Courbes d'accumulations des machines à signaux

Aurélien Emmanuel

UNIVERSITÉ D'ORLÉANS
ÉCOLE DOCTORALE MIPTIS
LIFO

THÈSE présentée par :

Aurélien EMMANUEL

soutenue le : 8 décembre 2023

pour obtenir le grade de : Docteur de l'Université d'Orléans

Discipline/ Spécialité : Informatique théorique

Courbes d'Accumulations des Machines à Signaux

THÈSE dirigée par :

M. Jérôme DURAND-LOSE	Professeur, LIFO, Université d'Orléans

RAPPORTEURS :

M. Olivier BOURNEZ	Professeur, LIX, École Polytechnique
M. Serghei VERLAN	Professeur, LACL, Université Paris Est Créteil

PRESIDENT DU JURY :

M. Pablo ARRIGHI	Professeur, LMF, Université Paris-Saclay

JURY :

M. Pablo ARRIGHI	Professeur, LMF, Université Paris-Saclay
M. Olivier BOURNEZ	Professeur, LIX, École Polytechnique
M. Jérôme DURAND-LOSE	Professeur, LIFO, Université d'Orléans
Mme Carine LUCAS	Maître de conférences HDR, IDP, Université d'Orléans
M. Serghei VERLAN	Professeur, LACL, Université Paris Est Créteil

Résumé

Cette thèse s'inscrit dans l'étude d'un *modèle de calcul géométrique* : les *machines à signaux*. Nous y montrons comment tracer des graphes de fonctions à l'aide d'arbres unaire-binaires.

Dans le monde des automates cellulaires, il est souvent question de particules ou signaux : des structures périodiques dans le temps et l'espace, autrement dit des structures qui se déplacent à vitesse constante. Lorsque plusieurs signaux se rencontrent, une collision a lieux, et les signaux entrant peuvent continuer, disparaître ou laisser place à d'autres signaux, en fonctions des règles de l'automate cellulaire.

Les machines à signaux sont un modèle de calcul qui reprend ces signaux comme briques de base. Visualisées dans un diagramme espace-temps, l'espace en axe horizontal et le temps vertical s'écoulant vers le haut, ce modèle revient à calculer par le dessin de segments et demi-droites colorés. On trace, de bas en haut, des segments jusqu'à ce que deux ou plus s'intersectent, et l'on démarre alors de nouveau segments, en fonctions de règles prédéfinies.

Par rapport aux automates cellulaires, les machines à signaux permettent l'émergence d'un nouveau phénomène : la densité des signaux peut être arbitrairement grande et même infinie, y compris en partant d'une configuration initiale de densité finie. De tels points du diagramme espace-temps, des points au voisinage desquels se trouvent une infinité de signaux, sont appelés points d'accumulation. Ce nouveau phénomène permet de définir de nouveau problèmes, géométriquement. Par exemple : quels sont les points d'accumulations isolés possible en utilisant des positions initiales et des vitesses rationnelles ? Peut-on faire en sorte que l'ensemble des points d'accumulation forment un segment ? un ensemble de Cantor ?

Dans cette thèse, nous nous attelons à caractériser des graphes de fonctions qu'il est possible de dessiner par un ensemble d'accumulation. Il s'inscrit dans l'exploration de la puissance de calcul des machines à signaux, qui s'inscrit plus généralement dans l'étude de la puissance de calcul de modèles non standards.

Nous y montrons que les fonctions d'un segment compact dans $\mathbb{R}$ dont le graphe coïncide avec l'ensemble d'accumulation d'une machine à signaux sont exactement les fonctions continues. Nous montrons plus généralement comment les machines à signaux peuvent dessiner n'importe quel fonction semi-continue inférieurement. Nous étudions aussi la question sous des contraintes de calculabilité, avec le résultat suivant : si un diagramme de machine à signaux calculable coïncide avec le graphe d'un fonction suffisamment lipschitzienne, cette fonction est limite calculable d'une suite croissante de fonctions en escalier rationnelles.

Mot- clefs : Abstract Geometrical Computation ; Automates Cellulaires ; Diviser pour régner ; Synchronisation d'une ligne de fusiliers ; Fractales ; Machines à Signaux.

Remerciements

Je souhaite tout d'abord exprimer mon immense gratitude à mon directeur de thèse, Jérôme Durand-Lose (Professeur), pour sa patience, sa sensibilité et son esprit rigoureux.

Je souhaite remercier Jérôme Durand-Lose, Shahrzad HEYDARSHAHI (Doctorant), Florent BECKER (Maitre de Conférence), Mathieu LIEDLOFF (Professeur), Nicolas OLLINGER (Professeur), Jean Michel COUVREUR (Professeur), Carine LUCAS (Maître de Conférence HDR) et bien d'autres, pour les échanges stimulants et enrichissants que j'ai pu avoir avec eux en matière d'enseignement, de recherche et de méthodes de travail.

Je souhaite remercier le personnel administratif et technique du LIFO, du IIIA et plus généralement de l'université d'Orléans, dont le support est précieux.

Je souhaite encore remercier Pablo ARRIGHI (Professeur), et Maureen CLERC (directrice de recherche) pour mes premières expériences du monde de la recherche.

Je souhaite enfin remercier Alice KARAKACHIAN, ma conjointe, ainsi Marie-Annick BERGERAT, ma mère, pour leur soutien psychologique et la relecture d'une grosse partie de mon manuscrit.

Sommaire

Introduction

Cette thèse se place dans le cadre de l'étude d'un *modèle de calcul non standard*, les machines à signaux. Ce modèle, qui consiste à calculer par le dessin de traits – ou signaux, tire son origine de certains diagrammes espace-temps d'automates cellulaires. Il s'agit pour nous ici d'étudier les courbes de fonctions que l'on fait apparaître en *accumulant* une infinité de signaux.

Modèles de Calcul non Standard

Les fonctions récursives, les machines de Turing [1] [2], le λ-calcul [3, 4], les ordinateurs d'architecture de Babbage et les ordinateurs d'architecture de von Neumann sont autant de modèles de calcul bien étudiés, discrets, finis, et dont l'équivalence est bien établie. Par équivalence, on entend, de manière informelle, qu'ils se simulent l'un l'autre.

Cette équivalence, a priori frappante, a donné lieu à la thèse de Church-Turing [5]. Celle-ci postule que la notion de calculabilité, telle que réalisable par un humain ou un procédé physique, est entièrement capturée par les modèles de calcul usuels, par exemple les machines de Turing [2].

Cette thèse possède aussi des variantes incluant la complexité en temps : une thèse de Turing étendue postule que tout phénomène physique peut être simulé avec un degré de précision arbitraire par une machine à calcul en un temps raisonnable. "En un temps raisonnable" signifie généralement en un temps fonction polynomiale de la taille du problème, et "machine à calcul" signifie par exemple machine de Turing probabiliste pour la thèse de Church-Turing étendue classique [2], machine de Turing quantique pour la thèse de Church-Turing étendue quantique, et cætera. Tous les modèles de calcul discrets, finis et déterministes connus peuvent être simulés efficacement par une machine de Turing, et la plupart leur sont équivalents – certains sont moins puissants – et satisfont donc la thèse simple et celle étendue de Church-Turing. Le terme *modèles de calcul standards* réfère ainsi à de tels modèles, auxquels s'opposent les autres modèles, dits non standards.

Explorer ces modèles de calcul non standards peut permettre, selon le modèle, de réfléchir sur ce qu'il est possible de faire pour un esprit humain ou une machine, sur la définition de ce qu'est un problème algorithmique, sur la puissance des machines à calcul, ou encore sur la nature du monde. Ces modèles permettent encore de raisonner sur et de hiérarchiser des problèmes a priori insolubles en pratique, car s'il existe des problèmes indécidables, certains le sont plus que d'autres.

Nous présentons ici quelques modèles de calcul non standard à titre d'illustration, puis les mettons en lien avec le modèle des machines à signaux, que nous présentons aussi.

Modèles Distribués

Les modèles de calcul distribués permettent un temps d'exécution plus court, mais a priori pas davantage que proportionnel au nombre d'entités de calcul. Les automates cellulaires [1] peuvent être vus comme un exemple de modèle de calcul distribué. Des cellules sont disposées sur une droite (automates cellulaires de dimension 1) ou une grille régulière (dimension 2), chaque cellule étant dans un certain état, parmi une liste finie commune à toutes les cellules. A chaque pas de temps, elles changent simultanément d'état, selon une fonction, commune et déterministe, des états des cellules de leur voisinage, le voisinage d'une cellule étant un ensemble fini de cellules défini par leur position relative à celle initialement considérée.

La simulation d'un automate cellulaire par une machine de Turing pose déjà un problème de définition : si les modèles standards peuvent tous se rapporter à un calcul de fonction des entiers dans les entiers ou à un problème de décision, un nombre dénombrable de cellules contient plus d'informations que ne peut contenir un seul nombre entier. Le ruban infini d'une machine de Turing suffit à encoder tout l'espace – initial ou non – d'un automate cellulaire, mais la notion usuelle de complexité ne fait alors plus sens. On a toutefois les résultats suivants : les automates cellulaires peuvent être simulés par des machines de Turing, au sens où la fonction qui associe les coordonnées d'une cellule et un temps à un état est calculable, et ce de manière polynomiale. Sous certaines conditions, par exemple si la grille de cellules est

1. pour introduction aux machines de Turing et aux automates cellulaires, on peut par exemple se référer à C. Corge, *Machines de Turing et automates cellulaires: du trait gravé au très animé.* Ellipses, 2008
2. S. Aaronson, "Introduction to quantum information science, lecture notes," Fall 2018, p.9

finie, alors il est possible de calculer l'état de toutes les cellules à un temps de simulation donné, en un temps de calcul polynomial de ce temps de simulation et de la taille de la grille.

Modèles Non déterministes

Les modèles de calcul probabilistes, classiques ou quantiques, peuvent aussi permettre de réduire le temps d'exécution. Par exemple, les machines de Turing probabilistes sont des machines de Turing qui peuvent utiliser du hasard, par exemple à l'aide d'un ruban rempli de bits aléatoires, ou à l'aide de transitions non déterministes munies de probabilités[7]. Le non déterminisme n'augmente généralement pas la capacité de calcul. En effet, tant que les branchements non déterministes sont en nombre fini, une machine déterministe peut tous les explorer séquentiellement, et même, pour les machines de Turing probabilistes, garder trace des probabilités de chaque branche. En terme de complexité cependant, il est tout à fait possible que les modèles probabilistes soient plus puissants, ce qui donne lieu à de nouvelles classes de complexité.

Par exemple la classe BPP (bounded-error probabilistic polynomial time) est la classe de décision des problèmes décidés par une machine probabiliste en temps polynomial avec probabilité d'erreur inférieure à 1/3.

Modèles Continus

Les ordinateurs analogiques reposent sur des systèmes dynamiques continus. De manière très générale, ils peuvent être vus comme des solveurs d'équations aux dérivées partielles. Une manière d'en interpréter un résultat est de regarder la valeur finale d'un paramètre.

Marian B. Pour-El et Ning Zhong ont montré que, si on suppose une précision infinie, la propagation d'une vague peut être non calculable bien que ses conditions initiales le soient [8]. Daniel Graça, Manuel Campagnolo et Jorge Buescu ont montré quant à eux qu'un système d'équations différentielles ordinaires polynomiales peut simuler une machine de Turing universelle [9].

Réciproquement, une machine de Turing peut résoudre numériquement des équations différentielles de manière approchée à un degré de précision arbitraire.

En terme de vitesse de calcul, des manipulations standards d'équations différentielles permettent d'effectuer un calcul en un temps arbitrairement court. Les ordinateurs analogiques sont donc, a priori, plus rapides que les machines de Turing. En pratique, les ordinateurs analogiques ne peuvent être accélérés à l'infini, par exemple ceux électriques sont limités par les interférences provoquées par de trop hautes tensions. Olivier Bournez, Daniel Graça et Amaury Pouly ont établi une notion de complexité analogique pour les systèmes d'équations différentielles, basée sur la longueur des courbes [10]. Cette notion est physiquement pertinente et ils montrent l'équivalence en complexité des ordinateurs analogiques avec les machines de Turing.

Modèles de Super Calcul

Certains modèles de calcul abstraits sont conçus spécifiquement pour être strictement plus puissant que les machines de Turing. Les machines à oracle [3] sont des machines de Turing munies d'une boîte noire (l'oracle) capable de résoudre certains problèmes de décision en une opération.

Ces machines permettent de créer de nouvelles classes de complexité, notées A^L où A est une classe de complexité existante et L est un langage reconnu par un oracle. Ces nouvelles classes de complexité constituent un outil de comparaison des classes existantes, mais aussi de hiérarchisation des problèmes indécidables.

Machines à Signaux

De manière informelle, les Machines à Signaux [4] peuvent se définir de la manière suivante : on trace, sur le plan euclidien, des rayons colorés, les signaux, de bas en haut. Lorsque plusieurs de ces signaux se rencontrent, ils s'interrompent – et sont donc des segments plutôt que des demi-droites – et on trace, depuis leur intersection, d'autres signaux, de couleurs et d'angles prescrits par des règles valables pour tout le dessin. Le dessin ainsi produit est appelé diagramme de machine à signaux. L'ensemble des rayons colorés que l'on s'est initialement donné de tracer au bas de la page est appelé configuration initiale. Cette configuration initiale constitue le moyen d'encoder des entrées lorsque l'on veut interpréter les machines à signaux comme des machines à calculer. L'apparition ou non d'une certaine couleur, par exemple, peut servir à encoder la sortie d'un problème de décision. C'est sans surprise que les machines à signaux simulent des machines de Turing [5]

3. H. Christos, "Papadimitriou: Computational complexity," *Addison-Wesley*, vol. 2, no. 3, p. 4, 1994, Section 14.3 : Oracle, pp. 339–343

4. Les machines à signaux sont introduites dans J. Durand-Lose, *Calculer géométriquement sur le plan – machines à signaux*. Habilitation à Diriger des Recherches, École Doctorale STIC, Université de Nice-Sophia Antipolis, 2003, et on en redonne une définition formelle au chapitre 1

5. Dans [13] il est montré qu'une machine à signaux peux simuler un automate à deux compteurs, donc une machine de Turing.

Ce modèle de calcul non standard est intéressant à plusieurs égards.

Tout d'abord, on peut simuler les machines à signaux, en partie au moins, à la main, avec à peine plus que du papier, une règle et des crayons de couleur.

Ensuite, il mêle des caractéristiques continues – comme la position ou l'angle des signaux – et discrètes – comme les collisions entre signaux. On retrouve ce caractère hybride d'information continue et de pas de calcul discret dans les machines Blum Shub Smale [14], des machines à registre capables d'effectuer comme opération élémentaire un calcul arithmétique [14, 15]. Machines à signaux et machines Blum Shub Smale linéaires sont prouvées équivalentes [16] [6] , nous le redémontrons au chapitre 3 en construisant des outils – dont un langage et son compilateur associé – facilitant la création de machines à calcul équivalentes à Blum Shub Smale et leur simulation par machines à signaux.

