Chapitre 1

Réseaux de Petri temporels : méthodes d'analyse et vérification avec TINA

1.1. Introduction

Parmi les techniques proposées pour spécifier et vérifier des systèmes dans lesquels le temps apparaît comme paramètre, deux sont largement utilisées : Les Automates Temporisés (voir le chapitre 4 de ce volume, consacré à ce modèle) et les Réseaux de Petri Temporels (ou *Time Petri Nets*), introduits dans [MER 74].

Les réseaux temporels sont obtenus depuis les réseaux de Petri en associant deux dates min et max à chaque transition. Supposons que t soit devenue sensibilisée pour la dernière fois à la date θ, alors t ne peut être tirée avant la date $\theta + min$ et doit l'être au plus tard à la date $\theta + max$, sauf si le tir d'une autre transition a désensibilisé t avant que celle-ci ne soit tirée. Le tir des transitions est de durée nulle. Les réseaux temporels expriment nativement des spécifications «en délais». En explicitant débuts et fins d'actions, ils peuvent aussi exprimer des spécifications «en durées». Leur domaine d'application est donc large.

Nous proposons dans ce chapitre un panorama des méthodes d'analyse disponibles pour les réseaux temporels et discutons de leur mise en œuvre. Ces méthodes, basées sur la technique des classes d'états, ont été initiées par [BER 83 ; 91]. Ces graphes constituent des abstractions finies du comportement des réseaux temporels bornés. Différentes abstractions ont été définies dans [BER 83 ; 01 ; 03b], préservant diverses

Chapitre rédigé par Bernard BERTHOMIEU et François VERNADAT.

classes de propriétés. Dans ce chapitre, nous discuterons de plus du problème pratique de la vérification de propriétés (*model-checking*) de certaines logiques sur les graphes de classes construits. La mise en œuvre de ces techniques nécessite des outils logiciels, tant pour la construction des abstractions requises que pour leur vérification elle-même. Les exemples discutés dans cet article ont été traités avec les outils de l'environnement `Tina` [BER 04].

Les concepts de base des réseaux temporels sont rappelés section 1.2. Les sections 1.3 à 1.5 introduisent les constructions de graphes de classes, fournissant des abstractions finies de l'espace infini des états des réseaux temporels. Les sections 1.3 et 1.4 présentent les constructions préservant les propriétés des logiques temporelles basées sur une sémantique linéaire telles que LTL; la construction de la section 1.3 assure la préservation des seules propriétés de LTL (traces maximales et marquages), tandis que celle exposée en section 1.4 assure de plus la préservation des états du réseau (marquage et information temporelle). La section 1.5 discute de la préservation des propriétés des logiques à temps arborescent. La section 1.6 discute de l'analyse d'échéanciers et présente une méthode permettant de caractériser exactement les dates de tir possibles des transitions le long d'une séquence finie.

La boîte à outils `Tina`, permettant la mise en œuvre de ces techniques, ainsi que des techniques de vérification discutées ensuite, est présentée en section 1.7. Les sections suivantes sont consacrées à la vérification (*model-checking*) de formules de logique temporelle linéaire sur les graphes de classes. La section 1.8 présente la logique retenue, $SE{-}LTL$, une extension de la logique LTL, ainsi que sa mise en œuvre par le module `selt` de `Tina`. La section 1.9 présente deux exemples d'applications et leur vérification.

1.2. Les réseaux de Petri temporels

1.2.1. *Réseaux temporels*

Soit $\mathbf{I}^+$ l'ensemble des intervalles réels non vides à bornes rationnelles non négatives. Pour $i \in \mathbf{I}^+$, $\downarrow i$ désigne sa borne inférieure et $\uparrow i$ sa borne supérieure (si i est borné) ou ∞ (sinon). Pour tout $\theta \in \mathbf{R}^+$, $i \dot{-} \theta$ désignera l'intervalle $\{x - \theta | x \in i \wedge x \geq \theta\}$.

Définition 1. *Un* réseau temporel *(ou* Time Petri net, TPN *en abrégé) est un tuple* $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$*, dans lequel* $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0 \rangle$ *est un réseau de Petri et* $I_s : T \to \mathbf{I}^+$ *est une fonction appelée* Intervalle Statique.

L'application I_s associe une intervalle temporel $I_s(t)$ à chaque transition du réseau. Les rationnels $Eft_s(t) = \downarrow I_s(t)$ et $Lft_s(t) = \uparrow I_s(t)$ sont appelés date statique de tir au plus tôt de t et date statique de tir au plus tard de t, respectivement. Un réseau temporel est représenté figure 1.1.

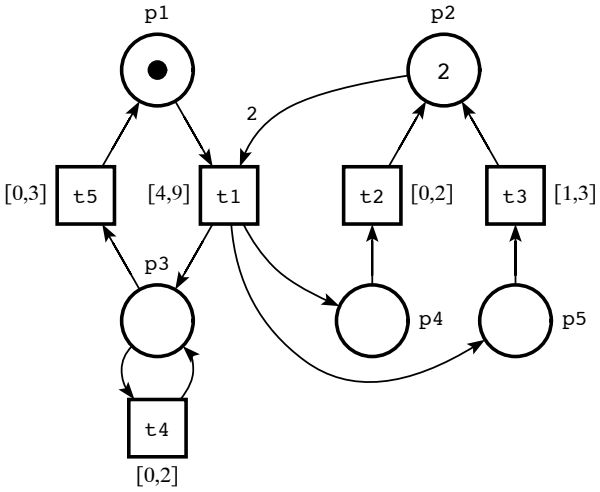


Figure 1.1. *Un réseau temporel.*

1.2.2. *États et relation d'accessibilité*

Définition 2. *Un état d'un réseau temporel est un couple $e = (m, I)$ dans lequel m est un marquage et $I : T \to \mathbf{I}^+$ une fonction qui associe un intervalle temporel à chaque transition sensibilisée par m.*

L'état initial est $e_0 = (m_0, I_0)$, où I_0 est la restriction de I_s aux transitions sensibilisées par le marquage initial m_0. Toute transition sensibilisée doit être tirée dans l'intervalle de temps qui lui est associé. Cet intervalle est relatif à la date de sensibilisation de la transition.

Franchir t, à la date θ, depuis $e = (m, I)$, est donc permis si et seulement si :

$$m \geq \mathbf{Pre}(t) \wedge \theta \in I(t) \wedge (\forall k \neq t)(m \geq \mathbf{Pre}(k) \Rightarrow \theta \leq \uparrow(I(k))).$$

L'état $e' = (m', I')$ atteint depuis e par le tir de t à θ est alors déterminé par :

1) $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$ (comme dans les réseaux de Petri)

2) Pour chaque transition k sensibilisée par m' :

I'(k) = si $k \neq t$ et $m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)$ alors $I(k) \dot{-} \theta$ sinon $I_s(k)$.

Notons $\xrightarrow{t@\theta}$ la relation d'accessibilité temporisée définie ci-dessus et définissons $e \xrightarrow{t} e'$ par $(\exists \theta)(e \xrightarrow{t@\theta} e')$. Un échéancier de tir est une séquence de transitions temporisées $t_1@\theta_1 \ldots t_n@\theta_n$; il est réalisable depuis l'état e si les transitions de la séquence

$\sigma = t_1 \dots t_n$ sont successivement tirables depuis l'état e, aux dates relatives qui leurs sont associées. La séquence σ est appelée support de l'échéancier. Une séquence de transitions est tirable si et seulemet si elle est le support d'un échéancier réalisable.

Notons que, dans les réseaux temporels (contrairement aux automates temporisés) l'écoulement du temps ne peut qu'augmenter l'ensemble des transitions tirables, mais en aucun cas le réduire. Si une transition est tirable à la date relative θ (ou, de façon équivalente, immédiatement après une attente de θ unités de temps), alors elle le restera tant que le temps peut s'écouler. Nous ne démontrerons pas ici cette propriété, mais il en résulte que la relation $\xrightarrow{t}$ définie ci-dessus caractérise exactement le comportement «discret» d'un TPN (préservant la bissimilarité après abstraction du temps). Si l'on s'intéresse au comportement temporel (capturé par la bissimulation temporisée), on doit ajouter aux transitions de la relation $\xrightarrow{t@\theta}$ celles traduisant le seul écoulement du temps, c'est-à-dire celles de la forme $(m, I) \xrightarrow{\delta(r)} (m, I \dot{-} r)$ (le délai r n'étant pas plus grand que $\uparrow(I(k))$, pour toute composante $I(k)$ de I).

Enfin, notons que le concept d'état présenté associe exactement un intervalle temporel à chaque transition sensibilisée, que celle-ci soit ou non multi-sensibilisée (t est multi-sensibilisée par m s'il existe un entier $k > 1$ tel que $m \geq k.\mathbf{Pre}(t)$). Une interprétation alternative est discutée dans [BER 01], dans laquelle un intervalle temporel est associé à chaque instance de sensibilisation des transitions. Cette interprétation sera brièvement discutée en section 1.3.4.1.

1.2.3. *Illustration*

Les états peuvent être représentés par des paires (m,D), dans lesquelles m est un marquage et D un ensemble de vecteurs de dates appelé domaine de tir. La i-ième projection de l'espace D est l'intervalle de tir $I(t_i)$ associé à la i-ième transition sensibilisée. Les domaines de tir seront décrits par des systèmes d'inéquations linéaires avec une variable par transition sensibilisée (notées comme les transitions).

L'état initial $e_0 = (m_0, D_0)$ du réseau figure 1.1 est ainsi représenté par :

$$\begin{array}{ll} m_0 : & p_1, p_2 * 2 \\ D_0 : & 4 \leq t_1 \leq 9 \end{array}$$

Le tir de t_1 depuis e_0, à une date relative $\theta_1 \in [4, 9]$, mène en $e_1 = (m_1, D_1)$:

$$\begin{array}{ll} m_1 : & p_3, p_4, p_5 \\ D_1 : & 0 \leq t_2 \leq 2 \\ & 1 \leq t_3 \leq 3 \\ & 0 \leq t_4 \leq 2 \\ & 0 \leq t_5 \leq 3 \end{array}$$

Le tir de t_2 depuis e_1, à une date relative $\theta_2 \in [0,2]$, mène en $e_2 = (m_2, D_2)$:

$$\begin{array}{ll} m_2 : & p_2, p_3, p_5 \\ D_2 : & \max(0, 1-\theta_2) \leq t_3 \leq 3-\theta_2 \\ & 0 \leq t_4 \leq 2-\theta_2 \\ & 0 \leq t_5 \leq 3-\theta_2 \end{array}$$

Comme θ_2 peut prendre toute valeur réelle dans $[0,2]$, l'état e_1 admet une infinité de successeurs par t_2.

1.2.4. *Quelques théorèmes généraux*

Rappelons tout d'abord un résultat d'indécidabilité :

Théorème 1. *Les problèmes de l'accessibilité d'un marquage, d'un état, du caractère borné et du caractère vivant, sont indécidables pour les $TPN's$.*

Démonstration : Il est démontré dans [JON 77] que le problème de l'accessibilité d'un marquage dans les $TPN's$ peut être réduit à celui, indécidable, de l'arrêt d'une machine à deux compteurs. Il en résulte l'indécidabilité des autres problèmes.

Représenter le fonctionnement d'un réseau temporel par le graphe d'accessibilité de ses états (comme le fonctionnement d'un réseau de Petri est représenté par le graphe d'accessibilité de ses marquages) est en général impossible : les transitions pouvant être tirées à tout instant dans leur intervalle de tir, les états admettent en général une infinité de successeurs par la règle de tir. Les classes d'états définies par la suite ont pour but de fournir une représentation finie de cet ensemble infini d'états, lorsque le réseau est borné, par regroupements de certains ensembles d'états. Toutefois, il existe deux sous-classes remarquables de réseaux temporels pour lesquels chaque état n'admet qu'un nombre fini de successeurs et donc admettant un graphe d'états fini lorsqu'ils sont bornés.

