

LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

MODÉLISATION DE SYSTÈMES RÉACTIFS PAR UNE APPROCHE GRAPHIQUE SYNCHRONES : SYNCCHARTS

Charles André

Projet SPORTS

Rapport de recherche
ISRN I3S/RR-2003-25-FR

Octobre 2003

RÉSUMÉ :

Les SyncCharts sont un modèle graphique qui s'appuie sur les concepts d'états et transitions et qui adopte un point de vue synchrone. Les SyncCharts utilisent un nombre réduit de primitives graphiques. Cette apparente simplicité peut cacher des comportements complexes qui combinent évolutions parallèles, préemptions et réincarnations d'états. Cette présentation explique de telles réactions en termes de "microsteps". La sémantique du modèle est une sémantique constructive pleinement compatible avec celle du langage synchrone Estérel. Cette sémantique est présentée de façon semi formelle comme le résultat de la coopération de "reactive cells" évoluant concurremment.

Cours donné à l'école d'été "Temps réel", Toulouse, Septembre 2003

MOTS CLÉS :

systèmes réactifs, programmation synchrone, SyncCharts

ABSTRACT:

SyncCharts are a graphical model that relies on the concepts of states and transitions, and that adopts a synchronous point of view. SyncCharts use only a restricted set of graphical primitives. This syntactic simplicity may hide complex behaviours, with concurrent evolutions, pre-emptions and state re-incarnations. In this paper we explain such reactions in terms of "microsteps". The semantics of SyncCharts is a constructive semantics a la Esterel. This semantics is introduced in a semi-formal way and it reflects the cooperation of concurrent reactive cells.

Lecture given at the "Real Time Summer School", Toulouse, Sept 2003

KEY WORDS :

reactive systems, synchronous programming, SyncCharts

Modélisation de systèmes réactifs par une approche graphique synchrone : SyncCharts

Charles André⁽¹⁾

⁽¹⁾I3S, BP 121, 06903 Sophia Antipolis Cedex, andre@unice.fr

RÉSUMÉ. Les SyncCharts sont un modèle graphique qui s'appuie sur les concepts d'états et transitions et qui adopte un point de vue synchrone. Les SyncCharts utilisent un nombre réduit de primitives graphiques. Cette apparente simplicité peut cacher des comportements complexes qui combinent évolutions parallèles, préemptions et réincarnations d'états. Cette présentation explique de telles réactions en termes de « microsteps ». La sémantique du modèle est une sémantique constructive pleinement compatible avec celle du langage synchrone Estérel. Cette sémantique est présentée de façon semi formelle comme le résultat de la coopération de « reactive cells » évoluant concurremment.

MOTS-CLÉS : langages synchrones, sémantique constructive, systèmes réactifs.

1. Introduction

Les systèmes réactifs sont des systèmes à événements discrets. Leur comportement peut être représenté par une séquence de réactions à des stimuli. Classiquement, les évolutions séquentielles sont décrites en termes d'états et de transitions. Les modèles simples comme les machines séquentielles sont inadaptés lorsqu'il s'agit de représenter des applications modernes. Ces applications sont généralement complexes et conçues comme un ensemble de sous-systèmes communicants évoluant concurremment avec de nombreuses interactions (synchronisations, préemptions). Un sous-système peut lui-même être décomposé. En conséquence, les modèles pour systèmes réactifs devraient être hiérarchiques, supporter les évolutions parallèles, des formes de synchronisation et de communication diverses.

Les langages synchrones ont été introduits spécialement pour la programmation des systèmes réactifs. En Estérel [1], un langage synchrone impératif, le concept de *signal* apporte une représentation abstraite à la fois de la communication et de la synchronisation. Les réactions du système réactif, en réponse aux stimuli, sont caractérisées par des émissions et réceptions de signaux. Les signaux sont également les causes de *préemptions*. Les préemptions se manifestent sous deux formes principales : la suspension ou l'abandon de l'activité de sous-systèmes. Cette deuxième forme est dénommée *abortion* en anglais. Dans la suite nous utiliserons parfois ce terme anglais. Les modèles synchrones adoptent une hypothèse simplificatrice : le système n'évolue que par phases temporellement disjointes appelées *instants*, dont la durée est nulle. De plus, les signaux sont instantanément diffusés (instantaneous broadcast) à tout le programme. Ainsi, lors d'une réaction (instantanée), tous les signaux qui résultent d'interaction complexes entre sous-systèmes, sont *simultanés*. On dit aussi qu'ils sont *synchrones*. Ces fortes hypothèses permettent de garantir des comportements déterministes tout en ayant du parallélisme et des préemptions.

Le langage Estérel est un langage spécialisé pour la programmation de systèmes de contrôle. Toutefois, il ne supporte pas directement des spécifications en termes de machines à états hiérarchiques et communicantes. Les SyncCharts [2] qui ont été introduits comme une forme graphique du langage Estérel, comblent cette lacune en adoptant les états et transitions comme des primitives du modèle. En tant que descendants directs du langage Estérel, les SyncCharts s'appuient sur la même sémantique mathématique qu'Estérel. Ils ont donc les mêmes avantages et inconvénients qu'Estérel. La syntaxe des SyncCharts a été directement influencée par les Statecharts [3] de D. Harel. Ce double héritage Estérel/Statecharts peut être une source de confusion pour l'utilisateur débutant : un syncChart² ressemble à un statechart mais son comportement est différent.

¹ Présenté à l'École d'Été Temps Réel, 9-12 Septembre 2003, Toulouse (F)

² SyncCharts est le modèle. Un syncChart est une instance du modèle.

Les parties 2 et 3 de cet article ont pour objet de présenter la syntaxe et la sémantique des SyncCharts de façon informelle au travers d'exemples simples. Malgré leur apparente simplicité, les SyncCharts peuvent présenter des comportements complexes qui reflètent les hypothèses synchrones. La sémantique des SyncCharts est décrite en termes de micro-pas (*microsteps*) qui respectent la *causalité constructive* : à chaque instant, le statut de présence de tout signal doit être déterminé avant toute utilisation. Cette règle est l'essence même de la sémantique constructive [4] définie par G. Berry pour le langage Estérel. La partie 4 exprime cette sémantique sous une forme opérationnelle qui s'appuie sur la structure des SyncCharts. Cette structure est elle-même définie de façon précise par un modèle UML.

2. Aperçu sur les SyncCharts

2.1. Exemple illustratif

Le syncChart de la figure 1 est un exemple typique qui contient les principaux éléments graphiques des SyncCharts. Il s'agit d'un détecteur d'occurrences synchronisées. Le signal AB doit être émis chaque fois que A et B se sont réalisés, dans n'importe quel ordre et qu'entre ces occurrences il y a eu au plus deux occurrences de T. Le signal Reset réinitialise le système de façon prioritaire. Quant au signal Inhib il permet de « suspendre le temps » : en présence de Inhib, les occurrences de T sont ignorées.