La nature géométrique des machines à signaux permet de définir des problèmes uniques sans contrepartie standard évidente comme nous allons le voir dans la section suivante .

Le modèle des machines à signaux donne aussi lieu à l'émergence de phénomènes infinis : certaines règles et configurations initiales donnent lieu au tracé de signaux de plus en plus courts de sorte qu'il y en a une infinité dans un espace fini [12]. Ce phénomène, dit d'accumulation, a des implications quant au non-déterminisme [12] et à la puissance de calcul du modèle [17]. Dans [18], il est montré que les machines à signaux peuvent résoudre, en temps fini, le problème de l'arrêt et en utiliser le résultat. Ce résultat peut être élargi au calcul Blum Shub Smale [19], et ce de manière conservative (nombre de signaux constant) et réversible si l'on veut [20]. Ce type de calcul, consistant à effectuer un nombre arbitrairement grand, voire infini, d'opérations en un temps fini borné, y est appelé calcul par trou noir, en rapport avec la densité d'information. Dans [21], il est montré que les machines à signaux peuvent aussi s'inscrire dans le cadre de l'analyse computationnelle et simuler des machines de Turing de type 2, c'est-à-dire des machines à deux rubans, un de lecture stockant une entrée infinie, et un ruban d'écriture, à sens unique, sur lequel doit être écrit un résultat infini.

Calcul et Géométrie

Parmi les nouveaux types de problèmes que les modèles de calcul non standards soulèvent, il y a les problèmes géométriques.

Automates Cellulaires

Les automates cellulaires peuvent être vus comme des « modèles-jouets ». Par « modèle-jouet », expression que nous empruntons à Pablo Arrighi [22], nous entendons qu'ils présentent, de manière épurée, certaines caractéristiques jugées essentielles à des modèles ou à des phénomènes plus complexes sur lesquels nous cherchons à bâtir une intuitions. Pour les automates cellulaires ces caractéristiques sont la causalité (le futur se calcule à l'aide du présent), le déterminisme, la synchronie, la localité et l'uniformité. Le modèle plus complexe sous-jacent peut être celui de la mécanique Newtonienne ou plus généralement de la physique classique.

Il est ensuite possible de jouer sur la définition du modèle et d'observer les phénomènes qui émergent. Par exemple, dans [22], Pablo Arrighi, Simon Martiel et Simon Perdrix explorent, pour une généralisation des automates cellulaires sur des graphes quelconques et dynamiques, la relation entre réversibilité et création d'espace.

Il y a principalement deux approches à l'étude des automates cellulaires.

La première consiste à partir d'un automate cellulaire défini par un ensemble de règles, arbitraires ou bien modélisant un système existant, et d'observer et étudier les phénomènes émergents.

Par exemple : la Fig. 1, issue de [23] [7] , montre le diagramme espace-temps d'un automate cellulaire où l'on voit des signaux et leurs interactions. Un de ces phénomènes émergents est la présence de signaux, structures se déplaçant dans l'espace au cours du temps.

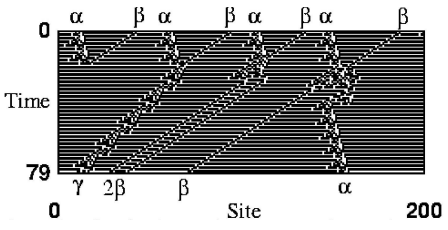


FIGURE 1 – Automate cellulaire exhibant des signaux.

6. Les machines à signaux peuvent simuler une machine Blum Shub Smale, et une Machine Blum Shub Smale peut calculer le diagramme d'une machine à signaux jusqu'à la première accumulation.

7. Figure réimprimée depuis W. Hordijk, C. R. Shalizi, and J. P. Crutchfield, "Upper bound on the products of particle interactions in cellular automata," *Physica D: Nonlinear Phenomena*, vol. 154, p. 240–258, Jun 2001, Copyright (2001), avec permission d'Elsevier

La seconde approche consiste à concevoir les règles définissant un automate cellulaire répondant à un problème. Un tel problème est, par exemple, la synchronisation d'une ligne de fusiliers : comment l'ordre de tir d'un général peut être transmis de sorte qu'une ligne de fusiliers tire de manière synchrone, sous la contrainte d'un déplacement d'information local et lent relativement au niveau de précision de synchronisation (le général ne peut pas crier pour être entendu par toute la ligne à la fois) [24, 25, 26]. En terme d'automate cellulaire, la contrainte d'un général donnant un ordre de tir est traduite par le fait d'avoir une seule cellule active initialement, les cellules inactives – dites encore quiescentes – étant des cellules dont l'état reste inchangé lorsque chacune des cellules de leur voisinage est dans un état inactif. La contrainte de synchronie se traduit par un état remarquable que l'on veut obtenir sur toute une ligne en un moment exact (pas plus tôt). Enfin, le déplacement relativement lent se traduit par un voisinage ne dépendant pas de la longueur de la ligne de fusiller et qui est petit devant elle.

On peut interpréter ce problème géométriquement : il s'agit de tracer une ligne horizontale dans le diagramme espace temps.

La Fig. 2, extraite de [24] [8] , montre comment résoudre ce problème.

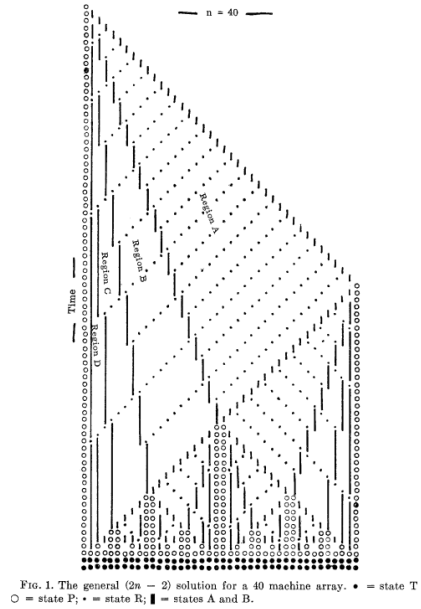


FIG. 1. The general $(2n - 2)$ solution for a 40 machine array. • = state T ○ = state P; • = state R; ▌ = states A and B.

FIGURE 2 – Solution du problème de synchronisation d'une ligne de fusiliers pour les automates cellulaires.

Ici encore, on remarque des signaux, qui sont cette fois-ci utilisés pour résoudre le problème via des constructions géométriques. Dans le même article se trouve la Fig. 3 [8], qui montre le schéma de construction géométrique imaginé fournissant les signaux principaux à tracer. Pour obtenir la Fig. 2, il faut adjoindre des signaux secondaires pour marquer les intersections, et, comme ils sont arbitrairement lents, ils sont pris statiques et décalés d'une cellule à la fois à l'aide d'un dernier jeu de signaux auxiliaires. Enfin, des efforts relatifs à la discrétisation sont nécessaires pour le déclenchement simultané des cellules.

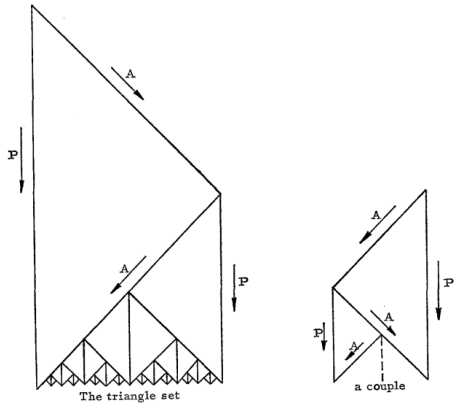


FIGURE 3 – Construction .

Les automates cellulaires regorgent d'autres tels problèmes de nature géométrique. Dans [27], des cercles et paraboles discrets sont tracés, encore une fois à l'aide de signaux.

8. Figures réimprimée depuis A. Waksman, "An optimum solution to the firing squad synchronization problem," *Inf. Control.*, vol. 9, no. 1, pp. 66–78, 1966, Copyright (2001), avec permission d'Elsevier

Machines à Signaux

Le modèle de machine à signaux reprend comme éléments de base les signaux, tels que ceux observés et utilisés dans les automates cellulaires, mais sans chercher à les discrétiser. Une machine à signaux est la donnée d'un ensemble fini de signaux, de leur vitesse, ainsi que d'un jeu de règles de collision [9]. Ces signaux évoluent sur la droite réelle à vitesse constante jusqu'à se croiser. Ce faisant, ils peuvent disparaître, changer, ou d'autres signaux peuvent apparaître, comme spécifié par les règles de collision, qui sont déterministes.

Les machines à signaux peuvent donc être vues comme un autre « modèle-jouet », à support continu et à dynamique discrète, présentant, à l'instar des automates cellulaires, les aspects de causalité, déterminisme, synchronie, localité et uniformité.

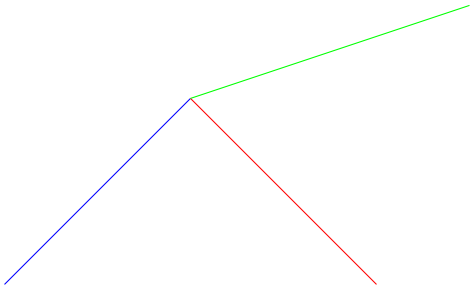


FIGURE 4 – Exemple de diagramme espace-temps d'une machine à signaux.

La Fig. 7 montre un exemple plus complexe de machine à signaux.

Pour l'étude des machines à signaux, on retrouve les deux approches évoquées pour les automates cellulaires. Les machines à signaux sont encore peu utilisées pour modéliser des systèmes existant, mais on peut étudier des propriétés ou phénomènes émergents de l'ensemble ou d'un sous-ensemble de machines à signaux quelconques. On peut aussi concevoir des machines à signaux dans le but qu'elles exhibent des propriétés ou phénomènes désirés.

Un premier type de questions géométriques relatives aux machines à signaux est celui de l'atteignabilité d'un point : étant donné un point de l'espace-temps, existe-t-il un diagramme de machine à signaux tel que ce point soit, par exemple, une collision. Ou encore, étant donné une machine à signaux, que peut-on dire de la position des collisions. Ce type de problèmes est généralement étudié sous la contrainte que les machines à signaux considérées soient rationnelles, c'est-à-dire celles où la vitesse et la position initiale des signaux sont rationnelles. D'une part, ces problèmes seraient triviaux sans cette contrainte, il suffit de tracer n'importe quels deux signaux se collisionnant au point cible. D'autre part, cette contrainte est une manière de limiter la quantité d'information contenue dans la configuration initiale. Par là, nous entendons que la position initiale est descriptible en un nombre fini de bits.

Dans [28], Jérôme Durand-Lose montre que dans les diagrammes de machines à signaux rationnelles ayant au plus trois vitesses de signaux, les collisions se situent sur un réseau, une grille.

Les machines à signaux, même rationnelles, ne constituent pas un modèle géométriquement équivalent aux automates cellulaires. L'absence de discrétisation rend possible une densité de signaux et de collisions arbitrairement grande, voire infinie [18, 19]. Un point de l'espace-temps auquel se produit un tel phénomène est appelé point d'accumulation. Il n'existe donc pas de manière générale de simuler une machine à signaux avec un automate cellulaire dont le temps et l'espace de calcul serait proportionnel au temps et à l'espace de calcul de la machine à signaux. Dans le cas particulier des machines rationnelles à trois vitesses, la conversion automatique d'une machine à signaux en automate cellulaire est possible [29].

La Fig. 5 montre un premier exemple de point d'accumulation.

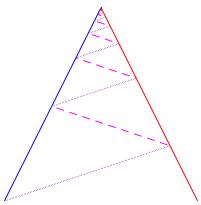


FIGURE 5 – Exemple simple de densité infinie de signal (accumulation) dans un diagramme machine à signaux.

Ces points d'accumulation soulèvent des problèmes aussi bien pratiques que théoriques. Toujours à l'occasion de [12], Jérôme Durand-Lose a développé un interpréteur de machines à signaux, qui permet, à l'aide d'un code spécifiant une machine et une configuration initiale, d'en dessiner le diagramme [10].

9. Les machines à signaux sont définies formellement au chapitre 1 section 1.1.1
10. `https://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose/Recherche/AGC/intro_AGC.html`

Ce diagramme n'est tracé que jusqu'au temps de première accumulation, et il n'est a priori pas possible de faire beaucoup mieux. L'autre problème est celui de la définition même de la dynamique opérée par une machine à signaux. Le comportement au-delà d'une accumulation n'est a priori pas défini. Dans [12] (deuxième partie), Jérôme Durand-Lose explore des manières de le définir. Dans [30] ainsi que dans cette thèse (au chapitre 1), nous choisissons une telle manière de les traiter, que nous voulons minimaliste.

Un problème géométrique relatif aux accumulations est celui d'accumulations isolées : il s'agit de savoir quels points peuvent être accumulations isolée d'un diagramme d'une machine à signaux rationnelle. Ce problème est résolu dans [31] : un point d'accumulation isolé d'un diagramme de machine à signaux rationnelle est nécessairement d'ordonnée (temps) un réel semi-calculable – c'est-à-dire limite croissante d'une suite calculable, et d'abscisse (espace) la différence entre deux réels semi-calculables. De plus, il existe un procédé suffisant à construire, étant donné n'importe quel de ces points, une machine à signaux et une configuration initiale de sorte que le diagramme espace-temps ait une accumulation isolée en ce point.

Ensuite, il est naturel de s'interroger sur les ensembles d'accumulation. La Fig. 6, tirée de [12], fournit quelques exemples d'ensembles d'accumulation. Le diagramme le plus à gauche peut s'interpréter comme une synchronisation d'une ligne de fusiliers.

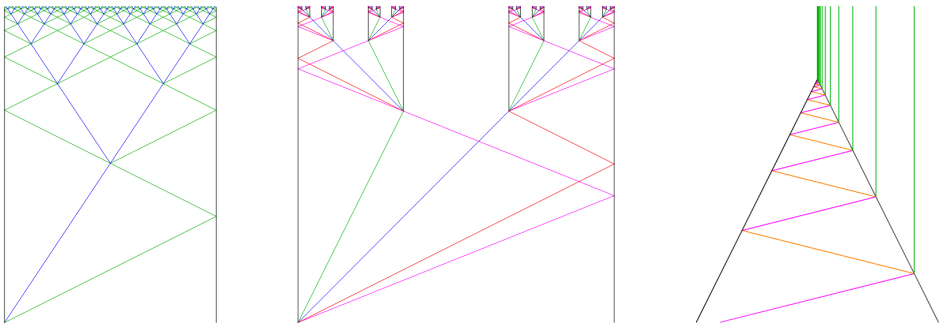


FIGURE 6 – Exemple d'ensembles d'accumulation.

Dans cette thèse, nous nous intéressons au tracé de courbes par ensembles d'accumulations de machines à signaux, et étudions, plus particulièrement et par simplicité, le tracé de graphes de fonctions. Nous étudions principalement la conception de machines à signaux permettant de tracer des graphes, mais aussi l'étude des graphes possibles par une dynamique arbitraire.

