Séminaire MeFoSyLoMa **le 21 Mars 2008**

Timed Recursive ECATNets : Application to analysis of timed-constrained reconfigurable workflow processes

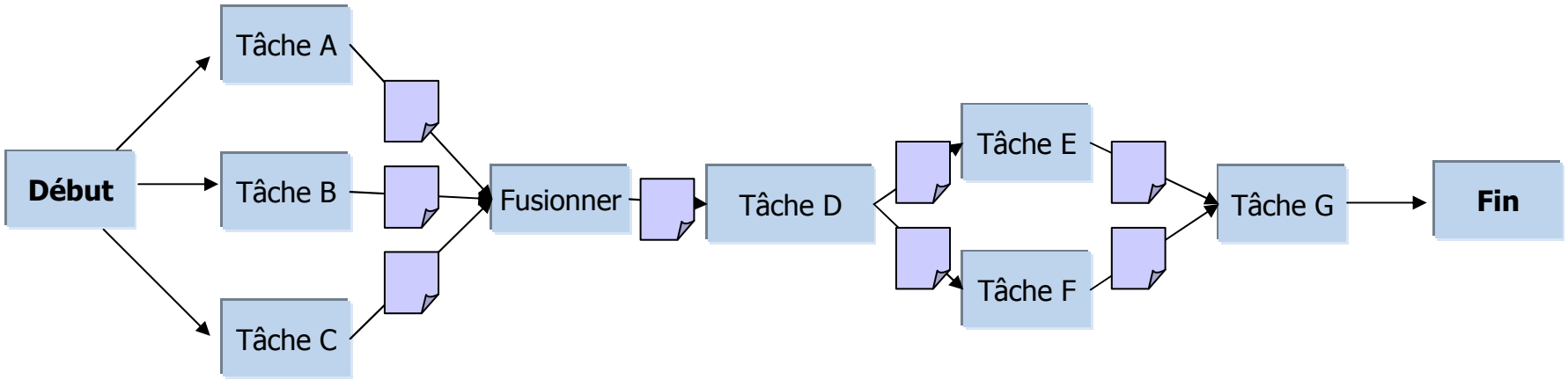
Awatef Hicheur
CEDRIC-CNAM, France

Plan

- Introduction
- Objectifs
- Les ECATNets récursifs temporisés (T-RECATNets)
- Les T-RECATNets pour la modélisation et l'analyse de processus workflows reconfigurables (WF-T-RECATNets)
- Simulation et analyse formelle des T-RECATNets par le système Maude
- Conclusion et travaux futurs

Introduction

- **Workflow**
  - Automatisation partielle ou totale de processus métiers
  - Transfert de données d'un participant à un autre
  - Suit un ensemble de règles procédurales



- **Schéma de workflow**

Déscription d'un processus en termes de :

- Tâches
- Relation de dépendence (condition) entre tâches
  - Logique
  - Temporelle

Problématique

- Les processus workflows sont:

✓ **Distribués** et **collaboratifs**
- exp. fabrication de plusieurs lots de produits sur différents sites en parallèle

✓ Influencés en grande partie par la **dimension temporelle**
  - délais de disponibilité d'une ressource
  - durée des tâches
  - deadlines

✓ Sujets à des **déviations fréquentes** de leurs comportements lors de l'occurence d'exceptions
  - indiponibilité d'une ressource
  - violation de deadlines
  - . . .

Objectifs

Un formalisme pour la modélisation des processus workflows permettant:

- La description de l'exécution **distribuée** et **coopérative** des processus
- La représentation des **caractéristiques temporelles** des processus
- La spécification de workflows à structures **reconfigurables** et **flexibles**
- La **vérification formelle** du comportement et des contraintes temporelles des processus

ECATNets récursifs Temporisés

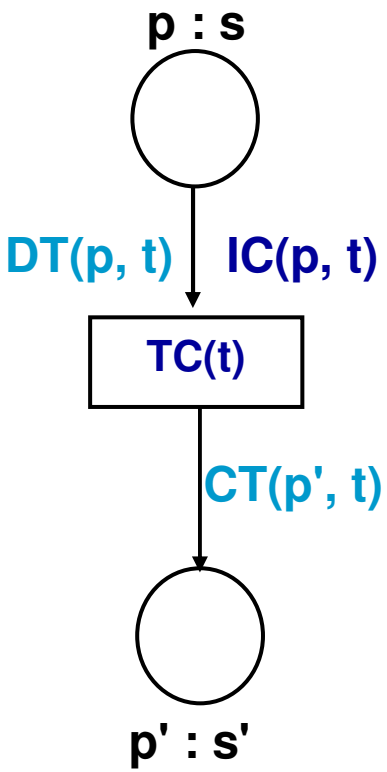
- Extension temporelle des ECATNets récursifs

- ECATNets récursifs (RECATNets)

- Combinaison **saine** des
  - ✓ ECATNets “Extended Concurrent Algebraic Term Nets “ (Bettaz & Maouche, 1992)
  - ✓ Réseaux de Petri récursifs (Haddad & Poitrenaud, 2007)

- Sémantique décrite en termes de **logique de réécriture**

RECATNets: syntaxe et sémentique



- Intégration de la **récursivité** dans les ECATNets «ordinaires»
- Deux types de transitions: **élémentaires** et **abstraites**

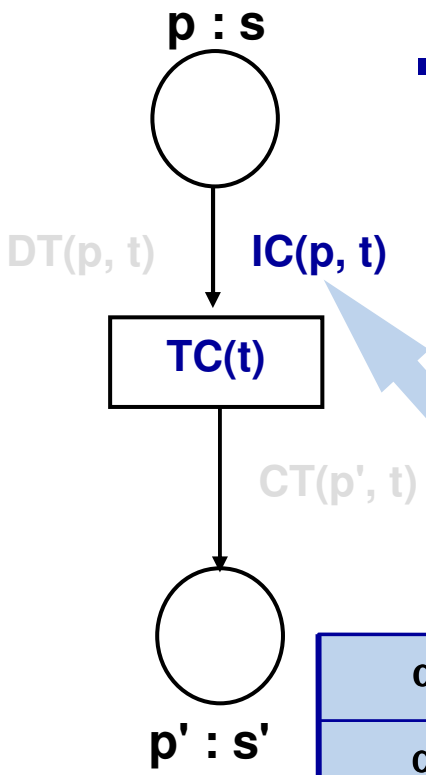
Chaque place *p* est associée à une sorte

IC(p, t) (Input Conditions)
DT(p, t) (Destroyed Tokens)
CT(p', t) (Created Tokens)

} Multi-ensemble de termes algébriques

TC(t) (Transition Conditions) : Terme booléen

RECATNets: syntaxe et sémentique



- Franchissment d'une **transition (élémentaire ou abstraite)** dans un réseau

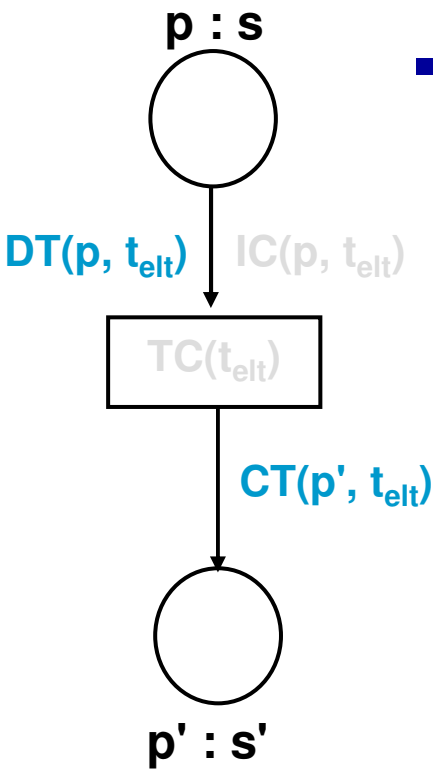
Condition :

