Introduction à la synthèse de superviseur

Mathilde Machin, Jérémie Guiochet, David Powell, Hélène Waeselynck

LAAS-CNRS

Introduction à la synthèse de superviseur

Rapport LAAS n°13067

Mathilde Machin
Jérémie Guiochet
David Powell
Hélène Waeselynck

19 Février 2013

Sommaire

Introduction

La théorie du contrôle par supervision (*Supervisory Control Theory*, SCT), ou synthèse de superviseur, a été inventée en 1987 par P.Ramadge et W.M.Wonham [RW87, W04]. La SCT s'appuie sur la théorie des langages, et a recours aux machines à état. Elle vise à synthétiser un superviseur correct par construction, qui veille à ce que le comportement d'un système à événements discrets reste admissible vis-à-vis d'une spécification. La synthèse requiert la modélisation du système à contrôler et de la spécification, et ne donne qu'un modèle pour la commande, qu'on appelle superviseur. La spécification peut être de type fonctionnel ou non-fonctionnel (par exemple des propriétés de sécurité-innocuité).
Le système à contrôler a des actionneurs et des capteurs. La spécification restreint les possibilités en interdisant certaines séquences d'actions/observations. Comme une observation provient de l'environnement, le superviseur ne peut pas l'interdire, il ne peut inhiber que des actions. Sa conception nécessite donc la connaissance d'un modèle du système. La recherche des actions à interdire est faite automatiquement par un algorithme de synthèse.
Le superviseur ne doit pas être confondu avec le contrôleur. Un superviseur est ici un objet théorique, qui peut inhiber, interdire des actions, mais ne prend pas l'initiative de les déclencher. Le superviseur n'est donc pas directement implémentable ; il peut être considéré comme une spécification ou un sous-ensemble du contrôleur qui, lui, a l'initiative des commandes.

Ce document a pour but d'exposer les principes théoriques de la synthèse de superviseur, et de donner des pistes en vue de l'implémentation du contrôleur.
Le cadre théorique est défini dans la section 2. La section 3 expose les principes de la synthèse. Des exemples sont traités en section 3, notamment en utilisant l'outil de synthèse TCT [W04]. La section 4 s'intéresse à l'implémentation du superviseur de Wonham.

1 Formalisme

Quelques éléments théoriques sont nécessaires pour comprendre le mécanisme de la synthèse de superviseur.

1.1 Automate fini

1.1.1 Définition

On définit un automate fini par le quintuple $(Q, \Sigma, \delta, q_0, Q_m)$ avec

- Q l'ensemble fini des états
- Σ l'ensemble fini des évènements, ou alphabet. Un événement étant un symbole.
- δ la fonction transition
- q_0 l'état initial, élément de Q
- Q_m l'ensemble des états marqués ou marquage (défini ci-après)

On définit Σ^* l'ensemble des séquences finies des éléments de Σ, c.-à-d. l'ensemble des mots faisables avec l'alphabet Σ. Σ^* contient la séquence vide, notée ε.
La fonction transition est définie par

$$\delta : Q \times \Sigma \nrightarrow Q$$

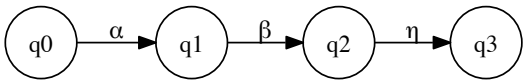

Figure 1. Exemple d'automate

Exemple : Sur l'automate de la Figure 1, $\delta(q_0, \alpha)=q_1$
La fonction transition est partielle, elle n'est pas définie pour tout élément de Q x Σ. Les états n'ont pas une transition sortante pour chaque événement. En d'autres termes, certaines séquences ne sont pas réalisables dans l'automate.
Exemple : Sur l'automate de la Figure 1, $\delta(q_0, \beta)$ et $\delta(q_1, \alpha)$ ne sont pas définis.
Le fait que les transitions soient définies par une fonction, implique qu'on ne s'intéresse ici qu'aux automates déterministes. Etant donné un état de départ et un événement, il n'y a qu'une transition sortante, et donc qu'un seul état d'arrivée.
On étend la définition de δ par

$$\delta : Q \times \Sigma^* \rightarrow Q$$

Exemple : Sur l'automate de la Figure 1, $\delta(q_0, \alpha\beta\eta)=q_3$

Le fait de marquer ou non un état est un choix de modélisation. Un état marqué représente un état où une tache s'achève, où un objectif est atteint. Les états marqués peuvent être vus comme des états stables, alors que les états non-marqués sont des états intermédiaires, transitoires durant l'accomplissement des taches. Par exemple pour une tache cyclique, l'état initial est généralement marqué. Les états marqués sont parfois appelés états finaux ou états d'acceptation.

Notations : On notera un événement quelconque $\sigma \in \Sigma$, et une séquence $s \in \Sigma^*$.
Les états marqués sont notés par un double cercle (ou parfois une flèche sortante).
L'état initial est représenté en bleu (ou gris) dans la suite (ou numéroté 0 avec TCT)

1.1.2 Sémantique et hypothèses

Dans le cadre de la théorie des langages, la sémantique des automates est telle que, au départ d'un état, si aucune transition n'est étiquetée σ, alors il est impossible que l'évènement σ advienne.
Exemple : On veut modéliser la vie d'un chat. Sa maison dispose de 2 chatières, une pour l'entrée, une pour la sortie. Sur chacune des chatières, un capteur envoie un front montant à chaque ouverture. On peut modéliser ce système par l'automate de la Figure 2.

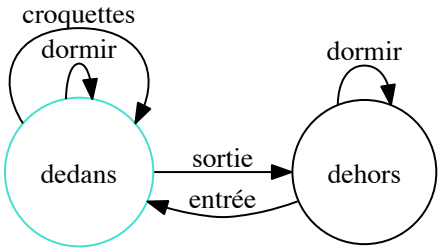

Figure 2. Vie d'un chat

Quand le chat est à l'intérieur, il n'est pas possible que la chatière d'entrée s'ouvre, ainsi il n'y a pas de transition étiquetée 'entrée' au départ de l'état 'dedans'. Le chat peut dormir partout, en revanche il ne peut manger des croquettes qu'à l'intérieur.
On remarque que l'absence d'un événement au départ d'un état est très porteuse de sens. En pratique on lit sur un automate autant les transitions présentes qu'absentes. On peut résumer cette sémantique par "un événement non mentionné n'advient pas".

On associe à l'automate des hypothèses temporelles classiques :

- évènements instantanés
- évènements asynchrones (2 événements ne peuvent pas advenir simultanément dans un même automate)

1.1.3 Propriété d'un automate : Émondé

On définit l'ensemble des états accessibles par

$$Q_r = \{ q \in Q \mid \exists s \in \Sigma^*, \delta(q_0,s) = q \}$$

Et l'ensemble des états co-accessibles par

$$Q_{cr} = \{ q \in Q \mid \exists s \in \Sigma^*, \delta(q,s) \in Q_m \}$$

Un automate est dit accessible si $Q_r = Q$, c.-à-d. si tous ses états sont accessibles.
Un automate est dit co-accessible si $Q_{cr} = Q$, c.-à-d. si on peut atteindre un état marqué depuis chaque état. Un automate accessible et co-accessible est dit *émondé* (ou *trim* en anglais). En pratique si un automate est émondé, il est débarrassé de tout ce qui est inutile : les états où il ne passera jamais, les branches qui ne servent pas à accomplir, à avancer une tache.

1.2 Lien entre automate et langage

Un automate peut être vu comme un mécanisme décrivant un langage. Un mot du langage est alors une séquence d'évènements, partant de l'état initial.

Un automate fini est représenté par deux langages.
Soit l'automate $A=(Q, \Sigma, \delta, q_0, Q_m)$. Le langage L(A) (parfois appelé langage généré) est l'ensemble des séquences réalisables dans l'automate.

$$L(A) = \{s \in \Sigma^* \mid \delta(q_0,s) \text{ est défini}\}$$

Le langage généré représente la structure des états et des transitions étiquetées. Il ne prend pas en compte le marquage des états. Le langage généré ne représente que la partie accessible de l'automate.
Le langage marqué est l'ensemble des séquences se terminant dans un état marqué.

$$L_m(A) = \{s \in L(A) \mid \delta(q_0,s) \in Q_m\}$$

Le langage marqué traduit le marquage de l'automate (ensemble des états marqués). Le langage marqué ne dépend que de la partie accessible et co-accessible de l'automate c.-à-d. qu'on peut réduire l'automate à sa partie émondée pour construire son langage marqué.
Un automate génère un langage généré unique et un langage marqué unique.