Contribution

Nous avons d'abord une approche effective du problème, et établissons des outils permettant de construire une large classe de machines à signaux permettant de tracer une large classe de graphes de fonction.

Nous apportons aussi quelques éléments en vue d'une caractérisation des ensembles d'accumulations possibles.

Ces problématiques sont abordées différemment selon que l'on se restreigne ou non aux machines rationnelles.

Synchronisation en Saccade d'une Ligne de Fusiliers

Le premier résultat de cette thèse, publié dans [30], est qu'il existe une machine à signaux capable de produire un diagramme dont l'ensemble d'accumulation est un segment de pente arbitraire – la pente dépend de la configuration initiale, comme illustré par la Fig. 7.

Ce résultat est exposé au chapitre 2.

Calculer avec des Signaux

Il est théoriquement possible de simuler des machines Blum Shub Smale par des machines à signaux [16], et de combiner ceci au calcul par trou noir [19].

Dans cette thèse, au chapitre 3, nous élaborons un langage effectif permettant de coder des machines Blum Shub Smale. Nous implémentons un interpréteur permettant de simuler de telles machines lorsque toutes les données sont rationnelles. Enfin, nous compilons ce code vers du code AGC, qui permet de définir des machines à signaux et leurs configurations.

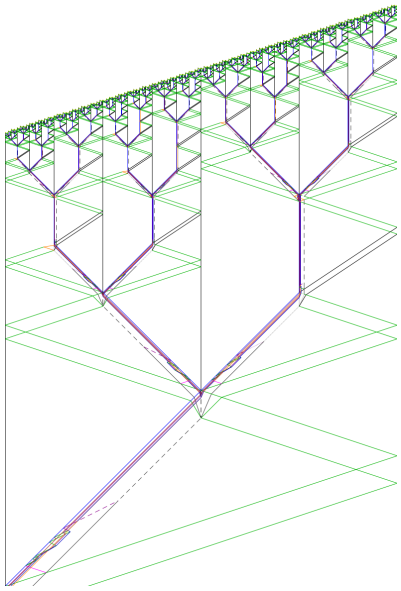


FIGURE 7 – Solution du problème de « saccade » d'une ligne de fusiliers pour les machines à signaux.

Dessiner

Nous proposons ensuite, au chapitre 4 une méthode générale de tracé de courbe par dessin récursif d'arbre unaire binaire infini (sans feuille).

Cette méthode, basée sur le principe « diviser et conquérir », généralise celle développée pour le tracé de segment de pente paramétrée—[30], illustré par la Fig. 7.

Sur chaque branche se trouve une machine à calcul, ayant la puissance d'un calcul Blum Shub Smale arbitrairement long, qui décide de l'arité du prochain sommet et transmet de l'information à ses enfants.

Caractérisation

Nous montrons, au chapitre 4 toujours, qu'avec de l'information réelle sur la condition initiale, il est possible d'encoder n'importe quel arbre unaire binaire, et ce faisant de tracer n'importe quelle fonction semi-continue inférieurement.

Les deux résultats suivants sont exposés au chapitre 5.

Si on se restreint aux machines et configurations rationnelles, il est possible de tracer n'importe quel arbre dont l'encodage est calculable, ce qui correspond aux fonctions qui sont limites croissantes d'une suite calculable de fonctions en escalier rationnelles. Cela signifie que chaque fonction en escalier est de subdivision et de hauteur rationnelles, donc est finiment descriptible, et la suite des descriptions de fonctions est calculable.

Réciproquement, on montre que, sous certaines conditions, si un diagramme de machine à signaux a un ensemble d'accumulation qui est le graphe d'une fonction suffisamment – relativement aux vitesses des signaux – lipschitzienne, alors cette fonction est encore limite croissante d'une suite calculable de fonctions en escalier rationnelles.

Chapitre 1

Définitions et Outils Usuels

1.1 Définitions

Cette section a pour but de définir un certain nombre de notions utilisées tout au long de cette thèse. Les notions dont l'usage est limité à une section ou un chapitre sont susceptibles d'être définies localement plutôt qu'ici.

On note $\mathbb{N}$ l'ensemble des entiers naturels, et $\mathbb{Z}$ l'ensemble des entiers relatifs.

On note $\mathbb{R}$ le corps des nombres réels et $\mathbb{Q}$ le corps des nombres rationnels. Dans toutes les parties théoriques, il sera question de plan euclidien et de manipulation de réels tandis que toutes les implémentations machine sont faites avec des calculs exacts sur nombres rationnels.

On note D l'ensemble des *nombres rationnels dyadiques*, c'est-à-dire l'ensemble $\{\frac{a}{2^b} | (a,b) \in \mathbb{Z} \times \mathbb{N}\}$

1.1.1 Machine à Signaux

Une machine à signaux se définit par ses types de signaux et sa dynamique : la vitesse des signaux dans le vide et les conséquences de leur collision.

Définition 1 (Machine à Signaux(MaS))**.** Une *Machine à Signaux* est un triplet (M, S, R) tel que :
- M est un ensemble fini. On appelle ses éléments *méta-signaux* ;
- $S : M \to \mathbb{R}$ est une *fonction de vitesse* (chaque méta-signal est associé à une vitesse) ;
- R est un ensemble fini de *règles de collision* ; une règle de collision ρ consiste en deux ensembles de méta-signaux de vitesses distinctes – au sein de chaque ensemble : un ensemble d'entrée ρ^- contenant au moins deux signaux et un ensemble de sortie ρ^+ ; cette règle peut s'écrire $\rho^- \to \rho^+$. R est déterministe : $\rho \neq \rho'$ implique $\rho^- \neq \rho'^-$.

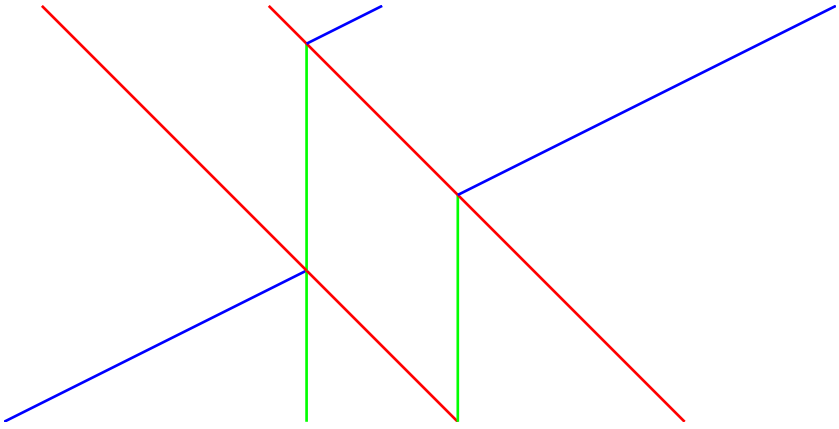


FIGURE 1.1 – Un exemple de diagramme de machine à signaux

Par exemple, la machine de la figure 1.1 a trois méta-signaux, $\overrightarrow{\text{Bleu}}$, $\overleftarrow{\text{Rouge}}$, et Vert, de vitesses respectives 2, -1 et 0. Les règles de collisions en jeu sont les suivantes : $\{\overrightarrow{\text{Bleu}}, \text{Vert}, \overleftarrow{\text{Rouge}}\} \to \{\overleftarrow{\text{Rouge}}, \text{Vert}\}$ et $\{\text{Vert}, \overleftarrow{\text{Rouge}}\} \to \{\overleftarrow{\text{Rouge}}, \overrightarrow{\text{Bleu}}\}$.

Définition 2 (Configuration)**.** Une *configuration* c est une application de $\mathbb{R}$ dans $M \cup R \cup \{\oslash, ✺\}$, c'est-à-dire qu'elle associe chaque point de la ligne réelle à un méta-signal, une règle, une valeur $\oslash$ dénotant un espace vide, ou encore ✺ dénotant une accumulation.

Une configuration est dite finie s'il y a un nombre fini de points associés à une valeur différente de $\oslash$.

Par exemple, dans la figure 1.1, la configuration initiale est la suivante :

$$\begin{cases} c(-4) = \overrightarrow{\text{Bleu}} \\ c(0) \ = \text{Vert} \\ c(2) \ = \{\overrightarrow{\text{Bleu}}, \text{Vert}, \overleftarrow{\text{Rouge}}\} \to \{\overleftarrow{\text{Rouge}}, \text{Vert}\} \\ c(x) \ = \oslash \qquad \text{partout ailleurs} \end{cases}$$

La configuration finale contient deux signaux $\overleftarrow{\text{Rouge}}$ et deux signaux $\overrightarrow{\text{Bleu}}$.

Dans le reste de la thèse, $\mathbb{T}$ désigne un espace de **temps**, à savoir un intervalle de $\mathbb{R}^+$. Dans les figures, le temps est représenté de manière verticale, s'écoulant vers le haut.

Définition 3 (Espace-Temps)**.** *L'espace-temps* est le produit cartésien de l'espace et du temps : $\mathbb{R} \times \mathbb{T}$.

Diagramme Espace-Temps, Définition Constructive

Pour simplifier les notations, on introduit la relation *« être issue de »*, $\ltimes \in M \times (M \cup R \cup \{\oslash\})$, définie de la manière suivante :

— $\forall \mu \in M, \mu \ltimes \mu$

— $\forall \rho \in R, \forall \mu \in \rho^+, \mu \ltimes \rho$

Définition 4 (Dynamique Constructive)**.** Définissons une *dynamique* qui, partant d'une configuration finie c_t au temps t, définit des configurations ultérieures.

Le temps de la prochaine collision, noté $\Delta(c_t)$, est égal au temps minimal d tel que :

$$\exists x_1, x_2 \in \mathbb{R}, \exists \mu_1, \mu_2 \in M \begin{cases} \quad x_1 + d{\cdot}S(\mu_1) = x_2 + d{\cdot}S(\mu_2) \\ \wedge\, \mu_1 \ltimes c(x_1) \\ \wedge\, \mu_2 \ltimes c(x_2) \end{cases} .$$

d vaut $+\infty$ si un tel temps n'existe pas.

Pour un temps t' compris strictement entre t et $\Delta(c_t)$, on définit la configuration $c_{t'}$ comme suit : $c_{t'}(x) = \mu$ si et seulement si $\mu \ltimes c_t(x + (t - t'){\cdot}S(\mu))$. Par définition de $\Delta(c)$, il y a au plus un méta-signal μ séant pour chaque x, et $c_{t'}(x) = \oslash$ quand il n'y en a aucun.

Pour la configuration au temps $t + \Delta(c_t)$, on établit d'abord les collisions : on prend $c_{t+\Delta(c_t)} = \rho$ lorsque $\rho^- = \{\, \mu \in M \mid \mu \ltimes c_t(x - \Delta(c_t){\cdot}S(\mu)) \,\}$; ensuite on établit les signaux comme au cas précédent $(t < t' < \Delta(c_t))$ lorsqu'il n'y a pas déjà une collision ; enfin le reste est vide, $\oslash$.

Définition 5 (Diagramme Espace-Temps, Définition Constructive)**.** Un *diagramme espace-temps défini de manière constructive* $\mathcal{D}$ est une collection de configurations au cours du temps construites de proche en proche en partant d'une configuration initiale finie et en utilisant la dynamique 4 itérée.

Remarque 1. La dynamique peut être itérée une infinité dénombrable de fois, sauf si l'on tombe à cours de collisions, auquel cas le diagramme est défini pour tout temps de $\mathbb{R}$.

Remarque 2. Cette première définition de diagramme espace-temps (5) d'une machine à signaux est celle usuelle (par ex. [32]). Elle est constructive et fournit une manière effective, à partir d'une machine et d'un configuration initiale, de tracer ce diagramme, qui est unique.

La figure 1.1 est un exemple de diagramme de machine à signaux.

Remarque 3. Cette première définition de diagramme de machine à signaux possède cependant la limitation suivante : le diagramme n'est pas défini pour tout temps postérieur (ou concourant) à la première accumulation. En particulier, la valeur d'accumulation ✺ n'est pas utilisée.

Dans la figure 5 vue en introduction et obtenue à l'aide de la machine définie à la figure 1.3, la dynamique itérée permet de définir le diagramme seulement jusqu'au temps du sommet de la pyramide (l'accumulation) exclu. En particulier, le point d'accumulation ne fait pas partie du diagramme.

La figure 1.2 est un second exemple de la même machine (1.3), partant cette fois de la configuration initiale définie à la figure 1.4.

Diagramme Espace-Temps Définition Implicite

La limitation de la première définition de diagramme de machine à signaux nous conduit à formuler une seconde définition davantage permissive.

Diverses manières de définir ce qui se passe lors et au-delà d'une accumulation sont explorées dans [12], et servent de socle au calcul dit par trou noir [19, 33, 17]. Pour l'objet de cette thèse, il suffit que rien ne se passe lors d'une accumulation et qu'elle n'empêche pas la définition du reste du diagramme. Notre définition est donc très similaire à [12] (chapitre 8).

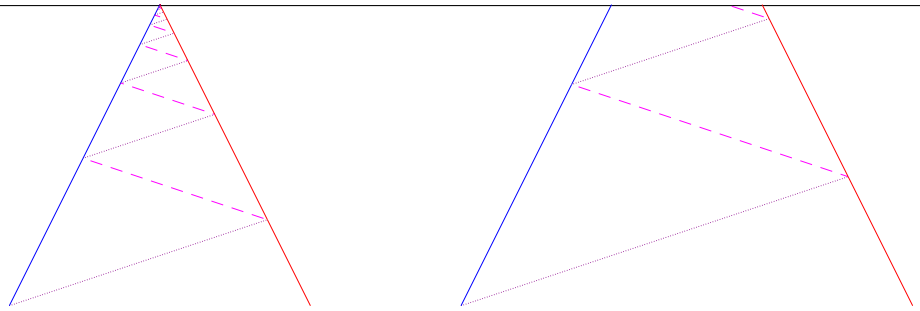


FIGURE 1.2 – Le diagramme d'une machine à signaux est défini seulement jusqu'à la première accumulation exclue.

Meta-signal	Speed	
$\overleftarrow{\text{zag}}$	-1	
$\overleftarrow{\text{ri}}$	-1/2	$\{\overrightarrow{\text{zig}},\overleftarrow{\text{ri}}\} \to \{\overleftarrow{\text{zag}},\overleftarrow{\text{ri}}\}$
$\overrightarrow{\text{le}}$	1/2	$\{\overrightarrow{\text{le}},\overleftarrow{\text{zag}}\} \to \{\overrightarrow{\text{le}},\overrightarrow{\text{zig}}\}$
$\overrightarrow{\text{zig}}$	1	

FIGURE 1.3 – Définition de la machine zig-zag.

$$\begin{cases} c(-1) = \{\overrightarrow{\text{le}},\overleftarrow{\text{zag}}\} \to \{\overrightarrow{\text{le}},\overrightarrow{\text{zig}}\} \\ c(1) \;= \overleftarrow{\text{ri}} \\ c(2) \;= \{\overrightarrow{\text{le}},\overleftarrow{\text{zag}}\} \to \{\overrightarrow{\text{le}},\overrightarrow{\text{zig}}\} \\ c(5) \;= \overleftarrow{\text{ri}} \\ c(x) \;= \oslash \qquad \text{partout ailleurs} \end{cases}$$

FIGURE 1.4 – Configuration initiale d'un zig-zag double.

