Une algèbre de processus avec suspension ...

Jérôme Ermont - Frédéric Boniol

ONERA - CERT - DTIM,
2 avenue Edouard Belin - BP4025,
31055 Toulouse Cedex - France
{ermont,boniol}@cert.fr

Résumé Cet article présente une algèbre de processus temporisés appelée TPAPa et une extension de celle-ci, TPAPas. Le but de ces algèbres est de modéliser un ensemble de processus embarqués se partageant des ressources communes et sensibles aux délais de communication et aux stratégies de séquencement. La communication par diffusion temporisée et la suspension sans reprise (abort) sont les notions fondamentales de TPAPa. Elles permettent la définition de séquenceurs et de médiums de communication asynchrones qui seront pris en compte lors de la vérification du comportement temps-réel du système. Après avoir défini les caractéristiques de l'algèbre TPAPa, une étude de cas du domaine avionique est modélisée à l'aide de TPAPa puis vérifiée à l'aide du model checker UPPAAL. Une amélioration de TPAPa par un opérateur de suspension de processus avec reprise est ensuite proposée et formalisée dans l'algèbre TPAPas. Les processus de TPAPas sont équivalents à des automates non plus temporisés mais hybrides et ne peuvent donc plus être vérifiés à l'aide de model checker tels que UPPAAL.

1 Introduction

La modélisation et la maîtrise des systèmes embarqués sont devenues un défi de plus en plus important avec l'arrivée de nouvelles générations d'avions. Leur nature critique, temps-réel, distribuée et intégrée rend ces systèmes complexes. Pour surmonter cette complexité les constructeurs d'avions ont choisi de construire les systèmes avioniques sur le principe d'architecture fédérée : les systèmes avioniques des avions modernes (tels que les Airbus A340/A330, le Boeing B777, l'EuroFighter, ou le Rafale...) sont composés d'un ensemble de fonctions s'exécutant en parallèle, communicant à l'aide de bus multiplexés, et partageant un ensemble de calculateurs. L'architecture fonctionnelle de tels systèmes peut être décrite par le terme général $S = (S_1 \| S_2 \cdots \| S_n)$. Afin d'éviter toute défaillance catastrophique, les communications entre les fonctions avioniques $S_1 \dots S_n$ sont fondées sur la diffusion (c'est-à-dire transmission non bloquante de 1 vers n) à travers des canaux asynchrones partagés. Malheureusement de tels canaux introduisent des délais non déterministes (mais nécessairement bornés). De plus, les fonctions avioniques peuvent partager des calculateurs communs. Le temps sur chaque calculateur, dans le cas des systèmes embarqués, sera divisé en tranches de temps contrôlées par des séquenceurs (schedulers) sur chaque calculateur. Les mécanismes fondamentaux sous-jacents sont alors l'activation de processus et la préemption (abandon ou suspension de fonction). Cependant la préemption peut introduire des délais non déterministes. En conséquence, tenir compte de l'architecture fonctionnelle du système est nécessaire mais non suffisant lorsque l'on souhaite étudier le comportement du système global. Il faut aussi considérer les stratégies de séquencement exécutées par chaque calculateur et les délais de communication à travers les canaux de communication.

L'objectif de cet article est de présenter une algèbre de processus temps-réel permettant de modéliser à la fois le séquencement de processus partageant des ressources communes et des canaux de communication à latence variable. L'intérêt est ainsi de permettre indirectement la ressource partagée dans la modélisation et la vérification.

La plupart des algèbres de processus temps-réel (ATP [Nic94], ARTS [D'A99], Open Timed CCS [Fec01]...) capture les délais dus à la synchronisation des processus ou aux actions, mais font souvent abstraction des ressources opératoires en présupposant un environnement de fonctionnement idéal. En conséquence, la modélisation et l'analyse de systèmes sensibles aux stratégies de partage de ressources (et donc des systèmes temps réel en général) ne peuvent être réalisés dans de telles algèbres.

Une exception est cependant à noter par l'algèbre de processus ACSR [BGL97] définie par P. Bremond-Gregoire et I. Lee pour spécifier le partage de ressources, la préemption, les priorités et l'exception. Tandis que dans les algèbres classiques, la préemption est due uniquement à un timeout, en ACSR, elle s'effectue en fonction de la priorité d'accès à la ressource. Ainsi si P et Q sont deux processus utilisant la ressource r avec une priorité respective de 2 et 3, alors Q préempte P et s'exécute sur r. A la différence de ACSR, le modèle de notre algèbre est fondé sur le principe qu'un système embarqué critique consiste souvent en un assemblage de processus communicants pouvant être préemptés à l'arrivée d'un événement. Le mécanisme de préemption mis en œuvre est très proche de l'instruction "abort" du langage réactif synchrone ESTEREL [Ber93].

ACSR ne permet cependant pas la suspension avec reprise (instruction "suspend" de ESTEREL). L'algèbre de processus TPAPas, en plus de propriétés de l'algèbre TPAPa, possède ce type de préemption. L'opérateur réalisant cette suspension s'apparente au processus $[\mathsf{A} \uparrow @\mathsf{x} : \mathsf{B}]$ du langage asynchrone non-temporisé ELECTRE [ER86]. Pour ce processus, le module A est préempté sur l'événement x, B s'exécute alors. Puis, dès que B termine, le module A s'exécute à nouveau à partir de son point de suspension. Dans TPAPas, lorsqu'un processus est suspendu sur un événement, son exécution est reprise sur un autre événement à partir de son point de suspension. Le partage des ressources sera modélisé alors par la préemption de processus, et la communication sera effectuée à l'aide d'actions de diffusion non bloquantes et de délais. L'exécution d'un processus sera alors soumise aux interruptions provenant d'un séquenceur contrôlant la ressource qui utilise ce processus, c'est-à-dire soumise à la disponibilité des ressources.

L'article est organisé comme suit. La section 2 décrit la syntaxe et la sémantique de l'algèbre TPAPa, algèbre possédant uniquement un opérateur de suspension sans reprise. Cette section présente aussi une étude de cas provenant du monde avionique formalisé à l'aide de l'algèbre TPAPa puis vérifié à l'aide du model checker UPPAAL [LPY97, upp]. La section 3 présente la syntaxe et la sémantique de l'algèbre TPAPas. Cette algèbre est une extension de l'algèbre TPAPa munie d'un opérateur de suspension avec reprise.

2 Une algèbre de processus avec opérateur d'abandon : TPAPa

2.1 Syntaxe de l'algèbre

Soit $\mathcal{A}$ l'ensemble des actions urgentes représentant les événements d'émission. Les éléments de $\mathcal{A}$ sont notés $\mathsf{a}, \mathsf{b}, \mathsf{c}$ ... L'ensemble $\overline{\mathcal{A}}$ est défini par $\overline{\mathcal{A}} = \{\overline{\mathsf{a}} | \mathsf{a} \in \mathcal{A}\}$. Les éléments de $\overline{\mathcal{A}}$ représentent les événements de réception des éléments de $\mathcal{A}$. L'action invisible ε telle que $\varepsilon \notin \mathcal{A} \cup \overline{\mathcal{A}}$ est utilisée pour interrompre un délai. L'ensemble de variables $\mathcal{V} = \{\mathsf{X}, \mathsf{Y}, \cdots\}$ est nécessaire à la définition de la récursion. Le domaine temporel de l'algèbre est l'ensemble des réels positifs noté $\mathcal{T}$.