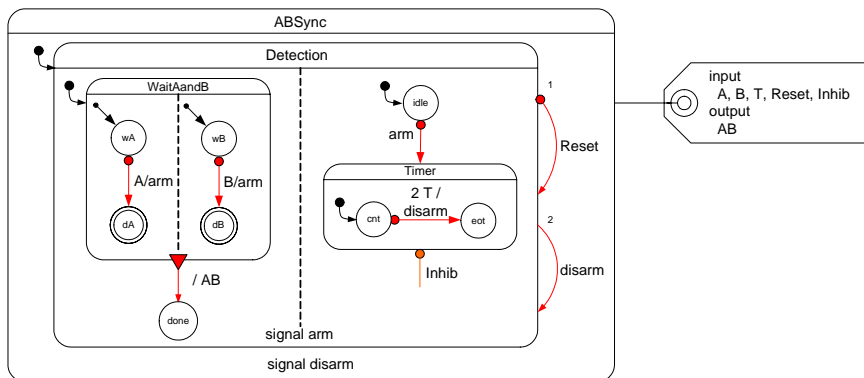


Figure 1. Exemple de syncChart

2.2. Syntaxe simplifiée des SyncCharts

2.2.1. Structure

Un état (*state*) peut être soit un état simple (*simple state*) représenté par une ellipse, soit un macro-état (*macrostate*) représenté par un rectangle à coins arrondis. Contrairement au état simple, un macro-état est raffiné : il contient d'autres états. Un simple état peut être nommé, son identificateur est inscrit dans l'état. Le nom facultatif d'un macro-état est écrit dans un cartouche situé dans la partie supérieure du macro-état.

Une *transition* est un lien orienté entre deux états, depuis un état *source* vers un état cible (*target*). Il existe trois types de transitions : transitions de préemption forte, de préemption faible et de terminaison normale (respectivement appelées *strong abortion*, *weak abortion* et *normal termination*). La transition de l'état wA à l'état dA est une transition de préemption forte. La boucle sur l'état Detection et étiquetée disarm est une transition de préemption faible. La transition de l'état WaitAandB vers l'état done est une transition de terminaison normale.

Un graphe d'état-transition (*State-Transition Graph*, ou STG) est un graphe connexe dont les nœuds sont des états et les arêtes des transitions. Un STG doit avoir un état initial, représenté graphiquement par un état sur lequel arrive une flèche. Detection, wA, wB sont les états initiaux de différents STG. Un STG peut également avoir des états finals, représentés par une bordure dédoublée. Un STG est nécessairement contenu dans un macro-état. Contrairement aux Statecharts, les SyncCharts n'acceptent pas des transitions coupant les limites de macro-états (*inter-level transition*). Le macro-état Detection contient deux STG

concurrents, de même pour le macro-état `WaitAandB`. En revanche le macro-état `ABSync` ne contient qu'un seul STG. Les STG contenus dans un même macro-état sont dits concurrents et ils sont séparés par un segment de droite en pointillés.

Lorsque plusieurs transitions ont pour source un même état, les transitions sont totalement ordonnées (priorité). La priorité est représentée par un entier situé près de l'origine de la transition. 1 est la priorité la plus élevée. Pour l'état `Detection`, la transition étiquetée `Reset` a priorité sur la transition étiquetée `disarm`.

L'arc spécial aboutissant sur la partie inférieure du macrostate `Timer` est un arc de *suspension*.

Une étiquette peut être associée à chaque transition, arc initial ou arc de suspension. Une étiquette fait référence à des signaux, ceux-ci sont présentés dans le paragraphe suivant.

2.2.2. Signaux

Le signal est l'abstraction permettant de d'exprimer les communications et les synchronisations. L'émission et la réception d'un signal correspondent à la notion classique d'événement. Un signal à un *statut de présence* qui peut être présent (+), absent (-) ou inconnu (\perp). Un signal peut porter une *valeur* d'un type donné, on parle alors de *signal valué*. Dans le cas contraire il s'agit d'un *signal pur*. Lorsqu'un signal valué peut avoir plusieurs émissions simultanées, une *fonction de combinaison* doit être associée au signal. Un signal a également une portée (*scope*). Il est soit *externe* soit *local* à un macro-état. Pour un signal externe on distingue signal d'entrée (*input*) imposé par l'environnement et signal de sortie (*output*) calculé par le syncChart. Les signaux locaux sont bi-directionnels et ils servent aux communications internes du modèle.

Les signaux locaux sont explicitement mentionnés dans le syncChart : `arm` est local au macro-état `Detection` ; `disarm` est local à `ABSync`. Les signaux d'entrée sont `A`, `B`, `T`, `Reset` et `Inhib` ; il n'y a qu'un signal de sortie `AB`.

2.2.3. Étiquette associée à une transition

La syntaxe de l'étiquette d'une transition est `trigger [guard] / effect`, les trois champs sont facultatifs. Le déclencheur (*trigger*) peut ne tester que la simple présence d'un signal (e.g. `A` dans le macro-état `WaitAandB`). Il peut être précédé d'un facteur à valeur entière (e.g. `2T` dans le macro-état `Timer`). Les triggers les plus complexes sont des expressions construites sur les noms de signaux et les opérateurs `and`, `or` et `not`. Une garde (*guard*) est un prédicat qui utilise les valeurs de signaux et des constantes. L'effet (*effect*) est un ensemble d'actions instantanées. L'émission d'un signal est une telle action.

2.2.4. Étiquette associée à un état

On peut associer un effet à un état. La syntaxe est `/ effect`.

2.3. Sémantique informelle

Jusqu'à présent nous n'avons considéré que les aspects syntaxiques du modèle. Les SyncCharts étant conçus pour représenter la dynamique des systèmes réactifs, il faut pouvoir « calculer » les réactions d'un syncChart. Ceci passe par la définition d'une sémantique pour le modèle. Dans ce paragraphe les explications resteront informelles. De plus, la présentation se limitera aux SyncCharts Purs (ne contenant que des signaux purs). Cette restriction est tout à fait pertinente car elle permet de se focaliser sur la présence/absence des signaux. L'évaluation des triggers, qui jouent un rôle essentiel dans la réactivité, ne dépend que de ces statuts de présence. Se limiter aux modèles purs est une approche qui à déjà été appliquée avec succès pour expliquer la sémantique du langage Estérel [4].

2.3.1. Réaction

Un syncChart, comme les autres modèles synchrones, est exécuté « cycliquement ». Un cycle d'évolution comprend trois phases :

1. *Lecture des entrées* : avec les SyncCharts purs ceci revient à déterminer le statut de présence de tous les signaux d'entrée. Cette valuation des entrées constitue l'image d'entrée.
2. *Calcul de la réaction* : en fonction de l'état du modèle (sa *configuration*) et de l'image d'entrée il faut calculer la nouvelle configuration et le statut de présence de chaque signal de sortie.
3. *Écriture des sorties* : le statut de présence de chaque signal de sortie (image de sortie) est appliqué à l'environnement.

Ce cycle est supposé instantané. La réaction est déterministe. Les réactions successives, qui correspondent aux instants successifs, ne se chevauchent pas.

