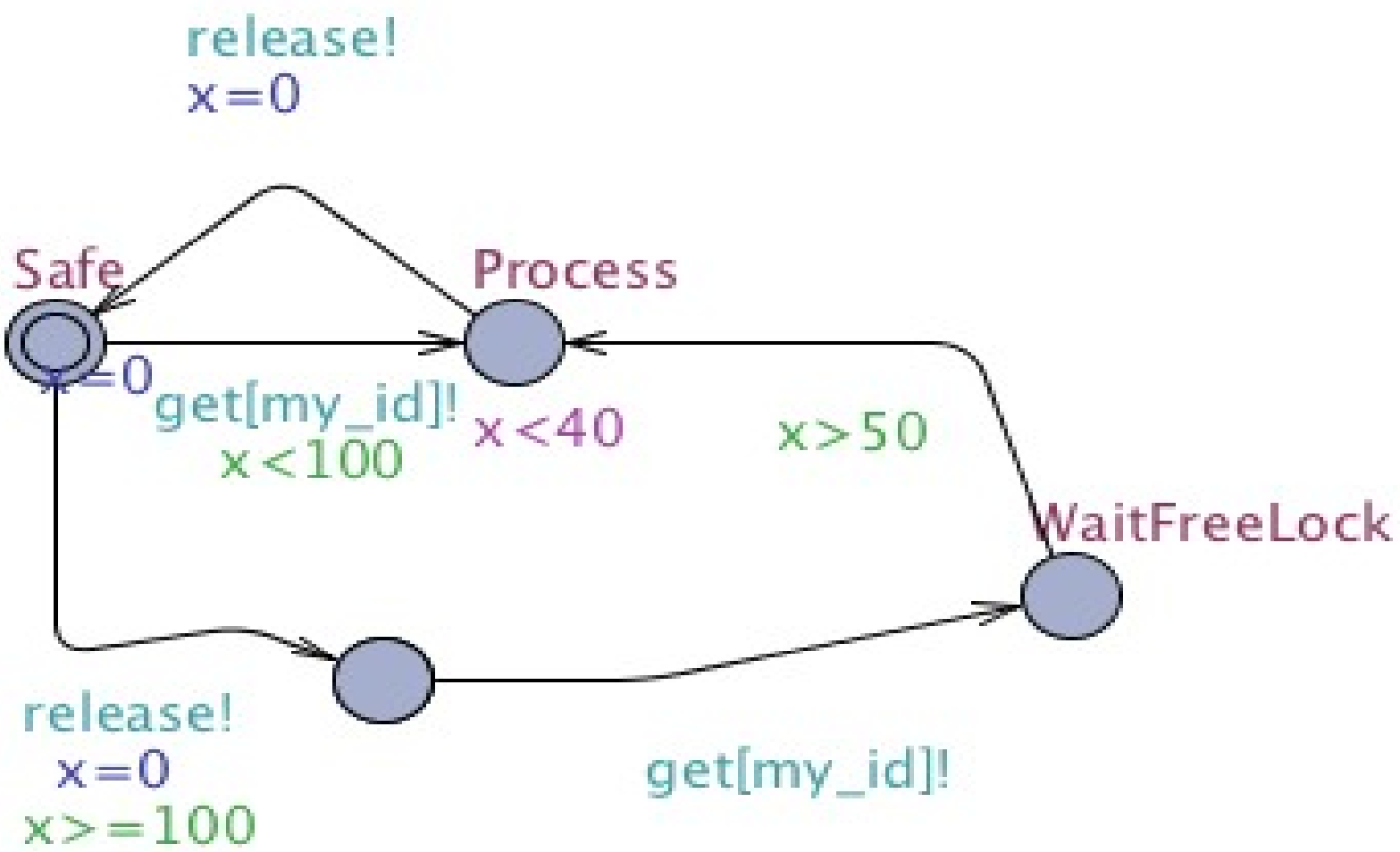
Extension temporisée des automates

Des automates finis vers les automates temporisés (II)

- **Pb: comment connait-on le langage de trace d'un automate temporisé ?**
- **Le système à transition est à priori infini avec un nombre infini de transition**
  - État :
    - *l* : une location (un élément dans un ensemble fini)
    - *u* : une valuation des horloges (un vecteur de réels positifs)
  - Les transitions :
    - Écoulement du temps sans «déplacement » seul u change
    - Changement de lieu sans écoulement du temps seul l change
    - Et les deux en même temps => deux transitions

