Normalisation et fermeture causale d'une courbe d'arrivée comprennant des portions affines et impact de la fermeture causale dans l'outil ac2lus

Guillaume Sarrazin
TER 2011
Verimag
Encadrants : Karine Altisen et Matthieu Moy

***Résumé*—ac2lus est un outil utilisant une paire de courbes définissant le flux minimal et maximal d'événements qui peuvent arriver durant un intervalle de temps. Ces courbes sont appelées courbes d'arrivée. Sans aucun traitement, ces courbes peuvent amener l'outil dans une situation de dead-lock à cause de contraintes implicites exprimées par la courbe. Un algorithme de causalification permet d'expliciter les contraintes implicites. La normalisation est une part de cet algorithme. Durant ce TER, j'ai implémenté cet algorithme de normalisation dans le cas de courbes d'arrivée contenant des morceaux affines.**
J'ai de plus créé des exemples traités par ac2lus pour lesquels les courbes d'arrivée n'étaient pas causales. Ces exemples ont montré que les courbes de sortie n'étaient pas forcemment plus précises quand on appliquait l'algorithme de causalification sur les courbes d'arrivée en entrée du sytème.

I. Introduction

La conception de système coûte de plus en plus cher. Pour minimiser les coûts de fabrication, la connaissance précise des systèmes est nécessaire. Pour cela, une méthode possible est la validation fonctionnelle. Mais cela n'est pas suffisant. Il est nécessaire de connaître un certain nombre de propriétés non fonctionnelles comme par exemple la quantité d'événements arrivant et sortant d'un système pendant un intervalle de temps. C'est dans ce cadre d'étude des caractéristiques d'un système que vient s'inscrire l'outil ac2lus [1]. ac2lus se base sur le Real Time Calculus (RTC) [4]. Il s'agit d'une méthode analytique utilisant des équations mathématiques rapidement résolvables et se basant sur le concept de flux d'événements. Le RTC permet d'obtenir le minimum et le maximum des performances du système du point de vue flux de donnée. Malheureusement cette technique ne permet pas de modéliser tous les types de système. Typiquement la présence d'états dans la structure du système n'est pas modélisable. Pour tenter de pallier à ce problème, on tente de combiner le RTC avec d'autres techniques. ac2lus implémente une approche possible en utilisant le langage synchrone `Lustre`.

Un flux d'événement peut être représenté par une courbe cumulative, c'est-à-dire : on ajoute le nombre d'événements arrivant en t à ceux déjà présent en $t-1$. On peut sinon utiliser une courbe appelée `courbes d'arrivée` qui représentent un ensemble de courbes cumulatives (exemple figure 1). Elles sont définies par une paire de courbes représentant le minimum et le maximum d'événements pouvant survenir pendant un intervalle de temps Δ. Le RTC et ac2lus utilisent des `courbes d'arrivée`.

A partir de la modélisation d'un système à l'aide d'équations propre au RTC et d'une courbe d'arrivée spécifiant les caractéristiques des courbes cumulatives pouvant arrivées en entrée du système, le RTC permet d'obtenir les courbes d'arrivée caractérisant le flux de sortie du système étudié.

La `causalité` est une propriété des `courbes d'arrivée` permettant de repérer des zones appelées `région interdite`, i.e. si une courbe cumulative arrive à un moment donné dans cette zone, elle ne peut pas en sortir sans violer à un moment les propriétés de la `courbe d'arrivée`. L'étude de la `causalité` pour ces courbes est récente. [2]. Cela permet de rendre les courbes plus précises et aussi d'éviter de faux contre-exemples ou des deadlocks dans des programmes prenant en entrée ces courbes.

Deux courbes d'arrivée sont `équivalentes` si elles admettent le même ensemble de courbes cumulatives.

Comme défini dans [3], l'action de `normaliser` une courbe d'arrivée ne peut se faire que si la courbe est définie en partie par des points et en partie par des droites. `Normaliser` correspond à prolonger la partie définie par des points jusqu'à ce que la courbe basse (resp. haute) passe définitivement au-dessous (resp. au-dessus) des droites de la courbe basse (resp. haute), à égaliser la taille de la partie finie des 2 droites puis à appliquer une propriété particulière à la courbe d'arrivée ce qui rend explicite un certain nombre de contraintes qui étaient jusque là implicites. La `causalité` englobe la `normalisation` et rend explicite toutes les contraintes implicites restantes en appliquant la propriété de causalité à la courbe d'arrivée.

La `causalité` (et donc la `normalisation`) a été

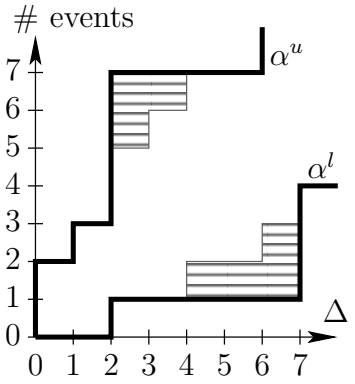

FIGURE 1: Courbe d'arrivée présentant des régions inatteignables

implémentée pour des `courbes d'arrivée` finies avant le début de mon stage. Mais cela n'a pas été fait pour des courbes d'arrivée infinies (i.e. définie avec une partie finie puis des droites) telles que les accepte ac2lus.

Les contributions de ce TER sont l'implémentation de l'algorithme de normalisation pour des courbes d'arrivée contenant des parties affines ainsi que l'étude de l'impact de la causalité sur les résultats apportés par ac2lus.

Ce papier se découpe en deux parties. Tout d'abord nous allons présenter les courbes d'arrivée et les différentes propriétés qui permettent de passer d'une courbe d'arrivée à une courbe d'arrivée équivalente mais plus précise, cette forme est appelée `forme causale`. Nous montrerons alors l'algorithme de normalisation implémenté. Nous présenterons ensuite ac2lus et l'impact de la `causalité` sur les résultats donnés par ac2lus.