Définition 6 (Signaux, Chemins et Collisions)**.** Soit $\mathcal{D}$ une *collection de configurations au cours du temps* que l'on peut encore voir, formellement, comme une application : $\mathbb{R}\times\mathbb{T} \mapsto M \cup R \cup \{\oslash, ✺\}$.

Un *signal* de $\mathcal{D}$ est un intervalle – segment ou demi-droite possiblement ouvert – maximal de l'espace-temps associé à un méta-signal μ par $\mathcal{D}$. Les méta-signaux peuvent être vus comme des types dont les signaux sont les instances. Par souci de concision, le méta-signal d'un signal est aussi appelé le type de ce signal.

Les extrémités hâtives et tardives d'un signal sont appelées, respectivement, *origine* et *fin* de ce signal.

Un *chemin de signaux* est une suite indexée par un intervalle de $\mathbb{Z}$ de signaux telle que, pour toute paire de signaux consécutifs, le premier a une fin qui est l'origine du suivant.

Une *collision* de $\mathcal{D}$ est simplement un point de l'espace-temps envoyé sur un élément de R.

Définition 7 (influence)**.** Soit $\mathcal{D}$ une collection de configurations au cours du temps. On dit d'une collision ou d'un signal A de $\mathcal{D}$ qu'elle *influence* une autre collision ou signal B lorsqu'il existe un chemin de A à B.

Définition 8 (Point d'Accumulation)**.** Soit $\mathcal{D}$ une *collection de configurations au cours du temps.*

Un *point d'accumulation de collisions* de $\mathcal{D}$ est un point de l'espace-temps au voisinage duquel se trouve une infinité de collisions (tout voisinage de ce point contient une infinité de collisions).

De la même manière, un *point d'accumulation de signaux* est un point de l'espace-temps au voisinage duquel se trouve une infinité de signaux (tout voisinage de ce point intersecte une infinité de signaux).

La figure 1.5 contient, au sommet de la pyramide, un point d'accumulation de collisions au-delà duquel se trouve toute une droite d'accumulation de signaux.

Un point d'accumulation de collisions est un point d'accumulation de signaux, l'inverse n'étant pas nécessairement vrai.

Définition 9 (Diagramme Espace-Temps, Définition Implicite)**.** Un *diagramme espace-temps défini de manière implicite* $\mathcal{D}$ est une collection de configurations au cours du temps vérifiant les conditions suivantes :

1. la configuration du temps zéro, aussi appelée *configuration initiale*, est finie ;
2. tout point de l'espace-temps associé à un méta–signal par $\mathcal{D}$ appartient à un signal de $\mathcal{D}$;
   - un signal de type μ est de pente inverse $S(\mu)$ (le temps s'écoule vers le haut, les signaux ne peuvent pas être horizontaux) ;
   - l'origine d'un signal de type μ est un point de la configuration initiale ou un point associé à une règle ρ tel que $\mu \in \rho^+$;

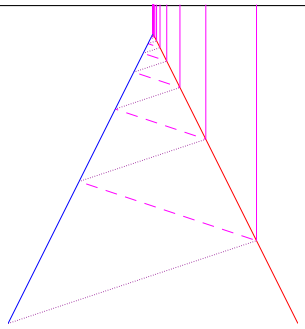


FIGURE 1.5 – Au-dessus de l'accumulation de collisions : une accumulation de signaux.

- la fin d'un tel signal, s'il en est, est associé à une règle ρ telle que $\mu \in \rho^-$ ou bien à une valeur d'accumulation ✺ ;

3. tout point de l'espace temps associé à une règle de collision ρ est :
   - l'origine d'un signal de type μ pour chaque μ dans ρ^+
   - située sur la configuration initiale ou bien la fin d'un signal de type μ pour chaque μ dans ρ^-.
4. *traçage* : pour tout signal, tout chemin dont il est le dernier élément est fini.
5. *accumulations* : un point est associé à ✺ si et seulement si c'est un point d'accumulation.

Un tel diagramme est représenté par un dessin bi-dimensionnel, avec l'axe du temps vertical et orienté vers le haut.

En reprenant la machine zig-zag définie en 1.3) et la configuration initiale définie en 1.4, le diagramme espace-temps suivant la définition implicite est tel qu'à la figure 1.6. Contrairement au diagramme selon la définition explicite (1.2), les deux pyramides sont tracées.

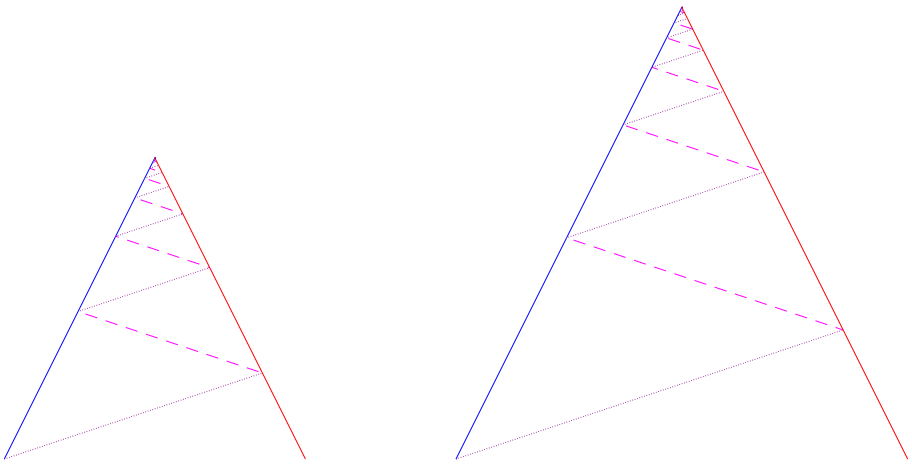


FIGURE 1.6 – La définition implicite d'un diagramme permet que celui-ci soit défini parfois même pour des temps postérieurs à une accumulation.

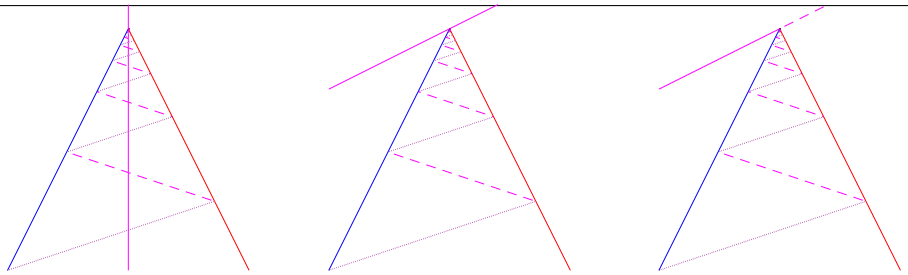
La condition de traçage permet que rien ne se passe au-delà d'une accumulation.

La figure 1.7 montre des exemples de diagrammes impossibles contenant une accumulation, construits toujours à l'aide du même exemple mais en essayant cette fois-ci de faire passer un signal par une accumulation. Les trois diagrammes sont impossibles car ils nécessiteraient qu'une accumulation, associée à la valeur ✺, soit origine d'un signal.

La figure 1.8 est un diagramme valide qui illustre l'interaction entre les signaux et les accumulations. Le sommet de la pyramide est une accumulation de collisions. Il est donc associé à ✺. Au-dessus de lui se trouve une demi-droite verticale de points d'accumulation de signaux, aussi associés à ✺. Un des points de cette demi-droite sert de fin au signal bleu le plus à gauche.

Quelques Propriétés des Diagrammes

Définition 10 (Profondeur)**.** On définit *la profondeur d'un signal* comme étant le supremum des longueurs des chemins dont ce signal est le dernier élément et dont le premier élément a son origine sur la configuration initiale. De la même manière, on définit *la profondeur d'une collision* comme le supremum des longueurs des chemins dont le dernier élément a pour fin cette collision et le premier élément son origine sur la configuration initiale.



(a) (b) (c)

FIGURE 1.7 – Diagrammes impossibles : rien ne se passe depuis un point d'accumulation.

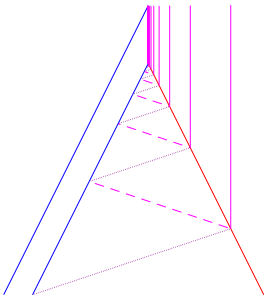


FIGURE 1.8 – Exemple de diagramme avec accumulation.

Cette notion est introduite pour travailler sur les diagrammes espace-temps définis de manière implicite, mais demeure valide pour ceux définis de manière constructive.

Remarque 4. La profondeur d'un signal ou d'une collision d'un diagramme espace-temps défini de manière constructive ou implicite est finie.

Démonstration. Si le diagramme espace-temps est défini de manière constructive, la preuve est triviale : la profondeur d'un signal ou d'un collision au temps t est bornée par le nombre d'itérations de la dynamique qu'il a fallu pour définir le diagramme au temps t.

Pour un diagramme espace-temps défini de manière implicite, c'est une conséquence de la condition de traçage. Intuitivement, celle-ci dit qu'il n'y a pas de chemin inverse de longueur infinie, et il s'agit de prouver qu'il n'y en a pas non plus de longueur arbitrairement grande.

On remarque d'abord que :

— la profondeur d'une collision est égale au maximum des profondeurs des signaux entrants (qui sont en nombre fini) ;

— la profondeur d'un signal sortant d'une collision est égale à 1 plus la profondeur de cette collision.

Supposons qu'il existe un diagramme espace-temps défini implicitement $\mathcal{D}$ dans lequel un signal s est de profondeur infinie. Alors son origine est une collision de profondeur infinie, qui possède un signal entrant de profondeur infinie. On peut donc, par récurrence, construire un chemin de signaux infini de dernier élément s, ce qui contredit la condition de traçage.

Cela prouve que les signaux sont de profondeur finie et, par conséquent, que les collisions le sont aussi. □

Remarque 5. L'ensemble des signaux d'une certaine profondeur d'un diagramme espace-temps défini de manière constructive ou implicite est fini.

En particulier, l'ensemble des signaux et collisions d'un diagramme espace-temps est au plus dénombrable, et on peut les lister – les mettre en bijection avec tout ou partie de $\mathbb{N}$– par profondeur croissante.

En effet, si il y a n le nombre de collisions de profondeur au plus p, il y a au plus $k \times n$ collisions de profondeur $p+1$, où k est la taille maximale des sortie de règle de collision.

Lien Entre les deux Définitions de Diagramme Espace-Temps

Lemme 1 (Causalité fermante)**.** *Soit une machine à signaux, et soient c^+ la vitesse maximale et c^- la vitesse minimale des méta-signaux de cette machine. Soit $(x_f, t_f), t_f > 0$ les coordonnées d'un point de l'espace-temps et soit $\mathcal{D}$ et $\mathcal{D}'$ deux diagrammes espace-temps définis et sans accumulation en (x_f, t_f).*

La valeur de $\mathcal{D}(x_f, t_f)$ est entièrement déterminée par la valeur de $\mathcal{D}$ au voisinage de (x_f, t_f) dans le cône de lumière $\mathcal{C} = \{(x,t) \in \mathbb{R}\times\mathbb{T} | t < t_f, x_f - c^- * (t_f - t) \leq x \leq x_f - c^+ * (t_f - t)\}$.

En particulier, s'il y a un voisinage $\mathcal{V}$ de (x_f, t_f) tel que $\mathcal{D}$ et $\mathcal{D}'$ coïncident sur $\mathcal{V} \cap \mathcal{C}$, alors $\mathcal{D}(x_f, t_f) = \mathcal{D}'(x_f, t_f)$

Démonstration. Considérons les signaux entrant dans (x_f, t_f), c'est-à-dire les signaux de $\mathcal{D}$ restreints aux temps $[0\,;t_f[$ ayant pour extrémité (ouverte) (x_f, t_f). Ils sont en nombre fini, puisqu'ils ne peuvent pas se superposer et que leur vitesse est dans un ensemble fini. Ils sont aussi dans le cône $\mathcal{C}$. Ils sont encore des signaux entrant dans (x_f, t_f) lorsque $\mathcal{D}$ est aussi restreint à un voisinage de (x_f, t_f).

Traitons le cas où $\mathcal{D}$ est défini de manière implicite, en se référant à la définition 9. Le second tiret de la partie 3 de la définition peut se reformuler ainsi : hors configuration initiale, l'ensemble des types de signaux entrants dans un point associé à une règle de collisions ρ vaut ρ^-. La partie 2 de la définition (début et premier tiret) implique qu'un point hors configuration initiale associé à un méta-signal μ a un signal entrant de type μ. Le troisième tiret implique qu'il n'y a pas d'autres signaux entrant, autrement ce point associé à μ serait une fin invalide de ces signaux. De la même manière, un point associé à $\oslash$ n'a aucun signal entrant, sinon il en serait une fin invalide.

On vient de montrer que la nature d'un point détermine l'ensemble des signaux entrants. Cette détermination étant bijective (déterminisme de l'ensemble des règles de collision), on peut conclure que l'ensemble des signaux entrants, et donc la valeur du diagramme $\mathcal{D}$ restreint $\mathcal{C}$ et dans un voisinage de (x_f, t_f) détermine la valeur $\mathcal{D}(x_f, t_f)$.

Le cas où $\mathcal{D}$ est défini de manière constructive (définition 5) peut se traiter de manière similaire. □

Lemme 2. *Soit c une configuration initiale d'une machine à signaux, $\mathcal{D}$ son diagramme espace-temps défini constructivement, et $\mathcal{D}'$ un diagramme espace-temps défini de manière implicite. $\mathcal{D}'$ n'a pas de collision de temps compris strictement entre 0 et $\Delta(c)$.*

Démonstration. Par l'absurde, soit (x, t) les coordonnées d'une collision de profondeur minimale parmi celles de temps compris strictement entre 0 et $\Delta(c)$.

De deux choses l'une :

— ou bien cette profondeur minimale est supérieure ou égale à 2, auquel cas cette collision possède un antécédent de profondeur au moins 1, donc de temps compris strictement entre 0 et $\Delta(c)$, ce qui contredit la minimalité de cette profondeur ;

— ou bien cette profondeur minimale vaut 1, mais alors les signaux entrants de cette collision sont issus de la configuration initiale ; les vitesses de méta-signaux étant utilisées de manière consistante d'une définition de diagramme à l'autre, cela implique $\Delta(c) \leq t$, ce qui est absurde.

