Contrôle des SED avec urgence, évitabilité et inéluctabilité

Jean-Luc Béchennec[1], Didier Lime[2], and Olivier (H.) Roux[2]

[1] CNRS, LS2N, Nantes France
jean-luc.bechennec@ls2n.fr
[2] Ecole Centrale de Nantes, LS2N, Nantes France
didier.lime@ec-nantes.fr
olivier-h.roux@ec-nantes.fr

Résumé

– La synthèse de contrôleurs pour les systèmes réactifs peut être faite en calculant les stratégies gagnantes dans les jeux à deux joueurs. Les automates (à jeu) temporisés sont un formalisme approprié pour modéliser les systèmes embarqués temps réel mais ne sont pas faciles à utiliser pour la synthèse de contrôleurs pour deux raisons : i) les modèles temporisés nécessitent la connaissance des temps précis du système (par exemple, si une action doit se produire à l'avenir, la date limite de cette action doit être connue) ii) Dans la pratique, la taille de l'espace d'états rend le calcul du contrôleur souvent impossible pour les systèmes complexes. Cet article présente une extension des automates à jeu non temporisés avec du temps logique. La nouvelle sémantique introduit deux nouveaux types d'actions incontrôlables : des actions *non immédiates* qui peuvent éventuellement être évitées par exemple par une action contrôlable effectuée en urgence, et des actions *inéluctables* qui finiront par se produire si rien n'est fait pour les annuler. Le problème de synthèse de contrôleur est adapté à cette nouvelle sémantique. Cet article se concentre spécifiquement sur les objectifs d'accessibilité et de sûreté et donne des algorithmes pour générer un contrôleur. L'utilité de ce nouveau modèle est illustrée par un exemple de synthèse de pilotes de périphériques.

1 Introduction

La théorie de la supervision a été développée depuis une trentaine d'année à partir des travaux fondateurs de [13, 18, 21] et est devenue un paradigme de base pour le contrôle des systèmes à événements discrets (DES) modélisés sous la forme de machines d'états finies.

Depuis [18], différents formalismes ont été proposés pour modéliser des actions (non)contrôlables pour des problèmes de contrôle pour lesquels une formulation sous la forme de jeux à deux joueurs a fourni des solutions efficaces [19]. Dans ce contexte, le contrôleur est modélisé par un joueur et l'environnement par son adversaire. Déterminer s'il existe un contrôleur revient à déterminer s'il peut gagner, et élaborer une stratégie gagnante revient à synthétiser un contrôleur. Cependant, ces jeux *au tour par tour* [19] où un joueur choisit son action avant que l'autre choisisse la sienne, sont séquentiels et ne permettent pas de modéliser la concurrence. Par conséquent, des jeux concurrents [9, 11, 12] ont été proposés, pour lesquels, à chaque tour, le joueur 1 (le contrôleur) et le joueur 2 (l'environnement) choisissent indépendament et simultanément leur coup pour déterminer le prochain état du jeu.

Outre les actions contrôlables et incontrôlables utilisées dans un contexte non temporisé, les systèmes de contrôle nécessitent souvent sur des capacités comportementales supplémentaires, basées notamment sur deux notions importantes : les délais et l'urgence.

Sans délais, nous ne pouvons pas exprimer le fait que certaines actions (telles qu'une conversion analogique, ou l'émissions de messages sur un bus de communication) prennent du temps,

et que le contrôleur peut effectuer des actions pendant cette période, éventuellement en interrompant l'opération en cours de l'environnement. Dans ce cas, le contrôleur doit avoir recours à une sorte d'urgence.

De plus, sans urgence, nous ne pouvons pas modéliser des comportements inéluctables de l'environnement (tels que l'arrivée d'un produit au bout d'un tapis roulant sur laquelle il est placé), car dans les jeux non temporisés, l'environnement a la possibilité de ne pas jouer pour faire perdre le contrôleur.

Les automates temporisés [2] et les jeux temporisés [10] est un formalisme approprié pour exprimer et modéliser ces propriétés temporisées. Dans un jeu temporisé, les dates à laquelles les deux joueurs (contrôleur et environnement), jouent leurs mouvements sont explicitement prises en compte.

Leur expressivité, les techniques bien connues de synthèse de contrôleurs et les outils [1, 4] associés, permettent la modélisation de systèmes avec des interactions complexes, tout en fournissant une preuve formelle du comportement du système. Cependant, la complexité de calcul des algorithmes impliqués limite la taille des systèmes pouvant être traités en pratique.

De plus, ces formalismes temporisés nécessitent une très bonne connaissance de tous les composants du système, y compris la connaissance précise des données temporelles des actions des deux joueurs. Or ces quantifications temporelles sont rarement connus avec précision et lorsqu'elles le sont (ou tout au moins lorsque l'on en connait des bornes), la complexité des algorithmes de synthèse des contrôleurs temporisés reste un problème.

Par conséquent, il serait très intéressant de dériver un contrôleur sans temps explicite (c'est-à-dire sans quantification temporelle précise) tout en conservant la notion d'urgence et de délai. Les comportements que nous aimerions capturer peuvent être ramenés à deux types d'actions incontrôlables :

— Actions non immédiates (évitables), qui prennent du temps à se terminer ou peuvent ne pas se produire immédiatement, comme écrire dans une mémoire externe, envoyer un message sur un bus effectuant un calcul spécifique sur une unité matériel dédiée, etc. Ces actions viennent généralement avec un mécanisme d'avortement, donc elles sont *evitables* d'un certain point de vue. Dans des modèles à temps explicite, elles sont modelisées par des contraintes temporelles exprimant des limites inférieures non nulles sur les horloges.
— Actions inéluctables, dont on sait qu'elles vont se produire dans un contexte nominal : la fin d'une transmission ou d'une conversion, ou plus généralement, un accusé de réception d'une commande. L'occurence d'ne action inéluctable est garantie si rien n'est fait pour l'abandonner, ce qui diffère de la notion d'équité. Dans le contexte non temporisé, il ne suffit pas de considérer ces actions comme contrôlables. Premièrement, sauf si l'action est explicitement évitable, le contrôleur ne peut empêcher une action inéluctable, même si cela entraîne de perdre le jeu. Deuxièmement, quand il y a un choix entre deux actions contrôlables, le contrôleur choisit mais lorsque c'est entre deux actions inéluctables, c'est l'environnement qui choisit.

Notre contribution. Nous proposons d'étendre le cadre des jeux non temporisé avec les notions d'évitabilité et d'inéluctabilité pour les actions incontrôlables et d'urgence pour les actions contrôlables :

— une *action évitable* ne peut pas se produire immédiatement ainsi le contrôleur peut effectuer une *action urgente* pour l'éviter si nécessaire.
— une action *ineluctable* est garantie si rien n'est fait pour l'arrêter, et le contrôleur peut compter sur son occurence pour gagner le jeu.

Nous revisitons le problème de synthèse du contrôleur pour des jeux d'accessibilité et de

sureté dans ce contexte conduisant à ce que nous appelons des jeux à temps logiques.

Ce document est organisé comme suit :

Nous donnons d'abord dans la section 2, les définitions de base et les notations pour les jeux à temps logique. A partir de ces définitions, nous justifions notre nouveau modèle dans la section 3. Puis, dans la section 4, nous résolvons le problème de synthèse du contrôleur pour les jeux à temps logique. Dans la section 5 et la section 6 nous nous concentrons respectivement sur les jeux d'accessibilité et les jeux de sureté et nous traitons le cas de l'objectif conjoint d'accessibilité sûre dans la section 7. Nous discutons de la complexité de l'algorithme de calcul des états gagnants implémenté dans notre outil Roméo dans la section 8. Enfin, dans la section 9, nous illustrons notre méthode sur une étude de cas basée sur le contrôleur CAN de Microchip.

