Contribution formelle à l'expression des spécifications pour la conception de contrôleurs discrets

Laurent Piétrac, Emil Dumitrescu, and Eric Niel

Université de Lyon, INSA Lyon, Ampère (UMR5005),
F-69621 Villeurbanne, France
(e-mail : prenom.nom@insa-lyon.fr)

Résumé

La théorie du contrôle par supervision fournit un cadre formel pour la modélisation des Systèmes à Evénements Discrets (SED) et la synthèse de contrôleurs en fournissant une distinction explicite entre le système non contrôlé et les spécifications. Elle a de nombreuses extensions et applications et a donné lieu à de nombreux travaux de par le monde. L'utilisation des événements comme élément central de l'étude des systèmes est la clé de ce succès.

Elle a cependant l'inconvénient de contraindre le concepteur à étudier la dynamique des systèmes uniquement à travers des successions d'événements. Cela n'est en théorie pas une limite, mais en pratique cela peut rendre difficile l'expression des spécifications. Dans cet article nous proposons un nouveau type d'automate à états facilitant l'expression de ces spécifications tout en restant dans le cadre du contrôle de systèmes modélisés par des automates à états tels qu'utilisés dans la théorie RW.

1 Introduction

La théorie du contrôle par supervision a été initiée par les travaux de Ramadge et Wonham dans les années 80 [10]. Elle permet la synthèse de contrôleurs pour des systèmes considérés comme des systèmes à événements discrets ainsi que la formalisation de propriétés fondamentales telles que la contrôlabilité, le blocage ou l'observabilité. Le formalisme classiquement utilisé est l'automate à états, même si de nombreux travaux proposent d'utiliser des extensions des automates [8] ou des réseaux de Petri [12].

Plusieurs problèmes peuvent être abordés dans le cadre de cette théorie. Un problème classique consiste à construire un automate modélisant un procédé non contrôlé, à exprimer les contraintes (les spécifications) que doit respecter ce procédé puis à synthétiser le modèle du procédé sous contrôle respectant ces contraintes. Ces trois modèles sont donc exprimés par des automates à états générant des langages L_i et marquant des langages $L_{mi} \subseteq L_i$ représentant la réalisation des tâches du système.

En pratique, les automates sont un formalisme classique relativement intuitif pour la modélisation du procédé. Par contre la modélisation des spécifications sous la même forme d'automates nécessite de réfléchir en termes de successions d'événements autorisés ou interdits. Cela n'est pas toujours naturel et pour ceux qui pensent plutôt en termes d'états autorisés ou interdits [2] cela implique une transformation du mode de pensée. D'ailleurs, dès les débuts des travaux sur cette théorie, des mécanismes de spécification et d'étude d'états interdits ont été proposés. Les auteurs de [5] ont ainsi spécifié des états d'exclusion mutuelle de ressources sous forme d'états interdits. Repris par les auteurs de [9], ces travaux nécessitent d'identifier dans le procédé toutes les trajectoires possibles et de modifier le procédé en scindant chaque état en plusieurs. Ces travaux sont difficilement exploitables hors du cadre d'exclusion mutuelle pour lequel ils ont été conçus.

Depuis d'autres travaux ont repris l'idée de proposer un formalisme de spécification basé sur les états, notamment ceux fondés sur les logiques temporelles. Par exemple, les auteurs de [11] suggèrent l'utilisation d'une logique temporelle linéaire alors que dans [4] c'est une logique temporelle arborescente (CTL*) qui est utilisée. Dans les deux cas les expressions produites permettent de spécifier des trajectoires d'états, sans tenir compte des événements. Par contre elles ne sont pas simples à comprendre et demandent un haut niveau d'expertise pour être capable de les produire. D'autres travaux utilisent les BDD pour exprimer les spécifications [6, 3, 7]. Le but principal est de proposer une solution au problème d'explosion combinatoire, et cet objectif est bien atteint. Par contre ce type d'approche ne permet pas d'exprimer de trajectoires et les spécifications sont uniquement des spécifications d'états conjoints interdits. Dans tous les cas, l'inconvénient majeur est d'imposer un mode de pensée orienté états et de ne plus permettre de spécifications orientées événements. Les travaux de [3, 7] proposent bien de construire des expressions à partir de prédicats d'états et de prédicats d'événements mais ces derniers ne sont utilisés que pour spécifier des exclusions d'états conjoints ou interdire des événements à partir d'états particuliers. Des trajectoires mixant des états conjoints et des successions d'événements ne peuvent pas être spécifiées.