□

Corollaire 3. *Avec les notations du lemme précédent, les signaux de $\mathcal{D}'$ ayant un point dans l'intervalle de temps $]0\,;\Delta(c)[$ ont leur origine en 0, et leur éventuelle fin au-delà de $\Delta(c)$.*

Lemme 4 (Causalité ouvrante)**.** *Soient c une configuration initiale d'une machine à signaux, $\mathcal{D}$ son diagramme espace-temps défini constructivement, et $\mathcal{D}'$ un diagramme espace-temps défini de manière implicite. $\mathcal{D}$ et $\mathcal{D}'$ coïncident pour les temps inférieurs strictes à $\Delta(c)$.*

Démonstration. Soit (x, t) un point de $\mathbb{R} \times]0\,;\Delta(c)[$. D'après le lemme 2, $\mathcal{D}'(x, t)$, tout comme $\mathcal{D}(x, t)$, est un méta-signal ou le vide (est un élément de $M \cup \{\oslash\}$).

Si $\mathcal{D}'(x, t) \in M$, alors le point (x, t) appartient à un signal de $\mathcal{D}'$ (point 2 de la définition implicite de diagramme), d'origine dans la configuration initiale (corollaire 3), et on peut montrer, par définition de la dynamique constructive, qu'un signal de même type passe par (x, t) dans $\mathcal{D}$, autrement dit $\mathcal{D}(x, t) = \mathcal{D}'(x, t)$.

Si $\mathcal{D}(x, t) \in M$, alors le point (x, t) appartient à un signal de $\mathcal{D}$, d'origine dans la configuration initiale (par construction). Cette origine appartient à ou est à l'origine d'un signal du diagramme $\mathcal{D}'$, selon les points 2 et 3 de la définition implicite de diagramme (selon que cette origine est envoyée sur un méta-signal ou une règle). Ce signal de $\mathcal{D}'$ est ininterrompu jusqu'au moins au temps t (corollaire 3), et on peut montrer qu'il passe par (x, t), autrement dit $\mathcal{D}'(x, t) = \mathcal{D}(x, t)$.

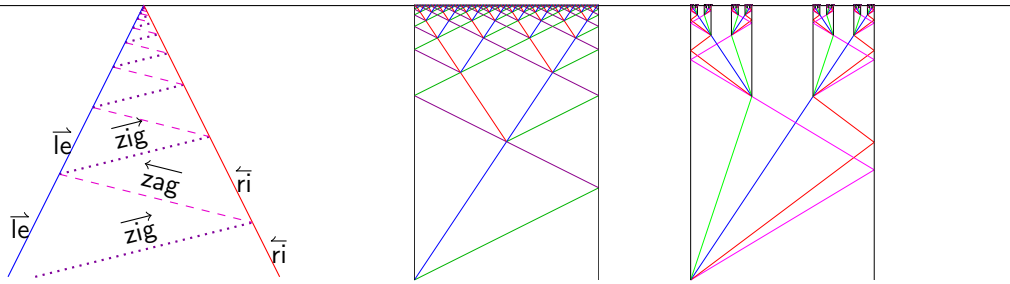
Cela suffit à conclure que $\mathcal{D}(x, t) = \mathcal{D}'(x, t)$. □

Corollaire 5 (équivalence des définitions)**.** *En combinant les lemmes 4 et 1, on obtient que deux diagrammes d'un même couple machine-configuration initiale, l'un défini de manière constructive, l'autre implicite, coïncident le temps d'une dynamique.*

Une récurrence simple suffit à montrer qu'ils coïncident jusqu'au temps de définition maximale du diagramme constructif.

Conjecture 6. *Toute configuration initiale c de machine à signaux admet un unique diagramme espace-temps défini de manière implicite.*

Dans le reste de cette thèse, lorsqu'il s'agit d'énoncer des contraintes sur ce que peuvent faire les machines à signaux, nous partons donc de diagrammes dont on suppose l'existence et non de configurations initiales. Lorsqu'il s'agit de montrer que les machine à signaux peuvent faire telle ou telle chose, nous construisons des machines et configurations initiale idoines, et l'existence du diagramme correspondant découle de la définition particulière de la machine et de la configuration initiale considérée.



(a) accumulation isolée (b) segment d'accumulation (c) accumulation sur un espace de Cantor

FIGURE 1.9 – Diagramme espace-temps et accumulations.

1.1.2 Ensembles et Courbes d'Accumulation

Un *ensemble d'accumulation* d'un diagramme espace-temps est simplement l'ensemble des points d'accumulation de ce diagramme. On dit encore d'un diagramme de machine à signaux qu'il *s'accumule* sur un ensemble lorsque celui-ci est son ensemble d'accumulation.

Une *courbe d'accumulation* est un ensemble d'accumulations qui est une courbe, c'est-à-dire en bijection continue avec un intervalle de $\mathbb{R}$.

Remarque 6. Un ensemble d'accumulation d'un diagramme $\mathcal{D}$ est fermé. En effet, un point qui n'est pas d'accumulation possède un voisinage ouvert avec un nombre fini de collisions, donc sans accumulation. Le complémentaire d'un ensemble d'accumulation est donc ouvert.

1.1.3 Machine à Signaux Augmentés

Les *machines à signaux augmentés*, abrégées MASA, sont définies comme les machine à signaux avec les différences suivantes :

— Chaque méta-signal est associé à un domaine d'information que peuvent porter les signaux qui le représentent. Ce domaine est une partie de $\mathbb{R}^n$ ou n est un entier. Un signal portant un telle information est appelé *signal augmenté*.

— Les règles de collision dépendent de et font changer cette information. Seuls les calculs linéaires sont autorisés.

Les machines à signaux sont des cas particuliers de machines à signaux augmentés où le domaine associé à chaque méta-signal est un singleton – $\mathbb{R}^0$.

1.1.4 Arbre Unaire-Binaire

Dans le contexte de cette thèse, un *arbre unaire-binaire* est un arbre enraciné, potentiellement infini, d'arité maximale 2 et d'enfants différenciés (un sommet ayant deux enfants : un fils gauche et un fils droit).

Définition 11 (profondeur, hauteur)**.** On appelle *profondeur* d'un sommet d'un arbre la longueur du chemin, en nombre d'arêtes, qui le lie à la racine.

On appelle *hauteur* d'un arbre la profondeur maximale de ses sommets

Définition 12 (2-profondeur)**.** On appelle *2-profondeur* d'un sommet d'un arbre unaire-binaire son nombre d'ancêtres de degré sortant 2.

On appelle *2-hauteur* d'un arbre la 2-profondeur maximale de ses sommets.

Définition 13 (sous-arbre enraciné)**.** Soit G un arbre. Un *sous-arbre* de G est un arbre obtenu en sélectionnant un sommet de G et tous ses descendants.

Définition 14 (sous-arbre préfixe)**.** Soit G un arbre. Un *arbre préfixe* de G est un arbre obtenu à partir de G en enlevant des sommets.

Remarque 7. Une suite d'arbres finis, croissante au sens préfixe, peut permettre de définir un arbre infini, par union croissante.

Remarque 8. Réciproquement, tout arbre peut s'écrire comme limite croissante d'arbres préfixes finis.

Il suffit de prendre par exemple les réciproques par la fonction profondeur des segments initiaux de $\mathbb{N}$, c'est-à-dire les arbres dont les sommets sont les sommets de G sont de profondeur au plus n.

On note *UB* l'ensemble des arbre unaires-binaires.

Définition 15 (tracé)**.** Le *tracé* d'une arbre unaire-binaire est une sous-partie du demi-plan Euclidien $\mathbb{R}\times\mathbb{T}$ défini récursivement de la manière suivante :

— la racine est aux coordonnées $(0, 0)$;

— si la racine n'a pas d'enfant, le reste du demi-plan n'est pas dans l'arbre ;

— si la racine est de degré 1, son enfant se trouve aux coordonnées $(0, 1)$, et le segment le reliant à son père fait aussi partie du tracé ; le tracé contient aussi le tracé du sous-arbre enraciné en ce fils translaté par le vecteur $(0, 1)$;

— si la racine est de degré 2, ses fils gauche et droit se trouvent respectivement aux coordonnées $(-1/2, 1/2)$ et $(1/2, 1/2)$; les segments les reliant à leur père font aussi partie du tracé ; le tracé contient aussi les tracés des sous-arbres enracinés aux fils gauche et droit, chaque tracé étant translaté, respectivement, par le vecteur $(-1/2, 1/2)$ et $(1/2, 1/2)$, puis réduit de moitié par une homothétie de centre respectivement $(-1/2, 1/2)$ et $(1/2, 1/2)$.

Définition 16 (accumulation)**.** Par analogie aux machines AGC, on parle d'*accumulation* d'un tracé d'arbre-binaire lorsqu'un point est voisinage d'une infinité de sommets. On parle d'*ensemble d'accumulation* pour désigner l'ensemble de ces tels points.

1.2 MASA vers MAS : Outils de Passage

Il est possible de simuler une machine à signaux augmentés par une machine à signaux. La présente section décrit des outils pour le faire, sans aller jusqu'à une preuve formelle de l'équivalence.

1.2.1 Encodage

Un signal augmenté peut être représenté par une bande de signaux normaux, l'un marquant l'emplacement précis du signal augmenté encodé, et d'autres dont les écarts encodent des composantes de l'information. De manière informelle, on appelle une telle bande de signaux un macro-signal.

La figure 1.10 illustre une manière concrète de procéder : le signal main permet de localiser l'emplacement précis du signal augmenté encodé tandis que la distance entre les signaux x_i^0 et x_i^v encode x_i. L'ordre des signaux importe : x_2^v et x_2^0 sont inversés et codent ainsi une valeur négative pour x_2. Enfin, une superposition de signaux, c'est-à-dire un tiers méta-signal $x_i^{v=0}$, permet d'encoder une valeur nulle. Dans notre exemple, le signal $x_3^{v=0}$ remplace une paire de signaux de type x_3^0 et x_3^v et signifie $x_3 = 0$.

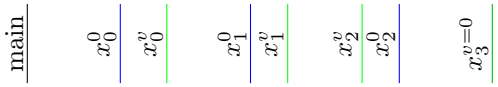


FIGURE 1.10 – Encodage d'un signal augmenté.

1.2.2 Calcul

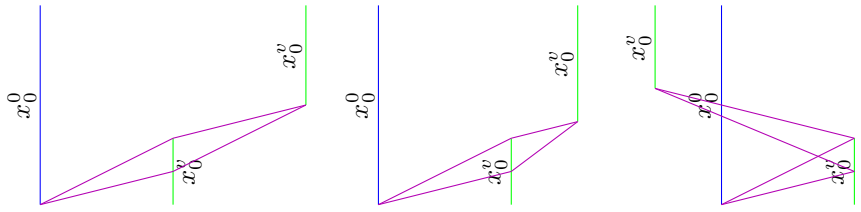
Doubler une Valeur

Il est possible de doubler une valeur en dessinant un losange tel que montré dans la figure 1.11a.

La figure 1.11b montre qu'en changeant certaines vitesses de signaux, ici celle du signal inférieur droit, il est possible de multiplier par n'importe quelle valeur supérieure à 1.

Avec des signaux de vitesse opposée, il est possible de multiplier n'importe quelle valeur par une valeur inférieure à 1, comme l'illustre la figure 1.11c.

Par homothétie, chaque jeu de vitesses correspond à une seule constante multiplicative – indépendante de la longueur qu'on multiplie.



(a) doublage d'une valeur (b) multiplication d'une valeur par 3/2 (c) multiplication d'une valeur par $-1/2$

FIGURE 1.11 – Multiplications de valeurs.

Addition

De manière très similaire au doublement d'une valeur, on peut additionner une valeur positive à une autre valeur, par tracé d'un parallélogramme, tel que présenté dans la figure 1.12.

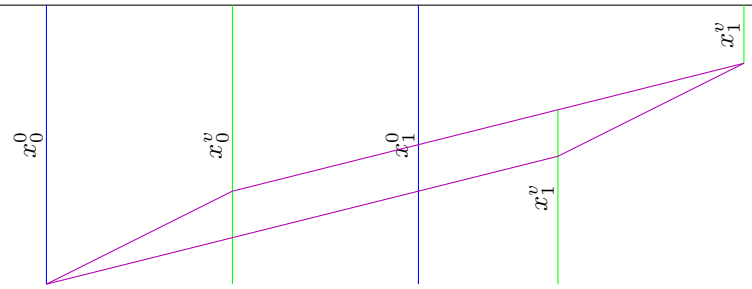


FIGURE 1.12 – Addition.

Soustraction

De manière très similaire à l'addition, on peut soustraire une valeur positive à une autre valeur, en changeant le signe des vitesses de certains signaux, tel qu'illustré par la figure 1.13.

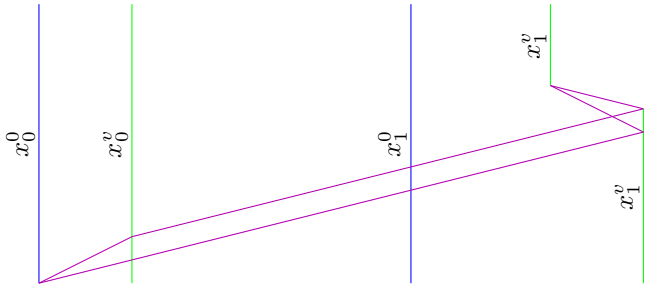
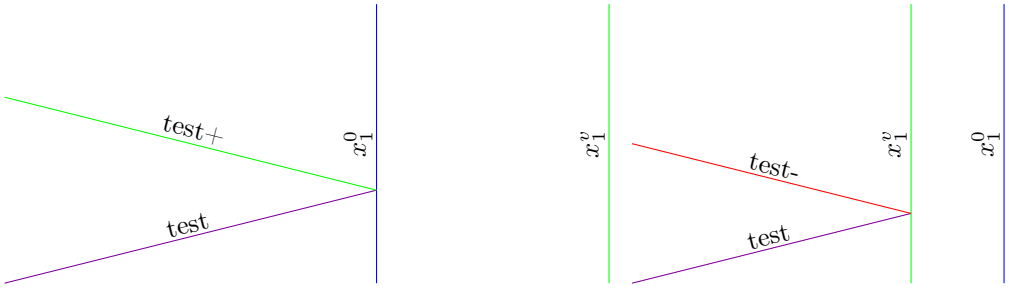


FIGURE 1.13 – Soustraction.

Test

Tester le signe d'une valeur est simplement une affaire de voir quel signal, entre x_i^0, x_i^v et $x_i^{v=0}$, est rencontré en premier, tel qu'illustré par la figure 1.14.



(a) valeur positive (b) valeur négative

FIGURE 1.14 – Test de signe d'une valeur.

Un test peut être combiné avec les techniques précédentes pour effectuer des additions et des soustractions avec des valeurs de signes arbitraires.

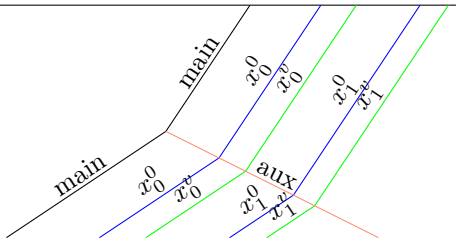
1.2.3 Redirection

Lorsque plusieurs macro-signaux se rencontrent, une *macro-collision* doit avoir lieu, pour simuler la collision de signaux augmentés. En plus des calculs mentionnés précédemment, ces macro-collisions peuvent nécessiter duplications et redirection d'information. Une manière de procéder est d'effectuer un changement de direction des signaux à leur rencontre d'un signal auxiliaire.