2.3.2. *Machines communicantes*

Une première approche de la sémantique des SyncCharts est de considérer un syncChart comme un ensemble de machines communicantes. (une machine par chaque STG contenu dans le syncChart). « A finite state machine is a machine specified by a finite set of conditions of existence (called states) and a likewise finite set of transitions among states triggered by events ». Cette définition donnée par B. P. Douglass [5] s'applique aux SyncCharts en remplaçant événements par signaux. Un état caractérise une condition qui peut persister un certain temps. Quand le système est dans un état il est *réceptif* à un ensemble de signaux et peut atteindre d'autres états (franchissement de transition). L'état effectivement atteint dépend des signaux effectivement reçus. Considérons le cas où les états *wA*, *wB* et *idle* sont actifs dans le syncChart *ABSync*. Si *A* est présent alors on franchit la transition de l'état *wA* vers l'état *dA*. L'effet associé (émission du signal local *arm*) est exécuté. Or ce signal est instantanément diffusé. Comme l'état *idle* était en attente d'une occurrence de *arm*, la transition de l'état *idle* à l'état *Timer* est franchie. L'entrée du contrôle dans le macro-état *Timer* entraîne l'activation de l'état initial (*cnt*) de l'unique STG contenu dans *Timer*. Il n'y a plus d'autres évolutions possibles au cours de cette réaction. Le signal *AB* n'a donc pas été émis au cours de la réaction, son statut de présence est donc déterminé à *absent* pour cette réaction. Ainsi une réaction apparaît comme une séquence de transitions franchies instantanément (micro-pas). L'ordre des franchissements respecte les relations de causalités exprimées dans le syncChart : les déclencheurs causent les effets. Si la chaîne de causalité est cyclique, l'exécution n'est pas acceptable. Dans ce cas le syncChart est rejeté.

La réaction précédente n'impliquait que des préemptions fortes et des états simples. La préemption s'applique également à des macro-états. Quand *Reset* est présent, quel que soit l'état des autres signaux, le contrôle sort du macro-état *Detection* en interdisant toute évolution interne au macro-état (cette interdiction de toute exécution justifie le qualificatif fort pour la préemption). La cible de la transition étant l'état *Detection* lui-même, ce macro-état est ré-entré récursivement et les états initiaux des STG contenus dans le macro-état sont activés. Une préemption faible s'exécute différemment : avant de quitter le macro-état préempté, on exécute la réaction du corps du macro-état (d'où le qualificatif de faible). Un exemple complet est donné plus bas.

Une transition de terminaison normale n'a pas de déclencheur explicite. Une telle transition est franchie dès que tous les STG du macro-état source sont dans un état final. Par exemple, si les états *dA* et *wB* sont actifs et que *B* devient présent, la transition de *wB* à *dB* est franchie (préemption forte). Les deux STG de *WaitAandB* sont maintenant dans un état final (*dA* et *dB*, respectivement). Instantanément la transition de terminaison normale est franchie, causant l'émission de *AB* et l'activation de l'état *done*.

Illustrons à présent le franchissement de l'unique transition de préemption faible de notre exemple. Il s'agit de la transition étiquetée *disarm*. Plaçons nous dans le cas où *dA*, *wB* et *cnt* sont actifs. Quelle est la réaction lorsque *B* et *T* sont présents, sachant qu'il s'agit de la deuxième occurrence de *T* depuis que *cnt* est actif ? La transition dont le trigger est *2T* est franchie causant l'émission du signal *disarm*. Il faut donc effectuer une préemption faible sur le macro-état *Detection*. Au préalable il faut terminer les possibles réactions à l'intérieur de ce macro-état. La présence de *B* cause une évolution, expliquée précédemment, émettant *AB* et conduisant à l'état *done* à partir duquel il n'y a plus d'évolution possible interne à *Detection*. Le macro-état *Detection* est alors préempté et l'état cible de la transition activé. Il se trouve qu'ici cet état est *Detection* lui-même qui se trouve ainsi réinitialisé comme nous l'avions expliqué pour la préemption forte par le signal *Reset*.

Remarquons que lorsque plusieurs transitions ayant la même source sont franchissables simultanément, seulement celle avec la plus grande priorité (le plus petit entier associé) est franchie. Pour éviter des situations difficiles à interpréter voire illégales (conduisant à rejeter un syncChart), les SyncCharts exigent

que les préemptions fortes aient priorité sur les préemptions faibles qui elles-mêmes ont priorité sur la terminaison normale. Les éditeurs de SyncCharts vérifient le respect de cette règle.

La terminaison normale n'est pas strictement nécessaire : on peut obtenir le même comportement en utilisant une préemption faible et des signaux locaux. La figure 2 illustre une telle transformation. Il est toutefois préférable d'utiliser les transitions de terminaison normale qui sont plus lisibles et plus intuitives que leur équivalent utilisant la préemption faible.

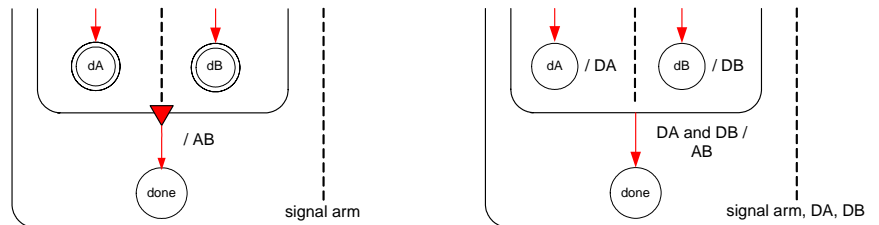


Figure 2. Terminaison normale vue comme une préemption faible

Finalement, la *suspension* est utilisée pour « geler » les évolutions à l'intérieur d'un macro-état. Quand *Inhib* est présent, *Timer* est suspendu et il ignore les possibles occurrences de *T*. Bien entendu, *disarm* ne peut pas être émis. Noter que la suspension interdit les évolutions internes au macro-état, mais n'empêche pas sa préemption.

Ces quelques exemples de réactions ont montré la complexité d'une réaction qui résulte de franchissements de transitions partiellement ordonnés. Le problème avec les machines à états finis est qu'elles sont mal adaptées à la prise en compte de la hiérarchie. C'est pour cette raison que dans la suite nous préférons exprimer le comportement d'un syncChart comme résultant de la coopération d'agents actifs appelés cellules réactives (*reactive cells*).

2.3.3. Cellules réactives

Dans un syncChart, le comportement par défaut est de rester indéfiniment dans un état. On ne peut quitter un état qu'en franchissant une transition de préemption issue de cet état (rappelons que la terminaison normale est un cas particulier de préemption) ou indirectement par préemption d'un macro-état englobant. Ainsi les agents actifs dans un syncChart sont-ils les états avec leurs transitions de sortie. Nous appelons cellule réactive l'entité constituée d'un état et ses transitions de sortie. Une cellule réactive est soit oisive (*idle*) soit active (*active*). Une cellule active teste à chaque instant les conditions de franchissement (trigger et éventuelle garde) de ses transitions. Dès qu'au moins une transition est validée, la plus prioritaire est franchie, la cellule réactive est désactivée et la cellule réactive vers laquelle pointe la transition franchie est activée. Une réaction apparaît donc maintenant comme une propagation d'activations/désactivations entre cellules réactives. Ce modèle d'exécution a été inspiré par les objets réactifs de F. Boussinot [6]. Toutefois, les objets réactifs sont plutôt conçus pour des applications distribuées et ils ne respectent pas les hypothèses synchrones.

3. Possibilités avancées des SyncCharts

3.1. Préemption immédiate

Quand on entre dans (on active) l'état wA du macro-état $WaitAandB$, si A est présent la transition déclenchée par A n'est pas franchie. Ce comportement est conforme à la règle qui dit qu'un état qui vient d'être activé se met en attente d'une *occurrence strictement future* d'un de ses triggers. Ce comportement par défaut peut être modifié par l'usage du symbole # (qui se lit *immediate*) placé en préfixe d'un trigger. La transition correspondante est alors qualifiée d'immédiate. Elle permet de prendre en compte d'éventuelles présences de signaux déclencheurs dès l'instant d'activation de l'état. Avec les transitions immédiates certains états peuvent devenir transitoires, c'est à dire qu'ils sont activés et désactivés au cours

d'une réaction. Notez que ces états restent de véritables états car ils peuvent très bien rester actifs pendant plusieurs instants.

