Vers la complétude interactive: exigences pour une machine abstraite orientée interaction

Alice Martin, Mathieu Magnaudet, Stéphane Conversy

Vers la complétude interactive: exigences pour une machine abstraite orientée interaction

Toward Interactive Completeness: Requirements for an Interactive Abstract Machine

Alice Martin
Mathieu Magnaudet
Stéphane Conversy
ENAC
France

ABSTRACT

Turing's work on computability served as the founding theoretical framework for computer science. It aimed at formalizing the intuitive notion of algorithm and gave birth to the concepts of logical automaton and abstract machine. The latter are used to evaluate the expressiveness of programming languages. However, current systems are difficult to model within an algorithmic framework and question the relevance of the classical model. The systems we know are indeed characterized by continuous interactions between human agents, physical and computational processes. Programming these interactive systems requires to specify causal relationships between processes, and not simply the execution of computation. In this paper, we propose to define an equivalent of the classical abstract machine, to model programmable interactive systems. Based on our distinction between computation programming and interaction programming and previous theoretical works that have questioned the classical framework, we elicit the minimal requirements of an abstract machine for interaction. This work is a fist step toward an interactive model to evaluate the expressivity of interaction-oriented languages.

CCS CONCEPTS

• **Human-centered computing → HCI theory, concepts and models**.

KEYWORDS

interactive programming, model, abstract machine

RÉSUMÉ

Les travaux de Turing sur la calculabilité ont servi de cadre théorique fondateur à l'informatique. Ils visaient à formaliser la notion intuitive d'algorithme et ont donné naissance aux concepts d'automate logique et de machine abstraite. Ces derniers sont utilisés pour évaluer l'expressivité des langages de programmation. Cependant, l'informatique actuelle se modélise difficilement dans un cadre algorithmique et remet en question la pertinence du modèle classique. Les systèmes que nous connaissons se caractérisent en effet par des interactions continues entre agents humains, processus physiques et calculatoires. Programmer ces systèmes interactifs demande que l'on spécifie des relations de causalité entre processus, et non plus simplement l'exécution d'un calcul. Dans cet article, nous proposons de définir un équivalent de la machine abstraite classique, pour modéliser les systèmes interactifs programmables. En se basant sur la distinction que nous faisons entre la programmation de calcul et la programmation d'interaction et sur les travaux théoriques précédents qui ont remis en question le cadre classique, nous dégageons les exigences minimales d'une machine abstraite pour l'interaction. Ce travail est une première étape vers un modèle interactif pour évaluer l'expressivité des langages orientés vers l'interaction.

MOTS-CLÉS

programmation interactive, modèle, machine abstraite

ACM Reference Format:
Alice Martin, Mathieu Magnaudet, and Stéphane Conversy. 2021. Vers la complétude interactive: exigences pour une machine abstraite orientée interaction: Toward Interactive Completeness: Requirements for an Interactive Abstract Machine. In *32e Conférence Francophone sur l'Interaction Homme-Machine (IHM '21 Adjunct), April 13–16, 2021, Virtual Event, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3451148.3458644

1 INTRODUCTION

Les dispositifs informatiques avec lesquels nous interagissons effectuent des calculs. Le concept de calcul mis en oeuvre par ces systèmes est hérité des travaux de Turing sur les fondements de l'arithmétique. Turing a apporté une définition rigoureuse d'une notion qui était restée intuitive jusqu'à ses travaux : celle de *fonction calculable* ou de *procédure effective*, permettant de caractériser formellement l'ensemble des fonctions calculables. Ses travaux ont servi de référence pour évaluer l'expressivité des langages de programmation, selon le critère de *Turing-complétude* : les langages sont évalués selon leur aptitude à exprimer l'ensemble des fonctions Turing-calculables. Les dispositifs techniques programmables que nous connaissons aujourd'hui ne se réduisent pas à des automates de calcul. Ces appareils interagissent en effet dans le temps avec un environnement externe. Ils sont démarrés, mis en pause, relancés, et ils reçoivent des données de manière continue. De nombreux auteurs [16, 17, 24, 36, 38, 39] ont déjà fait remarquer que la vocation de la machine de Turing n'était pas de modéliser tout

IHM '21 Adjunct, April 13–16, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8377-6/21/04.
https://doi.org/10.1145/3451148.3458644

ce que pouvaient et pourraient faire les systèmes informatiques. Depuis Turing, de multiples extensions du modèle originel ont été proposées et formalisées, notamment pour rendre compte de la concurrence ou de la gestion du temps dans des systèmes de plus en plus interactifs. Toutefois, il n'existe pas à notre connaissance de modèle unifié pour les systèmes interactifs, équivalent au modèle de calcul que constituait la machine de Turing. De plus, malgré les extensions à la machine de Turing, un argument fort persiste et soutient que toutes extensions de la machine de Turing sont équivalentes au modèle classique.