Théorème 2. *Soit un réseau temporel $\mathcal{N} = \langle \mathcal{P}, \mathcal{T}, \mathbf{Pre}, \mathbf{Post}, \mathcal{M}_\prime, \mathcal{I}_f \rangle$. Si, pour toute transition $t \in T$, $I_s(t)$ est non borné supérieurement, alors son graphe des états est isomorphe au graphe des marquages du réseau de Petri sous-jacent.*

Démonstration : Par induction. Si chaque transition porte un intervalle statique non borné, alors la condition de franchissement des transitions se réduit à celle des réseaux de Petri (sans contrainte temporelle). De plus, la règle de tir préserve le caractère non borné des intervalles.

Théorème 3. *Soit un réseau temporel $\mathcal{N} = \langle \mathcal{P}, \mathcal{T}, \mathbf{Pre}, \mathbf{Post}, \mathcal{M}_\iota, \mathcal{I}_\int \rangle$. Si I_s associe à chaque transition un intervalle réduit à un seul point, alors :*
(i) le graphe des états du réseau est fini si et seulement si le réseau est borné ;
(ii) si, de plus, les intervalles statiques sont tous identiques, alors son graphe des états est isomorphe au graphe des marquages du réseau de Petri sous-jacent.

Démonstration : Par induction. (i) Par la règle de tir, si tous les intervalles sont ponctuels, alors un état a, au plus, autant d'états suivants que de transitions qu'il sensibilise. De plus, la règle de tir préserve le caractère ponctuel des intervalles. Pour (ii), noter que la condition de franchissement se réduit alors à celle des réseaux de Petri.

Les théorèmes 2 et 3 permettent d'interpréter de diverses façons les réseaux de Petri comme des réseaux temporels. L'interprétation la plus fréquente est les considérer comme des réseaux temporels dont chaque transition porte l'intervalle $[0, \infty[$.

1.3. Graphes de classes préservant marquages et propriétés LTL

1.3.1. *Classes d'états*

L'ensemble des états d'un réseau temporel peut être infini pour deux raisons : d'une part parce qu'un état peut admettre une infinité de successeurs, d'autre part, parce qu'un réseau peut admettre des échéanciers de longueur infinie passant par des états dont les marquages sont tous différents. Le deuxième cas sera discuté section 1.3.3. Pour gérer le premier cas, on regroupera certains ensembles d'états en classes d'états. Plusieurs regroupements sont possibles ; le regroupement discuté dans cette section est celui introduit dans [BER 83 ; 91].

Pour chaque séquence de tir tirable σ, notons C_σ l'ensemble des états atteints depuis l'état initial en tirant des échéanciers de support σ. Pour tout ensemble C_σ, définissons son marquage comme celui des états qu'il contient (tous ces états ont nécessairement le même marquage) et son domaine de tir comme la réunion des domaines de tirs des états qu'il contient. Enfin, notons $\cong$ la relation satisfaite par deux ensembles C_σ et $C_{\sigma'}$ si ceux-ci ont même marquage et même domaine de tir. Si deux ensembles d'états sont liés par $\cong$, alors tout échéancier réalisable depuis un état de l'un de ces ensembles est réalisable depuis un état se trouvant dans l'autre ensemble.

Le graphe des classes d'états de [BER 83], ou SCG, est l'ensemble des ensembles d'états C_σ, pour toute séquence σ tirable, considérés modulo la relation $\cong$, muni de la relation de transition : $C_\sigma \xrightarrow{t} X$ si et seulement si $C_{\sigma.t} \cong X$. La classe initiale est la classe d'équivalence de l'ensemble d'états constitué du seul état initial.

Le SCG est obtenu comme suit. Les classes d'états sont représentées par des paires (m, D), où m est un marquage et D un domaine de tir décrit par un système

d'inéquations linéaires $W\underline{\phi} \leq \underline{w}$. Les variables ϕ sont bijectivement associées aux transitions sensibilisées par m. On a $(m, D) \cong (m', D')$ si et seulement si $m = m'$ et les domaines D et D' ont même ensemble de solutions.

Algorithme 1 (Calcul du SCG).
Pour toute séquence σ tirable, L_σ peut être calculée comme suit. Calculer le plus petit ensemble de classes contenant L_ϵ et tel que, lorsque $L_\sigma \in C$ et $\sigma.t$ est tirable, alors $(\exists X \in C)(X \cong L_{\sigma.t})$.

– La classe initiale est $L_\epsilon = (m_0, \{Eft_s(t) \leq \underline{\phi}_t \leq Lft_s(t) \mid m_0 \geq \mathbf{Pre}(t)\})$

– Si σ est tirable et $L_\sigma = (m, D)$, alors $\sigma.t$ est tirable si et seulement si :
(i) $m \geq \mathbf{Pre}(t)$ (t est sensibilisée par m) et ;
(ii) le système $D \wedge \{\underline{\phi}_t \leq \underline{\phi}_i \mid i \neq t \wedge m \geq \mathbf{Pre}(i)\}$ est consistant.

– Si $\sigma.t$ est tirable, alors $L_{\sigma.t} = (m', D')$ est obtenue depuis $L_\sigma = (m, D)$ par :
$m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$,
D' obtenu comme suit :
(a) Les contraintes (ii) ci-dessus de tirabilité de t depuis L_σ sont ajoutées à D ;
(b) Pour chaque k sensibilisée par m', une variable $\underline{\phi}'_k$ est introduite, par :
$\underline{\phi}'_k = \underline{\phi}_k - \underline{\phi}_t$, *si* $k \neq t$ *et* $m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)$,
$Eft_s(k) \leq \underline{\phi}'_k \leq Lft_s(k)$, *sinon ;*
(c) les variables $\underline{\phi}$ sont éliminées.

L_σ est la classe d'équivalence pour la relation $\cong$ de l'ensemble C_σ des états atteints en tirant depuis s_0 des échéanciers de support σ. L'équivalence $\cong$ est vérifiée en mettant les systèmes représentant les domaines de tir sous forme canonique. Ces systèmes sont des systèmes de différences, calculer leur forme canonique se ramène à un problème de plus court chemin entre toutes paires de sommets, résolu en temps polynômial en utilisant, par exemple, l'algorithme de Floyd/Warshall.

Remarque : deux ensembles C_σ et $C_{\sigma'}$ peuvent être équivalents par $\cong$ bien que de contenus différents. La notation des classes par une paire (m, D), identifie de façon canonique une classe d'équivalence d'ensembles d'états pour la relation $\cong$, mais pas un ensemble d'états.

1.3.2. *Illustration*

A titre d'illustration, construisons quelques classes du réseau temporel représenté figure 1.1. La classe initiale c_0 est décrite comme l'état initial e_0 (voir section 1.2.3). Le tir de t_1 depuis c_0 conduit à une classe c_1 décrite comme l'état e_1. Le tir de t_2

depuis c_1 conduit à $c_2 = (m_2, D_2)$, avec $m_2 = (p_2, p_3, p_5)$ et D_2 déterminé en 3 étapes :

(a) on ajoute à D_1 les conditions de tirabilité de t_2, exprimée par le système :

$$t_2 \leq t_3$$
$$t_2 \leq t_4$$
$$t_2 \leq t_5$$

(b) aucune transition n'est nouvellement sensibilisée et les transition t_3, t_4, t_5 restent sensibilisées lors du tir de t_2. On ajoute donc les équations $t'_i = t_i - t_2$, pour $i \in \{3, 4, 5\}$;

(c) on élimine les variables t_i, ce qui conduit au système :

$$0 \leq t'_3 \leq 3 \quad t'_4 - t'_3 \leq 1$$
$$0 \leq t'_4 \leq 2 \quad t'_5 - t'_3 \leq 2$$
$$0 \leq t'_5 \leq 3$$

Le graphe des classes d´états du réseau figure 1.1 admet douze classes et vingt-neuf transitions, il figure dans [BER 01]. La figure 1.2 montre un autre réseau temporel, qui nous servira pour comparer différentes sortes de graphes de classes, ainsi que son graphe de classes SCG.

1.3.3. *Vérification à la volée du caractère borné*

Il reste à examiner les conditions sous lesquelles l'ensemble des classes d'états est fini. Rappelons qu'un réseau de Petri ou temporel est borné si le marquage de toute place admet une borne supérieure. L'ensemble des domaines de tir d'un réseau temporel étant fini [BER 83], son graphe des classes est fini si et seulement si le réseau est borné. Cette propriété est indécidable (voir théorème 1), mais des conditions suffisantes peuvent être établies. Le théorème suivant en propose certaines, de nature comportementale, l'analyse structurelle en fournit d'autres.

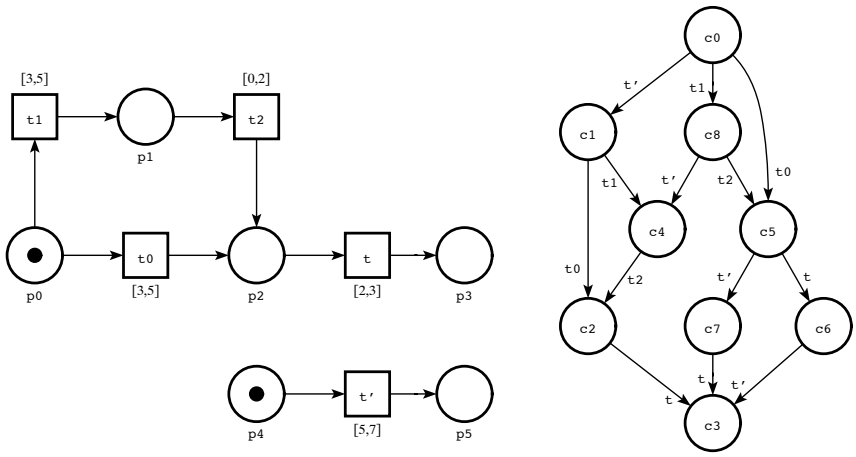
Théorème 4. *[BER 83] Un réseau temporel est borné si son SCG ne contient pas de paire de classes d'états $c = (m, D)$ et $c' = (m', D')$ telles que :*

i) c' *est accessible depuis* c,

ii) $m' \gtrdot m$,

iii) $D' = D$,

iv) $(\forall p)(m'(p) > m(p) \Rightarrow m'(p) \geq \max_{(t \in T)}\{\mathbf{Pre}(p, t)\})$.



classe marquage domaine de tir	$c0$ $p0,\ p4$ $5 \leq t' \leq 7$ $3 \leq t0 \leq 5$ $3 \leq t1 \leq 5$	$c1$ $p0,\ p5$ $0 \leq t0 \leq 0$ $0 \leq t1 \leq 0$	$c2$ $p2,\ p5$ $2 \leq t \leq 3$	$c3$ $p3,\ p5$	$c4$ $p1,\ p5$ $0 \leq t2 \leq 2$
classe marquage domaine de tir	$c5$ $p2,\ p4$ $2 \leq t \leq 3$ $0 \leq t' \leq 4$	$c6$ $p3\ , p4$ $0 \leq t' \leq 2$	$c7$ $p2\ , p5$ $0 \leq t \leq 3$	$c8$ $p1\ , p4$ $0 \leq t' \leq 4$ $0 \leq t2 \leq 2$	

Figure 1.2. *Réseau temporel et son SCG. La variable $\underline{\phi}_t$ est notée t.*

Les propriétés (i) à (iv) sont nécessaires pour qu'un réseau soit non borné, mais pas suffisantes. Ce théorème permet, par exemple, de démontrer que les réseaux des figures 1.1, 1.3 (gauche) et 1.3 (milieu) sont bornés, mais il ne permet pas de démontrer que le réseau figure 1.3 (droit) est borné bien que celui-ci n'admette que quarante-huit classes d'états. Si l'on omet (iv), on ne peut plus montrer que 1.3 (milieu) est borné, omettant de plus (iii), on ne peut inférer que 1.3 (gauche) est borné. La condition obtenue se résume alors à la propriété borné pour le réseau de Petri sous-jacent [KAR 69].

