Abstraction d'horloges dans les systèmes synchrones flot de données *

Louis Mandel & Florence Plateau

LRI, Université Paris-Sud 11
INRIA Saclay
{mandel,plateau}@lri.fr

Résumé

Les langages synchrones flot de données tels que Lustre manipulent des séquences infinies de données comme valeurs de base. Chaque flot est associé à une *horloge* qui définit les instants où sa valeur est présente. Cette horloge est une information de type et un système de types dédié, le calcul d'horloges, rejette statiquement les programmes qui ne peuvent pas être exécutés de manière synchrone. Dans les langages synchrones existants, cela revient à se demander si deux flots ont la même horloge et repose donc uniquement sur l'égalité d'horloges. Des travaux récents ont montré l'intérêt d'introduire une notion relâchée du synchronisme, où deux flots peuvent être composés dès qu'ils peuvent être synchronisés par l'introduction d'un buffer de taille bornée (comme c'est fait dans le modèle SDF d'Edward Lee). Techniquement, cela consiste à remplacer le typage par du sous-typage. Ce papier est une traduction et amélioration technique de [11] qui présente un moyen simple de mettre en oeuvre ce modèle relâché par l'utilisation d'horloges abstraites. Les valeurs abstraites représentent des ensembles d'horloges concrètes qui ne sont pas nécessairement périodiques. Cela permet de modéliser divers aspects des logiciels temps-réel embarqués, tels que la gigue bornée présente dans les systèmes vidéo, le temps d'exécution des processus temps réel et, plus généralement, la communication à travers des buffers de taille bornée. Nous présentons ici l'algèbre des horloges abstraites et leurs principales propriétés théoriques.

1. Introduction

Les langages synchrones flot de données tels que Lustre [3] ont été introduits dans les années 80 pour l'implémentation de systèmes temps-réel critiques. Depuis, ils ont été utilisés dans diverses applications industrielles telles que les commandes de vol dans les avions Airbus. Ils ont été créés dans l'objectif de construire un langage de programmation proche des modèles mathématiques utilisés dans les systèmes embarqués tels que les équations sur des flots de données ou la composition d'automates à états finis. Dans ces langages, le synchronisme a une justification très pratique : à un certain niveau d'observation, le temps est considéré comme une séquence de réactions instantanées du système à des événements extérieurs et quand les processus sont composés en parallèle, ils s'accordent sur ces réactions [4]. Cela coïncide aussi avec l'interprétation de Milner du synchronisme dans SCCS [19] ou avec le produit synchronisé d'automates [2]. Dans les langages flot de données comme Lustre, le synchronisme est celui de la théorie du contrôle et peut-être interprété comme une contrainte de typage sur les séquences : quand on combine deux séquences $(x_i)_{i\in D_1}$ et $(y_i)_{i\in D_2}$ dans $(x_i)_{i\in D_1} + (y_i)_{i\in D_2}$, leurs domaines de temps D_1 et D_2 doivent être compatibles. Cette vérification statique est réalisée par un système de types appelé *calcul d'horloges* [12, 1] qui impose que D_1 et D_2

*Ce travail a été partiellement soutenu par le projet de recherche INRIA Synchronics.

soient égaux. Une telle analyse n'est pas bornée aux langages synchrones et est d'un intérêt plus large. Par exemple, une analyse des horloges est faite dans les outils de modélisation comme SIMULINK [9] ou SCICOS [21]. Le calcul d'horloges consiste essentiellement à se demander si deux flots ont la même horloge. Pour cela, les informations d'horloges sont représentées par des types. Considérons le langage de types suivant (tiré de [12]) :

schémas d'horloges	σ	::=	$\forall\alpha.\ \forall X_1, ..., X_m.\ ct$
types d'horloges	ct	::=	$ct \to ct \mid ct * ct \mid ck \mid (X : ck)$
horloges	ck	::=	ck `on` $c \mid ck$ `on` ***not*** $c \mid \alpha$
séquences booléennes	c	::=	$X \mid n$ où n est un ensemble de noms dénombrable

Comme dans le système de types de Hindley-Milner [18], les types sont séparés en schémas d'horloges (σ) et types d'horloges (ct). Un type d'horloges est quantifié sur des variables d'horloges (α) ou des variables booléennes (X). Le type d'horloges de $(+)$ est $\forall\alpha.\ \alpha \times \alpha \to \alpha$, qui indique que si x et y sont deux flots de même horloge α alors $x + y$ a aussi l'horloge α. Exprimé différemment, l'addition attend deux arguments synchrones et produit un flot résultat sur la même horloge. Un autre exemple de primitive synchrone est le délai unitaire, qui décale son entrée. Si x et y sont deux séquences $x_1\, x_2\, x_3$ ... et $y_1\, y_2\, y_3$ ... alors x `fby` y représente $x_1\, y_1\, y_2\, y_3$ .... x et y doivent avoir la même horloge et x `fby` y est produit sur cette horloge. On peut donner à `fby` la signature d'horloges : $\forall\alpha.\ \alpha \times \alpha \to \alpha$.

Les choses deviennent plus intéressantes quand l'échantillonnage intervient. Deux structures de contrôle typiques sont les structures d'échantillonnage et de combinaison de flots :

`when`	:	$\forall\alpha.\ \forall X.\ \alpha \to (X : \alpha) \to \alpha$ `on` X
`merge`	:	$\forall\alpha.\ \forall X.\ (X : \alpha) \to \alpha$ `on` $X \to \alpha$ `on` ***not*** $X \to \alpha$

x `when` y est bien typé pour le calcul d'horloges si x et y ont la même horloge α. Dans ce cas, le résultat a une horloge plus lente correspondant aux instants où y est vrai et on écrit cela α `on` y (la meta-variable X est substituée par la valeur effective y). Par exemple, si *half* est une séquence booléenne 10101010101... (c'est à dire l'expression régulière $(10)^\omega$, notée par la suite (10)) et x est une séquence $x_1\, x_2\, x_3$ ... alors x `when` *half* est un échantillonnage de x de fréquence un demi, c'est à dire $x_1\, x_3\, x_5$ .... Si x a un type d'horloges ck, alors x `when` *half* a le type d'horloges ck `on` *half*. L'expression $x + (x$ `when` *half*$)$, qui calculerait la séquence $(x_i + x_{2i})_{i\in\mathbb{N}}$ n'est pas bien typée puisque ck n'est pas égal à ck `on` *half*, et elle est statiquement rejetée par le compilateur. L'opérateur `merge` est symétrique : il attend une horloge, une séquence sur cette horloge et une séquence sur le complémentaire de cette horloge et les combine pour construire une séquence plus longue.

Pour comparer les horloges, la plupart des implémentations s'en tiennent à l'égalité des noms : ck `on` n_1 et ck `on` n_2 peuvent être unifiées seulement si $n_1 = n_2$. Cette restriction forte est raisonnable quand n_1 est le résultat d'un calcul complexe (pour lequel l'égalité est non décidable). Elle est néanmoins trop restrictive pour un ensemble important d'applications où les horloges sont périodiques, car elle ne permet pas de tenir compte des propriétés de ces horloges. En particulier, des travaux récents ont montré l'intérêt d'un modèle relâché de synchronisme, permettant de composer des flots dès qu'ils peuvent être synchronisés par insertion de buffers bornés. Ce modèle est appelé modèle de Kahn n-synchrone [10] et poursuit les travaux fondateurs de Edward Lee sur les *Synchronous Data-Flow graphs* (SDF) [17, 8].

Du point de vue du typage, le n-synchronisme revient à rajouter une règle de sous-typage au système de types qu'est le calcul d'horloges :

$$\text{(SUB)}\ \frac{H \vdash e : ck \quad ck <: ck'}{H \vdash e : ck'}$$

Intuitivement, $ck <: ck'$ assure qu'un flot produit sur l'horloge ck peut être consommée sur ck' par insertion d'un buffer borné. Dans le cas particulier des *horloges ultimement périodiques* [25],

le sous-typage est décidable. Les détails techniques sont rappelés en section 2.

Le point de vue adopté dans [11] est légèrement différent et plus simple que celui adopté dans [10]. Au lieu de se concentrer sur les horloges périodiques, on donne la possibilité de raisonner sur des ensembles d'horloges comme des abstractions [13] des horloges concrètes. En effet, dans certaines applications, les horloges exactement périodiques ne sont pas indispensables et il est suffisant de raisonner sur des intervalles d'horloges dont les bornes sont néanmoins périodiques. C'est typiquement le cas dans quatre genres d'applications nécessitant la communication par buffers (1) les applications avec gigue bornée, (2) les applications utilisant des horloges dont la taille du motif périodique rend rédhibitoire les calculs exacts, (3) les applications nécessitant la modélisation du temps d'exécution des processus temps-réel (4) les applications où les noeuds de calcul manipulent plusieurs valeurs d'un même flot au sein d'un instant logique.