Les triggers immédiats combinés avec les différents types de préemption conduisent à des comportements différents. La figure 3 en est une illustration sur un cas très simple. L'état p est actif et il y a occurrence de a. Le tableau précise ce qu'il se passe suivant le type de la transition t. On notera par exemple qu'en cas de préemption forte immédiate l'état q est carrément « court-circuité » dans la réaction.

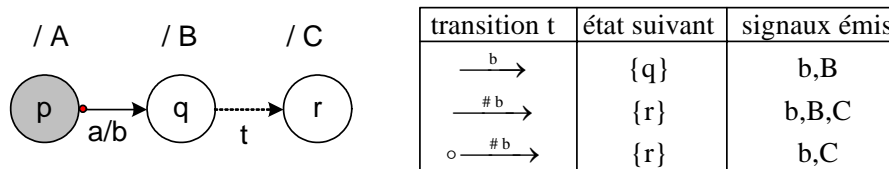


Figure 3. *Préemptions immédiates*

3.2. Réincarnation

La combinaison de boucles, de préemptions immédiates et de priorités peut conduire à des comportements surprenants au prime abord, mais complètement prévisibles. La figure 4 contient un syncChart spécialement conçu pour illustrer ces interactions complexes. Dans cet exemple nous utilisons des signaux valués qui facilitent la visualisation des micro-pas, toutefois les évolutions auraient été les mêmes en n'ayant recours qu'à des signaux purs. Le signal v est du type entier et sa fonction de combinaison est la multiplication. L'effet associé à chaque transition est l'émission du signal v avec pour valeur un nombre premier distinct. En cas d'émissions multiples dans un instant, la décomposition du résultat en facteurs premiers donnera de façon non ambiguë le nombre exact de franchissement de chaque transition dans la réaction.

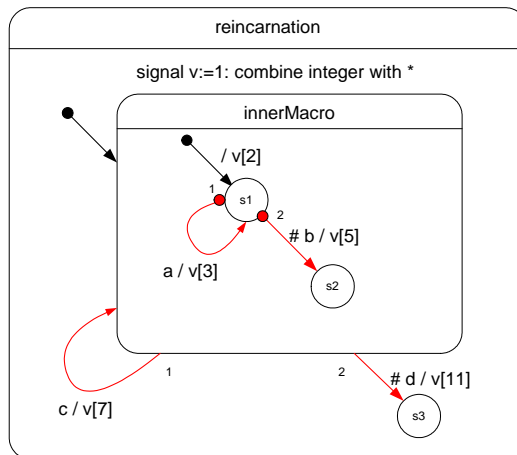


Figure 4. *Exemple de réincarnation d'états*

Supposons l'état s1 actif. Si a, b, c et d sont présents, alors v est émis avec la valeur $11550 = 2 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11$. Le nouvel état actif est s3. Cette réaction peut s'expliquer ainsi :

- Le macro-état innerMacro doit être faiblement préempté par la transition déclenchée par c qui a priorité sur la transition déclenchée par d.
- Au préalable, il faut faire régir le corps du macro-état. La forte préemption de s1 déclenchée par a, qui a priorité sur celle déclenchée par b, est franchie. Ce franchissement cause l'émission de v avec la valeur 3.
- La cible de cette transition étant l'état s1, celui-ci est réactivé. Il s'agit d'une instance fraîche de s1 (une réincarnation). Cette instance est réceptive à une occurrence *strictement* future de a et une

occurrence présente ou future de b (qui est préfixé par #). b étant présent, la transition étiquetée par b est franchie, causant la désactivation de s_1 , l'émission de v avec la valeur 5 et l'activation de s_2 .

- Puisque s_2 n'a aucune transition sortante, il n'y a plus d'évolution possible dans le corps du macro-état `innerMacro`. La préemption faible de ce macro-état est donc effectuée, émettant v avec la valeur 7. L'état `innerMacro` est désactivé et immédiatement réactivé (boucle). Il s'agit d'une instance fraîche (réincarnation) de `innerMacro` qui est réceptive à une occurrence *strictement* future de c et à une occurrence présente ou future de d (préfixé par #). La réactivation de `innerMacro` cause l'émission de v avec la valeur 2 (pseudo transition d'initialisation) et une réincarnation de s_1 est activée.
- d étant présent il va falloir effectuer la préemption faible de la réincarnation de `innerMacro`. Il faut au préalable exécuter le corps de `innerMacro`.
- Dans `innerMacro` la transition déclenchée par b est franchie, causant la désactivation de s_1 , l'émission de v avec la valeur 5 et l'activation de s_2 .
- s_2 étant sans transition de sortie, plus aucune évolution n'est possible dans `innerMacro`. La préemption faible par la transition déclenchée par d cause la désactivation de `innerMacro`, l'émission de v avec la valeur 11 et l'activation de l'état s_3 .
- Plus aucune évolution n'est possible. La réaction est terminée et v prend la combinaison par l'opération de multiplication des différentes valeurs émises soit $3*5*7*2*5*11$ qui est le résultat annoncé.

En résumé, ce résultat est tout à fait justifiable mais il est loin d'être intuitif. Nous devons proposer une approche systématique pour calculer les réactions. Ceci est l'objet de la partie qui suit.

4. Présentation formelle des SyncCharts

Les SyncCharts sont basés sur les notions d'instant et d'états. La sémantique des modèles basés sur les instants est généralement donnée en termes de séquences de paires stimuli/réaction. L'ensemble des séquences possibles est donné implicitement par une procédure d'acceptation ou par un mécanisme de construction inductif. C'est cette seconde solution qui est retenue dans cet article. Pour cela, nous avons besoin d'une définition formelle de la structure des SyncCharts (section 4.1). Nous avons également besoin de caractériser l'état d'un syncChart et de son environnement. Les concepts de *configuration* et de *contexte de signaux* sont introduits à ces fins (section 4.2). Finalement il faut donner une sémantique. Nous choisissons une *sémantique opérationnelle constructive* (section 4.3).

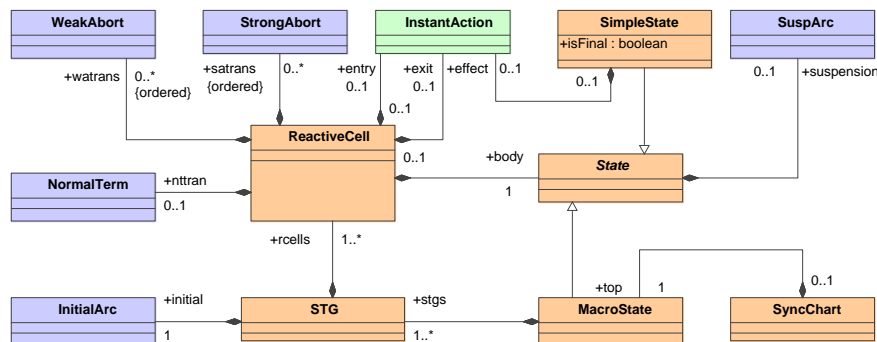


Figure 5. Méta modèle des SyncCharts (États)

4.1. Méta modèle

La syntaxe abstraite pour les SyncCharts a été définie en utilisant les notations UML. Les figures 5 et 6 couvrent les concepts de base. Une `ReactiveCell` est faite d'un corps (*body*) et d'ensembles éventuellement vides de transitions de sortie. Le *body* est un état : soit un macro-état (`MacroState`) soit un simple état

(SimpleState). *satrans* (*watrans*) est l'ensemble des transitions de préemption forte (faible) ; *nttran* est l'ensemble des transitions de terminaison normale. Cet ensemble contient au plus une transition.