- **IC(p, t)** est valide pour chaque place en entrée de la transition ***t***

- **TC(t)** est vraie

α^0	M(p) égale à α (cas particulier IC(p, t)=$\varnothing^0$)
α^+	α inclus dans M(p) (cas particulier IC(p, t)= $\varnothing^+$: condition toujours satisfaite)
α^-	α non inclus dans M(p)
$\alpha_1 \wedge \alpha_2$	Les deux conditions α_1 et α_2 sont évaluées à vraie
$\alpha 1 \vee \alpha 2$	α_1 ou α_2 est évaluée à vraie

RECATNets: syntaxe et sémentique



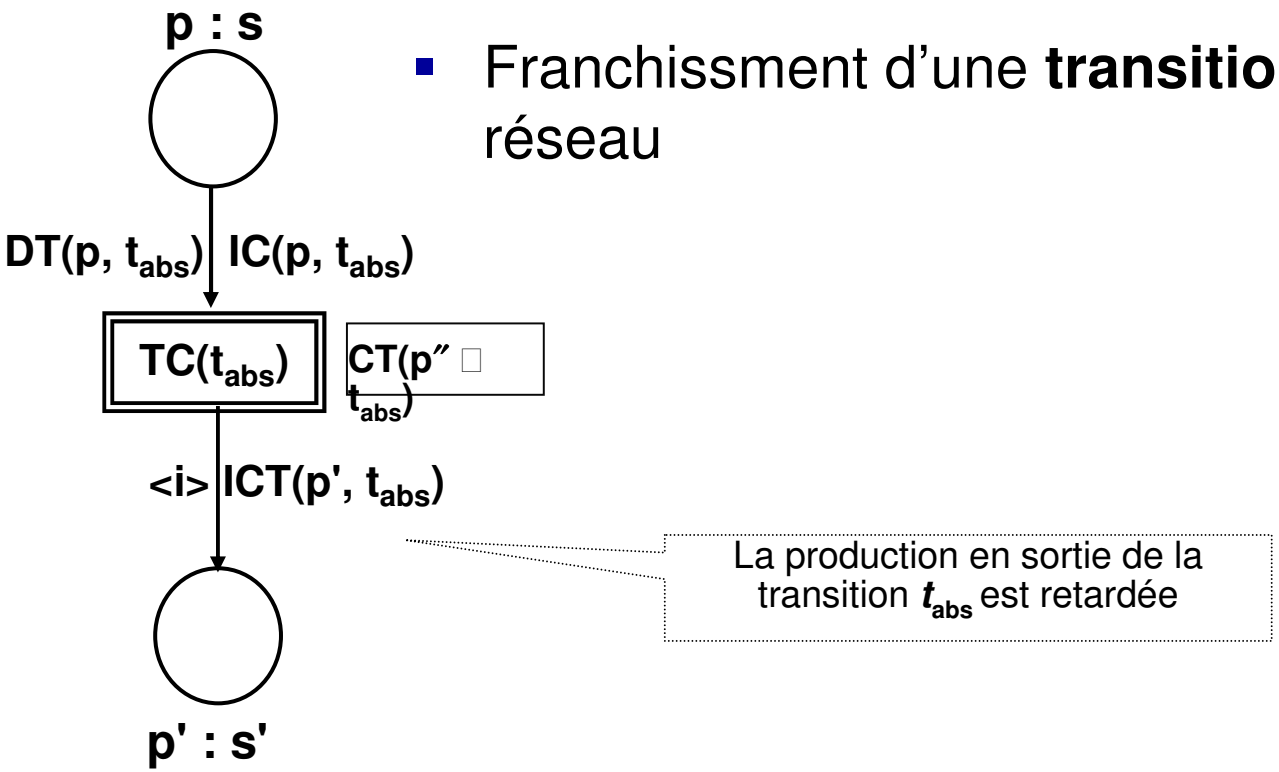
- Franchissment d'une **transition élémentaire** dans un réseau

Action :

- Le marquage **DT(p, t_{elt})** est soustrait de chaque place d'entrée ***p***

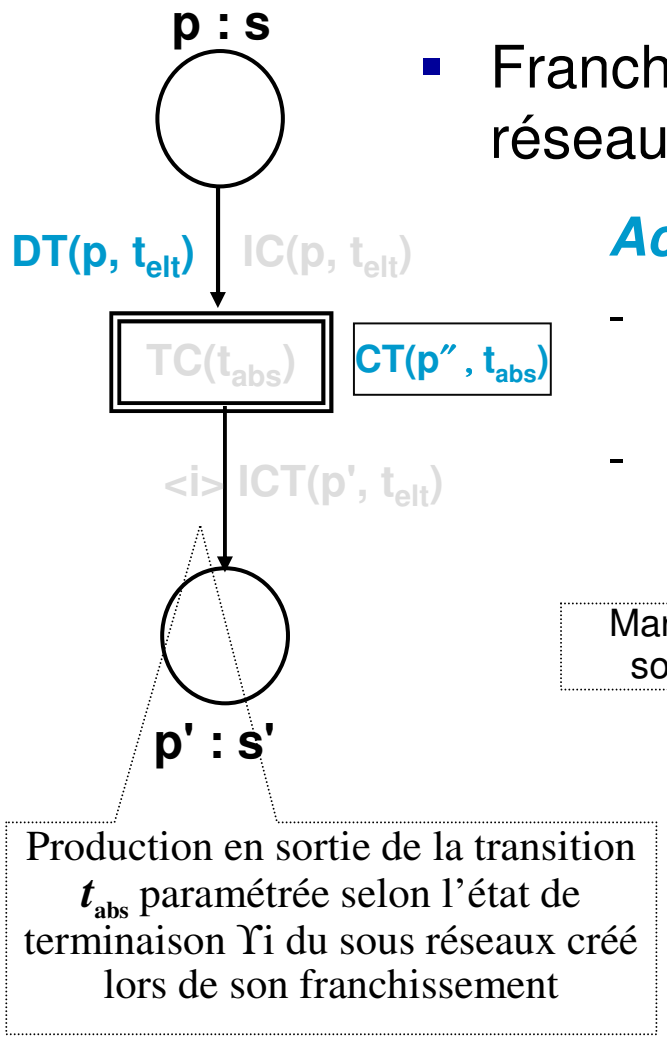
- Le marquage **CT(p', t_{elt})** est *créé* dans chaque place de sortie ***p'***