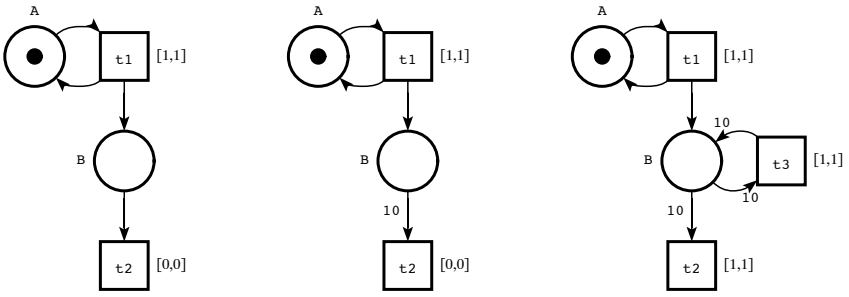


Figure 1.3. *Trois réseaux temporels bornés.*

1.3.4. *Variantes*

1.3.4.1. *Sensibilisations multiples*

Une transition t est multi-sensibilisée par un marquage m s'il existe un entier $k > 1$ tel que $m \geq k.\mathbf{Pre}(t)$. Dans les classes du graphe de classes introduit en section 1.3, chaque transition sensibilisée est associée à une et une seule variable temporelle, qu'elle soit ou non multi-sensibilisée ; les différentes dates de sensibilisation sont systématiquement identifiées avec la plus grande (la dernière) de ces dates.

Dans certaines applications, cette interprétation est limitative et on peut vouloir considérer les différentes instances de sensibilisations de façon distinctes. Ces aspects sont abordés dans [BER 01], où plusieurs interprétations opératoires de la multi-sensibilisation sont distinguées. On peut, naturellement, considérer les instances de sensibilisation comme indépendantes, ou encore comme ordonnées selon leurs dates de création. Dans tous les cas, l'algorithme 1 est aisément adapté et produit des graphes de classes «avec multi-sensibilisation». Le lecteur est renvoyé à [BER 01] pour plus de précisions.

1.3.4.2. *Préservation des seuls marquages*

Il est possible de compacter encore le graphe SCG en ne mémorisant pas les classes inclues dans une classe déjà construite. Plus précisément, soit $L_\sigma = (m, D)$ et $L_{\sigma'} = (m', D')$ deux classes, et $L_\sigma \sqsubseteq L_{\sigma'}$ si et seulement si $m = m'$ et $D \subseteq D'$. Alors, plutôt que de procéder comme dans l'algorithme 1, on construit un ensemble C de classes tel que, lorsque $L_\sigma \in C$ et $\sigma.t$ est tirable, on a $(\exists X \in C)(L_{\sigma.t} \sqsubseteq X)$.

Intuitivement, si une telle classe X existe, tout échéancier tirable depuis un état de $L_{\sigma.t}$ est aussi tirable depuis un état de X, on ne trouvera donc pas de nouveaux marquages en mémorisant $L_{\sigma.t}$. Bien entendu, cette construction ne préserve plus les séquences de tir du graphe d'états, donc ses propriétés LTL, mais seulement les marquages. Elle produit typiquement des graphes plus compacts.

1.4. Graphes de classes préservant états et propriétés LTL

Comme mentionné dans la section précédente, les classes construites par l'algorithme 1 représentent des classe d'équivalences d'ensembles d'états, mais ne représentent pas canoniquement les ensembles C_σ définis au paragraphe 1.3.1. La conséquence est que le SCG ne peut pas être utilisé pour démontrer l'accessibilité ou non d'un état particulier du TPN.

Les classes d'états fortes introduites dans cette section coïncident exactement avec ces ensembles d'états C_σ, considérées ici sans autre forme d'équivalence que l'égalité. Le graphe des classes fortes préserve les propriétés LTL et les états, dans un sens que nous préciserons par la suite.

1.4.1. *Domaines d'horloges*

Pour construire le graphe des classes d'états fortes (ou $SSCG$), il est nécessaire de représenter ces ensembles d'états C_σ de façon canonique. Une représentation adéquate est fournie par les domaines d'horloges.

A tout échéancier on peut associer une fonction horloge γ comme suit : à toute transition sensibilisée après le tir de l'échéancier, la fonction γ associe le temps écoulé depuis la dernière sensibilisation de cette transition. Pour un état donné, la fonction horloge sera aussi vue comme un vecteur $\underline{\gamma}$, indexé par les transitions sensibilisées.

L'ensemble d'états décrit par un marquage m et un système d'horloges $Q = \{G\underline{\gamma} \leq \underline{g}\}$ est l'ensemble $\{(m, \Phi(\underline{\gamma}))|\underline{\gamma} \in Sol(Q)\}$, où $Sol(Q)$ est l'ensemble des solutions de Q, et le domaine de tir $\Phi(\underline{\gamma})$ est l'ensemble des solutions en $\underline{\phi}$ du système :

$$\underline{0} \leq \underline{\phi}\ ,\ \underline{e} \leq \underline{\phi} + \underline{\gamma} \leq \underline{l} \text{ où } \underline{e_k} = Eft_s(k) \text{ and } \underline{l_k} = Lft_s(k)$$

Notons $\equiv$ la relation qui lie deux paires (marquages, système d'horloges) lorsqu'elles décrivent exactement le même ensemble d'états. En général, différents vecteurs d'horloges peuvent décrire le même domaine de tir. Toutefois, l'équivalence $\equiv$ est aisément établie dans un cas particulier :

Théorème 5. *[BER 03b] Soit $c = (m, Q = \{G\underline{\gamma} \leq \underline{g}\})$ et $c' = (m', Q' = \{G'\underline{\gamma}' \leq \underline{g}'\})$. Si toute transition sensibilisée par m ou m' a un intervalle statique borné, alors $c \equiv c'$ ssi $m = m' \wedge Sol(Q) = Sol(Q')$*

Le cas général sera évoqué plus loin, noter toutefois que, lorsqu'une transition k est tirable à toute date au-delà de sa date de tir au plus tôt e, alors tous les vecteurs d'horloge dont la composante k est au moins égale à e, et ne différant que par cette composante, décrivent le même état.

1.4.2. *Construction du* $SSCG$

Les classes d'états fortes sont représentées par des paires (m, Q), où m est un marquage et Q un domaine d'horloges décrit par un système d'inéquations $G\underline{\gamma} \leq \underline{g}$. Les variables $\underline{\gamma}$ sont bijectivement associées aux transitions sensibilisées par m. L'équivalence $\equiv$ est implantée comme dans le théorème 5.

Algorithme 2 (Calcul du $SSCG$, classes fortes).
Une classe R_σ calculée comme ci-dessous peut être associée à toute séquence tirable σ. Calculer le plus petit ensemble C de classes contenant R_ϵ et tel que, lorsque $R_\sigma \in C$ et $\sigma.t$ est tirable, alors $(\exists X \in C)(X \equiv R_{\sigma.t})$.

– *La classe initiale est* $R_\epsilon = (m_0, \{0 \leq \underline{\gamma}_t \leq 0 \mid m_0 \geq \mathbf{Pre}(t)\})$.

– *Si σ est tirable et $R_\sigma = (m, Q)$, alors $\sigma.t$ est tirable si et seulement si :*

(i) $m \geq \mathbf{Pre}(t)$ *(t est sensibilisée par m) et,*

(ii) le système $Q \wedge \{0 \leq \theta\} \wedge \{Eft_s(t) - \underline{\gamma}_t \leq \theta \leq Lft_s(i) - \underline{\gamma}_i \mid m \geq \mathbf{Pre}(i)\}$ *est consistant (θ est une nouvelle variable).*

– *Si $\sigma.t$ est tirable, alors $R_{\sigma.t} = (m', Q')$ est calculée depuis $R_\sigma = (m, Q)$ par :*

$m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$,

Q' obtenu comme suit :

(a) Une nouvelle variable est introduite, θ, contrainte par les conditions (ii),

(b) Pour chaque i sensibilisée par m', une variable $\underline{\gamma}'_i$ est introduite, par :
$\underline{\gamma}'_i = \underline{\gamma}_i + \theta$, *si* $i \neq t$ *et* $m - \mathbf{Pre}(t) \geq \mathbf{Pre}(i)$,
$0 \leq \underline{\gamma}'_i \leq 0$, *sinon ;*

(c) Les variables $\underline{\gamma}$ et θ sont éliminées.

La variable temporaire θ décrit les dates de tir possible de t depuis les états de la classe de départ. Il y a un arc étiqueté t entre les classes R_σ et X si et seulement si $X \equiv R_{\sigma.t}$. Comme les systèmes décrivant les domaines de tir dans les classes du SCG, les systèmes d'horloges sont des systèmes de différences, l'équivalence $\equiv$ est vérifiée en temps polynômial en mettant ces systèmes sous forme canonique.

Si le réseau satisfait à la restriction introduite dans le théorème 5 (tous les intervalles statiques sont bornés), alors l'ensemble des systèmes d'horloges distincts que l'on peut construire avec l'algorithme 2 est fini. Si cette condition n'est pas satisfaite, alors, pour assurer la terminaison de la construction, on doit ajouter dans l'algorithme 2 une étape (d) de relaxation de la classe construite. Cette étape est expliquée en détail dans [BER 03b], elle revient à noter l'ensemble d'états C_σ par le plus grand ensemble d'horloges décrivant cet ensemble d'états. Ce plus grand ensemble d'horloges n'est pas en général convexe, mais il est toujours constitué d'une réunion finie d'ensembles

convexes. En pratique, la représentation des classes et la relation $\equiv$ sont inchangées, mais une classe pourra avoir plusieurs classes suivantes par la même transition.

Comparé au SCG, le $SSCG$ préserve aussi les traces et traces maximales du réseau et donc ses propriétés exprimables en logique LTL, mais il permet de plus de vérifier l'accessibilité d'un état. Soit $s = (m, I)$ un état dont on veut vérifier l'accessibilité. Alors, on peut toujours calculer un vecteur d'horloges $\underline{\Gamma}$ tel que, pour toute transition k sensibilisée par m, l'on ait $I(k) = I_s(k) \dot{-} \underline{\Gamma}_k$. Les propriétés du $SSCG$ assurent alors que s est accessible si et seulement si il existe une classe forte (m', Q) du $SSCG$ telle que $m = m'$ et $\underline{\Gamma} \in Q$.

Le $SSCG$ du réseau figure 1.2 admet onze classes et seize transitions, il est représenté figure 1.4 (gauche). Si on le compare au SCG de ce réseau, figure 1.2, on notera que les ensembles d'états $C_{t'.t1}$ et $C_{t1.t'}$ sont distingués dans le $SSCG$ (représentés par les classes $c4$ et $c9$, respectivement), alors qu'ils étaient équivalents par $\cong$ dans le SCG. Il en est de même pour les ensembles $C_{t1.t2}$ ($c10$) et C_{t0} ($c5$).