Les SyncCharts bien formés respectent quelques contraintes non représentées sur les diagrammes :

- Un STG doit être un graphe connexe ;
- Une transition ne peut relier que des reactiveCells appartenant à un même STG ;
- Un simpleState final ne doit avoir ni transition sortante, ni effet associé.

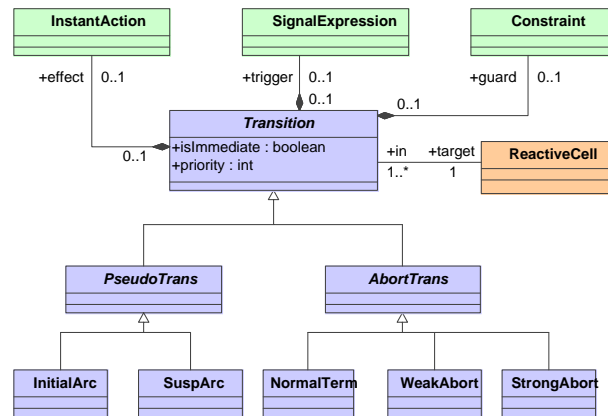


Figure 6. Méta modèle des SyncCharts (Transitions)

La structure d'un syncChart peut être représentée par un arbre qui reflète la hiérarchie des états. De façon plus précise, un macro-état contient au moins un STG, un STG est fait d'au moins une reactiveCell, une reactiveCell a un et seul descendant qui est son corps. Dans l'arbre on ne retient que les états et les STG qui alternent sur tout chemin descendant. Les STG sont notés par des cercles creux. Les états sont distingués en états simples et macro-états. Ces derniers sont eux-mêmes divisés en macro-état avec et sans suspension. La figure 7 précise ces notations et donne l'arbre correspondant au syncChart ABSync.

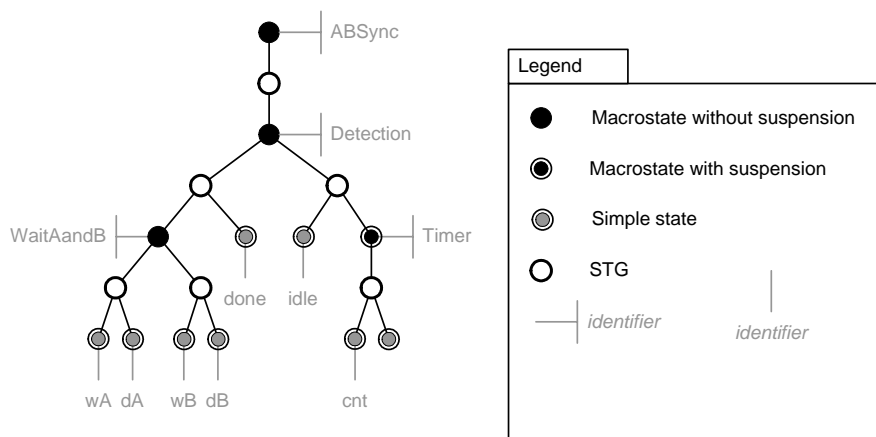


Figure 7. Arbre associé à un syncChart

4.2. Configuration et contexte de signaux

4.2.1. Configuration

Il faut pouvoir parler de l'état (global) d'un syncChart. Le mot état est déjà employé pour désigner un objet de base du modèle, il fallait trouver un autre terme. Nous empruntons le mot *configuration* qui a été introduit par Harel et Naamad [7] pour les statecharts. Une configuration est définie comme un ensemble

maximal d'états qui peuvent être simultanément actifs. Cette définition demande une adaptation pour traiter des possibles suspensions qui existent en SyncCharts.

Soit T le macro-état racine (*top*) d'un syncChart. Une configuration C de T (et donc du syncChart lui-même) doit satisfaire les règles suivantes :

1. T est dans C
2. Si M est un macro-état sans suspension dans C , alors C doit aussi contenir pour chaque STG G directement contenu dans M , un et un seul état de G .
3. Si M est un macro-état avec suspension dans C alors
 - a. Ou bien C ne contient aucun descendant de M
 - b. Ou bien C doit aussi contenir pour chaque STG G directement contenu dans M , un et un seul état de G .

Les configurations peuvent être déduites de l'arbre associé au syncChart. Cet arbre est alors considéré comme un arbre ET/OU, avec les états comme nœud ET et les STG comme nœud OU. $\{\text{ABSync}, \text{Detector}, \text{WaitAandB}, wA, wB, \text{idle}\}$, $\{\text{ABSync}, \text{Detector}, \text{WaitAandB}, wA, wB, \text{Timer}\}$, $\{\text{ABSync}, \text{Detector}, \text{WaitAandB}, wA, wB, \text{Timer}, \text{Cnt}\}$ sont des configurations du syncChart ABSync . Il convient de noter que ces ensembles sont clos vers le haut (la présence d'un état dans une configuration implique la présence de tous ses ancêtres dans cette configuration).

4.2.2. Contexte de signaux

Puisque les réactions en SyncCharts peuvent résulter d'interactions instantanées entre sous-systèmes via des signaux, ces derniers jouent un rôle essentiel dans la sémantique du modèle. A chaque réaction on associe un ensemble de signal E appelé le contexte de signaux de la réaction. L'ensemble E est partitionné en deux ensembles E^+ et E^- . E^+ est l'ensemble des signaux présents dans l'instant, E^- est l'ensemble des signaux absents dans l'instant³. Pendant une réaction un signal doit être soit présent, soit absent. Il n'est pas question de considérer lors des micro-pas qui constituent une réaction qu'un signal est parfois présent et parfois absent. Lorsqu'on calcule une réaction, seuls les signaux d'entrée ont un statut de présence initialement défini. Le statut de présence de tous les autres signaux est inconnu (\perp) et c'est justement le but du calcul de déterminer le statut exact de chacun de ces signaux. Comme en Esterel, nous considérons qu'un signal qui n'est pas signal d'entrée est présent si et seulement si il est émis dans l'instant.

4.3. Introduction à la sémantique constructive

Problème : *Étant donné un syncChart, une configuration et le statut de présence de tous ses signaux d'entrée, calculer la réaction* (c'est à dire déterminer la configuration suivante et le statut de présence de tous les signaux de sortie).

4.3.1. Principe du calcul d'une réaction

Le syncChart est considéré comme un ensemble de cellules réactives qui interagissent. Chaque cellule reçoit des signaux qui peuvent déclencher des évolutions et émettre des signaux. Tous les signaux sont instantanément diffusés.

Conceptuellement, les cellules réactives s'exécutent en parallèle. Chaque cellule réactive détermine localement son comportement (exécuter une action, franchir une transition et ainsi devenir inactive ou décider de ne rien faire dans l'instant et rester active). Les prises de décision se font en fonction du statut de présence des signaux. Afin d'assurer le déterminisme de la réaction, toutes les cellules réactives doivent être d'accord sur le statut effectif de chaque signal. Ceci suppose de trouver une solution point fixe par un dialogue entre cellules.

