



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

- Spécification et vérification
- Enjeux du temps réel
- Extension des automates
- Expression des exigences
- Introduction à la mécanique de vérification



Spécification & vérification de modèles



Spécification : Exigences & modèle

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

- **Spécification de comportements :**
 - Sémantique : représentation d'une exécution (séquence de symboles représentant des évènements)
 - Syntaxe : ensemble d'éléments textuels ou graphiques « interprétables » en un ensemble d'exécutions

- **Exemple « UML » :**
 - Sequence diagrams (syntaxe),
 - Séquences d'appels de méthodes (sémantique).



Les différents modèles abstraits d'exécution

■ Les structures de base :

- Les séquences : crée un ordre total permettant de décrire la progression d'une exécution

Ex : séquences des valeurs d'un ensemble de variables

- Les ordres partiels : permet de décrire la notion de causalité et permet la prise en compte du parallélisme

Ex : les scénarios d'exécution des « sequence diagrams » avancés (composition ||)



Séquence d'états, traces et exécutions

- **Séquence d'états** : Soit Σ un ensemble d'états
 - Une séquence d'états représente chaque état séparé par le symbole '.'

$\sigma_1.\sigma_2.\dots.\sigma_n$

- **Trace** : Soit Σ un ensemble d'événements
 - Une trace est une séquence d'événements (instantanés)

$e_1.e_2.\dots.e_3$

- **Une exécution** : une séquence alternant les états et les événements (marquant les changements d'états)



Modèles vs Exigences

■ Un modèle ==

- Définit l'espace d'état du système ou son abstraction
- Définit sa « **mécanique d'interaction** » avec son **environnement** (poignée de main;;messages)
- ... Décrit l'implémentation du système en bref
Ex un state-chart UML, un subprogram en AADL

■ Une exigence ==

- Définit une contrainte relative à tout ou partie de l'état du système
- Décrit un besoin à satisfaire dans l'implémentation du système

■ 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 nécessaire / suffisantes.

- **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 dit de section critique dans chaque modèle.
 - Modèle du fonctionnement du sémaphore
- **Exigence :**
 - 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 de traces

- S = ensemble d'états
- $E = (S \times S)$ ensemble de transitions
- $L : E \rightarrow \Sigma$, fonction d'étiquetage des transitions

■ Une exécution == un chemin :

$(e_n)_{n \in \mathbb{N}} \Rightarrow$ deux visions: séquences finies/infinies

■ Types d'exigences et méthodes de vérification

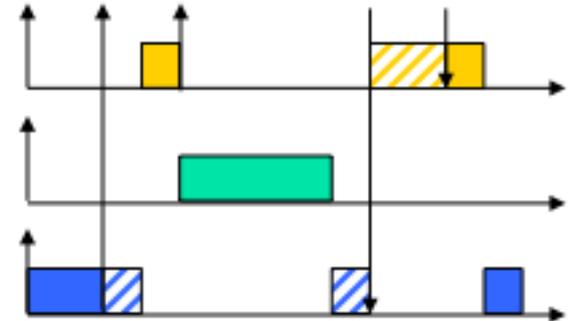
- Eviter des transitions ou états redoutés
- Atteindre des états ou franchir des transitions
- Invariant d'état / de topologie



Enjeux spécifiques aux Systèmes embarqués temps réels

■ 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
 - Vérification de modèles à transitions
- **Les autres vérification**
 - 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



Deux visions en opposition du « temps »

■ Temps physique – horloges

- Variation constante
- Dates totalement comparables (après normalisation)

■ Chronomètres

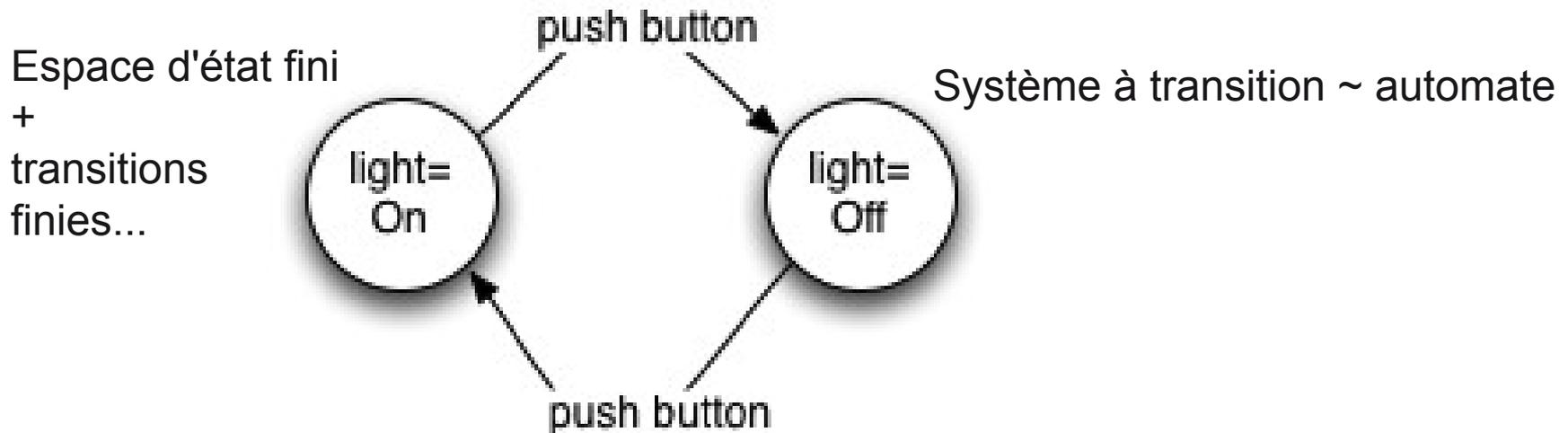
- Variation constante par morceau
- Peut être mis en pause
- Dates parfois non comparables (entre deux chronomètres par exemple)
- Peut être remis à zéro