Un langage L est dit *reconnu* par un automate A si il est le langage marqué de A, c.-à-d. si $L_m(A)=L$. La reconnaissance d'un langage dépend du marquage et de la structure de l'automate. Un langage peut être reconnu par plusieurs automates. Quand un langage est reconnu par (au moins) un automate, il est dit régulier ou rationnel. Les langages de cette classe peuvent être décrits (de manière non unique) par des expressions régulières ([O10]).

Exemple : Ecriture d'un langage par les ensembles et les expressions régulières
Soit l'automate A, Figure 3

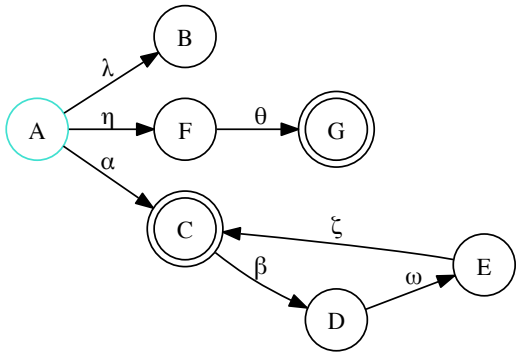

Figure 3. Automate A

$L(A) = \{ \lambda, \eta, \eta\theta, \alpha, \alpha\beta, \alpha\beta\omega, \alpha\beta\omega\zeta, \alpha\beta\omega\zeta\beta, \alpha\beta\omega\zeta\beta\omega, \alpha\beta\omega\zeta\beta\omega\zeta, \ldots\}$
$L_m(A) = \{\eta\theta, \alpha, \alpha\beta\omega\zeta, \alpha\beta\omega\zeta\beta\omega\zeta, \ldots\}$
Qu'on peut écrire par les expressions régulières $L_m(A) = \eta\theta + \alpha(\beta\omega\zeta)^*$

Définition : Préfixe-clos
L'ensemble des préfixes du langage L, est le langage

$$\overline{L} = \{ s \in \Sigma^* \mid \exists\, t \in \Sigma^*,\ st \in L \}$$

Par définition de l'ensemble des préfixes, on a $L \subseteq \overline{L}$
L est dit préfixe-clos (ou clos) ssi $\overline{L}=L$.

Si L est généré par un automate A alors L est préfixe-clos, c.-à-d. $\overline{L}=L$. En effet dans un automate, les mots du langage s'interprètent comme des séquences d'évènements. Deux évènements ne pouvant pas avoir lieu simultanément, une séquence ne peut exister que si toutes les étapes intermédiaires existent, c.-à-d. que tous les préfixes font partie du langage.

Exemple : Préfixe-clos
Considérons l'alphabet $\Sigma = \{ \alpha , \beta , \eta , \lambda \}$
Et le langage $L = \{ \alpha\beta\eta\alpha , \alpha\eta\lambda \}$
L'ensemble des préfixes de L est $\overline{L} = \{ \alpha , \alpha\beta , \alpha\beta\eta , \alpha\beta\eta\alpha , \alpha\eta , \alpha\eta\lambda \}$
$L \neq \overline{L}$ donc L n'est pas clos.

Un automate co-accessible est aussi dit non-bloquant, ce qui s'exprime par l'égalité sur les langages

$$L(G) = \overline{L}_m(G)$$

1.3 Produits d'automates

On utilisera dans la suite deux compositions (ou produits synchrones) d'automates distinctes : la composition parallèle, notée ||, et la composition par produit, notée ×. Dans les deux compositions, l'état initial est la composition de deux états initiaux. Un état est marqué si il est la composition de deux états marqués.

La composition par produit (ou produit) est la synchronisation des automates, l'intersection des langages. Une séquence n'est possible dans le produit que si elle est possible dans les deux facteurs. Ainsi si il n'y aucun événement en commun, le produit n'a aucune transition. Cette composition amène une synchronisation contraignante entre les deux automates. Par exemple un automate de contrôle peut empêcher l'occurrence d'un événement.

Dans la composition parallèle, on ne synchronise que les évènements présents dans les deux automates, commun aux deux alphabets. Si tous les évènements sont communs, les deux compositions sont équivalentes. Au contraire, si il n'y aucun événement en commun, la composition parallèle se ramène au produit libre d'automates ou au produit cartésien de graphes. Cette composition permet de modéliser un système par des sous-systèmes qui ne sont synchronisés que par quelques évènements.

Exemple 1 : Compositions parallèle et par produit

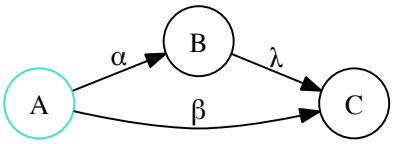

Figure 4. G_1

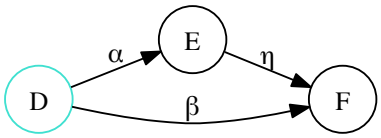

Figure 5. G_2

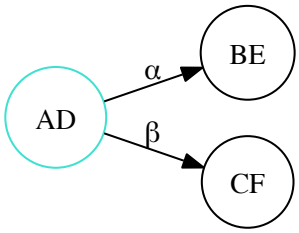

Figure 6. $G_{produit} = G_1 \times G_2$

A partir des états initiaux, α et β sont possibles dans G_1 et G_2, donc dans $G_{produit}$.
On observe bien ici que le produit est une intersection.

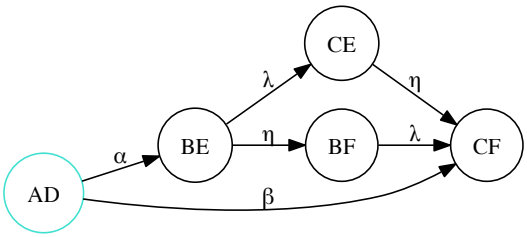

Figure 7. $G_{\text{parallèle}} = G_1 || G_2$

Les évènements présents dans G_1 et G_2 sont α et β. γ et δ ne sont pas synchronisés, il suffit qu'ils soient possibles dans G_1 ou G_2 pour être dans le produit.

Exemple 2 : Produit

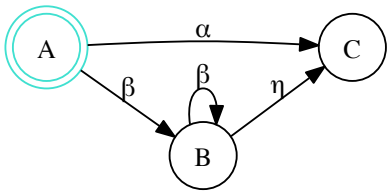

Figure 8. G_1

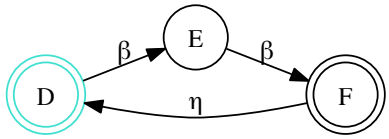

Figure 9. G_2

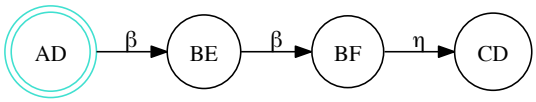

Figure 10. $G = G_1 \times G_2$

Exemple 3 : Composition parallèle **(**extrait de [W04])

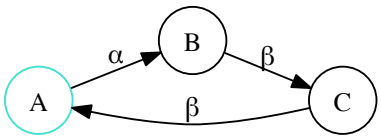

Figure 11. G_1

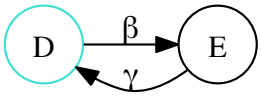

Figure 12. G_2

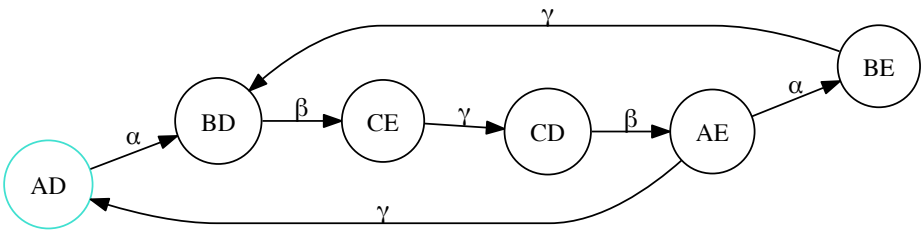

Figure 13. G = $G_1||G_2$

Ici seul l'événement β est synchronisé. α (resp. γ) peut se produire à chaque fois que G_1 (resp. G_2) est en A (resp. en E), indépendamment de G_2 (resp. de G_1).