1.4.3. *Variantes*

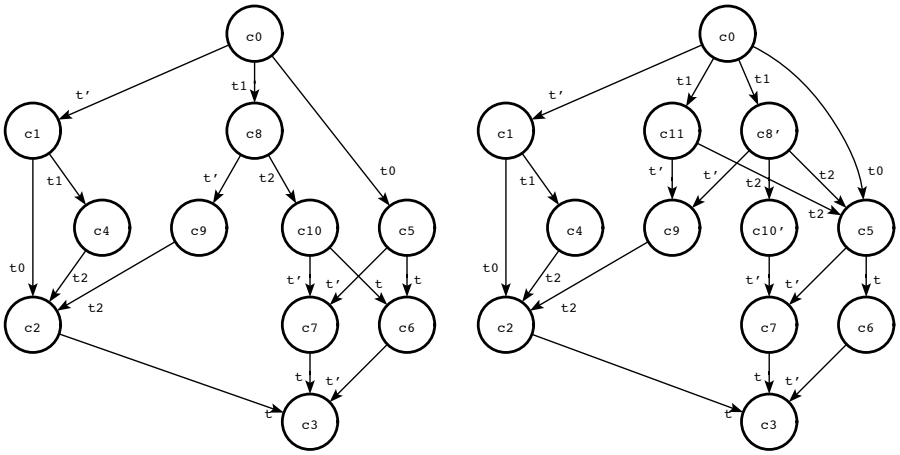
Comme pour le calcul du SCG (voir section 1.3.3), on peut aisément ajouter à l'algorithme 2 la vérification à la volée d'une condition suffisante pour le caractère borné.

Enfin, comme pour le SCG (voir section 1.3.4.2), comme proposé dans [BOU 04], on peut construire une version généralement plus compacte du $SSCG$ ne préservant que les états (mais ne préservant pas les séquences de tir). Il suffit pour celà de ne pas mémoriser une classe contenue dans une classe déjà construite.

1.5. Graphes de classes préservant états et propriétés CTL

Les propriétés de branchement sont celles exprimées dans les logiques temporelles à temps arborescent comme CTL, ou dans les logiques modales comme HML ou le μ-calcul. Ni le SCG, ni le $SSCG$, ne préservent ces formules. Considérons le réseau représenté figure 1.2 avec son graphe de classes SCG. Un exemple de formule HML dont la valeur n'est pas préservée par ce graphe est $<t1>[t2]<t>\mathbf{T}$, exprimant que, *via* $t1$, il est possible d'atteindre un état depuis lequel tout tir de $t2$ conduit à un état permettant de tirer t. Cette formule est vraie sur le SCG, mais fausse sur le graphe des états du réseau : en effet, après avoir tiré l´échéancier $(t1@5.t2@2)$, les transitions t' et t sont toutes deux sensibilisées, mais seule t' est tirable, à la date 0.

En l'absence de transitions silencieuses, les propriétés de branchement sont capturées par la notion de bissimulation ; tout graphe de classes bissimilaire au graphe



classe marquage domaine d'horloges	$c0$ $p0,\ p4$ $0 \leq t' \leq 0$ $0 \leq t0 \leq 0$ $0 \leq t1 \leq 0$	$c1$ $p0,\ p5$ $5 \leq t0 \leq 5$ $5 \leq t1 \leq 5$	$c2$ $p2,\ p5$ $0 \leq t \leq 0$	$c3$ $p3\ , p5$
classe marquage domaine d'horloges	$c4$ $p1\ , p5$ $0 \leq t2 \leq 0$	$c5$ $p2\ , p4$ $0 \leq t \leq 0$ $3 \leq t' \leq 5$	$c6$ $p3\ , p4$ $5 \leq t' \leq 7$	$c7$ $p2\ , p5$ $0 \leq t \leq 3$
classe marquage domaine d'horloges	$c8$ $p1\ , p4$ $3 \leq t' \leq 5$ $0 \leq t2 \leq 0$	$c9$ $p1\ , p5$ $0 \leq t2 \leq 2$	$c10$ $p2\ , p4$ $0 \leq t \leq 0$ $3 \leq t' \leq 7$	$c8'$ $p1,\ p4$ $3 < t' \leq 5$ $0 \leq t2 \leq 0$
classe marquage domaine d'horloges	$c10$ $p2,\ p4$ $0 \leq t \leq 0$ $5 < t' \leq 7$	$c11$ $p1,\ p4$ $3 \leq t' \leq 3$ $0 \leq t2 \leq 0$		

Figure 1.4. *$SSCG$ et $ASCG$ du réseau figure 1.2. La variable $\underline{\gamma}_t$ est notée t.*

des états (annotations temporelles omises) préserve toutes ses propriétés de branchement. Une première construction pour un tel graphe de classes a été proposée dans [YON 98], pour la sous-classe des $TPN's$ dans lesquels les intervalles statiques des transitions sont bornés supérieurement. Une spécialisation de cette construction, ne préservant que les formules de la logique $ACTL$, est proposée dans [PEN 01]. Plutôt que ces constructions, nous rappellerons celle de [BER 03b], qui s'applique à tout TPN et produit généralement des graphes plus compacts.

La technique «standard» de calcul de bissimulation est celle du «raffinement minimal de partition» introduite dans [PAI 87]. Soit $\rightarrow$ une relation binaire sur un ensemble fini U. Pour tout $S \subseteq U$, soit $S^{-1} = \{x | (\exists y \in S)(x \rightarrow y)\}$. Une partition P de U est stable si, pour toute paire (A, B) de blocs de P, soit $A \subseteq B^{-1}$ soit $A \cap B^{-1} = \emptyset$ (A sera dite stable par rapport à B). Calculer une bissimulation depuis une partition initiale P d'états, est calculer un raffinement stable Q de P [KAN 90]. Un algorithme est le suivant [PAI 87] :

Initialement : $Q = P$
tant que : il existe $A, B \in Q$ tels que $\emptyset \not\subseteq A \cap B^{-1} \not\subseteq A$
faire : replacer A par $A_1 = A \cap B^{-1}$ et $A_2 = A - B^{-1}$ dans Q

Notre contexte est sensiblement différent de celui supposé ci-dessus, car nos ensembles d'états ne sont pas finis, mais la méthode peut être adaptée. Suivant [YON 98], appelons atomique une classe stable par rapport à toutes ses classes suivantes, c'est-à-dire dont chaque état a un successeur dans chacune de ses classes suivantes. Les classes fortes du $SSCG$ sont adéquates comme partition initiale (contrairement aux classes du SCG, l'équivalence $\cong$ ne préservant pas l'atomicité). Toutefois, le fait que cet ensemble soit un recouvrement plutôt qu'une partition, implique que le résultat final sera généralement non minimal en nombre de blocs, et non unique.

Le graphe des classes d'états atomiques (ou $ASCG$) sera obtenu par raffinement du graphe des classes fortes ($SSCG$). La technique de partition des classes est expliquée en détail dans [BER 03b]. Intuitivement, pour chaque transition du graphe courant, on calcule depuis la classe destination c' l'ensemble des (horloges possibles des) états ayant un successeur dans c'. L'intersection de cet ensemble avec celui capturé par la classe source c définit une partition de c. Les classes atomiques sont considérées modulo l'équivalence $\equiv$, comme les classes fortes. Si un intervalle statique au moins est non borné, alors le $ASCG$ doit être construit en prenant comme recouvrement initial le $SSCG$ «relaxé» (voir section 1.4.2).

Par exemple, le $ASCG$ du réseau figure 1.2 admet douze classes et dix-neuf transitions, il est représenté figure 1.4. La seule classe non atomique de son $SSCG$, $c10$, a d'abord été partitionnée en $c10'$ et une classe équivalente par $\equiv$ à $c5$. La classe $c8$ du $SSCG$ est alors devenue non atomique, et a à son tour été partitionnée en $c8'$ et $c11$. Il peut être vérifié sur le $ASCG$ qu'il préserve la valeur de vérité de notre formule HML exemple $<t1>[t2]<t>\mathbf{T}$.

1.6. Calculs d'échéanciers

1.6.1. *Systèmes d'échéanciers*

Pour chaque séquence tirable σ, les dates possible de franchissement des transitions le long de σ sont liées par un système d'inéquations linéaires $P\underline{\theta} \leq \underline{p}$; la i-ième composante du vecteur $\underline{\theta}$ décrit les dates de tir possible pour la i-ième transition de σ.

Pour calculer ces systèmes, on peut procéder comme pour le calcul des classes d'états fortes par l'algorithme 2, mais en n'éliminant pas les variables auxiliaires θ. Cette méthode est exprimée par l'algorithme 3 ci-après.

Algorithme 3 (Calcul des échéanciers relatif à une séquence de tir σ).

Notons σ^i le préfixe de la séquence σ de longueur i et σ_i la i-ième transition de σ.
On calcule par la méthode ci-dessous une suite de paires Z^i, pour $i \in \{0, \ldots, k\}$, constituées d'un marquage et d'un système d'inéquations de la forme $P(\underline{\theta}|\underline{\gamma}) \leq \underline{p}$.
Intuitivement, si $Z^i = (m, K)$, alors m est le marquage atteint après le tir de σ^i, le système K caractérise simultanément les dates de tir possibles des transitions le long de σ^i, et les valeurs d'horloges pour les transitions sensibilisées par m.
Le système recherché est celui obtenu en éliminant les variables d'horloge $\underline{\gamma}$ dans le système d'inéquations de la dernière paire calculée, Z^k.

– Initialement, $Z^0 = (m_0, \{0 \leq \underline{\gamma}_t \leq 0 \mid m_0 \geq \mathbf{Pre}(t)\})$. Cette paire a la forme requise, bien qu'aucune variable «de chemin» θ n'apparaisse dans le système d'inéquations.

– Soit $t = \sigma_i$ et $Z^i = (m, K)$. Si σ^i est tirable, alors $\sigma^i.t$ l'est si et seulement si :

(i) $m \geq \mathbf{Pre}(t)$ (t est sensibilisée par m) et,

(ii) le système $K \wedge \{0 \leq \theta\} \wedge \{Eft_s(t) - \underline{\gamma}_t \leq \theta \leq Lft_s(i) - \underline{\gamma}_i \mid m \geq \mathbf{Pre}(i)\}$ est consistant (θ est une nouvelle variable «de chemin»).

– La paire $Z^{i+1} = (m', K')$ est obtenue comme suit depuis la paire $Z^i = (m, K)$:

$m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$,

K' obtenu par :

(a) Une nouvelle variable est introduite, θ, contrainte comme en (ii) ci-dessus ;

(b) Pour chaque j sensibilisée par m', une variable $\underline{\gamma}'_j$ est introduite, par :

$\underline{\gamma}'_j = \underline{\gamma}_j + \theta$, *si* $i \neq t$ *et* $m - \mathbf{Pre}(t) \geq \mathbf{Pre}(j)$,
$0 \leq \underline{\gamma}'_j \leq 0$, *sinon ;*

(c) Les variables $\underline{\gamma}$ sont éliminées.

La mise en œuvre exacte de l'algorithme 3 ne sera pas détaillée ici. Plusieurs optimisations sont essentielles pour de bonnes performances sur de longues séquences. En

particulier, ces systèmes sont typiquement structurés en sous-systèmes indépendants, propriété qu'il est intéressant d'exploiter. Alternativement, plutôt que par l'algorithme ci-dessus, les systèmes d'échéanciers peuvent être reconstruits depuis la séquence de classes d'états ou de classes fortes (sans relaxation) construites pour la séquence σ.

Ces systèmes permettent de résoudre certains problèmes «quantitatifs» concernant les échéanciers, notamment celui de l'échéancier le plus rapide, ou le plus lent, de support donné, ou l'existence d'échéanciers de support donné satisfaisant des conditions particulières impliquant les délais de tir.