Notre objectif est au contraire de proposer un formalisme laissant au concepteur l'opportunité d'écrire des spécification suivant le point de vue qui lui semble le plus adapté ou le plus facile. Il pourra ainsi aborder les contraintes avec le point de vue états ou transitions ou même un mélange des deux. Enfin ce formalisme doit permettre la spécification de trajectoires, définies par des successions d'événements ou d'états, et notamment ne doit pas être moins expressif que le formalisme le plus généralement utilisé, i.e. l'automate à état. Cependant, par manque de place, et sans perte de généricité, nous ne considérerons pas le cas de l'exclusion mutuelle d'états dans cet article.

Dans la section suivante nous allons tout d'abord présenter les bases théoriques de la théorie du contrôle par supervision. Dans la section 3 nous aborderons ensuite le problème qui est au cœur de notre proposition : la modélisation des spécifications. Les sections 4 et 5 nous permettrons de développer notre proposition et enfin nous présenterons des exemples pédagogiques dans la section 6 .

2 Rappels sur la théorie de Ramadge et Wonham

La théorie du contrôle par supervision considère qu'un système est constitué d'un procédé à contrôler et d'un superviseur assurant le contrôle de ce procédé. Dans ce cadre, le procédé à contrôler est un générateur d'événements modélisé par un automate à état G tel que $G = (Q, \Sigma, \delta, q_0, Q_m)$ avec Q l'ensemble des états, Σ l'alphabet, $q_0 \in Q$ l'état initial, $Q_m \subseteq Q$ l'ensemble des états marqués de G. La fonction de transition δ est un sous-ensemble du produit cartésien $Q \times \Sigma \times Q$.

Le problème classiquement traité consiste à exprimer des contraintes de fonctionnement devant être absolument respectées. Ces contraintes sont modélisées sous forme d'un automate de spécification E tel que $E = (X, \Sigma', \delta', x_0, X_m)$. X, $x_0 \in X$ et $X_m \subseteq X$ représentent respectivement l'ensemble des états, l'état initial et l'ensemble des états marqués de E. L'automate E peut être construit sur un sous-ensemble Σ' de Σ. Dans ce cas il est très important de bien clarifier le sens donné au fait que certains événements soient présents dans Σ mais absents de Σ'. Deux cas sont possibles [1] : ces événements sont interdits ou au contraire la spécification ne porte pas sur ces événements et ils sont donc autorisés. Pour éviter ce choix qui peut mener à une ambiguïté, une solution simple consiste à construire la spécification sur le même alphabet que celui du procédé : $\Sigma' = \Sigma$. Quant à la fonction de transition, c'est un sous-ensemble du

produit cartésien $X \times \Sigma \times X$ (en supposant $\Sigma' = \Sigma$).

La démarche d'étude consiste ensuite à construire le modèle du procédé respectant cette spécification. Le modèle obtenu H est construit à partir d'une opération de composition de deux automates, le produit. Pour deux automates G et E, ce produit de $G \times E$ est défini par :

$$G \times E = Ac(Q \times X,\ \Sigma, \delta'',\ (q_0, x_0),\ Q_m \times X_m)$$

où l'opérateur Ac signifie que seuls les états accessibles (à partir de l'état initial) sont considérés. La fonction de transition δ'' est définie ainsi :

$$\delta''((q,x),\ \sigma) = \begin{cases} (q', x') & \text{si } (q, \sigma, q') \in \delta \wedge (x, \sigma, x') \in \delta' \\ \text{non défini} & \text{sinon} \end{cases}$$

Si tous les événements étaient contrôlables cette opération suffirait pour déterminer le procédé sous contrôle respectant la spécification. Cependant l'alphabet est scindé en deux ensembles : l'ensemble des événements contrôlables et l'ensemble des événements incontrôlables, $\Sigma = \Sigma_c \cup \Sigma_{uc}$. Cette partition oblige à vérifier la contrôlabilité du modèle obtenu puisque le contrôleur ne peut jamais interdire l'occurrence d'un événement incontrôlable. Heureusement un algorithme permet de vérifier de manière systématique cette contrôlabilité et de construire par itérations l'automate maximal respectant la spécification. Cet automate générant le langage appelé suprême contrôlable est aussi construit à partir du produit de G et de E puis par éliminations successives de certaines transitions dans le modèle obtenu. Le produit de deux automates reste donc l'opération de référence à partir de laquelle l'étude du système est possible.

3 Les spécifications

Construire un modèle de procédé ne pose pas de problème particulier. Comme dans toute modélisation il est indispensable de bien définir les frontières physiques du système. Il faut également faire attention à modéliser tous les comportements possibles tout en se limitant à ceux qui doivent être pris en compte. Ceci est vrai quel que soit le formalisme utilisé, et l'utilisation des automates à états n'est pas une difficulté en soi.