Par exemple, un flot x qui est présent en moyenne 3 fois sur 7 par rapport à une horloge de base ck et est soumis à une potentielle gigue de 4 instants aura le type d'horloges $\exists n \in [-2,2](7/3).\ ck\ \texttt{on}\ n$. Le quantificateur existentiel cache l'instant exact où l'élément est présent mais en donne une borne. Intuitivement, la notation $[-2,2](7/3)$ représente toutes les horloges dont le $j^{\text{ème}}$ `1` se trouve de deux instants avant à deux instants après le $j^{\text{ème}}$ `1` d'une « horloge parfaite » dont les occurrences de `1` seraient séparés par exactement sept tiers d'instant. Si f est de la forme $\lambda x.x\ \texttt{when}\ e$ pour une certaine expression booléenne compliquée e mais dont on peut prouver qu'elle appartient à l'ensemble des valeurs représentées par $[-2,2](7/3)$, alors un type d'horloges valide pour f est $\forall\alpha.\ \alpha \to \alpha\ \texttt{on}\ n$ avec $n \in [-2,2](7/3)$.

Le concept et une mise en oeuvre de l'abstraction d'horloges ont été présentés dans [11]. Cet article reprend en français la structure du précédent, en le résumant et en introduisant une nouvelle abstraction plus simple et plus précise.

Après la définition formelle des mots binaires infinis et de leur propriétés utiles au modèle n-synchrone (section 2), nous présenterons la nouvelle version de l'abstraction et ses propriétés algébriques (section 3). Nous discuterons ensuite de ses avantages par rapport à la précédente (section 4), avant de présenter des applications du mécanisme d'abstraction d'horloges (section 5).

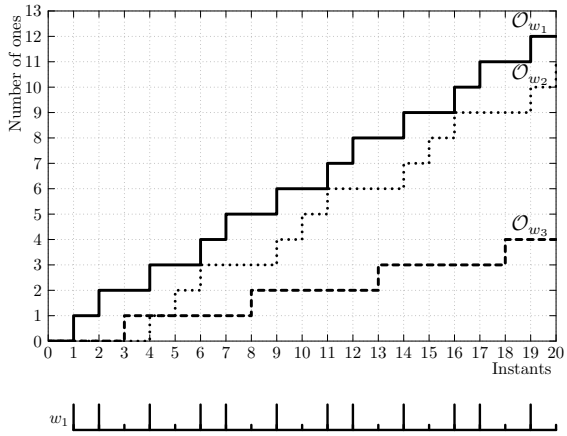
Un grand nombre de propositions énoncées dans ce papier ont été prouvées en Coq [6]. La formalisation et la preuve en Coq ont été un support à la réflexion, et nous ont surtout permis d'essayer plusieurs mécanismes d'abstraction différents. La modification de l'abstraction nécessite l'adaptation des preuves. Cette activité source d'erreurs à la main était dans notre cas automatiquement vérifiée par le compilateur Coq. Cela nous a permis d'avoir une plus grande confiance en les différents mécanismes essayés, et en particulier en celui que nous avons retenu.

Une version longue de l'article ainsi que les développements Coq sont disponibles à l'adresse http://www.lri.fr/~plateau/jfla09. Les ✿ pointent vers le code Coq correspondant à la définition ou la propriété énoncée.

2. Horloges et mots binaires infinis

Une relation de sous-typage peut-être vérifiée seulement si les types d'horloges sont exprimés par rapport à la même variable d'horloge. Intuitivement, cela tient au fait que l'échantillonnage est toujours relatif à une horloge ($ck\ \texttt{on}\ c$ est relatif à ck). Dans ce cas la relation de sous-typage correspond à une relation sur les séquences booléennes $\alpha\ \texttt{on}\ c <: \alpha\ \texttt{on}\ c' \Leftrightarrow c <: c'$.

Dans cette section, nous présentons les mots binaires infinis et une opération booléenne on sur celles-ci, telle que $(ck\ \texttt{on}\ c_1)\ \texttt{on}\ c_2 = ck\ \texttt{on}\ (c_1\ \mathit{on}\ c_2)$. Nous présentons ensuite la relation de sous-typage sur ces mots. Comme nous manipulons principalement la partie « flot booléen » (c) du type d'horloges, nous l'appellerons aussi horloge.



Les chronogrammes représentent les fonctions $\mathcal{O}_w$, qui associent à chaque instant i le nombre de 1 vus dans w depuis l'instant 1. Un front montant à l'instant i signifie que l'élément à l'indice i de w est un 1 (*i.e.* $w[i] = 1$). Si la fonction ne change pas de valeur à l'instant i, alors l'élément à l'indice i de w est un 0 (*i.e.* $w[i] = 0$).
Le chronogramme à une dimension situé sous le graphique représente le mot w_1, c'est une projection de $\mathcal{O}_{w_1}$.

FIG. 1 – Chronogrammes représentant les mots $w_1 = (11010)$, $w_2 = 0(00111)$, $w_3 = (00100)$.

2.1. Définitions

Une horloge peut-être un mot binaire infini ou une composition de ceux-ci. Si on identifie les noms à leurs valeurs, les horloges ont la grammaire suivante :

$$\begin{array}{lcll} c & ::= & w \mid \textbf{\textit{not}}\ w \mid c\ \textbf{\textit{on}}\ c & \text{✿} \\ w & ::= & 0.w \mid 1.w & \text{✿} \end{array}$$

$\textbf{\textit{not}}\ w$ est la négation de w, $c_1\ \textbf{\textit{on}}\ c_2$ est l'horloge échantillonnée et w un mot binaire infini. Dans les mots binaires infinis, un 1 dénote la présence d'une valeur sur le flot et un 0 l'absence de valeur.

La concaténation d'un mot binaire fini u et d'un mot binaire v est notée $u.v$. On notera 0^n la concaténation de n valeurs 0, 1^n la concaténation de n valeurs 1. On appellera *mots binaires infinis ultimement périodiques*, ou plus simplement *mots binaires périodiques* et on notera $u(v)$, les mots constitués d'un préfixe fini u suivi de la répétition infinie d'un mot fini v. L'élément à l'indice i de w est noté $w[i]$ (le premier élément du mot est à l'indice 1). On définit la fonction $\mathcal{O}_w(i)$ qui donne le nombre de 1 apparaissant dans w jusqu'à l'indice i :

Définition 1 (nombre de 1 vus dans w jusqu'à l'indice i : $\mathcal{O}_w$)**.** ✿

$$\begin{array}{lcl} \mathcal{O}_w(0) & = & 0 \\ \forall i \geq 1,\ \mathcal{O}_w(i) & = & \left\{ \begin{array}{ll} \mathcal{O}_w(i-1) & si\ w[i] = 0 \\ \mathcal{O}_w(i-1) + 1 & si\ w[i] = 1 \end{array} \right. \end{array}$$

Par convention, le nombre de 1 vus dans w avant l'instant 1 vaut 0.

La fonction $\mathcal{O}_w$ est une fonction discrète qui ne varie que par fronts montants de hauteur 1 ($\forall i \geq 0,\ 0 \leq \mathcal{O}_w(i+1) - \mathcal{O}_w(i) \leq 1$). ✿

Notons que l'on peut toujours reconstruire un mot w à partir de sa fonction $\mathcal{O}_w$. ✿ ✿

La figure 1 montre des exemples de mots binaires infinis, représentés par les chronogrammes de leur fonction $\mathcal{O}_w$. Si un flot est produit (resp. consommé) sur l'horloge w, alors une valeur est produite (resp. consommée) à chaque front montant du chronogramme.

Définissons maintenant l'opérateur d'échantillonnage. Si un flot x de type d'horloges α `on` w_1 est échantillonné avec l'horloge w_2, alors le flot résultant $x' = x$ `when` w_2 sera de type d'horloges (α `on` w_1) `on` $w_2 = \alpha$ `on` (w_1 $\textbf{\textit{on}}$ w_2). Si x est présent, alors w_2 est consulté pour savoir si la valeur doit être conservée. Si x est absent, alors x' est lui aussi absent et w_2 n'est pas consulté.

Définition 2 (opérateur on).

$$\begin{array}{lcl} 1.w_1 \ \mathit{on}\ 1.w_2 & = & 1.(w_1 \ \mathit{on}\ w_2) \\ 1.w_1 \ \mathit{on}\ 0.w_2 & = & 0.(w_1 \ \mathit{on}\ w_2) \\ 0.w_1 \ \mathit{on}\ w_2 & = & 0.(w_1 \ \mathit{on}\ w_2) \end{array}$$

Par exemple (10) on (10) est égal à (1000).