Les systèmes $P\underline{\theta} \leq \underline{p}$ produits pour σ par l'algorithme 3 ont une structure particulière : chaque inéquation a l'une des formes suivantes, dans lesquelles a et b sont des constantes entières et I est un intervalle d'entiers de $\{1, \ldots, |\sigma|\}$. Noter que les variables impliquées dans les sommes concernent nécessairement des transitions contigües de la séquence de tir support :

$$\begin{array}{ll} a \leq \theta & \theta \leq b \\ a \leq \Sigma_{i \in I}(\theta_i) & \Sigma_{i \in I}(\theta_i) \leq b \end{array}$$

1.6.2. *Délais (dates relatives) et dates (absolues)*

Les systèmes construits par l'algorithme 3 caractérisent des dates relatives de tir (des délais). Il est possible de reformuler cet algorithme afin de produire des systèmes en «dates absolues» de tir, plutôt que relatives. Dans ce cas, on introduira une variable supplémentaire, $start$, qui représente la date d'initialisation du réseau.

La date absolue de tir de la i-ième transition de σ est la date initiale $start$ augmenté des dates relatives de tir des i premières transitions de σ. On passe donc aisément d'un système «en délais» (θ_i) à un système «en dates» (ψ_i), ou inversement, par la transformation suivante (toutes les variables étant non négatives) :

$\psi_i - \psi_{i-1} = \theta_i$ (en posant $\psi_0 = start$)

Noter que, appliquée à un système «en délais», et compte tenu de la forme générale de ces systèmes, cette transformation produit un système de différences.

1.6.3. *Illustration*

A titre d'exemple, tout échéancier de support $t1.t2.t.t'$ dans le réseau figure 1.2 a la forme $t1@\theta_0.t2@\theta_1.t@\theta_2.t'@\theta_3$, les variables θ_i satisfaisant les inéquations suivantes :

$$
\begin{array}{l}
3 \leq \theta_1 \leq 5 \\
3 \leq \theta_1 + \theta_2 \leq 7 \\
0 \leq \theta_2 \leq 2 \\
5 \leq \theta_1 + \theta_2 + \theta_3 \leq 7 \\
2 \leq \theta_3 \leq 3 \\
5 \leq \theta_1 + \theta_2 + \theta_3 + \theta_4 \leq 7 \\
0 \leq \theta_4
\end{array}
$$

Exprimé en dates, plutôt qu'en délais, on obtiendrait le système suivant, dans lequel $start$ est la date d'initialisation du réseau, et ψ_i la date à laquelle est tirée la i-ième transition de σ :

$$
\begin{array}{l}
3 \leq \psi_1 - start \leq 5 \\
3 \leq \psi_2 - start \leq 7 \\
0 \leq \psi_2 - \psi_1 \leq 2 \\
5 \leq \psi_3 - start \leq 7 \\
2 \leq \psi_3 - \psi_2 \leq 3 \\
5 \leq \psi_4 - start \leq 7 \\
0 \leq \psi_4 - \psi_3
\end{array}
$$

1.7. Mise en œuvre, l'environnement `Tina`

`Tina` (*TIme Petri Net Analyzer*)[1] est un environnement logiciel permettant l'édition et l'analyse de réseaux de Petri et réseaux temporels. Les différents outils constituant l'environnement peuvent être utilisés de façon combinée ou indépendante. Ces outils incluent :

`nd` (*NetDraw*) : `nd` est un outil d'édition de réseaux temporels et d'automates, sous forme graphique ou textuelle. Aussi, il intègre un simulateur «pas à pas» (graphique ou textuel) pour les réseaux temporels et permet d'invoquer les outils ci-dessous sans sortir de l'éditeur.

`tina` : cet outil construit des représentation de l'espace d'états d'un réseau de Petri, temporel ou non. Aux constructions classiques (graphe de marquages, arbre de couverture), `tina` ajoute la construction d'espaces d'états abstraits, basés sur les techniques d'ordre partiel, préservant certaines classes de propriétés, comme l'absence de blocage, les propriétés de certaines logiques, ou les équivalences de test ; ces facilités sont décrites dans [BER 04]. Pour les réseaux temporels, `tina` propose toutes les constructions de graphes de classes discutées dans ce chapitre. Tous ces graphes peuvent être produits dans divers formats : «en clair» (à but pédagogique), dans un format d'échange compact à destination des autres outils de l'environnement, ou bien dans

1. http//www.laas.fr/tina.

les formats acceptés par certains vérificateurs de modèles externes, comme MEC [ARN 94] pour la vérification de formules du μ-calcul, ou les outils $CADP$ [FER 96] pour notamment la vérification de préordres ou d'équivalences de comportements.

Dans toutes ces constructions `tina` peut vérifier à la volée certaines propriétés «générales» telles que le caractère borné (condition souvent nécessaire pour une implantation ultérieure et souvent requise pour envisager d'autres vérifications), la présence de blocage - suivant le cas il sera attendu (toujours le cas pour un calcul qui doit en particulier terminer) ou non désiré (souvent le cas pour les systèmes réactifs) - la pseudo-vivacite qui permet de rechercher le «code mort» (transition jamais activée) ou encore la vivacité qui permet de s'assurer qu'une transition (ou une configuration du système) est atteignable à partir de tout état du comportement du système.

`plan` : cet outil permet le calcul de systèmes d'échéanciers comme expliqué section 1.6. Il produit à la demande le système complet d'échéanciers, ou une solution de ce système, en délais ou dates, mis ou non sous forme canonique.

`struct` : il s'agit d'un outil d'analyse structurelle, non décrit ici, calculant des ensembles générateurs de flots ou semiflots, sur les places et/ou transitions.

`selt` : nn plus des propriétés générales vérifiées par `tina`, il est le plus souvent indispensable de pouvoir garantir des propriétés spécifiques relatives au système modélisé. L'outil `selt` est un vérificateur de modèle (*model-checker*) pour les formules d'une extension de la logique temporelle $SE{-}LTL$ (State/Event LTL) de [CHA 04]. Il opère sur les abstractions d'espaces d'état produites par `tina`, ou, à travers un outil de conversion, sur des graphes produits par d'autres outils tels que les outils $CADP$ [FER 96]. La mise en œuvre de ce vérificateur sera détaillée dans le paragraphe 1.8.2.

`ndrio`, `ktzio` : il s'agit d'outils de conversion de formats de réseaux (`ndrio`) et de systèmes de transitions (`ktzio`).

Ces différents outils peuvent coopérer au travers de fichiers dans des formats d'échange documentés. Une capture d'écran d'une session `Tina` est reproduite figure 1.5, avec un réseau temporel en cours d'édition, un résultat textuel de construction de graphe de classes et une représentation graphique de ce graphe.

1.8. La vérification de formules $SE{-}LTL$ dans `Tina`

Dans cette section, nous nous focalisons sur la vérification de propriétés spécifiques et présentons les techniques et outils mis à disposition dans `Tina` pour réaliser cette tâche. La vérification de propriétés spécifiques nécessite en premier lieu un langage formel permettant d'énoncer les propriétés que l'on cherche à vérifier, nous utiliserons pour cela une variante de la logique temporelle à temps linéaire $SE{-}LTL$,

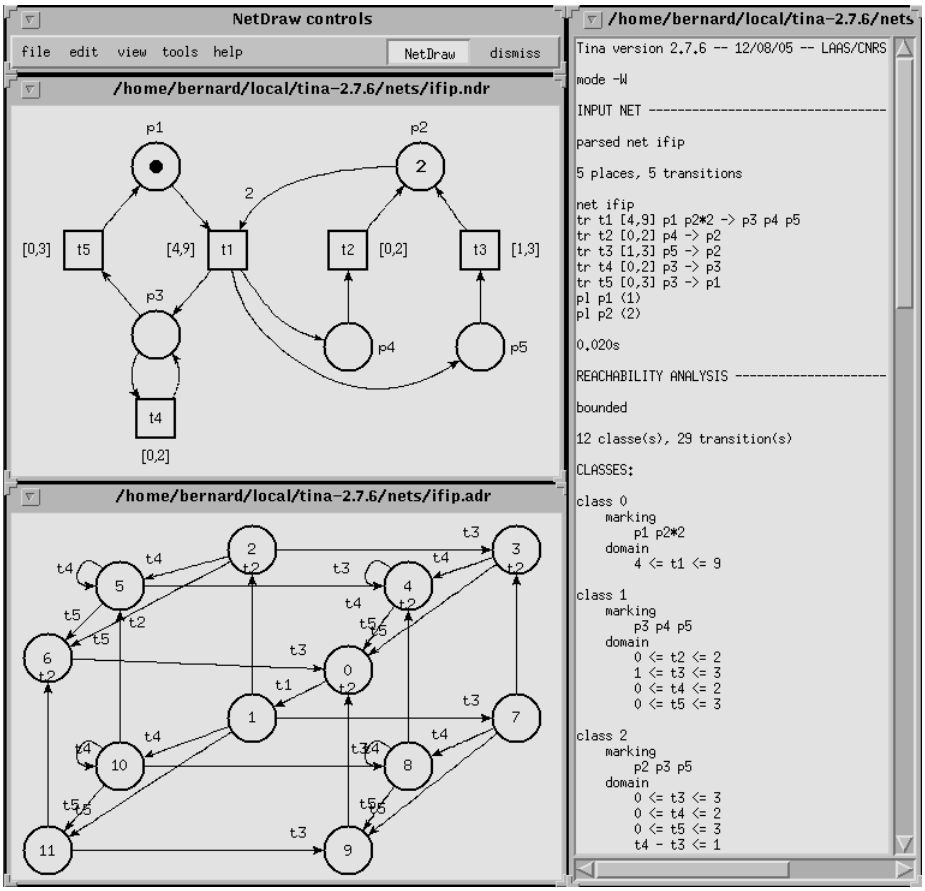


Figure 1.5. *Capture d'écran d'une session* `Tina`.

ensuite une abstraction du comportement du réseau préservant les formules de cette logique (les constructions SCG et $SSCG$ de `tina` sont adéquates) et enfin un outil (le module `selt`) qui permet de contrôler la satisfaction des propriétés formellement énoncées sur la représentation produite du comportement du réseau.

1.8.1. *La logique temporelle* $SE-LTL$

La logique LTL étend le calcul propositionnel en permettant l'expression de propriétés spécifiques sur les séquences d'exécution d'un système. $SE-LTL$ est une variante de LTL récemment introduite [CHA 04], qui permet de traiter de façon homogène des propositions d'états et des propositions de transitions. Les modèles pour

la logique $SE{-}LTL$ sont des structures de Kripke étiquetées (ou SKE), aussi appelées systèmes de transitions de Kripke (ou KTS).

Définition 3. *Structure de Kripke étiquetée*

Un tuple $\mathcal{SKE} = \langle S, Init, AP, \nu, T, \Sigma, \epsilon \rangle$ *constitué de :*
– S*, un ensemble fini d'états,*
– $Init \subset S$*, un sous-ensemble d'états initiaux,*
– AP*, un ensemble fini de propositions atomiques d'états,*
– $\nu : S \to 2^{AP}$*, un étiquetage des états par des ensembles de propositions,*
– $T \subseteq S \times S$*, un ensemble fini de transitions,*
– Σ*, un ensemble fini d'événements, ou actions,*
– $\epsilon : T \to 2^{\Sigma}$*, un étiquetage des transitions par des ensembles d'événements.*

On écrira $s \stackrel{E}{\to} q$ lorsque $(s,q) \in T \; et \; \epsilon((s,q)) = E$. Un chemin, ou exécution, d'une $\mathcal{SKE}$ est une suite infinie $\pi = \langle s_1, a_1, s_2, a_2, \ldots \rangle$ alternant états et transitions telle que $\forall i \geq 1, s_i \in S, a_i \in \Sigma \; et \; s_i \stackrel{a_i}{\to} s_{i+1}$. La paire (s_i, a_i) sera appelée étape i. Classiquement, la relation d'accessibilité est supposée totale (tout état admet un successeur). On note $\Pi(\mathcal{SKE})$ l'ensemble de tous les chemins ayant leur origine dans l'un des états $s \in Init$. Enfin, pour un chemin $\pi = \langle s_1, a_1, s_2, a_2, \ldots \rangle$, π^i désigne le suffixe du chemin π commençant à l'état s_i.