Pour la modélisation des contraintes cela est plus difficile. En effet pour construire une spécification un seul mécanisme est utilisé : l'interdiction d'événements. Cela est tout à fait naturel lorsque l'objectif est d'interdire une succession particulière d'événements, pour des raisons de sécurité. Cela l'est moins lorsque l'objectif est d'imposer une succession particulière, même si cela revient à interdire toute autre dynamique. [1] présente quatre types de spécifications qui montrent que ceci n'est pas toujours évident à mettre en œuvre :

A États illégaux : une spécification identifie certains états de G comme illégaux ;

B Partition d'état : une spécification nécessite de se souvenir comment un état particulier de G a été atteint afin de déterminer quel comportement futur est admissible ;

C Alternance d'événements : une spécification nécessite que les occurrences de deux événements alternent ;

D Sous-mot illégal : une spécification identifie comme illégaux tous les mots de $L(G)$ qui contiennent le sous-mot $s_f = \sigma_1 \ldots \sigma_n \in \Sigma^*$.

Tous ces cas peuvent être modélisés sous la forme d'un automate représentant l'interdiction de certaines occurrences d'événements. Cependant les deux premiers cas font apparaître dans leur description textuelle la notion d'état. La première n'utilise que la notion d'état alors

que la seconde utilise la notion d'état et la notion de trajectoire, c'est-à-dire de succession d'événements. Dans ces deux cas, le modélisateur doit donc transformer cette notion d'état en trajectoires menant à l'état ou en sortant. On imagine bien que ce changement de paradigme n'est pas toujours évident.

Notre objectif est donc de proposer un formalisme permettant de modéliser une spécification en mixant les paradigmes états et événements. Ce formalisme doit aussi permettre de respecter le cadre de la théorie RW et notamment ne pas remettre en cause les principes et calculs relatifs à la contrôlabilité et au langage suprême contrôlable.

4 Formalisation

Afin de permettre la modélisation d'une spécification tenant compte aussi bien des états que des événements, nous proposons un formalisme distinguant les occurrences des événements, donc les transitions du modèle du procédé. Nous définissons donc un nouveau type d'automate à états dans lequel les transitions sont associées aux transitions du procédé. Formellement pour un procédé $G = (Q, \Sigma, \delta, q_0, Q_m)$ nous avons donc un automate $E_G = (X, \Sigma, \Delta, x_0, X_m)$ dans lequel les éléments X, Σ, x_0 et X_m sont définis classiquement. Par contre la fonction de transition Δ est un sous-ensemble du produit cartésien $X \times (Q \times \Sigma \times Q) \times X$. Il s'agit donc bien d'une fonction de transition permettant de passer d'un état à l'autre. Par contre l'évolution de cet automate est conditionnée par l'évolution de l'automate G et pas uniquement par l'alphabet Σ. Ainsi, l'évolution de E, présenté dans la section 2, peut être déterminé par rapport à son alphabet, sans connaître le procédé G, alors que cela n'est pas possible pour E_G. Comme l'automate E_G peut seulement être utilisé pour définir une spécification concernant les évolutions de l'automate G, nous avons nommé ce type d'automate un *automate de spécification*.

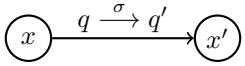
Cette définition de l'automate de spécification nécessite la définition d'un nouvel opérateur de produit entre G et E_G, noté $G \boxtimes E_G$ qui jouera le même rôle que celui entre G et E ($G \times E$). Il est défini par :

$$G \boxtimes E_G = Ac(Q \times X,\ \Sigma,\ \varphi,\ (q_0, x_0),\ Q_m \times X_m)$$

La fonction de transition φ est définie ainsi :

$$\varphi((q, x),\ \sigma) = \begin{cases} (q', x') \text{ si } (q, \sigma, q') \in \delta \wedge (x, q, \sigma, q', x') \in \Delta \\ \text{indéfini sinon} \end{cases}$$

La figure 1 montre un exemple de procédé G et d'automate de spécification E_G ainsi que l'automate $G \boxtimes E_G$. Pour évoquer le lien entre les transitions de G et celles de E_G nous avons choisi de représenter chaque élément $(x, q, \sigma, q', x') \in \Delta$ sous la forme :



Remarquons que le résultat de $H = G \boxtimes E_G$ est un automate classique générant un langage $\mathcal{L}(H)$ et marquant le langage $\mathcal{L}_m(H)$. Il est évident que par construction $\mathcal{L}(H) \subseteq \mathcal{L}(G)$ et $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(G)$. La condition de contrôlabilité peut donc s'exprimer sous la forme classique :

Définition 4.1 (Contrôlabilité)**.** $\mathcal{L}_m(H)$ est dit contrôlable par rapport à $\mathcal{L}(G)$ et Σ_{uc} si :

$$\overline{\mathcal{L}_m(H)}.\Sigma_{UC} \cap \mathcal{L}(G) \subseteq \overline{\mathcal{L}_m(H)}$$