II. Normalisation et causalification des courbes d'arrivée

A. Besoin de normaliser les courbes d'arrivée

Présentation des courbes d'arrivée: Comme dit précédemment une courbe d'arrivée est un ensemble formé de deux courbes (α^u, α^l), chaque courbe représentant une fonction : $\alpha(\Delta) = \gamma$ où γ représente le nombre d'événements arrivant pendant un intervalle de temps Δ. L'une des deux courbes α^u indique le nombre maximal d'événements pouvant survenir, et l'autre α^l le nombre minimum d'événements. Dans la figure 1, on voit que pour un intervalle de temps de taille 2, le système peut recevoir au maximum 3 événements, et au minimum 1 événement.

L'intérêt de la normalisation se voit clairement dans un exemple. En continuant de lire ainsi, on remarque que le système peut ne recevoir qu'un événement pendant un intervalle de temps de taille 5. Or cet intervalle est décomposable en 2 intervalles l'un de taille 3 et l'autre de taille 2. Si on applique les contraintes de la courbe, il faudrait que le système reçoive au minimum 2 fois un événement. Ainsi en se contentant de lire les propriétés explicites de la courbe, certaines contraintes implicites nous manquent.

Cela engendre des problèmes quand il s'agit d'utiliser ces courbes dans des algorithmes de calcul comme par exemple un générateur de courbe cumulative. Comme on verra plus loin, l'abscence de causalification pose aussi des problèmes. On peut notamment arriver dans des situations de deadlock si les courbes utilisées ne sont pas causales.

B. Présentation des différentes propriétés des courbes d'arrivée

La monotonie: les courbes d'arrivée sont croissantes. En effet soit une courbe d'arrivée (α^u, α^l). Soit 2 intervalles de temps Δ_1 et Δ_2 tel que $\Delta_2 > \Delta_1$, alors $\alpha^u(\Delta_1) \leq \alpha^u(\Delta_2)$ car pendant un intervalle de temps plus grand, il est logique que l'on puisse recevoir au moins autant d'informations (de même pour α^l).

La super-additivité et la sous-additivité: l'exemple des contraintes implicites décrit précédemment montre les contraintes implicites dûes à la super-additivité. Il s'agit de régions inatteignables car on ne peut pas les atteindre sans violer une propriété de la courbe.

Dit autrement, si on coupe un intervalle Δ en deux intervalles Δ_1 et Δ_2, on peut par exemple obtenir une borne plus haute si on regarde la courbe basse : $\alpha^l(\Delta) \leq \alpha^l(\Delta_1) + \alpha^l(\Delta_2)$.

Cela révèle des régions inatteignables. Pour enlever ces régions, on utilise la super-additivité pour les courbes basses et la sous-additivité pour les courbes hautes.

Une courbe c croissante de $\mathcal{N}$ (ensemble des naturels) dans $\overline{\mathcal{Q}^+}$ (ensemble des rationnels union $+\infty$) passant par le point $(0,0)$ est dite sous-additive (resp. super-additive) si et seulement si $\forall (s,t) \in \mathcal{N}$, $c(t+s) \leq c(t) + c(s)$ (resp. $c(t+s) \geq c(t) + c(s)$).

Une droite appartenant à α^l (resp. α^u) est dite pertinente si sa pente est supérieure (resp. inférieure) à la pente de la courbe.

Une courbe est dite normalisée ssi elle est définie à l'aide d'un préfixe fini de points et d'un ensemble de droites pertinentes, elle vérifie la monotonie, le préfixe fini des courbes basses et hautes sont de même longueur et elle est super-additive pour la courbe basse

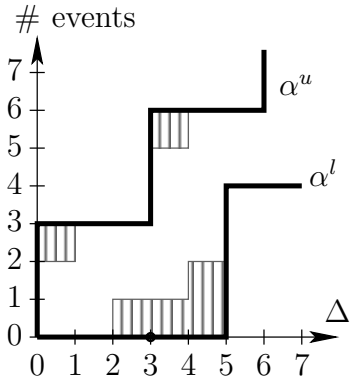

FIGURE 2: Courbe d'arrivée présentant des régions interdites

et sous-additive pour la courbe haute.

La causalité: prenons l'exemple de la figure 2. La figure 2 montre des zones appellées `zones interdites`. En effet, si une courbe cumulative arrive dans ces zones, alors elle ne peut plus les quitter sans être sûr de violer les propriétés de la courbe d'arrivée à un moment donné. Imaginons la courbe cumulative suivante : aucun événement n'arrive jusqu'à $t = 4$. Pour respecter la courbe basse, il faut au moins que 4 événements arrivent pour $t = 5$. Or d'après la courbe haute, il ne faut pas plus de 3 événements durant un intervalle de temps de taille 1. D'où un blocage, la courbe cumulative ne peut plus être prolongée.

Une courbe d'arrivée est dite causale si toute courbe cumulative vérifiant les propriétés de la courbe d'arrivée jusqu'à un temps T quelconque peut être prolongée jusqu'à un temps $T + \delta$ avec δ quelconque, tout en continuant à vérifiant les propriétés de la courbe d'arrivée.

Une caractérisation d'une courbe causale utilisant des opérateurs de déconvolution est donnée dans [2]. Soit l'opérateur de déconvolution classique $(min, +)$ (resp. (max, +)) $\oslash$ (resp. $\overline{\oslash}$) défini par : pour c et d des courbes croissantes de $\mathcal{N}$ dans $\overline{Q^+}$ passant par le point $(0, 0)$, pour $x \geq 0$, $(c \oslash d)(x) = \sup_{t \geq 0}\{c(x + t) - d(t)\}$ et $(c\overline{\oslash}d)(x) = \inf_{t \geq 0}\{c(x + t) - d(t)\}$. Alors (α^u, α^l) est une courbe causale si et seulement si α^u et α^l sont normalisées et $\alpha^l = \alpha^l \oslash \alpha^u$ et $\alpha^u = \alpha^u \overline{\oslash} \alpha^l$.