Les éléments de w_1 on w_2 correspondent aux éléments de w_2, quand w_2 est parcouru au rythme des 1 dans w_1. Donc on sait que si le nombre de 1 vus dans w_1 à l'indice i est j, alors le nombre de 1 vus dans w_1 on w_2 à l'indice i est égal au nombre de 1 vus dans w_2 à l'indice j :

$$\forall i \geq 0,\ \mathcal{O}_{w_1 \ \mathit{on}\ w_2}(i) = \mathcal{O}_{w_2}(\mathcal{O}_{w_1}(i))$$

Cette formulation de l'opérateur on est très utile pour les preuves. Par exemple, il est alors simple de prouver que l'opérateur on est associatif : $(w_1 \ \mathit{on}\ w_2) \ \mathit{on}\ w_3 = w_1 \ \mathit{on}\ (w_2 \ \mathit{on}\ w_3)$.

2.2. Taille de buffer

La taille de buffer minimale pour transmettre un flot produit sur une sortie de type d'horloges α `on` w vers une entrée de type d'horloges α `on` w' est la différence maximale entre le nombre de valeurs produites et le nombre de valeurs consommées au cours de l'exécution :

$$size(w, w') = \max_{i \in \mathbb{N}}(\mathcal{O}_w(i) - \mathcal{O}_{w'}(i))$$

Notons que si le minimum de cette différence est négatif, alors au moins une lecture dans le buffer se produira à un instant où celui-ci est vide. En effet, quand une valeur négative est atteinte, moins de valeurs ont été produites que consommées.

Dans les graphiques, la taille de buffer nécessaire pour communiquer de α `on` w vers α `on` w' est égale à la plus grande différence entre les courbes $\mathcal{O}_w$ et $\mathcal{O}_{w'}$. Par exemple, dans la figure 1, le nombre maximum de données produites et pas encore consommées durant la communication de α `on` w_1 vers α `on` w_2 est 2, atteint pour la première fois à l'instant 2. En revanche, le nombre de données à stocker durant la communication de α `on` w_1 vers α `on` w_3 augmente à l'infini.

2.3. Relation de sous-typage

La relation de sous-typage α `on` $w_1 <: \alpha$ `on` w_2 est vérifiée si et seulement si un flot produit sur α `on` w_1 peut être consommé sur α `on` w_2 par insertion d'un buffer de taille bornée.

Pour cela, il faut que les données soient toujours produites avant d'être attendues (relation de *précédence*), et que le nombre de valeurs présentes dans le buffer au cours de l'exécution soit borné (relation de *synchronisabilité*).

Définition 3 (précédence $\preceq$).
On dit que w_1 précède w_2 et on note $w_1 \preceq w_2$ si et seulement si $\forall i \geq 0,\ \mathcal{O}_{w_1}(i) \geq \mathcal{O}_{w_2}(i)$.

Un mot w_1 précède un mot w_2 si à tout instant il a produit autant ou plus de 1 que w_2. Cette relation est un ordre partiel. Elle permet de vérifier que la relation de causalité entre les flots est préservée, c'est à dire que le producteur écrit toujours ses sorties dans le buffer avant que le consommateur n'en aie besoin.

Sur la figure 1, $\mathcal{O}_{w_1}$ est toujours au dessus de $\mathcal{O}_{w_2}$ et $\mathcal{O}_{w_3}$, donc $w_1 \preceq w_2$ et $w_1 \preceq w_3$. Par contre, $\mathcal{O}_{w_2}$ et $\mathcal{O}_{w_3}$ sont entrelacées, donc $w_2 \not\preceq w_3$ et $w_3 \not\preceq w_2$.

Le supremum $\sqcup$ et l'infimum $\sqcap$ d'un ensemble de mots binaires infinis $W = \{w_1, \ldots, w_n\}$ pour la relation de précédence sont définis par :

$$\forall i \geq 0,\ \mathcal{O}_{\sqcup W}(i) = \min_{w \in W}(\mathcal{O}_w(i))$$
$$\forall i \geq 0,\ \mathcal{O}_{\sqcap W}(i) = \max_{w \in W}(\mathcal{O}_w(i))$$

Sur la figure 1, $w_2 \sqcup w_3$ est égal à w_2 jusqu'à l'instant 4, et ensuite égal à w_3. Les fronts montants du supremum sont situés à l'indice maximal des fronts montants des mots considérés. Pour tout $w \in W$, $\sqcap W \preceq w \preceq \sqcup W$.

Définition 4 (synchronisabilité $\bowtie$)**.** *On dit que w_1 et w_2 sont synchronisables et on note $w_1 \bowtie w_2$ si et seulement si $\exists b_1, b_2 \in \mathbb{Z},\ \forall i \geq 0,\ b_1 \leq \mathcal{O}_{w_1}(i) - \mathcal{O}_{w_2}(i) \leq b_2$.*

Deux mots w_1 et w_2 sont synchronisables si la différence entre le nombre de 1 vus dans w_1 et le nombre de 1 vus dans w_2 est bornée. [1] Cette relation est une relation d'équivalence.

Elle permet de vérifier qu'il existe une taille correcte pour le buffer, c'est à dire telle qu'il n'y aura jamais de dépassement de capacité.

Par exemple, sur la figure 1, $\mathcal{O}_{w_1}$ et $\mathcal{O}_{w_2}$ restent à une distance verticale bornée l'une de l'autre, donc $w_1 \bowtie w_2$ ($\forall i \geq 0,\ 0 \leq \mathcal{O}_{w_1}(i) - \mathcal{O}_{w_2}(i) \leq 2$). Un buffer de taille 2 permettra une communication sans perte. En revanche, la courbe de w_3 est de plus en plus basse par rapport à celles de w_1 et w_2, donc $w_3 \not\bowtie w_1$ et $w_3 \not\bowtie w_2$ ($\forall i \geq 0,\ -\infty \leq \mathcal{O}_{w_3}(i) - \mathcal{O}_{w_2}(i) \leq 1$ et $-\infty \leq \mathcal{O}_{w_3}(i) - \mathcal{O}_{w_1}(i) \leq 0$).

Définition 5 (sous-typage $<:$)**.** *On dit que w_1 est un sous-type de w_2 et on note $w_1 <: w_2$ si et seulement si $w_1 \preceq w_2$ et $w_1 \bowtie w_2$.*

En effet, si les données sont toujours écrites avant d'être lues, et si le nombre de données en attente de lecture ne croit pas à l'infini, alors la communication peut-être rendue synchrone par l'insertion d'un buffer de taille bornée. Par exemple, dans la figure 1, $w_1 <: w_2$.

Toutes les définitions de cette section peuvent être étendues aux horloges composées (c) par calcul des opérateurs ***not*** et ***on***.

3. Abstraction des horloges

L'abstraction d'un mot binaire infini w que nous proposons ici consiste à conserver uniquement le taux asymptotique r de 1 dans w et deux valeurs qui donnent les décalages minimum et maximum du nombre de 1 vus dans w par rapport au taux r. On note cette valeur abstraite $\left(b^0, b^1, r\right)$.

Les horloges abstraites peuvent être une valeur abstraite ou une composition de valeurs abstraites. Elles sont définies par la grammaire suivante :[2]

$$\begin{array}{lcl} ac & ::= & a \mid \mathbf{not}^{\sim}\ a \mid ac\ \mathbf{on}^{\sim}\ ac \\ a & ::= & \left(b^0, b^1, r\right) \text{ avec } b^0, b^1, r \in \mathbb{Q}, 0 \leq r \leq 1 \end{array}$$

Dans cette section, nous allons décrire l'ensemble d'horloges représenté par une valeur abstraite et nous montrerons que cet ensemble est reconnaissable par un automate fini. Les opérateurs $\mathbf{on}^{\sim}$ et $\mathbf{not}^{\sim}$ sont ensuite définis. Ils permettent de réduire de manière efficace une horloge abstraite en une valeur abstraite. Finalement, nous montrerons que les relations présentées en section 2 peuvent être facilement vérifiées sur une abstraction, et qu'une taille de buffer correcte peut être calculée de manière efficace.

[1] Cette définition de la relation de synchronisabilité est moins restrictive que celle de [10], mais correspond exactement à ce que l'on veut exprimer ici : si $w_1 \bowtie w_2$, la taille de buffer nécessaire pour communiquer de w_1 à w_2 ou de w_2 à w_1 est bornée.

[2] $\mathbb{Q}$ est l'ensemble des nombres rationnels.