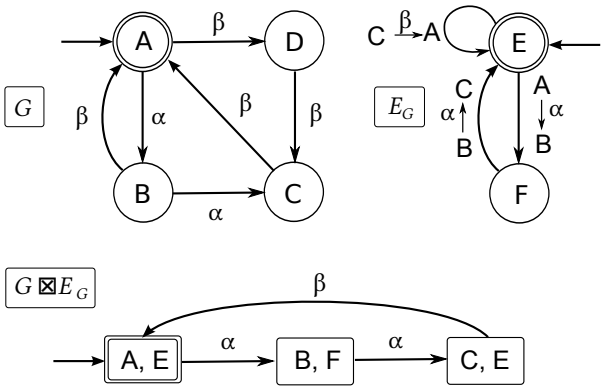


FIGURE 1 – Exemple de principe

De même la condition d'existence d'un superviseur non bloquant n'est pas modifiée. Quant à l'algorithme 1 de recherche de l'automate générant le langage suprême contrôlable, il n'est modifié que dans la phase de calcul de H_0 puisque les éléments de départs ne sont pas les mêmes. Par contre, l'algorithme 1, présenté page suivante, montre que les autres étapes de calcul sont les mêmes que l'algorithme classique.

Algorithme 1 : Algorithme modifié de calcul de H

Entrées : $G = (Q, \Sigma, \delta, q_0, Q_m)$ et $E_G = (X, \Sigma, \Delta, x_0, X_m)$
Sorties : $H, K^{\uparrow c}, \overline{K^{\uparrow c}}$

début
  initialisation $H_0 = (Y_0, \Sigma, \varphi, (q_0, x_0), Y_{0,m}) \longleftarrow G \boxtimes E_G$ *avec* $Y_0 \subseteq Q_0 \times X_0$;
    Par hypothèse, $\mathcal{L}_m(H_0) = K$ *et* $\mathcal{L}(H_0) = \overline{K}$;
    Les états de H_0 *sont désignés par des paires* (q, x);
  $i \longleftarrow 0$;
  tant que H_{i+1} *n'est pas un automate vide* **faire**
    $Y_i' \longleftarrow \{(q,x) \in Y_i \mid \Gamma_G(x) \cap \Sigma_{uc} \subseteq \Gamma_{H_i}((q,x))\}$;
      avec $\Gamma_G(x) = \{\sigma \in \Sigma \mid \delta(q,\sigma)!\}$;
      et $\Gamma_{H_i}((q,x)) = \{\sigma \in \Sigma \mid \varphi_i((q,x),\sigma)!\}$;
    $\varphi_i' \longleftarrow \varphi_i | Y_i'$;
    $Y_{i,m}' \longleftarrow Y_{i,m} \cap Y_i'$;
    $H_{i+1} \longleftarrow Trim(Y_i', \Sigma, \varphi_i', (q_0, x_0), Y_{i,m}')$;
      avec $H_{i+1} = (Y_{i+1}, \Sigma, \varphi_{i+1}, (q_0, x_0), Y_{i+1,m})$;
    si $H_{i+1} = H_i$ **alors**
      $H \longleftarrow H_{i+1}$;
      $K^{\uparrow c} \longleftarrow \mathcal{L}_m(H_{i+1})$;
      $\overline{K^{\uparrow c}} \longleftarrow \mathcal{L}(H_{i+1})$;
      STOP;
    sinon
      $i = i + 1$

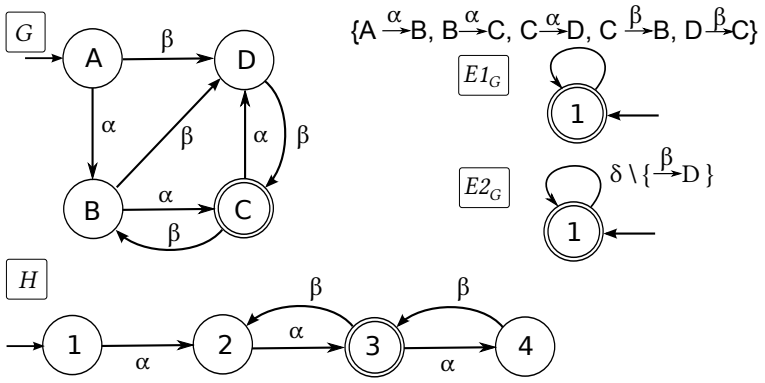


FIGURE 2 – Exemple de simplification syntaxique

Le formalisme de spécification permet donc de rester dans le cadre de la théorie RW tout en offrant un formalisme de spécification plus précis. En effet il permet d'interdire une transition particulière sans avoir à se soucier des trajectoires passées. Par contre cette précision peut se répercuter sur la taille du modèle lorsqu'il s'agit d'autoriser plusieurs transitions portant le même événement. Pour éviter ceci nous proposons d'étendre la notation graphique afin de simplifier l'écriture des modèles.