³ En Esterel, un ensemble de signaux simultanément présents s'appelle un événement (*event*). Nous évitons d'employer ce terme qui est généralement associé à l'occurrence d'un unique signal.

Pour résoudre ce problème, une cellule qui a des doutes concernant le statut d'un signal déclencheur doit suspendre son évolution (dans le cadre de la réaction en cours). Une autre cellule, qui s'exécute en parallèle, pourra ultérieurement lever ces doutes. Émettre un signal, donc faire passer son statut de présence d'inconnu à présent est un *fait* (certitude) qui est diffusé aux autres cellules et peut résoudre des interrogations qui étaient en suspens dans d'autres cellules. Cette coopération avec suspension, émission, reprises se poursuit jusqu'à ce que chaque cellule ait soit terminé son évolution pour l'instant considéré, soit est suspendue en attente de décision. Lorsque toutes les cellules ont terminé, on a obtenu la solution cherchée. En revanche, si au moins une cellule reste suspendue il y a un manque d'information pour conclure. Le complément d'information à fournir dépend de la sémantique choisie. Cette situation sera analysée plus loin. Au préalable il nous faut donner une interprétation précise pour les triggers qui sont les sources de blocage potentiels.

Remarque : Cette approche est voisine de celle préconisée par G. Berry [4] pour le langage Esterel. Il a introduit la notion *sémantique constructive*. L'adjectif « constructif » fait référence au fait que les statuts sont calculés par des *preuves explicites* et non par des procédures d'essai et erreur.

4.4. Algèbre de signaux

La partition du contexte de signaux E en E^+ et E^- n'est effective que lorsque la réaction a été calculée avec succès. Pendant le calcul lui-même certains signaux peuvent avoir un statut de présence inconnu (à comprendre comme « pas encore connu »). Assimiler présent à `true` et absent à `false` n'est pas suffisant. Il faut considérer un domaine $B_{\perp} = \{+, -, \perp\}$ avec une relation d'ordre partiel $\perp < +$ et $\perp < -$. Lors des calculs on ne prend que des suites croissantes pour les contextes de signaux : un signal initialement à inconnu peut passer à présent ou à absent, mais ensuite il ne peut plus changer.

La technique adoptée pour déterminer E est de propager des faits et de construire la solution incrémentalement, si elle existe. Le calcul s'appuie sur des fonctions monotones avec la relation d'ordre suivante : $E \leq E' \Leftrightarrow E^+ \subseteq E'^+ \wedge E^- \subseteq E'^-$.

Les signaux sont combinés en expressions de signaux utilisées dans les triggers. Il s'agit d'expressions sur les signaux utilisant les opérateurs `and`, `or`, `not` et des parenthèses. Pour un contexte de signaux E , une expression de signaux Φ s'évalue sur B_{\perp} : avec $\Phi ::= \sigma$ (σ est un signal) | `not` ϕ | ϕ `or` ψ | ϕ `and` ψ ,

$eval(\sigma, E) = +$ si $\sigma \in E^+$, $-$ si $\sigma \in E^-$, \perp autrement

$eval(not \phi, E) = not \ eval(\phi, E)$

$eval(\phi or \psi) = eval(\phi, E) or \ eval(\psi, E)$

$eval(\phi and \psi) = not \ eval(not \phi or not \psi)$

Les opérateurs `not` et `or` sont définis par

	<code>not</code>
<code>-</code>	<code>+</code>
\perp	\perp
<code>+</code>	<code>-</code>

<code>or</code>	<code>-</code>	\perp	<code>+</code>
<code>-</code>	<code>-</code>	\perp	<code>+</code>
\perp	\perp	\perp	<code>+</code>
<code>+</code>	<code>+</code>	<code>+</code>	<code>+</code>

4.5. Réaction d'une cellule réactive

La réaction d'une cellule réactive s'appuie sur la réaction de ses composants. Une méthode `react` a été définie pour chaque classe (`SyncChart`, `MacroState`, `SimpleState`, `STG`, `ReactiveCell`). Ces méthodes retournent un code de terminaison qui prend sa valeur dans `{DONE, DEAD, PAUSE}`.

Lorsque `react()` retourne `DONE`, cela signifie que l'objet a terminé sa réaction dans l'instant et n'aura plus rien à exécuter à l'instant suivant. Lorsque le retour est `PAUSE`, l'objet a terminé sa réaction dans l'instant, mais il a encore des choses à exécuter à l'instant suivant (l'objet reste actif). `DEAD` est une forme spéciale de `DONE` : l'objet a terminé sa réaction et se trouve dans un état final. Il n'a plus rien à faire à l'instant suivant, si ce n'est d'attendre la terminaison synchronisée des `STG` parallèles. Il sera alors désactivé par le franchissement d'une transition de terminaison normale. Donc, une fois entré dans un état final, la méthode `react()` retourne `DEAD` jusqu'à la terminaison normale.

Le pseudo code des méthodes `react` est présenté en annexe. Il convient de bien noter que notre objectif est seulement d'expliquer les micro-pas. Il ne s'agit pas de proposer une compilation efficace de SyncCharts. Notre solution s'appuie fortement sur des exécutions concurrentes avec blocage/déblocage de threads, alors que la plupart des compilateurs de programmes Esterel s'efforcent de sérialiser ces évolutions concurrentes (voir les compilateurs de S. Edwards [8] et SAXO-RT [9]).

4.6. Usage d'une fonction potentiel

Lorsque tous les threads d'exécution ont terminé (aucun n'est suspendu en l'attente d'information complémentaires) tous les signaux qui ont toujours un statut de présence inconnu, reçoivent la valeur absent. En effet on vient de *prouver* que ces signaux n'étaient pas émis pendant la réaction ; ils prennent donc le statut de présence absent. Il en est autrement lorsqu'au moins un des threads d'exécution est suspendu. Pour le langage Esterel et pour les objets réactifs, F. Boussinot [10] a étudié différentes façons de poursuivre le calcul de la réaction. Ses idées s'appliquent également aux SyncCharts. La méthode la plus expéditive, qui est celle utilisée dans les objets réactifs, est de mettre à absent tous les signaux dont le statut était encore inconnu, de reprendre les évaluations pendantes et de reporter à l'instant suivant les effets des décisions prises dans l'instant courant. SyncCharts et Esterel ne reportent pas les décisions. Ils essaient plutôt d'enrichir leur connaissance sur le contexte de signaux. Puisque tous les threads sont bloqués, il ne peut plus y avoir d'émission de signaux, donc il ne peut pas y avoir croissance de E^+ . En revanche, on peut accroître E^- . Pour cela il faut diffuser le fait qu'on est sûr qu'un signal de statut encore inconnu ne sera *certainement pas émis* dans l'instant courant. Ce type d'information peut être tiré d'une analyse de la structure du syncChart. L'idée est de construire un ensemble monotone décroissant de signaux potentiellement émis dans une réaction appelé *potentiel*. Tout signal qui n'est pas dans potentiel ne sera certainement pas émis. Ceci n'est bien sûr vrai que si le potentiel est *correct* (c'est à dire qu'il contient effectivement *tous* les signaux qui peuvent être émis).