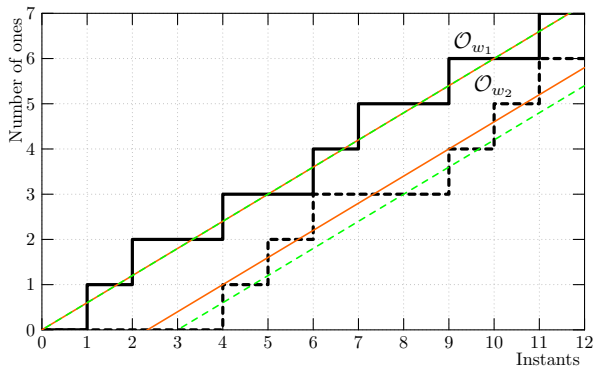


FIG. 2 – Si les fronts montants d'un mot w débutent tous sous la droite Δ^1 d'équation $r \times i + b^1$ et l'absence de front ne se produit qu'au dessus de la droite Δ^0 d'équation $r \times i + b^0$, alors w est dans l'abstraction $\left(b^0, b^1, r\right)$. Donc w_1 est dans $a_1 = \left(0, 0, \frac{3}{5}\right)$ et w_2 est dans $a_2 = \left(-\frac{9}{5}, -\frac{7}{5}, \frac{3}{5}\right)$.

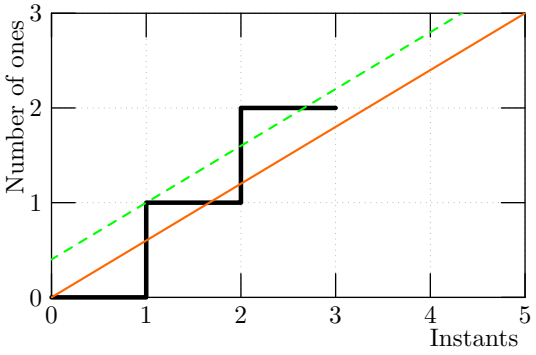


FIG. 3 – La concrétisation de $a_3 = \left(\frac{2}{5}, 0, \frac{3}{5}\right)$ est vide : il n'y a pas d'élément valide au troisième instant. En effet, la courbe se situe au dessus de la droite Δ^1, donc un front montant n'est pas autorisé, et en dessous de la droite Δ^0 donc une absence de front n'est pas autorisée non plus.

3.1. Abstraction des mots binaires infinis

Une valeur abstraite $\left(b^0, b^1, r\right)$ représente l'ensemble de mots binaires infinis suivant :

Définition 6 (concrétisation)**.**

$$concr\left(\left(b^0, b^1, r\right)\right) \stackrel{def}{\Leftrightarrow} \left\{ w,\ \forall i \geq 1,\quad \begin{array}{lcl} w[i] = 1 & \Rightarrow & \mathcal{O}_w(i-1) < r \times i + b^1 \\ \wedge & & \\ w[i] = 0 & \Rightarrow & \mathcal{O}_w(i-1) \geq r \times i + b^0 \end{array} \right\}$$

Un taux $r > 1$ représenterait des mots qui ont en moyenne plus qu'un 1 par instant, qui ne sont pas considérés ici (comme il est mentionné en section 2.1, $\mathcal{O}_w(i+1) - \mathcal{O}_w(i) \leq 1$).

Dans les chronogrammes, l'abstraction $\left(b^0, b^1, r\right)$ peut être représentée par deux droites Δ^0 et Δ^1 qui bornent les fronts montants et les absences de fronts des mots qu'elle contient. Tout front montant doit démarrer sous la droite Δ^1 (représentée par une ligne rouge continue) et toute absence de front démarre au dessus de la droite Δ^0 (représentée une ligne verte en pointillés). Les équations de ces droites sont $\Delta^1 :\ r \times i + b^1$ et $\Delta^0 :\ r \times i + b^0$. Par exemple, dans la figure 2 le mot $w_2 = 0(00111)$ est abstrait par $a_2 = \left(-\frac{9}{5}, -\frac{7}{5}, \frac{3}{5}\right)$. Le mot w_1 est abstrait par a_1 dont les droites Δ^0 et Δ^1 sont confondues.

On peut remarquer que tout mot restant à une distance bornée de son taux asymptotique peut être abstrait.

On définit $early_a$ comme le plus petit mot binaire infini (au sens de $\preceq$) respectant la propriété sur la position des 1 dans les mots abstraits par a. De même, $late_a$ est le plus grand mot binaire infini respectant la propriété sur la place des 0.

Définition 7 ($early_a$, $late_a$)**.** *Soit $a = \left(b^0, b^1, r\right)$ une valeur abstraite.*

$$\begin{array}{rcl} early_a & = & \sqcap\{w,\ \forall i \geq 1,\ w[i] = 1 \Rightarrow \mathcal{O}_w(i-1) < r \times i + b^1\} \\ late_a & = & \sqcup\{w,\ \forall i \geq 1,\ w[i] = 0 \Rightarrow \mathcal{O}_w(i-1) \geq r \times i + b^0\} \end{array}$$

Tous les mots w appartenant à la concrétisation de a sont tels que $early_a \preceq w$ et $w \preceq late_a$. Par conséquent, si la concrétisation de a est non vide, alors $early_a \preceq late_a$.

À l'inverse, si $early_a \preceq late_a$, alors $early_a$ et $late_a$ appartiennent à la concrétisation de a. Cela implique que la concrétisation est non vide, et que $early_a$ en est l'infimum et $late_a$ le supremum :

Proposition 1. $early_a \preceq late_a \Rightarrow (concr(a) \neq \varnothing) \wedge (early_a = \sqcap a) \wedge (late_a = \sqcup a)$

Par exemple, dans la figure 2, $w'_2 = \mathtt{00}(\mathtt{10110})$ est l'infimum de la concrétisation de a_2, les fronts montants sont effectués dès que possible, et $w''_2 = \mathtt{00}(\mathtt{01101})$ en est le supremum, l'absence de front est effectuée tant que possible (*i.e.* les fronts montants sont effectués aussi tard que possible). Il est montré dans la section 3.2, que ces deux horloges particulières peuvent être efficacement calculées par un programme synchrone.

Par conséquent, l'ensemble de concrétisation est vide si et seulement si le plus petit mot respectant la contrainte sur les $\mathtt{1}$ ne précède pas le plus grand mot vérifiant la contrainte sur les $\mathtt{0}$ (l'ensemble des mots vérifiant les conditions à la fois sur les $\mathtt{0}$ et les $\mathtt{1}$ est alors vide) : $concr(a) = \varnothing \Leftrightarrow early_a \not\preceq late_a$.

Le nombre de $\mathtt{1}$ vus dans $early_a$ et $late_a$ sont définis par les formules suivantes : [3]

$$\begin{array}{lrcl} \forall i, & \mathcal{O}_{early_a}(i) & = & \max(0, \min(i, \lceil r \times i + b^{\mathtt{1}} \rceil)) \\ & \mathcal{O}_{late_a}(i) & = & \max(0, \min(i, \lceil r \times i + b^{\mathtt{0}} \rceil)) \end{array}$$

Celles-ci sont utilisées de manière déterminante dans les preuves.

La proposition suivante donne une condition suffisante pour assurer que l'ensemble de concrétisation d'une valeur abstraite a est non vide :

Proposition 2 (test de non vacuité)**.** $\forall a = \left(b^{\mathtt{0}}, b^{\mathtt{1}}, r\right),\ b^{\mathtt{0}} \leq b^{\mathtt{1}} \Rightarrow concr(a) \neq \varnothing.$

En effet, si la courbe est toujours strictement sous $\Delta^{\mathtt{1}}$ et/ou au dessus ou sur $\Delta^{\mathtt{0}}$, alors il existe toujours un élément valide à l'index i. C'est le cas quand $\Delta^{\mathtt{1}}$ est au dessus de ou égale à $\Delta^{\mathtt{0}}$, c'est à dire quand $b^{\mathtt{1}} \geq b^{\mathtt{0}}$. Au contraire, dans la figure 3, la concrétisation de $a_3 = \left(\frac{2}{5}, 0, \frac{3}{5}\right)$ est vide : il n'y a pas d'élément valide au troisième instant car la courbe est au dessus de $\Delta^{\mathtt{1}}$ et en dessous de $\Delta^{\mathtt{0}}$.

La concrétisation contient un et un seul élément si $b^{\mathtt{0}} = b^{\mathtt{1}}$. C'est le cas dans la figure 2 où w_1 est l'unique élément de la concrétisation de a_1. En effet, la courbe est toujours soit en dessous de $\Delta^{\mathtt{1}}$, soit sur ou au dessus de $\Delta^{\mathtt{0}}$ (mais pas les deux). Il n'y a alors à chaque indice qu'un seul élément valide.

Cette proposition n'est pas une équivalence. En effet, $\Delta^{\mathtt{1}}$ peut être légèrement au dessous de $\Delta^{\mathtt{0}}$ si la fonction $\mathcal{O}_w$ ne peut jamais prendre une valeur située entre les deux droites.