Cet article est une version étendue de l'article [3] présenté à la conférence ACSD 2019. Nous avons, pour cette version, étendu les résultats pour un objectif conjoint d'accessibilité et de sûreté et réalisé l'implémentation correspondante dans notre outil Roméo illustré sur l'étude de cas.

2 Jeux à temps logique

Dans cette section, nous proposons une variante des automates de jeux non temporisés classiques. avec une nouvelle sémantique de temps logique capturant l'évitabilité et l'inéluctabilité.

Soient C et U les deux joueurs respectivement appelés contrôleur et environnement.

Définition 1 (Structure de jeu)**.** *Une structure de jeu est n-uplet* $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ *tel que*

- Q *est un ensemble d'états*
- $q_0 \in Q$ *est l'état initial*
- A_C *et* A_U *sont deux ensembles disjoints d'actions respectivement pour le contrôleur et l'environnement.*
- $\delta : Q \times (A_C \cup A_U) \times Q$ *est un ensemble d'arêtes entre états. On note* $q \xrightarrow{a} q'$ *pour* $(q, a, q') \in \delta$.

Par souci de simplicité, nous supposons que l'automate fini sous-jacent est déterministe.

En plus de cette définition, nous définissons $A_U^{\star} \subseteq A_U$ et $A_U^{\diamond} \subseteq A_U$ les sous ensembles des actions respectivement *évitables* et *inéluctable*. Notons que ces sous-ensembles sont indépendants, et leur intersection n'est pas nécessairement vide.

Nous notons également $A_U^{\overline{\star}\overline{\diamond}}$, $A_U^{\star\overline{\diamond}}$, $A_U^{\overline{\star}\diamond}$, $A_U^{\star\diamond}$, $A_U^{\overline{\star}}$ et $A_U^{\overline{\diamond}}$, les autres sous-ensembles de A_U basés sur ces deux notions. Par exemple, $A_U^{\overline{\star}\overline{\diamond}}$ est le sous-ensemble des actions de A_U qui ne sont pas *évitables* et pas *inéluctable* et $A_U^{\overline{\diamond}}$ est le sous-ensemble des actions de A_U qui ne sont pas *inéluctable* mais peuvent être *évitable* ou non.

2.1 Notations graphiques

Pour les figures suivantes, nous allons utiliser les notations suivantes illustrées à la figure 1 :

- Les états sont représentés par des cercles et l'état initial est noté q_0.
- Les transitions contrôlables sont représentées par des flèches en trait plein.
- Les transitions incontrôlables sont représentées par des flèches en pointillés.
- Les transitions évitables commencent par un cercle.
- Les transitions inéluctables se terminent par une double flèche.

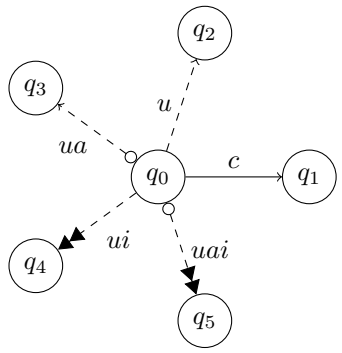


FIGURE 1 – Illustrations des notations graphiques : Ici q_0 est l'état initial, et $c \in A_C, u \in A_U^{\overline{\star}\overline{\diamond}}, ua \in A_U^{\star\overline{\diamond}}, ui \in A_U^{\overline{\star}\diamond}$ et $uai \in A_U^{\star\diamond}$.

2.2 Comportements dans les structures de jeu

Les comportements dans les structures de jeu sont des comportements temporels, mais uniquement à un niveau logique, dans lequel nous distinguons les actions immédiates des autres : nous désignons donc par Δ l'ensemble $\{\mathbf{0}, \bullet\}$, qui représente le temps logique auquel une action est jouée. Il peut être instantané ($\mathbf{0}$) ou inconnu ($\bullet$). Sémantiquement, $\langle a, \mathbf{0}\rangle$ signifie que l'action a est exécutée *immédiatement*, alors que dans $\langle a, \bullet\rangle$, l'action est effectuée à un temps inconnu qui peut être zéro.

Actions évitables. À partir d'un état donné q, une action évitable u (comme dans la figure 4) peut être empêchée par toute autre action c à partir du même état q par l'action temporisée $\langle c, \mathbf{0}\rangle$.

Actions Inéluctables. À partir d'un état donné, une action inéluctable $\langle u, \bullet\rangle$ se produira immanquablement si nous ne faisons rien d'autre à partir de cet état, c'est-à-dire si nous attendons assez longtemps.

Évitabilité et inéluctabilité vs équité. Une action inéluctable peut également être évitée à partir d'un état donné, ainsi le jeu d'accessibilité illustré à la figure 2 est gagnant en effectuant $\langle c, \mathbf{0}\rangle$.

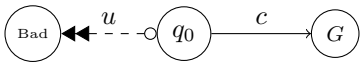


FIGURE 2 – Une action évitable (même inéluctable) peut être empêchée par le contrôleur

De plus, comme toute action incontrôlable non évitable, une action inéluctable (non évitable) ne peut être empêchée par une action contrôlable. D'un autre côté, elle peut être empêchée par une action incontrôlable qui réaliserait une boucle en temps nul. Par conséquent, les jeux d'accessibilité de la Figure 3 ne sont pas gagnants, ce qui montre la différence avec la notion d'équité.

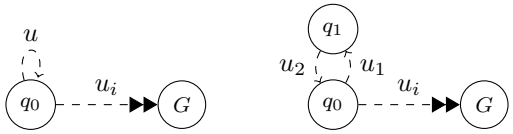


FIGURE 3 – Inéluctabilité n'est pas équité.

2.3 Prédécesseur, successeur et exécution

Pour $\Sigma \subseteq A_C \cup A_U$, nous définissons les fonctions prédécesseur et successeur $\text{pre}_\Sigma : 2^Q \to 2^Q, \text{suc}_\Sigma : 2^Q \to 2^Q$: soit $X \subseteq Q$, $\forall q \in Q, q \in \text{pre}_\Sigma(X)$ ssi $\exists a \in \Sigma$ et $q' \in X$, t.q. $q \xrightarrow{a} q'$, et $\forall q' \in Q, q' \in \text{suc}_\Sigma(X)$ ssi $\exists a \in \Sigma$ et $q \in X$, t.q. $q \xrightarrow{a} q'$. Si $\Sigma = A_C \cup A_U$, nous notons $\text{pre}(X)$ and $\text{suc}(X)$

L'*exécution* d'une structure de jeu est une séquence $q_0 \langle a_1, t_1\rangle q_1 \langle a_2, t_2\rangle q_2 \ldots$ avec $a_i \in A_C \cup A_U$, $t_i \in \Delta$, $q_i \in Q$, et telle que $q_i \xrightarrow{a_i} q_{i+1}$ pour tout $i >= 0$. Nous notons $\mathcal{R}$, l'ensemble des exécutions, et $\overline{\mathcal{R}}$ l'ensemble des exécutions finies. Notons qu'une exécution finie termine toujours avec un état.

Pour une exécution $r \in \mathcal{R}$, nous définissons $\text{First}(r)$ le premier état de r, $\text{States}(r)$ l'ensemble des états qui apparaissent dans r, et $\text{Act}(r)$ l'ensemble des actions qui apparaissent dans r. Si $r \in \overline{\mathcal{R}}$, nous définissons $\text{Last}(r)$ le dernier état de r. Nous définissons la longueur $|r|$ d'une exécution r comme la taille de la sous-séquence $\langle a_1, t_1\rangle\langle a_2, t_2\rangle \ldots$.