2 La synthèse de superviseur

La SCT permet de générer un superviseur, étant donné un système à contrôler (sous forme d'automate) et une spécification (sous forme de langage). On appelle ici *procédé* le système à contrôler, aussi appelé *plant*. On le modélise par un automate. Ce modèle est dit *physique*, indépendant des objectifs de commande. On associe à l'automate du procédé une sémantique particulière : il génère spontanément les évènements qui le font évoluer. L'automate est appelé *générateur*.

Afin de respecter la spécification, le superviseur peut (seulement) inhiber l'occurrence des évènements contrôlables dans le générateur. Les événements contrôlables sont généralement les commandes envoyées aux actionneurs ; les évènements incontrôlables proviennent du procédé ou de l'environnement, et sont liés aux capteurs.

Vue générale de la méthode

① Identification du modèle du procédé, sous la forme d'un automate (générateur) G, avec son langage marqué $L_m(G)$.

② Identification des propriétés à respecter, sous la forme d'un langage E.

③ Identification du comportement du système supervisé sous la forme d'un langage K_c, physiquement possible, respectant les propriétés de E, et tel que l'occurrence d'évènements incontrôlables ne puisse pas conduire à la violation des propriétés (K_c est dit contrôlable). Le langage K_c est également choisi de façon à inhiber le moins possible le comportement.

④ Extraction de la carte de contrôle à partir de K_c, c.-à-d. d'une table donnant pour chaque séquence l'événement suivant autorisé (ou l'ensemble des évènements autorisés).

2.1 Identification du modèle du procédé

Ici, le modèle du procédé est supposé connu. Dans la suite, le procédé est modélisé par le générateur $G = (Q, \Sigma, \delta, q_0, Q_m)$, il est le modèle *physique*, indépendant des objectifs de commande.

L'alphabet du générateur est partitionné en Σ_c, l'ensemble des évènements contrôlables et Σ_u, l'ensemble des évènements incontrôlables.

$$\Sigma = \Sigma_c \cup \Sigma_u \quad \text{et} \quad \Sigma_c \cap \Sigma_u = \varnothing$$

2.2 Identification des propriétés

On exprime les propriétés à respecter par un langage E. E est un langage marqué, qui ne mentionne que les séquences complètes, terminées. Néanmoins tous les stades intermédiaires (qui forment $\overline{E}$) sont évidemment autorisés.

2.3 Identification du comportement supervisé

Le comportement supervisé est un comportement physiquement possible pour le système, et qui respecte les propriétés de E. On le modélise donc par

$$K = E \cap L_m(G)$$

Comme E et $L_m(G)$, K ne comporte que les séquences terminées, marquées. Les étapes intermédiaires étant également possibles et autorisées, on utilisera souvent $\overline{K}$ dans le contrôle.

Pour un procédé donné, certaines propriétés ne pourront pas être respectées, quel que soit le superviseur, car on ne peut pas inhiber les évènements incontrôlables. Par exemple si un événement incontrôlable est indésirable, on ne pourra pas l'inhiber. La faisabilité des propriétés par rapport aux actions et aux aléas du système est reflétée par la propriété de contrôlabilité.

Définition : Contrôlabilité[1]

Un langage K est contrôlable ssi

$$\overline{K}\,\Sigma_u \cap L(G) \subseteq \overline{K}$$

(où un élément de $\overline{K}\,\Sigma_u$ est la concaténation d'un élément de $\overline{K}$ et d'un évènement incontrôlable). Toute séquence de $\overline{K}$, c.-à-d. vérifiant les propriétés, suivie d'un évènement incontrôlable, doit continuer à vérifier les propriétés. On ne prend en compte que les séquences ayant du sens pour G, physiquement possibles, faisant partie de L(G). On constate que la contrôlabilité de K ne fait pas intervenir K mais seulement $\overline{K}$.

Si K n'est pas contrôlable, on prend un sous-langage contrôlable. Ainsi les propriétés seront toujours vérifiées. On choisit le plus grand sous-ensemble contrôlable de K, noté K_c, afin d'ajouter un minimum de restrictions en plus de celles spécifiées par les propriétés. (On a $K_c = K$ si K est contrôlable). On parle alors de superviseur optimal, le moins restrictif, le plus permissif.

Il est important de noter que la recherche d'un sous-langage contrôlable est le point clé de la méthode. Les restrictions supplémentaires correspondent à l'anticipation par le superviseur de son impossibilité future (c.-à-d. dans un état suivant) à inhiber certains évènements (cf exemple en 3.3).

[1] On définit ici la contrôlabilité d'un langage. On notera que le caractère contrôlable d'un événement est un choix de modélisation alors que la contrôlabilité d'un langage est une propriété formalisée.

2.4 Extraction de la carte de contrôle

Le superviseur a pour seule action d'autoriser ou non un événement, sous certaines conditions, et non de les déclencher ou pas. L'ensemble des évènements autorisés à l'instant t est nommé pattern de contrôle et est noté γ. Les évènements incontrôlables ont la particularité de ne pas pouvoir être interdits, c.-à-d. que $\gamma \supseteq \Sigma_u$. (On note Γ l'ensemble des patterns possibles avec Σ)
Le superviseur est défini par une carte de contrôle V *(map* en anglais*)*.

$$V : L(G) \rightarrow \Gamma$$

On définit quels sont les évènements autorisés pour des séquences de G.

Remarque :
Si on définissait un pattern pour chaque état de G, la supervision reviendrait à supprimer quelques transitions dans G. Comme on définit un pattern pour chaque séquence, le superviseur pourra appliquer des patterns différents à un même état en fonction du passé du système, du chemin emprunté (ce qui revient à dire que la représentation des états physiques du système n'est pas suffisante pour la commande)

Le système supervisé est le couplage du générateur et de la carte de contrôle, un événement ne pouvant être généré que si il est autorisé.

Construction de la carte de contrôle
Soit K_c un langage contrôlable vis-à-vis de G.
On définit la carte de contrôle V par

$$\forall s \in \overline{K_c} \quad V(s) = \Sigma_u \cup \{ \sigma \in \Sigma_c \mid s\sigma \in \overline{K_c} \}$$

Les évènements incontrôlables font d'office partie du pattern de chaque séquence, c.-à-d. que les évènements incontrôlables sont toujours autorisés. A partir de s, ne sont autorisés que les évènements contrôlables qui ne violent pas les propriétés.

2.5 Procédure de la synthèse de superviseur (orientée langage)

La méthode vue page 7 peut être formulée grâce aux notions théoriques vues précédemment.
① Modéliser le système à contrôler par un automate G
② Modéliser les propriétés à respecter sous forme d'un langage E (par intersection des langages des différentes propriétés)
③ Se ramener à l'hypothèse $K \subseteq L_m(G)$ avec $K = E \cap L_m(G)$
Puis trouver K_c le plus grand sous-langage contrôlable de K ($K_c \subseteq K$ tel que $\overline{K_c}\,\Sigma_u \cap L(G) \subseteq \overline{K_c}$)
④ Calculer la carte de contrôle avec $\forall s \in \overline{K_c} \quad V(s) = \Sigma_u \cup \{ \sigma \in \Sigma_c \mid s\sigma \in \overline{K_c} \}$

Le superviseur est représenté par la carte de contrôle. Le superviseur est optimal (le moins restrictif). De plus le superviseur est dit marquant car le marquage de E peut restreindre le marquage du système supervisé par rapport à celui du procédé de départ. Le superviseur est dit non-bloquant, c.-à-d. que le système supervisé est co-accessible, il peut atteindre un état marqué depuis chacun de ses états.
La démarche est appliquée sur un exemple en 3.1.

2.6 Représentation du superviseur sur un automate

Le superviseur est totalement spécifié par sa carte de contrôle. Cependant pour des raisons de compréhension et d'implémentation, on le représente souvent par un automate S.

$$S= (X, \Sigma, \xi, x_0, X_m)$$

S et G ont en commun l'alphabet des évènements de Σ. L'ensemble des états X est différent de Q, l'ensemble des états de G. La carte de contrôle définit toutes les séquences pertinentes, ces séquences donnent les états X de S. Typiquement le superviseur est capable de distinguer plusieurs sous-états au sein d'un état de G selon le chemin par lequel on y est arrivé, et d'appliquer des patterns différents. X contient ces sous-états. (cf exemple 3.1.5)

Plusieurs automates sont possibles pour reconnaître un langage et donc représenter un superviseur. On choisit pour le superviseur un automate trim, on dit alors du superviseur qu'il est strict.