Nous avons un ordre partiel sur les horloges abstraites :

Définition 8 (relation d'ordre $\sqsubseteq^{\sim}$)**.** $ac_1 \sqsubseteq^{\sim} ac_2 \stackrel{def}{\Leftrightarrow} concr(ac_1) \subseteq concr(ac_2)$

Cette relation d'ordre peut être testée de manière efficace :

Proposition 3 (test $\sqsubseteq^{\sim}$)**.** *Soient* $a_1 = \left({b^{\mathtt{0}}}_1, {b^{\mathtt{1}}}_1, r_1\right)$ *et* $a_2 = \left({b^{\mathtt{0}}}_2, {b^{\mathtt{1}}}_2, r_2\right)$ *deux valeurs abstraites telles que* $concr(a_1) \neq \varnothing$ *et* $concr(a_2) \neq \varnothing$*. Alors*

$$(r_1 = r_2)\ \textit{et}\ ({b^{\mathtt{0}}}_1 \geq {b^{\mathtt{0}}}_2)\ \textit{et}\ ({b^{\mathtt{1}}}_1 \leq {b^{\mathtt{1}}}_2) \Rightarrow (a_1 \sqsubseteq^{\sim} a_2)$$

Nous noterons $abs(w)$ toute fonction telle que $abs(w) = a \Rightarrow w \in concr(a)$. [4]

[3] $\lceil x \rceil$ est la partie entière supérieure de x.

[4] Traditionnellement [13], les fonctions d'abstraction sont notées α et les fonctions de concrétisation γ.

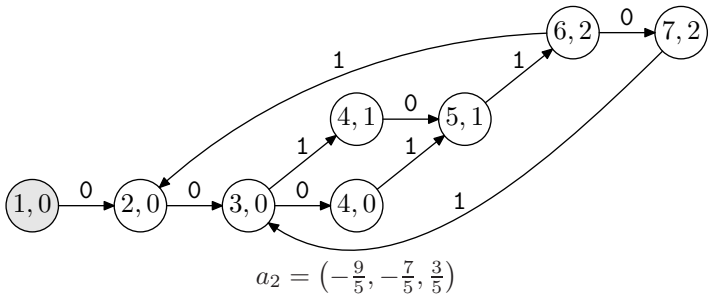


FIG. 4 – Automates reconnaissant les mots représentés par a_2.

3.2. Abstractions et automates

Nous avons vu que la concrétisation d'une horloge abstraite est un ensemble de mots binaires qui peut être vide, contenir un seul mot ou une infinité de mots. Cet ensemble peut être représenté par un automate fini déterministe qui reconnaît tous les mots de l'ensemble et seulement eux. Nous commençons par définir un automate infini tel que le langage reconnu soit l'ensemble de concrétisation. Puis, nous montrons qu'il est équivalent à un automate fini.

Définition 9 (automate associé à une valeur abstraite)**.**
Soit une valeur abstraite $a = \left(b^0, b^1, r\right)$*. L'automate infini associé à* a *est* $I_a = \langle Q, \Sigma, \delta, q_o \rangle$ *avec :*

- *l'ensemble d'états* Q *est un ensemble de paires* $(i, j) \in \mathbb{N}^2$ *;*
- *l'état initial* q_o *est* $(1, 0)$ *;*
- *l'alphabet* Σ *est* $\{0, 1\}$ *;*
- *la fonction de transition* δ *est définie par :*
  $\delta(0, (i, j)) = (i + 1, j)$ *si* $j \geq r \times i + b^0$
  $\delta(1, (i, j)) = (i + 1, j + 1)$ *si* $j < r \times i + b^1$
  la fonction est non définie sinon.

Les étiquettes des états correspondent aux coordonnées dans les chronogrammes. La valeur i est l'indice de l'instant courant, et j est le nombre de $1s$ vus avant l'instant courant ($\mathcal{O}_w(i-1)$). Une transition de l'état (i, j) à l'état $(i + 1, j + 1)$ correspond à un front montant dans la courbe $\mathcal{O}_w$, *i.e.* l'occurrence d'un 1 à l'indice i. Cette transition peut être prise si le point (i, j) est sous la droite d'équation $r \times i + b^1$. Une transition de (i, j) à $(i+1, j)$ correspond à l'absence de front dans la courbe, *i.e.* l'occurrence d'un 0 à l'indice i. Elle est autorisée si le point (i, j) est au-dessus de ou sur la droite d'équation $r \times i + b^0$.

Pour se ramener à un automate fini, on peut remarquer que pour une valeur abstraite de taux $r = \frac{n}{l}$, tous les états de la forme $(i + x \times l, j + x \times n)$ avec $x \in \mathbb{N}$ sont équivalents. En effet, la condition $j + x \times n \geq r \times (i + x \times l) + b^0$ est équivalente à $j \geq r \times i + b^0$ et la condition $j + x \times n < r \times (i + x \times l) + b^1$ est équivalente à $j < r \times i + b^1$. Il y a donc un nombre fini d'états équivalents. La fonction de transition de l'automate fini déterministe A_a équivalent à l'automate infini I_a est :
$\delta(0, (i, j)) = nf(i + 1, j)$ si $j \geq r \times i + b^0$
$\delta(1, (i, j)) = nf(i + 1, j + 1)$ si $j < r \times i + b^1$
avec $r = \frac{n}{l}$, $nf(i, j) = (i - x \times l, j - x \times n)$, $x = \max\{x \in \mathbb{N}, (x \times l \leq i) \wedge (x \times n \leq j)\}$

Il est donc possible de représenter un ensemble de concrétisation infini par un automate fini. La figure 4 montre l'automate associé à $a_2 = \left(-\frac{9}{5}, -\frac{7}{5}, \frac{3}{5}\right)$ de la figure 2.

L'infimum de l'ensemble de concrétisation (selon $\preceq$) correspond au chemin dans l'automate où les transitions 1 sont prises prioritairement. Pour le supremum, ce sont les transitions 0 qui sont prises prioritairement. Pour une abstraction de taux $r = \frac{n}{l}$ tous les cycles sont de même longueur (l) et contiennent le même nombre de 1 (n). Choisir une transition 1 ne fait que retarder la transition 0 correspondante.

Ces automates sont intéressants car ils permettent de vérifier dynamiquement ou statiquement (avec du *model checking*) qu'une horloge est dans la concrétisation d'une valeur abstraite. Il est à remarquer qu'il n'est pas nécessaire de construire explicitement l'automate. Voici un exemple de programme LUCID SYNCHRONE [20] qui vérifie qu'une horloge `clk` est abstraite par la valeur $(\texttt{b0}, \texttt{b1}, \texttt{r})$.

```
type rat = { num: int; den: int; }
let norm { num = n; den = l; } i j = if i >= l && j >= n then (i - l, j - n) else (i, j)
let node check (b0, b1, r) clk = ok where
  rec i, j = (1,0) fby norm r (i+1) (if clk then j + 1 else j)
  and ok = if clk then (rat_of_int j) <: r *: (rat_of_int i) +: b1
           else (rat_of_int j) >=: r *: (rat_of_int i) +: b0
```

Les opérateurs suffixés par « : » (`+:`, `*:`, `<:`, etc.) sont les opérateurs sur les rationnels. La fonction `norm` calcule de façon incrémentale la forme normale de l'état `(i,j)` en utilisant le taux $\frac{\texttt{n}}{\texttt{l}}$. Le noeud `check` prend en entrée une abstraction et un flot de booléens `clk`. Il maintient la valeur courante de l'état. L'état est initialisé à la valeur `(1,0)`, puis à chaque instant, `i` est incrémenté, et `j` est incrémenté si l'horloge `clk` était vraie à l'instant précédent. La fonction de normalisation est appliquée à ce nouvel état. Puis, suivant la valeur courante de `clk`, on vérifie que l'on va faire une transition valide. En suivant le même principe, on peut également générer des horloges contenues dans une abstraction.

```
let node generate choice (b0, b1, r) = clk where
  rec i, j = (1,0) fby norm r (i+1) (if clk then j + 1 else j)
  and one = (rat_of_int j) <: (r *: (rat_of_int i) +: b1)
  and zero = (rat_of_int j) >=: (r *: (rat_of_int i) +: b0)
  and clk = choice one zero
```

Le noeud `generate` est paramétré par une fonction `choice` qui choisit quelle transition prendre lorsqu'une transition `1` et une transition `0` sont possibles. Ainsi, on peut par exemple définir les noeuds qui génèrent l'infimum et le supremum d'une valeur abstraite `a` :

```
let node early a = generate (fun x y -> x) a
let node late a = generate (fun x y -> not y) a
```

Il est aussi possible de définir une fonction de choix qui génère des horloges aléatoires dans `a`.

3.3. Opérateurs abstraits

Dans cette section, nous définissons les opérateurs sur les valeurs abstraites, correspondant aux opérateurs sur les mots définis en section 2. Calculer ces opérations dans le domaine abstrait se fait en temps et mémoire constants. Le résultat dans le domaine abstrait contient le résultat de l'opération dans le domaine concret.