Pour $R \subseteq \mathcal{R}$ et $X \subseteq Q$, nous notons $R|_X$ le sous-ensemble de R tel que $\forall r \in R|_X, \text{States}(r) \subseteq X$.

3 Justification de ce nouveau modèle

Pour les exemples utilisés dans cette section, nous considérons un jeu d'accessibilité (défini formellement dans la Section 5) démarrant en q_0 où l'objectif est d'atteindre un état noté G.

3.1 Action évitable

Le problème des actions évitables (non immédiates) peut être résolu en utilisant des modèles temporisés tels que des automates temporisés. Les actions évitables peuvent être traduites directement en gardes avec une borne temporelle inférieure non nulle sur les horloges, comme illustré dans les figures 4.a et 4.b. Ainsi, les jeux temporisés [10, 16] permettent de résoudre le problème de synthèse du contrôleur pour des objetifs d'accessibilité ou de sureté.

Dans [20], les auteurs considèrent une abstraction des automates temporisés [2] pour laquelle une transitions τ représentent le fait que du temps s'écoule. Les auteurs considèrent ces transitions temporelles abstraites (τ) comme équivalentes à des transitions contrôlables vis à vis de la synthèse de contrôleur. Cette abstraction ne nécessite pas de délais explicites et si τ représente un écoulement de temps non nul, à partir de l'état q_0, une action τ suivie d'une action incontrôlable u est équivalente à une action évitable u à partir de q_0 comme illustré à la Figure 4.c. Dans [20], cette abstraction est générée par un quotient des automates de jeu temporisés par une bisimulation en temps abstrait et peut être considérée comme un graphe de jeu sur lequel la complexité de l'algorithme de synthèse du contrôleur est quadratique dans la taille du graphe.

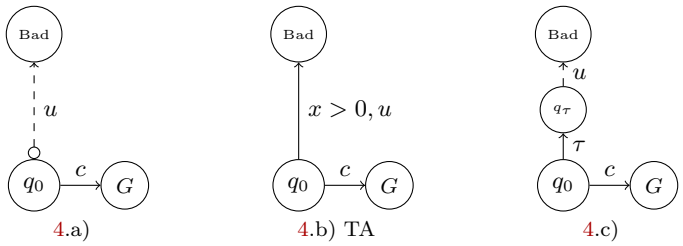


FIGURE 4 – Une action évitable incontrôlable peut être empêchée par le contrôleur

3.2 Action inéluctable.

Les actions inéluctables se produisent immanquablement dans un contexte nominal : la fin d'une transmission ou d'une conversion, ou plus généralement, un accusé de réception d'une commande.

Action ineluctable vs action contrôlable Dans le contexte non temporisé, il ne suffit pas de considérer ces actions comme contrôlables. Premièrement, contrairement à une action contrôlable, une action inéluctable ne peut pas être empêchée par le contrôleur, même si cela entraîne la perte du jeu (voir Figure 5). Deuxièmement, quand il y a un choix entre deux actions contrôlables, le contrôleur choisit mais entre deux actions inéluctables, c'est l'environnement qui choisit. Par exemple, dans la Figure 6, considérons l'émission d'un message sur un bus de communication (action c). Cela peut mener à un succès immédiat (action u_2) ou il peut d'abord échouer (action u_1) et peut devenir un succès plus tard. Il est inéluctable que soit u_1 soit u_2 va se produire, mais le choix entre u_1 et u_2 ne dépend pas du contrôleur qui doit s'assurer que les deux états q_1 et q_2 sont gagnants.

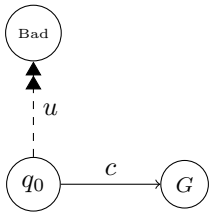


FIGURE 5 – Une action Ineluctable ne peut pas être empêchée par le contrôleur

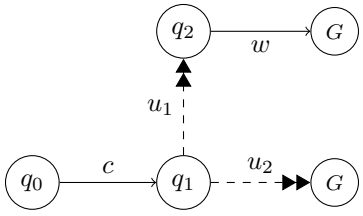


FIGURE 6 – Le choix d'une action ineluctable (u_1 ou u_2) ne dépend pas du contrôleur

Action inéluctable vs action temporisée. Dans le contexte temporel, l'inéluctabilité ne peut pas être traduite *telle-quelle* en automates temporisés. Nous pouvons utiliser des invariants sur les localités pour forcer l'environnement à jouer, mais cela nécessite la connaissance d'une limite supérieure du délai, ce qui est souvent impossible.

De plus, les invariants s'appliquent à tous les joueurs, y compris le contrôleur, alors que les actions inéluctables ne font que restreindre le comportement de l'environnement.

Dans [8], les jeux temporisés sont basés sur des automates temporisés avec des invariants restreints aux contraintes de la forme $x \leq k$ (où x est une horloge et k est une constante). Cependant, l'environnement peut décider de ne pas agir si un invariant impose de quitter un état alors que le contrôleur peut le faire.

Bien que les travaux en cours ne l'aient pas encore fait, il est possible d'étendre les automates de jeux temporisés afin de prendre en compte l'inéluctabilité, par exemple en élargissant la notion d'échéance ou d'urgence [6].

Cependant, les jeux temporisés d'accessibilité et de sureté sont certes décidables, mais de complexité EXPTIME-complet et les états symboliques manipulés par les algorithmes sont des *régions* ou des *zones* qui sont trop puissants pour les modèles non temporisés, complexifient les calculs et limitent la taille des systèmes qui peuvent être traités en pratique.

Notre modèle élimine le besoin de mettre des valeurs explicites sur les invariants de temps et ne contraint que le comportement de l'environnement et non celui du contrôleur.

4 Synthèse de contrôleur

Dans cette section, nous allons résoudre le problème de synthèse du contrôleur en utilisant notre sémantique modifiée. L'objectif est d'élaborer une stratégie permettant au contrôleur de restreindre le comportement du jeu afin de le gagner. Ces stratégies préconisent soit un ensemble de mouvements contrôlables à effectuer immédiatement ou non, soit d'attendre et de ne rien faire jusqu'à ce qu'une action soit réalisée par l'environnement, ce qui est représenté par un ensemble vide.

Définition 2 (Stratégie)**.** *Une stratégie s_i pour un joueur $i \in \{C, U\}$ est une fonction $s_i : \overline{\mathcal{R}} \to 2^{(A_i \times \Delta)}$. Elle est dite* sans mémoire *si elle dépend seulement de l'état courant de l'exécution, i.e. $s_i : Q \to 2^{(A_i \times \Delta)}$.*

Nous imposons que si $\langle a, d\rangle \in s(r)$, alors a est en effet possible à partir de $\mathrm{Last}(r)$.

Définition 3 (Stratégies avec actions inéluctables et évitables)**.** *Soit $s_U : \overline{\mathcal{R}} \to 2^{(A_U \times \Delta)}$, une stratégie de l'environnement et soit r, une exécution dans le jeu, avec $\mathrm{Last}(r) = q$.*

Si il existe $a \in A_U^{\diamond}$, $d \in \Delta$, et un état q' tel que $q \xrightarrow{\langle a, d\rangle} q'$ alors $s_U(r) \neq \emptyset$.