Le présent travail en cours vise à conceptualiser, et à terme formaliser, un modèle de machine abstraite dédié à la programmation interactive et à l'évaluation de langages interactifs. La motivation de ce travail repose sur deux constats. D'abord, (i) comme évoqué, l'informatique s'inscrit traditionnellement dans le cadre théorique de la calculabilité, qui est bien antérieure aux développements plus récents de la programmation de systèmes interactifs. Or, la programmation d'interaction ne consiste pas simplement en la spécification de calculs. En l'absence de modèle dédié, les spécificités de la programmation interactive ne sont donc pas rendues explicites. Ensuite, (ii) à défaut de modèle unifié, la programmation de systèmes interactifs et les langages de programmation employés ne disposent pas d'un critère d'évaluation adapté, comme l'est classiquement le critère de Turing-complétude où un langage est expressif à l'aune des fonctions calculables qu'il peut exprimer. Nous étayerons dans la section 2 la motivation de ce travail, à savoir le besoin d'une machine abstraite dédiée à l'interaction. Pour cela, nous développons une distinction entre la programmation de *calculs* et la programmation de *comportements interactifs* et articulons la relation entre les concepts de machine de Turing et de machines abstraites avec les langages de programmation. Le travail réalisé jusqu'ici a abouti à trois résultats, présentés en section 3. Premièrement, un état de l'art a permis de décrire la conceptualisation et l'évolution du concept d' "interaction" en informatique. A notre connaissance, ce travail épistémologique n'a pas encore été proposé et constitue une réelle contribution. Deuxièmement, nous avons cherché à identifier les éléments minimaux pour l'expressivité d'un modèle de système interactif. Troisièmement, nous introduisons une proposition, celle de développer et formaliser un critère de complétude pour les langages interactifs à partir de ce modèle. Nous discuterons dans une dernière section les approfondissements à venir de ce travail, qui consiste principalement en la formalisation du modèle (section 4).

2 MOTIVATION: LE BESOIN D'UNE MACHINE ABSTRAITE POUR L'INTERACTION

Avant de présenter le travail réalisé, nous développons dans cette section la motivation du projet et son intérêt pour la communauté IHM. Nous introduisons une distinction entre la programmation de calculs et la programmation d'interactions. Nous rappelons que la programmation de calculs a pour modèle formel la machine abstraite. Il existe une vaste littérature sur les machines abstraites, pour chaque grand paradigme de programmation (impératif, orienté-objet, fonctionnel) [12]. Mais aucun de ces paradigmes et machines abstraites associées ne sont consacrés à la programmation de comportements interactifs. L'absence d'un modèle abstrait dédié pose selon nous des difficultés.

2.1 Programmer du calcul, programmer de l'interaction

Il y a plusieurs manières de caractériser l'activité de programmation. Dans un sens étroit, programmer signifie spécifier une procédure effective, ou un algorithme de calcul, au moyen d'un langage interprétable par une machine. Ce sens étroit est parfaitement illustré par les ouvrages classiques de Donald Knuth [18] entièrement dévolus à des problèmes d'efficience et d'efficacité algorithmique. On peut cependant comprendre la programmation en un sens plus large, comme la spécification du comportement d'une machine au moyen d'un langage. La différence réside dans le fait qu'on veut pouvoir spécifier des comportements qui ne relèvent pas à proprement parler du calcul. On souhaite programmer, par exemple, l'apparition d'une icône, visible pendant trois secondes, notifiant l'utilisateur de l'arrivée d'un email. Ou encore, on souhaite programmer la variation de la luminosité de l'écran de telle sorte qu'elle s'adapte à la lumière ambiante.