Définition 4. $SE{-}LTL$

SE–LTL est définie sur les ensembles AP et Σ d'une $\mathcal{SKE}$. p dénote une variable de AP et a une variable de Σ.

Les formules Φ de $SE{-}LTL$ sont définies par la grammaire suivante :

$$\Phi ::= p \mid a \mid \neg\Phi \mid \Phi \vee \Phi \mid \bigcirc \Phi \mid \Box \Phi \mid \Diamond \Phi \mid \Phi \; U \; \Phi$$

Leur sémantique est définie inductivement comme suit :

$\mathcal{SKE} \models \Phi$ *ssi* $\pi \models \Phi$ *pour tout chemin* $\pi \in \Pi(\mathcal{SKE})$
$\pi \models p$ *ssi* $p \in \nu(s_1)$ *(où* s_1 *est le premier état de* π*)*
$\pi \models a$ *ssi* $a \in \epsilon(a_1)$ *(où* a_1 *est la première transition de* π*)*
$\pi \models \neg\Phi$ *ssi* $non \; \pi \models \Phi$
$\pi \models \bigcirc \Phi$ *ssi* $\pi^2 \models \Phi$
$\pi \models \Box \Phi$ *ssi* $\forall i \geq 1, \pi^i \models \Phi$
$\pi \models \Diamond \Phi$ *ssi* $\exists i \geq 1, \pi^i \models \Phi$
$\pi \models \Phi_1 \; U \; \Phi_2$ *ssi* $\exists i \geq 1, \pi^i \models \Phi_2$ *et* $\forall 1 \geq j \geq i-1, \pi^j \models \Phi_1$

Exemple 1 Quelques formules de $SE{-}LTL$:

	(Pour tout chemin)
P	P vraie au départ du chemin (pour l'étape initiale),
$\bigcirc P$	P vraie dans l'étape suivante,
$\Box P$	P vraie tout le long du chemin,
$\Diamond P$	P vraie une fois au moins le long du chemin,
$P\,U\,Q$	Q sera vraie dans le futur et P est vraie jusqu'à cet instant,
$\Box \Diamond P$	P vraie infiniment souvent,
$\Box (P \Rightarrow \Diamond Q)$	Q «répond» à P.

1.8.2. *Préservation des propriétés LTL par les constructions de* `tina`

Les graphes de classes décrits dans les sections 1.3 à 1.5 préservent tous les marquages et traces maximales du graphe des états, et donc la valeur de vérité de toutes les formules de $SE{-}LTL$. Les structures de Kripke associées à ces graphes sont constituées de ces graphes, dont les nœuds (états de la structure de Kripke) sont étiquetés par les marquages (interprétés comme les propriétés d'états satisfaites) et les arcs par les transitions tirées (interprétées comme les propriétés de transitions satisfaites).

Les intervalles temporels statiques associés aux transitions d'un réseau temporel n'étant pas nécessairement bornés, il est possible que l'on puisse résider indéfiniment dans certains états, sans qu'une «boucle» sur les classes contenant ces états ne matérialise dans le graphe de classes ce temps d'attente arbitraire ; cette information figure toutefois dans l'information temporelle capturée par les classes. L'utilisateur a le choix, à la génération du graphe de classes ou à son chargement dans le vérificateur, de prendre ou non en compte ces temps d'attente arbitraires. S'ils sont pris en compte, une boucle sur les états d'attente sera ajoutée dans la structure de Kripke, ainsi qu'une propriété spécifique *div* (pour divergence temporelle).

D'autre part, la relation d'accessibilité dans les structures de Kripke étant supposée totale, on ajoutera au graphe obtenu une boucle sur tous les nœuds n'ayant aucun suivant, ainsi qu'une propriété spécifique (*dead*).

1.8.3. `selt` *: le vérificateur $SE{-}LTL$ de* `Tina`

1.8.3.1. *Technique de vérification*

Dans cette section, nous décrivons les principales fonctionnalités du module `selt`, le *model-checker* pour la logique $SE{-}LTL$ de la boîte à outils `Tina`. `selt` permet de vérifier la satisfaction d'une propriété $SE{-}LTL$ sur une structure de Kripke obtenue comme indiqué ci-dessus depuis un graphe de classes. La vérification comporte deux phases :

1) construire un automate de Büchi acceptant les mots qui ne satisfont pas la formule $SE-LTL$ à vérifier ; cette phase est réalisée de façon transparente pour l'utilisateur en invoquant le logiciel `ltl2ba` développé au LIAFA par Paul Gastin et Denis Oddoux[GAS 01] ;

2) construire la composition de la structure de Kripke obtenue depuis le graphe des classes et de l'automate de Büchi, et rechercher à la volée une composante fortement connexe contenant un état acceptant de l'automate de Büchi. Si aucune telle composante n'est trouvée, alors la formule est satisfaite, sinon elle définit un contre-exemple.

En cas de non-satisfaction d'une formule, `selt` peut fournir une séquence contre-exemple en clair ou sous un format exploitable par le simulateur de `Tina`, afin de pouvoir l'explorer pas à pas. Notons que, dans le cas d'un modèle temporisé, il faut au préalable associer à cette exécution un échéancier temporel. Celui-ci sera calculé de façon transparente par le module `plan` décrit en section 1.6.

Dans certains cas - notamment pour les systèmes temporisés - la séquence contre-exemple d'une formule peut être très longue et donc difficilement exploitable par l'utilisateur. Afin d'en faciliter l'exploitation, `selt` peut aussi produire des contre-exemples sous forme compactée (symbolique). Le contre-exemple est alors présenté comme une suite de séquences, chacune imprimée comme une seule transition, ne matérialisant que les changements d'états «essentiels» pour la compréhension du problème (les changements d'états de l'automate de Büchi).

1.8.3.2. *La logique de* `selt`

Les structures de Kripke étiquetées que nous manipulons ici sont obtenues à partir d'un réseau de Petri. Dans ce contexte, AP correspond à l'ensemble P des places du réseau et Σ à l'ensemble de ses transitions.

Notons que l'interprétation des formules $SE-LTL$ (void définition 4) confond deux états dès lors que, dans ces deux états, les mêmes places sont marquées, indépendamment du nombre de marques dans les places marquées. Si cette approche est exacte dans le cas d'un réseau sauf (pour lequel chaque place est bornée par 1), elle implique dans le cas général une perte d'information significative.

Afin de conserver l'information de multiplicité de marques exprimée par les marquages, `selt` travaille sur des structures de Kripke enrichies, basées sur des multi-ensembles de propriétés atomiques plutôt que des ensembles. La seule différence avec la définition 3 est le remplacement des applications ν et ϵ par des applications ν' et ϵ' qui associent à chaque état de S un multi-ensemble construit sur AP (c'est-à-dire une application de $AP \mapsto \mathbb{N}$, ou encore un marquage, dans notre contexte) et à chaque transition de T un multi-ensemble construit sur Σ, respectivement. Cette dernière possibilité, pour les transitions, n'est pas exploitée dans les exemples que nous présenterons ici, mais elle est utile pour l'analyse des constructions à base de pas couvrants aussi fournies par `tina` (non discutées ici).

A cet enrichissement des structures de Kripke correspond un enrichissement du langage des formules afin d'exploiter au mieux l'information qu'elles contiennent. Pour cela, on va substituer au calcul propositionnel (bi-valué par {true,false}), un calcul propositionnel multi-valué et étendre le langage d'interrogation de `selt` avec des opérateurs logico-arithmétiques.

Pour $p \in AP$, $a \in \Sigma$, $c \in \mathbb{N}$, les formules Φ, propositions r et expressions arithmétiques e de `selt` obéissent à la grammaire suivante :

$$\begin{array}{ll} \Phi ::= r \mid \neg\Phi \mid \Phi \vee \Phi \mid \bigcirc\ \Phi \mid \Box\,\Phi \mid \Diamond\,\Phi \mid \Phi\ U\ \Phi & \\ r ::= e \mid e \Delta e & (\Delta \in \{=, <, \leq, >, \geq\}) \\ e ::= p \mid a \mid c \mid e \nabla e & (\nabla \in \{+, -, *, /\}) \end{array}$$

Pour tout chemin π, l'interprétation $\lceil r \rceil(\pi)$ d'une proposition r et la valeur entière $\langle e \rangle(\pi)$ d'une expression sont définies comme suit :

$$\begin{array}{rcl} \lceil e \rceil(\pi) & = & \langle e \rangle(\pi) \geq 1, \\ \lceil r_1 \Delta r_2 \rceil(\pi) & = & \lceil r_1 \rceil(\pi) \Delta \lceil r_2 \rceil(\pi), \end{array}$$

$$\begin{array}{rcl} \langle c \rangle(\pi) & = & c, \\ \langle p \rangle(\pi) & = & \nu'(s)(p), \text{ où s est le premier état de } \pi, \\ \langle a \rangle(\pi) & = & \epsilon'(t)(p), \text{ où t est la première transition de } \pi, \\ \langle e_1 \nabla e_2 \rangle(\pi) & = & \langle e_1 \rangle(\pi) \nabla \langle e_2 \rangle(\phi). \end{array}$$

La sémantique des formules est définie comme pour $SE{-}LTL$ (voir la définition 4), sauf pour les règles relatives à p et a, remplacées par :

$\pi \models e$ ssi $\lceil e \rceil(\pi)$.

Enfin, `selt` permet la déclaration d'opérateurs dérivés ainsi que la redéclaration des opérateurs existants ; un petit nombre de commandes sont aussi fournies pour contrôler l'impression des résultats ou encore permettre l'utilisation de bibliothèques de formules. Nous renvoyons le lecteur au manuel de l'outil `selt` pour plus de détails.

Exemple 2 Quelques formules de `selt` concernant le réseau figure 1.1

$t1 \wedge p2 \geq 2$	tout chemin commence par $t1$ et $m_0(p2) \geq 2$,
$\Box\,(p2 + p4 + p5 = 2)$	un invariant de marquage linéaire,
$\Box\,(p2 * p4 * p5 = 0)$	un invariant de marquage non linéaire,
$infix\ q\ R\ p = \Box\,(p \Rightarrow \Diamond\ q)$	déclare l'opérateur «répond à», noté R,
$t1\ R\ t5$	$t1$ «répond à» $t5$.

1.9. Quelques exemples d'utilisation de `selt`

1.9.1. *John et Fred*

1.9.1.1. *Énoncé*

Cet exemple [DEC 91] est emprunté au domaine de l'intelligence artificielle et à la problématique de satisfaction de contraintes. L'énoncé est le suivant :

> John se rend à son travail en utilisant sa voiture (la durée de son trajet est alors de 30 à 40 minutes) ou le bus (au moins 60 minutes).
> Fred se rend à son travail en utilisant sa voiture (durée 20 à 30 minutes) ou un système de covoiturage (40 à 50 minutes).
> Aujourd'hui John a quitté son domicile entre 7h10 et 7h20, Fred est arrivé à son travail entre 8h00 et 8h10. De plus John est arrivé 10 à 20 minutes après que Fred ait quitté sa maison.

Il s'agit de répondre à trois questions :

1) Les contraintes temporelles figurant dans ce scénario sont-elles consistantes ?