Si il existe $a \in A_U^{\star}$, $d \in \Delta$, et un état q' tel que $q \xrightarrow{\langle a, d\rangle} q'$, et si $\langle a, d\rangle \in s_U(r)$, alors $d \neq \mathbf{0}$.

Partant d'une exécution comprenant un certain état (généralement l'état initial), les deux joueurs construisent inductivement un ensemble d'exécutions (à cause du non-déterminisme) en jouant leur stratégie. Puisque nous nous intéressons aux stratégies permettant au contrôleur de gagner quelle que soit la stratégie (conforme aux définitions ci-dessus) de l'environnement, nous définissons directement les résultats d'une stratégie du contrôleur, en tant que l'union sur toutes les stratégies de l'environnement de tous ces ensembles d'exécutions.

Définition 4 (Résultat)**.** *Soit $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$, une structure de jeu, r une de ses exécutions, et s_C, une stratégie pour le contrôleur. Le* résultat $\mathrm{Outcome}(q, s_C)$ *de s_C à partir de l'état q est le sous-ensemble de $\mathcal{R}$ défini inductivement par :*

— $q \in \text{Outcome}(q, s_C)$
— *Si* $r \in \text{Outcome}(q, s_C)$ *est fini,* $r' = r \xrightarrow{\langle a,d \rangle} q' \in \text{Outcome}(q, s_C)$ *si* $r' \in \overline{\mathcal{R}}$ *et une de ces conditions est vraie :*
  — $a \in A_U^{\overline{\star}}$ *et* $d = \bullet$ *si* $\nexists (r \xrightarrow{a'} q''$ *t.q.* $\langle a', \mathbf{0} \rangle \in s_C(r))$*, ou* $d = \mathbf{0}$ *sinon ;*
  — $a \in A_U^{\star}$*,* $d = \bullet$*, et* $\nexists (r \xrightarrow{a'} q''$ *t.q.* $\langle a', \mathbf{0} \rangle \in s_C(r))$*.*
  — $\langle a, d \rangle \in s_C(r)$*.*
— *une exécution infinie appartient à* $\text{Outcome}(q, s_C)$ *si tous ses prefixes finis appartiennent aussi à* $\text{Outcome}(q, s_C)$

Intuitivement, nous nous intéressons aux exécutions qui sont suffisamment longues pour avoir une chance d'atteindre l'objectif. La maximalité distingue les exécutions les plus longues que le contrôleur peut produire, par ses actions (éventuellement en détournant les mouvements de l'environnement) ou en s'appuyant sur les actions inéluctables de l'environnement.

Une exécution r est *maximale* dans un ensemble d'exécutions R si, soit elle est finie et il n'y a ni $a \in A_C \cup A_U^{\diamond}$, ni $q' \in Q$ tel que $r \xrightarrow{a} q' \in R$, soit elle est infinie et aucun de ses préfixes finis n'est maximal. Nous notons $\text{MaxOutcome}(q, s_C)$ l'ensemble des exécutions qui sont maximales dans $\text{Outcome}(q, s_C)$.

Le problème de synthèse de contrôleur peut être énoncé en utilisant des objectifs, ou des conditions gagnantes. Pour une structure de jeu donnée $\mathcal{G}$, une condition gagnante $C_{\mathcal{W}}$ est un ensemble d'exécutions autorisées. Nous appelons la paire $(\mathcal{G}, C_{\mathcal{W}})$ un *jeu*.

Dans un tel jeu, une stratégie s pour le contrôleur est gagnante à partir de l'état q si $\text{MaxOutcome}(q, s) \subseteq C_{\mathcal{W}}$. Un état q est gagnant s'il existe une stratégie gagnante à partir de q. Le jeu lui-même est gagnant si q_0 est gagnant.

5 Jeux d'accessibilité

Un objectif d'accessibilité du contrôleur est de forcer le jeu à atteindre un certain ensemble d'états. Formellement :

Définition 5 (Objectif d'accessibilité)**.**
Soit $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ *une structure de jeu, et* $\text{Goal} \subseteq Q$ *un ensemble d'états objectifs. L'objectif d'accessibilité* $\text{Reach}(\text{Goal})$ *pour* Goal *est l'ensemble des exécutions* r *qui sont maximales dans* $\mathcal{R}$ *et telles que* $\text{States}(r) \cap \text{Goal} \neq \emptyset$*.*

Par exemple, pour le jeu de la Figure 7, l'objectif est d'atteindre l'état G et nous avons $\text{Goal} = \{G\}$ et $\text{Reach}(\text{Goal}) = \{q_0\ \langle c, \bullet \rangle\ q_1\ \langle u, \bullet \rangle\ G\}$.

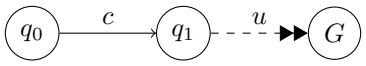


FIGURE 7 – L'objectif est d'atteindre l'état G.

5.1 Calcul de la stratégie

Le calcul de la stratégie est obtenu à partir de l'ensemble des états gagnants. Un état est gagnant pour le contrôleur s'il est possible d'atteindre un état d'objectif à partir de la stratégie, c'est-à-dire si le contrôleur dispose d'une stratégie pour atteindre un état d'objectif contre

toutes les stratégies de l'environnement. L'algorithme principal pour le calcul des stratégies gagnantes pour les jeux d'accessibilité est un algorithme de point fixe en arrière sur la fonction *prédécesseur contrôlable*.

Intuitivement, un état s est un prédécesseur contrôlable de X si les conditions suivantes sont remplies :

— il y a une action dont l'occurence est garantie (soit contrôlable, soit incontrôlable inéluctable) et conduit à X ;

— aucune autre action de l'environnement ne peut empêcher le jeu d'atteindre un état de X.

Définition 6 (Prédécesseurs Contrôlables)**.**
Soit $\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$, une structure de jeu, et $X \subseteq Q$ un ensemble d'états. Le prédécesseur contrôlable $\pi(X)$ de X est le sous-ensemble de Q défini par :

$$\begin{aligned} \pi(X) = \mathrm{pre}_{A_C}(X) \setminus \mathrm{pre}_{A_U^{\overline{\tau}}}(\overline{X}) \\ \cup \mathrm{pre}_{A_U^{\diamond}}(X) \setminus \mathrm{pre}_{A_U}(\overline{X}) \end{aligned} \tag{1}$$

Les deux parties de la formule représentent deux manières différentes de gagner :

— s'il existe une action contrôlable allant de s à un état dans X, toutes les actions incontrôlables doivent être soit évitables, soit aussi conduire à des états dans X

— s'il existe une action incontrôlable inéluctable, toutes les autres actions incontrôlables doivent également conduire à un état dans X.

A partir de cette nouvelle définition de π, l'ensemble des états gagnants est calculé à l'aide de l'algorithme de point fixe classique suivant :

$\mathcal{W}_0 = \mathrm{Goal}$ and $\mathcal{W}_{n+1} = \mathcal{W}_n \cup \pi(\mathcal{W}_n)$. Lorsqu'il existe, l'ensemble point fixe final est noté $\mathcal{W}$.