Aussi bien la notification de l'email que la luminosité adaptative sont des comportements dont la spécification ne relève pas du concept de procédure effective. Ils décrivent plutôt une relation de causalité, c'est-à-dire ordonnée dans le temps, entre des événements ou des processus. Cette dimension physique des dispositifs actuels a conduit à l'émergence d'un nouveau terme, celui de systèmes cyber-physiques [22, 23]. Programmer ces systèmes impose une contrainte générale: être capable de spécifier des relations de causalité entre événements physiques qui se produisent dans le temps, d'une manière imprévue et non-déterministe[1], et des processus informatiques. Parmi les réactions possibles du système, on peut vouloir spécifier certains processus calculatoires, mais ce n'est là qu'une tâche parmi d'autres. Comme l'ont déjà relevé certains auteurs [7, 8], les systèmes interactifs ont des caractéristiques fondamentalement non-algorithmiques. Ils sont intégrés dans un monde physique avec lequel ils interagissent en continu et on veut pouvoir aussi spécifier ces comportements. Les langages de programmation ont dû évoluer afin de rendre possible la conception de tels systèmes. Ainsi, trouve-t-on dans la littérature en programmation de nombreuses extensions "réactives" à des langages existants [6, 33]. On peut citer à titre d'exemple Scala.React [26], basé sur le langage objet fonctionnel Scala ou encore Reactive ML basé sur le langage fonctionnel non réactif ML [27, 28]).

2.2 Rappel: le lien entre modèle de calcul, machine abstraite et programmation

En informatique théorique, une machine abstraite est un modèle de calcul. C'est une description abstraite des composants d'un automate et un ensemble d'instructions élémentaires permettant de décrire l'exécution d'un calcul. Une machine abstraite est aussi un outil de formalisation pour un langage de programmation : on dit que le langage décrit une machine abstraite. Un langage de programmation peut être vu comme un moyen commode d'exprimer

[1] D'un point de vue épistémique, on ne peut pas savoir à l'avance *quand* l'événement va se produire, même si on a programmé de manière déterministe la réaction à l'événement en question.

le fonctionnement d'une machine abstraite, plutôt que de recourir à la représentation d'un automate, parfois difficilement lisible [19]. Un langage de programmation est alors une manière alternative de spécifier l'algorithme pour une fonction calculable. En retour, un langage de programmation peut être évalué en fonction de la classe d'automates qu'il est capable d'exprimer [9]. S'il peut exprimer toutes les fonctions calculables par une machine de Turing, il est dit *Turing-complet*. L'expressivité du langage est alors évaluée quant à sa puissance de calcul.

Pour illustrer ce concept de machine abstraite, on peut renvoyer à un exemple canonique tel que présenté dans un manuel universitaire [15]. Une machine abstraite pour un langage définit des opérations sur des types de données et l'exécution pas-à-pas d'un programme. Les algorithmes à exécuter doivent représenter des instructions écrites dans un langage de programmation Λ. Pour tout langage de programmation Λ, sa machine abstraite M est un ensemble de structures de données et d'algorithmes qui peuvent permettre le stockage en mémoire et l'exécution des instructions du programme écrit dans le langage Λ.

Une machine abstraite classique présente deux composants principaux. D'une part, il y a un composant mémoire qui stocke le programme à exécuter, ainsi que les données qui constituent les entrées du programme. D'autre part, il y a un interpréteur qui contient un certain nombre d'opérations pour exécuter le programme. Ces opérations sont divisées en *structure de contrôle*, *structure de données*, et *gestion de mémoire*.

2.3 Les objectifs d'un modèle abstrait dédié à la programmation d'interaction

Le modèle de la machine abstraite n'intègre ni des interactions possibles avec un environnement physique, ni une notion de temps. Les horloges ou les entrées et sorties ne sont pas conçues comme des composants de la machine abstraite, et donc ne sont pas représentées dans les langages de programmation correspondants. Par conséquent, si la théorie de la calculabilité offre des garanties de Turing-complétude sur les langages, elle ne dit rien sur leur aptitude à exprimer des comportements interactifs. Même si de nouvelles pratiques en programmation ont émergé pour permettre le design d'interaction, ces langages ne s'appuient pas sur un modèle unifié qui exprimerait les spécificités de l'interaction, comme le fait la machine de Turing pour la programmation de calculs. L'absence d'un modèle unifié pour les systèmes interactifs et leur programmation pose plusieurs difficultés. D'une part, cela n'encourage pas le développement de langages consacrés à l'interaction et ne permet pas de les évaluer de manière pertinente ; il manque notamment un critère de complétude qui permettrait de qualifier l'aptitude d'un langage à exprimer des comportements interactifs. D'autre part, cela empêche les programmeurs d'expliciter, au moyen d'un modèle pertinent, les contraintes propres au design d'interactions.

3 TRAVAIL EFFECTUÉ

Le travail effectué jusqu'ici consiste en (i) un état de l'art sur la notion d'interaction en informatique théorique, (ii) une identification des besoins minimaux pour programmer de l'interaction, d'où pourrait être ensuite dérivé un modèle, (iii) une proposition de typologie et de critère d'évaluation pour les langages interactifs.