L'état du système supervisé est représenté à la fois par un état de G et par un état de S. A chaque instant, les événements pouvant être générés sont présents à la fois dans l'état de G et dans l'état de S. Le système supervisé est donc obtenu par le produit de G et S, considéré comme un générateur.

L'action d'un superviseur peut être définie par sa carte de contrôle mais le superviseur est généralement implémenté sous la forme d'un automate. On a évidemment intérêt à réduire la taille de l'automate. On utilise un algorithme de réduction qui cherche à minimiser le nombre d'états de l'automate en conservant la même carte de contrôle (cf exemple 3.2.5).

Cette approche dispose d'un outil : TCT [W04], développé à l'université de Toronto, gratuit et disponible à [W13]. TCT permet de saisir et manipuler des automates. Il implémente les algorithmes de la SCT comme la recherche du plus grand sous-langage contrôlable. Le superviseur est visualisable par la carte de contrôle et par l'automate. TCT donne la carte de contrôle sous la forme de la liste des évènements interdits en fonction de l'état de l'automate du superviseur. Seuls les évènements interdits et possibles physiquement sont mentionnés.

On ré-écrit la méthode en utilisant les fonctions de TCT. On ajoute l'étape de réduction.
① Modéliser le système à contrôler par un automate G (éventuellement par une composition parallèle (**sync)** entre plusieurs automates de sous-systèmes)
② Modéliser les propriétés sous la forme d'un automate P (éventuellement par un produit (**meet)** de plusieurs automates de propriétés)
③Trouver le plus grand sous-automate S contrôlable de P (qui soit aussi un trim) S=**supcon**(G,P)
④ Calculer la carte de contrôle V=**condat**(G,S) pour information ou réduction
⑤ Eventuellement, réduire le superviseur S_{red}=**supreduce**(G,$S_{,red}$,V)
On peut avoir un modèle du système supervisé par **meet**(G,S_{red}).
Il reste à implémenter S ou S_{red}.
La démarche est appliquée sur un exemple en 3.2.

3 Applications et exemples

3.1 Application (orientée langage)

Le procédé à superviser est constitué de deux machines, qui peuvent défaillir. Il faut réparer les machines, en respectant la propriété : « les machines doivent être réparées dans l'ordre dans lequel elles ont défailli »

① Modèle du procédé

La machine 1 est modélisée par l'automate M_1 (Figure 14)

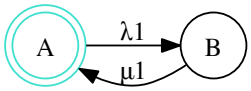

Figure 14. M_1

avec λ_1, défaillance (événement incontrôlable) et μ_1, réparation (événement contrôlable). La machine 2 a un comportement similaire avec les évènements λ_2 et μ_2. Ses états sont notés a et b.

Le procédé complet est modélisé par le produit synchrone $G = M_1||M_2$ (Figure 15). Comme aucun événement n'est commun à M_1 et M_2, le produit synchrone est ici identique à un produit cartésien.

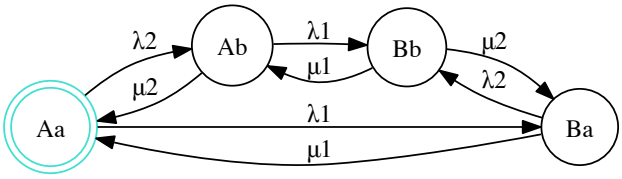

Figure 15. G

② Modèle de la propriété

La propriété est modélisée par l'automate P (Figure 16)

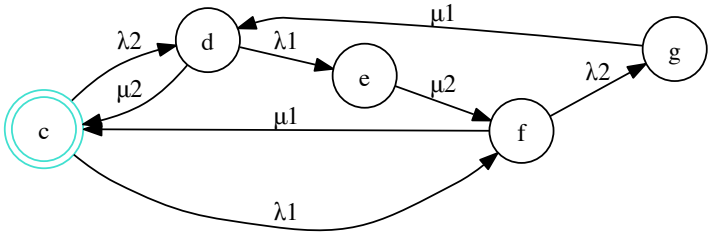

Figure 16. P

On n'a ici qu'une seule propriété. On l'exprime par le langage marqué de l'automate P

$E = L_m(P) = (\lambda_1(\lambda_2\mu_1\lambda_1\mu_2)^*(\mu_1 + \lambda_2\mu_1\mu_2) + \lambda_2(\lambda_1\mu_2\lambda_2\mu_2)^*(\mu_2 + \lambda_1\mu_2\mu_1))^*$

Remarque : Le procédé et la propriété sont symétriques, les 2 machines sont interchangeables. On peut interchanger les couples (λ_1,μ_1) et (λ_2,μ_2).

③ Le plus grand sous-langage contrôlable

On constate que toutes les séquences de E sont possibles dans G, on a donc $E \subset L_m(G)$, et puisque $K=E \cap L_m(G)$, on a $K = E$.

Tentons de montrer que K est contrôlable. Comme K a un nombre infini de mots, on se propose de vérifier dans le Tableau 1 la contrôlabilité de trois mots jugés représentatifs du comportement : $\lambda_1\mu_1$ et $\lambda_1\lambda_2\mu_1\mu_2$ (ce qui ne démontre rien mais permet de comprendre le principe).
Examinons tous leurs préfixes, et ajoutons chaque événement incontrôlable. Pour vérifier la contrôlabilité le mot résultant doit, soit ne pas appartenir à L(G), soit appartenir à $\overline{K}$.

Définition de la contrôlabilité :	$K\Sigma_u$	$\cap L(G)$	$\subseteq \overline{K}$

Mot de K	Concaténation sσ		$s\sigma \notin L(G)$	$s\sigma \in \overline{K}$
	$s \in \overline{K}$ Préfixes du mot de K	$\sigma \in \Sigma_u$ Évènement incontrôlable	sσ impossible physiquement	sσ satisfait les propriétés
ε	ε	λ_1		✓
		λ_2		✓
$\lambda_1\mu_1$	λ_1	λ_1	✓	
		λ_2		✓
	$\lambda_1\mu_1$	λ_1	✓	
		λ_2		✓
$\lambda_1\lambda_2\mu_1\mu_2$	λ_1	λ_1	✓	
		λ_2		✓
	$\lambda_1\lambda_2$	λ_1	✓	
		λ_2	✓	
	$\lambda_1\lambda_2\mu_1$	λ_1	✓	
		λ_2	✓	
	$\lambda_1\lambda_2\mu_1\mu_2$	λ_1	✓	
		λ_2	✓	

Tableau 1. Controlabilité de K

Rappellons ε est la séquence vide, au départ de l'état initial, ce qui correspond donc à l'état initial.
D'après ce tableau, pour les 3 mots choisis, soit l'ajout d'un évènement incontrôlable mène à une séquence impossible, soit la séquence respecte les propriétés. Ces mots sont donc contrôlables. Considérons que tous les mots de K sont contrôlables, alors K est contrôlable (et sera dorénavant noté K_c).

Tous les mots sont contrôlables (et le système étant symétrique, on sait aussi que $\lambda_2\mu_2$ et $\lambda_2\lambda_1\mu_2\mu_1$ sont contrôlables). K est donc considéré comme contrôlable (on le notera dorénavant K_c).

④ Carte de contrôle

Comme dans ③, on illustre le calcul de carte de contrôle seulement sur quelques séquences dans le Tableau 2. Examinons pour chaque séquence, chaque événement contrôlable. (Les évènements incontrôlables font partie de tous les patterns de contrôle.)