C. La fermeture causale dans ac2lus

Implémentation des courbes d'arrivée dans ac2lus: Dans ac2lus, comme décrit dans la section 5.2 *Piecewise Affine, Convex/Concave Curves* dans [3], les courbes d'arrivée sont sous la forme d'un ensemble fini de points suivi d'un ensemble fini de droites. Autrement dit les premiers points de la courbe haute (resp. basse) sont définies jusqu'à $P(\alpha^u)$ (resp. $P(\alpha^l)$) dans un tableau. Il y a ensuite $N(\alpha^u)$ (resp. $N(\alpha^l)$) droites décrivant la courbe haute (resp. la courbe basse) stockées sous la forme $(ax + b)/c$ ce qui permet de ne stocker que des entiers mais d'exprimer des rationnels.
De façon plus formelle, les courbes sont définies par :

- $p_i^u \in \mathcal{N}$, $i \in [0, P(\alpha^u)]$, décrivant l'ensemble fini des premiers points de la courbe haute
- $p_i^l \in \mathcal{N}$, $i \in [0, P(\alpha^l)]$, décrivant l'ensemble fini des premiers points de la courbe basse
- $a_j^u, b_j^u, c_j^u \in \mathcal{N}$, $j \in [0, N(\alpha^u)]$, décrivant les droites définissant la courbe haute
- $a_j^l, b_j^l, c_j^l \in \mathcal{N}$, $j \in [0, N(\alpha^l)]$, décrivant les droites définissant la courbe basse

On définit aussi pour $x \in \mathcal{N}$:

- $C^u(x) = \min_{j \in [0, N(\alpha^u)]}\{(a_j^u.x + b_j^u)/c_j^u\}$ si $\mathcal{N}(\alpha^u) > 0$, $+\infty$ sinon
- $C^l(x) = \max_{j \in [0, N(\alpha^l)]}\{(a_j^l.x + b_j^l)/c_j^l\}$ si $\mathcal{N}(\alpha^l) > 0$, 0 sinon
- $I^u(x) = p_x^u$ s'il existe, $+\infty$ sinon
- $I^l(x) = p_x^l$ s'il existe, 0 sinon

Les courbes ainsi représentées sont définies par :

- $\alpha^u(\Delta) = \min(C^u(\Delta), I^u(\Delta))$
- $\alpha^l(\Delta) = \max(C^l(\Delta), I^l(\Delta))$

En utilisant des droites pour prolonger la partie définie par des points, on permet de représenter des courbes infinies avec un nombre fini de paramètres.

Algorithme pour la normalisation: La normalisation se passe en trois temps :

- vérifier que les courbes sont croissantes.
- mettre le préfixe fini des courbes à la même taille.
- appliquer la sous-additivité/super-additivité aux courbes.

L'algorithme qui vérifie que les courbes sont croissantes et qui les rend croissantes si elles ne le sont pas totalement est simple. Il suffit de parcourir les courbes et de regarder si la valeur en $\Delta + 1$ est supérieur ou égale à celle en Δ. Sinon, $\alpha(\Delta+1) \leftarrow \alpha(\Delta)$.

Etendre la partie finie est plus compliqué. Tout d'abord il faut déterminer le point jusqu'auquel il faut étendre la partie finie constituée de points. Pour cela, nous allons définir la pente de la courbe :

- $S^p(\alpha^u) = \min_\Delta\{\alpha^u(\Delta)/\Delta\}$
- $S^p(\alpha^l) = \max_\Delta\{\alpha^l(\Delta)/\Delta\}$

Puis nous allons utiliser la propriété suivante[3] : la fermeture sous-additive de la courbe reste au-dessus de la droite $S^p(\alpha).\Delta$ et touche cette droite régulièrement, avec une période inférieure à la taille de la partie finie de la courbe.