RECATNets: syntaxe et sémentique



- Franchissment d'une **transition abstraite** dans un réseau

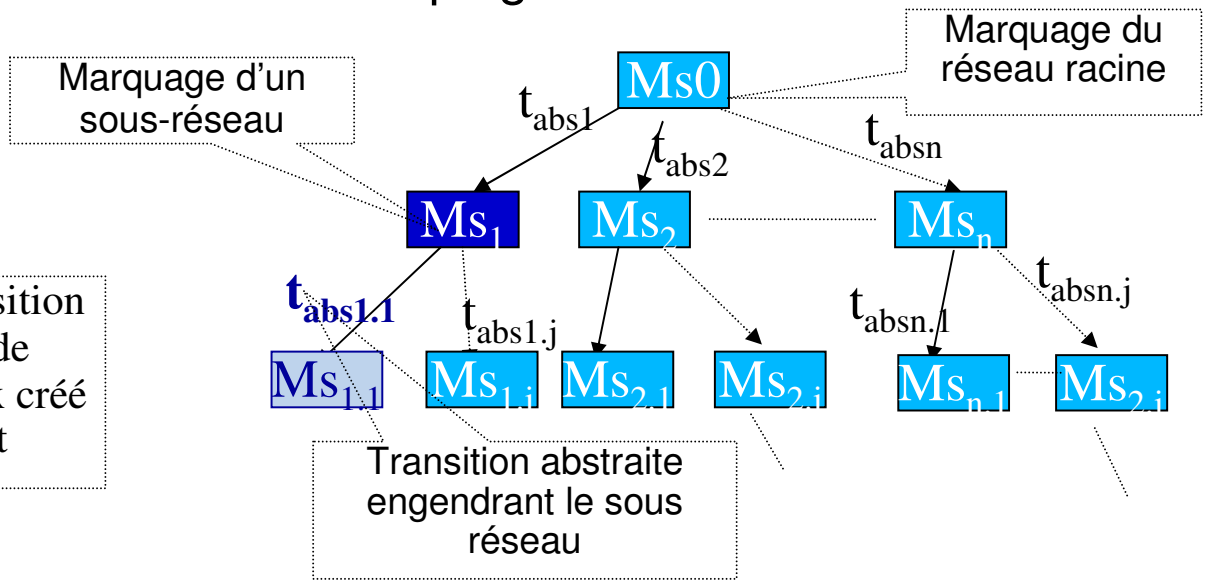
RECATNets: syntaxe et sémentique



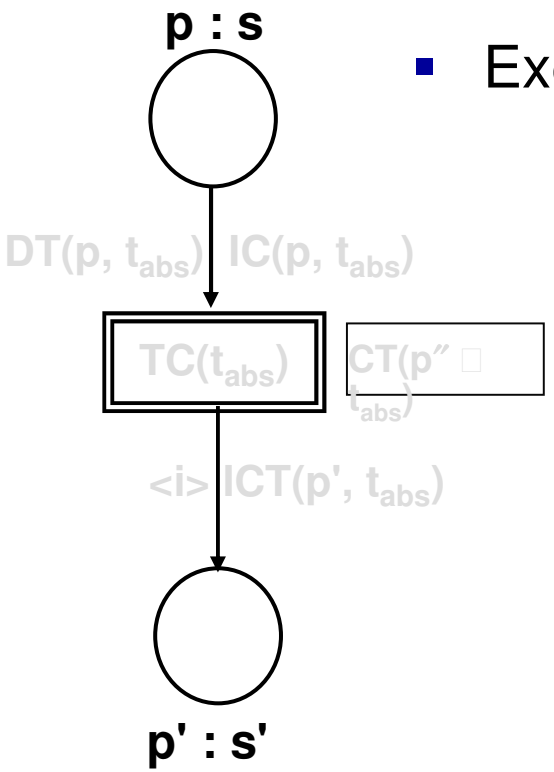
- Franchissment d'une **transition abstraite** dans un réseau

Action :

- Le marquage **DT(p, t_{abs})** est soustrait de chaque place d'entrée ***p***
- Création d'un sous-réseau fils avec **CT(p'', t_{abs})** comme marquage initial



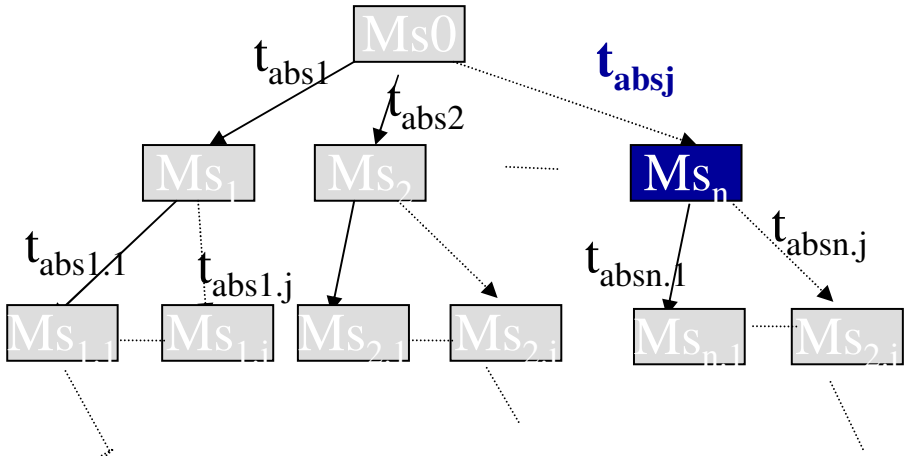
RECATNets: syntaxe et sémentique



- Execution d'une **étape de coupure** dans un réseau

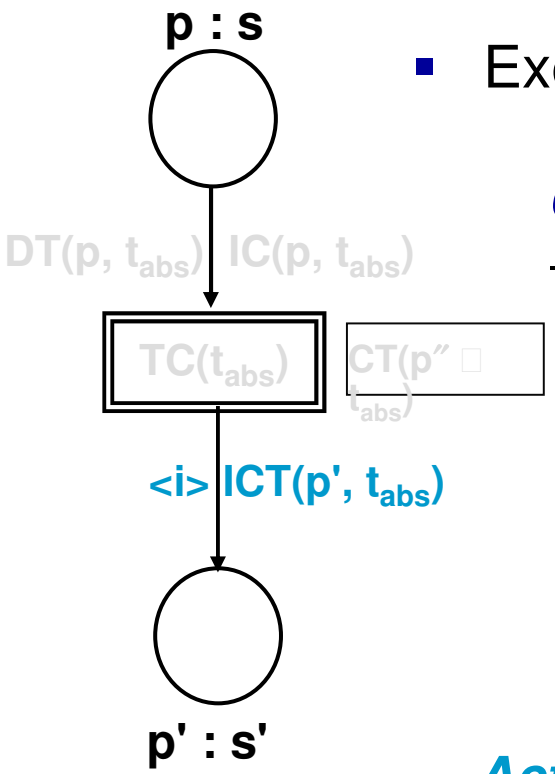
Condition :

- Un **marquage final** ϒi est atteint



Etat en cours du RECATNet

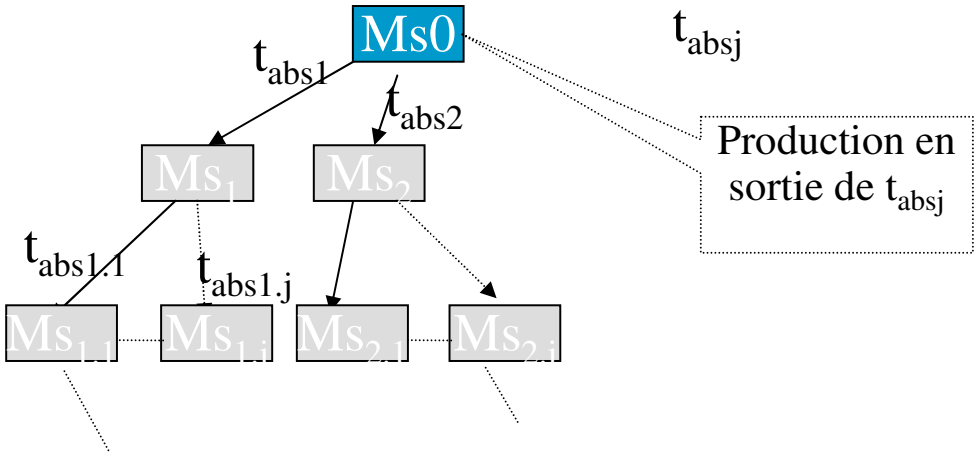
RECATNets: syntaxe et sémentique



- Execution d'une **étape de coupure** dans un réseau

Condition :

- Un **marquage final** est atteint



Action :

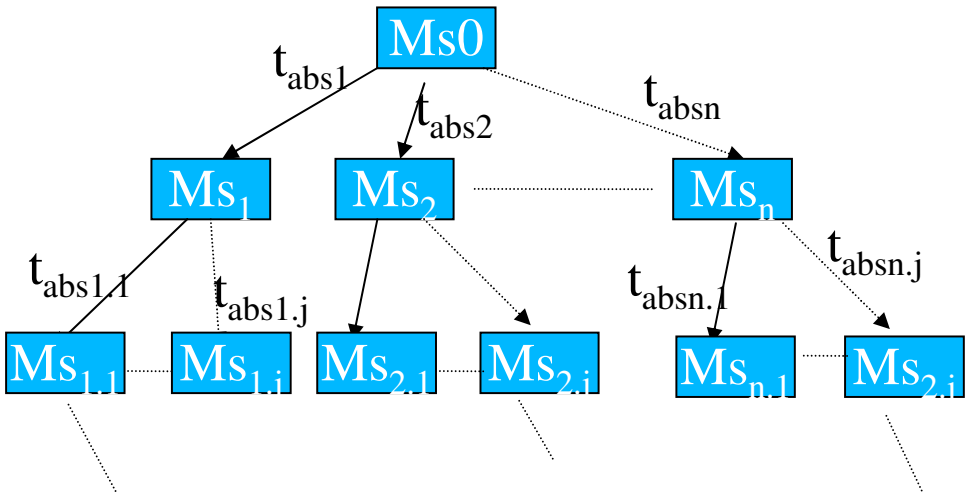
- Le réseau se **termine** et **supprime** tous ses sous-réseaux fils
- Création de **ICT (p',t_{abs}, i)** en place de sortie p' de la transition t_{abs} ayant engendré ce réseau