3.1 Etat de l'art: le concept d'interaction en informatique théorique

Des extensions à la machine de Turing (conçue pour décrire du calcul) ont été proposées pour rendre compte de comportements interactifs. On relèvera, à titre d'exemple, des solutions algorithmiques proposées à des problèmes de synchronisation [20] et d'accès concurrent [13]; ou encore des automates temporisés intégrant au modèle classique une certaine notion de temps [2]. Il s'agit là d'extensions variées du modèle du calcul pour contourner les contraintes qu'il impose. Ces extensions qui chacune vise à résoudre un problème qui n'a pas de solution évidente dans un modèle de calcul, ne participent pas non plus à expliciter un modèle de l'interaction. On trouve dans la littérature, au moins depuis les travaux de Milner sur les systèmes communicants [29–31], des interrogations fortes sur la compatibilité du cadre théorique classique, celui de la calculabilité, avec l'émergence de nouvelles pratiques et le design de systèmes de plus en plus interactifs. Pour rendre compte de ces nouveaux systèmes, des modèles ont été proposés: certains sont des extensions de la machine de Turing [3, 5, 25, 34, 35, 40–42], d'autres se conçoivent comme radicalement distincts [36, 37, 39]. Nous avons ainsi conduit une revue de la littérature sur les travaux qui questionnent les fondements de l'informatique théorique. Nous en avons isolé les approches principales de l'opposition entre modèle classique calculatoire et système interactif.

Ces interrogations sur le cadre classique de la calculabilité permettent de capturer une intuition forte: le concept d'algorithme ne permet pas de rendre compte de caractéristiques importantes des systèmes interactifs. Toutefois, les travaux que nous avons examinés précédemment n'ont pas construit de lien clair avec l'élaboration de *langages de programmation* pour l'interaction. C'est ce à quoi nous nous sommes attachés dans un second temps, et qui constitue le second pan du travail en cours.

3.2 Identification des besoins minimaux pour programmer de l'interaction

Notre objectif est donc de proposer un modèle de machine abstraite dédié à la programmation des systèmes interactifs, à l'image de ce que sont les machines abstraites pour le paradigme de programmation classique. Rappelons (voir Section 2) que les machines abstraites classiques définissent un critère d'évaluation des langages: leur capacité à exprimer les fonctions Turing-calculables. De la même manière, nous souhaitons disposer d'un critère d'évaluation pertinent pour évaluer l'expressivité d'un langage interactif. Lorsqu'on programme des comportements interactifs, spécifier des algorithmes de calcul ne constitue qu'une tâche parmi d'autres. Ce qui importe avant tout, c'est de pouvoir définir ce qui va déclencher un mécanisme de calcul ou un changement d'état pour la machine. L'expressivité d'un langage interactif doit donc se mesurer à son aptitude à exprimer de telles relations. Nous proposons de détailler trois besoins minimaux pour l'expression de comportements interactifs : la *causalité*, la *transduction*, et le *temps physique*.

(1) *Causalité* : Spécifier un comportement interactif d'une machine programmable repose sur la description de relations causales entre l'occurence d'événements et le déclenchement de divers processus, certains de ces processus étant calculatoires, au sein de la machine. Cela peut être aussi bien

un événement interne au système informatique (comme la fin d'un processus de calcul) qu'externe (comme un clic de souris). Cette dimension est absente de la machine de Turing: on n'y trouve aucun moyen de décrire le lancement de l'exécution en réaction à des événements. Pour la machine de Turing, tout est supposé donné avant le début de l'exécution, les entrées sont déjà spécifiées. L'interaction présuppose, elle, une *inversion du contrôle* : ce n'est plus une machine de Turing qui contrôle elle-même le cours de son exécution, mais c'est au contraire l'environnement externe qui contrôle le flot d'exécution.

(2) *Transduction* : Les mécanismes de gestion d'interruption ou tout mécanisme qui assurerait une gestion de la causalité entre processus externes et internes ne sont pas suffisants pour assurer le lien entre les processus calculatoires de la machine et le monde physique. Un mécanisme supplémentaire est requis pour convertir les grandeurs physiques en données digitales. C'est la transduction. L'importance de la transduction pour l'informatique de l'interaction a déjà été relevée [35]. Il a également été défendu qu'un modèle de bas niveau pour la transduction était crucial pour concevoir des systèmes interactifs [1]. La machine de Turing, elle, en modélisant les entrées comme une séquence de symboles, ne permet pas de rendre compte de la variété des phénomènes physiques qui peuvent provoquer des processus de calculs.