La grammaire suivante permet de définir les expressions des éléments de l'algèbre $\mathcal{P}$, notés $\mathsf{P}, \mathsf{Q}, \cdots$:

$$\mathsf{P} ::= \delta \mid \mathsf{X} \mid \mathsf{aP} \mid \overline{\mathsf{a}}\mathsf{P} \mid \mathcal{I}\mathsf{P} \mid \mathsf{P} + \mathsf{P} \mid \mathsf{P} \| \mathsf{P} \mid \mathsf{rec}\mathsf{X}.\mathsf{P} \mid (\mathsf{P} \Downarrow \overline{\mathsf{a}})\mathsf{P}$$

pour laquelle :

- δ est le processus qui ne fait rien sauf laisser passer le temps.
- aP est le processus qui effectue l'action urgente a (émission de l'événement a) puis se comporte comme P.
- $\overline{\mathsf{a}}\mathsf{P}$ est le processus qui attend l'évènement a, puis qui, sur réception de a, se comporte comme P. Tant que a n'est pas reçu, $\overline{\mathsf{a}}\mathsf{P}$ laisse passer le temps. $\overline{\mathsf{a}}$ se comporte ainsi comme l'instruction "await a" du langage ESTEREL et l'événement fugace $@\mathsf{a}$ du langage ELECTRE.
- $\mathsf{P} \| \mathsf{Q}$ peut soit effectuer des actions de P et de Q indépendemment, soit synchroniser ces deux processus sur des actions complémentaires (comme a ou $\overline{\mathsf{a}}$), soit laisser passer du temps si P et Q peuvent laisser passer du temps.
- Le processus $\mathsf{P} + \mathsf{Q}$ représente un choix entre P et Q. Ce choix n'est pas seulement effectué par les actions mais aussi par le temps. Par exemple, si le temps peut progresser plus dans P qu'il ne pourrait le faire dans Q alors le choix $\mathsf{P} + \mathsf{Q}$ peut se comporter comme P.
- Le symbole $\mathcal{I}$ correspond à un intervalle de temps fermé représentant un délai non-déterministe dont les bornes appartiennent à $\mathcal{T} \cup \{\infty\}$. Lorsque la borne supérieure de l'intervalle est non nulle, le processus $\mathcal{I}\mathsf{P}$ peut laisser passer t unités de temps à condition que t soit inférieur (ou égal) à cette borne supérieure. De plus, lorsque sa borne inférieure est nulle, le processus $\mathcal{I}\mathsf{P}$ peut exécuter l'action invisible ε (interruption d'un délai) et devenir P.
- Le processus $(\mathsf{P} \Downarrow \overline{\mathsf{a}})\mathsf{Q}$ permet de modéliser l'abandon du processus P au profit du processus Q lorsque l'événement a est reçu. Tant que cet événement n'est pas reçu, ce processus se comporte comme P. $(\mathsf{P} \Downarrow \overline{\mathsf{a}})$ se comporte comme l'instruction ESTEREL "abort P when a". Soit $\mathsf{R} = (\mathsf{P} \Downarrow \overline{\mathsf{a}})\mathsf{Q}$, P est appelé sous-terme preempté de R.
- Finalement, le processus $\mathsf{rec}\mathsf{X}.\mathsf{P}$ est la définition classique de la récursion permettant de spécifier des comportements infinis.

Définition 1 *(Variables libres)*
Soit $\mathsf{P} \in \mathcal{P}$. L'ensemble des variables libres de P noté $\mathsf{free}(\mathsf{P})$ est défini par induction

sur $\mathcal{P}$:

$$
\begin{array}{lcl}
\mathsf{free}(\delta) & = & \emptyset \\
\mathsf{free}(aP) & = & \mathsf{free}(P) \\
\mathsf{free}(\overline{a}P) & = & \mathsf{free}(P) \\
\mathsf{free}(\mathcal{I}P) & = & \mathsf{free}(P) \\
\mathsf{free}(P+Q) & = & \mathsf{free}(P) \cup \mathsf{free}(Q) \\
\mathsf{free}(P\|Q) & = & \mathsf{free}(P) \cup \mathsf{free}(Q) \\
\mathsf{free}((P \Downarrow \overline{a})Q) & = & \mathsf{free}(P) \cup \mathsf{free}(Q) \\
\mathsf{free}(X) & = & \{X\} \\
\mathsf{free}(\mathsf{rec}X.P) & = & \mathsf{free}(P) \backslash \{X\}
\end{array}
$$

Une variable X d'un terme P de $\mathcal{P}$ est dite libre si et seulement si elle n'est pas contenue dans un sous-terme de P de type $\mathsf{rec}X.P'$. Par exemple, pour le terme $P = \mathsf{rec}X.aY + X$, Y est libre dans P mais X ne l'est pas. Un terme $P \in \mathcal{P}$ est clos si et seulement toutes ses variables sont libres.

On appelle alphabet d'un processus de TPAPa l'ensemble des événements produits par ce processus.

Définition 2 *(Alphabet d'un processus)*
Soit $P \in \mathcal{P}$. L'alphabet $\Sigma(P)$ est l'ensemble des actions effectuées par le processus P. $\Sigma(P)$ est défini par induction structurelle

$$
\begin{array}{lcl}
\Sigma(\delta) & = & \emptyset \\
\Sigma(aP) & = & \{a\} \cup \Sigma(P) \\
\Sigma(\overline{a}P) & = & \Sigma(P) \\
\Sigma(\mathcal{I}P) & = & \Sigma(P) \\
\Sigma(P+Q) & = & \Sigma(P) \cup \Sigma(Q) \\
\Sigma(P\|Q) & = & \Sigma(P) \cup \Sigma(Q) \\
\Sigma((P \Downarrow \overline{a})Q) & = & \Sigma(P) \cup \Sigma(Q) \\
\Sigma(X) & = & \emptyset \\
\Sigma(\mathsf{rec}X.P) & = & \Sigma(P)
\end{array}
$$

Un terme $P \in \mathcal{P}$ est régulier si et seulement si tous ses sous-termes parallèles et préemptés sont clos et tous les sous-termes du type $(R \Downarrow \overline{a})Q$ sont tels que $a \notin \Sigma(R)$, c'est-à-dire ne peut jamais émettre l'événement a [1]. Par exemple, $P = \mathsf{rec}X.(aX\|b\delta)$ et $Q = (\mathsf{rec}X.aX \Downarrow \overline{a})\delta$ ne sont pas réguliers. Par contre $R = \mathsf{rec}X.((\mathsf{rec}Y.aY \Downarrow \overline{b})cX$ est régulier.

Afin de pouvoir définir la sémantique de l'algèbre les opérateurs **Sup** et **Inf** ont été définis. $\mathbf{Sup}(P) = N$ signifie que le processus P peut laisser passer au plus N unités de temps tandis que $\mathbf{Inf}(P) = n$ signifie que le processus P doit laisser passer au moins n unités de temps.

Définition 3 *(Bornes supérieures et inférieures)*
Soit $\mathbf{Sup} : \mathcal{P} \to \mathcal{T} \cup \{\infty\}$ la borne temporelle supérieure d'un processus définie par induction :

$$
\begin{array}{ll}
\mathbf{Sup}(\delta) = \infty & \mathbf{Sup}(\mathcal{I}P) = p \ \mathsf{si}\ \mathcal{I} = [m,p] \\
\mathbf{Sup}(aP) = 0 & \mathbf{Sup}(\overline{a}P) = \infty \\
\mathbf{Sup}(X) = 0 & \mathbf{Sup}(\mathsf{rec}X.P) = \mathbf{Sup}(P) \\
\mathbf{Sup}(P+Q) = \max(\mathbf{Sup}(P), \mathbf{Sup}(Q)) & \mathbf{Sup}(P\|Q) = \min(\mathbf{Sup}(P), \mathbf{Sup}(Q)) \\
\mathbf{Sup}((P \Downarrow \overline{a})Q) = \mathbf{Sup}(P) &
\end{array}
$$

[1] Cette restriction est identique à celle de l'instruction ESTEREL "abort" qui interdit à un processus de déclencher lui-même son interruption.

Soit $\mathbf{Inf} : \mathcal{P} \to \mathcal{T} \cup \{\infty\}$ *la borne temporelle inférieure d'un processus définie par induction :*

$$\begin{array}{ll}
\mathbf{Inf}(\delta) = \infty & \mathbf{Inf}(\mathcal{I}P) = m \text{ si } \mathcal{I} = [m, p] \\
\mathbf{Inf}(aP) = 0 & \mathbf{Inf}(\overline{a}P) = 0 \\
\mathbf{Inf}(X) = 0 & \mathbf{Inf}(recX.P) = \mathbf{Inf}(P) \\
\mathbf{Inf}(P + Q) = \min(\mathbf{Inf}(P), \mathbf{Inf}(Q)) & \mathbf{Inf}(P \| Q) = \min(\mathbf{Inf}(P), \mathbf{Inf}(Q)) \\
\mathbf{Inf}((P \Downarrow \overline{a})Q) = \mathbf{Inf}(P) &
\end{array}$$

Les systèmes à modéliser ne doivent pas bloquer le temps. C'est pourquoi la notion de processus fortement bien temporisés a été définie. Elle consiste à ce que tous les processus doivent exécuter un délai minimum non nul, c'est-à-dire posséder au moins un intervalle $\mathcal{I}$ ayant une borne inférieure non nulle dans tout sous-terme récursif de P. Ainsi afin de savoir si un processus est fortement bien temporisé ou non, le prédicat fbt a été défini.