Algorithm 1 Algorithme de calcul des états gagnants pour le jeu d'accessibilité

Input: $\mathcal{G} = (Q, q_0, A_C, A_U, \rightarrow), \mathrm{Goal} \subseteq Q$
Output: $\mathcal{W}$
  $\mathcal{W} \leftarrow \mathrm{Goal}$
  while $\pi(\mathcal{W}) \not\subseteq \mathcal{W}$ **do**
    $\mathcal{W} \leftarrow \mathcal{W} \cup \pi(\mathcal{W})$
  end while
  return $\mathcal{W}$

Lemme 1. *Soit $(\mathcal{G}, C_{\mathcal{W}})$, un jeu d'accessibilité. Soient q_1 et q_2, deux états de $\mathcal{G}$. Soient s_1, une stratégie sans mémoire qui est gagnante à partir de q_1 et s_2, une stratégie sans mémoire qui est gagnante à partir de q_2. Soit Q_1, un ensemble d'états des exécutions r de* Outcome(q_1, s_1) *telles que* States$(r) \cap$ Goal $= \emptyset$ *(i.e. les états parcourus avant d'atteindre* Goal*).*

Soit s, la stratégie sans mémoire définie par : pour tout $q \in Q$, si $q \in Q_1$ alors $s(q) = s_1(q)$, sinon $s(q) = s_2(q)$. Alors s est gagnante à la fois de q_1 et q_2.

Lemme 2. *Si $q \in \mathcal{W}_n$ (i.e. la valeur de $\mathcal{W}$ à la fin de la n-ème itération de la boucle while) alors il existe une stratégie gagnante sans mémoire à partir de q qui permet de gagner en n étapes ou moins.*

A partir des Lemmes 2 et **??**, nous pouvons déduire le résultat suivant :

Théorème 1 (Complétude et correction)**.** *$q \in \mathcal{W}$ si et seulement si q est gagnant.*

Théorème 2 (Strategies sans mémoire)**.** *Si le jeu est gagnant, alors il est gagnant avec une stratégie sans mémoire.*

On peut construire effectivement une stratégie gagnante sans mémoire lorsque le jeu est gagnant : à chaque itération, chaque nouvel état ajouté à $\mathcal{W}$ a soit au moins une transition contrôlable ou une transition incontrôlable inéluctable vers un état de $\mathcal{W}$ qui a été ajouté à une précédente itération. La stratégie peut être l'ensemble (ou un de ses sous-ensembles) de ces actions contrôlables.

Il est clair que cette stratégie garantit également que les états objectif sont atteints en le nombre minimal d'étapes possible.

Notons également que, comme toujours pour les jeux d'accessibilité, la stratégie canonique qui permettrait toujours de passer à n'importe quel état de $\mathcal{W}$ n'est pas gagnante en général puisque cela permettrait des boucles dans $\mathcal{W}$, et donc des exécutions infinies n'atteignant jamais les états de l'objectif.

5.2 Exemple de jeu d'accessibilité

Considérons le jeu d'accessibilité$\mathcal{G} = (Q, q_0, A_C, A_U, \delta)$ de la Figure 8 pour lequel l'objectif est d'atteindre l'état G : Goal $= \{G\}$. En appliquant l'algorithme 1 de point fixe en arrière : $\mathcal{W}_0 =$ Goal and $\mathcal{W}_{n+1} = \mathcal{W}_n \cup \pi(\mathcal{W}_n)$, nous obtenons successivement :

$\mathcal{W}_0 = \{G\}$, $\pi(\mathcal{W}_0) = \{q_4\}$, $\mathcal{W}_1 = \{G, q_4\}$, $\pi(\mathcal{W}_1) = \{q_3, q_4\}$, $\mathcal{W}_2 = \{G, q_3, q_4\}$, $\pi(\mathcal{W}_2) = \{q_2, q_3, q_4\}$, $\mathcal{W}_3 = \{G, q_2, q_3, q_4\}$, $\pi(\mathcal{W}_3) = \{q_0, q_2, q_3, q_4\}$, $\mathcal{W}_4 = \{G, q_0, q_2, q_3, q_4\}$, $\pi(\mathcal{W}_4) = \{q_0, q_2, q_3, q_4\}$

Une stratégie gagnante sans mémoire est $s(q_0) = \{\langle c_1, \mathbf{0}\rangle\}$, $s(q_3) = \{\langle c_2, \mathbf{0}\rangle\}$, $s(q_4) = \{\langle c_3, \bullet\rangle\}$ et $s(q_1) = s(q_2) = s(G) = \emptyset$.

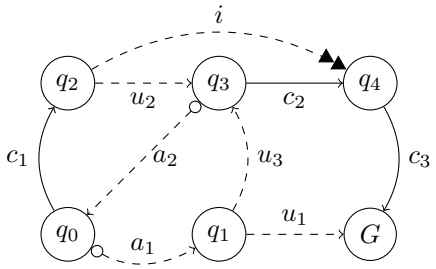


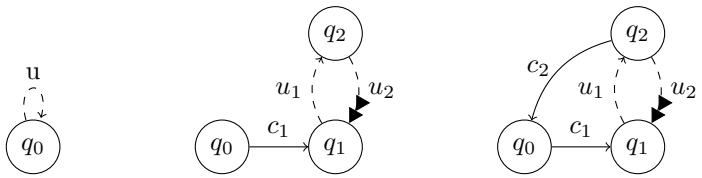
FIGURE 8 – Un jeu d'accessibilité. L'objectif est d'atteindre l'état G.

6 Jeu de sûreté

Un objectif de sûreté pour le contrôleur est de forcer le jeu à rester dans un ensemble d'états spécifié ou, de manière équivalente, d'éviter un ensemble d'états.

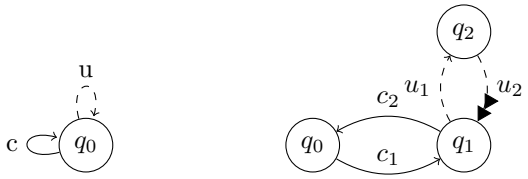
Définition 7 (Objectif de sûreté)**.**
Soit $\mathcal{G} = (Q, q_0, A_C, A_U, \rightarrow)$, une structure de jeu et Safe $\subseteq Q$ *un ensemble d'états* sûrs *. L'objectif de sûreté pour* Safe *est l'ensemble de toutes les exécutions maximales infinies r de $\mathcal{G}$ telles que* States$(r) \subseteq$ Safe*.*

Notons que nous excluons les exécutions maximales finies de l'objectif car nous ne voulons pas que le contrôleur gagne en bloquant ou en atteignant un livelock incontrôlable, c'est-à-dire un ensemble d'états sans transition contrôlable sortante. Cela signifie que lorsque l'environnement décide de ne pas jouer, le contrôleur doit pouvoir bouger. Par conséquent, les jeux de sûreté de la figure 9 où tous les états sont dans l'ensemble des états *sûrs*, soont perdants. En effet, pour le jeu de la Figure 9.a, nous avons $q_0 \notin \pi(\{q_0\})$et pour les jeux des Figures 9.b et 9.c, nous avons $q_1 \notin \pi(\{q_0, q_1, q_2\})$ signifiant que l'environnement peut bloquer dans q_1 (en ne jouant pas u_1 qui n'est pas inéluctable) et pour éviter q_1, le contrôleur doit bloquer dans q_0. A contrario, les jeux de la Figure 10 sont gagnants.



9.a) 9.b) 9.c)

FIGURE 9 – Tous les états sont sûrs mais les jeux ne sont pas gagnants.



10.a) 10.b)

FIGURE 10 – Tous les états sont sûrs et les jeux sont gagnants.

6.1 Calcul de la stratégie

La stratégie est calculée à partir des états gagnants. Un état est gagnant pour le contrôleur si il est possible de forcer le jeu à rester dans Safe.

Etant donnée notre nouvel définition de π, l'ensemble des états gagnants pour le contrôleur est calculé en utilisant l'algorithme de point fixe en arrière classique : $\mathcal{W}_0 = \text{Safe}$ et $\mathcal{W}_{n+1} = \mathcal{W}_n \cap \pi(\mathcal{W}_n)$.

Quand il existe, le point fixe final est noté $\mathcal{W}$.