(3) *Temps physique* : Dans la théorie classique des automates, il est fait abstraction du temps physique. On n'y trouve qu'une notion de temps logique, réduit à un ordre. Ainsi, la machine de Turing ou toute machine abstraite dérivée, permet de spécifier pas-à-pas une séquence depuis une entrée à jusqu'à une sortie. Ce n'est pas le temps physique mais le nombre d'étapes de calcul qui sert d'étalon. La durée, pour un système interactif, importe pour au moins deux raisons. Premièrement, les processus calculatoires en jeu sont eux-mêmes le fait de processus physiques internes à la machine et ces processus prennent du temps [21]. Deuxièmement, les systèmes dont nous parlons interagissent avec des humains et mettent en jeu la perception humaine qui est sensible à la durée. Ainsi, le programmeur peut avoir besoin d'ajuster et exprimer une durée pour l'affichage d'une notification, de telle sorte qu'elle soit perceptible.

3.3 Implications pour un modèle interactif

Au niveau du modèle, ces besoins impliquent l'exigence de trois composants minimaux. D'abord, la machine abstraite interactive doit donc intégrer les phénomènes qui produisent les entrées, en réaction à des événements physiques. Nous proposons ainsi de définir un premier composant essentiel: les ***sources causales*** qui englobent les phénomènes physiques externes et les transductions. Ces sources causales ont une structure spatiale et temporelle. Une description formelle de ces sources causales devrait offrir un modèle de l'organisation de ses éléments et caractériser les magnitudes associées (rythme, type). Une souris, par exemple, peut être décrite comme la composition d'un signal 2D envoyant en continu la mesure d'un déplacement, associé à un ensemble de trois signaux valués (absent/appuyé/relâché) pour ses boutons. Une théorie des systèmes interactifs devrait ainsi proposer la formalisation d'une ontologie pour les sources causales. Une première proposition semi-formalisée a déjà été suggérée (voir [11]). A plus haut niveau, il est suffisant de considérer une machine abstraite comme composée d'un ensemble de sources causales, chacune étant conçue comme l'origine possible d'une activation causale. La variété de ces sources, tout comme les possibilités de composition, sont un élément déterminant de la capacité d'un langage à spécifier le comportement d'un système interactif.

Ensuite, nous proposons de définir les ***horloges***. Comme mentionné précédemment, le programmeur de comportements interactifs a besoin de spécifier des instructions temporisées. Ici, la notion de temps requise est celle d'un temps physique, d'une durée, et non celle d'un temps logique réduit à de l'ordre. La notion d'horloge pour cadencer les micro-processeurs synchrones n'est pas celle qui est nécessaire au programmeur d'interaction, car cette horloge est cachée au programmeur (sauf par l'intermédiaire de librairies), et ne permet que d'implémenter l'ordonnancement des exécutions. Une horloge pour l'interaction est une source causale d'un type particulier au sens où elle peut être contrôlée par le programme interactif lui-même : un programme peut spécifier de déclencher à un temps futur défini un comportement particulier, voire annuler ce comportement futur. Elle peut agir comme un minuteur (*timer*) pour spécifier un délai ou comme un métronome pour spécifier un rythme d'exécution.

Enfin, nous dérivons des besoins un troisième composant minimal: le ***vérificateur causal***. Quand on programme des comportements interactifs, une exigence essentielle est que le système réponde de manière déterministe à l'arrivée imprévisible d'événements. La machine interactive doit donc garantir que l'ordre d'exécution corresponde aux relations de causalité spécifiées dans le programme. Dans une architecture classique Von Neumann, au niveau des registres mémoire, c'est le rôle du compteur de programme (*« program counter »*) de garantir le bon ordonnancement des séquences d'instructions. Pour la machine abstraite interactive, nous suggérons de renvoyer à un "vérificateur de causalité", composant abstrait qui garantit les bonnes relations causales entre processus.