Définition 4 *Soit* $\mathsf{fbt} : \mathcal{P} \times 2^{\mathcal{V} \times \{\mathbf{tt}, \mathbf{ff}\}}$ *le prédicat défini par induction :*

$$\begin{array}{lcl}
\mathsf{fbt}(\delta, C_{\mathcal{V},B}) & & \\
\mathsf{fbt}(aP, C_{\mathcal{V},B}) & \Leftrightarrow & \mathsf{fbt}(P, C_{\mathcal{V},B}) \\
\mathsf{fbt}(\overline{a}P, C_{\mathcal{V},B}) & \Leftrightarrow & \mathsf{fbt}(P, C_{\mathcal{V},B}) \\
\mathsf{fbt}(\mathcal{I}P, C_{\mathcal{V},B}) & \Leftrightarrow & (\mathbf{Inf}(\mathcal{I}P) > 0 \wedge \mathsf{fbt}(P, C_{\mathcal{V},\mathbf{tt}})) \vee \mathsf{fbt}(P, C_{\mathcal{V},B}) \\
\mathsf{fbt}(P + Q, C_{\mathcal{V},B}) & \Leftrightarrow & \mathsf{fbt}(P, C_{\mathcal{V},B}) \wedge \mathsf{fbt}(Q, C_{\mathcal{V},B}) \\
\mathsf{fbt}(P \| Q, C_{\mathcal{V},B}) & \Leftrightarrow & \mathsf{fbt}(P, C_{\mathcal{V},B}) \wedge \mathsf{fbt}(Q, C_{\mathcal{V},B}) \\
\mathsf{fbt}((P \Downarrow \overline{a})Q, C_{\mathcal{V},B}) & \Leftrightarrow & \mathsf{fbt}(P, C_{\mathcal{V},B}) \wedge \mathsf{fbt}(Q, C_{\mathcal{V},B}) \\
\mathsf{fbt}(recX.P, C_{\mathcal{V},B}) & \Leftrightarrow & \mathsf{fbt}(P, C_{\mathcal{V},B} \cup \{X, \mathbf{ff}\}) \\
\mathsf{fbt}(X, C_{\mathcal{V},B} \cup \{X, B_X\}) & \Leftrightarrow & (B_X = \mathbf{tt})
\end{array}$$

avec $C_{\mathcal{V},B} = \{(X, B) | X \in \mathcal{V} \text{ et } B \in \{\mathbf{tt}, \mathbf{ff}\}\}$ *et* $C_{\mathcal{V},\mathbf{tt}} = \{(X, B) \in C_{\mathcal{V},B} | B = \mathbf{tt}\}$

Un processus P sera fortement bien temporisé si et seulement si $\mathsf{fbt}(P, \emptyset) = \mathbf{tt}$. Par exemple, le processus $P = recX.[0, 2]aX$ n'est pas fortement bien temporisé ($\mathsf{fbt}(P, \emptyset) = \mathbf{ff}$). En effet, ce processus peut effectuer une infinité de fois l'action a dans une durée finie. Inversement, le processus $Q = recX.[1, 2]aX$ est fortement bien temporisé car il doit laisser passer au moins une unité de temps entre deux actions a consécutives.

L'ensemble des processus réguliers et fortement bien temporisés de $\mathcal{P}$ définit l'ensemble des termes de TPAPa notés $\mathcal{P}^{TPAPa}$.

2.2 Sémantique

Les éléments de $\mathcal{T}$ seront notés t. Le système de transitions $(\mathcal{P}^{TPAPa}, \to)$ définit la sémantique opérationnelle de TPAPa où $\to \subseteq (P^{TPAPa} \times \mathcal{A} \cup \mathcal{T} \cup \{\varepsilon\} \times \mathcal{P}^{TPAPa})$. Pour spécifier que le processus P produit l'événement a pour devenir P', nous écrirons $P \xrightarrow{a} P'$. De la même manière, $P \xrightarrow{t} P'$ signifiera que le processus P laissera passer t unités de temps avant de se comporter comme P'.

La relation $\to$ n'est pas suffisante pour spécifier l'attente d'un événement par le processus $\overline{a}P$. En effet, un tel processus ne peut devenir P que s'il reçoit l'événement a émis par un co-processus via la composition parallèle. Tant qu'un tel événement n'est pas reçu, le processus $\overline{a}P$ doit uniquement laisser passer le temps. Afin de pouvoir modéliser ce comportement, une relation auxiliaire $\rightsquigarrow \subseteq (\mathcal{P}^{TPAPa} \times \mathcal{A} \times \mathcal{P}^{TPAPa})$ a été définie. Écrire $P \overset{a}{\rightsquigarrow} P'$ signifiera que le processus P attend l'événement a et deviendra P' lorsque cet événement sera reçu.

Les règles définissant les deux relations précédentes, la relation principale $\rightarrow$ et sa relation auxiliaire $\rightsquigarrow$, sont données dans le tableau 1.
Quelques règles sont décrites ici :

- La règle 8, $\overline{a}P \overset{a}{\rightsquigarrow} P$, définit la sémantique des actions de réception : "le processus attend l'événement a et lorsqu'il le reçoit devient le processus P."
- La synchronisation de deux processus (règles 6c et 6d) n'est possible que lorsque l'un des deux transmet un événement (relation $\overset{a}{\rightarrow}$) et l'autre est en attente de ce même événement (relation $\overset{a}{\rightsquigarrow}$).
- La règle 12c montre que la communication est réalisée par diffusion : une transmission peut être reçue par plusieurs processus simultanément.
- Comme expliqué précédemment, le choix peut être réalisé aussi par le temps. A l'aide des règles 19b et 19c, si P laisse uniquement passer le temps pendant t unités de temps et devient P et que le processus Q en est incapable (car $t > \mathbf{Sup}(Q)$ alors $P + Q$ laissera passer t unités de temps et deviendra P'.
- L'action invisible ε permet de "casser" l'opérateur intervalle. Le processus $\mathcal{I}P$ peut effectuer ε à n'importe quel instant (ε est une action persistante) et devenir P si et seulement si la borne inférieure de $\mathcal{I}$ est égale à 0 (règle 2).
- La règle 9a définit la sémantique de la préemption sans reprise. Ainsi $(P \Downarrow \overline{a})Q$ peut recevoir a pour devenir Q. Lorsque la préemption n'a pas lieu, le processus $(P \Downarrow \overline{a})Q$ exécute les actions de P (règle 3), laisse passer le temps si P laisse passer du temps et peut aussi se synchroniser sur un événement b si P se synchronise sur cet événement (règle 9b). Ce comportement est similaire à celui de l'instruction "abort p when a do q" du langage réactif ESTEREL [Ber93].

L'algèbre TPAPa possède une congruence. L'axiomatique, complète et adéquate, est donnée dans [EB02].

2.3 Propriétés

Cette section discute les propriétés de l'algèbre TPAPa par rapport aux propriétés génériques des systèmes temps-réel [NS91].

Déterminisme temporel. La propriété de déterminisme temporel est vérifiée si et seulement si le processus résultant du passage du temps est déterminé.

Proposition 1 *TPAPa est temporellement déterministe, formellement*

$$\forall P, P', P'', t : (P \overset{t}{\rightarrow} P' \wedge P \overset{t}{\rightarrow} P'') \Rightarrow P' = P''$$

Additivité temporelle. La propriété d'additivité temporelle est vérifiée si et seulement si un processus P peut laisser passer $t + t'$ unités de temps et devenir P' et si et seulement si P laisse passer t unités de temps et devenir le processus Q qui peut laisser passer t' unités de temps et devenir P'.

Proposition 2 *TPAPa satisfait la propriété d'additivité temporelle, formellement*

$$\forall P, P', t, t' : (\exists P'' : P \overset{t}{\rightarrow} P'' \wedge P'' \overset{t'}{\rightarrow} P') \Leftrightarrow P \overset{t+t'}{\rightarrow} P'$$

Deadlock-freeness. Un processus est deadlock-freeness si et seulement il peut toujours effectuer une action ou laisser passer du temps.