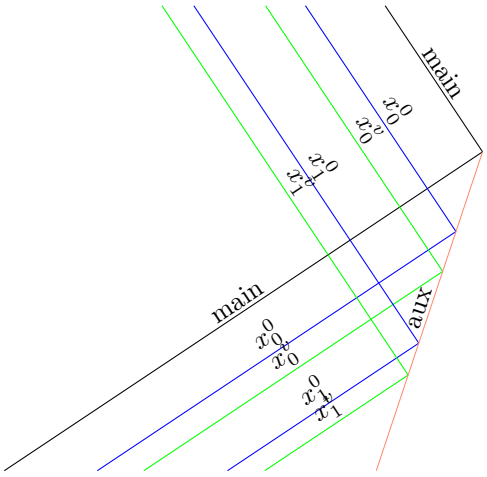
On parle de `réfraction` lorsque le macro-signal traverse cette barrière, comme illustré par la figure 1.15a.

On parle de `réflexion` lorsque les signaux résultant sont du même côté de cette barrière que les signaux entrant, comme illustré par la figure 1.15b.

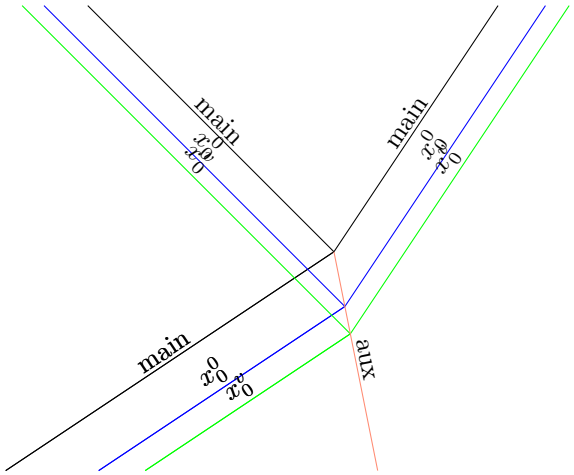
La figure 1.15c montre que réflexion et réfraction peuvent être combinées pour dupliquer un macro-signal. Plus généralement, on peut combiner une ou plusieurs réflexions et une ou plusieurs réfractions afin de multiplier un macro-signal.



(a) une réfraction



(b) une réflexion



(c) une réfraction et une réflexion utilisées de conserve pour dupliquer les macro-signaux.

FIGURE 1.15 – Redirection de macro-signaux.

Contraction

Les macro-signaux doivent être suffisamment étroits pour le calcul de macro-collisions, et ce pour deux raisons. La première est que les calculs arithmétiques prennent du temps, proportionnellement à la largeur des macro-signaux dans lesquels ils ont lieu ; réduire la taille des macro-signaux à des moments opportuns est une manière d'assurer que le résultat des calculs est disponible à temps. La seconde raison est d'éviter les interférences entre macro-signaux voisins ; lors d'une macro-collision, les macro-signaux se rencontrent et leur étroitesse devient nécessaire pour savoir quels macro-signaux doivent interagir ou non, et pour s'assurer que ceux ne devant pas interagir ne s'entremêlent pas.

La notion de « suffisamment étroit » est donc dynamique et requiert une étroitesse arbitraire. Il faut donc savoir contracter un macro-signal, avec respect des proportions. Cela peut être fait par exemple à l'aide de deux `réfractions` successives, comme l'illustre la figure 1.16.

Quant au déclenchement automatique de telles contractions, il est très simple pour les machines étudiées dans cette thèse et est abordé dans les chapitres ultérieurs. Un traitement plus général qui permettrait de traduire n'importe quelle machine à signaux augmentée en machine à signaux est possible, et est, en substance, accompli dans [32].

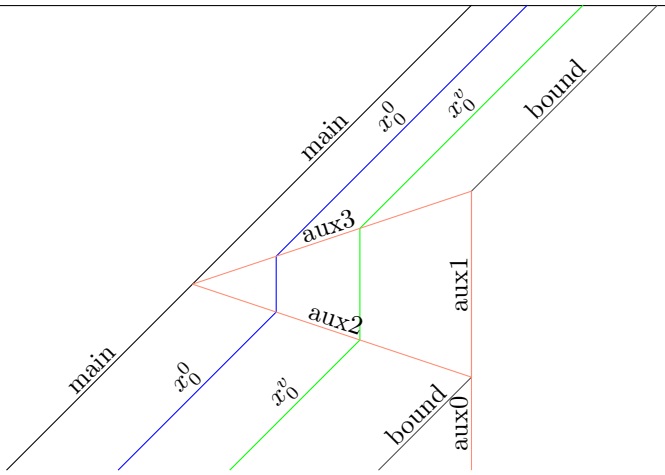


FIGURE 1.16 – Une contraction de macro-signaux.

1.3 Définition des Problématiques Abordées

1.3.1 Accumulation sur une Fonction Continue

On dit qu'un diagramme ou tracé *s'accumule* sur une fonction f continue lorsque son ensemble d'accumulation coïncide avec le graphe de cette fonction.

Un premier type de problème est le suivant : étant donnée une fonction continue f, existe-t-il une machine à signaux et une configuration initiale admettant un diagramme s'accumulant sur cette fonction ? Un exemple de ce type de problème est la Synchronisation d'une Ligne de Fusiliers.

Ainsi que le type de problème converse : étant donnée une machine à signaux et une configuration initiale, admettent-elles un diagramme s'accumulant sur une fonction ?

Ce qui nous intéresse est la forme locale des fonctions atteignables, et non une quelconque notion de vitesse ou d'optimalité comme ce peut être le cas par exemple pour les travaux traitant du problème de synchronisation d'une ligne de fusiliers [24, 25, 26].

On s'intéresse donc à l'accumulation, à homothétie près, sur des fonctions. On ne perd rien à considérer, au besoin, les fonctions bornées par une borne inférieure suffisamment grande. On perd peu à considérer les fonctions aux variations suffisamment faibles, définies sur des segments *ad hoc*.

1.3.2 Accumulation sur une Classe de Fonction, Universalité

Pour monter en généralité, on peut se poser la question suivante : étant donnée une classe de fonctions, existe-t-il, pour chacun de ses éléments, un couple machine-configuration initiale admettant un diagramme s'accumulant dessus ?

Le problème converse est le suivant : étant donné une classe de machines à signaux et de configurations initiales, quelle est la classe de fonctions sur laquelle s'accumulent les diagrammes correspondants ?

On s'intéresse aussi au problème d'universalité suivant : étant donné une classe de fonctions, existe-t-il une machine qui, en faisant varier la configuration initiale, produit des diagrammes s'accumulant sur chaque fonction ?

Le problème de Synchronisation en Saccade d'une Ligne de Fusiliers, évoqué en introduction, traité par [30] et repris au chapitre 2 en est un exemple, pour la classe des fonctions affines.

1.3.3 Contraintes de Rationalité et de Quantité d'Information en Entrée

L'espace étant la droite réelle, on peut stocker une grande quantité d'information dans une configuration initiale, comme par exemple une suite de $\mathbb{R}^{\mathbb{N}}$ encodant le résultat d'une fonction non calculable, ou directement une droite continue de $\mathbb{R}^{\mathbb{Q}}$ (chapitre 4).

On décline donc aussi les questions précédentes en contraignant la configuration initiale et la machine à être rationnelles.

Chapitre 2

Problème de Synchronisation en Saccade d'une Ligne de Fusiliers

On peut désormais, avec les outils de la section précédente, expliquer le premier exemple de Synchronisation en Saccade d'une Ligne de Fusilier. On a un algorithme qui trace dans l'espace un arbre unaire-binaire dont la frontière est le segment ciblé. Une fois traduite en machine à signaux, cette frontière devient un ensemble d'accumulation, comme désiré.

2.1 Algorithme

2.1.1 Présentation de l'Algorithme

L'algorithme prend en entrée deux paramètres l et r tous deux supérieurs à 1, et renvoie en sortie le dessin d'un arbre unaire-binaire qui dessine le segment $[(-1, l-1)\,;(1, r-1)]$, appelé *segment cible*, au sens suivant : ce segment est exactement l'adhérence de l'ensemble des sommets de l'arbre, privée de l'ensemble des sommets.

Par abus et analogie avec les machines à signaux, on appelle *ensemble d'accumulation de l'arbre* l'adhérence de l'ensemble des sommets de l'arbre, privée de l'ensemble des sommets. De manière équivalente, c'est l'ensemble des points au voisinage desquels il y a une infinité de sommets. Pareillement, on dit que *l'arbre s'accumule* sur son ensemble d'accumulation.

Le but de l'algorithme et la paramétrisation du segment cible sont illustrés par la figure 2.1a.

L'algorithme, récursif, repose sur une stratégie "diviser pour régner" simple : pour s'accumuler sur le segment $[(-1, l)\,;(1, r)]$, il suffit, selon la situation, de savoir :

— soit accumuler sur le segment $[(-1, l-1)\,;(1, r-1)]$ translaté d'une unité de temps vers le haut ;

— soit accumuler sur les segments $[(-1, l-1/2)\,;(0, (l+r)/2-1/2)]$ et $[(0, (l+r)/2-1/2)\,;(1, r-1/2)]$ translatés par les vecteurs $(-1/2, 1/2)$ et $(1/2, 1/2)$ respectivement.

En utilisant le premier cas tant que possible (l et r suffisamment grands, c'est-à-dire plus grands que 2), et en opérant deux changements de repère pour le second cas de sorte que les segments tracés soient toujours d'abscisses extrêmes -1 et 1, on obtient l'algorithme 1.

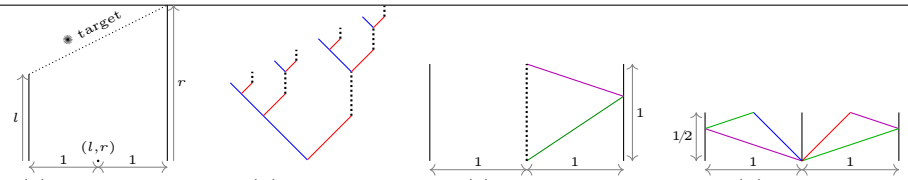
La figure 2.1b illustre cette récurrence infinie. Les segments verticaux en pointillés épars noir servent à effectuer une translation de 1 vers le haut. Une telle étape d'attente sera désormais désignée un `delay`. Par extension, les sommets d'arité 1 d'un arbre unaire-binaire seront aussi appelés des *sommets* `delay`. Sur la figure, cela correspond aux segments diagonaux de pente ± 1, bleu ou rouge selon qu'ils vont vers la gauche ou la droite (en montant) respectivement. Une telle étape de branchement sera désormais désignée un `split`.

On définit des primitives pour guider le tracé de l'algorithme.

La primitive [`delay` (l, r)]. Le sommet courant est de degré 1. Son unique fils est une unité d'espace au-dessus de lui, représenté par une connexion en pointillés longs dans la figure 2.1b. De cet enfant part l'algorithme avec les paramètres (l, r).

La primitive [`split` $((l, r),(l', r'))$]. Le sommet courant est de degré 2. L'enfant gauche est à la position $(-1/2, 1/2)$ relative au nœud courant, l'enfant droit à $(1/2, 1/2)$ – les connexions resp. bleues et rouges de la figure 2.1b. À chaque enfant, l'algorithme est appelé avec l'échelle d'espace et de temps réduite de moitié, et les paramètres (l, r) pour l'enfant gauche, et (l', r') pour le droit.

On peut donc, à l'aide de ces primitives, traduire plus précisément l'idée "diviser pour régner" en l'algorithme 1, initialisé avec les valeurs l_0 et r_0 correspondant au segment ciblé. On attend tant que l'on peut, c'est-à-dire tant que les deux extrémités sont à hauteur 2 ou plus, et on branche sinon. Le plus bas segment que l'on puisse atteindre avec ces primitives est le segment horizontal de hauteur 1, le plus bas que l'on puisse atteindre après un `delay` est celui de hauteur 2.



(a) paramètres (b) arbre (c) `delay` (d) `split`

FIGURE 2.1 – Paramètres, arbre unaire-binaire et étapes élémentaires.

Listing 1 Code for augmented collision rules.

```
if 2 <= l and 2 <= r :
        delay ( l - 1 , r - 1 )
else :
        split ( ( l + l - 1 , r + l - 1 ) ,
                ( l + r - 1 , r + r - 1 ) )
```

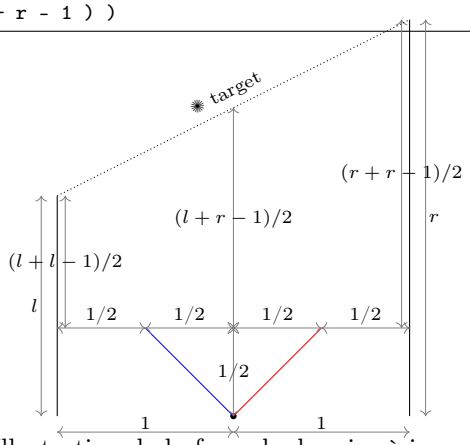


FIGURE 2.2 – Illustration de la formule de mise à jour après un `split`.

La figure 2.2 explique l'obtention de la formule de mise à jour des paramètres après un `split` : les hauteurs sont réduites de moitié, puis multipliées par 2 pour tenir compte du nouveau facteur d'échelle.

Cet algorithme décrit une récurrence infinie : l'arbre unaire-binaire construit est infini pour produire à sa frontière le segment cible.

2.1.2 Correction de l'Algorithme

Les sommets `split` jouent un rôle clef dans la démonstration de la correction de l'algorithme. On rappelle que la *2-profondeur* d'un sommet est son nombre d'ancêtres de degré sortant 2.

Lemme 7 (Invariants)**.** *L'Algorithme 1 satisfait les invariants suivants :*

- $1 \leq l$ *et* $1 \leq r$ *(les paramètres restent valides) ;*
- $(r - l)/2$ *(la pente est préservée) ;*
- *les points-frontières du segment ciblé courant appartiennent au segment ciblé initial : notons* l_0 *et* r_0 *les paramètres initiaux, et en considérant un sommet de 2-profondeur* d *et de coordonnées* (x, t)*, avec paramètres* l *et* r *; alors les points extrêmes de coordonnées* $(x, t) + 2^{-d}(-1, l)$ *et* $(x, t) + 2^{-d}(+1, r)$ *sont sur le segment initialement ciblé* $[(-1, l_0) ; (1, r_0)]$*.*

Démonstration. Les deux premiers invariants découlent directement et facilement de l'algorithme 1.

Prouvons le troisième et dernier invariant par induction.

Il est vrai pour la racine.

Soit un sommet de 2-profondeur d et de coordonnées (x, t) avec paramètres l, r (≥ 1).

