# Causalité dans les calculs d'événements

Bernard P Serpette, David Janin

# Causalité dans les calculs d'événements

Bernard P. Serpette[1] & David Janin[2]

*1: Université Côte d'Azur, Inria, France*
*2: LaBRI, Bordeaux INP, France*

**Résumé**

Si l'on considère un événement comme une valeur quelconque associée à une date de réception, un calcul d'événements est une fonction produisant un ensemble d'événements à partir d'un ensemble d'événements reçus. Un sous-ensemble particulièrement intéressant de ces fonctions correspond aux fonctions *causales* dont les événements produits avant une date donnée ne dépendent que des événements reçus avant cette même date. Dans cet article, nous proposons une définition très simple permettant de caractériser ces fonctions causales.

## 1. Introduction

Nous étudions ici la sémantique des programmes lisant, transformant et produisant des structures temporelles. L'élément essentiel d'une structure temporelle est l'événement : une valeur quelconque, spécifique au domaine que l'on manipule, à laquelle on attribue une date : quelque chose se passe à cette date. Une structure temporelle est alors simplement définie comme un ensemble d'événements.

La musique offre de nombreux exemples de structures temporelles. Les événements sont, par exemple, des débuts ou des fins de note. L'informatique musicale est donc largement utilisatrice de programmes manipulant des structures temporelles. On distingue cependant les transformations « hors temps », applicables, par exemple, lors de la composition d'une œuvre, des transformations « dans le temps », appliquées, à la volée, pendant l'exécution d'une œuvre. A l'évidence, les secondes forment une sous-classe des premières.

Prenons, par exemple, le programme réduisant par deux, dans une pièce musicale, la durée de chaque note de musique, sans pour autant déplacer le début de ces notes. Les blanches deviennent des noires, les noires des croches, etc.... Ce programme est applicable à une partition. La durée de chaque note étant connue, on peut la diviser par deux. Par contre, cette transformation n'est pas applicable à la volée ou autrement dit « dans le temps ». La durée d'une note n'est connue qu'à la fin de son écoute. Lorsque sa fin arrive, l'événement lié à la fin de la note correspondante aurait déjà dû être émis, cet événement fait déjà partie du passé. On dira que ce programme n'est pas causal.

La propriété de causalité (ou de non causalité) n'est cependant pas toujours si évidente à détecter. Prenons, par exemple, le programme qui ajoute la quinte [1] de toute note jouée, deux unités de temps après le début de cette note et en la jouant pendant la même durée, sous la condition qu'aucune autre note n'a été jouée pendant ces deux unités de temps. Bien que cette transformation soit spécifiée à l'aide de dépendances temporelles plutôt complexes entre les événements d'entrée et les événements de sortie, elle n'en est pas moins causale.

Autrement dit, un programme peut réaliser des calculs complexes sur les intervalles de temps écoulés entre deux événements, comme illustré par l'exemple des quintes retardées, tout en restant causal, c'est à dire exécutable à la volée, dans le temps. Pour un langage de programmation dédié à la conception de systèmes musicaux interactifs, tel que le T-calcul [2], l'étude et l'analyse de cette notion

1. un sol pour un do, un la pour un ré, un si pour un mi, etc...

de fonctions causales sont importantes. Elles permettraient, par exemple, via une analyse statique appropriée, de garantir que les programmes réalisés appartiennent (à une sous-classe décidable) des programmes effectivement exécutables dans le temps. Pour une étude plus approfondie on se réfèrera à [6].

Dans cet article nous donnerons une définition très simple de la causalité et nous prouverons que cette définition caractérise exactement les programmes pouvant émettre leurs sorties soit en réaction aux événements d'entrées, soit en réaction au temps qui passe.

### Notations

Dans toute la suite, **nil** représente la liste vide. Nous utiliserons la notation $x \cdot l$ pour ajouter un élément $x$ dans une liste $l$, et $l_1 \bullet l_2$ pour effectuer la concaténation de deux listes.

## 2. Temps, événements et chronologies

Le temps est central dans l'étude que nous allons aborder. La propriété la plus importante n'est pas que le temps soit mesurable, via une valeur numérique, mais qu'il soit comparable ; que l'on puisse parler d'*avant* et d'*après* une date donnée. Nous nous plaçons donc dans un domaine de temps $T$ muni d'une relation d'ordre totale.

Un événement est une valeur datée, c'est à dire une valeur quelconque estampillée par une date. Si l'on suppose un domaine de valeur $V$, un événement est un élément de $T \times V$. Nous noterons donc $(t, v)$ un événement construit sur une date $t \in T$ et une valeur $v \in V$.