La sémantique des RECATNets en termes de logique de réécriture

Un RECATNet → **Une Théorie de réécriture**

État d'un RECATNet **axiomatisé par** → **Une Théorie équationnelle**

L'état courant d'un RECATNet *RN* est un **arbre** *Tr* de réseaux marqués **décrit par** → Un terme algébrique de la forme générale **récursive**



$[Ms_0, nullTrans,$
$[Ms_1, t_{abs1}, [Ms_{1.1}, t_{abs1.1}, \ldots] \ldots [Ms_{1.j}, t_{abs1.j}, \ldots]]$
$[Ms_2, t_{abs2}, [Ms_{2.1}, t_{abs2.1}, \ldots] \ldots [Ms_{2.j}, t_{abs2.j}, \ldots]] \ldots$
$[Ms_n, t_{absn}, [Ms_{n.1}, t_{absn.1}, \ldots] \ldots [Ms_{n.j}, t_{absn.j}, \ldots]]]$

Le réseau racine n'est engendré par aucune transition

La sémantique des RECATNets en termes de logique de réécriture

Un RECATNet → **Une Théorie de réécriture**

Franchissement d'une **transition** (abstraite ou élémentaire) ou **exécution** d'une étape de coupure → **Réécriture** modulo ACI avec la **règle** de réécriture correspondante

Exp. transition abstraite $\mathbf{t_{abs}}$ → Règle de réécriture (***abstract rule***) (t_{abs} : Th -> Th') avec Th, Th' de sorte *Thread* de la forme

```
crl [t_absi]:[M ⊗ <p, mp ⊕ DT(p, t) >, T, mTh] →
      [M ⊗ <p, mp>, T, mTh [<p″,CT(p″,t_absi)> ⊗ <p_1, Ems> ⊗...⊗ <p_n, Ems>,
      t_absi, nullThread]] if [(InputCond) and TC(t)] .
```

Propriété d'**atteignabilité** →

$$InputCond = \begin{cases} mp \equiv \alpha \text{ if } IC(p,t) = [\alpha]^0 \\ \alpha \textbf{ Inclu } mp \text{ if } IC(p,t) \equiv [\alpha]^+ \\ \text{not } (\beta \textbf{ Inclu } mp) \text{ if } IC(p,t) \equiv [\beta]^- \end{cases}$$

Introduction du temps dans les réseaux de Petri

- Plusieurs façons de représenter le temps dans les réseaux de Petri

  - Délais de sensibilisation
  - Durée de franchissement
  - Durée de disponibilité des jetons

- Le choix d'une valeur de temps associée à un événement

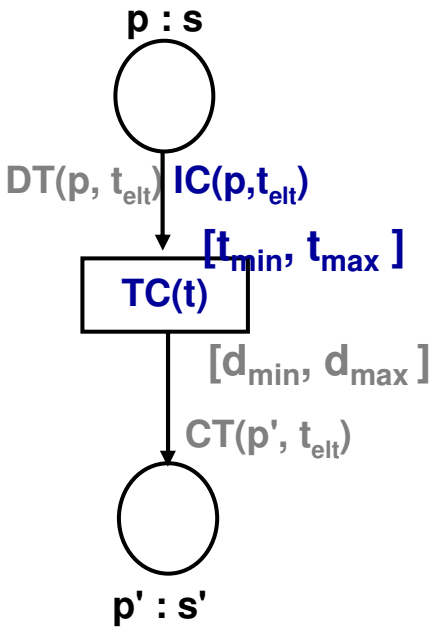
  - Déterministe (durée fixe)
  - Intervalle de temps
  - Stochastique

ECATNets récursif Temporisés (T-RECATNets)

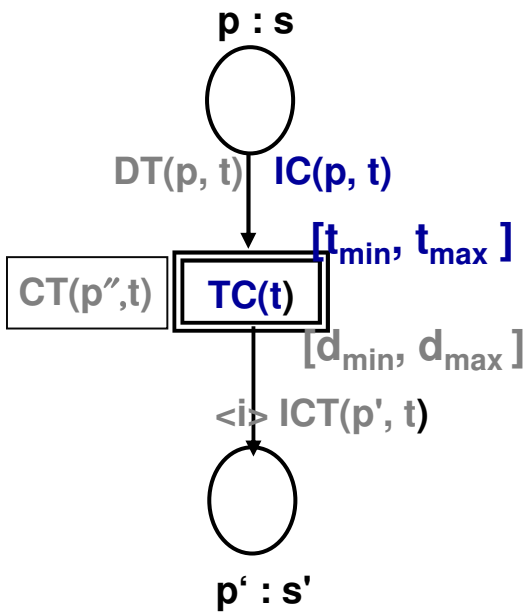
❖ Temps sur la **condition**

Intervalles de temps pour le **franchissement** des transitions (Merlin, 74)

- A partir de l'instant de la sensibilisation de la transition



Transition élémentaire



Transition abstraite

➢ La transition est **franchissable** si elle est *sensibilisée* d'une manière continue jusqu'à ce que la borne **min** de cette intervalle soit écoulée

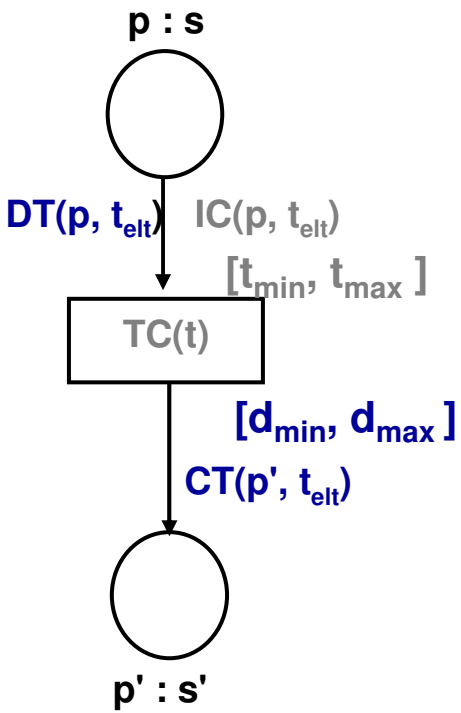
➢ La transition **doit être franchie** si elle est toujours *sensibilisée* lorsque la borne **max** est atteinte

- Première phase du tir de la transition *t* notée - *t*

ECATNets récursif Temporisés (T-RECATNets)

❖ Temps sur l'**action**

Durée de production associées aux **transitions élémentaires** (Aalst, 92)



Transition élémentaire

- Lors du franchissement d'une transition élémentaire

➢ **DT(p, t_{elt})** est immédiatement détruit de la place d'entrée

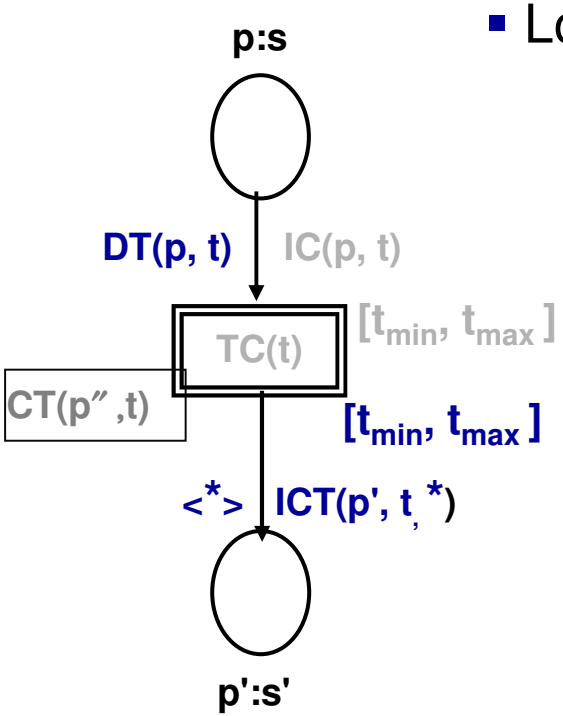
➢ Les jetons **CT(p', t_{elt})** sont produits en place de sortie qu'après un temps ***t*** écoulé dans l'intervalle **[d_{min}, d_{max}]**