Si c'est un sommet `delay`, on a $(x - 2^{-d}, (t + 2^{-d}) + 2^{-d}(l - 1)) = (x - 2^{-d}, t + 2^{-d}l)$, et pareillement pour r. Les extrémités du segment ciblé demeurent donc inchangées.

Si c'est un sommet `split`, on a $(x - 2^{-(d+1)} - 2^{-(d+1)}, (t + 2^{-(d+1)}) + 2^{-(d+1)}.2.(l - 1/2)) = (x - 2^{-d}, t + 2^{-d}l)$. L'extrémité gauche de la nouvelle branche gauche coïncide donc avec l'extrémité gauche du segment précédemment ciblé.

D'autre part, on a $(x - 2^{-(d+1)} + 2^{-(d+1)}, (t + 2^{-(d+1)}) + 2^{-(d+1)}.2.((l + r)/2 - 1/2) = (x, t + 2^{-d}(l + r)/2)$. L'extrémité droite de la nouvelle branche gauche se trouve donc au milieu du segment précédemment ciblé.

La preuve pour la branche droite est la même, ce qui conclut la preuve de cet invariant. □

Lemme 8. *Un sommet* `split` *de 2-profondeur* d *et de coordonnées* (x, t) *a lieu en-dessous du point du segment ciblé de même abscisse, mais en est distant de moins de* $2^{-d}(2 + |\alpha|)$*. En d'autre termes :*

$$t + 2^{-d}(l + r)/2 - 2^{-d}(2 + |\alpha|) \;\leq\; t \;\leq\; t + 2^{-d}(l + r)/2 \;.$$

Démonstration. La traduction de la première partie du lemme en l'inégalité de droite découle du troisième point du lemme précédent 7 ; l'inégalité elle-même est évidente.

Pour celle de gauche, observons que : $(l+r)/2 = \min\{l,r\} + |\alpha|$. En effet :

$$\begin{aligned} 2\min\{l,r\} + |r-l| &= 2\min\{l,r\} + \max\{l,r\} - \min\{l,r\} \\ &= \min\{l,r\} + \max\{l,r\} = l + r \ . \end{aligned}$$

Nous pouvons alors écrire :

$$t = t + 2^{-d}(l+r)/2 - 2^{-d}(\min\{l,r\} + |\alpha|) \ .$$

Finalement, puisque c'est un sommet `split`, $\min\{l,r\} \leq 2$, ce qui donne l'inégalité voulue. □

Remarque 9. N'importe quelle branche infinie de l'arbre contient une infinité de sommets `split`. *A fortiori*, n'importe quel sommet a un descendant de type `split`.

En effet, depuis n'importe quel sommet de paramètres l, r, il y a un nombre fini, égal à $\min\{l,r\} - 1$, de sommets `delay` avant un sommet `split`.

Theorème 9 (Correction de l'Algorithme de Tracé d'un Segment Paramétré.)**.** *Étant donné des paramètres d'entrée l_0 et r_0 supérieurs ou égaux à 1, l'algorithme trace un arbre s'accumulant sur le segment $[(1, r_0) ; (-1, l_0)]$*

Démonstration. Soit $x \in [-1 ; 1]$. Notre but est de prouver qu'il y a une accumulation en $(x, (l_0 + r_0)/2 + \alpha x)$.

Remarquons que l'ensemble des abscisses de sommets `split` est exactement l'ensemble des nombres dyadiques compris strictement entre -1 et 1, qui est dense dans $[-1 ; +1]$. En effet, ces abscisses sont de la forme $\Sigma_{i=1}^{d} a_i 2^{-i}$ où $(a_i)_{i\in[0,d]}$ est une suite finie arbitraire d'éléments de $\{+1, -1\}$. Si on note x un tel nombre, $(x+1)/2$ peut s'exprimer de la sorte : $\frac{1}{2}(1 + \Sigma_{i=1}^{d} a_i 2^{-i}) = \frac{1}{2}(\Sigma_{i=1}^{\infty} 1 \times 2^{-i} + \Sigma_{i=1}^{d} a_i 2^{-i}) = \frac{1}{2}(\Sigma_{i=1}^{d}(1+a_i) \times 2^{-i} + \Sigma_{i=d+1}^{\infty} 1 \times 2^{-i}) = \Sigma_{i=1}^{d} \frac{(1+a_i)}{2} \times 2^{-i} + \Sigma_{i=d+2}^{\infty} 1 \times 2^{-i}$, ce qui est un développement binaire impropre quelconque d'un nombre dyadique de $]0 ; 1[$.

Il existe donc une suite de coordonnées de sommets $(x_d, t_d)_{d\in\mathbb{N}}$ telle que :

— pour tout entier d, les coordonnées (x_d, t_d) sont celles d'un sommet `split` de 2-profondeur d ;

— $|x_d - x| < 2^{-d}$.

En particulier, la suite des abscisses $(x_d)_{d\in\mathbb{N}}$ est convergente, de limite x.

On a ensuite :

$$\begin{aligned} |t_d - ((l_0+r_0)/2 + \alpha x)| &\leq |t_d - ((l_0+r_0)/2 + \alpha x_d)| \\ &\quad + |((l_0+r_0)/2 + \alpha x_d) - ((l_0+r_0)/2 + \alpha x)| \\ &\leq |t_d - ((l_0+r_0)/2 + \alpha x_d)| + |\alpha(x_d - x)| \\ &\leq |t_d - ((l_0+r_0)/2 + \alpha x_d)| + |\alpha| \times 2^{-d} \end{aligned}$$

En combinant les lemmes 8 et 7, on obtient :

$$-2^{-d}(2 + |\alpha|) \leq t_d - ((l_0+r_0)/2 + \alpha x_d) \leq 0$$

Ce qui implique :

$$|t_d - ((l_0+r_0)/2 + \alpha x_d)| \leq 2^{-d}(2 + |\alpha|) \ .$$

Enfin :

$$|t_d - (l_0+r_0)/2 + \alpha x| \leq 2^{-d}(2 + |\alpha| + |\alpha|)x \xrightarrow[d\to+\infty]{} 0 \ .$$

Ce qui finit de prouver que (t_d) est convergente de limite $(l_0 + r_0)/2 + \alpha x$. Cela suffit à prouver que $(x, (l_0 + r_0)/2 + \alpha x)$ est bien un point d'accumulation, et donc que le segment ciblé est bien inclus dans l'ensemble d'accumulation.

Les points strictement au-dessus du segment ciblé ne peuvent être des points d'accumulation, car il découle du lemme 8 et de la remarque 9 qu'aucun point de l'arbre n'est strictement au-dessus du segment ciblé.

Soit un point strictement en-dessous du segment cible, par une distance Δt. D'après le lemme 8, il y a un nombre d suffisamment grand de sorte que tout sommet de 2-profondeur supérieur à d est à au plus $\Delta t/2$ unités de distance en-dessous du segment d'accumulation. Il est ensuite aisé de construire un voisinage du point considéré ne pouvant contenir que des sommets de 2-profondeur au plus d, qui sont en nombre fini – remarque9.

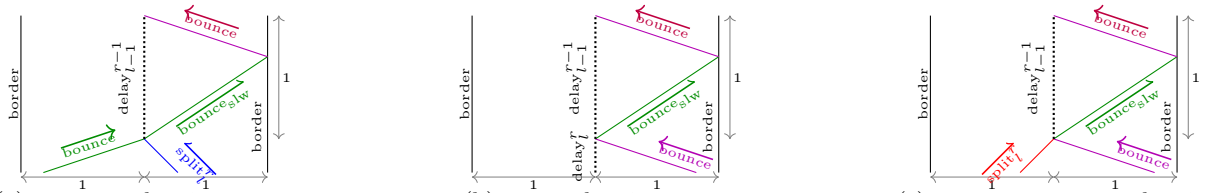
L'arbre tracé s'accumule donc bien exactement sur le segment ciblé. □

2.2 Machine à Signaux Augmentés

Les méta-signaux augmentés contenant de l'information réelle – correspondant à des macro-signaux – ont leur nom de méta-signal écrit en italique. Les autres signaux augmentés, tels que border – correspondant simplement à des signaux – sont écrits en fonte normale. L'information qu'ils portent réfère aux paramètres du sommet suivant.

La configuration initiale commence avec des bordures en $+1$ et -1, comme montré précédemment en 2.1a. Chaque sous-arbre est borné par une paire de signaux de type border. Pour énumérer toutes les règles de collision, il y a, pour chaque type de sommet – `delay` et `split` – trois cas à considérer, selon que le sommet est lui-même fils unique, gauche ou droit.

La figure 2.3 montre comment un sommet ***delay*** est simulé : un signal auxiliaire (vert ou violet) marque, en rencontrant un signal principal (en pointillés noirs ou en bleu ou en rouge), l'emplacement du sommet. De cette collision résulte un signal augmenté principal, delay_{l-1}^{r-1}, ainsi qu'un signal auxiliaire vert de type $\overrightarrow{\text{bounce}_{\text{slw}}}$ (« slw » pour « slow ») qui va rebondir sur le bord – border – et revenir en tant que signal de type $\overleftarrow{\text{bounce}}$, violet, de sorte à marquer l'emplacement du sommet suivant. En effet, la quantité de temps écoulé est proportionnelle à l'écart entre les bords, qui dicte l'échelle (diamètre 2). Il est donc possible de choisir des vitesses telles que le sommet suivant se trouve à la moitié de l'écart bord à bord dans le futur. Ces vitesses sont données dans la liste des méta-signaux, figure 2.5.

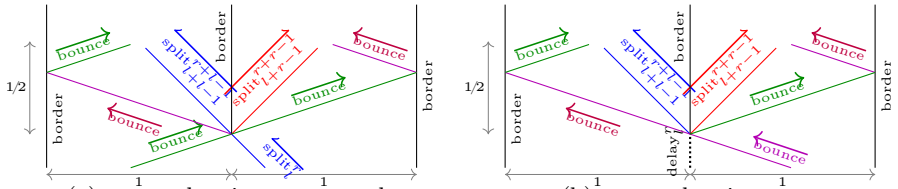


(a) ***delay*** depuis ***split*** gauche (b) ***delay*** depuis ***delay*** (c) ***delay*** depuis ***split*** droit

FIGURE 2.3 – Sommets ***delay***, lorsque $2 \leq l$ et $2 \leq r$.

Les rebonds des signaux auxiliaires – tels que $\overrightarrow{\text{bounce}}$– sont obtenus à l'aide des règles de collision listées à la figure 2.6a. Les deux dernières règles de cette liste traitent le cas où de tels signaux se rencontrent en bordure en même temps.

Les trois règles de la figure 2.6b correspondent à la collision centrale de chacun des trois cas de la figure 2.3 La partie gauche de chacune de ces règles correspond à la collision de la branche entrante en fonction de sa provenance ; la partie droite, la même pour toutes, est nécessaire pour simuler un ***split***.



(a) ***split*** depuis ***split*** gauche (b) ***split*** depuis ***delay*** (c) ***split*** depuis ***split*** droit

FIGURE 2.4 – Sommets ***split***, lorsque $l < 2$ ou $r < 2$.

La figure 2.4 montre comment les sommets ***split*** sont simulés. Chaque signal auxiliaire permet, par son intersection avec un sommet principal, de marquer l'emplacement de la collision. De cette collision sortent, à gauche, un signal augmenté principal bleu $\overleftarrow{\text{split}}_{l+l-1}^{r+l-1}$ et le signal auxiliaire $\overleftarrow{\text{bounce}}$, qui devient $\overrightarrow{\text{bounce}}$ après rebond et rencontre le signal principal à l'emplacement du fils gauche, et, de manière similaire, à droite, un signal principal $\overrightarrow{\text{split}}_{l+r-1}^{r+r-1}$ et un signal rebond $\overrightarrow{\text{bounce}}$.

Les trois règles de la figure 2.6c correspondent à la collision centrale de chacun des trois cas de la figure 2.4

Meta-signal	**Speed**
border	0
$\overrightarrow{\text{bounce}_{\text{slw}}}$	3/2
$\overleftarrow{\text{bounce}}$	-3
$\overrightarrow{\text{bounce}}$	3

Meta-signal	**Speed**
$l, r \in [1, +\infty)$ delay_l^r	0
$l, r \in [1, +\infty)$ $\overrightarrow{\text{split}}_l^r$	1
$l, r \in [1, +\infty)$ $\overleftarrow{\text{split}}_l^r$	-1

FIGURE 2.5 – Signaux augmentés.

2.3 Machine à Signaux

Par souci de clarté, certains signaux sont omis des figures schématiques, car redondants ou sans rapport avec l'objet de la figure. Les figures sans annotation (nom de signaux) ont été générées par un simulateur de Machine à Signaux Java et sont complètes.

$$\{\overrightarrow{\text{bounce}_{\text{slw}}}, \text{border}\} \to \{\overleftarrow{\text{bounce}}, \text{border}\}$$
$$\{\overrightarrow{\text{bounce}}, \text{border}\} \to \{\overleftarrow{\text{bounce}}, \text{border}\}$$
$$\{\text{border}, \overleftarrow{\text{bounce}}\} \to \{\text{border}, \overrightarrow{\text{bounce}}\}$$
$$\{\overrightarrow{\text{bounce}_{\text{slw}}}, \text{border}, \overleftarrow{\text{bounce}}\} \to \{\overleftarrow{\text{bounce}}, \text{border}, \overrightarrow{\text{bounce}}\}$$
$$\{\overrightarrow{\text{bounce}}, \text{border}, \overleftarrow{\text{bounce}}\} \to \{\overleftarrow{\text{bounce}}, \text{border}, \overrightarrow{\text{bounce}}\}$$

(a) rebonds

$$\{\text{delay}_l^r, \overleftarrow{\text{bounce}}\} \xrightarrow{2\le l\wedge 2\le r} \{\text{delay}_{l-1}^{r-1}, \overrightarrow{\text{bounce}_{\text{slw}}}\}$$
$$\{\overrightarrow{\text{bounce}}, \overleftarrow{\text{split}_l^r}\} \xrightarrow{2\le l\wedge 2\le r} \{\text{delay}_{l-1}^{r-1}, \overrightarrow{\text{bounce}_{\text{slw}}}\}$$
$$\{\overrightarrow{\text{split}_l^r}, \overleftarrow{\text{bounce}}\} \xrightarrow{2\le l\wedge 2\le r} \{\text{delay}_{l-1}^{r-1}, \overrightarrow{\text{bounce}_{\text{slw}}}\}$$

(b) `delay`

$$\{\text{delay}_l^r, \overleftarrow{\text{bounce}}\} \xrightarrow{\neg(2\le l\wedge 2\le r)} \{\overleftarrow{\text{bounce}}, \overleftarrow{\text{split}_{l+l-1}^{r+l-1}}, \text{border}, \overrightarrow{\text{split}_{l+r-1}^{r+r-1}}, \overrightarrow{\text{bounce}}\}$$
$$\{\overrightarrow{\text{bounce}}, \overleftarrow{\text{split}_l^r}\} \xrightarrow{\neg(2\le l\wedge 2\le r)} \{\overleftarrow{\text{bounce}}, \overleftarrow{\text{split}_{l+l-1}^{r+l-1}}, \text{border}, \overrightarrow{\text{split}_{l+r-1}^{r+r-1}}, \overrightarrow{\text{bounce}}\}$$
$$\{\overrightarrow{\text{split}_l^r}, \overleftarrow{\text{bounce}}\} \xrightarrow{\neg(2\le l\wedge 2\le r)} \{\overleftarrow{\text{bounce}}, \overleftarrow{\text{split}_{l+l-1}^{r+l-1}}, \text{border}, \overrightarrow{\text{split}_{l+r-1}^{r+r-1}}, \overrightarrow{\text{bounce}}\}$$

(c) `split`

FIGURE 2.6 – Règles de collision.