Si l'on considère plusieurs événements, le domaine le plus naturel à considérer est celui des ensembles d'événements, c'est à dire le domaine $\mathcal{P}(T \times V)$. Une implémentation des multi-ensembles, quand les éléments possèdent un ordre total, peut être faite avec des listes triées. Cette implémentation est particulièrement agréable car l'identité ensembliste correspond à l'identité structurelle sur les listes. La différence ensemble versus multi-ensemble n'est pas ici primordiale, la vision multi-ensemble est plus générale en considérant la possibilité de la présence simultanée d'un même événement.

Voir un ensemble d'événements comme une liste triée selon la date des événements est assez naturel. Nous appellerons *chronologie* une telle structure. Une chronologie est donc une liste d'événements ordonnée par ordre décroissant de temps. Implicitement, la variable $c$ est une chronologie, élément de l'ensemble $C$ suivant :

**Définition 1 (Chronologie)** $C = \{l : List(T \times V) \mid sorted(l)\}$

Le prédicat *sorted* caractérise les listes triées par ordre décroissant de temps. La figure 1 donne

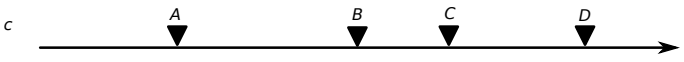


FIGURE 1 – Exemple d'une chronologie

l'exemple d'une chronologie. Le temps est représenté par la ligne horizontale et ce temps augmente de gauche à droite. Les événements sont décrits par des triangles pointant sur leurs valeurs de temps. Si les événements $A$, $B$, $C$ et $D$ sont arrivés dans cet ordre dans le temps, la chronologie $c$ les contenant est $[D\ C\ B\ A]$. La tête d'une chronologie contient le dernier événement arrivé. Cet ordre décroissant a été choisi pour faciliter les récurrences dans les preuves. L'arrivée d'un nouvel événement $E$ dans la chronologie $c$ se fait par $E \cdot c$ et non $c \bullet (E \cdot \mathbf{nil})$ si on avait choisi un ordre croissant.

L'intérêt que le temps soit comparable est, pour un certain temps $t$, de pouvoir couper une chronologie en deux, en séparant l'*après* et l'*avant* $t$. Dans une chronologie $c$, les événements antérieurs à un certain temps $t$, seront notés $c \downarrow t$, et sont définis par :

**Définition 2** *[Antériorité dans une chronologie]*

$c \downarrow t =$ **match** $c$ **with**
| **nil** $\Rightarrow$ **nil**
| $(t', v) \cdot c' \Rightarrow$ **if** $t' \geq t$
**then** $c' \downarrow t$
**else** $c$

Dans l'intention, et en supposant que les chronologies soient représentées par des ensembles, $c \downarrow t$ représente l'ensemble $\{(t', x) \in c \mid t' < t\}$. On remarquera que la définition 2 prend en compte le fait qu'une chronologie est une liste triée. De plus, on observe qu'un événement au temps $t$ n'est pas considéré antérieur à $t$.

De manière similaire à l'antériorité, les événements postérieurs à un certain temps $t$, dans la chronologie $c$, seront notés $c \uparrow t$, et sont définis par :

**Définition 3** *[postériorité dans une chronologie]*

$c \uparrow t =$ **match** $c$ **with**
| **nil** $\Rightarrow$ **nil**
| $(t', v) \cdot c' \Rightarrow$ **if** $t' \geq t$
**then** $(t', v) \cdot c' \uparrow t$
**else nil**

La figure 2 marque avec des triangles pleins l'antériorité d'une chronologie $c = [D\ C\ B\ A]$ pour le temps $t$, et avec des triangles vides sa postériorité. On a $c \uparrow t = [D\ C]$ et $c \downarrow t = [B\ A]$.

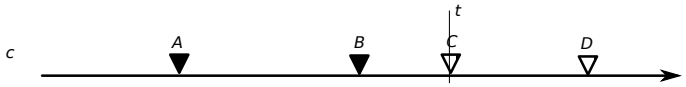


FIGURE 2 – Exemple d'antériorité et de posteriorité

## 3. Causalité

Une chronologie correspond à une donnée contenant des données temporelles. Nous allons nous intéresser aux calculs manipulant ce type de données et donc aux fonctions calculant des chronologies à partir de chronologies : c'est-à-dire les fonctions de type $C \to C$. La figure 3 donne la représentation du résultat de l'application d'une telle fonction sur une chronologie particulière. Dans cet exemple, on peut imaginer que la fonction $f$ translate sa chronologie d'entrée d'un certain quantum de temps.