2) Est-il possible que Fred ait pris le bus et John utilisé le covoiturage ?

3) Dans quelle plage horaire Fred a pu partir ?

La modélisation du problème en réseau temporel est donnée figure 1.6. Les parties non grisées correspondent aux comportements respectifs habituels de John et de Fred. La partie grisée spécifie deux «observateurs» mis en place pour prendre en compte les contraintes temporelles spécifiques au scénario considéré. Sur la composante centrale sont représentées les contraintes absolues : heure de départ de John, heure d'arrivée de Fred. La contrainte relative stipulant que «John est arrivé entre 10 et 20 minutes après que Fred ait quitté sa maison» est représentée par un second observateur qui complète le réseau de Fred en déclenchant un «chronomètre» après que Fred ait quitté sa maison.

1.9.1.2. *Les contraintes temporelles figurant dans ce scénario sont-elles consistantes ?*

Le graphe de classes de ce réseau, calculé par l'algorithme 1 exécuté par `tina`, admet 3676 classes et 7578 transitions. Comme ce graphe est sans circuit, il suffit de s'assurer de l'existence d'une exécution permettant à John et Fred de travailler et satisfaisant les contraintes exprimées, autrement dit satisfaisant la formule ϕ_1 suivante :

$\phi_1 = \Diamond\,(john\_left \wedge 7h10\_7h20)$	John est parti entre 7h10 et 7h20
$\wedge\ \Diamond\,(fred\_arrives \wedge 8h00\_8h10)$	Fred est arrivé entre 8h et 8h10
$\wedge\ \Diamond\,(john\_arrives \wedge 10\_20)$	John arrive 10 à 20 minutes après le départ de Fred
$\wedge\ \Diamond\,(john\_working \wedge fred\_working)$	

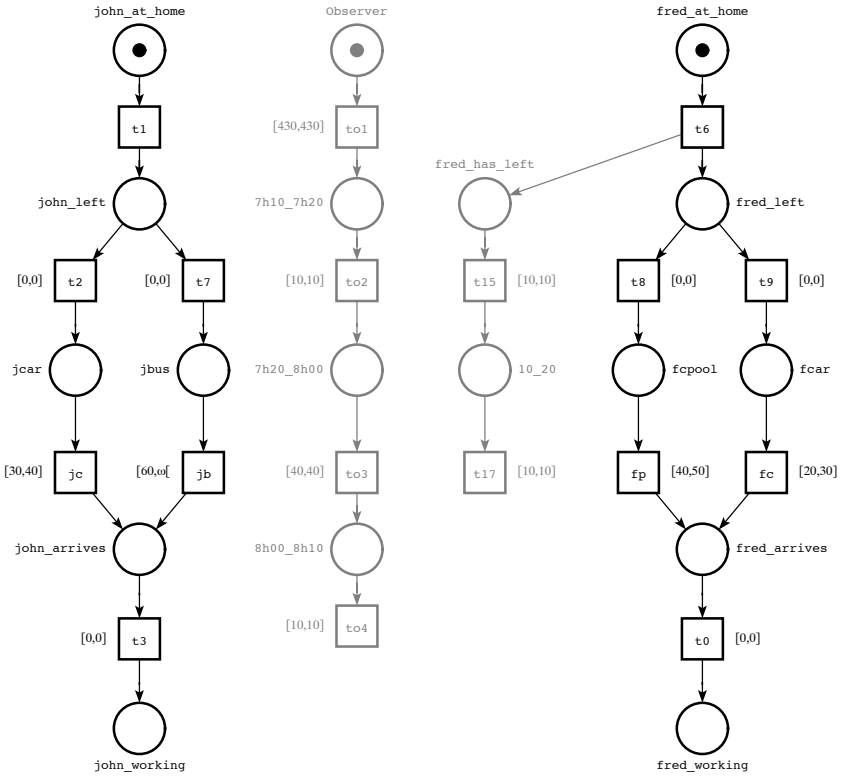


Figure 1.6. *Modélisation en réseau temporel de John et Fred.*

Comme LTL évalue les propriétés sur toutes les exécutions possibles, on considère la formule $\neg\phi_1$ et on la soumet à `selt`. La réponse est négative : toutes les exécutions ne satisfont pas $\neg\phi_1$, donc l'une au moins satisfait sa négation (ϕ_1). `selt` fournit simultanément un contre-exemple de $\neg\phi_1$, c'est-à-dire une séquence démontrant que le scénario est consistant. Dans le contre-exemple fourni, John a utilisé sa voiture et Fred le covoiturage. L'outil `plan` discuté en section 1.6 permet de trouver un échéancier, ou tous, ayant cette séquence pour support. La séquence exhibée est la suivante :

```
       state 0: Observer fred_at_home john_at_home
-t1->  state 1: Observer fred_at_home john_left
-to1-> state 1694: fred_at_home john_left 7h10_7h20
-t2->  state 1883: fred_at_home jcar 7h10_7h20
-t6->  state 1884: fred_has_left fred_left jcar 7h10_7h20
```

```
-t8->  state 1885: fcpool fred_has_left jcar 7h10_7h20
-to2-> state 1886: fcpool fred_has_left jcar 7h20_8h00
-t15-> state 1887: 10_20 fcpool jcar 7h20_8h00
-jc->  state 1888: 10_20 fcpool john_arrives 7h20_8h00
-t17-> state 1901: fcpool john_arrives 7h20_8h00
-t3->  state 1902: fcpool john_working 7h20_8h00
-to3-> state 1903: fcpool john_working 8h00_8h10
-fp->  state 1904: fred_arrives john_working 8h00_8h10
-t0->  state 1909: fred_working john_working 8h00_8h10
-to4-> state 1910: dead fred_working john_working
```

1.9.1.3. *Est-il possible que Fred ait pris le bus et que John ait utilisé le covoiturage ?*

Il suffit d'affiner la formule ϕ_1 précédente pour s'assurer que Fred a pris le bus et John le covoiturage. Un tel scénario est possible si une exécution au moins satisfait $\phi_2 = \phi_1 \wedge (\Diamond\, jbus \wedge \Diamond\, fcpool)$.

À la question $\neg\phi_2$, `selt` répond vrai. Aucune exécution ne satisfait donc ϕ_2 ; il n'est donc pas possible de satisfaire les contraintes temporelles lorsque Fred part en Bus et John en covoiturage.

1.9.1.4. *A quels horaires Fred a pu partir ?*

Contrairement aux questions précédentes, celle-ci n'appelle pas une simple réponse booléenne mais doit fournir une plage horaire pour le départ de Fred. Il ne s'agit plus dans ce cas d'effectuer une vérification mais une activité de synthèse.

selt apportera aisément une réponse à la question de savoir si Fred a pu partir dans un intervalle de dates donné. Il suffit pour cela d'associer cet intervalle à la transition $t6$ du réseau (matérialisant le départ de Fred) et de vérifier par la méthode précédente qu'il existe une exécution consistante. Mais dans ce cas, la plage horaire a été proposée et il a suffi de vérifier que les nouvelles contraintes sont satisfaisables.

Toutefois, l'outil `plan` permet de répondre partiellement à ce problème de synthèse. Pour une séquence donnée satisfaisant les contraintes temporelles, `plan` peut déterminer les dates de départ possibles de Fred, mais pas pour toutes ces séquences en une seule invocation. Pour avoir toutes ces dates possibles, il faut invoquer l'outil `plan` pour toutes les séquences consistantes possibles ; le résultat recherché est la réunion des ensembles de dates calculées. Ces séquences peuvent être obtenues *via* `selt`, en construisant le graphe de synchronisation en totalité (incluant donc tous les contre-exemples).

1.9.2. *Le protocole du bit alterné*

Les protocoles de communication font un large usage de contraintes temporelles : les mécanismes de reconfiguration pour la perte de messages, par exemple, sont typiquement implantés à l'aide de temporisations. Le Protocole du Bit Alterné est certainement le plus simple de ces protocoles. Il s'agit d'un protocole de transfert de données du type envoi-attente : avant d'émettre à nouveau un message, le processus émetteur attend l'arrivée de l'accusé de réception du message qu'il a précédemment envoyé.

Les hypothèses sur le fonctionnement du médium de communication sont que les messages ou accusés de réception peuvent être perdus ou endommagés (dans ce dernier cas, ils sont simplement rejetés). Pour corriger les pertes de messages et/ou d'accusés, une temporisation est lancée lorsqu'un message est émis. Si l'accusé de réception du message n'arrive pas avant l'expiration de la temporisation, le message est retransmis.

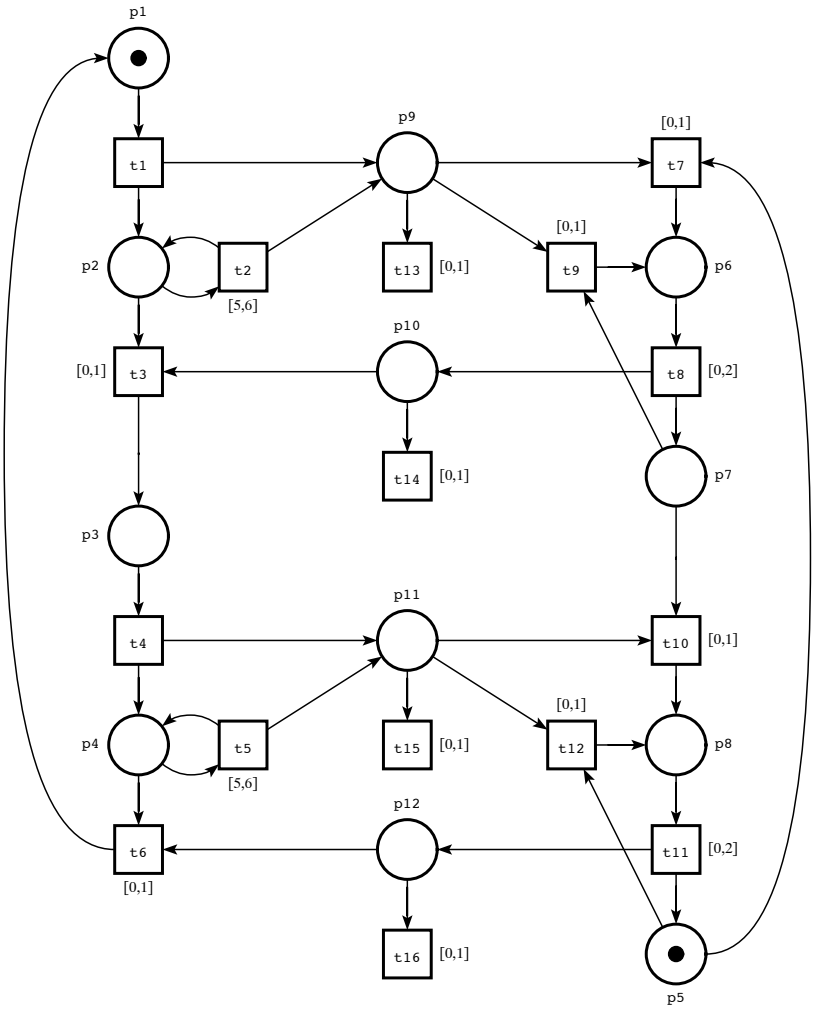
Le protocole du bit alterné peut être représenté par le réseau temporel de la figure 1.7. Par souci de simplification, les messages endommagés sont assimilés à des messages perdus. Notons que, dans ce réseau, les pertes de messages ou d'accusés sont simplement représentées par des transitions qui n'ont pas de place de sortie. Afin de produire la spécification, des estimations pour la durée de toutes les opérations élémentaires du protocole doivent être fournies. La temporisation de retransmission doit être choisie suffisamment longue pour que le médium de transmission ne contienne au plus qu'un message ou accusé ; elle se produira au bout d'un temps compris entre 5 et 6 unités de temps après émission de sa dernière copie. Des estimations identiques sont données pour la perte et la réception de messages ou d'accusés de réception (entre 0 et 1 unité). Par contre, aucune contrainte de date n'est donnée pour l'émission de la première copie de chaque message ; les transitions correspondantes portent implicitement l'intervalle $[0, \infty[$.