- Deuxième phase du tir de t_{elt} notée + t_{elt}

ECATNets récursif Temporisés (T-RECATNets)

❖ Temps sur l'**action**

Durée pour les **processus** générés par les **transitions abstraites**



Transition abstraite

- Lors du franchissement d'une transition abstraite

  ➢ Suppression immédiate de **DT(p, t)** de la place d'entrée et création d'un nouveau processus fils

  ➢ Le processus fils se termine si un **marquage final** γi est atteint ➡ exécution d'une **étape de coupure** d'indice ***<i>***

  ➢ La terminaison du processus fils est nécessaire si sa **durée** est dans l'intervalle **[t_{min}, t_{max}]** ➡ exécution d'une **étape de coupure** d'indice **<*>**

- Deuxième phase du tir de t_{abs} notée $+ t_{abs}$

La sémantique des T-RECATNets en termes de logique de réécriture

Un T-RECATNet → **Une Théorie de réécriture temps réel**

État d'un T-RECATNet **décrit par** **Une spécification équationnelle**

Transition (absraite ou élémentaire) ou **étape de coupure** **décrit par** **Régles de réécriture** conditionnelle (instantanées)

Progression du temps → Une **régle de réécriture temporisée**

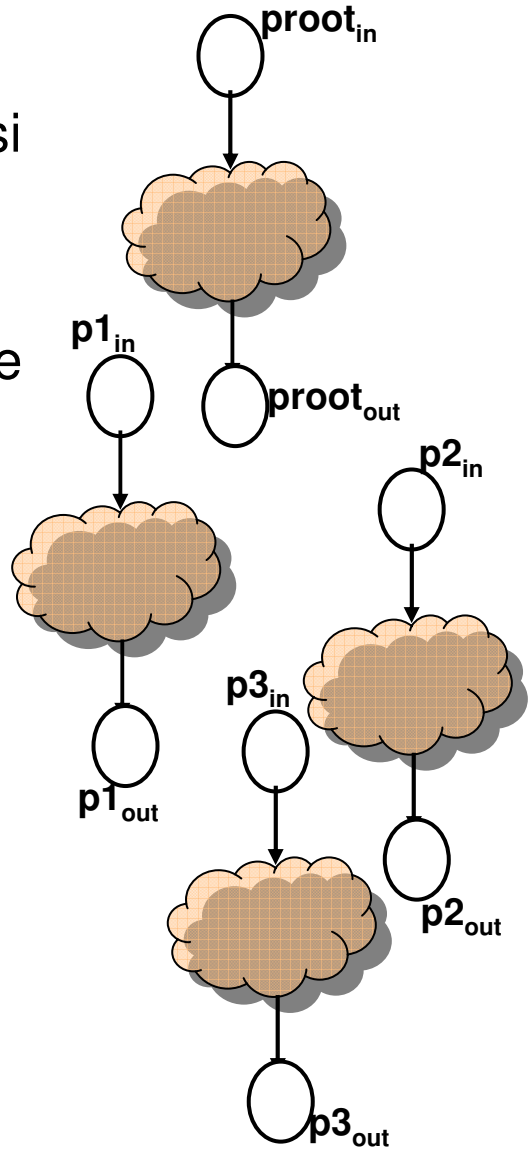
```
crl [tick]:{[ TM | nullTrans | Th ]}=>{delta([ TM | nullTrans | Th ], R)}
in time R if (R <= mte([ TM | nullTrans | Th ])) and
not eagerEnabled([ TM | nullTrans | Th ]) )
```

ECATNets récursif Temporisés pour la modélisation de workflows (WF-T-RECATNets)

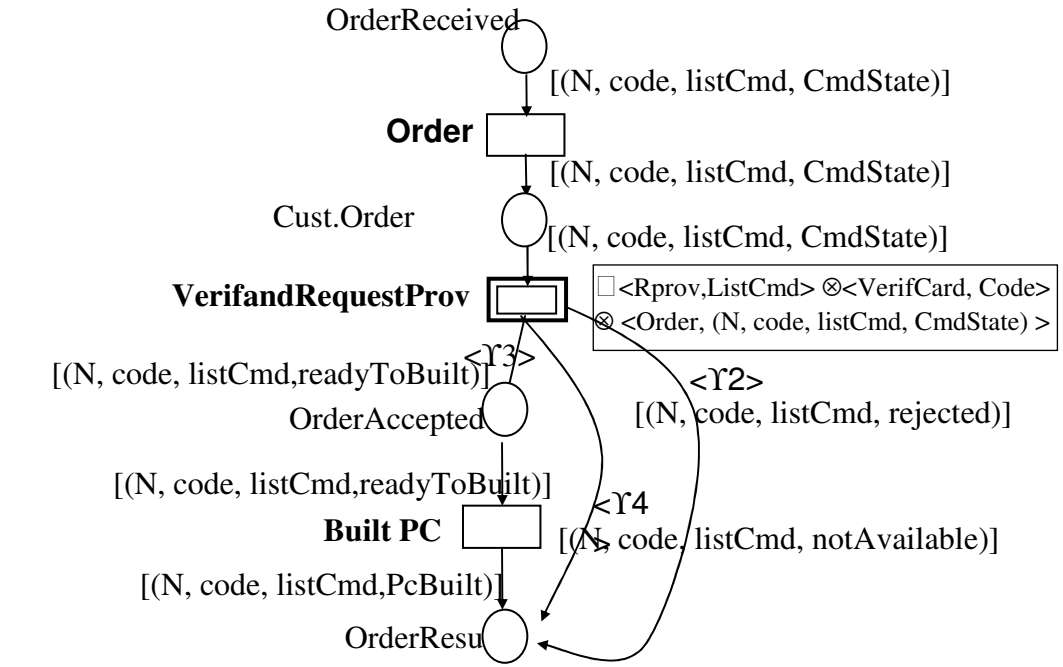
❖ Un T-RECATNet *N* est un **WF-T-RECATNet** si et ssi

- *N* dispose d' un ensemble de place d'entrée P_{in} et de place de sortie P_{out}

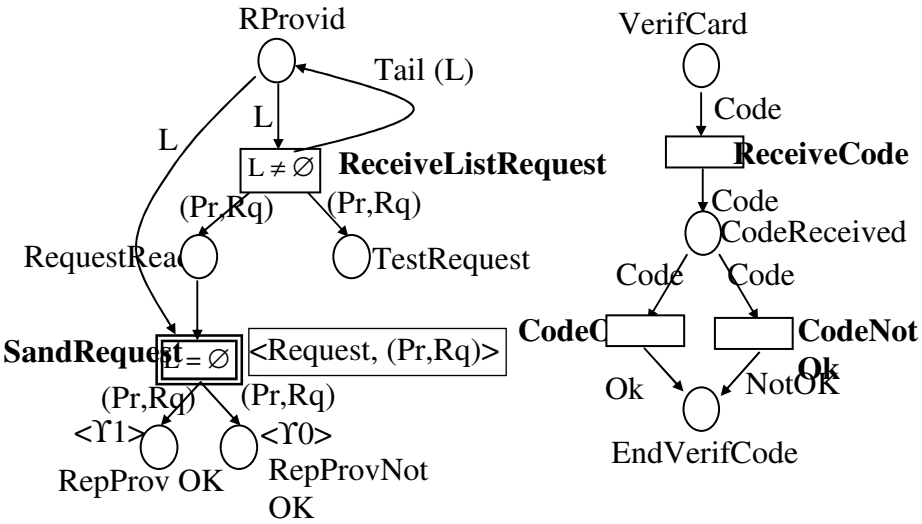
- Pour chaque composant $x \in P \cup T$ (*P* ensemble des places de *N*, et *T* ensemble des transitions de *N*), il existe un chemin d'une place *p* à une place *p'* et passant par *x* tel que $p \in P_{in}$ et $p' \in P_{out}$