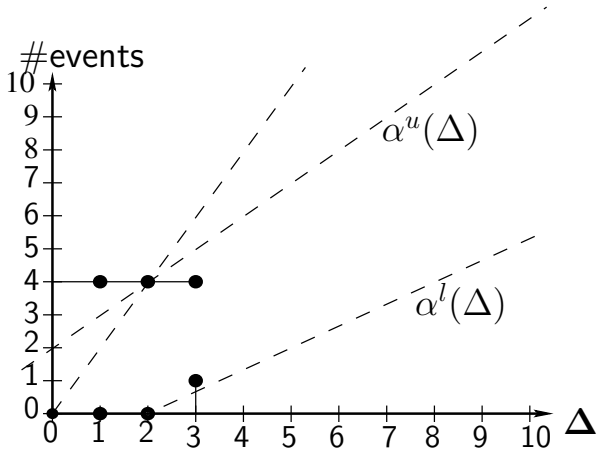

FIGURE 3: Courbe de départ

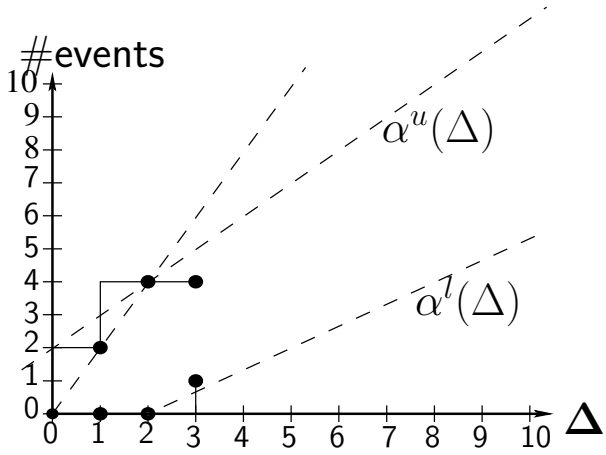

FIGURE 4: Courbe après première étape

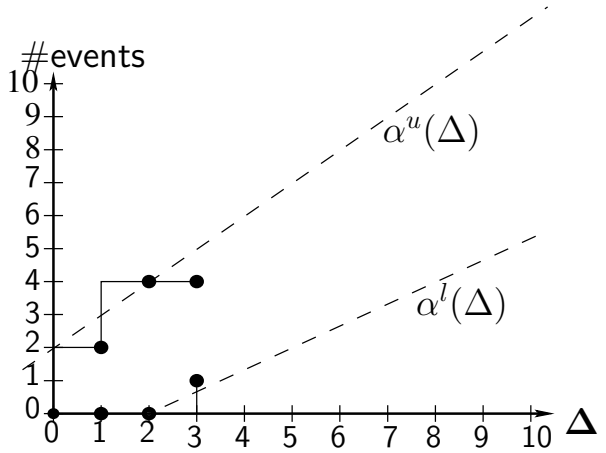

FIGURE 5: Suppression des droites inutiles

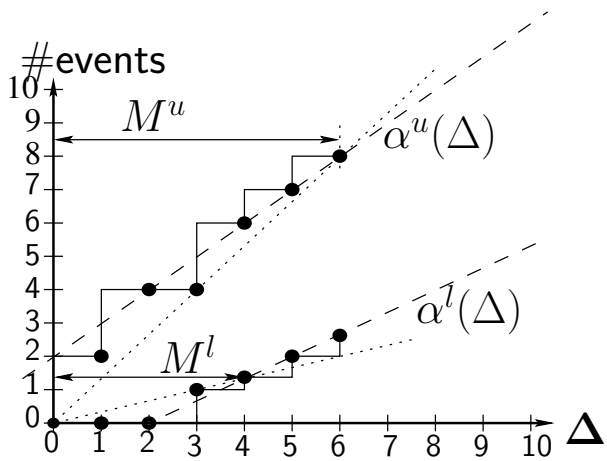

FIGURE 6: Prolongation jusqu'à $\max(M^u, M^l)$

De même la fermeture super-additive reste au-dessous de $S^p(\alpha).\Delta$ et touche cette droite avec une période inférieure à la taille de la partie finie de la courbe.

L'intersection entre $S^p(\alpha^u).\Delta$ et le $\min_j\{(a^u_j.\Delta + b^u_j)/c^u_j\}$ nous fournit la borne M^u (abscisse de l'intersection) pour α^u , et l'intersection entre $S^p(\alpha^l).\Delta$ et le $\max_j\{(a^l_j;\Delta + b^l_j)/c^l_j\}$ nous fourni la borne M^l pour α^l (cf figure 6). Nous avons besoin de ces deux bornes pour étendre nos parties finies jusqu'à un Δ suffisant pour que la partie finie contienne toutes les informations nécessaire pour la suite.

De plus on en déduit que les droites ayant une pente supérieur (resp. inférieur) à S^p pour la fermeture sous-additive (resp. super-additive) n'apportent pas d'information supplémentaire et peuvent donc être enlevées.

A la fin de l'algorithme, les courbes doivent se trouver dans une des situations suivantes comme indiqué dans [3] : $P(\alpha^u) = P(\alpha^l) = P$ et au moins l'une des conditions suivantes est vérifiée

- la courbe n'est pas réalisable, c'est-à-dire qu'il n'existe pas de courbe cumulative pouvant vérifier les propriétés de la courbe pour tout Δ.
- il n'y a pas de droites prolongeant la courbe et la courbe est sous-additive/super-additive jusqu'à P.
- α^u n'a pas de droites et est sous-additive jusqu'à P, et α^l est super-additive.
- α^l n'a pas de droites et est super-additive jusqu'à P, et α^u est sous-additive.
- α^u et α^l sont déjà sous-additive/super-additive.

A partir de là, l'algorithme pour normaliser les courbes expliqué dans [3] est le suivant :

1) S'assurer que tous les points sont au-dessus (resp. au-dessous) des droites définissant α^l (resp. α^u). Ajouter des points jusqu'à ce que $P(\alpha^u) = P(\alpha^l) = P$.

2) Eliminer les droites de α^l dont la pente est inférieure à $S^p(\alpha^l)$ et les droites de α^u dont la pente est supérieure à $S^p(\alpha^u)$;
3) Il y a 4 cas possibles selon que α^u et α^l possèdent ou ne possèdent pas de droites :
   a) S'il n'y a aucune droite :
      – Appliquer la fermeture sous-additive/super-additive jusqu'à P.
   b) Si les deux courbes ont des droites :
      – Calculer M^u et M^l. Soit $M = \max(M^u, M^l)$.
      – Ajouter des points au deux courbes jusqu'à ce que $P(\alpha^u) = P(\alpha^l) = M$.
      – Appliquer la fermeture sous-additive/super-additive jusqu'à M pour les deux courbes.
   c) Si α^u n'a pas de droites mais α^l en a au moins 1 :
      – Calculer M^l.
      – Ajouter des points aux deux courbes jusqu'à M^l.
      – Appliquer la fermeture sous-additive/super-additive jusqu'à M^l pour les deux courbes.
   d) Si α^l n'a pas de droites mais α^u en a au moins 1 : appliquer comme précédemment mais en inversant α^u et α^l.

Les figures 3, 4, 5 et 6 illustrent cet algorithme.