3.4 Proposition: typologie de langages interactifs et critère de complétude

Une fois définie une telle machine abstraite interactive, il faut encore construire un lien avec des langages de programmation. On peut examiner ce lien selon deux axes principaux. Le premier est celui de la traduction ou de la sémantique, i.e. comment une expression bien formée de mon langage se traduit en termes de fonctionnement d'une machine abstraite interactive et réciproquement. Pour une machine abstraite interactive par exemple, une relation causale peut être décrite par l'activation d'un processus consécutive à l'activation d'une source soit : $S_{ource} \rightarrow P_{rocess}$. La traduction d'un tel schéma en langage *C* devra probablement faire appel à une fonction système type *select* et à une *callback*. Dans le cas d'un langage orienté interaction, la traduction sera plus immédiate; en Esterel par exemple on écrirait ainsi :

```
input S;
output P;
loop
```

```
        await S;
        emit P
end loop
```

Le second axe d'analyse possible est celui de la mesure de l'expressivité d'un langage donné, i.e. que l'on évalue si un langage L permet d'exprimer tous les comportements interactifs souhaités. Quelques travaux ont été menés dans une direction proche [10, 14], bien qu'ils n'aient pas fait le choix de la description d'une machine abstraite.

Dans cette perspective, il semble possible de définir des classes de langage caractérisant leur expressivité. Un critère de classification pourrait être par exemple celui de la dynamicité des relations causales. Trois niveaux peuvent être identifiés[2] :

(1) Les relations causales sont statiques
(2) Les relations causales peuvent être modifiées en cours d'exécution (certaines sont activées, d'autres sont désactivées)
(3) Les relations causales peuvent être créées ou supprimées en cours d'exécution

De la même manière, il semble que l'on puisse caractériser un langage selon son aptitude à spécifier de nouvelles sources causales ou à les composer, par opposition à un ensemble statique de sources. Le raffinement de ce modèle et des critères de complétude sera réalisé dans un futur travail.

4 PERSPECTIVES FUTURES: FORMALISER LA MACHINE ABSTRAITE INTERACTIVE ET ÉVALUER LES LANGAGES INTERACTIFS

Il n'existe pas de machine abstraite interactive, à notre connaissance. Nous proposons d'en conceptualiser une qui puisse servir de modèle théorique pour rendre compte des spécificités des langages de programmation existants, les évaluer, voire servir également de support au design de nouveaux langages consacrés à la programmation d'interactions.

Nous avons identifié trois dimensions qui nous paraissent caractériser la programmation de systèmes interactifs. Ces dimensions devraient définir trois éléments formalisables pour une machine abstraite interactive: (i) des sources causales, permettant d'initier des relations de causalité, (ii) des horloges permettant de spécifier des durées, (iii) un vérificateur causal, permettant d'orchestrer l'exécution des processus.

Ce travail devra être complété au moins de deux façons. D'une part, une justification rigoureuse des exigences minimales proposées fait encore défaut, et leur intuition reste largement à étayer. Que le temps comme durée, la transduction, la notion de processus physique et leur précise orchestration soient essentiels est une perspective qui n'est pas isolée. On la retrouve chez des designers de systèmes cyber-physiques [21]. Mais il reste à montrer que nos trois critères sont suffisants pour un modèle abstrait général des systèmes interactifs. D'autre part, une formalisation du modèle devra être proposée. La description de la machine interactive est la représentation abstraite d'une architecture matérielle qui décrit les composants indispensables à l'exécution d'un système interactif, tout comme le sont le ruban, la tête de lecture et la table de la machine de Turing. Les programmes qui peuvent s'exécuter sur une telle architecture exigent également une modélisation particulière. En lieu et place des ensembles d'états et transitions de la machine abstraite classique, un modèle pour les systèmes interactifs doit permettre de décrire des relations causales entre des "processus" et en intégrant une notion de temps physique.

Des "briques" pertinentes pour un tel formalisme nous semblent déjà disponibles. Pour ce qui est de la description formelle des relations causales entre processus, il existe des travaux récents autour du concept de structures d'événements dits à *causalité dynamique*, par opposition aux structures d'événements *statiques* qui n'autorisent pas l'ajout ou la suppression de relations causales au cours d'une exécution [4, 32]. L'élaboration d'une telle formalisation doit permettre de rendre compte de l'évolution du graphe de causalité, notamment le fait que, dans un programme interactif, de nouveaux processus peuvent être créés ou détruits et faire ainsi évoluer le graphe de connectivité. Dans la suite de nos travaux nous envisageons de nous appuyer sur ces formalismes en y incluant notamment, le concept d'horloge.

5 CONCLUSION

Nous avons posé la question suivante: le modèle classique, celui de la machine abstraite, est-il suffisamment expressif pour rendre compte des exigences des systèmes interactifs programmables et évaluer les langages de programmation consacrés? Nous avons montré que certaines caractéristiques essentielles de ces systèmes ne pouvaient pas être modélisées dans la cadre conceptuel de la théorie de la calculabilité hérité de la machine de Turing. L'interaction entre processus physiques et processus programmables ou une notion riche de temporalité qui inclut la durée (non pas un temps réduit à une notion d'ordre) imposent le développement d'un cadre conceptuel différent.