$\{\sigma \in \Sigma_c$	$\mid s\sigma \in \overline{K}_c\}$	$\cup$	Σ_u	=	V(s)	
Concaténation de sσ		$s\sigma \in \overline{K}_c$	Σ_u	V(s)	$s\sigma \notin$
$s \in \overline{K}_c$ Séquence de départ	$\sigma \in \Sigma_c$ Evènement contrôlable	sσ satisfait les propriétés (donc $\sigma \in$ V(s))	Ensemble des évènements incontrôlables	Pattern, ensemble des évènements autorisés	L(G)
ε	μ_1		$\{\lambda_1;\lambda_2\}$	$\{\lambda_1;\lambda_2\}$	✓
	μ_2				✓
λ_1	μ_1	✓	$\{\lambda_1;\lambda_2\}$	$\{\lambda_1;\lambda_2;\mu_1\}$	
	μ_2				✓
$\lambda_1\mu_1$	μ_1		$\{\lambda_1;\lambda_2\}$	$\{\lambda_1;\lambda_2\}$	✓
	μ_2				✓
$\lambda_1\lambda_2$	μ_1	✓	$\{\lambda_1;\lambda_2\}$	$\{\lambda_1;\lambda_2;\mu_1\}$	
	μ_2			**μ_2 interdit**	
$\lambda_1\lambda_2\mu_1$	μ_1		$\{\lambda_1;\lambda_2\}$	$\{\lambda_1;\lambda_2;\mu_2\}$	✓
	μ_2	✓			
$\lambda_1\lambda_2\mu_1\mu_2$	μ_1		$\{\lambda_1;\lambda_2\}$	$\{\lambda_1;\lambda_2\}$	✓
	μ_2				✓

Tableau 2. Calcul de la carte de contrôle

On remarque que la plupart des évènements absents des patterns, comme μ_2 dans le deuxième pattern V(λ_1), sont en fait impossibles physiquement, c.-à-d. qu'ils n'appartiennent pas à L(G). Le superviseur ne les autorise pas mais ils sont de toute façon impossibles pour le procédé. Dans ce tableau le superviseur n'interdit qu'un seul événement vis-à-vis de L(G) : pour $s=\lambda_1\lambda_2$ et $\sigma=\mu_2$, on a $\lambda_1\lambda_2\mu_2 \notin \overline{K}_c$ mais $\lambda_1\lambda_2\mu_2 \in$ L(G), c.-à-d. qu'une fois que la séquence $\lambda_1\lambda_2$ s'est produite, μ_2 est interdit ce qui correspond bien à la propriété.

On a donné la carte de contrôle sous la forme d'un ensemble des évènements autorisés. D'autres représentations sont possibles. Le Tableau 3 reprend le Tableau 2 en notant 1 pour les évènements autorisés et 0 pour les évènements interdits. D'autre part on ajoute l'état de G correspondant à la séquence s.

$s \in \overline{K}$	Etat correspondant à s dans G	Pattern V(s)			
		λ_1	λ_2	μ_1	μ_2
ε	Aa	1	1	0	0
λ_1	Ba	1	1	1	0
$\lambda_1\mu_1$	Aa	1	1	0	0
$\lambda_1\lambda_2$	Bb	1	1	1	**0**
$\lambda_1\lambda_2\mu_1$	Ab	1	1	0	1
$\lambda_1\lambda_2\mu_1\mu_2$	Aa	1	1	0	0

Tableau 3. Carte de contrôle et correspondance entre séquences et états de G (1° partie)

Par symétrie du Tableau 3 on obtient le Tableau 4.

$s \in \overline{K}$	Etat correspondant à s dans G	Pattern V(s)			
		λ_2	λ_1	μ_2	μ_1
ε	Aa	1	1	0	0
λ_2	Ab	1	1	1	0
$\lambda_2\mu_2$	Aa	1	1	0	0
$\lambda_2\lambda_1$	Bb	1	1	1	**0**
$\lambda_2\lambda_1\mu_2$	Ba	1	1	0	1
$\lambda_2\lambda_1\mu_2\mu_1$	Aa	1	1	0	0

Tableau 4. Carte de contrôle et correspondance entre séquences et états de G (2° partie)

A chaque état de G, on a le même pattern, sauf pour Bb. En effet, vis-à-vis du procédé les séquences $\lambda_1\lambda_2$ et $\lambda_2\lambda_1$ sont équivalentes, mais pour mettre en œuvre la propriété on a besoin de les différencier, de scinder l'état Bb en 2 pour construire l'automate représentant le superviseur. Comme $L_m(P) = E = K = K_c$, le superviseur est représenté par l'automate P, l'état Bb étant scindé en e et g.

L'ensemble des patterns forme la carte de contrôle. Nous avons synthétisé un superviseur optimal (le moins restrictif), non-bloquant.

3.2 Application avec outil TCT (orientée automate)

Dans TCT, tous les éléments (états et évènements) sont numérotés. Les évènements contrôlables sont impairs, les évènements incontrôlables sont pairs. L'alphabet, c.-à-d. la numérotation des événements, est commun à tous les automates. En revanche les états sont numérotés aléatoirement (à part l'état initial noté 0). Les correspondances entre les états des différents automates doivent être faites à partir des séquences d'évènements y menant, et non des numéros d'états.

On traite ici le même exemple que précédemment avec TCT. La correspondance des évènements est donnée en Tableau 5.

Nom usuel	λ_1	μ_1	λ_2	μ_2
Nom dans TCT	0	1	2	3

Tableau 5. Correspondance des noms d'évènement

① Modèle du procédé

Les machines sont représentées par les automates les Figure 17 et Figure 18. M_1 correspond à la Figure 14.

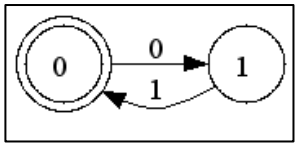

Figure 17. Modèle de la machine 1 avec TCT : Automate M1

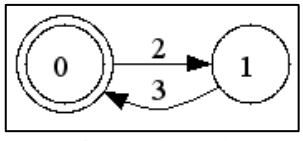

Figure 18. M2

Le procédé complet est calculé par **G = sync (M1, M2)** (Figure 19). G correspond à la Figure 15.

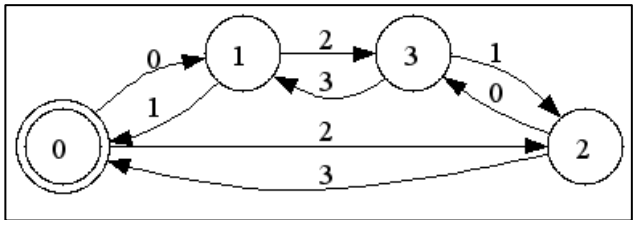

Figure 19. G

② Modèle de la propriété

La propriété est modélisée par l'automate P (Figure 20).

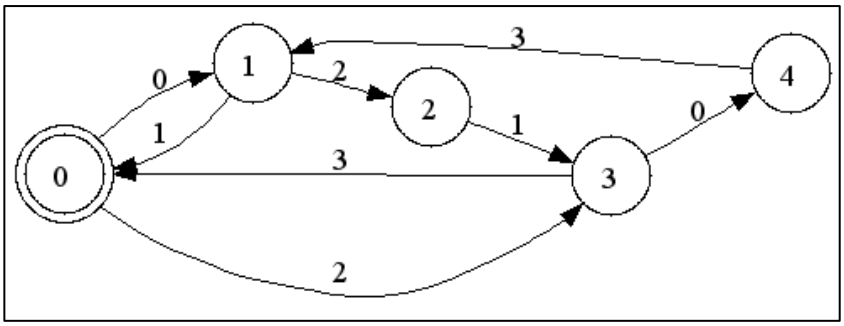

Figure 20. P

③ Le plus grand sous-langage contrôlable

On cherche le plus grand sous-langage contrôlable de P par **S = supcon(G,P)** (Figure 21)

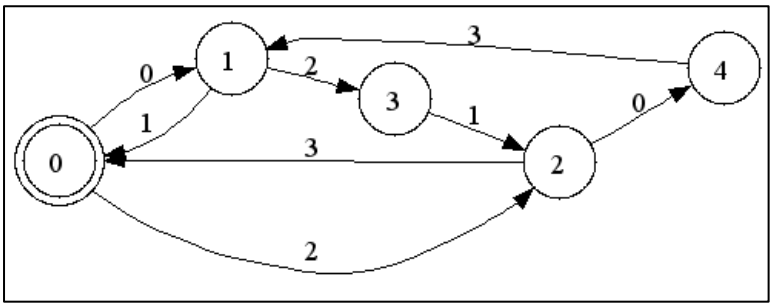

Figure 21. S

On remarque que **S** = **P**, on le vérifie avec **isomorph(P,S)**. P est donc contrôlable, comme notre précédent raisonnement semblait l'indiquer.