Concernant la sous-additivité/super-additivité, il s'agit simplement d'une double boucle. Par exemple pour la sous-additivité, l'algorithme est le suivant :

```
for(i=0 ; i<T ; i++) {
    for (j=0 ; j<i ; j++) {
        if (alpha(i) > alpha(j)+alpha(i-j)){
            alpha(i) = alpha(j)+alpha(i-j);
        }
    }
}
```

Implémentation de la normalisation de courbes définies avec des droites: Ma contribution principale pendant ce TER a été d'implémenter l'algorithme de normalisation décrit précédemment.
La fonction de normalisation se présente sous la forme suivante :

```
void arrival_curve::normalization() {
        monotonic_closure(false);

        // make sure that all points from the upper curve are under the affines pieces
        if (m_upper_curve.has_segments()) {
                m_upper_curve.compare_with_segments(0, m_upper_curve.size()-1);
        }
        // make sure that all points from the lower curve are above the affines pieces
        if (m_lower_curve.has_segments()) {
                m_lower_curve.compare_with_segments(0, m_lower_curve.size()-1);
        }
        // remove the useless affine pieces
        m_lower_curve.remove_useless_segments();
        m_upper_curve.remove_useless_segments();
        // extension of the finite prefix parts of the curves
        int M;
        if (m_upper_curve.has_segments() && m_lower_curve.has_segments()) {
                M = max(m_upper_curve.extension_limite(),
                        m_lower_curve.extension_limite());
        } else if (m_upper_curve.has_segments() && !(m_lower_curve.has_segments())) {
                M = m_upper_curve.extension_limite();
        } else if (!(m_upper_curve.has_segments()) && m_lower_curve.has_segments()) {
                M = m_lower_curve.extension_limite();
        } else {
                M = max(m_upper_curve.size(), m_lower_curve.size());
        }
        // extension of curves
        m_upper_curve.add_points(M);
        m_lower_curve.add_points(M);
        // remove the useless affine pieces
        m_lower_curve.remove_useless_segments();
        m_upper_curve.remove_useless_segments();
        // subadditive/superadditive closure
        subadditive_closure_to_rank(false , M+1);
//      return M;
}
```

Les points importants à expliquer concernant cette implémentation sont les suivants :

- factorisation de la fermeture sous-additive/super-additive :
  dans l'algorithme de base, vous avez pu constater que dans les quatre cas du point 3), il fallait appliquer la fermeture sous-additive/super-additive jusqu'à la limite M (ou P dans le cas 3.a). Par conséquent il fallait mieux factoriser l'appel à la fermeture sous/super-additive et utiliser une variable pour stocker cette limite.

- choix d'implémentation pour l'ajout de points :
  `add_points(M)` est la méthode servant à étendre une courbe jusqu'à M. Le principe de l'extension est le suivant : on recopie la partie finie déjà existante par "copier-coller". Si une droite passe en dessous (resp. au dessus) pour α^u (resp. α^l), on prend le point défini par la droite comme point de la courbe.
  Pour cela il est nécessaire de connaitre la courbe affine par morceaux décrite par les droites. Mon implémentation calcule au fur et à mesure de ses besoins les différents points de cette courbe. Pour cela elle maintient à jour un tableau contenant le point courant de chaque droite, un entier indiquant l'abscisse du point courant et un autre tableau contenant les points de la courbe affine.

- abscence d'égalisation des longueurs au début :
  selon l'algorithme présenté précédemment, il faut égaliser la taille de la partie finie avant de passer au point 2). Or dans mon implémentation, on ne le fait presque qu'à la fin quand on appelle `add_points`, ce qui correspond en réalité à l'égalisation de la taille demandée en 3). Nous allons montrer que cela n'a aucun effet sur l'algorithme.
  Tout d'abord, la liste des fonctions appelées entre l'emplacement théorique de l'égalisation de dimension et l'emplacement réel est la suivante :
  - `remove_useless_segments`
  - `extension_limite`

  `remove_useless_segments` a besoin uniquement de S^p et des droites définissant les courbes. De plus `extension_limite` permet de calculer M. Or pour effectuer ce calcul, nous avons uniquement besoin de connaitre S^p et les courbes utiles. Autrement dit, pour savoir si l'absence d'égalisation de la taille a une influence, il suffit de voir s'il influe sur le calcul de M et sur la supression de droites. Commençons par M.
  Prenons deux courbes α_1 et α_2. Posons T_1 la taille initiale de la partie finie de α_1, T_2 la taille de la partie finie de α_2 avec $T_2 \geq T_1$, S_i^p la pente de α_1 avant extension, S_e^p la pente de α_1 après extension. Selon l'algorithme, on devrait étendre α_1 jusqu'à T_2 mais dans l'implémentation cela n'est pas fait.
  Deux cas peuvent se produire :

  1) Soit aucune droite ne coupe la partie finie de α_1 jusqu'à T_2. Alors il est évident que la pente de la courbe de change pas : $S_i^p = S_e^p$. Donc le calcul de M ne change pas.
  2) Soit au moins une droite coupe la partie discrète de α_2. Deux nouveaux cas se présentent alors.
     a) Soit il existe un point d'abscisse a tel que : $a \in [0, T_2]$ et $\forall x \geq a, \alpha_1(x)$ est défini par un point d'une droite de α_1. Comme les courbes sont croissantes, $S_e^p = \alpha_1(T_2)/T_2 \neq S_i^p$. Mais cela n'est pas gênant pour M car dans notre implémentation `extension_limite` retournera a. Or $a \leq T_2$, donc on étendra α_1 jusqu'au $\max(T_2, \alpha_2.extension\_limite()) = M$. Dans l'algorithme initiale, `extension_limite` pour α_1 aurait retourné T_2. Or $T_2 \leq \max(T_2, \alpha_2.extension\_limite())$. Donc la valeur de M n'est pas changée par notre implémentation dans ce cas.
     b) Soit le point a défini comme tel n'existe pas.
        α_1 peut ne pas avoir de droite, comme $T_1 \leq T_2$, c'est α_2 qui va déterminer la valeur de M.
        Mais α_1 peut aussi avoir des droites. Dans ce cas le point a se trouve après T_2. Alors $S_i^p = S_e^p$ car l'extension consiste en une simple prologation de courbe par recopiage. Quelques points de la partie finie de la courbe ont pu être modifiés mais ce n'était pas des points faisant partie de la droite $S_i^p.\Delta$ car sinon les droites décrivant α_1 auraient généré un point a comme défini dans le cas précédent. $S_i^p = S_e^p$ donc M n'est pas modifié.