Nous avons esquissé les contours de ce que pourrait être une machine abstraite interactive. Nous avons vu que trois exigences majeures doivent être satisfaites pour exprimer l'interaction : (i) l'existence d'un ensemble de sources causales, permettant d'initier des relations de causalité, (ii) l'existence d'horloges permettant de spécifier des durées, et (iii) l'existence d'un vérificateur causal, permettant d'orchestrer l'exécution des processus.

Le raffinement de ce modèle devrait permettre de définir des classes de langages interactifs et des critères d'évaluation adaptés pour qualifier ces langages. Ce travail doit être approfondi et raffiné mais nous pensons qu'il permet d'ores et déjà de définir un cadre d'analyse complémentaire pour les langages de programmation.

REMERCIEMENTS

Ce travail a bénéficié du "Programme d'investissement d'avenir" ANR-17-EURE-0005, conduit par l'ANR.

RÉFÉRENCES

[1] Johnny Accot, Stéphane Chatty, Sébastien Maury, and Philippe Palanque. 1997. Formal Transducers: Models of Devices and Building Bricks for the Design of Highly Interactive Systems. https://doi.org/10.1007/978-3-7091-6878-3_10
[2] Rajeev Alur and David L. Dill. 1994. A theory of timed automata. *Theoretical Computer Science* (1994). https://doi.org/10.1016/0304-3975(94)90010-8

[2] On pourrait ajout un niveau 0, celui des langages qui ne permettent pas d'exprimer de relations de causalité tels que les langages purement fonctionnels (i.e. sans effet de bord et sans extension réactive).

[3] Henrik Reif Andersen, Simon Mørk, and Morten Ulrik Sørensen. 1997. A universal reactive machine. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/3-540-63141-0_7

[4] Youssef Arbach, David Karcher, Kirstin Peters, and Uwe Nestmann. 2015. Dynamic causality in event structures. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-319-19195-9_6

[5] Jos C.M. Baeten, Bas Luttik, and Paul Van Tilburg. 2013. Reactive turing machines. *Information and Computation* (2013). https://doi.org/10.1016/j.ic.2013.08.010

[6] Engineer Bainomugisha, Andoni Lombide Carreton, Tom Van Cutsem, Stijn Mostinckx, and Wolfgang De Meuter. 2013. A survey on reactive programming. *Comput. Surveys* (2013). https://doi.org/10.1145/2501654.2501666

[7] Michel Beaudouin-Lafon. 2004. Designing interaction, not interfaces. In *Proceedings of the Workshop on Advanced Visual Interfaces AVI*. https://doi.org/10.1145/989863.989865

[8] Michel Beaudouin-Lafon. 2006. Human-computer interaction. In *Interactive Computation: The New Paradigm*. https://doi.org/10.1007/3-540-34874-3_10

[9] Robert Boyer and Jstrother Moore. 1997. A Mechanical proof of the Turing completeness of pure Lisp. *Automated Theorem Proving: After 25 Years* (07 1997). https://doi.org/10.1090/conm/029/08

[10] Philippe Brun and Michel Beaudouin-Lafon. 1996. A Taxonomy and Evaluation of Formalisms for the Specification of Interactive Systems. In *Proceedings of the HCI'95 Conference on People and Computers X*. Cambridge University Press, USA.

[11] Stuart K. Card, Jock D. Mackinlay, and George G. Robertson. 1990. The design space of input devices. In *Conference on Human Factors in Computing Systems - Proceedings*. https://doi.org/10.1145/97243.97263

[12] Stephan Diehl, Pieter Hartel, and Peter Sestoft. 2000. Abstract machines for programming language implementation. *Future Generation Computer Systems* (2000). https://doi.org/10.1016/S0167-739X(99)00088-6

[13] E W Dijkstra. 1965. Solution of a Problem in Concurrent Programming Control. *Commun. ACM* 8, 9 (sep 1965), 569—-. https://doi.org/10.1145/365559.365617

[14] Jean Daniel Fekete, Martin Richard, and Pierre Dragicevic. 1998. Specification and Verification of Interactors: A Tour of Esterel. In *Formal Aspects of Human Computer Interaction Workshop*. Sheffield University, 25–32.

[15] M Gabbrielli and S Martini. 2010. Abstract Machine. In *Programming Languages: Principles and Paradigms*. Springer-Verlag, Chapter 1.