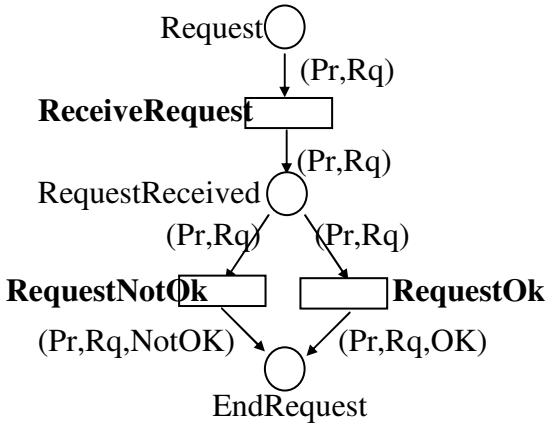


Exemple : « online shopping workflow »
Modélisation à l'aide des RECATNets



$\Upsilon 0 : (Pr, Rq, NotOk) \subseteq M(P_{EndRequest})$
$\Upsilon 1 : (Pr, Rq, Ok) \subseteq M(P_{EndRequest})$
$\Upsilon 2 : M(P_{EndVerifCode}) = NotOk$
$\Upsilon 3 : [M(P_{EndVerifCode}) = Ok\ ] \wedge [M(P_{RepProvOk}) = M(P_{TestRequest})]$
$\Upsilon 4 : M(P_{RepProvNotOk}) \neq \emptyset$





Simulation et analyse formelle des WF-T-RECATNets par l'outil Real Time MAUDE

- Plate-forme utilisée : version 2.3. de Real Time Maude (SRI International) sous Linux

- *Environnement* et *langage* basé sur la **logique de réécriture** supportant la spécification de systèmes temps réels

  - ✓ simulation (prototypage rapide)
  - ✓ vérification formelle des propriétés
    - Analyse d'accessibilité
    - (time-bounded) LTL model checking

Simulation et analyse formelle des WF-T-RECATNets par l'outil Real Time MAUDE

✘ La procédure d'analyse de Real Time Maude est semi décidable dans le cas des systèmes temporisés adoptant un temps continu

⟹ Les états couverts par l'analyse dépendent de la stratégie adoptée pour faire avancer le temps dans le système

⟹ La complétude n'est assurée que sous certains critères

- La durée de la sensibilisation des transitions et de leur franchissement dans un WF-T-RECATNets doit être fixe (borne min = borne max)

Simulation et analyse formelle des WF-T-RECATNets par l'outil Real Time MAUDE

➡ Calcule du graphe des classes d'états

- Regrouper des états en classe d'équivalence

- Représentation d'une classe d'états d'un T-RECATNet
  $C = (Tr, D)$

  - *Tr* l'état du T-RECATNet (arbre des processus)

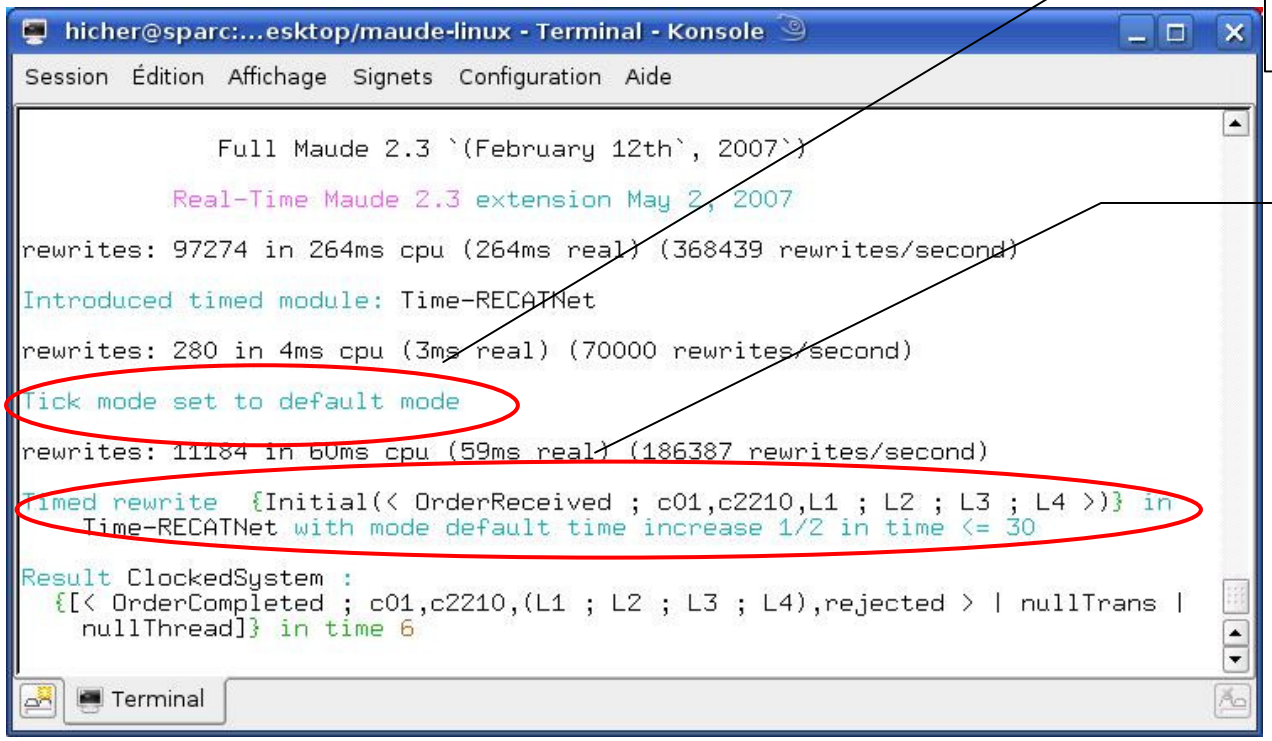
  - *D* la réunion de tous les domaines de tir des états de *C*

  - Les variables des systèmes d'inégalités de *D* sont de la forme *v.t* où
    - *t* de la forme - *t'* ou *+t'* avec $t' \in T_{abs} \cup T_{elt}$
    - *v* un noeud de l'arbre *Tr*

Simulation sous Real Time maude

- Plusieurs modes d'exécution selon la stratégie adoptée pour la progression du temps

  - ✓ proposés par défaut dans Maude (`trew`, `tfrew`)
  - ✓ définis par l'utilisateur à l'aide de stratégies de réécriture