  Pourquoi cette restriction ?
  Réponse : compatible avec les traces temporisées

Automates temporisés



Complexité de l'analyse des automates temporisés – les résultats théoriques

- **Résultats basés langage:**
  - Stabilité de l'ensemble des automates temporisés par union, intersection, concaténation.
  - Pas de stabilité par l'opération 'complémentaire de'
- **Corolaire :**
  - L'inclusion du langage d'un automate temporisé dans un autre est indécidable (pas d'algo général)
- **Complexité**
  - Le problème *« L(A) est vide »* est P-SPACE complet
  - Le problème *L(A) inclus dans L(B)* est: décidable si B est déterministe, sa complexité est P-SPACE complet

Les extensions UPPAAL

- **Extension des états :**
  - Ajout de variables de type entiers finis et tableaux
  - Ajout d'une programmation impérative simplifiée pour décrire l'impact des transitions sur les variables discrètes.
- **Extension des relations de transitions**
  - Ajout de règles supplémentaires pour réduire certaines situations de non déterminisme.
- **Allez ici pour en savoir plus**

  http://www.di.unipi.it/~maggiolo/Lucidi_TA/VerifyingTA-Uppaal.pdf

Expression des exigences et stratégies de vérification

Les exigences comportementales

- **Approche observer**
  - Définir un automate plus simple : se concentre sur quelques événement, leur ordre et des échéances
  - Danger : faire la différence entre vrai exigence et dérivés de solutions
- **Approche logique**
  - Utiliser un formalisme de logique pour identifier les invariants et les exigences de type réponse en temps borné
  - Pb : comment représenter le temps ?

Interprétation des opérateurs temporels

- **Supposons que la trace π soit constituée de la séquence s0.s1.s2.....sk**
- **Gp: «p est toujours vérifié dans le futur de l'exécution»**

  pour tout j sj prouve p *Permet de prouve***r Gp**
- **Fp : «p finit par être vrai dans le futur »**

  il existe j≥0, tel que sj prouve p *Permet de prouver* **Fp**
- **Xp : «p est vrai au prochain pas d'exécution »**

  s1 prouve p *Permet de prouver* **Xp**
- **p U q : « p est vrai tant que q ne l'est pas au moins une fois »**

  Il existe j, tel que pour tout k<j, sk ne prouve pas q et sj prouve q, et sk prouve q *Permet de prouver* **que p U q**

Extensions temporisées

- **Un nouveau modèle de traces :**
  - Idée : ajouter une durée physique aux états
    => (Etat, durée)* ou (événement,date)* ou un mix
- **Pour la logique :**
  Ajout de contraintes temporelles sur les opérateurs
  - $G_{<10}$: G s'applique sur l'intervalle relatif [0,10[
  - $G_{>10}$: G s'applique sur l'intervalle]10;+∞[

    ...
- **Pour les systèmes à transitions Observer :**

  Automates temporisés déterministes (dont on peut construire le complémentaire)

Vérification
théorie et savoir faire

Vérification

- **Confrontation entre les exigences et le modèle d'implémentation**

  IL DOIT EXISTER UN DOMAINE D'INTERPRÉTATION COMMUN
  (qui est souvent la trace ou l'exécution)

- **En pratique : logique contre Système à transition**
  - Un langage d'expressions logiques pour contraindre le comportement (Linear Temporal Logic)
  - Un système à transition modélisant l'implémentation du système étudié (TS)
- **Vérification : s'assurer que les exécutions d'une Formule F sont inclues dans celle du TS**

Stratégie de vérification F contre TS
Les Observateurs formels

- Transformer la formule F en sa négation : non (F)
- Construire le système à transition reconnaissant toutes les traces satisfaisant non (F) TNF
- Réaliser le produit synchronisé entre TNF et TS

- Si l'automate résultant reconnaît une trace
  => contre exemple de « TS valide F »

Formalisation des propriétés « safety » et « liveness »

- **Une propriété de safety :: Je ne veux pas de …**
  - G not (…) ou G (…)
- **Une propriété démontrant une progression :**
  - A U B :: correspond à une transition
  - + borne temporelle == propriété de vivacité bornée

    => ces cas peuvent se prouver en montrant que l'on atteint ou pas certains état

    ON PARLE D'ANALYSE D'ACCESSIBILITE
- **Une propriété d'équité → il faut combiner un invariant et une propriété de progression :**
  - G (E (…)) ce cas est un peu à part ...