Proposition 3 *TPAPa satisfait la propriété de deadlock-freeness, formellement*

$$\forall P \ \exists l \in \mathcal{A} \cup \{\varepsilon\} \cup \mathcal{T} \ \exists P' : P \overset{l}{\rightarrow} P'$$

Variabilité finie. La propriété de variabilité finie est satisfaite si et seulement si pour tout processus seulement un nombre fini d'actions peuvent s'exécuter dans un intervalle de temps fini.

$$\text{(1)}\quad \frac{}{aP \xrightarrow{a} P}\ a \in \mathcal{A} \qquad\qquad \text{(2)}\quad \frac{}{[0,p]P \xrightarrow{\varepsilon} P}$$

$$\text{(3)}\quad \frac{P \xrightarrow{b} P' \qquad b \neq a}{(P \Downarrow \overline{a})Q \xrightarrow{b} (P' \Downarrow \overline{a})Q}\ b \in \mathcal{A} \qquad\qquad \text{(4)}\quad \frac{P[recX.P/X] \xrightarrow{a} P'}{recX.P \xrightarrow{a} P'}\ a \in \mathcal{A} \cup \{\varepsilon\}$$

$$\text{(5a)}\quad \frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'}\ a \in \mathcal{A} \cup \{\varepsilon\} \qquad\qquad \text{(5b)}\quad \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}\ a \in \mathcal{A} \cup \{\varepsilon\}$$

$$\text{(6a)}\quad \frac{P \xrightarrow{a} P' \qquad Q \stackrel{a}{\not\rightsquigarrow}}{P\|Q \xrightarrow{a} P'\|Q}\ a \in \mathcal{A} \cup \{\varepsilon\} \qquad\qquad \text{(6b)}\quad \frac{P \stackrel{a}{\not\rightsquigarrow} \qquad Q \xrightarrow{a} Q'}{P\|Q \xrightarrow{a} P\|Q'}\ a \in \mathcal{A} \cup \{\varepsilon\}$$

$$\text{(6c)}\quad \frac{P \xrightarrow{a} P' \qquad Q \stackrel{a}{\rightsquigarrow} Q'}{P\|Q \xrightarrow{a} P'\|Q'}\ a \in \mathcal{A} \qquad\qquad \text{(6d)}\quad \frac{P \stackrel{a}{\rightsquigarrow} P' \qquad Q \xrightarrow{a} Q'}{P\|Q \xrightarrow{a} P'\|Q'}\ a \in \mathcal{A}$$

$$\text{(7)}\quad \frac{}{\overline{a}P \stackrel{a}{\rightsquigarrow} P}\ \overline{a} \in \overline{\mathcal{A}}$$

$$\text{(8a)}\quad \frac{}{(P \Downarrow \overline{a})Q \stackrel{a}{\rightsquigarrow} Q} \qquad\qquad \text{(8b)}\quad \frac{P \stackrel{b}{\rightsquigarrow} P' \qquad b \neq a}{(P \Downarrow \overline{a})Q \stackrel{b}{\rightsquigarrow} (P' \Downarrow \overline{a})Q}$$

$$\text{(9a)}\quad \frac{P \stackrel{a}{\rightsquigarrow} P'}{P + Q \stackrel{a}{\rightsquigarrow} P'} \qquad\qquad \text{(9b)}\quad \frac{Q \stackrel{a}{\rightsquigarrow} Q'}{P + Q \stackrel{a}{\rightsquigarrow} Q'}$$

$$\text{(10a)}\quad \frac{P \stackrel{a}{\rightsquigarrow} P' \qquad Q \stackrel{a}{\not\rightsquigarrow} \qquad Q \stackrel{a}{\not\rightarrow}}{P\|Q \stackrel{a}{\rightsquigarrow} P'\|Q} \qquad\qquad \text{(10b)}\quad \frac{P \stackrel{a}{\not\rightsquigarrow} \qquad P \stackrel{a}{\not\rightarrow} \qquad Q \stackrel{a}{\rightsquigarrow} Q'}{P\|Q \stackrel{a}{\rightsquigarrow} P'\|Q}$$

$$\text{(10c)}\quad \frac{P \stackrel{a}{\rightsquigarrow} P' \qquad Q \stackrel{a}{\rightsquigarrow} Q'}{P\|Q \stackrel{a}{\rightsquigarrow} P'\|Q'} \qquad\qquad \text{(11)}\quad \frac{P[recX.P/X] \stackrel{a}{\rightsquigarrow} P'}{recX.P \stackrel{a}{\rightsquigarrow} P'}$$

$$\text{(12)}\quad \frac{}{\delta \xrightarrow{t} \delta}$$

$$\text{(13)}\quad \frac{}{\overline{a}P \xrightarrow{t} \overline{a}P} \qquad\qquad \text{(14)}\quad \frac{}{[m,p]P \xrightarrow{t} [\max(0, m-t), p-t]P}\ t \leq p$$

$$\text{(15)}\quad \frac{P \xrightarrow{t} P'}{(P \Downarrow \overline{a})Q \xrightarrow{t} (P \Downarrow \overline{a})Q} \qquad\qquad \text{(16)}\quad \frac{P[recX.P/X] \xrightarrow{t} P'}{recX.P \xrightarrow{t} P'}$$

$$\text{(17a)}\quad \frac{P \xrightarrow{t} P' \qquad Q \xrightarrow{t} Q'}{P + Q \xrightarrow{t} P' + Q'} \qquad\qquad \text{(18)}\quad \frac{P \xrightarrow{t} P' \qquad Q \xrightarrow{t} Q'}{P\|Q \xrightarrow{t} P'\|Q'}$$

$$\text{(17b)}\quad \frac{P \xrightarrow{t} P'}{P + Q \xrightarrow{t} P'}\ t > \mathbf{Sup}(Q) \qquad\qquad \text{(17c)}\quad \frac{Q \xrightarrow{t} Q'}{P + Q \xrightarrow{t} Q'}\ t > \mathbf{Sup}(P)$$

TAB. 1 – Règles de sémantique de TPAPa

Proposition 4 *TPAPa satisfait la propriété de variabilité finie.*

En effet, par définition les processus de TPAPa étant fortement bien temporisés, tout processus doit être capable d'exécuter un délai minimum non nul. Il est donc impossible d'exécuter un nombre infini d'actions dans un temps fini.

Persistance des actions. La propriété de persistance des actions est satisfaite si et seulement si la progression du temps n'empêche pas la possibilité d'effectuer une action, formellement

$$\forall P, Q, P', t, a : P \xrightarrow{a} P' \wedge P \xrightarrow{t} Q \Rightarrow \exists P'' : Q \xrightarrow{a} P''$$

TPAPa ne satisfait pas la propriété de persistance des actions. Par exemple, le processus $aP + [2,3]Q$ peut exécuter a immédiatement. Mais il peut aussi laisser passer 2 unités de temps et devenir $[0,1]Q$. Il n'est plus possible d'effectuer l'action a.

Cependant, l'action invisible ε est utilisée pour "casser" un intervalle de temps. Un processus $[0,p]P$ peut effectuer une action ε ou laisser passer t unités de temps (si $t \leq p$) et devenir $[0, p-t]P$ qui à son tour peut effectuer l'action ε. Nous parlons alors ε-persistency, formellement

$$\forall P, Q, P' : P \xrightarrow{\varepsilon} P' \wedge P \xrightarrow{t} Q \Rightarrow \exists P'' : Q \xrightarrow{\varepsilon} P''$$

2.4 Traduction en automates temporisés

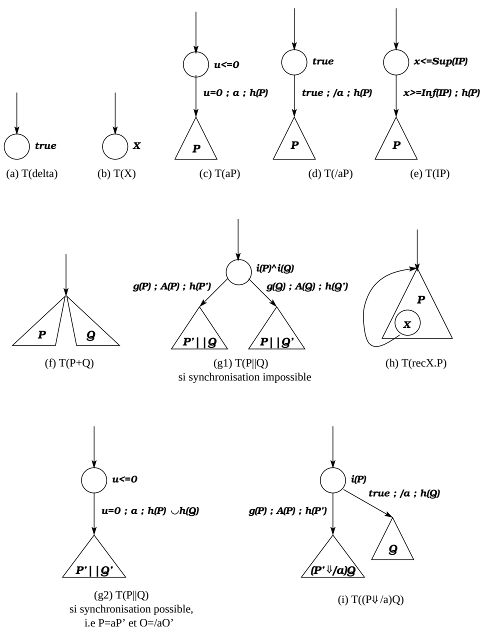

FIG. 1 – Traduction des différents opérateurs de l'algèbre en automates

Afin de pouvoir effectuer la vérification à l'aide d'UPPAAL, les règles de traduction en automates temporisés [AD94] ont été définies.