Lemme 3. *Si $q \in \mathcal{W}_n$ alors il existe une stratégie sans mémoire s telle que pour tout préfixe r de longueur n d'une exécution dans* $\text{MaxOutcome}(q, s)$*, nous avons* $\text{States}(r) \subseteq \text{Safe}$.

Lemme 4. *Si il existe une stratégie s et une exécution r telle que pour tout préfixe r' de longueur n d'une exécution dans* $\text{MaxOutcome}(q, s)$*, nous avons* $\text{States}(r') \subseteq \text{Safe}$*, donc* $\text{Last}(r) \in \mathcal{W}_n$.

A partir de ces deux lemmes, nous obtenons le résultat principal :

Théorème 3 (Complétude et correction)**.** *$q \in \mathcal{W}$ si et seulement si q est gagnant.*

Algorithm 2 Algorithme de calcul des états gagnants pour les jeux de sûreté

Input: $\mathcal{G} = (Q, q_0, A_C, A_U, \rightarrow), \text{Safe} \subseteq Q$
Output: $\mathcal{W}$
  $\mathcal{W} \leftarrow \text{Safe}$
  while $\mathcal{W} \not\subseteq \pi(\mathcal{W})$ **do**
    $\mathcal{W} \leftarrow \mathcal{W} \cap \pi(\mathcal{W})$
  end while
  return $\mathcal{W}$

Théorème 4 (Stratégies sans mémoire)**.** *Si le jeu est gagnant alors il est gagnant avec une stratégie sans mémoire*

Pour des jeux de sûreté, et étant donné le résultat précédent, il est clair qu'aller dans n'importe quel état gagnant est toujours une stratégie gagnante pour le contrôleur. Nous définissons une stratégie sans mémoire canonique $s^s : \mathcal{W} \rightarrow 2^{(A_C \times \Delta)}$ qui agit exactement ainsi :

Soit $s^s(q) = \{\langle a, d\rangle | a \in A_C, q \xrightarrow{a} q' \Rightarrow q' \in \mathcal{W}\}$, avec $d = \mathbf{0}$ si $\exists a' \in A_U^\star, q'' \notin \mathcal{W}$ et $d = \bullet$ sinon.

La permissivité d'une stratégie est une notion importante dans le domaine du contrôle et de la supervision [18]. La permissivité est mesurée en terme d'ensemble de comportements autorisés par la stratégie [5]. Ainsi la stratégie la plus permissive n'existe pas nécessairement selon le type d'objectif gagnant.

Théorème 5. *La stratégie s^s est la stratégie gagnante la plus permissive pour l'objectif de sûreté* Safe*, i.e, pour toute stratégie gagnante s', on a* $\text{Outcome}(q_0, s') \subseteq \text{Outcome}(q_0, s^s)$.

6.2 Exemple de jeu de sûreté

Considérons le jeu de sûreté $\mathcal{G} = (Q, q_0, A_C, A_U, \rightarrow)$ de la Figure 11 pour lequel l'objectif est d'éviter l'état B. Ainsi $\text{Safe} = \{q_0, q_1, q_2\}$ est l'ensemble des états *sûrs*.

En appliquant l'algorithme 2 de point fixe en arrière : $\mathcal{W}_0 = \text{Safe}$ and $\mathcal{W}_{n+1} = \mathcal{W}_n \cap \pi(\mathcal{W}_n)$, nous obtenons successivement :

$\mathcal{W}_0 = \{q_0, q_1, q_2\}$, $\pi(\mathcal{W}_0) = \{q_0, q_1\}$, $\mathcal{W}_1 = \{q_0, q_1\}$, $\pi(\mathcal{W}_1) = \{q_0, q_1\}$.

La stratégie gagnante la plus permissive est $s(q_0) = \emptyset$ et $s(q_1) = \{\langle c, \mathbf{0}\rangle\}$.

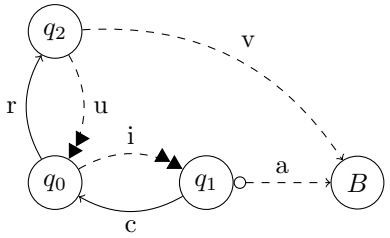


FIGURE 11 – Un jeu de sûreté gagnant. L'objectif est d'éviter l'état B.

7 Accessibilité sûre

D'un point de vue pratique, l'accessibilité et la sûreté doivent souvent être menées conjointement. Il s'agit d'atteindre un état objectif en évitant les états qui ne sont pas sûrs.

Il ne suffit pas, pour ce faire, d'appliquer successivement le calcul des états gagnants pour jeu d'accessibilité puis pour le jeu de sûreté car si la stratégie pour l'accessibilité consiste à passer par des états qui ne sont pas sûrs, ces états seront retirés par le jeu de sûreté et l'état objectif ne sera plus accessible. Ainsi pour le jeu de le Figure 12, sans considérer qu'il faut éviter B, tous les états sont gagnants pour le jeu d'accessibilité. Ainsi une stratégie du jeu d'accessibilité pour atteindre l'état G consiste à faire c_1 puis c_2 puis c_3 mais le jeu de sûreté retire ensuite l'état B et le système bloque dans l'état q_1.

De même, si on applique d'abord le jeu de sûreté, la stratégie peut consister à attendre une action inéluctable à partir d'un état qui était sûr à condition de le quitter immédiatement comme illustré sur la Figure 12. Si on applique successivement le calcul des états gagnants pour le jeu de sûreté puis pour le jeu d'accessibilité, on obtient d'abord que le jeu de sûreté retire l'état B. Puis, une stratégie du jeu d'accessibilité consiste à jouer c_4 puis attendre l'occurence de u_2 alors que cette attente peut permettre l'occurence de u_1 ce qui mènerait dans B.

Il est ainsi nécessaire de faire un point fixe dédié à ce type de propriétés.

L'ensemble des états gagnants pour un jeu d'accessibilité sûre peut ainsi être calculé à l'aide de l'algorithme 3 de point fixe en arrière $\mathcal{W}_0 = \text{Goal} \cap \text{Safe}$ and $\mathcal{W}_{n+1} = \mathcal{W}_n \cup \pi(\mathcal{W}_n) \cap \text{Safe}$.

Lorsqu'il existe, l'ensemble point fixe final est noté $\mathcal{W}$.

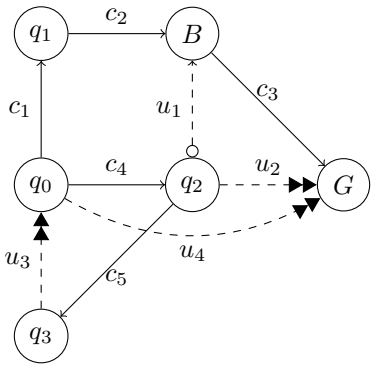


FIGURE 12 – L'objectif est d'atteindre l'état G en évitant l'état B.

Algorithm 3 Algorithme de calcul des états gagnants pour le jeu d'accessibilité sûre

Input: $\mathcal{G} = (Q, q_0, A_C, A_U, \rightarrow), \text{Goal} \subseteq Q, \text{Safe} \subseteq Q$
Output: $\mathcal{W}$
  $\mathcal{W} \leftarrow \text{Goal} \cap \text{Safe}$
  while $\pi(\mathcal{W}) \cap \text{Safe} \not\subseteq \mathcal{W}$ **do**
    $\mathcal{W} \leftarrow \mathcal{W} \cup \pi(\mathcal{W}) \cap \text{Safe}$
  end while
  return $\mathcal{W}$

L'application de cet algorithme sur le jeu de la Figure 12 donnera successivement $W_0 = \{G\}$ et $W_1 = \{G, q_0\}$. $W_2 = \{G, q_0, q_3\}$. $\mathcal{W} = W_3 = \{G, q_0, q_3, q_2\}$. La stratégie pour atteindre l'état

G en évitant l'état B consiste donc à attendre dans l'état q_0 l'occurence de l'action inéluctable u_4.