Définition 10 (opérateur $\boldsymbol{on}^{\sim}$)**.** ✿ *Définissons un opérateur* $\boldsymbol{on}$ *abstrait, noté* $\boldsymbol{on}^{\sim}$ *:*

$$\left(b^0{}_1, b^1{}_1, r_1\right) \boldsymbol{on}^{\sim} \left(b^0{}_2, b^1{}_2, r_2\right) = \left(b^0{}_{12}, b^1{}_{12}, r_{12}\right)$$

avec : $r_{12} = r_1 \times r_2, \quad b^0{}_{12} = b^0{}_1 \times r_2 + b^0{}_2, \quad b^1{}_{12} = b^1{}_1 \times r_2 + b^1{}_2 + \delta$
et $b^0{}_1 \leq 0$, $b^0{}_2 \leq 0$, $r_1 = \frac{n_1}{l_1}$, $b^1{}_1 = \frac{k^1_1}{l_1}$, $r_2 = \frac{n_2}{l_2}$, $b^1{}_2 = \frac{k^1_2}{l_2}$. $\delta = \frac{l_1 - 1}{l_1} \times \frac{n_2 - 1}{l_2}$.

L'opérateur abstrait $\boldsymbol{on}^{\sim}$ ne s'applique que sur des valeurs abstraites a dont la borne b^0 est négative ou nulle. On peut toujours se ramener à ce cas en considérant l'horloge abstraite la plus précise a' dont la borne b^0 est négative ou nulle et telle que $a \sqsubseteq^{\sim} a'$. Si $a = \left(b^0, b^1, r\right)$ alors $a' = \left(\min(0, b^0), b^1, r\right)$. ✿
Cet opérateur abstrait est correct :
$\forall w_1 \in \mathit{concr}(a_1),\ \forall w_2 \in \mathit{concr}(a_2),\ w_1 \ \boldsymbol{on}\ w_2 \in \mathit{concr}(a_1 \ \boldsymbol{on}^{\sim}\ a_2)$. ✿

Définition 11 (opérateur $\texttt{not}^\sim$). *Définissons un opérateur* $\texttt{not}$ *abstrait, noté* $\texttt{not}^\sim$ *:*

$$\texttt{not}^\sim\ \big((b^0, b^1, r)\big) = \big(-b^1 - \varepsilon, -b^0 - \varepsilon, 1 - r\big)$$

avec $r = \frac{n}{l}, \quad b^0 = \frac{k^0}{l}, \quad b^1 = \frac{k^1}{l}, \quad \varepsilon = \frac{l-1}{l}$.

Cet opérateur abstrait est correct : $\forall w \in concr(a),\ \texttt{not}\ w \in concr(\texttt{not}^\sim\ a)$.

Étant donnée une fonction d'abstraction des mots ($abs(w)$), on peut calculer l'abstraction des horloges composées de ces mots. Elle est définie récursivement ainsi :

Définition 12 (fonction d'abstraction des horloges).

$$\begin{array}{lcl} abs(\texttt{not}\ w) & \overset{def}{\Leftrightarrow} & \texttt{not}^\sim\ abs(w) \\ abs(c_1\ \texttt{on}\ c_2) & \overset{def}{\Leftrightarrow} & abs(c_1)\ \texttt{on}^\sim\ abs(c_2) \end{array}$$

Cette abstraction des horloges est correcte : $c \in concr(abs(c))$.

3.4. Relations abstraites

Nous définissons maintenant les relations sur les valeurs abstraites correspondant aux relations sur les mots définies section 2. Une relation est vérifiée dans le domaine abstrait seulement si elle est vérifiée pour tout couple de mots dans les concrétisations respectives des abstractions considérées.

Définition 13 (synchronisabilité abstraite). *Définissons une relation* $\bowtie$ *abstraite, notée* $\bowtie^\sim$ *:*

$$ac_1 \bowtie^\sim ac_2 \overset{def}{\Leftrightarrow} \forall w_1 \in concr(ac_1), w_2 \in concr(ac_2),\ w_1 \bowtie w_2$$

Cette relation peut être vérifiée sur les valeurs abstraites :

Proposition 4 (test de synchronisabilité).
Soient $a_1 = \big({b^0}_1, {b^1}_1, r_1\big)$ *et* $a_2 = \big({b^0}_2, {b^1}_2, r_2\big)$ *telles que* $concr(a_1) \neq \varnothing$ *et* $concr(a_2) \neq \varnothing$. *On a :*

$$a_1 \bowtie^\sim a_2 \Leftrightarrow r_1 = r_2$$

En effet, si deux horloges restent à une distance bornée de leur taux asymptotique, alors le nombre de 1 de la première horloge reste à une distance bornée du nombre de 1 de la seconde si et seulement si leur taux sont égaux.

L'abstraction contient toutes les informations nécessaires pour vérifier la synchronisabilité de deux horloges sur leur abstraction : $abs(c_1) \bowtie^\sim abs(c_2) \Leftrightarrow c_1 \bowtie c_2$.

Définition 14 (précédence abstraite). *Définissons une relation* $\preceq$ *abstraite, notée* $\preceq^\sim$ *:*

$$ac_1 \preceq^\sim ac_2 \overset{def}{\Leftrightarrow} \forall w_1 \in concr(ac_1), w_2 \in concr(ac_2),\ w_1 \preceq w_2$$

La relation de précédence sur des valeurs abstraites de concrétisation non vide est vérifiée si et seulement si l'horloge la plus tardive dans la première abstraction précède l'horloge la plus précoce dans la seconde :

$$\begin{array}{rcl} a_1 \preceq^\sim a_2 & \Leftrightarrow & \sqcup(concr(a_1)) \preceq \sqcap(concr(a_2)) \\ & \Leftrightarrow & late_{a_1} \preceq early_{a_2} \end{array}$$

Quand les valeurs abstraites sont synchronisables, la relation de précédence abstraite peut être vérifiée par une condition suffisante.

Proposition 5 (test de précédence). *Soient* $a_1 = \big({b^0}_1, {b^1}_1, r\big)$ *et* $a_2 = \big({b^0}_2, {b^1}_2, r\big)$. *On a :*

$${b^0}_1 \geq {b^1}_2 \Rightarrow a_1 \preceq^\sim a_2$$

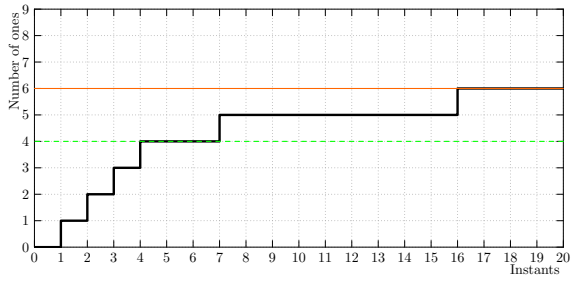


FIG. 5 – La valeur abstraite $(4, 6, 0)$ représente des mots qui contiennent entre quatre et six 1 et une infinité de 0.

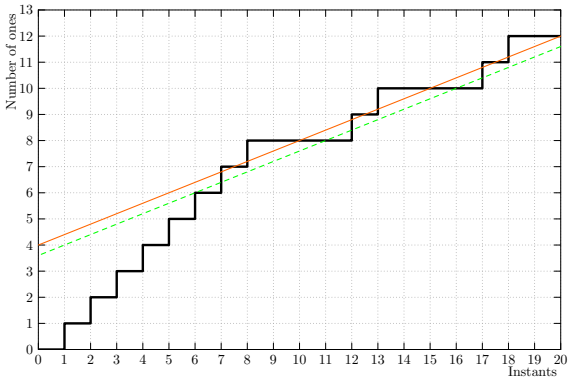


FIG. 6 – Tous les mots représentés par la valeur abstraite $\left(\frac{18}{5}, 4, \frac{2}{5}\right)$ débutent par sept 1.

Pour vérifier la relation de précédence entre des horloges sur leur abstraction, le manque d'informations à propos du nombre de 1 (qui est seulement encadré) force à considérer le pire cas de concrétisation. Cette vérification sur les abstractions est donc correcte, mais pas complète par rapport à la vérification sur les horloges concrètes : $abs(c_1) \preceq^{\sim} abs(c_2) \Rightarrow c_1 \preceq c_2$.

Nous définissons enfin la relation de sous-typage sur les valeurs abstraites :

Définition 15 (sous-typage abstrait)**.** *Définissons une relation* $<:$ *abstraite, notée* $<:^{\sim}$ *:*

$$ac_1 <:^{\sim} ac_2 \overset{def}{\Leftrightarrow} \forall w_1 \in concr(ac_1),\ w_2 \in concr(ac_2),\ w_1 <: w_2$$

Tout comme dans le domaine concret, cette relation est la conjonction des relations de synchronisabilité et de précédence : $a_1 <:^{\sim} a_2 \Leftrightarrow a_1 \bowtie^{\sim} a_2 \wedge a_1 \preceq^{\sim} a_2$.