Ce réseau est borné. Son graphe de classes, construit par `tina` en utilisant l'algorithme 1 admet seize classes d'états.

Intéressons-nous à quelques propriétés sur ce réseau, dans le langage de `selt` :

$\phi_1 = \Box\,(p_9 + p_{10} + p_{11} + p_{12} \leq 1)$
$\phi_2 = t1$
$\phi_3 = \Box\,(t1 \Rightarrow \Diamond\, t7)$
$\phi_4 = \Box\,(t1 \Rightarrow \Diamond\,(t13 \vee t7))$
$\phi_5 = -\Box\,\Diamond\,(t13 \vee t14) \Rightarrow \Box\,(t1 \Rightarrow \Diamond\, t3)$
$\phi_6 = \Box\ -dead$

ϕ_1 exprime qu'un message ou accusé au plus est en transit à chaque instant, ce qui garantit que le délai de retransmission est correctement choisi. ϕ_1 est vérifiée.



t_1	:	émission paquet 0	t_9	:	rejet paquet 0 dupliqué
t_2	:	réemission paquet 0	t_{10}	:	acceptation paquet 1
t_3	:	réception accusé 0	t_{11}	:	émission accusé 1
t_4	:	émission paquet 1	t_{12}	:	rejet paquet 1 dupliqué
t_5	:	réemission paquet 1	t_{13}	:	perte paquet 0
t_6	:	réception accusé 1	t_{14}	:	perte accusé 0
t_7	:	acceptation paquet 0	t_{15}	:	perte paquet 1
t_8	:	émission accusé 0	t_{16}	:	perte accusé 1

Figure 1.7. *Un réseau temporel pour le Protocole du Bit Alterné.*

ϕ_2 exprime que toute exécution commence par la transition $t1$. Elle s'avère fausse : n'étant pas temporellement contrainte, $t1$ peut être retardée indéfiniment ; l'état initial est temporellement divergent. L'outil `tina` laisse le choix à l'utilisateur d'interpréter les transitions non temporellement contraintes comme pouvant, ou non, être infiniment retardées. Par défaut, la première interprétation est retenue.

ϕ_3 exprime que tout message émis est reçu. Comme tout message peut être perdu puis retransmis une infinité de fois, elle n'est pas satisfaite. Le contre-exemple, montré ci-dessous, exhibe une séquence contenant une circuit passant par l'état 16 et dans lequel le message est perdu (t13) puis retransmis (t2). Par contre, tout message émis est soit reçu, soit perdu, comme exprimé par ϕ_4, qui s'évalue à vrai.

```
       state 0: L.div p1 p5
-t1->  state 16: p2 p5 p9
-t13-> state 17: p2 p5
-t2->  state 16: p2 p5 p9
```

ϕ_5 exprime que l'accusé de réception de tout message émis est reçu, dès lors que ni le message ni son accusé de réception n'ont été perdus une infinité de fois. ϕ_5 est vérifiée.

Enfin, la propriété ϕ_6 exprime l'absence de blocage, elle s'évalue à vrai. Comme déjà expliqué, les blocages sont représentés dans les systèmes de transition de Kripke utilisés par `selt` par une propriété d'état spécifique (*dead*).

1.10. Conclusion

Les constructions exposées dans cet article permettent d'abstraire de façon finie le comportement, généralement infini, des réseaux temporels bornés, en préservant certaines relations avec le graphe d'états du réseau temporel dont la plus riche est la bissimulation. Disposer d'une représentation finie de cet espace infini d'états permet de mettre en œuvre les techniques de vérification de modèles ; dans ce chapitre nous avons illustré cette possibilité dans le cadre de la logique temporelle LTL. La méthode «classique» du graphe des classes a donné lieu à de nombreux travaux, académiques ou industriels, concernant le matériel ou le logiciel et a été intégrée à plusieurs autres outils d'analyse, notamment [GAR 05]. L'ensemble des techniques présenté dans ce chapitre est supporté par l'environnement `Tina` [BER 04] qui offre des fonctionnalités d'édition, d'exploration et d'analyse de réseaux de Petri temporels.

Les limites intrinsèques de ces méthodes ne doivent pas être perdues de vue. Une première est qu'il ne peut être énoncé de condition nécessaire et suffisante pour la

propriété borné pour les réseaux temporels, mais il existe un bon nombre de conditions suffisantes, comportementales ou structurelles. Une seconde est que le nombre de classes d'états d'un réseau temporel peut être très grand. Ce nombre dépend de façon difficilement prévisible de l'interaction entre la structure de l'espace des marquages du réseau de Petri sous-jacent et les contraintes temporelles associées aux transitions. Il est essentiel donc de porter un effort sur les techniques de vérification incrémentielles ou limitant l'explosion combinatoire [RIB 05].

Les travaux en cours portent sur l'enrichissement du modèle de description. Deux aspects sont plus particulièrement retenus : la prise en compte des données et la possibilité de représenter les mécanismes de suspension/reprise d'activités pour permettre la modélisation et la vérification de systèmes régis par un ordonnancement préemptif [BER 05].

D'autre part, de nombreux projets (tels que Topcased [FAR 04] ou Cotre [BER 03a] auxquels nous participons ou avons participé) ont montré l'intérêt et la volonté des industriels d'intégrer les «techniques de description et de vérification formelles» (TDF) dans leur processus de conception. Ceci constitue un second axe prospectif de notre travail. L'intégration des TDF dans un atelier de développement industriel nécessite des travaux théoriques (passage à l'échelle), logiciels (efficacité/ergonomie des outils supports) et collaboratifs - en partenariat direct avec les utilisateurs finaux de l'atelier logiciel - pour réduire la distance entre les «standards métier» en vigueur dans l'industrie et les formalismes cibles pour la vérification.

1.11. Bibliographie

[ARN 94] ARNOLD A., BEGAY D., CRUBILLÉ P., *Construction and analysis of transition systems with MEC*, Word Scientific Publishing Co, Singapour, 1994.

[BER 83] BERTHOMIEU B., MENASCHE M., « An Enumerative Approach for Analyzing Time Petri Nets. », *IFIP Congress Series*, vol. 9, p. 41–46, Elsevier Science Publ. Comp. (North Holland), 1983.

[BER 91] BERTHOMIEU B., DIAZ M., « Modeling and Verification of Time Dependent Systems Using Time Petri Nets. », *IEEE Transactions on Software Engineering*, vol. 17, n° 3, p. 259–273, mars 1991.

[BER 01] BERTHOMIEU B., « La méthode des classes d'états pour l'analyse des réseaux Temporels – Mise en œuvre, Extension à la multi-sensibilisation », *Proc. Modélisation des Systèmes Réactifs*, Toulouse, France, octobre 2001.

[BER 03a] BERTHOMIEU B., RIBET P.-O., VERNADAT F., BERNARTT J., FARINES J.-M., BODEVEIX J.-P., FILALI M., PADIOU G., MICHEL P., FARAIL P., GAUFFILET P., DISSAUX P., LAMBERT J., « Towards the verification of real-time systems in avionics : the cotre approach », *Workshop on Formal Methods for Industrial Critical Systems (FMICS'2003)*, p. 201–216, 2003.

[BER 03b] BERTHOMIEU B., VERNADAT F., « State Class Constructions for Branching Analysis of Time Petri Nets », *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003), Warsaw, Poland, Springer LNCS 2619*, p. 442-457, 2003.

[BER 04] BERTHOMIEU B., RIBET P.-O., VERNADAT F., « The tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets », *International Journal of Production Research*, vol. 42, n° 14, p. 2741–2756, 15 Juillet 2004.

[BER 05] BERTHOMIEU B., LIME D., ROUX O. H., VERNADAT F., « Modélisation des Systèmes Réactifs (MSR'05) », *Journal Européen des Systèmes Automatisés*, vol. 39 (1-2-3), p. 223–238, 2005.

[BOU 04] BOUCHENEB H., HADJIDJ R., « Towards optimal CTL^* model checking of Time Petri nets », *Proceedings of 7th Workshop on Discrete Events Systems*, Reims, France, septembre 2004.

[CHA 04] CHAKI S., E M., CLARKE, OUAKNINE J., SHARYGINA N., SINHA N., « State/Event-based Software Model Checking », *4th International Conference on Integrated Formal Methods (IFM'04), Springer LNCS 2999*, p. 128–147, avril 2004.

[DEC 91] DECHTER R., MEIRI I., PEARL J., « Temporal constraint networks », *Artificial Intelligence*, vol. 49 (1-3), p. 61–95, 1991.

[FAR 04] FARAIL P., GAUFILLET P., FILALI M., MICHEL P., VERNADAT F., « Vérifications dans un AGL orienté modèles », *Génie Logiciel*, vol. 69B, p. 51-55, 2004.

[FER 96] FERNANDEZ J.-C., GARAVEL H., MATEESCU R., MOUNIER L., SIGHIREANU M., « Cadp, a protocol validation and verification toolbox », *8th Conference Computer-Aided Verification, CAV'2001, Springer LNCS 1102*, p. 437–440, juillet 1996.

[GAR 05] GARDEY G., LIME D., MAGNIN M., ROUX O. H., « Roméo : A Tool for Analyzing time Petri nets », *17th International Conference on Computer Aided Verification, CAV'05, Springer LNCS 3576*, juillet 2005.

[GAS 01] GASTIN P., ODDOUX D., « Fast LTL to Buchi Automata Translation », *13th Conference Computer-Aided Verification, CAV'2001, Springer LNCS 2102*, p. 53–65, juillet 2001.

[JON 77] JONES N. D., LANDWEBER L. H., LIEN Y. E., « Complexity of Some Problems in Petri Nets. », *Theoretical Computer Science 4*, p. 277–299, 1977.

[KAN 90] KANELLAKIS P. K., SMOLKA S. A., « CCS Expressions, Finite State Processes, and Three Problems of Equivalence », *Information and Computation*, vol. 86, p. 43–68, 1990.

[KAR 69] KARP R. M., MILLER R. E., « Parallel Program Schemata. », *Journal of Computer and System Sciences 3*, , n° 2, p. 147–195, mai 1969.

[MER 74] MERLIN P. M., *A Study of the Recoverability of Computing Systems*., Irvine : Univ. California, Thèse de doctorat, 1974.

[PAI 87] PAIGE P., TARJAN R. E., « Three Partition Refinement Algorithms », *SIAM Journal on Computing*, vol. 16, n° 6, p. 973–989, 1987.

[PEN 01] PENCZEK W., PÓŁROLA A., « Abstraction and partial order reductions for checking branching properties of Time Petri Nets », *Proc. 22st International Conference on Application and Theory of Petri Nets (ICATPN 2001), Springer LNCS 2075*, p. 323–342, 2001.

[RIB 05] RIBET P.-O., Vérification formelle de systèmes. Contribution à la réduction de l'explosion combinatoire, Rapport, Thèse de doctorat de l'Institut National des Sciences Appliquées, Toulouse (Rapport LAAS/CNRS 05631), juillet 2005.

[YON 98] YONEDA T., RYUBA H., « CTL Model Checking of Time Petri nets Using Geometric Regions », *IEEE Transactions on Information and Systems*, vol. E99-D, n° 3, p. 1-10, 1998.