[16] Dina Goldin and Peter Wegner. 2005. The Church-Turing thesis: Breaking the myth. In *Lecture Notes in Computer Science*. https://doi.org/10.1007/11494645_20

[17] Dina Q. Goldin, Scott A. Smolka, and Peter Wegner. 2002. Turing machines, transition systems, and interaction. In *Electronic Notes in Theoretical Computer Science*. https://doi.org/10.1016/S1571-0661(04)00220-8

[18] Donald E. Knuth. 1998. *The Art of Computer Programming Volumes 1-3 Boxed Set* (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[19] Donald E. Knuth and Richard H. Bigelow. 1967. Programming language for automata. *J. ACM* 14, 4 (Oct. 1967), 615–635. https://doi.org/10.1145/321420.321421

[20] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (jul 1978), 558–565. https://doi.org/10.1145/359545.359563

[21] Edward A. Lee. 2009. Computing needs time. *Commun. ACM* (2009). https://doi.org/10.1145/1506409.1506426

[22] Edward A. Lee. 2016. Fundamental Limits of Cyber-Physical Systems Modeling. *ACM Transactions on Cyber-Physical Systems* (2016). https://doi.org/10.1145/2912149

[23] A. A. Letichevsky, O. O. Letychevskyi, V. G. Skobelev, and V. A. Volkov. 2017. Cyber-Physical Systems. *Cybernetics and Systems Analysis* (2017). https://doi.org/10.1007/s10559-017-9984-9

[24] Giuseppe Longo. 1999. The Difference between Clocks and Turing Machines. In *Functional Models of Cognition*. https://doi.org/10.1007/978-94-015-9620-6_14

[25] Bas Luttik and Fei Yang. 2016. On the executability of interactive computation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-319-40189-8_32

[26] Ingo Maier and Martin Odersky. 2013. Higher-Order Reactive Programming with Incremental Lists. https://doi.org/10.1007/978-3-642-39038-8_29

[27] Louis Mandel and Florence Plateau. 2009. Interactive Programming of Reactive Systems. *Electronic Notes in Theoretical Computer Science* (2009). https://doi.org/10.1016/j.entcs.2008.01.004

[28] Louis Mandel and Marc Pouzet. 2005. ReactiveML, a reactive extension to ML. In *PPDP'05 - Proceedings of the Seventh ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*. https://doi.org/10.1145/1069774.1069782

[29] Robin Milner. 1993. Elements of Interaction: Turing Award Lecture. *Commun. ACM* (1993). https://doi.org/10.1145/151233.151240

[30] Robin Milner. 1999. *Communicating and Mobile Systems: the π-calculus*. Cambrdige University Press.

[31] Robin Milner. 2006. Turing, computing and communication. In *Interactive Computation: The New Paradigm*. https://doi.org/10.1007/3-540-34874-3_1

[32] G. Michele Pinna. 2020. Representing dependencies in event structures. *Logical Methods in Computer Science* (2020). https://doi.org/10.23638/LMCS-16(2:3)2020

[33] Guido Salvaneschi, Sven Amann, Sebastian Proksch, and Mira Mezini. 2014. An empirical study on program comprehension with reactive programming. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. https://doi.org/10.1145/2635868.2635895

[34] Jan Van Leeuwen and Jiří Wiedermann. 2001. Beyond the turing limit: Evolving interactive systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/3-540-45627-9_8

[35] Jan Van Leeuwen and Jiří Wiedermann. 2006. A theory of interactive computation. In *Interactive Computation: The New Paradigm*. https://doi.org/10.1007/3-540-34874-3_6

[36] Peter Wegner. 1995. Interaction as a Basis for Empirical Computer Science. *ACM Computing Surveys (CSUR)* (1995). https://doi.org/10.1145/214037.214092

[37] Peter Wegner. 1997. Why interaction is more powerful than algorithms. *Commun. ACM* (1997). https://doi.org/10.1145/253769.253801

[38] Peter Wegner and Dina Goldin. 2003. Computation beyond turing machines. https://doi.org/10.1145/641205.641235

[39] Peter Wegner and Dina Goldin. 2006. Principles of problem solving. https://doi.org/10.1145/1139922.1139942

[40] Jiří Wiedermann and Jan Van Leeuwen. 2008. How we think of computing today. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-540-69407-6_61

[41] Jiří Wiedermann and Jan van Leeuwen. 2015. What is computation: An epistemic approach. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-662-46078-8_1

[42] Jiří Wiedermann and Jan van Leeuwen. 2019. Finite State Machines with Feedback: An Architecture Supporting Minimal Machine Consciousness. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-030-22996-2_25