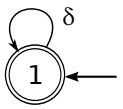
5 Abréviations

L'idée de base est de proposer une écriture permettant de définir simplement un ensemble de transitions de E_G. Par exemple la figure 2 présente un automate G dont nous cherchons à interdire les occurrences de β menant à l'état D. L'automate H, présenté sur la même figure, représente G sans les trajectoires interdites. Il peut être obtenu par le produit $G \boxtimes E1_G$. Nous remarquons que cet automate a une structure très simple, un seul état, mais de nombreuses transitions, toutes celles autorisées dans G.

Nous cherchons à obtenir une écriture plus compacte, nous permettant de définir des ensembles de transition. L'automate $E2_G$ utilise ainsi l'expression $\delta \setminus \{\xrightarrow{\beta} D\}$. Cette expression permet de définir de manière compacte l'ensemble $\{A \xrightarrow{\alpha} B, B \xrightarrow{\alpha} C, C \xrightarrow{\alpha} D, C \xrightarrow{\beta} B, D \xrightarrow{\beta} C\}$. Elle est construite en trois parties :

— L'opérateur $\setminus$ est l'opérateur classique de la théorie des ensembles. $A \setminus B$ représente l'ensemble A privé de l'ensemble B. Les opérateurs d'union ($\cup$) et d'intersection ($\cap$) d'ensembles sont aussi utilisables ;

— δ représente l'ensemble des transitions de G. L'objectif de la spécification est de restreindre les évolutions de G. Donc dans le cas où aucune évolution n'est interdite, toutes les transitions de G sont autorisées dans les transitions de E ce qui se représenterait par :



— $\{\xrightarrow{\beta} D\}$ est une notation proposée pour définir l'ensemble des transitions menant à l'état

D avec l'événement β.

Cette dernière écriture compacte peut être généralisée sous la forme $\{Q_1 \xrightarrow{\Sigma'} Q_2\}$ où Q_1 et Q_2 (resp. Σ') représentent des sous-ensembles de Q (resp. de Σ). Une transition (q_1, σ, q_2) de δ fait partie de cet ensemble si q_1 appartient à Q_1, σ appartient à Σ' et q_2 appartient à Q_2. Formellement :

$$\{Q_1 \xrightarrow{\Sigma'} Q_2\} \stackrel{\text{def}}{=} \{(q_1, \sigma, q_2) \in \delta \mid q_1 \in Q_1 \wedge \sigma \in \Sigma' \wedge q_2 \in Q_2\}$$

Cette expression peut elle même être abrégée, pour faciliter son écriture en utilisant les trois règles suivantes :

— Tout ensemble Q_1 ou Q_2 peut être omis s'il est égal à Q ;

— L'ensemble Σ' peut être omis s'il est égal à Σ ;

— Lorsqu'un des ensembles Q_1, Q_2 ou Σ' ne contient qu'un seul élément, les accolades symbolisant cet ensemble peuvent être omises.

La table 1 présente enfin quelques exemples de cas particuliers de cette notation abrégée.

TABLE 1 – Exemples d'abréviations

Syntaxe	Abréviation	Ensemble équivalent
$\{e\} \xrightarrow{\Sigma} Q$	$e \longrightarrow$	$\{(q_1, \sigma, q_2) \in \delta \mid q_1 = e\}$
$Q \xrightarrow{\Sigma} \{s\}$	$\longrightarrow s$	$\{(q_1, \sigma, q_2) \in \delta \mid q_2 = s\}$
$\{e\} \xrightarrow{\Sigma} \{s\}$	$e \longrightarrow s$	$\{(q_1, \sigma, q_2) \in \delta \mid q_1 = e \wedge q_2 = s\}$
$\{e\} \xrightarrow{\{\alpha\}} Q$	$e \xrightarrow{\alpha}$	$\{(q_1, \sigma, q_2) \in \delta \mid q_1 = e \wedge \sigma = \alpha\}$
$Q \xrightarrow{\{\alpha\}} \{s\}$	$\xrightarrow{\alpha} s$	$\{(q_1, \sigma, q_2) \in \delta \mid q_2 = s \wedge \sigma = \alpha\}$

Les deux premiers cas de la table 1 permettent de spécifier simplement toutes les transitions de sortie ou d'entrée dans un état particulier. Le troisième cas est utilisé pour spécifier toutes les transitions entre deux états e et s. Enfin les deux derniers cas permettent de spécifier toutes les transitions partant (respectivement menant) d'un état s (resp. e) avec un événement α.