Nous venons de voir que le calcul de S^p n'était perturbé que dans le cas 2)a). C'est donc le seul cas où il peut y avoir une différence concernant la suppression des droites n'apportant pas d'informations supplémentaires.
Dans ce cas, comme $S_e^p \geq S_i^p$, l'algorithme de base risque d'enlever plus de droites que nous. Il s'agit en réalité des droites ayant un impact sur la courbe uniquement de 0 à T_2. On pourrait conserver toutes les droites le temps des calculs mais cela ralentirait l'algorithme. Nous avons choisi d'enlever une première fois les droites de α^l dont la pente est inférieure à $S_i^p(\alpha^l)$ et les droites de α^u dont la pente est supérieure à $S_i^p(\alpha^u)$ puis de

recommencer la suppression des droites après avoir fait l'extension complète des courbes.

Ainsi dans aucun cas, ce report concernant la prolongation de la courbe ne perturbe l'algorithme.

Comme indiqué plus tôt, l'intéret de normaliser une courbe d'arrivée décrite avec une partie finie et des droites, puis de la rendre causale, est d'avoir des courbes plus précises et adaptées à l'outil ac2lus.

Algorithme de causalité: la présence de partie affine ne change rien à l'algorithme comme montré dans [3]. Il suffit d'appliquer l'algorithme de causalification à la courbe d'arrivée jusqu'à M.
L'algorithme consiste à appliquer le produit de convolution à la courbe basse puis à la courbe haute jusqu'à ce qu'aucune des 2 courbes ne subissent plus de modification pendant une itération complète. On est alors tombé sur la courbe d'arrivée vérifiant $\alpha^l = \alpha^l \oslash \alpha^u$ et $\alpha^u = \alpha^u \overline{\oslash} \alpha^l$. La courbe est maintenant causale.

III. ac2lus et la causalité

A. Présentation de ac2lus

ac2lus est un outil prenant en entrée des courbes d'arrivée et calculant des courbes d'arrivée en sortie qui représentent le flux de sortie du système. Le principe du système est représenté figure 7. Ac2lus utilise des outils de vérification formelle : Nbac et kind pour l'instant mais il est possible d'insérer d'autres outils relativement facilement.

Figure 7: Principe d'ac2lus

ac2lus utilise des courbes d'arrivée avec un temps relatif, or Lustre est un langage synchrone utilisant un flux de données avec un temps absolu. Il faut donc adapter les courbes d'arrivée pour qu'elles puissent être utilisées par un programme écrit en Lustre. De même, il faut adapter la sortie du programme Lustre pour retrouver une courbe de sortie.

Plus précisemment, ac2lus utilise des observateurs pour faire l'interface. Si le flux d'entrée satisfait la courbe d'arrivée alors le premier observateur va émettre "OK". Dans ce cas, ac2lus va émettre une hypothèse sur un point de la courbe de sortie. Si ce point marche, alors le deuxième observateur va émettre "OK".
Les observateurs sont des noeuds Lustre qui sont inclus automatiquement par ac2lus dans le programme Lustre testé.

B. Présentation de l'exemple utilisé

J'avais pour but de trouver des exemples montrant l'impact de la causalité dans ac2lus. J'ai trouvée des exemples qui montrent que les courbes de sortie obtenues sont moins précises avec des courbes d'arrivée causales qu'avec des courbes d'arrivées n'ayant eu que la normalisation.

Je vais maintenant présenter l'exemple que j'ai mis en place. Le principe du système est expliqué figure 9. Les courbes mesurent ici une quantité de flux de donnée.

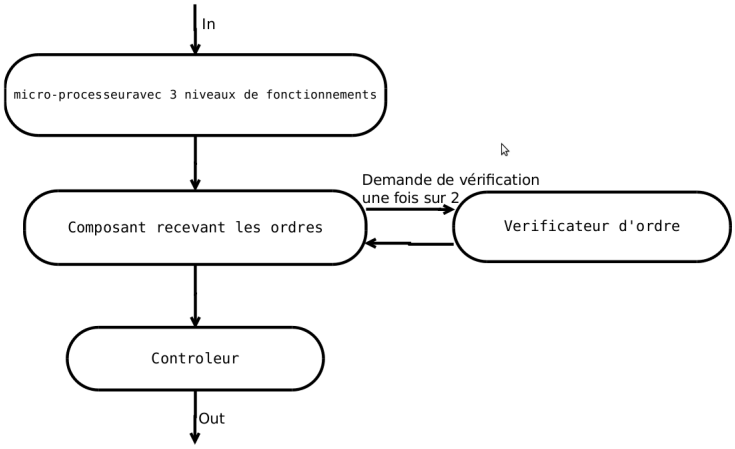

Figure 8: principe de fonctionnement du système servant d'exemple

Les points importants de cet exemple sont les suivants :

- le micro-contrôleur qui contient 3 états de fonctionnement selon son niveau saturation :
  1) ne fait rien
  2) faible puissance de calculs
  3) puissance maximale de calculs

  Un système comprenant des états ne peut pas être modélisé par le RTC seul. L'outil ac2lus arrive par contre à le faire.
- présence de niveaux de priorité différents au niveau du composant central. Les informations venant du composant appelé vérificateur d'ordre sont plus prioritaires que celles venant du micro-contrôleur.

Voici le code Lustre du micro-contrôleur pour montrer le système à plusieurs états :

```
node power_level(in_seq, level : int)
returns (out_seq, backlog : int)
    var empty_queue : bool;
    work : int;
    resource : int;
let
    resource = if (level = 0) then 0
```

```
                else if (level = 1) then 2
                     else 5;
    work = in_seq -> pre backlog + in_seq;
    empty_queue = work <= resource;
    out_seq = if (empty_queue) then work
              else resource;
    backlog = if (empty_queue) then 0
              else (work-out_seq);
tel


node micro_pro(in_seq : int)
returns (out_seq : int)
    var level, backlog : int;
let
    level = 0 ->
            if (((pre(level) = 0)
                 and (pre(backlog) < 3))
                or (pre(backlog) = 0))
            then 0
            else if ((pre(level) = 0)
                     and (pre(backlog) >= 3))
                    or ((pre(level) = 1)
                     and (pre(backlog) < 6))
                    or ((pre(level) = 2)
                     and (pre(backlog) <= 2))
                 then 1
                 else 2;
    out_seq, backlog = power_level(in_seq,
        level);
tel
```

C. *Courbe d'arrivée utilisée en entrée*

Sur cette exemple, j'ai utilisé plusieurs courbes d'arrivée non causales en entrée. Je présente ci-dessous 2 de ces courbes (figure 10 et figure 13) ainsi que les courbes normalisées et causales associées. La courbe haute est en rouge et la courbe basse en bleu.