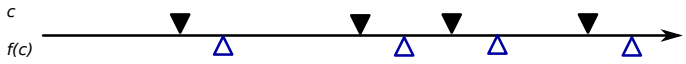


FIGURE 3 – Exemple d'une fonction sur des chronologies

Dans le cadre de la musique, une chronologie correspond à une partition, à un fichier MIDI (Musical Instrument Digital Interface), ou même à un fichier d'échantillons genre MIC (Modulation d'Impulsion Codée). Une fonction sur les chronologies est donc un programme créant un morceau de musique à

partir d'un autre morceau de musique. Ceci sans considération des meilleures manières de produire des données temporelles, ces fonctions se divisent en deux catégories, celles qui sont causales et celles qui ne le sont pas. Dans notre cadre, l'intuition de la causalité est : "*l'avenir ne peut pas influencer le passé*". Autrement dit une fonction est causale, si, pour tout temps $t$, elle peut produire les événements antérieurs à $t$ sans considérer les événements postérieurs à $t$. On voit que, dans ces deux définitions informelles, la causalité est définie en mentionnant, à la fois, le passé et le futur. L'originalité de la définition que nous proposons réside dans le fait qu'elle ne fait mention que du passé. Une fonction est causale si, pour deux chronologies partageant les mêmes événements avant un certain temps $t$, alors les images de ces deux chronologies par cette fonction doivent aussi avoir les mêmes événements avec ce temps $t$.

**Définition 4 (fonction causale)** $\forall f,\ causal(f) \Leftrightarrow \forall c_1\ c_2\ t,\ c_1 \downarrow t = c_2 \downarrow t \Rightarrow f(c_1) \downarrow t = f(c_2) \downarrow t$

L'intuition dérivant de cette définition est : "*ce qui est fait est fait*". Et derrière ces mots, si on considère que le passé est révolu pour une fonction causale, alors il y a sûrement une chance de calculer cette fonction au présent, ou, pour le moins, de façon incrémentale. Pour reprendre l'exemple de la musique et plus particulièrement de l'acoustique : la réverbération, l'écho et les filtres, entre autres, sont des fonctions qui transforment un signal sonore, et donc une chronologie si le signal est échantillonné. Ces transformations d'un signal sont causales car elles peuvent être directement appliquées sur le flux d'entrée. Par contre, la transformation qui consiste à inverser, dans le temps, un signal d'entrée, n'est pas causale. Cette transformation doit attendre la fin du flux d'entrée pour commencer.

Certaines fonctions ne sont pas immédiates à cerner concernant leur causalité. Prenons, par exemple, la fonction qui duplique les événements de son entrée d'un certain quantum de temps, disons $\Delta$, si et seulement si aucun autre événement n'est intervenu dans l'intervalle. Pour la chronologie [9 6 5 4 3 0], en omettant de mentionner les valeurs associées aux événements, et en prenant $\Delta = 2$, les événements 9, 6 et 0 seront dupliqués et le résultat de la transformation est [11 9 8 6 5 4 3 2 0]. Bien que regardant dans le futur pour prendre une décision, cette fonction est causale.

Etant donné $f$ une fonction sur les chronologies, pour exécuter $f$ de façon incrémentale sur une chronologie $c$, il faut pouvoir prendre les événements de $c$ par ordre croissant, donc dans l'ordre où ils arrivent dans le temps, les transmettre à $f$ et considérer, événement après événement, que les résultats sont révolus. Si on note $[\![f]\!]$, l'interprétation incrémentale de $f$, cette interprétation peut être définie de façon récursive (non terminale) de la manière suivante :

**Définition 5 (Calcul incrémental)**

$[\![f]\!] = \lambda c.$ **match** $c$ **with**
$\mid$ **nil**$\mid \Rightarrow f($**nil**$)$
$\mid (t, v) \cdot c' \Rightarrow (f(c) \uparrow t) \bullet ([\![f]\!](c') \downarrow t)$