D'autres cas peuvent bien sûr être envisagés. Le tableau 2 présente deux abréviations supplémentaires ne correspondant pas aux règles précédentes.

TABLE 2 – Deux abréviations particulières

Syntaxe	Abréviation	Ensemble équivalent
$Q \xrightarrow{\Sigma} Q$	δ	$\{(q_1, \sigma, q_2) \in \delta\}$
$Q \xrightarrow{\{\alpha\}} Q$	α	$\{(q_1, \sigma, q_2) \in \delta \mid \sigma = \alpha\}$

La première ligne de ce tableau permet de suivre n'importe quelle évolution du procédé G.

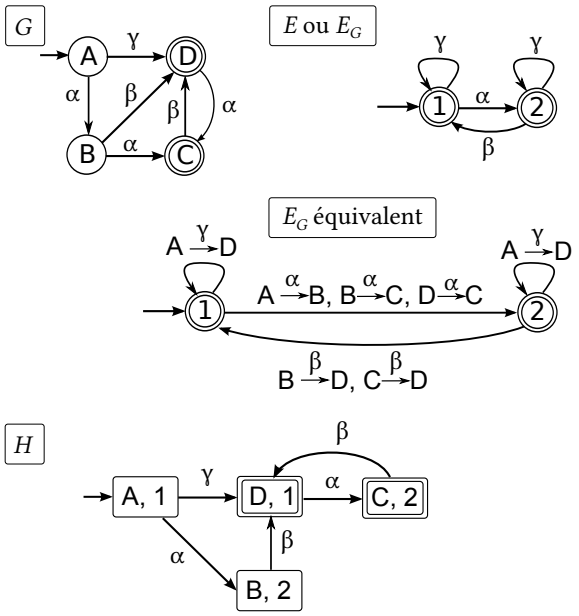


FIGURE 3 – Examples of equivalent models

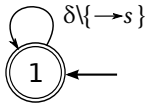


FIGURE 4 – Exemple d'interdiction d'un état

La deuxième ligne est particulièrement intéressante. En effet, elle permet de construire un automate E_G graphiquement et sémantiquement (i.e. $G \times E = G \boxtimes E_G$) équivalent à un automate E. La figure 3 présente ainsi un exemple de procédé G. La spécification d'alternance des événements α et β est exprimée avec l'automate classique E, en haut à droite de la figure. Sans utiliser d'abréviation, l'automate de spécification E_G équivalent est présenté en dessous de E. Avec les abréviations, la représentation de E_G devient plus simple : elle est la même que celle de E. La figure 3 présente également le modèle $H = G \times E = G \boxtimes E_G$.

6 Quelques exemples d'usage

Reprenons les quatre cas de spécification évoqués dans le section 3 .

Pour traiter le cas A, l'approche classique consiste à supprimer « à la main » l'état à interdire. Grâce au formalisme proposé, nous pouvons écrire un automate décrivant cette spécification. La figure 4 présente ainsi l'exemple d'une spécification interdisant l'état s, en utilisant les abréviations 1 et 4 de la table 1 . Il est intéressant de remarquer que cette structure est générique, donc simple à réutiliser, puisque seul le nom de l'état change pour plusieurs spécifications de même type.