Définition 5 *(Automates temporisés)*
Un automate temporisé est un 5-uplet $\mathcal{TA} = <\mathcal{N}, \mathcal{V}, n_0, \mathcal{E}, \iota>$ *où :*
– $\mathcal{N}$ *représente un ensemble fini d'états,* n_0 *est l'état initial de l'automate.*
– $\mathcal{V}$ *est un ensemble fini de variables.*

- $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{B}(\mathcal{V}) \times A \times 2^{\mathcal{V}} \times \mathcal{N}$ *est un ensemble fini de transitions.* $e = < n, g, \alpha, r, n_e > \in \mathcal{E}$ *est une transition ayant pour source l'état* n *et destination l'état* n_e*,* α *est l'action associée à la transition,* g *est la garde de la transition, i.e. la condition sur les variables* $\mathcal{V}$ *nécessaire pour franchir la transition, et* r *est l'ensemble des variables à réinitialiser (remettre à zéro) pour cette transition.*
- ι *est une fonction* $\mathcal{N} \to \mathcal{B}(\mathcal{V})$ *associant un invariant à chaque état.*

Les règles de traduction en automates temporisés sont données graphiquement par la figure 1. Différents opérateurs ont été définis :
- L'opérateur $h(P)$ retourne l'ensemble des horloges à réinitialiser. Cet opérateur est défini inductivement :

$$\begin{array}{ll} h(aP) = \{u\} & h(\overline{a}P) = \emptyset \\ h(\mathcal{I}P) = \{x\} & h(\delta) = \emptyset \\ h(P+Q) = h(P) \cup h(Q) & h(P \| Q) = h(P) \cup h(Q) \\ h(X) = \emptyset & h(recX.P) = h(P) \\ h((P \Downarrow \overline{a})Q) = h(P) & \end{array}$$

  Remarque : La variable u sera toujours utilisée pour les actions, elle permet de modéliser l'urgence.
- L'opérateur $g(P)$ retourne la condition nécessaire pour franchir une transition (garde) pour un processus P donné.

$$\begin{array}{ll} g(aP) = (u = 0) & h(\overline{a}P) = \mathbf{tt} \\ g(\mathcal{I}P) = (x >= \mathbf{Inf}(\mathcal{I}P)) & g(\delta) = \mathbf{tt} \\ g(P+Q) = g(P) \vee g(Q) & h(P \| Q) = g(P) \vee g(Q) \\ g(X) = \mathbf{tt} & g(recX.P) = g(P) \\ g((P \Downarrow \overline{a})Q) = g(P) & \end{array}$$

- L'opérateur $A(P)$ retourne l'action effectuée lors du franchissement d'une transition. Ainsi $A(aP) = a$ signifie que l'action exécutée par aP est a. Dans le cas du processus $P+Q$, la valeur de $A(P+Q)$ est indéterminée. Elle correspond à $A(P)$ si $P+Q$ devient P ou $A(Q)$ si $P+Q$ devient Q.
- L'opérateur $i(P)$ retourne l'invariant associé à un processus donné :

$$\begin{array}{ll} i(aP) = (u <= 0) & i(\overline{a}P) = \mathbf{tt} \\ i(\mathcal{I}P) = (x <= \mathbf{Sup}(\mathcal{I}P)) & i(\delta) = \mathbf{tt} \\ i(P+Q) = i(P) \wedge i(Q) & i(P \| Q) = i(P) \vee i(Q) \\ i(X) = \mathbf{tt} & i(recX.P) = i(P) \\ i((P \Downarrow \overline{a})Q) = i(P) & \end{array}$$

De manière plus formelle, la méthodologie de traduction mise en œuvre dans [Nic94] sera utilisée. Tout d'abord les opérateurs $\mathbf{Succ} : \overline{\mathcal{A}}\mathbf{x}\mathbf{P} \to \mathbf{P}$ et $\mathbf{att} : \mathbf{P} \to \mathcal{H}$ avec $\mathcal{H} \in \mathcal{A}$ sont définis par induction sur l'ensemble des opérateurs de TPAPa. Le premier permet de connaître le successeur d'un processus donné lors d'une synchronisation. Le deuxième retourne l'ensemble des événements attendus par un processus particulier.

$$\begin{array}{lcl} \mathbf{Succ}(\overline{a}, \delta) & = & \emptyset \\ \mathbf{Succ}(\overline{a}, aP) & = & \emptyset \\ \mathbf{Succ}(\overline{a}, \overline{a}P) & = & \{P\} \\ \mathbf{Succ}(\overline{a}, IP) & = & \emptyset \\ \mathbf{Succ}(\overline{a}, (P \Downarrow \overline{a})Q) & = & \{Q\} \\ \mathbf{Succ}(\overline{a}, P+Q) & = & \mathbf{Succ}(\overline{a}, P) \cup \mathbf{Succ}(\overline{a}, Q) \\ \mathbf{Succ}(\overline{a}, P \| Q) & = & \{P' \| Q | P' \in \mathbf{Succ}(\overline{a}, P) \wedge \overline{a} \notin \mathbf{att}(Q)\} \cup \\ & & \{P \| Q' | Q' \in \mathbf{Succ}(\overline{a}, Q) \wedge \overline{a} \notin \mathbf{att}(P)\} \cup \\ & & \{P' \| Q' | P' \in \mathbf{Succ}(\overline{a}, P) \wedge Q' \in \mathbf{Succ}(\overline{a}, Q)\} \end{array}$$

$aP \xrightarrow{a,u=0,h(P)} P$	$\mathcal{I}P \xrightarrow{\varepsilon,x\leq \mathbf{Inf}(\mathcal{I}P),h(P)} P$
$\dfrac{P \xrightarrow{a,c,h} P'}{P + Q \xrightarrow{a,c,h} P'}$	$\dfrac{Q \xrightarrow{a,c,h} Q'}{P + Q \xrightarrow{a,c,h} Q'}$
$\dfrac{P \xrightarrow{a,c,h} P' \qquad a \notin \mathbf{att}(Q)}{P\|Q \xrightarrow{a,c,h} P'\|Q}$	$\dfrac{Q \xrightarrow{a,c,h} Q' \qquad a \notin \mathbf{att}(P)}{P\|Q \xrightarrow{a,c,h} P\|Q'}$
$\dfrac{P \xrightarrow{a,c,h(P')} P' \qquad Q' \in \mathbf{Succ}(\overline{a}, Q)}{P\|Q \xrightarrow{a,c,h(P')\cup h(Q')} P'\|Q'}$	$\dfrac{Q \xrightarrow{a,c,h(Q')} Q' \qquad P' \in \mathbf{Succ}(\overline{a}, P)}{P\|Q \xrightarrow{a,c,h(P')\cup h(Q')} P'\|Q'}$
$\dfrac{P \xrightarrow{a,c,h} P'}{(P \Downarrow \overline{b})Q \xrightarrow{a,c,h} (P' \Downarrow \overline{b})Q}$	$\dfrac{P[recX.P/X] \xrightarrow{a,c,h} P'}{recX.P \xrightarrow{a,c,h} P'}$

TAB. 2 – Règles permettant d'obtenir des arcs

$$
\begin{array}{lcl}
\mathbf{att}(\delta) & = & \emptyset \\
\mathbf{att}(aP) & = & \emptyset \\
\mathbf{att}(\overline{a}P) & = & \{a\} \\
\mathbf{att}(\mathcal{I}P) & = & \emptyset \\
\mathbf{att}((P \Downarrow \overline{a})Q) & = & \{a\} \\
\mathbf{att}(P + Q) & = & \mathbf{att}(P) \cup \mathbf{att}(Q) \\
\mathbf{att}(P\|Q) & = & \mathbf{att}(P) \cup \mathbf{att}(Q)
\end{array}
$$

Les invariants des nœuds des automates sont définis par l'opérateur i. Les règles du tableau 2 permettent de définir inductivement les arcs de l'automate.