Autrement dit, pour une chronologie dont le dernier élément dans le temps (tête de la chronologie) se situe au moment $t$, le comportement de $[\![f]\!]$ est le même que $f$ après ce temps. Le comportement de $[\![f]\!]$ avant ce temps pour cette chronologie est le comportement de $[\![f]\!]$ en considérant que ce dernier événement n'a pas lieu, et oubliant tous les événements produits qui sont postérieurs à $t$. On remarquera que la concaténation des deux listes, correspondant à l'union ensembliste, est bien fondée sachant que celle de gauche est triée et supérieure à $t$ et que la seconde est triée et inférieure à $t$. Calculons symboliquement $[\![f]\!](c)$ pour la chronologie $c = [D\ C\ B\ A]$.s Si, pour tout événement $X$, $t_X$ est le temps associé à $X$, on a $[\![f]\!](c) = (f(c) \uparrow t_D) \bullet ([\![f]\!]([C\ B\ A]) \downarrow t_D)$. Soit, en dépliant encore une fois, $[\![f]\!](c) = (f(c) \uparrow t_D) \bullet ((f([C\ B\ A]) \uparrow t_C) \bullet ([\![f]\!]([B\ A]) \downarrow t_C)) \downarrow t_D$. Si l'on continue à déplier et en simplifiant, on arrive à $[\![f]\!](c) = (f(c) \uparrow t_D) \bullet (f([C\ B\ A]) \uparrow t_C \downarrow t_D) \bullet (f([B\ A]) \uparrow t_B \downarrow t_C) \bullet (f([A]) \uparrow t_A \downarrow t_B) \bullet f(nil) \downarrow t_A$. On voit que $[\![f]\!](c)$ se calcule avec des tronçons de la forme $f(X) \uparrow t^1_X \downarrow t^2_X$ où $t^1_X$ est le plus grand temps de la chronologie $X$ et $t^2_X$ le temps du prochain événement qui adviendra après

$X$. La figure 4 montre ce calcul par tronçons. Les lignes successives montrent les diverses applications de $f(X)$ pour $X$ variant de *nil* à $c$. Sur chaque ligne, nous avons partiellement rempli les événements inclus dans le tronçon $f(X) \uparrow t_X^1 \downarrow t_X^2$. La dernière ligne, correspondant à $[\![f]\!](c)$, est l'union des tronçons préalablement calculés.

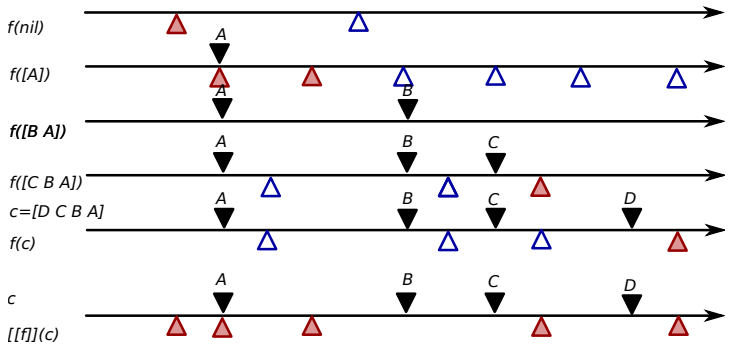


FIGURE 4 – Exemple de la fonction $[\![]\!]$

On remarque que, dans le cas général, seules les réponses de $[\![f]\!](c)$ au dernier événement de $c$ coïncident avec celles fournies par $f$. Par contre, pour le cas particulier des fonctions causales, nous obtenons le résultat suivant :

**Théorème 1 (causal)** $\forall f, causal(f) \Leftrightarrow \forall c, f(c) = [\![f]\!](c)$.

*Preuve.* Pour le sens $\Rightarrow$, la preuve se fait par induction sur $c$. Le cas de base est établi par définition de $[\![f]\!](nil) = f(nil)$. Pour l'induction on a :

$f((t,v)\cdot c) = f((t,v)\cdot c) \uparrow t \bullet f((t,v)\cdot c) \downarrow t$ sachant $\forall c\ t, c = c \uparrow t \bullet c \downarrow t$

$= f((t,v)\cdot c) \uparrow t \bullet f(c) \downarrow t$ par hyp. de causalité et sachant $\forall t\ v\ c, f((t,v)\cdot c) \downarrow t = c \downarrow t$

$= f((t,v)\cdot c) \uparrow t \bullet [\![f]\!](c) \downarrow t$ par hyp. de rec.

$= [\![f]\!](((t,v)\cdot c)$ par définition.

Pour le sens $\Leftarrow$, sachant $c_1 \downarrow t = c_2 \downarrow t$, on a :

$[\![f]\!](c_1 \downarrow t) \downarrow t = [\![f]\!](c_2 \downarrow t) \downarrow t$

$\Leftrightarrow [\![f]\!](c_1) \downarrow t = [\![f]\!](c_2) \downarrow t$ en utilisant le lemme qui suivra.