En utilisant l'information fournie par le potentiel, on peut débloquent des threads suspendus et faire de nouveaux micro-pas. En cas de nouveau blocage on a à nouveau recours à potentiel qui peut être plus petit que précédemment et donc donner de nouvelles certitudes d'absence de signal. Si ces informations complémentaires ne débloquent aucun thread suspendu, le syncChart est dit *non constructif* (on n'a pas pu construire une solution) et en conséquence, il est rejeté. Le choix de la fonction calculant potentiel a une très grande influence sur les rejets de syncCharts. Une approximation trop grossière rejettera à tort trop de syncCharts.

La version commerciale des SyncCharts utilisée dans Esterel Studio [11] traduit les syncCharts en programmes Esterel équivalents qui sont ensuite compilés normalement. Le potentiel utilisé est donc celui des compilateurs Esterel depuis la version 5 qui utilisent une variante basée sur deux ensembles `MUST` et `CANNOT` [4]. Des recherches pour trouver des potentiels qui exploitent au mieux la structure des syncCharts sont en cours.

5. Conclusion et perspective

Cette présentation a montré que sous une apparente simplicité, les SyncCharts pouvaient cacher des comportements complexes. En fait, les SyncCharts sont une forme graphique du langage Esterel. Tout syncChart peut être traduit en un programme Esterel équivalent, cette possibilité est d'ailleurs utilisée pour compiler les SyncCharts. Pour les utilisateurs familiers avec la sémantique d'Esterel, la sémantique des SyncCharts est facile à assimiler. La réincarnation d'état, par exemple, peut être vue comme une variante graphique de la réincarnation d'un signal.

Plutôt qu'exploiter cette relation étroite entre SyncCharts et Esterel, nous avons choisi ici de présenter les SyncCharts comme un modèle synchrone indépendant, basé sur les états. La structure des SyncCharts a été définie formellement en utilisant les notations UML. Le calcul d'une réaction du modèle a été exprimé

en termes de coopération entre Cellules Réactives. Le principe d'une sémantique constructive a également été expliqué mais non formalisé⁴.

Nous pensons qu'un modèle graphique comme les SyncCharts peut constituer une bonne introduction à la programmation synchrone pour de nombreux ingénieurs plus familiers avec les représentations à base d'états qu'avec les langages de programmation. Cependant, les SyncCharts sont aussi expressifs que le langage Esterel, ils peuvent donc être trop expressifs pour un utilisateur « standard ». La réincarnation d'état est une illustration de cette richesse d'expression des comportements ; elle peut être ignorée dans la plupart des exemples pratiques. Même un utilisateur avancé peut décider, en toute connaissance de cause, de se limiter à un sous-ensemble des possibilités offertes par les SyncCharts.

Nous voulons développer une plate-forme dédiée aux SyncCharts et qui permette à l'utilisateur de

- sélectionner un sous-ensemble des constructions (personnalisation du modèle, par bridage)
- tracer les micro-pas d'une réaction (aspect didactique : explication de la sémantique)
- tracer une séquence de réaction (simulation).

Ainsi l'utilisateur pourra adapter l'expressivité de son modèle à ses besoins. Un modèle plus simple est naturellement plus facile à apprendre et peut être compilé de façon plus efficace. La figure 8 est un bon point de départ pour sélectionner les constructions et comprendre les possibles simplifications. Par exemple, le modificateur *immediate* (#) est très intéressant pour les dialogues instantanés, mais il peut aisément induire des *cycles de causalité*, qui sont particulièrement difficiles à corriger pour les non-experts. Renoncer à utiliser le *immediate* rend les réactions plus simples : aucune préemption d'un état ne peut avoir lieu à son instant d'activation. Une autre possibilité à expérimenter est le choix de la fonction potentiel. Il est même possible d'introduire des variations dans la gestion des signaux : pour éviter des cycles de causalité, on peut différer l'émission de certains signaux jusqu'à l'instant suivant, comme le font certains statecharts et le permettront les versions futures d'Esterel. Toutes ces variations sont faciles à modéliser et à implémenter avec l'approche préconisée par adaptation des méthodes `react()` décrites dans l'annexe .

6. Annexe : Calcul d'une réaction (détail)

6.1. Réaction d'un SyncChart

```
SyncChart::react()  
  read inputs  
  for all s in outputs do s.reset() done  
  top.react()
```

La méthode `reset()` appliquée à un signal mets son statut de présence à \perp .

6.2. Réaction d'un MacroState

La méthode `react()` pour un macro-état peut retourner DEAD ou PAUSE (avec DEAD < PAUSE). Cet ordre caractérise le retour d'exécution en parallèle des STG (directement) contenus dans un macro-état : dès qu'au moins un STG retourne PAUSE, le macro-état retourne PAUSE.

```
MacroState::react()  
  for all s in locals do s.reset() done  
  parallel for all g in stgs do r[g] = g.react() done  
  return r.Max()
```

⁴ Un rapport technique [12] présente une version détaillée mais ancienne de la sémantique des SyncCharts. Un autre rapport [13] explique la sémantique des S.S.M (nom donné par Esterel-Technologies aux SyncCharts) de façon analogue à celle adoptée dans ce texte, mais en détaillant les diverses constructions du modèle et en s'appuyant sur des exemples variés.

6.3. Réaction d'un STG

La méthode `react()` pour un STG peut retourner `DEAD` ou `PAUSE`. Elle fait appel à la méthode `react()` de la cellule réactive courante. Lorsqu'un STG est activé, la cellule courante est `null`, il convient donc de prendre la cellule réactive initiale du STG. Les éventuels états transitoires, qui retournent `DONE`, sont traités dans la boucle `while`.

```
STG::react()
  if curCell == null then curCell = initialCell endif
  while ( r = curCell.react() ) == DONE do
    curCell = nextCell
  done
  return r
```

6.4. Réaction d'un SimpleState

La méthode `react()` est très simple.

```
SimpleState::react()
  if isFinal then
    return DEAD
  else
    perform effect
    return PAUSE
  endif
```

6.5. Réaction d'une ReactiveCell

Cette réaction est la plus complexe car elle traite les différentes formes de préemption et les appels en profondeur. Le traitement est donné par la figure 8. Les capsules à fond coloré (ou gris) sont des endroits où les statuts de présence des signaux sont testés. Lorsqu'un trigger est évalué à \perp le traitement correspondant doit être suspendu conformément aux explications données à la section 4.5. Cette suspension se fait dans la fonction `testP()`. Elle prend 2 arguments : l'expression de signaux à analyser (trigger) et un contexte de signaux. Elle retourne le booléen `true` si l'expression est évaluée à présent, `false` si l'expression est évaluée à absent. Elle reste suspendue autrement.

```
Boolean testP(expr,E)
  do r = eval(expr,E) while ( r ==  $\perp$  )
  return ( r == + )
```

La fonction `testP()` est utilisée dans les fonctions de test de transitions. Il convient de distinguer le premier instant (celui d'activation de la cellule réactive) des autres instants. La fonction générique `testTrans` prend 3 arguments : un ensemble ordonné de transitions, un contexte de signaux et un booléen indiquant s'il s'agit du premier instant de la cellule réactive appelante. Elle retourne la première transition dont le trigger est satisfait ou `null` si aucun trigger n'est validé. Dans les autres cas, la fonction est suspendue (appel à `testP`).

```
Transition testTrans(set,E,firstInstant)
  for (t:Transition = set'first to set'last) do
    if (t.isImmediate or not firstInstant) then
      if testP(t.trigger,E) then
        return t
```

```
        end if
    endif
done
return null
```

La fonction `testTrans()` se décline en trois fonctions `testSA()`, `testWA()` et `testSUSP()` qui correspondent respectivement au test des transitions de strong abort, de weak abort et de suspension. Ces fonctions sont appelés dans la méthode `ReactiveCell::react()`. Conformément à la sémantique donnée, les transitions de préemption fortes sont testées avant toute autre réaction de la cellule. Si aucune préemption forte n'est possible, on teste une possible suspension du corps du macro-état. Si il n'y a pas suspension, alors il y a exécution du corps du macro-état (caractère récursif de la réaction). Ce n'est que lorsque le corps a terminé sa réaction que les transitions de préemption faible et éventuellement la terminaison normale sont testées.