Propriétés des automates obtenus :

L'algèbre définie possède deux propriétés essentielles : deadlock-freeness et variation finie. Ces deux propriétés garantissent que les processus de TPAPa ne sont jamais bloqués : une action est toujours possible ou le temps progressera, et le temps diverge (les processus sont non-zénon).

La traduction en automate préservant la sémantique de TPAPa, les automates temporisés obtenus posséderont alors les deux propriétés essentielles de TPAPa : ils sont deadlock-freeness et non-zénon.

2.5 Etude de cas : un système de commande d'une gouverne

L'objectif de l'algèbre TPAPa définie dans la section précédente est de permettre la modélisation pour la vérification de systèmes embarqués possédant des ressources

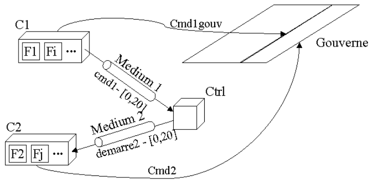

FIG. 2 – Système de commande d'une gouverne d'avion

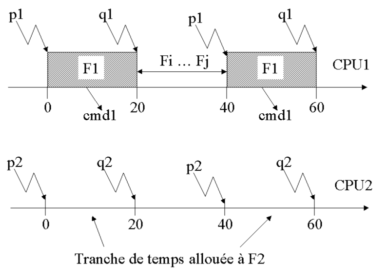

FIG. 3 – Comportement temporel de CPU1 et de CPU2 en fonctionnement normal

partagées, des séquenceurs (schedulers), des canaux de communication... Cette section propose de décrire le comportement d'un système de commande d'une gouverne à l'aide de TPAPa puis de vérifier deux propriétés en utilisant le model checker UPPAAL.

Ce système est composé de :
- deux calculateurs partagés C1 et C2, exécutant toutes les fonctions à bord de l'avion (F1, Fi,. . .F2, Fj,. . .)
- deux fonctions F1 et F2 (exécutées respectivement par C1 et C2) contrôlant la gouverne. F2 est appelée fonction de secours de F1 car elle remplace F1 lorsque celle-ci devient défaillante.
- un contrôleur Ctrl, permettant de connaître l'état de F1 : savoir si elle est défaillante ou non.
- deux canaux de communication, Medium1 et Medium2, transmettant le signal cmd1 de C1 vers Ctrl et demarre2 de Ctrl vers C2. La transmission de ces deux signaux dure entre 0 et 40 unités de temps à travers chaque canal.

En fonctionnement nominal, F1 est active et F2 n'est pas exécutée. F1 est activée périodiquement (toutes les 40ms) et émet à chaque activation un signal "cmd1gouv" vers la gouverne et un signal "cmd1" vers Ctrl via Medium1. Lorsque F1 devient défaillante, le signal "cmd1" n'est plus émis, et donc n'est plus reçu par Ctrl. Celui-ci attend 90 unités de temps avant de prendre en compte cette panne et émet "demarre2" vers C2 via Medium2. Pour simplifier le système, nous considérons que seul C1 et donc F1 peuvent être défaillants.

Les exécutions des fonctions de chaque calculateur sont contrôlées par des schedulers. Chacun de ces schedulers est chargé d'émettre périodiquement (toutes les 20 unités de temps) des signaux au CPU du calculateur dont il a la charge (signaux p1 et q1 pour CPU1) indiquant quelle fonction doit être exécutée. Concrètement, CPU1 (calculateur C1) et CPU2 (calculateur C2) sont structurés en deux tranches de temps. F1 est exécutée dans la première tranche de temps de CPU1. De même la première tranche de temps de CPU2 est réservée pour l'exécution de F2 en cas de panne de F1. En mode nominal cette tranche de temps est donc vide. Les autres fonctions (Fi ...Fj ...), n'agissant pas sur la gouverne et donc non considérées dans la modélisation, seront exécutées dans les deuxièmes tranches de temps de CPU1 et de CPU2. Le comportement de CPU1 et de CPU2 en fonctionnement nominal est montré sur la figure 3.

2.5.1 Modélisation à l'aide de TPAPa

La modélisation du système considéré précédemment peut être réalisée par la composition de sept processus définis formellement en TPAPa.

F1 et F2 : Chaque fois qu'elle est activée, F1 effectue des actions internes de calcul, abstraites ici par une durée variant entre 5 et 10 unités de temps, puis émet les événements "cmd1" et "cmd1gouv". De même, à chaque activation, F2 effectue des actions internes de calcul, abstraites par la même durée, puis émet l'événement "cmd2". Ces comportements sont donnés par les deux expressions suivantes :

$$\begin{aligned} \mathsf{F1} &= [5,10]\mathsf{cmd1}\ \mathsf{cmd1gouv}\ \delta \\ \mathsf{F2} &= [5,10]\mathsf{cmd2}\ \delta \end{aligned}$$

Mediums : Pour modéliser Medium1 et Medium2, nous utiliserons la composition parallèle de plusieurs canaux élémentaires comme montré par Milner ([Mil89]). Lorsque un canal élémentaire reçoit un signal, il le retransmet après un délai non-déterministe $t \in [0,40]$. Puisque F1 est activée toutes les 40 unités de temps, seulement deux canaux élémentaires sont nécessaires pour chaque médium. Nous formaliserons ces deux médiums par :

$$\begin{aligned} &\mathsf{M1} = \mathsf{recX}.\overline{\mathsf{cmd1}}\ [0,40]\ \mathsf{cmd1ctrl}\ \mathsf{X} \\ &\mathsf{M2} = \mathsf{recX}.\overline{\mathsf{demarre2ctrl}}\ [0,40]\ \mathsf{demarre2}\ \mathsf{X} \\ &\mathsf{Medium1} = \mathsf{M1}||\mathsf{M1} \\ &\mathsf{Medium2} = \mathsf{M2}||\mathsf{M2} \end{aligned}$$

Notons que nous différencions les signaux entrant et sortant d'un médium. Par exemple, "cmd1" est le signal émis sur Medium1 et "cmd1ctrl" est le nom de ce même signal sortant du médium après un certain délai.

Ctrl : Ctrl attend le signal "cmd1ctrl" de Medium1 provenant de l'émission de "cmd1" par F1. S'il n'a pas reçu d'occurrence du signal "cmd1ctrl" pendant 90 unités de temps, il émet le signal "demarre2ctrl" :

$$\begin{aligned} \mathsf{Ctrl} = \mathsf{recX}.(\overline{\mathsf{cmd1ctrl}}\mathsf{X}\quad &+\overline{\mathsf{timeout}}\ \mathsf{demarre2ctrl}\ \delta)|| \\ &\mathsf{recX}.(([90,90]\mathsf{timeout}\ \delta \Downarrow \overline{\mathsf{cmd1ctrl}})\mathsf{X}) \end{aligned}$$

Schedulers : Avant de définir le comportement des deux CPUs, intéressons nous à ceux des deux schedulers. Chaque scheduler émet une interruption vers son CPU toutes les 20 unités de temps : "qi"(respectivement "pi") interrompant Fi (respectivement les autres fonctions) sur Ci :

$$\begin{aligned} \mathsf{Sch1} &= \mathsf{recX}.\mathsf{p1}[20,20]\mathsf{q1}[20,20]\mathsf{X} \\ \mathsf{Sch2} &= \mathsf{recX}.\mathsf{p2}[20,20]\mathsf{q2}[20,20]\mathsf{X} \end{aligned}$$

CPU1 : La séquence T1 représente l'alternance de F1 et des autres fonctions lors des réceptions de "p1" et de "q1" sur CPU1 pendant le fonctionnement normal. Comme le comportement des autres fonctions ne nous intéresse pas, nous les modéliserons par le processus δ. Lorsque CPU1 est défaillant, il n'exécute plus d'action et laisse seulement passer le temps, ce qui est représenté par le processus δ. Pour simplifier le système, la mise en défaillance de CPU1 s'effectuera à l'aide du signal "defail" émis à n'importe quel instant par un processus externe Defaillance :

$$\begin{aligned}
&Defaillance = [0, \infty)defail\ \delta \\
&T1 = \overline{p1}\ recX.(F1 \Downarrow \overline{q1})(\delta \Downarrow \overline{p1})X \\
&CPU1 = (T1 \Downarrow \overline{defail})\delta
\end{aligned}$$