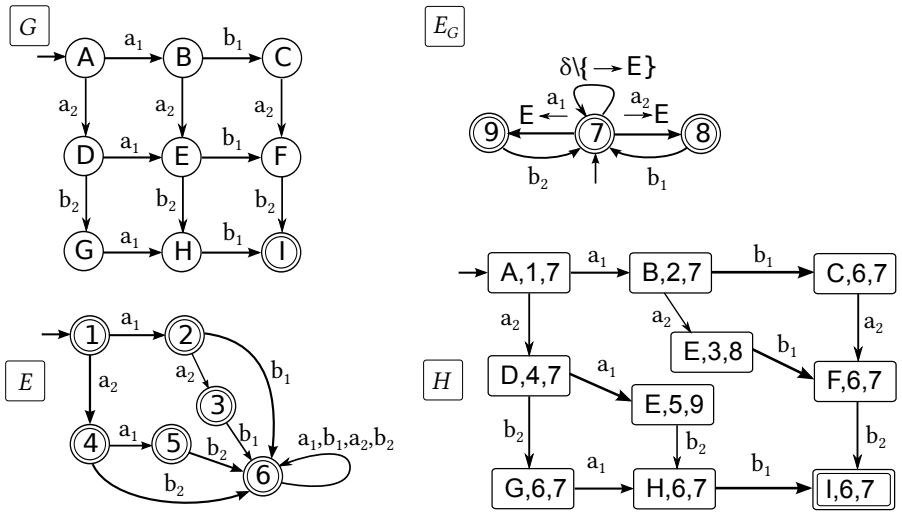


FIGURE 5 – Exemple de partition d'états

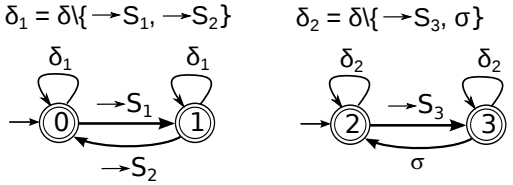


FIGURE 6 – Spécifications d'alternance

[1] présente le cas B à travers l'exemple de la figure 5 . L'objectif est d'imposer b_1 si l'état E a été atteint par la trajectoire a_1a_2 et d'imposer b_2 s'il a été atteint suite à a_2a_1. L'automate E présente la spécification permettant cette distinction des trajectoires ayant menées à l'état E. L'automate E_G permet de spécifier les mêmes contraintes. Les deux transitions $\xrightarrow{a_1} E$ et $\xrightarrow{a_2} E$ permettent de distinguer les deux trajectoires à partir du dernier événement ayant mené à l'état E. La structure de l'automate E_G est beaucoup plus simple que celle de E. Le modèle H représente le procédé respectant les spécifications E ou E_G. Le nom des états représente les états d'origine de G, E et E_G suivant que H soit obtenu par $G \times E$ ou par $G \boxtimes E_G$.

La figure 3 a déjà permis de présenter un exemple d'alternance d'événements (cas C). Par analogie il est possible de spécifier également des alternances d'états ou des alternances état – événement. La figure 6 présente ainsi un exemple de spécification d'alternance entre deux états S_1 et S_2 (à gauche) ou d'alternance entre un état S_3 et un événement σ (à droite) d'un procédé G ayant comme fonction de transition δ.

Comme le cas C, le cas D peut être modélisé par un automate E_G similaire à ce que nous pourrions faire avec un automate E. Là encore il est possible, par analogie, de spécifier un automate permettant d'interdire une trajectoire d'états. La figure 7 présente ainsi un automate (haut de la figure) autorisant tous les mots sur l'alphabet $\{a, b, c, d\}$ sauf ceux contenant le sous-mot *abcd*. L'automate du bas présente quant à lui un automate autorisant tous les comportements d'un procédé dont l'ensemble d'état est $\{S, T, O, P\}$ sauf ceux qui permettent de

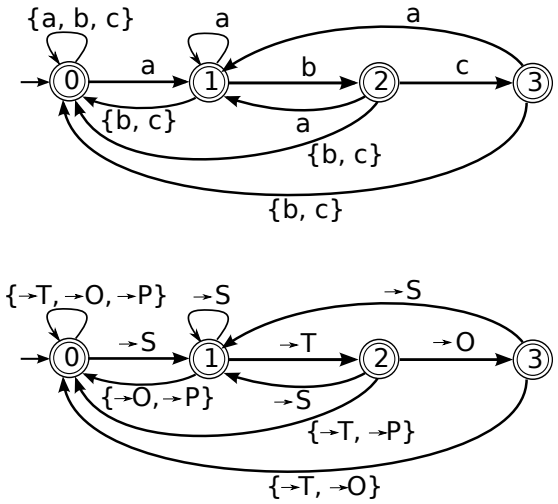


FIGURE 7 – Exemple de trajectoire d'états illégaux

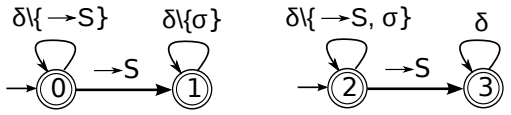


FIGURE 8 – Interdiction de l'événement σ avant ou après l'état S

passer successivement par S, T, O puis P.

La possibilité d'utiliser conjointement la notion d'état et d'événement dans une spécification permet de penser à d'autres exemples que ceux présentés précédemment.

Par exemple, il est possible d'interdire toute occurrence d'un événement σ dans toutes les trajectoires si le procédé est passé dans l'état S (gauche de la fig. 8) ou au contraire avant l'état S (droite de la fig. 8).

Sur le même type d'idée il est également possible d'interdire toutes les trajectoires passant par un état S après l'occurrence d'un événement α (gauche de la fig. 9) ou avant cette occurrence (droite de la fig. 9).

Plutôt que d'interdire un état suite à événement (ou réciproquement), il est également possible d'imposer les trajectoires passant par un état S si l'événement σ a eu lieu avant et *vice versa* (figure 10).