Il est donc aussi possible de tester efficacement la relation de sous-typage en utilisant les tests de synchronisabilité et de précédence.

Pour synchroniser les producteurs et les consommateurs, des buffers sont insérés. Une taille de buffer correcte pour communiquer de n'importe quelle horloge d'abstraction $a_1 = \left({b^0}_1, {b^1}_1, r\right)$ vers n'importe quelle horloge d'abstraction $a_2 = \left({b^0}_2, {b^1}_2, r\right)$, telles que $a_1 <:^{\sim} a_2$ est :

$$size(a_1, a_2) = \lceil {b^1}_1 - {b^0}_2 \rceil$$

4. Discussion

L'abstraction présentée ici apporte plusieurs avantages par rapport à l'abstraction à base d'*enveloppes* de [11]. En effet, l'abstraction à base d'enveloppes raisonne sur les indices des 1, et ne contraint que ceux-ci. Par conséquent, il est d'une part impossible de modéliser les mots ne contenant qu'un nombre fini de 1, et d'autre part impossible de forcer la présence d'une suite de 1 au début du préfixe de tous les mots représentés par la valeur abstraite. Ces deux inconvénients mènent à un opérateur abstrait not non défini quand le résultat concret contient un nombre fini de 1 (par exemple $\mathit{not}\ 000011011111111 0(1) = 1111001000000001(0)$), et qui perd de l'information $(a \sqsubseteq^{\sim} \mathit{not}^{\sim}\ \mathit{not}^{\sim}\ a)$.

Ici, on raisonne sur le nombre de 1 vus dans le mot, et on permet ou non la présence d'un 1 et/ou d'un 0. D'une part, ceci donne la possibilité de représenter les horloges de taux asymptotique nul. Dans la figure 5, les horloges représentées par la valeur abstraite $(4, 6, 0)$ contiennent de quatre à six 1. Grâce à cela, l'opérateur $\mathit{not}^{\sim}$ peut-être appliqué aux mots de taux 1 (dont la négation est un mot de

taux nul). D'autre part, cela permet de représenter des ensembles d'horloges qui débutent toutes par une suite de 1. Dans la figure 6, tous les mots représentés par la valeur abstraite $\left(\frac{18}{5}, 4, \frac{2}{5}\right)$ débutent par sept 1. Grâce à cela, l'application de l'opérateur $\mathit{not}^{\sim}$ ne perd aucune information sur la valeur abstraite (*i.e.* $\mathit{not}^{\sim}\ \mathit{not}^{\sim}\ a = a$).

Notons que la restriction ($b^0 \leq 0$) appliquée aux opérandes de l'opérateur $\mathit{on}^{\sim}$ n'est pas nouvelle, elle était intrinsèque à l'abstraction par enveloppes (où l'équivalent de cette condition est toujours vérifié). Se ramener à des opérandes traitées par $\mathit{on}^{\sim}$ revient à se ramener à une opération similaire au $\mathit{on}^{\sim}$ sur les enveloppes. Le résultat obtenu ici n'est donc pas plus précis que le résultat obtenu dans l'abstraction à base d'enveloppes. Nous pourrions imaginer la réalisation d'un opérateur $\mathit{on}^{\sim}$ plus précis, néanmoins c'est une tâche difficile car le préfixe du résultat comporte plusieurs phases. Par exemple, $1111011111(100)\ \mathit{on}\ 111(10) = 1111001010(100000)$. Une première phase du préfixe est une suite de 1, suit ensuite une phase où le préfixe du premier mot est échantillonné au taux $\frac{1}{2}$ et enfin vient le suffixe de taux $\frac{1}{6}$.

Enfin, on peut remarquer que toute enveloppe de [11] peut-être traduite en une valeur abstraite de notre abstraction, sans perte d'information : $[d, D]\,(T)$ se traduit en $\left(-\frac{D}{T}, -\frac{d}{T} - \varepsilon, \frac{1}{T}\right)$ avec $T = \frac{l}{n}$, $\varepsilon = \frac{n-1}{l}$. La traduction inverse est impossible si le taux est nul, et perd de l'information si $b^0 \geq 0$ (il faut d'abord se ramener à un $b^0 \leq 0$).

5. Applications

5.1. Abstraction d'horloges périodiques

Pour être capables de vérifier la relation de sous-typage et calculer la taille des buffers, l'usage exclusif de mots binaires ultimement périodiques à été proposée dans [10]. [5]

Le comportement périodique de ces mots permet de calculer statiquement les opérateurs on et not (les définitions deviennent des algorithmes). De même, il permet de vérifier les relations de précédence (si elle est vérifiée jusqu'à un certain rang, alors elle l'est pour toujours) et de synchronisabilité (qui est équivalente à l'égalité entre les taux de 1 du comportement périodique). Enfin, la définition de la taille de buffer nécessaire devient aussi un algorithme.

Cependant, il peut être intéressant d'éviter les calculs exacts sur les mots binaires infinis à cause de leur coût : par exemple, l'opérateur on nécessite un parcours complet des éléments des périodes données en argument. De plus, si les tailles des opérandes ne sont pas compatibles, le résultat est bien plus long que les arguments. Dans certains contextes comme les applications vidéos, ce coût pose problème car les tailles de périodes peuvent être énormes : dans l'exemple cité dans [10], un downscaler classique, l'horloge de sortie a un comportement périodique de longueur 17280. Si on ajoute un traitement spécifique à effectuer entre les images, on obtient un comportement périodique de longueur 2073600 (la taille d'une image haute définition). Calculer sur les valeurs abstraites donne une solution à ce problème.

Notons que quand des horloges périodiques sont utilisées, l'abstraction des mots peut-être automatiquement calculée :

Proposition 6 (fonction d'abstraction des mots binaires périodiques)**.**
Soit $w = u(v)$ *un mot binaire infini.* $abs(w) = \left(b^0, b^1, r\right)$ *avec :*
$r = \frac{|v|_1}{|v|}$ *avec* $|v|$ *la taille du mot binaire fini* v *et* $|v|_1$ *le nombre de* 1 *apparaissant dans* v*,*
$b^0 = \min_i(\mathcal{O}_w(i-1) - r \times i)$ *avec* $i \in \{x$ *tels que* $(1 \leq x \leq |u| + |v|)$ *et* $w[x] = 0\}$*,*
$b^1 = \max_i(\mathcal{O}_w(i-1) - r \times i) + \frac{1}{|v|}$ *avec* $i \in \{x$ *tels que* $(1 \leq x \leq |u| + |v|)$ *et* $w[x] = 1\}$*.*

[5] Ces mots binaires infinis ont été utilisés depuis lors pour spécifier statiquement des ordonnancements périodiques pour le *Latency Insensitive Design* [7].

L'intuition derrière cette formule n'est autre que de parcourir le mot et d'élargir l'abstraction au fur et à mesure des besoins. Comme le mot est périodique, il suffit de parcourir le préfixe et une fois le suffixe pour obtenir une abstraction correcte.

Par exemple, l'abstraction de l'horloge de sortie du downscaler est :
$abs((\mathsf{10100100})\ \mathit{on}\ 0^{3600}(1)\ \mathit{on}\ (1^{720}0^{720}1^{720}0^{720}0^{720}1^{720}0^{720}0^{720}1^{720}))$
$= \left(0, 0, \frac{3}{8}\right)\ \mathit{on}^{\sim}\ (-3600, -3600, 1)\ \mathit{on}^{\sim}\ \left(-400, \frac{4312}{9}, \frac{4}{9}\right) = \left(-2000, -1120, \frac{1}{6}\right)$

On peut enfin s'intéresser aux horloges affines, présentées dans [23, 22], qui sont un sous-ensemble des horloges périodiques, de la forme $0^{\phi}(10^{l-1})$. Elles ont été utilisées pour étendre le calcul d'horloges du langage synchrone flot de données SIGNAL [5], dans le contexte du co-design logiciel/matériel. Grâce à la forme régulière de ces horloges, le calcul d'horloges étendu de SIGNAL offre un algorithme d'unification plus puissant. Dans le cas des horloges affines, abstraire un mot est très simple : $abs(0^{\phi}(10^{l-1})) = \left(-\frac{\phi}{l}, -\frac{\phi}{l}, \frac{1}{l}\right)$ et ne perd aucune information. En effet, l'ensemble de concrétisation ne contient qu'un élément (comme spécifié en section 3, les bornes sont égales).