CPU2 : En fonctionnement nominal, CPU2 exécute la séquence T2 modélisant l'alternance des fonctions n'intervenant pas dans le système de contrôle de la gouverne lors de la réception des signaux "p2" et "q2". Ces fonctions sont abstraites par le processus δ. Lors de la panne de CPU1, c'est-à-dire dès réception du signal "demarre2", CPU2 exécute une nouvelle séquence T2' permettant l'alternance de F2 et des autres fonctions (elles aussi modélisées par δ) :

$$\begin{aligned}
&T2 = recX.(\delta \Downarrow \overline{q2})(\delta \Downarrow \overline{p2})X \\
&T2' = recX.(F2 \Downarrow \overline{q2})(\delta \Downarrow \overline{p2})X \\
&CPU2 = (T2 \Downarrow \overline{demarre2})T2'
\end{aligned}$$

Pour finir, le système global peut être représenté par la composition des processus précédents.

$$System = (CPU1 \| CPU2 \| Sch1 \| Sch2 \| Ctrl \| Medium1 \| Medium2)$$

Le problème est maintenant de vérifier deux propriétés pour ce système en considérant que seule F1 peut être défaillante. L'expression algébrique considérée sera alors

$$System' = (System \| Defaillance)$$

2.5.2 Vérification avec UPPAAL

Afin de pouvoir vérifier notre système a l'aide d'UPPAAL, les processus précédents ont été traduits en automates temporisés utilisés par UPPAAL en suivant les règles de la section 2.3.

Le système de contrôle d'une gouverne doit satisfaire deux propriétés de sûreté :
- la gouverne ne doit pas être orpheline plus de 190 unités de temps ;
- F2 doit être active seulement si F1 ne l'est plus.

Nous utiliserons la logique temporisé TCTL [ACD90] pour exprimer ces propriétés :

$$\begin{aligned}
\varphi_1 &= \forall\Box(\forall\Diamond_{\leq 190}(cmd1 \vee cmd2)) \\
\varphi_2 &= \forall\Box(cmd2 \rightarrow \forall\Box\neg cmd1)
\end{aligned}$$

Afin de vérifier ces deux propriétés, φ_1 et φ_2 seront traduites en automates temporisés comme montré dans la figure 4. Le verification consiste à prouver que les états "unhappy" (voir figure 4) sont inaccessibles. Cette vérification, réalisée à l'aide d'UPPAAL v3.2.1 sur une Sun Ultrasparc 10 possédant 256 Mo de mémoire, nécessite moins d'une seconde pour chaque propriété.

3 TPAPas : TPAPa étendue à l'aide de la suspension de processus avec reprise

Cette section présente une extension de l'algèbre TPAPa décrite à la section 2 avec un opérateur effectuant la suspension de l'exécution d'un processus au profit d'un autre et permettant la reprise du processus suspendu. L'algèbre obtenue est nommée TPAPas.

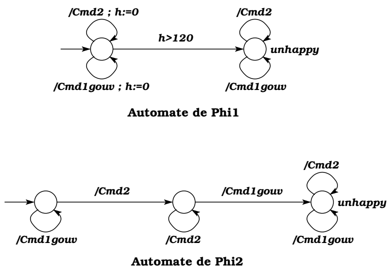

FIG. 4 – Automates représentant les propriétés à vérifier

3.1 Extension de la syntaxe et de la sémantique

La syntaxe de l'algèbre étendue est définie par la grammaire suivante :

$$\mathsf{P} ::= \delta \mid \mathsf{X} \mid a\mathsf{P} \mid \overline{a}\mathsf{P} \mid \mathcal{I}\mathsf{P} \mid \mathsf{P}+\mathsf{P} \mid \mathsf{P}\|\mathsf{P} \mid \mathsf{recX.P} \mid (\mathsf{P} \Downarrow \overline{a})\mathsf{P} \mid (\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{P}$$

Le nouvel opérateur $(\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q}$ réalise la suspension du processus P dès la réception de l'événement a et exécute le processus Q. Celui-ci peut à son tour être suspendu dès que l'événement b est reçu. Le processus P s'exécutera à nouveau à partir du point de suspension. Pour mettre en œuvre un tel comportement, la sémantique de ce nouvel opérateur sera la suivante :

$$(1)\ \frac{\mathsf{P} \xrightarrow{c} \mathsf{P}' \qquad c \neq a \qquad c \neq b}{(\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q} \xrightarrow{c} (\mathsf{P}' \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q}}\ c \in \mathcal{A}$$

$$(2a)\ \frac{}{(\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q} \overset{a}{\rightsquigarrow} (\mathsf{Q} \downarrow \overline{b} \uparrow \overline{a})\mathsf{P}} \qquad (2b)\ \frac{\mathsf{P} \overset{c}{\rightsquigarrow} \mathsf{P}' \qquad c \neq a \qquad c \neq b}{(\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q} \overset{c}{\rightsquigarrow} (\mathsf{P}' \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q}}$$

$$(3)\ \frac{\mathsf{P} \xrightarrow{t} \mathsf{P}'}{(\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q} \xrightarrow{t} (\mathsf{P}' \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q}}$$

La règle 2a ci-dessus réalise la préemption du processus P quand l'événement a est reçu, le processus Q s'exécute. Ainsi $(\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q}$ peut recevoir a et se comporter comme $(\mathsf{Q} \downarrow \overline{b} \uparrow \overline{a})\mathsf{P}$. Ce nouveau processus pourra à son tour recevoir b et relancer le processus P. Ici aussi lorsque la préemption n'a pas lieu, le processus P peut exécuter des actions (règle 1), peut se synchroniser avec un autre processus (règle 2b) ou bien laisser passer du temps (règle 3).

Les fonctions free, Σ et fbt définies en section 2.1 sont étendues par :

$$\begin{array}{lcl} \mathsf{free}((\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q}) & = & \mathsf{free}(\mathsf{P}) \cup \mathsf{free}(\mathsf{Q}) \\ \Sigma((\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q}) & = & \Sigma(\mathsf{P}) \cup \Sigma(\mathsf{Q}) \\ \mathsf{fbt}((\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q}) & \Leftrightarrow & \mathsf{fbt}(\mathsf{P}) \vee \mathsf{fbt}(\mathsf{Q}) \end{array}$$

On dit maintenant qu'un terme P est régulier si et seulement si tous ses sous-termes parallèles et préemptés sont clos et si tous les sous-termes du type $(\mathsf{R} \Downarrow \overline{a})\mathsf{Q}$ et $(\mathsf{R}' \downarrow \overline{a'} \uparrow \overline{b'})\mathsf{Q}'$ sont tels que $a \notin \Sigma(\mathsf{R})$, $a' \notin \Sigma(\mathsf{R}')$ et $b' \notin \Sigma(\mathsf{Q}')$. Comme pour TPAPa, les processus de TPAPas sont les processus réguliers et fortement bien temporisés.

Les propriétés énoncées à la section 2.3 restent vraies pour TPAPas, en particulier les propriétés de variabilité finie et deadlock-freeness.

3.2 Traduction en automates hybrides

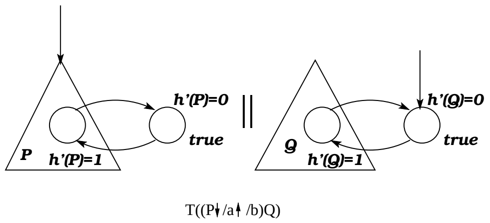

FIG. 5 – Traduction de l'opérateur de suspension en automate hybride linéaire

En observant la sémantique de TPAPas, la modélisation de l'opérateur de suspension $(\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q}$ doit proposer non seulement le blocage de toute action possible mais aussi de l'évolution du temps. La solution consiste à utiliser des chronomètres [Rus96] qui mesurent le temps lorsque le processus est actif et arrête la mesure lorsque le processus est passif. Par exemple, supposons que x est une variable d'horloge pour la traduction d'un processus P. La traduction de $(\mathsf{P} \downarrow \overline{a} \uparrow \overline{b})\mathsf{Q}$ doit permettre l'évolution de cette variable x. Lorsque l'événement a survient, P devient passif et l'évolution de x est arrêtée. Ceci est possible en utilisant les automates hybrides linéaires [ACH$^+$95].