Extension temporisée des automates



Vers des automates plus riches...



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



Des automates finis vers les automates temporisés (I)

■ Extensions de l'espace d'état

- Des états discrets (appelés désormais Location)
- Des variables (réels positifs) pour mesurer l'écoulement du temps
- Décomposition des états entre Discret / Continu.

■ Extension de la relation de transitions

- Abstraction des transitions entre états par regroupement : définition de règle de transitions
- Règles de transition entre Locations conditionnées par la valeur des horloges

Des automates finis

vers les automates temporisés (II)

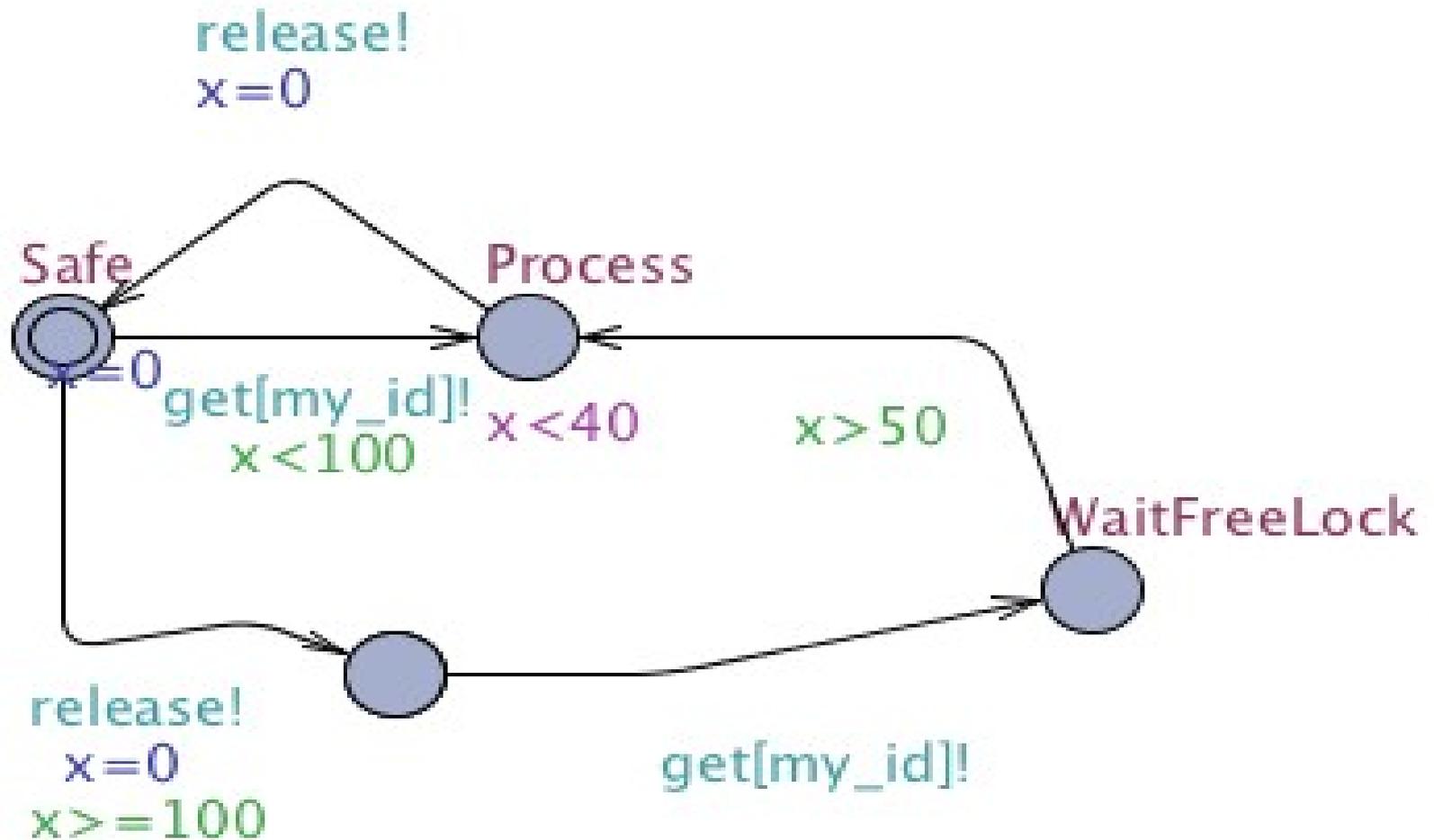
- **Pb: comment connaît-on le langage de trace d'un automate temporisé ?**
- **Le système à transition est à priori infini avec un nombre infini de transition**
 - **État :**
 - l : une location (un élément dans un ensemble fini)
 - u : une valuation des horloges (un vecteur de réels positifs)
 - **Les 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

■ Résultats basés langage:

- Stabilité de l'ensemble des TA par union, intersection
- Pas de stabilité par complément ...