④ Calcul de la carte de contrôle

On calcule la carte de contrôle par **V = condat(G,S)** (Figure 22)

```
V is CONTROLLABLE
control data:

      3:   3                                    4:   1
```

Figure 22. V

Dans l'état 4 de S, l'événement 1 est interdit (et dans l'état 3, l'événement 3 est interdit). Le μ_1 est interdit après la séquence $\lambda_2\lambda_1$, et μ_2 est interdit après $\lambda_1\lambda_2$, en se reportant au paragraphe précédent. On constate donc que **condat** ne donne pas tous les évènements absents du pattern, mais seulement ceux qui seraient possibles physiquement, et qu'il est utile d'interdire.

Le modèle du système supervisé est obtenu par **meet(G,S)** (Figure 23)

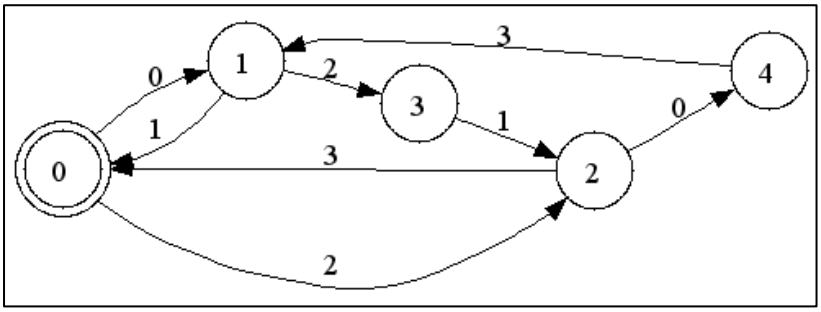

Figure 23. Automate du système supervisé

On remarque que l'automate du système complet est identique à S qui représente le superviseur.

⑤ Réduction

S est l'automate qui devra être implémenté pour obtenir la supervision ; on souhaite évidemment réduire sa taille si possible. Ici il est assez gros, il contient plus d'états que G.
Comme le système final est le produit du procédé et du superviseur le superviseur n'a besoin de contenir aucune information (et notamment aucune limitation) du procédé. Par exemple S interdit la séquence $\lambda_1\lambda_1$ (**00**) alors qu'elle est de toute façon interdite par le procédé. Ainsi on peut réduire le nombre d'états de l'automate représentant le superviseur, tout en gardant la même carte de contrôle, donc le même effet sur le procédé.

Il peut être difficile de concevoir un superviseur en s'affranchissant de la connaissance qu'on a du procédé.
TCT a un algorithme de réduction. On réduit le superviseur par **RED = supreduce(G,S,V)** (Figure 24). Cette procédure donne une borne inférieure du nombre d'états, ici 3. Comme RED a 3 états il est minimal.

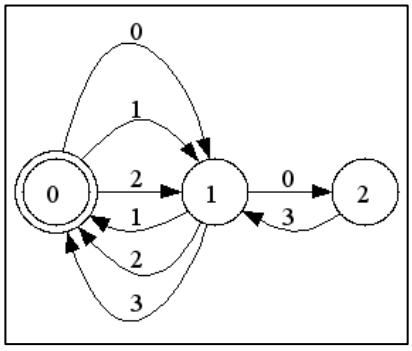

Figure 24. RED

Contrairement aux autres, cet automate n'est pas lisible facilement. La réduction aboutit à ne représenter que les états utiles au contrôle sans tenir compte de l'état du procédé, ce qui est assez contre-intuitif.
L'état 0 représente 'aucune machine défaillante' ou 'M1 puis M2 ont défailli'. L'état 1 correspond à '1 machine défaillante'. L'état 2 représente 'M2 puis M1 ont défailli'. On remarque par exemple que si une seule machine est défaillante, le superviseur n'a aucun besoin de savoir de laquelle il s'agit. Il n'interdit pas la réparation de la machine en bon état puisque le procédé ne le permet de toute façon pas. De même la séquence **00** est ici possible.

```
RED is CONTROLLABLE
control data:

      0:   3                                    2:   1
```

Figure 25. Carte de contrôle associée à RED

La carte de contrôle associée (Figure 25) est la même que précédemment, au nommage des états près.
Les états regroupent ici les séquences suivant les patterns qui leur sont associés. Les séquences impossibles sont indifféremment réparties dans les états afin de réduire le nombre d'états.

Le système avec le superviseur réduit est obtenu par **meet(G,RED)** (Figure 26). Il est identique au système supervisé par S (Figure 23).

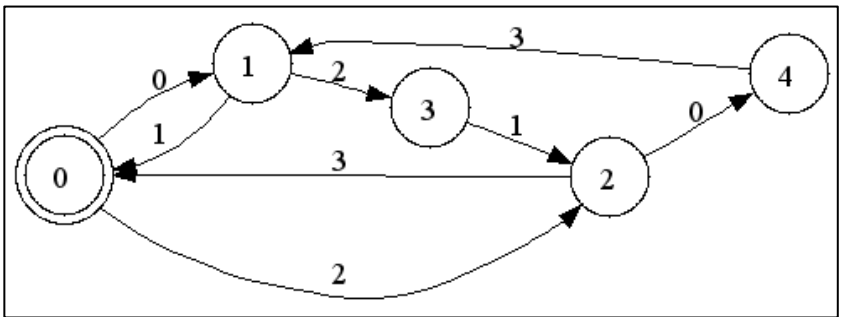

Figure 26. Automate du système supervisé par RED

3.3 Exemple : le plus grand sous-langage contrôlable

Soit un système G (Figure 27).

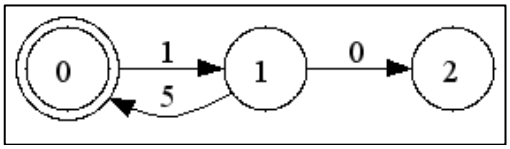

Figure 27. G

Tous les évènements sont contrôlables, sauf **0** qui est incontrôlable et néfaste.
On souhaite éviter que **0** arrive, ce qui est formalisé par l'automate P (Figure 28).

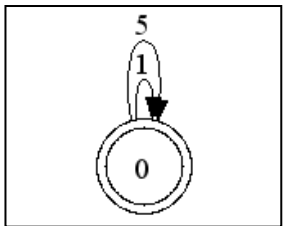

Figure 28. P

Calculons la carte de contrôle de P pour G **condat(G,P)** (Figure 29).

```
P is NOT CONTROLLABLE
control data:

        0:   0
```

Figure 29. Carte de controle de P vis-à-vis de G

L'interdiction d'un événement incontrôlable n'est évidemment pas possible. Satisfaire cette propriété n'est donc pas possible.

P n'étant pas contrôlable, on cherche le plus grand sous-langage contrôlable **supcon(G,P) = PC** (Figure 30) et sa carte de contrôle **condat(G,PC)** (Figure 31).

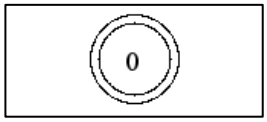

Figure 30. PC

```
PC is CONTROLLABLE
control data:

        0:   1
```

Figure 31. Carte de contrôle de PC vis-à-vis de G

Le seul moyen d'empêcher **0** est d'interdire l'événement contrôlable **1**, et donc d'interdire au procédé d'évoluer. Cette solution n'est pas satisfaisante, elle révèle un problème de conception de G. On modifie le procédé, et donc son modèle pour obtenir G1 (Figure 32). On reformule la propriété P en P1 (Figure 33)

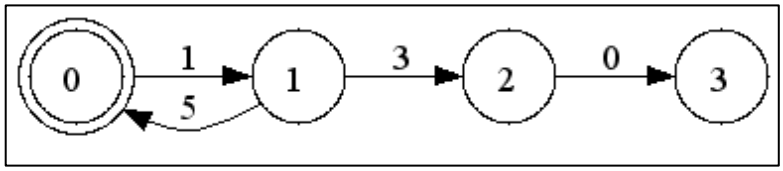

Figure 32. G1

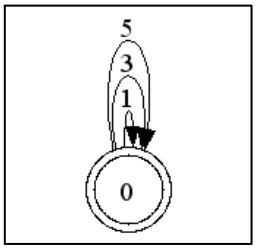

Figure 33. P1

G1 n'est toujours pas contrôlable vis-à-vis de P1. On cherche le plus grand sous-langage contrôlable **supcon(G1,P1) = PC_1** (Figure 34) et sa carte de contrôle **condat(G1,PC_1)** (Figure 35).