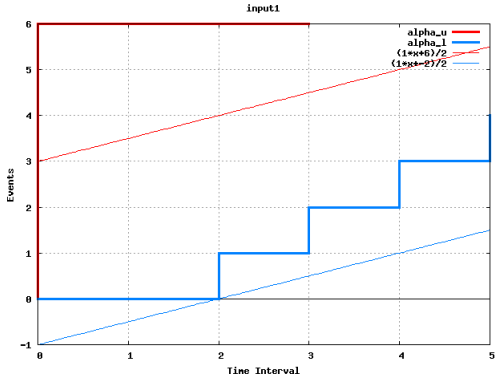

FIGURE 9: entrée 1

α^u : {0, 6, 6, 6}, et la droite (1x + 6)/2.
α^l : {0, 0, 1, 2, 3, 4}, et la droite (1x - 2)/2.

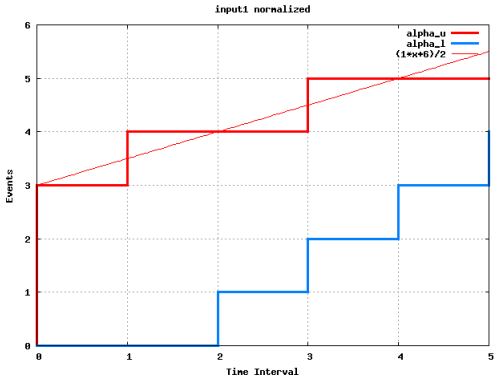

FIGURE 10: entrée 1 normalisée

α^u : {0, 3, 4, 4, 5, 5}, et la droite (1x + 6)/2.
α^l : {0, 0, 1, 2, 3, 4}.

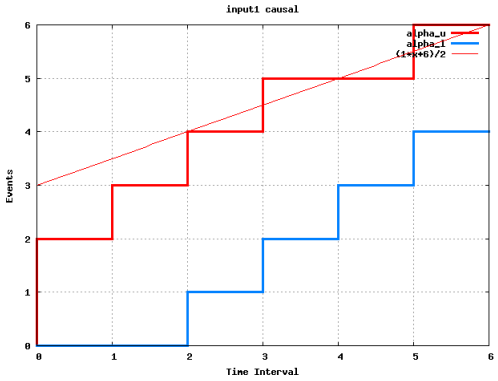

FIGURE 11: entrée 1 causale

α^u : {0, 2, 3, 4, 5, 5, 6}, et la droite (1x + 6)/2.
α^l : {0, 0, 1, 2, 3, 4, 4}.

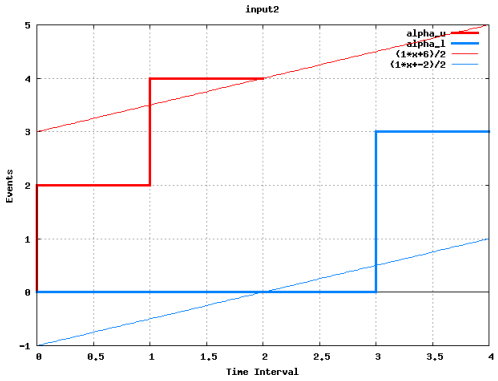

FIGURE 12: entrée 2

α^u : {0, 2, 4} et la droite (1x + 6)/2.
α^l : {0, 0, 0, 3, 3} et la droite (1x - 2)/2.

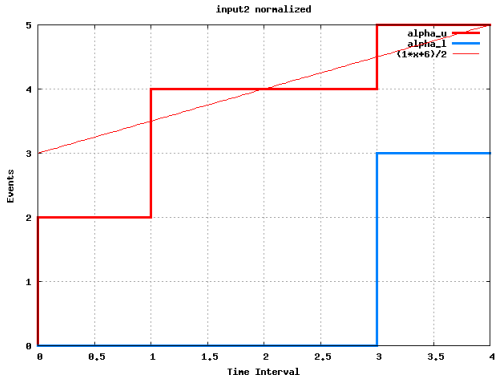

FIGURE 13: entrée 2 normalisée

α^u : {0, 2, 4, 4, 5} et la droite (1x + 6)/2.
α^l : {0, 0, 0, 3, 3}.

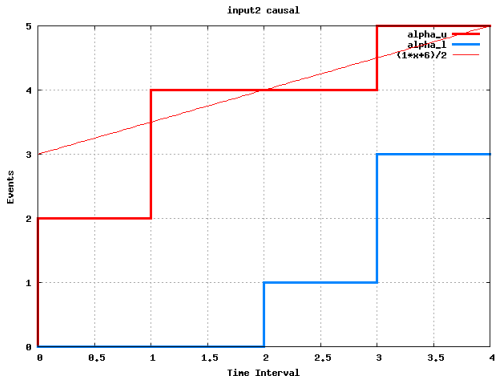

FIGURE 14: entrée 2 causale

α^u : {0, 2, 4, 4, 5} et la droite (1x + 6)/2.
α^l : {0, 0, 1, 3, 3}.