■ Corolaire :

- L'inclusion du langage d'un automate temporisé dans un autre est indécidable

■ 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 et de complexité P-SPACE complet

- **Extension des états :**
 - Ajout de variables de type entier fini et tableaux
- **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 ?

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

$\Phi, \Psi ::= p \mid q \mid \dots \mid \text{true} \mid \text{false} \mid$ (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 \forall et \exists sur les variables qu'ils manipulent**



Interprétation des opérateurs temporels

- Supposons que la trace π soit constituée de la séquence $s_0.s_1.s_2.....s_k$
- Gp : « p est toujours vérifié dans le futur de l'exécution»
pour tout j s_j prouve p *Permet de prouver* Gp
- Fp : « p finit par être vrai dans le futur »
il existe $j \geq 0$, tel que s_j prouve p *Permet de prouver* Fp
- Xp : « p est vrai au prochain pas d'exécution »
 s_1 prouve p *Permet de prouver* Xp
- $p \text{ U } q$: « p est vrai tant que q ne l'est pas au moins une fois »
Il existe j , tel que pour tout $k < j$, s_k ne prouve pas q et s_j prouve q , et s_k prouve p *Permet de prouver* que $p \text{ U } q$

■ 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 $]10; +\infty[$

...

■ Pour les systèmes à transitions Observer :

Automates temporisés déterministes (dont on peut construire le complémentaire)



Vérification théorie et savoir faire

■ 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 incluses 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 **reconnait une trace**
=> contre exemple de « **TS valide F** »



Formalisation des propriétés « safety » et « liveness »

- **Une propriété de safety :: Je ne veux pas de ...**
 - $G \text{ not } (\dots)$ ou $G (\dots)$
- **Une propriété démontrant une progression :**
 - $A \text{ U } B$:: 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 :**
 - $G (E (\dots))$ ce cas est un peu à part ...



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

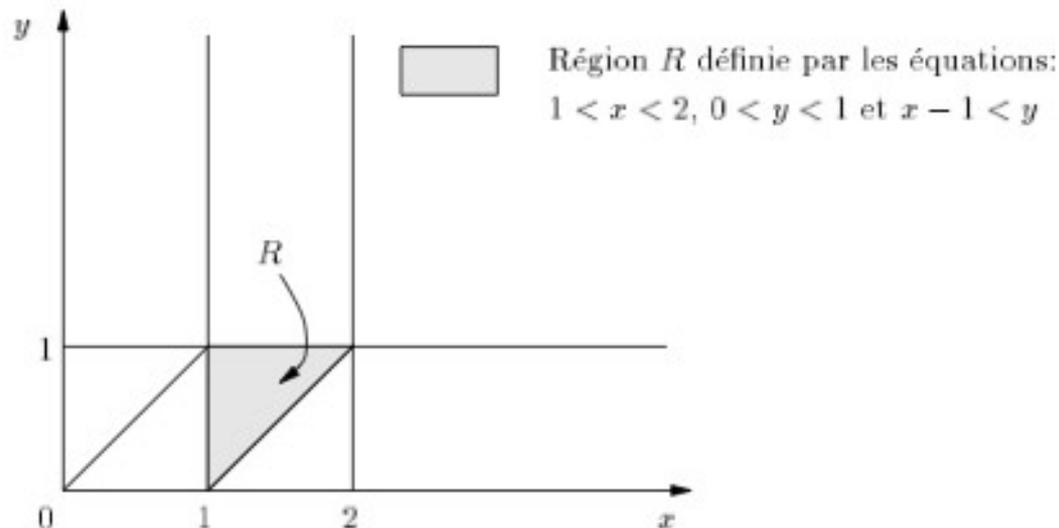
- **Une preuve (vision naïve) == un parcours exhaustif de l'espace d'état**
 - Impossible avec les automates temporisés \leq horloges
 - \Rightarrow on se ramène à un cas fini
- **Comment ça marche :**
 - Objectif : découper les « lieux » de l'automate en un ensemble fini d'ensembles d'états qui possèdent tous le même ensemble d'états futurs accessibles
 - Algorithme de construction :
 - Construction des régions puis fusion
 - Découper les lieux « en deux » jusqu'à valider l'objectif

préstabilité



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 est activé, il finit par terminer son exécution dans les temps.
- c) Dès qu'un thread atteint l'état overrun, il y demeure indéfiniment
- d) Le thread s'exécute jusqu'à ce qu'il ait complété son exécution