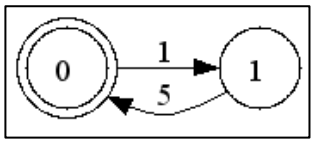

Figure 34. PC_1

```
PC_1 is CONTROLLABLE

control data:

      1:   3
```

Figure 35. Carte de contrôle de PC_1 vis-à-vis de G1

Le superviseur anticipe l'occurrence de **0**, il interdit **3**, qui, lui, est contrôlable, et donc **PC_1** est contrôlable. **supcon** introduit une restriction non-spécifiée pour pouvoir remplir la propriété.

A noter : Le résultat de **supcon** est toujours un trim.

4 Implémentation du superviseur

Après avoir synthétisé un superviseur sous la forme d'un automate, on veut utiliser ce superviseur pour commander un procédé réel, c.-à-d. obtenir un contrôleur.

La théorie postule des hypothèses asynchrones fortes (comme le caractère instantané des évènements), la sémantique des automates (« un événement non mentionné n'advient pas ») et l'architecture de la Figure 36.

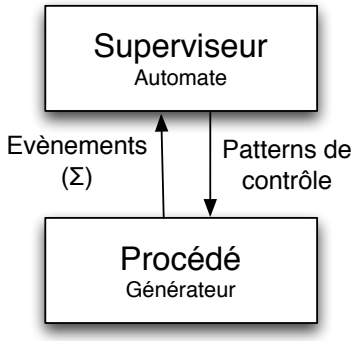

Figure 36. Architecture théorique de Wonham

Le déclenchement des évènements contrôlables et incontrôlables est réalisé par le modèle du procédé (qui est donc un générateur).
La mise en œuvre du superviseur peut se faire en substituant simplement au modèle du procédé le procédé réel. Cependant, en pratique, les procédés ne génèrent que les évènements incontrôlables, c.-à-d. les observations venant des capteurs. C'est pourquoi les systèmes commandés présentent usuellement l'architecture de la Figure 37 dans laquelle le contrôleur génère les évènements contrôlables et le procédé les évènements incontrôlables.

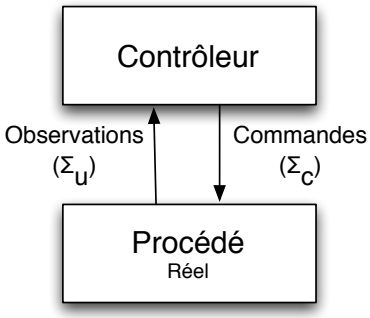

Figure 37. Architecture de contrôle

4.1 Difficultés d'implémentation

Pour obtenir un contrôleur on doit donc adapter l'architecture théorique pour la faire rentrer dans l'architecture de contrôle utilisée en pratique, tout en gérant les hypothèses de départ, très théoriques. L'implémentation du contrôleur pose donc un certain nombre de problèmes [GP04].

a) La modélisation du procédé et des spécifications est subjective. La justesse des modèles est déterminante pour avoir un superviseur performant.
b) Etant donné le niveau d'abstraction des modèles, l'interface entre le monde physique et les modèles haut niveau requiert un gros effort.

c) Les hypothèses temporelles du cadre théorique ne peuvent pas être satisfaites par l'implémentation. Par exemple, le système de commande s'exécutant pas à pas, il peut advenir 2 évènements incontrôlables entre 2 tops d'horloge, ce qui viole l'hypothèse d'asynchronisme (cf 1.1). Dans [QC02], cela oblige à faire l'hypothèse d'insensibilité aux entrelacements (*interleave insensitivity*) sur le couple superviseur/procédé.
d) La modélisation de systèmes importants se fait généralement par le produit des automates des sous-systèmes, ce qui entraine un accroissement exponentiel du nombre d'états. Même si les algorithmes de la SCT sont polynomiaux, le cout de calcul devient très important. Des extensions théoriques comme la modularité ont été proposées pour réduire ce problème. Il s'agit d'avoir plusieurs superviseurs, par exemple un par propriété à respecter.
e) Tel qu'exposé dans la théorie le superviseur est passif, il n'a pas l'initiative de déclencher des actions. Or les systèmes ont généralement besoin d'une commande, et pas seulement d'une supervision. Pour l'implémentation on doit transformer le superviseur en un contrôleur, qui peut déclencher des actions, et non les inhiber.
f) Il est généralement souhaitable que le contrôleur soit déterministe. Si plusieurs commandes sont physiquement possibles et autorisés, le contrôleur doit faire un choix. Des mécanismes de priorité ou d'optimisation sont généralement mis en place pour résoudre ce problème [ACMR03].

On expose dans la suite quelques solutions pour résoudre le problème de l'initiative.

4.2 Exemples d'implémentation

On s'intéressa plus particulièrement à la réponse au problème de l'initiative (e), c.-à-d. à la mise en place d'un superviseur actif.

4.2.1 Interprétation Entrée/Sortie

Dans [BHGW93], les auteurs proposent d'adopter une vision Entrée/Sortie du superviseur.

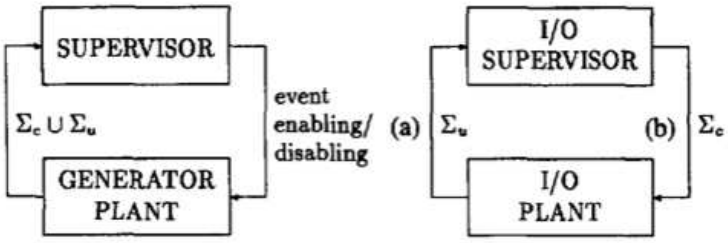

Figure 38. Superviseur de Wonham (à gauche) et Superviseur I/O (à droite) (extrait de [BHGW93])

On rappelle que Σ_c désigne l'ensemble des évènements contrôlables et Σ_u l'ensemble des évènements incontrôlables.

L'idée est de symétriser la boucle fermée telle qu'elle est définie par Wonham (à gauche dans la figure). Au lieu que le procédé génère tous les évènements, il ne génère plus que les évènements incontrôlables, et le superviseur I/O génère les évènements contrôlables, et devient donc ce que nous appelons ici un contrôleur. Les hypothèses sur le procédé correspondent donc plus à la réalité d'un système physique qui génère des observations et subit des commandes.

Cette modification des hypothèses de base a des conséquences sur la synthèse du superviseur. Dans la théorie de Wonham, l'automate du superviseur est dit contrôlable, c.-à-d. qu'à chaque état, il accepte tous les évènements incontrôlables possibles. De même le procédé I/O doit accepter les commandes, les évènements contrôlables. On a donc une deuxième hypothèse à satisfaire pour faire la synthèse du superviseur. Dans le cas général, il suffit de synthétiser le superviseur de Wonham puis de faire le produit synchrone avec l'automate du procédé pour avoir le superviseur I/O.
Le déclenchement des évènements est fait par l'automate-produit. Si on a plusieurs superviseurs I/O (implémentant plusieurs propriétés) il suffit qu'un seul superviseur I/O déclenche un événement pour qu'il ait lieu. Cet événement viole peut-être les propriétés des autres superviseurs. Cette interprétation ne permet donc pas de faire des superviseurs modulaires.

4.2.2 Séparation du contrôle et de la supervision

Dans [CAD95], les auteurs ont adopté une autre approche. Un « contrôleur logique » est ajouté au procédé pour former un générateur en adéquation avec la théorie. Le contrôleur logique ne peut déclencher que les évènements autorisés. Les auteurs nomment leur approche *contrôle supervisé* (*supervised control concept*).