Définition 6 *(Automate hybride linéaire)*
Un automate hybride linéaire est un 6-uplet $\mathcal{HA} =< \mathcal{N}, \mathcal{V}, n_0, \mathcal{E}, \iota, \mathcal{D} >$ où :
- *$\mathcal{N}$ représente un ensemble fini d'états, n_0 est l'état initial de l'automate.*
- *$\mathcal{V}$ est un ensemble fini de variables.*
- *$\mathcal{E} \subseteq \mathcal{N} \times \mathcal{B}(\mathcal{V}) \times \mathsf{A} \times 2^{\mathcal{V}} \times \mathcal{N}$ est un ensemble fini de transitions. $e =< n, g, \alpha, r, n_e > \in \mathcal{E}$ est une transition ayant pour source l'état n et destination l'état n_e, α est l'action associée à la transition, g est la garde de la transition, i.e. la condition sur les variables $\mathcal{V}$ nécessaire pour franchir la transition, et r est l'ensemble des variables à réinitialiser (remettre à zéro) pour cette transition.*
- *ι est une fonction $\mathcal{N} \rightarrow \mathcal{B}(\mathcal{V})$ associant un invariant à chaque état.*
- *$\mathcal{D}$ est une fonction $\mathcal{N} \rightarrow (\mathcal{V} \rightarrow \mathbb{R})$ associant à chaque état une fonction d'évolution pour une variable de $\mathcal{V}$. Cette évolution est souvent écrite sous une forme différentielle. Par exemple, $\mathcal{D}(n)(x) = k$ signifie que dans l'état n la variable x évolue suivant la différentielle $\frac{dx}{dt} = k$.*

Ainsi en associant la fonction d'évolution $\frac{dx}{dt} = 1$ à la variable x, x évoluera continûment avec le temps. Lorsque $\frac{dx}{dt} = 0$, la variable n'évoluera plus. Ce comportement s'apparente à celui d'un chronomètre pour x :
- déclenchement, le compteur évolue avec le temps, $\frac{dx}{dt} = 1$.
- arrêt, le compteur stoppe, $\frac{dx}{dt} = 0$.

C'est ce type de fonctionnement que nous mettons en œuvre pour la traduction de l'opérateur de suspension montrée figure 5.

Dans la figure, l'écriture $h'(\mathsf{P}) = 0$ signifie que pour toutes les horloges x de P l'évolution doit être arrêtée, $\frac{dx}{dt} = 0$. Inversement $h'(\mathsf{P}) = 1$, signifie que toutes les horloges x de P évoluent continûment avec le temps, $\frac{dx}{dt} = 1$.

Puisque les automates hybrides linéaires correspondent à une extension des automates temporisés à l'aide d'une fonction d'évolution, la traduction des opérateurs de TPAPas dans les automates hybrides linéaires est celle indiquée figure 1.

Propriétés des automates hybrides obtenus :

De la même manière que les automates temporisés de la section 2.4, par traduction des opérateurs de TPAPas, les automates hybrides obtenus à partir de TPAPas seront non-zénon :
- il sera toujours possible de franchir une transition effectuant une action ou laissant passer du temps ;
- il sera toujours possible d'effectuer une infinité d'actions dans un intervalle de temps fini ;
- une suite finie d'actions sera toujours suivie d'une ou plusieurs transitions modélisant un délai strictement non-nul.

Vérification de propriétés sur les automates hybrides obtenus :

Dans [Rus96], Vlad Rusu montre que les automates hybrides fortement non-zénon, i.e. dont le temps diverge et exécutant un nombre fini d'actions dans un intervalle de temps fini, rendent la vérification de propriétés décidable. Il montre ensuite les méthodes mises en œuvre pour vérifier des propriétés écrites en logique TCTL-bornée.

Forts de ces propriétés, un exemple simple, correspondant à celui de la section 2.5, a été testé en utilisant deux model-checkers différents : HyTech [HHWT97] et HCMC [LC00]. Malheureusement les résultats ne se sont pas montrés concluant.

4 Conclusion

Dans le but de pouvoir modéliser puis ensuite vérifier des systèmes temps réels embarqués composés d'un ensemble de processus communicants supportant des stratégies d'ordonnancement, qui seront définies à l'aide des opérateurs de préemption, sur des ressources partagées, nous avons décrit l'algèbre de processus temporisés appelé TPAPa permettant la diffusion et la préemption. Nous avons ensuite utilisé cette algèbre sur un cas d'étude avionique. L'expression algébrique obtenue fut ensuite traduite en automates temporisés et vérifiée à l'aide d'UPPAAL.

Nous avons ensuite étendu TPAPa par un opérateur de suspension avec reprise. Tandis que le mécanisme de préemption $(\mathsf{P} \Downarrow \overline{\mathsf{a}})\mathsf{Q}$ abandonne l'exécution du processus P dès réception de a, offrant ainsi une sémantique proche de la commande UNIX "Control C", l'opérateur de suspension $(\mathsf{P} \downarrow \overline{\mathsf{a}} \uparrow \overline{\mathsf{b}})\mathsf{Q}$ permet de suspendre l'activité du processus P dès réception de l'événement a sans perdre l'état dans lequel il était. L'exécution du P sera ensuite reprise dès réception de l'événement b. Un tel comportement correspond donc à celui des commandes UNIX "Control Z" puis "fg". L'abandon et la suspension sont les deux mécanismes de préemption fondamentaux nécessaires pour modéliser des systèmes temps-réel concrets.

En TPAPas, lorsqu'un processus est suspendu, son exécution s'arrête et les différentes horloges associées à ce processus ne doivent plus évoluer. Un tel comportement est similaire à celui des automates hybrides à chronomètres. Ainsi afin de vérifier certaines propriétés sur les modèles réalisés à l'aide de l'algèbre TPAPas, nous les avons traduits en automates hybrides à chronomètres. Bien qu'il ait été montré que la vérification sur cette classe d'automates hybrides est décidable, les résultats obtenus à l'aide des outils utilisés ne se sont pas montrés concluants. Aussi la perspective principale de ce travail est l'adaptation des techniques de model checking du monde des automates hybrides à l'algèbre TPAPas que nous avons définie.

Références

[ACD90] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for real-time systems. In *Proceedings of the 5th annual IEEE Symposium on Logic in Computer Science*, Philadelphia, Pensylvania, 1990. IEEE Computer Society Press.

[ACH$^+$95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1) :3–34, 1995.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[Ber93] Gerard Berry. Preemption in concurrent systems. In *Foundations of Software Technology and Theoretical Computer Science*, pages 72–93, 1993.

[BGL97] P. Bremond-Gregoire and I. Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science*, 189, 1997.

[D'A99] P. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* PhD thesis, Department of Computer Science, University of Twente, 1999.

[EB02] Jérôme Ermont and Frédéric Boniol. TPAP : an algebra of preemptive processes for verifying real-time systems with shared resources. In *Workshop on Theory and Practice of Timed Systems*, April 2002.

[ER86] J. P. Elloy and O. Roux. Electre : a language for control structuring in real time. *The Computer Journal*, 29(3) :229–234, 1986.

[Fec01] Harald Fecher. A real-time process algebra with open intervals and maximal progress. *Nordic Journal of Computing*, 8(3) :346–365, 2001.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH : A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2) :110–122, 1997.

[LC00] F. Laroussinie and F. Cassez. Model-checking for hybrid systems by quotienting and constraints solving. In E. Allen Emerson and A. Prasad Sistla, editors, *12th International Conference on Computer-Aided Verification.* LNCS 1855, Springer-Verlag, July 2000.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2) :134–152, 1997.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Nic94] Xavier Nicollin. *ATP : une algèbre pour la spécification et l'analyse des systèmes temps réel.* PhD thesis, Ecole Nationale Polytechnique de Grenoble, 1994.

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In K.G. Larsen and A. Skou, editors, *3rd. Computer-Aided Verification*, pages 376–398. LNCS 575, Springer-Verlag, July 1991.

[Rus96] Vlad Rusu. *Vérification temporelle de programmes ELECTRE.* PhD thesis, Ecole Centrale de Nantes, 1996.

[upp] http ://www.uppaal.com.