5.2. Spécifications à l'aide d'horloges abstraites

Nous avons vu dans les exemples de la section 3 que l'abstraction d'horloges permet de modéliser des horloges soumises à une gigue bornée. Elle permet de raisonner non plus sur une horloge précise mais sur une spécification de cette horloge : son abstraction. Toutes les horloges répondant à cette spécification ont un taux asymptotique commun et s'éloignent de ce taux dans la limite imposée par des bornes elles aussi communes. Cela permet de prouver des propriétés (comme l'absence de famine et de dépassement de capacité) sur le programme quelle que soit l'horloge effective, du moment qu'elle répond à sa spécification.

L'abstraction permet aussi de modéliser le temps d'exécution de processus temps-réel [14, 24, 16]. Pour spécifier qu'une fonction f est exécutée un cycle sur dix, et que son exécution prend entre deux et quatre cycles, on peut lui donner la signature d'horloges : $f : \forall\alpha.\ \alpha\ \mathtt{on}^{\sim}\ \left(0, 0, \frac{1}{10}\right) \to \alpha\ \mathtt{on}^{\sim}\ \left(-\frac{4}{10}, -\frac{2}{10}, \frac{1}{10}\right)$. Si on la compose avec elle même, on obtient : $f \circ f : \forall\alpha.\ (\alpha\ \mathtt{on}^{\sim}\ \left(0, 0, \frac{1}{10}\right)) \to \alpha\ \mathtt{on}^{\sim}\ \left(-\frac{8}{10}, -\frac{4}{10}, \frac{1}{10}\right)$. L'exécution de f composée à elle même prend donc entre quatre et huit cycles.

Enfin, il est aussi possible de modéliser un trait courant des applications vidéo : la lecture ou l'écriture de plusieurs valeurs du même flot au sein d'un même instant logique [15]. Considérons, par exemple un noeud f produisant un flot de type d'horloges $\alpha\ \mathtt{on}^{\sim}\ (0, 0, 1)$ transmis à travers un buffer à un noeud g consommant un flot de type d'horloges $\alpha\ \mathtt{on}^{\sim}\ (-4, 0, 1)$. À chaque instant, f produit exactement une valeur. Quand à lui, g peut attendre quatre instants avant de commencer à consommer. Au cinquième instant, il peut alors lire les cinq premières valeurs, et continuer ensuite à consommer des « tableaux » de cinq valeurs un instant sur cinq. Cet exemple est illustré figure 7.

6. Conclusion

Cet article généralise la notion d'horloge dans les langages synchrones flot de données en permettant la manipulation d'ensembles d'horloges. Cette généralisation est basée sur l'introduction de valeurs abstraites qui permettent l'encadrement d'horloges. Nous nous sommes ici concentrés sur les propriétés algébriques de ces horloges et nous avons illustré leur expressivité. La motivation de ce travail est essentiellement pragmatique et donne une réponse à la nécessité de modéliser la gigue, le temps d'exécution et, plus généralement, la communication par buffers de taille bornée. La véritable nouveauté est de traiter des propriétés quantitatives lors du calcul d'horloges au lieu de se limiter à du synchronisme strict. Nous avons expérimenté l'usage de ces horloges sur plusieurs exemples (*picture-in-picture* vidéo et filtres de radio logicielle). L'extension du calcul d'horloges du compilateur de LUCID SYNCHRONE [20] est actuellement en cours.

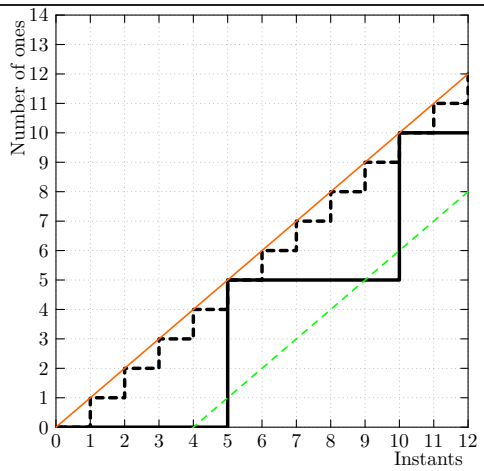


FIG. 7 – Modélisation de la lecture/écriture de plusieurs valeurs au sein du même instant

Remerciements. Pascal Raymond nous a montré une façon inattendue (et très élégante) d'utiliser le compilateur LUSTRE pour générer des horloges dans une abstraction. Gwenaël Delaval nous a fourni un exemple de radio logicielle. Sylvie Boldo, Stéphane Lescuyer et Matthieu Sozeau ont toujours été très disponibles pour nous faire découvrir COQ. Les figures ont été programmées en Mlpost, nous remercions Johannes Kanig et Stéphane Lescuyer pour leur aide. Nous remercions Marc Pouzet pour ses remarques constructives sur l'abstraction. Enfin, nous remercions les rapporteurs anonymes et le comité de programme.

Références

[1] T. Amagbegnon, L. Besnard, and P. Le Guernic. Implementation of the data-flow synchronous language signal. In *Programming Languages Design and Implementation*, pages 163–173. ACM, 1995.

[2] André Arnold. *Systèmes de transitions et sémantique des processus communicants*. Masson, 1992.

[3] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1), January 2003.

[4] Albert Benveniste and Gérard Berry. *The synchronous approach to reactive and real-time systems*, pages 147–159. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[5] Albert Benveniste, Paul Le Guernic, and Christian Jacquemot. Synchronous programming with events and relations : the SIGNAL language and its semantics. *Sci. Comput. Program.*, 16(2) :103–149, 1991.

[6] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development Coq'Art : The Calculus of Inductive Constructions*. Springer-Verlag. Series : Texts in Theoretical Computer Science. An EATCS Series, 2004.

[7] Julien Boucaron, Robert de Simone, and Jean-Vivien Millo. Formal methods for scheduling of latency-insensitive designs. *EURASIP Journal on Embedded Systems*, Issue 1(ISSN :1687-3955) :8 – 8, January 2007.

[8] J. Buck, S. Ha, E. Lee, and D. Messerschmitt. Ptolemy : A framework for simulating and prototyping heterogeneous systems. *International Journal of computer Simulation*, 1994. special issue on Simulation Software Development.

[9] P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis. Translating Discrete-Time Simulink to Lustre. *ACM Transactions on Embedded Computing Systems*, 2005. Special Issue on Embedded Software.

[10] Albert Cohen, Marc Duranton, Christine Eisenbeis, Claire Pagetti, Florence Plateau, and Marc Pouzet. *N*-Synchronous Kahn Networks : a Relaxed Model of Synchrony for Real-Time Systems. In *ACM International Conference on Principles of Programming Languages*, January 2006.

[11] Albert Cohen, Louis Mandel, Florence Plateau, and Marc Pouzet. Abstraction of Clocks in Synchronous Data-flow Systems. In *The Sixth ASIAN Symposium on Programming Languages and Systems (APLAS 08)*, Bangalore, India, December 2008.

[12] Jean-Louis Colaço and Marc Pouzet. Clocks as First Class Abstract Types. In *Third International Conference on Embedded Software*, Philadelphia, USA, October 2003.

[13] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[14] Adrian Curic. *Implementing Lustre Programs on Distributed Platforms with Real-time Constraints*. PhD thesis, Université Joseph Fourier, 2005.

[15] Léonard Gérard. Des horloges entières pour la répartition de programmes synchrones flot de données. Rapport de master, Université Paris-Sud 11, 2008.

[16] Nicolas Halbwachs and Louis Mandel. Simulation and verification of asynchronous systems by means of a synchronous model. In *Sixth International Conference on Application of Concurrency to System Design*, Turku, Finland, June 2006.

[17] E. Lee and D. Messerschmitt. Synchronous dataflow. *IEEE Trans. Comput.*, 75(9), 1987.

[18] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3) :348–375, 1978.

[19] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3) :267–310, 1983.

[20] Marc Pouzet. *Lucid Synchrone, version 3. Tutorial and reference manual*. Université Paris-Sud, LRI, April 2006. Distribution available at : www.lri.fr/~pouzet/lucid-synchrone.

[21] Scicos. http://www-rocq.inria.fr/scicos.

[22] Irina M. Smarandache, Thierry Gautier, and Paul Le Guernic. Validation of mixed SIGNAL-ALPHA real-time systems through affine calculus on clock synchronisation constraints. In *World Congress on Formal Methods (2)*, pages 1364–1383, 1999.

[23] Irina M. Smarandache and Paul Le Guernic. Affine transformations in SIGNAL and their application in the specification and validation of real-time systems. In *ARTS*, 1997.

[24] Christos Sofronis. *Embedded Code Generation from High-level Heterogeneous Components*. PhD thesis, Université Joseph Fourier, 2006.

[25] Jean Vuillemin. On Circuits and Numbers. Technical report, Digital, Paris Research Laboratory, 1993.