Bien évidemment d'autres exemples mixant les points de vue état et événement peuvent

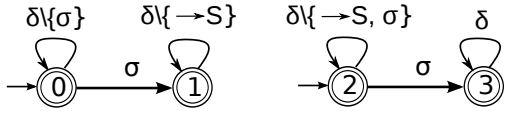


FIGURE 9 – Interdiction de l'état S avant ou après l'événement σ

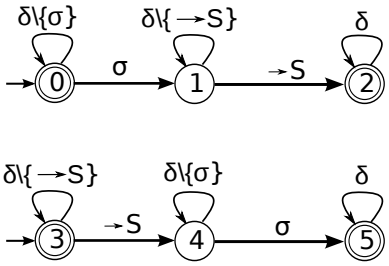


FIGURE 10 – Exemples de trajectoires illégales mixtes

être envisagés.

7 Conclusion

Dans cet article nous avons proposé un nouveau type d'automates de spécification. Ce nouveau formalisme nous permet d'aborder les spécifications aussi bien avec le point de vue états que transitions. Il permet donc à ceux qui le souhaitent de conserver leurs habitudes, quelles qu'elles soient. Ce formalisme permet en outre de mélanger les deux points de vues. Ceci permet d'obtenir parfois des modèles plus concis et simples à interpréter, mais surtout il permet d'obtenir des modèles formels dont l'expression peut être obtenue plus directement à partir de leur expression textuelle. En outre les modèles ne sont jamais plus grands puisque dans le pire des cas, les modèles construits sont les mêmes que dans l'approche classique orientée événements. Nous pouvons donc supposer un passage à l'échelle, sur des exemples industriels, sans difficulté. L'autre avantage de notre proposition est qu'elle permet de rester dans le cadre de la théorie en ce qui concerne la génération des langages et donc la contrôlabilité comme l'observabilité (non traité dans cet article) des trajectoires. Elle propose également un opérateur de produit du procédé et de la spécification.

Nous n'avons pas abordé dans cet article les aspects compositionnels. Le modèle du procédé comme de la spécification sont la plupart du temps réalisés par composition de modèles locaux. Nous aborderons la possibilité d'écrire des spécifications à partir des états ou des transitions de ces composants dans de futurs travaux.

Références

[1] Christos G. Cassandra and Stéphane Lafortune. *Introduction to Discrete Event Systems.* Springer, second edition, 2008.

[2] S. T. J. Forschelen, J. M. van de Mortel-Fronczak, R. Su, and J. E. Rooda. Application of supervisory control theory to theme park vehicles. In *10th International Workshop on Discrete Event Systems, WODES*, 2010.

[3] K. G. M. Jacobs, J. Markovski, D. A. Van Beek, J. E. Rooda, and L. J. A. M. Somers. Specifying state-based supervisory control requirements. Technical report, Eindhoven University of Technology, Department of Mechanical Engineering, Systems Engineering Group, 2009.

[4] S. Jiang and R. Kumar. Supervisory control of discrete event system with CTL* temporal logic specifications. *SIAM Journal of Control and Optimization*, 44(6) :2079–2103, 2006.

[5] F. Lin, A. F. Vaz, and W. Murray Wonham. Supervisor specification and synthesis for discrete event system. *International Journal of Control*, 48(1) :321–332, 1988.

[6] Chuan Ma and W. Murray Wonham. A symbolic approach to the supervision of state tree structures. In *13th Mediterranean Conference on Control and Automation*, pages 908–913, Limassol, Cyprus, June 27-29 2005.

[7] J. Markovski, K. G. M. Jacobs, D. A. Van Beek, L. J. A. M. Somers, and J. E. Rooda. Coordination of resources using generalized state-based requirements. In *10th International Workshop on Discrete Event Systems, WODES*, pages 287–292, 2010.

[8] S. Miremadi, K. Akesson, B. Lennartson, and M. Fabian. Supervisor computation and representation : A case study. In *10th International Workshop on Discrete Event Systems (WODES)*, 2010.

[9] Yeong-Chang Ou and Jwusheng Hu. A modified method for supervisor specification and synthesis of a class of discrete event systems. In *Proceedings of the American Control Conference*, pages 1981–1985, 1999.

[10] Peter J. G. Ramadge and W. Murray Wonham. Supervision of discrete event processes. In *21st IEEE Conference On Decision and Control, CDC*, pages 1228–1229, New York, December 1982.

[11] K. T. Seow. Integrating temporal logic as a state-based specification language for discrete-event control design in finite automata. *IEEE Transactions on Automation Science and Engineering*, 4(3) :451–464, 2007.

[12] W. Murray Wonham. Supervisory control of discrete-event systems. ECE 1636F/1637S 2009-10. departement of Electrical and Computer Engineering, University of Toronto, 2009.