Lorsqu'une transition doit être franchie, il faut désactiver la cellule réactive source ainsi que ses descendants. Cette désactivation se fait récursivement par les méthodes `kill()`. `ReactiveCell::kill()` appelle `kill` sur son corps, puis passe à l'état `IDLE`. `MacroState::kill()` appelle le `kill` sur ses `STG` puis se désactive. `STG::kill()` appelle `kill` sur sa `curCell` puis met `curCell` à `null`. `SimpleState::kill()` se contente de mettre l'état à `IDLE`.

7. Bibliographie

- [1] Berry G., « The foundations of Esterel », *Proofs, Language and Interaction : Essays in Honour of Robin Milner*, Stirling C., Plotkin G., Tofte M., editors. MIT Press, 2000.
- [2] André C., « Representation and Analysis of Reactive Behavior: A Synchronous Approach », *Computational Engineering in Systems Applications (CESA)*, Lille (F), July, 1996, pp. 19-29.
- [3] Harel D., « Analyse Statecharts : A Visual Formalism for Complex Systems », *Science of Computer Programming*, 8 :231-274,1987.
- [4] Berry G., « The Constructive Semantics of Pure Esterel », (revision 1999), Guyancourt (F), July 1999. <http://www.esterel-technologies.com>
- [5] Douglass B. P., *Real-Time design Patterns*. Object Technologies Series. Addison-Wesley, Reading (MA), 2003.
- [6] Boussinot F., Doumenc G., Stefani J-B., « Reactive Objects », *Ann. telecommunication*, 51(9-10):459-473,1996.
- [7] Harel D., Naamad A., « The Statemate semantics of Statecharts », *ACM Trans. Soft. Eng. Method.*, 5 (4):477-498,1996.
- [8] Edwards A. S., « An Esterel Compiler for Large Control-Dominated Systems », *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems*, 21(2):169-183,2002.
- [9] Closse E., Poize M., Poulou J., Vernier P., Weil D., « SAXO-RT: Interpreting Esterel Semantics on a Sequential Instruction Structure », *Electronic Notes in Theoretical Computer Science, SLAP 2002*, 65(5), Grenoble (F), 2002.
- [10] Boussinot F., Sugarcubes Implementation of Causality, Technical Report 3487, INRIA, September 1998.
- [11] Esterel-Technologies., Esterel Studio, V4, Reference Manual, Guyancourt (F), 2002, <http://www.esterel-technologies.com>
- [12] André C., SyncCharts: a Visual Representation of Reactive Behaviors, Technical Report 95-52, rev 96-56, I3S, Sophia Antipolis (F), April 1996.
- [13] André C., Semantics of S.S.M. (Safe State Machines), Guyancourt (F), April 2003, <http://www.esterel-technologies.com>

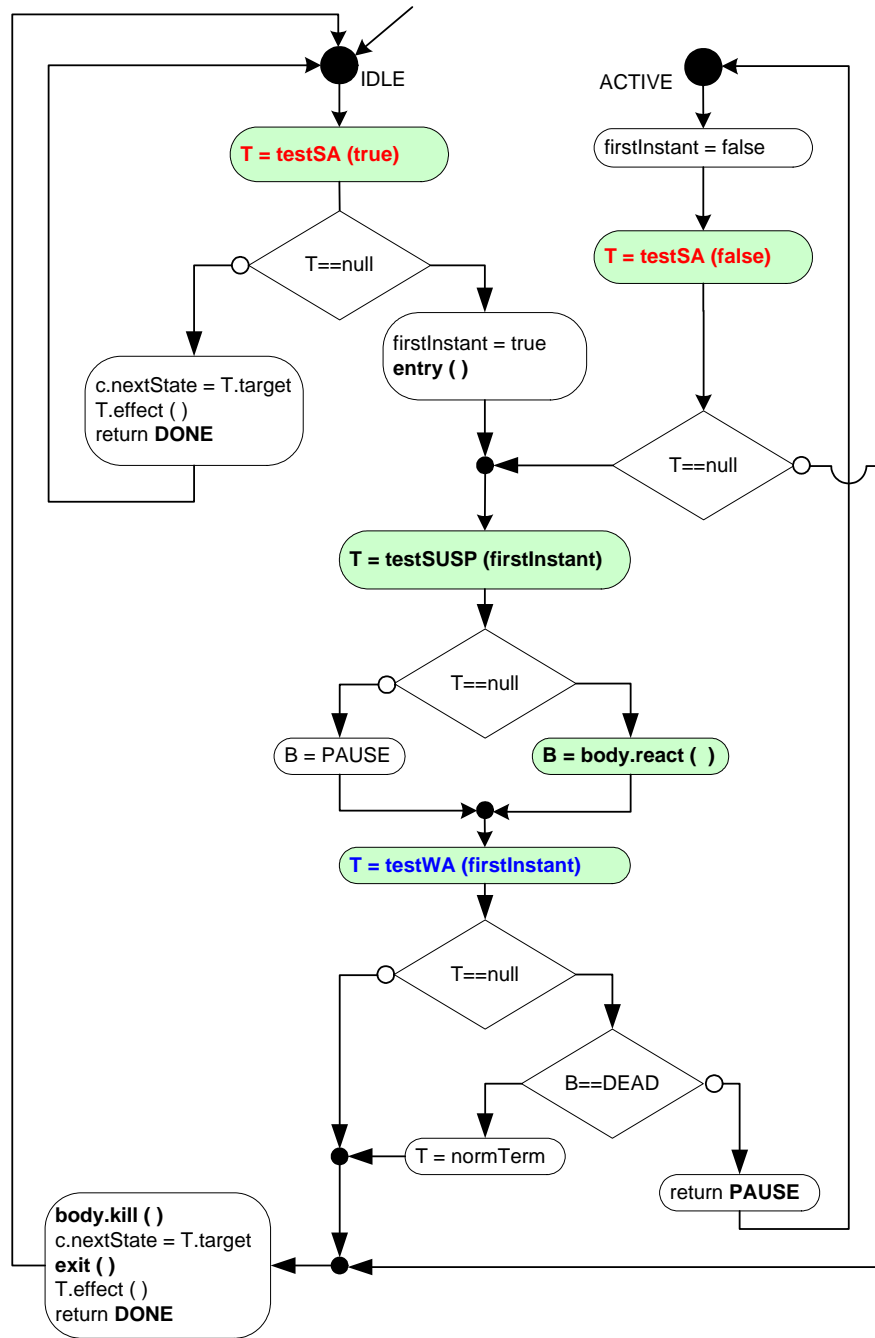


Figure 8. Réaction d'une ReactiveCell