8 Complexité et implémentation

Les algorithmes que nous donnons sont bien adaptés à la présentation pédagogique, et éventuellement à une implémentation utilisant des représentations symboliques d'ensembles d'états basées sur des diagrammes de décision. En revanche, ils ne sont pas optimaux pour une énumération explicite d'états. Néanmoins, en reliant notre définition des prédécesseurs contrôlables π à l'algorithme non temporisé de [7], nous pouvons calculer les états gagnants pour l'accessibilité, ou leur complément pour la sûreté, en temps linéaire par rapport au nombre de transitions de l'automate.

Sur la base de ce dernier algorithme, nous avons implémenté le calcul des états gagnants et la synthèse de la stratégie dans notre outil Roméo [15]. Dans son langage d'entrée textuel, Roméo gère un modèle appelé Clock Transition Systems (CTS) [14] qui englobe les automates finis et les réseaux de Petri. Nous avons étendu les CTS avec des actions contrôlables, incontrôlables, évitables et inéluctables afin de modéliser des jeux logiques temporisés. Le CTS peut être généré à partir de l'interface graphique.

9 Étude de cas

La synthèse des pilotes de périphériques est un bon exemple de synthèse de contrôleurs sur un jeu en temps logique. Dans ce cas, l'environnement est i) le périphérique matériel et ses connexions aux systèmes externes : réseaux de communication, signaux analogiques, etc. et ii) l'application utilisant le pilote. Dans le premier cas, les actions incontrôlables sont des interruptions qui sont déclenchées pour signaler, par exemple, la disponibilité d'une donnée dans un tampon matériel. Dans le second cas, il s'agit de demandes formulées par l'application. Dans les deux cas, les temps exacts ne sont pas connus car ils dépendent du matériel réel et du temps d'exécution du programme binaire qui n'est pas encore disponible. Cependant, certaines règles liées au temps sont connues comme le temps entre deux arrivée es messages sur un réseau de communication ou le temps entre deux interruptions d'un timer par exemple. Ainsi, lorsqu'il réagit à une action incontrôlable, le contrôleur a le temps d'effectuer sa tâche avant l'arrivée de la prochaine action incontrôlable. Dans ce cas, la deuxième action est incontrôlable et évitable.

9.1 Modélisation du pilote de contrôleur CAN

Le périphérique choisi pour l'étude de cas est le contrôleur CAN Microchip disponible dans la famille de microcontrôleurs PIC18Cxx8 [17]. Ce contrôleur CAN dispose de deux tampons de réception, RXB0 et RXB1 et de trois tampons de transmission, TXB0, TXB1 et TXB2. Chacun de ces tampons peut contenir un message CAN complet. Pour des raisons de simplicité, nous ne considérons dans cette étude de cas qu'un seul tampon d'émission, appelé TXB. Le périphérique est configuré de telle sorte que i) lorsqu'un message est reçu du bus, il est placé dans l'un des tampons de réception et qu'une interruption est activée. ii) lorsqu'un message est écrit dans le tampon d'émission, l'appareil l'envoie dès que possible et active une interruption pour avertir TXB qui vient d'être vidé.

Le modèle du pilote est présenté à la figure 13. Nous avons ajouté des variables booléennes : PW (En attente d'écriture), RXB0IF (Drapeau d'interruption de RXB0), RXB1IF (Drapeau

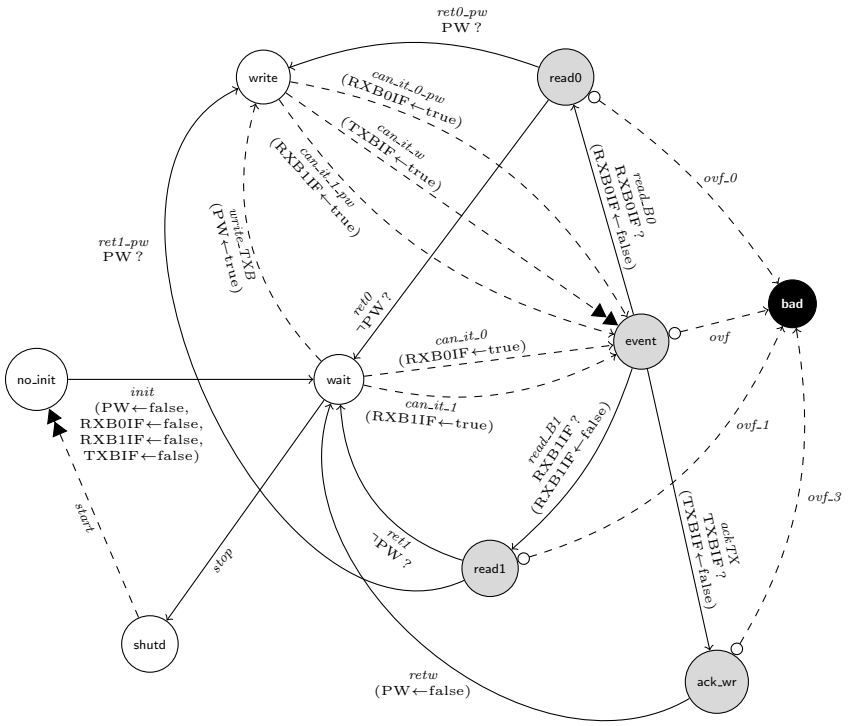


FIGURE 13 – Modèle du pilote du contrôleur CAN des PIC18Cxx8. Les gardes sont notées avec un ' ?', les négations sont notées avec un ┐ et les mises à jour sont notées entre parenthèses avec un ←.

d'interruption de RXB1), TXBIF (Drapeau d'interruption de TXB) pour simplifier le dessin du modèle. Le pilote est découpé en deux parties : la partie qui est exécutée en mode utilisateur, représentée par des états blancs et la partie qui est exécutée dans le gestionnaire d'interruption représentée par des états gris. De plus, un état noir bad qui doit être évité par le contrôleur est ajouté. À partir de l'état no_init le périphérique peut être configuré comme décrit ci-dessus et le pilote attend les requêtes dans l'état wait. De là, trois actions incontrôlables, correspondant à une demande d'écriture de l'application (*write_TXB*) ou à l'arrivée d'un message dans l'un des tampons de réception (*can_it_0* ou *can_it_1*), peuvent se produire et la variable booléenne correspondante est mise à 1. De l'état write on retrouve les deux actions incontrôlables correspondant à l'arrivée d'un message et aussi une action inéluctable qui est effectuée par le périphérique lorsque TXB est vidé. L'état event représente le point d'entrée du gestionnaire d'interruption de périphérique. À partir de là, le contrôleur peut lire les actions correspondant au traitement de l'événement : lire le tampon de réception qui a été rempli (read0 ou read1) ou acquitter la vidange du tampon de transmission (ack_write).

Pendant l'exécution du gestionnaire d'interruptions, les actions incontrôlables peuvent être évitées parce que i) les interruptions du dispositif sont masquées ii) le contrôleur a suffisamment de temps pour jouer ses actions avant l'arrivée d'une nouvelle interruption.