Ce qui se cache derrière : Analyse d'accessibilité d'un état

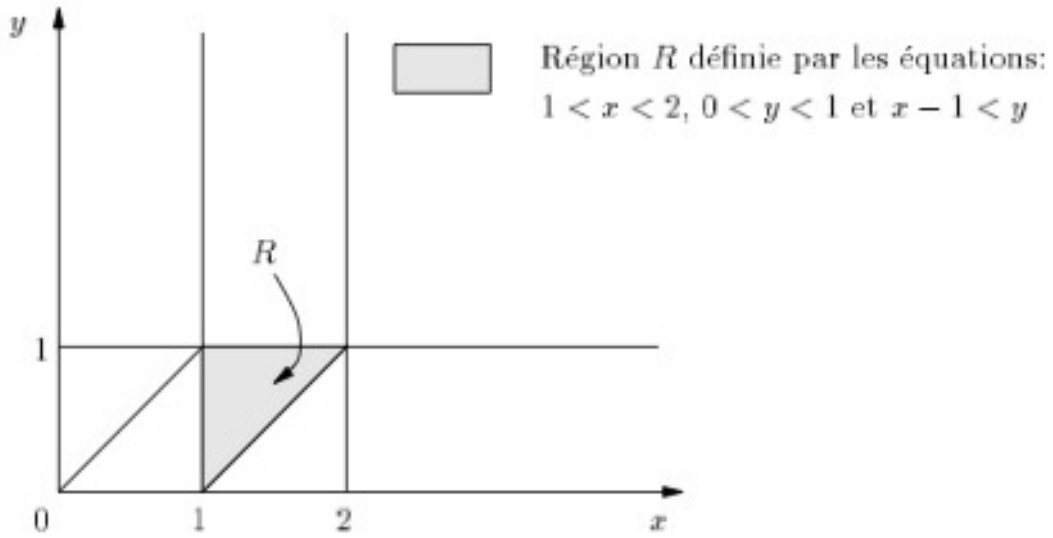
- **Une preuve (vision naïve) == un parcourt exhaustif de l'espace d'état**
  - Impossible avec les automates temporisés <= horloges
  - => on se ramène à un cas fini
- **Comment ça marche :**
  - Objectif : découper les « lieux » de l'automate en un ensemble fini d'ensembles d'états qui possèdent tous le même ensemble d'états futurs accessibles

préstabilité

  - Algorithme de construction :
    - Construction des régions puis fusion
    - Découper les lieux « en deux » jusqu'à valider l'objectif

La vision « naïve » du graphe des régions

- **Idée : Trouver une décomposition canonique des lieux vérifiant la « pré-stabilité »**
- **Mise en oeuvre :**
  - Fragmentation pour chaque lieu de l'espace d'horloge selon un quadrillage + des diagonales



Exercice sur la logique temporelle

On suppose un automate pour lequel on a défini les variables booléennes ready, running, completed, over-run
On suppose le thread dans l'état Ready initialement.

Ready identifie l'état d'un thread qui a été activé mais n'a pas démarré son exécution
Running identifie l'état d'un thread en cours d'exécution
Completed identifie l'état d'un thread attendant la prochaine activation
Over-run identifie l'état d'un thread ayant dépassé son échéance

Exprimez (attention ces formules ne sont parfois pas « implémentées »):
a) En l'absence d'overrun, le thread est toujours réactivé (retourne dans Ready)
b) Dès qu'un thread atteint l'état overrun, il y demeure indéfiniment
c) Un thread s'exécutant termine correctement son exécution ou est détecté comme étant en situation d'overrun.
d) Dès qu'un thread est activé, si il finit par terminer son exécution alors il ne peut pas y avoir eu d'overrun.

Logique temporelle – correction

- a) G (not overrun) => GF Ready
- b) G(overrrun => G overrun)
- c) G(running U (complete ou overrun))
- d) G(Ready => (
  F complete
  => (overrun U complete)
)

Logique CTL – raisonner sur un arbre ...

- Idée : raisonner sur les transitions franchissable (plutôt que sur un seul enchainement)
- Objet étudié : un état d'un automate + toutes ses « continuations » possibles (structure arborescente infinies)
- Exemple

Logique CTL – interprétation d'une formule

- A Fp