$\Leftrightarrow f(c_1) \downarrow t = f(c_2) \downarrow t$ par hyp.



**Lemme 1** $\forall f\ c\ t, [\![f]\!](c \downarrow t) \downarrow t = [\![f]\!](c) \downarrow t$

*Preuve.* Par induction sur $c$. Le cas de base est trivial et correspond à $f(\mathbf{nil}) \downarrow t = f(\mathbf{nil}) \downarrow t$. Pour l'induction sachant que l'on ajoute l'événement $(t_0, v)$ à $c$, il faut considérer les deux cas $t_0 \geq t$ et $t_0 < t$. Le premier cas utilise l'hypothèse de récurrence et le second cas n'utilise que des lemmes simples sur $\uparrow$ et $\downarrow$. Nous invitons à regarder les sources Coq pour plus de détails. 

## 4. Version opérationelle

Maintenant qu'est établi le bien-fondé de $[\![f]\!]$ pour les fonctions causales, nous pouvons en donner une version équivalente, totalement réactive :

$play(f, c) =$
  **let** $output = f(c) \uparrow t_c^1$ **in**
    **let** $event = emitUntilEvent(output)$ **in**
      $play(f,\ event \cdot c)$

A chaque appel récursif, la fonction *play* émet les événements du tronçon $f(c) \uparrow t_c^1 \downarrow t_c^2$. La variable *output* contient la partie droite du tronçon, et la fonction $EmitUntilEvent$ va émettre, dans le temps, les événements de ce tronçon, tant qu'un nouvel événement n'est pas apparu dans le flux d'entrée, présupposé au temps $t_c^2$. Cette fonction $EmitUntilEvent$ retourne ce nouvel événement. Pour prendre en compte la dissymétrie du premier tronçon, le premier appel à la fonction *play* doit se faire avec la chronologie $emitUntilEvent(f(\mathbf{nil})) \cdot \mathbf{nil}$.

# 5. Conclusion

Dans le cadre des fonctions manipulant des données temporelles, nous avons caractérisé celles qui sont causales. Cela nous a amené à prouver que ces fonctions pouvaient se simuler avec une autre fonction qui est réactive. Nous tenons à souligner ici que le fait qu'on *peut* simuler une fonction causale par une fonction réactive, n'implique pas qu'il faille le faire de cette manière. En effet, la simulation que nous avons donnée impose de conserver l'ensemble des événements déjà rencontrés (effet cumulatif dans la fonction *play*) [2].

Le but de notre projet, au-delà de la caractérisation des fonctions causales, est de trouver la meilleure manière de décrire les fonctions manipulant des données temporelles [4] tout en garantissant une certaine efficacité quant à leurs exécutions. Une approche plus théorique du travail proposé dans cet article peut se trouver dans [5].

Les preuves ont été effectuées en utilisant le système Coq[1]. Ces preuves sont sans difficultés majeures. Au contraire, il se trouve que l'ensemble des lemmes et le théorème final se prouvent sans l'hypothèse que les chronologies soient triées par ordre décroissant de temps. Il y a peut-être matière à réflexion quant à ce résultat. Les sources sont disponibles sur `ftp://ftp-sop.inria.fr/indes/rp/jfla2017.v`.

Les auteurs remercient Christine Huet pour sa relecture attentive ainsi que les rapporteurs de l'article pour leurs remarques constructives.

# Références

[1] The coq proof assistant. `https://coq.inria.fr`.

[2] S. Archipoff and D. Janin. Structured reactive programming with polymorphic temporal tiles. In *ACM Work. on Functional Art, Music, Modeling and Design (FARM)*, 2016.

[3] Conal Elliott and Paul Hudak. Functional reactive animation. In *International Conference on Functional Programming*, 1997.

[4] Paul Hudak and David Janin. Tiled polymorphic temporal media. In *Proceedings of the 2Nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling & Design*, FARM '14, pages 49–60, New York, NY, USA, 2014. ACM.

[5] David Janin and Bernard Paul Serpette. Timed Denotational Semantics for Causal Functions over Timed Streams. hal-01402209, LaBRI, November 2016.

[6] Eleftherios Matsikoudis and Edward A. Lee. The fixed-point theory of strictly causal functions. *Theoretical Computer Science*, 574 :39–77, April 2015.

2. C'est le principal reproche qui a été exprimé envers Fran[3]