9.2 Stratégie gagnante

Nous avons utilisé notre outil ROMÉO [15] pour la modélisation de cette étude de cas et pour calculer les états gagnants et la synthèse de la stratégie. Nous vérifions d'abord que la propriété de sûreté, où BAD n'est jamais atteinte, tient. Mais pour la sûreté, ROMÉO calcule en fait le complément du point fixe donné dans la section 6 et calcule donc une stratégie pour l'environnement de rendre fausse la propriété. Bien entendu, il n'en trouve pas. Ainsi, afin d'obtenir la stratégie pour le contrôleur, nous vérifions également un objectif d'accessibilité.

Pour exprimer cet objectif, nous devons ajouter deux variables booléennes appelées PLAYED_wait, qui est définie lorsque l'environnement joue une action pour quitter l'état wait, et PLAYED_write qui est définie lorsque l'environnement joue une action pour quitter l'état footnotesize write. De cette façon, rester dans l'état x (PLAYEDPLAYED_x est faux) et revenir à cet état après que l'environnement a joué (PLAYED_x est vrai) peut être différencié. Un état supplémentaire est également ajouté, shutd. L'état shutd modélise le fait que le système peut-être éteint. La transition *start* est la mise sous tension du système et la transition *stop* est la mise hors tension du système. Si l'environnement décide de ne jouer aucune action, shutdown sera accessible. Le but du contrôleur est d'atteindre l'un des états suivants :

— shutdown,
— wait avec PLAYED_wait = true,
— write avec PLAYED_write = true

TABLE 1 – Stratégie sans mémoire

state	variables	play	next
no_init	–	$\langle init, \bullet \rangle$	wait
wait	–	–	wait
write	–	–	write
write	–	–	event
event	¬RXB0IF, ¬RXB1IF, TXBIF	$\langle ackTX, \mathbf{0} \rangle$	ack_wr
event	RXB0IF, ¬RXB1IF, ¬TXBIF	$\langle read\_B0, \mathbf{0} \rangle$	read0
event	¬RXB0IF, RXB1IF, ¬TXBIF	$\langle read\_B1, \mathbf{0} \rangle$	read1
ack_wr	–	$\langle retw, \mathbf{0} \rangle$	wait
read0	PW	$\langle ret0\_pw, \mathbf{0} \rangle$	write
read0	¬PW	$\langle ret0, \mathbf{0} \rangle$	wait
read1	PW	$\langle ret1\_pw, \mathbf{0} \rangle$	write
read1	¬PW	$\langle ret1, \mathbf{0} \rangle$	wait
shutd	–	–	shutd

La stratégie est résumée dans le tableau 1. A partir de no_init le contrôleur doit jouer *init* pour atteindre un état gagnant. Dans wait, si l'environnement joue *can_it_0* ou *can_it_1*, PLAYED est défini et le contrôleur doit jouer immédiatement *read_B0* – *,ret0* ou *read_B1* – *ret1* respectivement, pour revenir à wait. Si l'environnement joue *write_TXB*, les deux PW et PLAYED sont définis. Dans write l'environnement peut choisir de jouer *can_it_0_pw* ou *can_it_1_pw*. Ensuite, le contrôleur doit jouer immédiatement *read_B0* – *ret0_pw* ou *read_B1* – *ret1_pw*, respectivement, pour retourner à write. Si l'environnement décide de ne pas jouer des actions incontrôlables, inévitablement *can_it_it_w* se produit et le contrôleur retourne immédiatement à l'état wait en jouant *ackTX* – *retw*.

10 Conclusion

Nous avons présenté une extension des automates finis avec du temps logique. Cette extension introduit deux nouvelles propriétés aux actions incontrôlables qui étendent le modèle de l'environnement :

— l'action *non immédiate* (évitable) ne peut pas se produire instantanément de sorte que le contrôleur peut effectuer une autre action de manière préventive si nécessaire.
— l'action *ineluctable* se produira finalement de manière garantie, et le contrôleur peut donc s'y fier.

Ce modèle combine une partie de l'expressivité des jeux temporisés, avec la simplicité des automates finis. Il permet une mise en œuvre plus aisée de ces modèles et est plus adapté aux systèmes temps réel embarqués.

Nous avons adapté la notion de contrôle, d'accessibilité et de jeux de sûreté pour cette extension et avons proposé des algorithmes pour résoudre ces problèmes dans le cas général.

Finalement nous avons implémenté le calcul des états gagnants et la synthèse de la stratégie dans notre outil Roméo.

Par la suite, l'approche sera étendue à des objectifs de contrôle plus complexes, tels que les conditions de Büchi, et traitera des comportements concurrents modélisés par des réseaux d'automates finis.

Références

[1] Karine Altisen and Stavros Tripakis. Tools for controller synthesis of timed systems. In *2nd Workshop on Real-Time Tools (RT-TOOLS'2002)*, july 2002.

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

[3] Jean-Luc Bechennec, Didier Lime, and Olivier H. Roux. Control of DES with urgency, avoidability and ineluctability. In *19th International Conference on Application of Concurrency to System Design (ACSD'19)*, June 2019.

[4] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim G Larsen, and Didier Lime. Uppaal-tiga : Time for playing games ! In *Computer Aided Verification*, pages 121–125. Springer, 2007.

[5] Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies : from parity games to safety games. *Theoretical Informatics and Applications (RAIRO : ITA)*, 36 :261–275, 2002.

[6] Sébastien Bornot and Joseph Sifakis. An algebraic framework for urgency. *Inf. Comput.*, 163(1) :172–202, 2000.

[7] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005–Concurrency Theory*, pages 66–80. Springer, 2005.

[8] Thomas Chatain, Alexandre David, and Kim G. Larsen. Playing games with timed games. In Alessandro Giua, Manuel Silva, and Janan Zaytoon, editors, *Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09)*, Zaragoza, Spain, September 2009.

[9] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Strategy improvement for concurrent reachability and safety games. *CoRR*, 2012.

[10] Luca De Alfaro, Marco Faella, Thomas A Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In *CONCUR 2003-Concurrency Theory*, pages 144–158. Springer, 2003.

[11] Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. *Theoretical Computer Science*, 386(3) :188 – 217, 2007.

[12] Luca De Alfaro, Thomas A Henzinger, and Rupak Majumdar. Symbolic algorithms for infinite-state games. In *CONCUR 2001—Concurrency Theory*, pages 536–550. Springer, 2001.

[13] C.H. Golaszewski and P.J. Ramadge. Control of discrete event processes with forced events. In *Proceedings of the 26th Conference on Decision and Control*, December 1987.

[14] Claude Jard, Didier Lime, and Olivier H Roux. Clock Transition Systems. In *21th international Workshop on Concurrency, Specification and Programming (CS&P 2012)*, Berlin, Germany, September 2012.

[15] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo : A parametric model-checker for Petri nets with stopwatches. In *TACAS 2009*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57, York, UK, March 2009. Springer.

[16] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS 95*, pages 229–242. Springer, 1995.

[17] Microchip. *PIC18CXX8 Data Sheet (DS30475A). High-Performance Microcontrollers with CAN Module.* http ://ww1.microchip.com/downloads/en/DeviceDoc/30475a.pdf, 2000.

[18] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1) :206–230, January 1987.

[19] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *STACS 95*, pages 1–13. Springer, 1995.

[20] Stavros Tripakis and Karine Altisen. On-the-fly controller synthesis for discrete and dense-time systems. In *In FM'99, volume 1708 of LNCS*, pages 233–252. Springer Verlag, 1999.

[21] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. In *The 23rd IEEE Conf. on Decision and Control, 1984.*, volume 23, pages 1073–1080, Dec 1984.