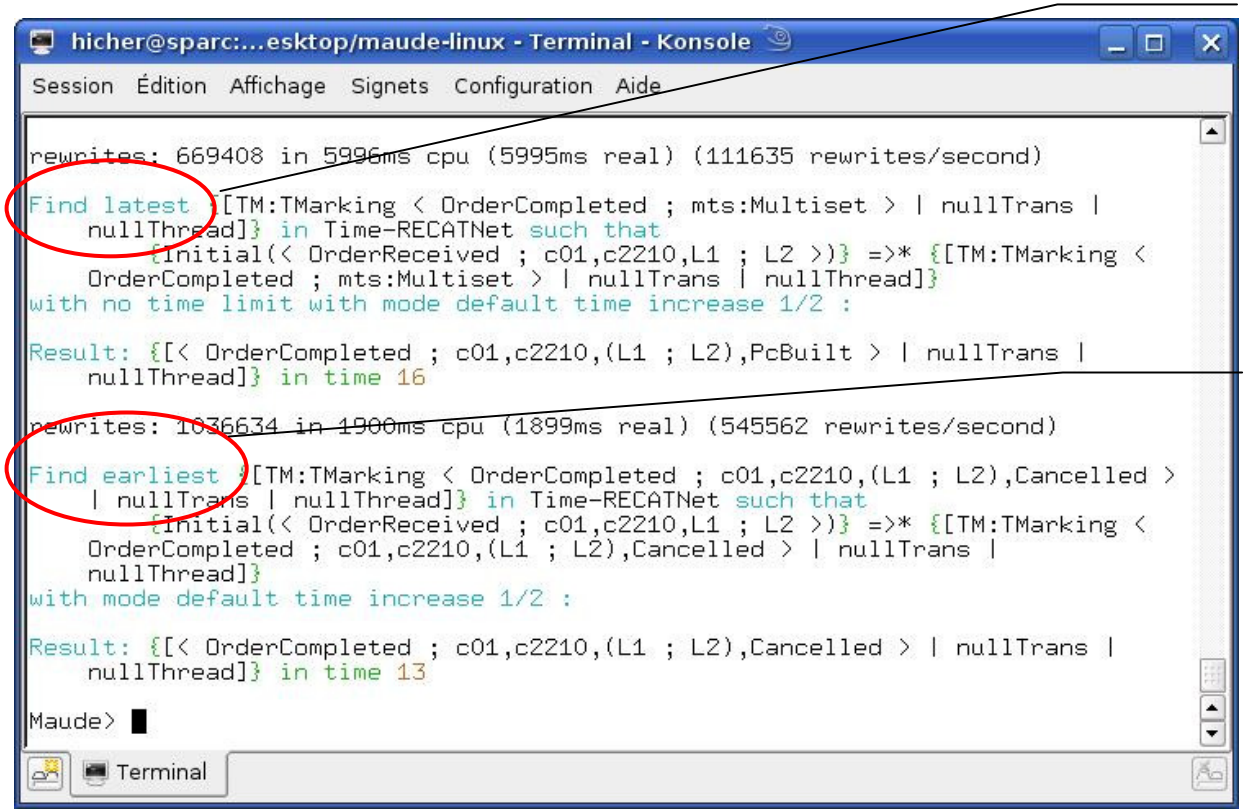
Choix de la stratégie de progression du temps

Lancer une exécution avec sans restriction sur le nombre de règles appliquées dans un temps inférieur à 30 u.t

Analyse d'accessibilité sous Real Time maude

- Calculer le temps le plus long et le temps le plus court pour atteindre l'état final d'un WF-T-RECATNet

- L'état final se réduit à la racine de l'arbre des réseaux avec un seule jeton en place finale



Rechercher le temps le plus long pour atteindre l'état final

Rechercher le temps le plus court pour atteindre l'état final

Conclusion et travaux futurs

Contribution

- Proposer une méthode basée sur les T-RECATNets pour la spécification de workflows reconfigurables à contraintes temporelles

➢ Spécifier d'une manière concise la flexibilité dans les processus workflows

➢ Décrire fidèlement l'exécution distribuée, concurrente et collaborative des processus

➢ Bénéficier d'un environnement d'exécution et d'analyse pour les T-RECATNets basé sur le système Maude: simulation, analyse d'accessibilité et vérification par model checking

- **Travaux en cours**

➢ Vérification de la **cohérence** des WF-T-RECATNets sous contraintes temporelles et de ressources

Merci de votre attention !
Questions ?

Bibliographie

- [BeMa92] M. Bettaz, M. Maouche « *How to specify non determinism and true concurrency with algebraic terms nets* », in LNCS, No 655, Springer-Verlag, 1992, pp. 164-180
- [HaPo07] Haddad S. and D. Poitrenaud, 2007. « *Recursive Petri nets: Theory and application to discrete event systems* ». In Acta Informatica. Springer Berlin / Heidelberg (Eds.).Vol 40 (7-8), 463-508.
- [BrMe06] Bruni R. and J. Meseguer, 2006. « Semantic foundations for generalized rewrite theories ». In Theoretical computer science, Vol 360, No 1-3, 386-414.
- [OLMe07] Ölveczky P. and J.Meseguer , 2007. « Semantics and pragmatics of Real-Time Maude ». In Higher-Order and Symbolic Computation. Vol 20 , 161-196.

Structure d'état d'un T-RECATNet

La structure de l'état **distribué** des T-RECATNets est axiomatisée par la théorie équationelle suivante (syntaxe Maude)

```
fmod THREAD is Protecting  MARKING . Sorts  Thread Trans TMarking .
op  nullTrans : -> Trans .
op  nullThread : -> Thread . ***** Constante qui implemente le thread et l'arbre vide (⊥)
*********************************************************
op CClock : Trans SuperTime -> TimerTrans [ctor] .
op CClock : Trans Time Time -> TimerTrans [ctor] .
op AClock : Marking Time -> TimerMarking [ctor frozen (1)] .
op ProcClock : Trans Time -> TimerProc [ctor] .
*********************************************************
op [_,_,_] Marking Trans Thread -> Thread .
op _ _: Thread Thread -> Thread [comm. Assoc id: nullThread].
*********************************************************************
****  Effet de l'écoulement du temps sur les marquages ********************
op delta : TMarking Time -> TMarking [frozen (1)] .
op delta : Thread Time -> Thread [frozen (1)] .
****** calcule du temps résdiduel dans les horloges  ***********************
op mte : TMarking -> TimeInf [frozen (1)] .
op mte : Thread -> TimeInf [frozen (1)] .
...
endfm.
```

RECATNets : Règles de réécriture (1/2)

Notation : ⊗ : union multi-ensembliste sur les pairs <p, $[m]_{\oplus}$> (ACI axioms)

Une transition t élémentaire est décrite par une règle de réécriture (*elementary rule*) (t_{elt} : M → M') avec M, M' de sorte *Marking*

crl [t_{elt}]: <p, mp ⊕ DT(p, t)> ⊗ <p′, mp′ > → <p, mp> ⊗ <p′, mp′ ⊕ CT(p′, t)> if (Nbr(mp′ ⊕ CT(p′,t))≤Cap(p′)) and (InputCond) and (TC(t)).

- **Une transition t abstraite est décrite par une règle de réécriture (*abstract rule*) (t_{abs} : Th → Th') avec Th, Th' de sorte *Thread***

crl [t_{absi}]: [M ⊗ <p, mp ⊕ DT(p, t) >, T, mTh] → [M ⊗ <p, mp>, T, mTh [<p″,CT(p″,t_{absi})> ⊗ <p_1, Ems> ⊗...⊗ <p_n, Ems>, t_{absi}, nullThread]] if [(InputCond) and TC(t)→ True] .

RECATNets : Règles de réécriture (2/2)

- **Une étape de coupure τ est décrite par une règle de réécriture (*Pruning rule*) (τi : Th → Th') avec Th, Th' de sorte *Thread***

- **Si l'étape de coupure est associée à un réseau différent de la racine de l'arbre Tr**

```
crl [τi]: [Mf ⊗ <p′, mp′>, Tf,[M ⊗ <pfinal, mpfinal>,tabsj,
mTh] mThf]→[Mf ⊗ <p′, mp′ ⊕ ICT(p′,tabsj,i)>, Tf, mThf] if (ϒi)

and (Nbr (mp′ ⊕ ICT(p′, tabsj, i)) ≤ Cap(p′)).
```

- **Si l'étape de coupure est associée au réseau racine de l'arbre Tr**

```
crl [τi]:[M ⊗<pfinal, mpfinal>, nullTrans, mTh] → nullThread

if (ϒi).
```

T-RECATNets : Règles de réécriture

Une transition t élémentaire est décrite par deux règles de réécriture (*elementary rule*) (t_{elt} : M → M') avec M, M' de sorte *TMarking*

Sensibilisation

crl [t_{elt}Enabled]: `<p, mp ⊕ DT(p, t)>` **CClock(t_{elt}, NoTimer)** → `<p, mp ⊕ DT(p, t)>` **CClock(t_{elt}, tmin, tmax)** **`if`** `(InputCond)` **`and`** `(TC(t)).`