D. Présentation des résultats

– Voici les résultats obtenus avec ac2lus dans le cas de la courbe d'entrée numéro1 (entrée normalisée figure 16, entrée causale figure 17)

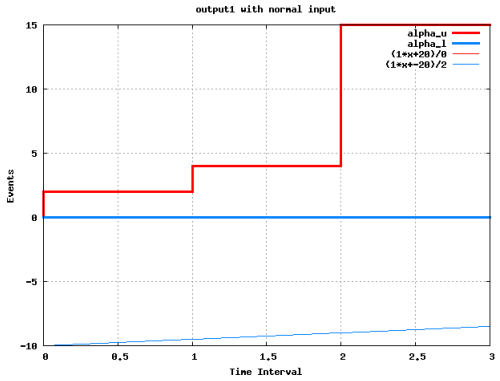

FIGURE 15: sortie 1, entrée normalisée

α^u : {0, 2, 4, 15}
α^l : {0, 0, 0, 0}.

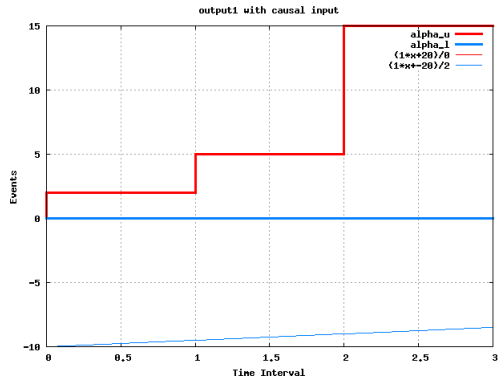

FIGURE 16: sortie 1, entrée causale

α^u : {0, 2, 5, 15} et la droite (1x + 6)/2.
α^l : {0, 0, 0, 0}.
Les points 1 et 2 se trouvent à 4 quand l'entrée est normalisée et à 5 quand l'entrée est causale.

– Voici les résultats obtenus avec ac2lus dans le cas de la courbe d'entrée numéro2 (entrée normalisée figure 18, entrée causale figure 19)

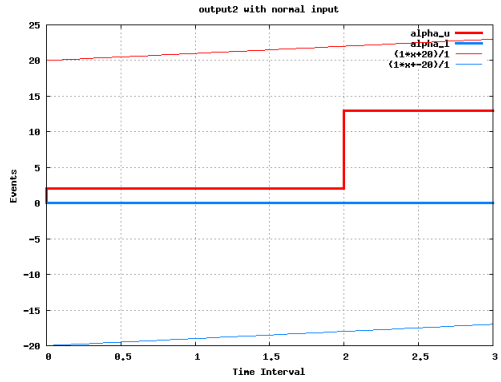

FIGURE 17: sortie 2, entrée normalisée

α^u : {0, 2, 2, 13}
α^l : {0, 0, 0, 0}.

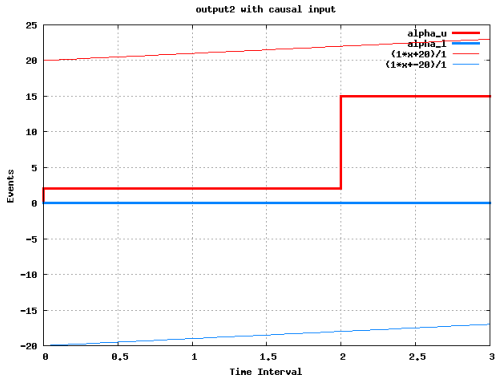

FIGURE 18: sortie 2, entrée causale

α^u : {0, 2, 2, 15}
α^l : {0, 0, 0, 0}.
Le point 3 se trouvent à 13 quand l'entrée est normale et à 15 quand l'entrée est causale.

E. Interprétation des résultats

On constate tout d'abord que les courbes de sortie sont moins précises dans le cas d'une entrée causale que dans le cas d'une entrée seulement normalisée.
ac2lus utilise des outils de preuves formelles qui nous indiquent si la propriété est vraie, fausse ou si l'outil ne peut pas savoir. ac2lus renvoie la meilleur valeur dont il est sûr qu'elle soit juste. Ainsi si on regarde les traces produites par ac2lus pour l'exemple 1, on constate qu'il arrive sur un timeout pour $\alpha^u(2) = 4$:

```
bisect: trying point alpha_u[2] = 4
...
Timeout (10 seconds) or killed! Failed to prove
    property.
```

Alors que $\alpha^u(2) = 5$ arrive à être prouvé comme étant un point correct.

```
bisect: trying point alpha_u[2] = 5
...
Property proved
Proof concluded by kind_runner
```

```
bisect: alpha_u[2]: found overapproximation 5
```

IV. Conclusion

Nous avons vu un outil appelé ac2lus qui utilise des courbes d'arrivée. Ces courbes d'arrivée ont certaines propriétés qui nous permettent de trouver des courbes équivalentes et plus précises. L'étude de ces courbes sous la forme d'un préfixe fini de points suivi par des droites nécessite une première étape qui est la normalisation, puis vient l'application de la causalité. Nous avons implémenté un algorithme permettant de normaliser des courbes d'arrivée définies avec des portions affines.

Une deuxième étape est la fermeture causale de ces courbes. Si cette fermeture n'est pas réalisée, on pourrait tomber sur des deadlock en théorie. Mais nous avons constaté qu'en pratique on peut avoir des courbes de sortie plus précises en n'utilisant que la normalisation, sans appliquer ensuite la fermeture causale.

Un exemple où l'abscence de fermeture causale entraîne un deadlock ou une précision moindre de la courbe de sortie reste à trouver.

Références

[1] Karine Altisen and Matthieu Moy. ac2lus : Bringing SMT-solving and abstract interpretation techniques to real-time calculus through the synchronous language Lustre. In *22nd Euromicro Conference on Real-Time Systems (ECRTS)*, Brussels, Belgium, Jully 2010.

[2] Karine Altisen and Matthieu Moy. Arrival curves for real-time calculus : the causality problem and its solutions. In *TACAS*, March 2010.

[3] Matthieu Moy and Karine Altisen. Arrival curves for real-time calculus : the causality problem and its solutions. Technical Report TR-2009-15, Verimag Research Report, 2009.

[4] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *ISCAS*, 2000.