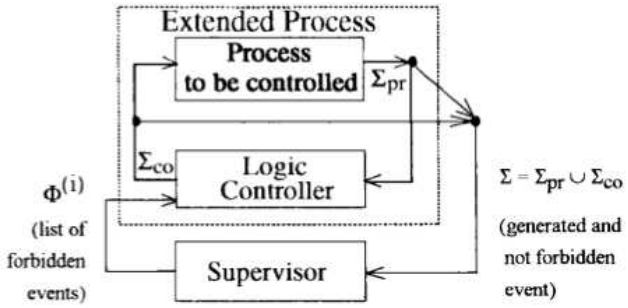

Figure 39. Superviseur séparé du contrôle (extrait de [CAD95])

Σ_{co} désigne l'ensemble des évènements qui sont interdits au moins une fois par la carte de contrôle, et Σ_{pr} les autres évènements. Dans le cas général Σ_{co} est l'ensemble des évènements contrôlables, et Σ_{pr} l'ensemble des évènements incontrôlables.
Le contrôleur contient le modèle du procédé et un mécanisme permettant de le synchroniser avec le superviseur. L'implémentation se faisant en Grafcet, le déclenchement est implicite dans la sémantique du langage. Cette approche sépare la supervision (autorisation) du contrôle (déclenchement), et permet la modularité. Il suffit de prendre l'intersection des patterns de contrôle des différents superviseurs.

4.3 Discussion

Il s'agit dans ce paragraphe de discuter les deux solutions données en 4.2.1 et 4.2.2 pour allier l'architecture de contrôle et le superviseur synthétisé sous forme d'automate, c.-à-d. pour faire cohabiter un superviseur synthétisé sous certaines hypothèses et un procédé qui ne les respecte pas.

Pour répondre au problème de l'initiative, il faut un mécanisme (ici appelé contrôleur) qui déclenche les commandes. Ces commandes doivent être :

- Autorisées pour que le système respecte les propriétés. Il suffit que le contrôleur prenne en compte le pattern de contrôle venant du superviseur

- Possibles physiquement, ce qui est déterminé par un modèle du procédé. On remarque que dans les deux solutions, le modèle du procédé, d'abord utilisé pour la synthèse, est réutilisé pour la commande.

On reformule les deux solutions dans la Figure 40.

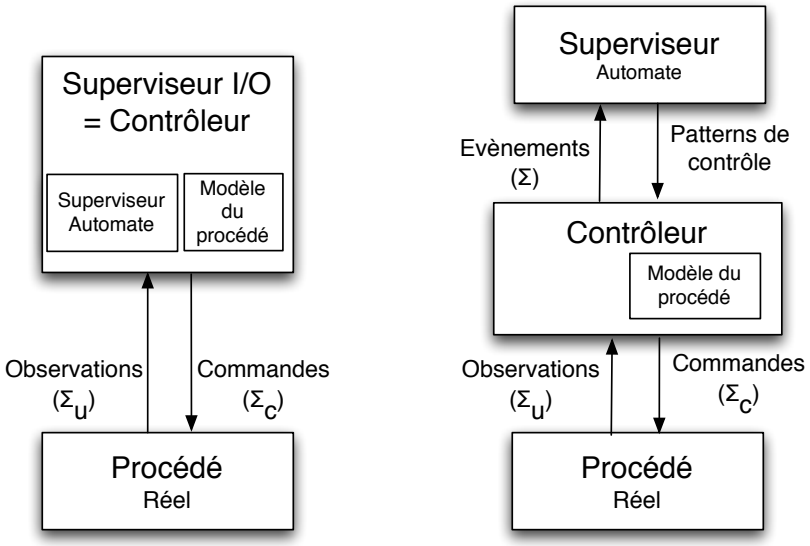

Figure 40. Approche I/O (4.2.1) et approche par séparation (4.2.2)

Bien que les 2 implémentations soient issues de démarches différentes, et n'aient pas les mêmes possibilités (notamment pour la modularité), elles se ramènent à implémenter le modèle du procédé en plus du superviseur. On retrouve cette approche dans [QC02]. En effet un contrôleur doit non seulement faire respecter les propriétés spécifiées, mais aussi ne pas générer des commandes inacceptables physiquement, ce qui risquerait d'endommager le procédé.

Comme la synthèse est faite à un haut niveau d'abstraction et s'appuie sur les machines d'états, un modèle simple et courant, les choix de techniques d'implémentation sont variés : par exemple du C sous Linux [BHGW93], du LADDER [QC02], du Grafcet [CAD95].

Conclusion

La synthèse de superviseur requiert la modélisation du système à contrôler. La modélisation se base sur une liste d'évènements contrôlables et incontrôlables et prend la forme d'un automate associé à la sémantique « un événement non mentionné n'advient pas ». Le superviseur contraint le comportement du système à respecter des propriétés (de commandes ou non-fonctionnelles, représentables par un langage régulier). Le superviseur est donc passif, il impose des limites mais ne gère pas l'évolution du système à l'intérieur de ces limites. La recherche du plus grand sous-langage contrôlable est le cœur de la synthèse. Elle revient à adapter le premier modèle du procédé pour en faire un modèle du procédé adapté à la supervision des propriétés à respecter. On déduit de ce modèle la carte de contrôle qui, à chaque séquence, associe un ensemble d'évènements interdits.
Pour l'implémentation, on traduit la carte de contrôle par un automate. Il existe plusieurs manières d'implémenter un superviseur. On retiendra que la commande nécessite de réutiliser le modèle du procédé afin de ne pas générer de commandes impossibles et dommageables pour le procédé.

La synthèse de superviseur génère automatiquement un superviseur correct par construction. Elle permet donc de reconfigurer un contrôleur rapidement, en économisant du temps de conception de vérification et de validation. Cependant l'explosion combinatoire des états pour les gros systèmes demande beaucoup de ressources pour la synthèse.

Parallèlement aux extensions de la théorie de base, d'autres approches de synthèse de contrôleur ont été développées. Elles suivent la même démarche, partant d'un modèle du procédé et de propriétés et produisant un superviseur (sous la forme d'un BDD[2] par exemple [PI04]) ou directement un contrôleur. Cependant ces approches ne s'appuient plus sur la théorie des langages, et le procédé est différent : dans [MBLL00] le procédé est synchrone (c.-à-d. avec une horloge), dans [VP11] il est non-déterministe.

[2] *Binary Decision Diagram*

Bibliographie

[ACMR03] K. Altisen, A. Clodic, F. Maraninchi and E. Rutten. Using controller synthesis to build property-enforcing layers, *European Symposium on Programming (ESOP),* 2003.

[BHGW93] S. Balemi, G.J. Hoffmann, P. Gyugyi, H. Wong-Toi, et G.F. Franklin. Supervisory control of a rapid thermal multiprocessor, *IEEE Transactions on Automatic Control*, vol.38, no.7, pp.1040-1059, Jul 1993.

[CAD95] F. Charbonnier, H. Alla et R. David. The supervised control of discrete event dynamic systems: a new approach, *Proceedings of the 34th IEEE Conference on Decision and Control*, vol.1, no.7, pp.913-920, 13-15, 1995.

[GP04] D. Gouyon, J-F Petin et A. Gouin. A pragmatic approach for modular control synthesis and implementation, *International Journal of Production Research*, vol. 42, no.14, 2004.

[MBLL00] H. Marchand, P. Bournai, M. Le Borgne et P. Le Guernic. Synthesis of Discrete-Event Controllers based on the Signal Environment, *Discrete Event Dynamical System : Theory and Applications*, vol. 10, no. 4, pages 325–346, 2000.

[O10] C. Oriat. Compilation I.4. membres-liglab.imag.fr/oriat/Compil/Cours/, visité le 12/2/2013.

[PI04] F. Py, F. Ingrand. Dependable execution control for autonomous robots, *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, vol.2, pp. 1136- 1141, 2004.

[QC02] M.H. de Queiroz et J.E.R. Cury. Synthesis and implementation of local modular supervisory control for a manufacturing cell, *Proceeding of Sixth International Workshop on Discrete Event Systems*, pp. 377- 382, 2002.

[RW87] P. J. Ramadge et W. M. Wonham. Supervisory control of a class of discrete event processes, *SIAM Journal of Control and Optimization*, vol.25, 206–230, 1987.

[VP11] G. Verfaillie, C. Pralet. Constraint Programming for Controller Synthesis, *Principle and practice of constraint programming,* LNCS, vol.6876, pp. 100-114, 2011.

[W13] Page personnelle de W. M. Woham
www.control.toronto.edu/people/profs/wonham/wonham.html, visité le 12/2/2013.

[W04] W. M. Wonham. *Supervisory control of discrete-event systems*, Dept. Elect. Comp. Eng., Univ. Toronto, 2004.