Franchissement

crl [t_{elt}Fire]: `<p, mp ⊕ DT(p, t)>` **CClock(t_{elt}, 0, R)** → `<p, mp>`

`AClock``(<p′, mp′ ⊕ CT(p′, t)>,` **`dmin`**`,` **`dmax`**`)` **CClock(t_{elt}, NoTimer)**

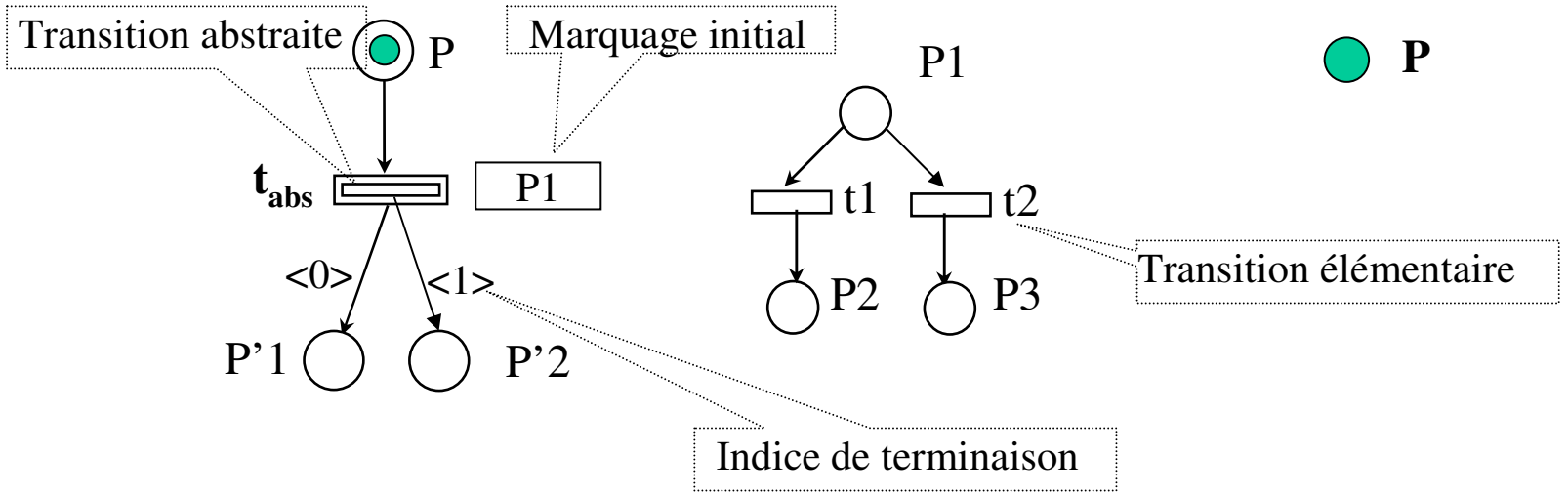
`if` `(InputCond)` **`and`** `(TC(t)).`

Production d'un marquage

`[unDelay] : Aclock(M, 0, R)` → **M**

Exemple illustratif

Etat du réseau : Arbre de processus



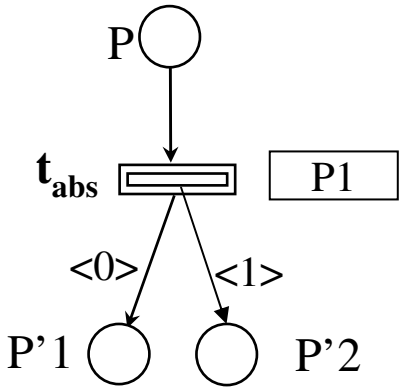
Marquages finaux :

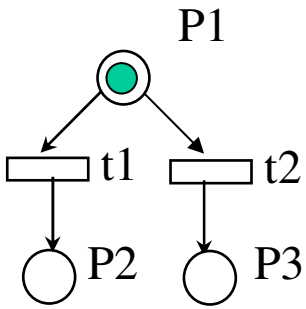
$\Upsilon 0$: M(P3)>0

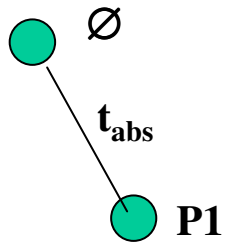
$\Upsilon 1$: M(P2)>0

Exemple illustratif (suite)

Etat du réseau : Arbre des processus



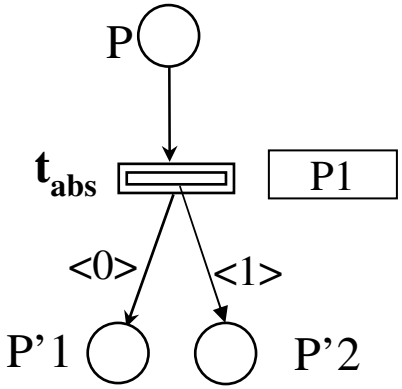


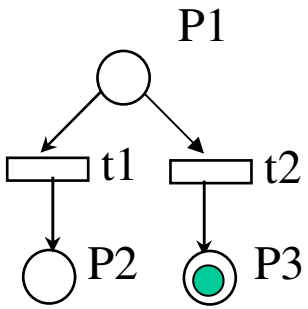


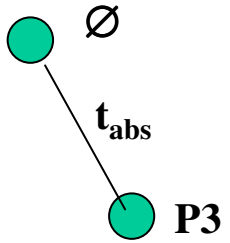
Marquages finaux :
ϒ0 : M(P3)>0
ϒ1 : M(P2)>0

Exemple illustratif (suite)

Etat du réseau : Arbre des processus



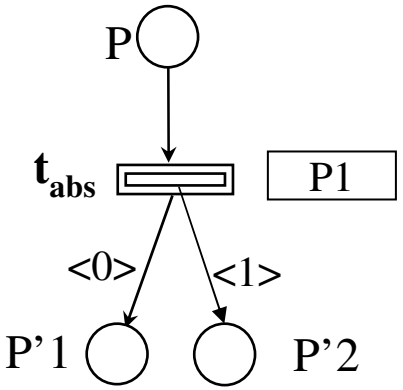


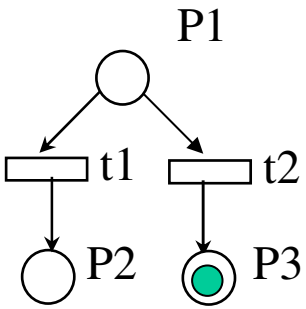


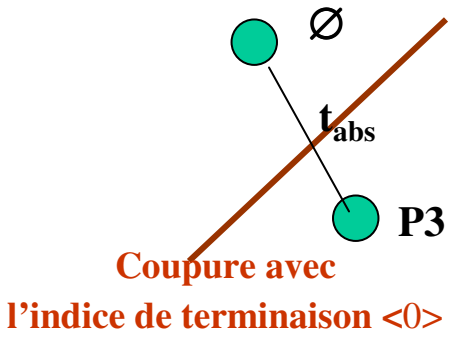
Marquages finaux :
ϒ0 : M(P3)>0
ϒ1 : M(P2)>0

Exemple illustratif (suite)

Etat du réseau : Arbre des processus





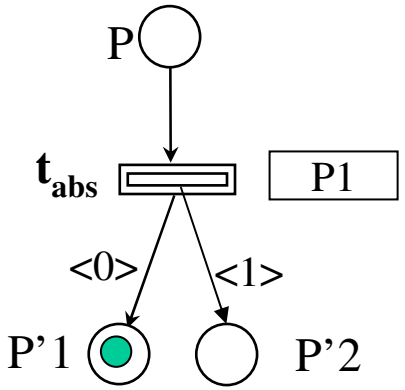


Marquages finaux :

ϒ0 : M(P3)>0

ϒ1 : M(P2)>0

Exemple illustratif (suite)



Etat du réseau : Arbre des processus

Marquages finaux :
ϒ0 : M(P3)>0
ϒ1 : M(P2)>0