2.3.1 Structure

La Figure 2.7 montre comment la machine à signaux augmentés est simulée par une machine à signaux. Les parties remplies en jaune correspondent à des macro-signaux et des macro-collisions. Les signaux noirs sont les bordures et sont définis de manière identique aux signaux augmentés correspondants, tandis que le signaux de rebonds, ici verts, ont été doublés pour tenir compte de la largeur des macro-signaux.

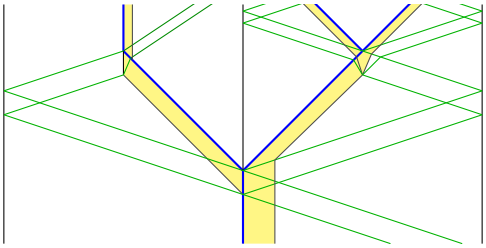


FIGURE 2.7 – Exemple schématique : après un `delay`, un `split`, puis un `delay` à gauche et un `split` à droite.

Le doublage des signaux conduit aussi à un doublage des méta-signaux pour distinguer le signal inférieur du signal supérieur : un signal augmenté de type $\overrightarrow{\text{bounce}}$ par exemple sera remplacé par une paire de signaux ($\overrightarrow{\text{bounce}^{\text{top}}}$,$\overrightarrow{\text{bounce}^{\text{bot}}}$), avec $\overrightarrow{\text{bounce}^{\text{top}}}$ au-dessus.

Les définitions de ces signaux bordures et rebonds, ainsi que les règles de collision les concernant sont présentées à la Figure 2.8.

Meta-signal	Speed
border	0
$\overrightarrow{\text{bounce}^{\text{bot}}_{\text{slw}}}$, $\overrightarrow{\text{bounce}^{\text{top}}_{\text{slw}}}$	3/2
$\overrightarrow{\text{bounce}^{\text{bot}}}$, $\overrightarrow{\text{bounce}^{\text{top}}}$	3
$\overleftarrow{\text{bounce}^{\text{bot}}}$, $\overleftarrow{\text{bounce}^{\text{top}}}$	−3

$$\forall i \in \{\text{bot}, \text{top}\}\ \{\overrightarrow{\text{bounce}^i_{\text{slw}}}, \text{border}\} \to \{\overleftarrow{\text{bounce}^i}, \text{border}\}$$
$$\forall i \in \{\text{bot}, \text{top}\}\ \{\overrightarrow{\text{bounce}^i}, \text{border}\} \to \{\overleftarrow{\text{bounce}^i}, \text{border}\}$$
$$\forall i \in \{\text{bot}, \text{top}\}\ \{\overleftarrow{\text{bounce}^i}, \text{border}\} \to \{\overrightarrow{\text{bounce}^i}, \text{border}\}$$

$$\forall i, j \in \{\text{bot}, \text{top}\}\ \{\overrightarrow{\text{bounce}^i_{\text{slw}}}, \text{border}, \overleftarrow{\text{bounce}^j}\} \to \{\overleftarrow{\text{bounce}^i}, \text{border}, \overrightarrow{\text{bounce}^j}\}$$
$$\forall i, j \in \{\text{bot}, \text{top}\}\ \{\overrightarrow{\text{bounce}^i}, \text{border}, \overleftarrow{\text{bounce}^j}\} \to \{\overleftarrow{\text{bounce}^i}, \text{border}, \overrightarrow{\text{bounce}^j}\}$$

FIGURE 2.8 – Définitions pour les signaux rebonds.

La hauteur entre deux signaux rebonds, h_b, et la largeur d'un macro-signal de l'arbre vérifient les deux conditions suivantes : $h_b = w_t$ aux deux sorties d'un `split`, et $h_b = 4/3w_t$ après un `delay`. La redirection des macro-signaux lors d'un sommet induit les mises à jour suivantes :

- lors d'un `split` depuis un `split`, $h_{b'} = 1/2h_b$ et $w_{t'} = 1/2w_t$;
- lors d'un `split` depuis un `delay`, $h_{b'} = 1/2h_b$ et $w_{t'} = 3/8w_t$;
- lors d'un `delay` depuis un `split`, $h_b = 4/3w_t$;
- lors d'un `delay` depuis un `delay`, inchangés.

Ces changements maintiennent la relation entre h_b et w_t. Nous ne prouvons pas ces mises à jour, mais elles peuvent être déduites des vitesses des méta-signaux donnés aux figures 2.8, 2.16, 2.18, 2.20.

2.3.2 Encodage des Macro-Signaux

Les signaux augmentés sont simulés par des macro-signaux, comme illustré par la Figure 2.9. Les signaux de type tree (ou variante) sont à l'emplacement exact des signaux augmentés que le macro-signal simule. Ensuite viennent quatre signaux parallèles, encodant chacun une valeur par son écart au signal de type tree : un signal de type one – en marron foncé – et un de type two – en orange foncé –

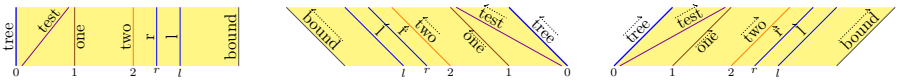
qui encodent respectivement les constantes 1 – nécessaire pour donner l'échelle – et 2 – pratique pour notre cas particulier ; deux signaux de types l et r – en bleu – qui encodent respectivement la valeur des paramètres l et r.

Un cinquième signal parallèle, de type bound, délimite la fin du macro-signal.

Les signaux codant l et r sont situés entre one inclus et bound exclu. Ils sont les seuls affectés par les mises à jour – les proportions de distance entre les quatre autres signaux parallèles, tree, one, two et bound, ne changent pas à l'issue d'une macro-collision. Grâce au Lem. 8, les paramètres, courants et subséquents, sont bornés par $2 + \alpha$ à partir du premier ***split***. Il est donc aisé de trouver un encodage initial tel que les signaux de type l et r demeurent entre one (Lem. 7) et bound.

Sur la Fig. 2.9, un dernier signal, de type test, démarre le test pour déterminer la nature de la prochaine étape.

Chaque macro-signal a la même vitesse que le signal augmenté qu'il encode : 0, −1 ou 1.



(a) ***delay*** (b) ***split*** gauche (c) ***split*** droit

FIGURE 2.9 – Encodage d'un signal augmenté juste après une macro-collision.

Dans le reste de cette section, les méta-signaux utilisés exclusivement à l'intérieur d'un macro-signal correspondant à delay_l^r, $\overrightarrow{\text{split}}_l^r$ et $\overleftarrow{\text{split}}_l^r$ portent respectivement aucune flèche, une flèche en pointillés vers la droite, une flèche en pointillés vers la gauche, et ce peu importe leur direction et vitesse. Par exemple, one, o͝ne͝ et ó͝ne͝ encodent chacun la position de l'unité dans différents macro-signaux. Les méta-signaux impliqués (excepté test) sont listés dans la Figure 2.10.

Meta-signal	Speed
tree, bound, o͝ne͝, two, l, r	1
tree, bound, one, two, l, r	0
tree, bound, ó͝ne͝, two, l, r	−1

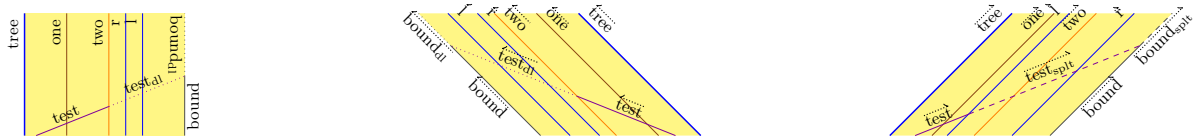
FIGURE 2.10 – Méta-signaux pour l'encodage de l'information du méta-signal augmenté.

2.3.3 Macro-Collisions

Les macro-collisions sont simulées en deux étapes, d'abord la redirection, puis la mise à jour des paramètres.

Pour savoir si le prochain sommet est un ***delay*** ou un ***split***, on teste si l et r sont tous deux supérieurs à 2. (Algo. 1). Avec notre encodage, il suffit d'envoyer un signal (test) et de voir s'il croise two avant l ou r. Comme montré à la Figure 2.11, un signal de type test (violet) est envoyé depuis tree et change de type – test_{dl} pointillés courts pour un ***delay*** et $\text{test}_{\text{splt}}$ pointillés longs pour un ***split*** – en fonction du résultat du test. Le résultat du test est ensuite reporté sur le dernier signal, de type bound, qui devient de type bound_{dl} pointillés courts ou $\text{bound}_{\text{splt}}$ pointillés longs respectivement. Tous les signaux du macro-signal sont ensuite parallèles est rien ne se passe jusqu'à la prochaine macro-collision.

Les figures 2.11a et 2.11b illustrent un test résultant en un ***delay*** tandis que Fig. 2.11c illustre un test résultant en un ***split***.



(a) depuis un ***delay*** (b) ***split*** depuis un ***split*** gauche (c) ***delay*** depuis un ***split*** droit

FIGURE 2.11 – Test.

Les méta-signaux et les règles de collision en jeu sont listées dans la Figure 2.12.

Une macro-collision est initiée par la collision d'un signal rebond ($\overleftarrow{\text{bounce}^{\text{bot}}}$ ou $\overrightarrow{\text{bounce}^{\text{bot}}}$) avec le macro-signal. Les cas ***delay*** et ***split*** sont considérés l'un après l'autre, avec chacun trois sous-cas en fonction de la provenance.

La redirection est faite à l'aide des outils présentés précédemment (s-Sect. 1.2.3). Puisque les signaux internes au macro-signal – one, two, l et r – sont tous redirigés de la même manière, seul le signal de type one est présent dans les figures schématiques. Détaillons maintenant les quatre cas de redirection à considérer.

$s_{\text{cmp}} = 2$

Meta-signal	Speed
$\overrightarrow{\text{test}}$, $\overrightarrow{\text{test}_{\text{dl}}}$, $\overrightarrow{\text{test}_{\text{splt}}}$	s_{cmp}
test, $\overrightarrow{\text{test}_{\text{dl}}}$, $\overrightarrow{\text{test}_{\text{splt}}}$	s_{cmp}
$\overleftarrow{\text{test}}$, $\overleftarrow{\text{test}_{\text{dl}}}$, $\overleftarrow{\text{test}_{\text{splt}}}$	$-s_{\text{cmp}}$

Meta-signal	Speed
$\overrightarrow{\text{bound}_{\text{splt}}}$, $\overrightarrow{\text{bound}_{\text{dl}}}$	1
$\text{bound}_{\text{splt}}$, $\overrightarrow{\text{bound}_{\text{dl}}}$	0
$\overleftarrow{\text{bound}_{\text{splt}}}$, $\overleftarrow{\text{bound}_{\text{dl}}}$	-1

$\{\text{test, two}\} \to \{\text{two, test}_{\text{dl}}\}$
$\{\text{test}_{\text{dl}}\text{, bound}\} \to \{\text{bound}_{\text{dl}}\}$

$\{\text{test, l}\} \to \{\text{two, test}_{\text{splt}}\}$
$\{\text{test, r}\} \to \{\text{two, test}_{\text{splt}}\}$
$\{\text{test}_{\text{splt}}\text{, bound}\} \to \{\text{bound}_{\text{splt}}\}$

Collision rules with arrows are the same as without arrows.

FIGURE 2.12 – Définitions relatives au test.

Un *delay* depuis un *delay*. Le macro-signal est déjà vertical et de la bonne taille, la seule chose à faire est de renvoyer les bons signaux-rebonds. La Figure 2.13 illustre ce cas : au croisement de bound_{dl}, le signal de rebond du bas $\overleftarrow{\text{bounce}^{\text{bot}}}$ prend note de la nature du sommet à venir en devenant de type $\overleftarrow{\text{bounce}_{\text{dl}}}$; ensuite, il rebondit simplement, devenant de type $\overrightarrow{\text{bounce}^{\text{bot}}_{\text{slw}}}$, tandis que le signal principal devient de type tree_{dl}. Le signal rebond du bas, croisant tree_{dl}, rebondit lui aussi ($\overrightarrow{\text{bounce}^{\text{top}}_{\text{slw}}}$) et le signal principal perd sa marque (redevient tree). Sur le chemin du départ, en croisant bound, ce signal de rebond $\overrightarrow{\text{bounce}^{\text{top}}}$ engendre un signal $\text{updt}^{*}_{\text{dl}}$ qui démarrera les calculs de la macro-collision (mis à jour des paramètres, précédant les tests) en atteignant tree. Les signaux internes au macro-signal demeurent non affectés.

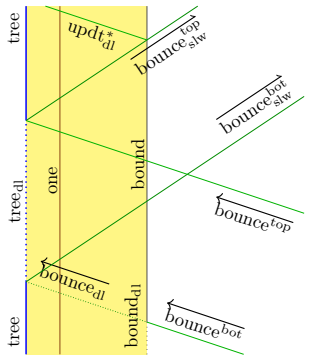


FIGURE 2.13 – *delay* après un *delay*.

Un *delay* depuis un *split* droit. Le sommet correspondant à la macro-collision est un fils droit, le macro-signal subit une simple ***réfraction*** sur wall_{dl} puis sur $\overleftarrow{\text{bounce}_{\text{dl}}}$, comme montré à la Figure 2.14 (voir aussi Fig. 2.23b). Cela assure à la fois la redirection et le rétrécissement du macro-signal : la vitesse intermédiaire du macro-signal est telle que sa largeur est réduite de moitié.

Encore une fois, l'information de collision, initialement portée par le signal de type $\overrightarrow{\text{bound}_{\text{dl}}}$, au bord du macro-signal, est propagée sur le mur, qui est un signal de type wall_{dl}. La sortie de la collision initiale (entre $\overleftarrow{\text{bounce}^{\text{bot}}}$ et $\overrightarrow{\text{bound}_{\text{dl}}}$) contient les signaux suivants : un signal wall_{dl}, qui sert à la première ***réfraction*** ; un signal $\overrightarrow{\text{bound}_{\text{sdl}}}$, qui est la ***réfraction*** du signal bord, portant aussi l'information ***delay*** pour la second ***réfraction*** ; et un signal $\overrightarrow{\text{bounce}_{\text{s}}}$, qui, en croisant $\overleftarrow{\text{bounce}^{\text{top}}}$, démarrera $\overrightarrow{\text{bounce}^{\text{bot}}_{\text{slw}}}$ à la bonne place.

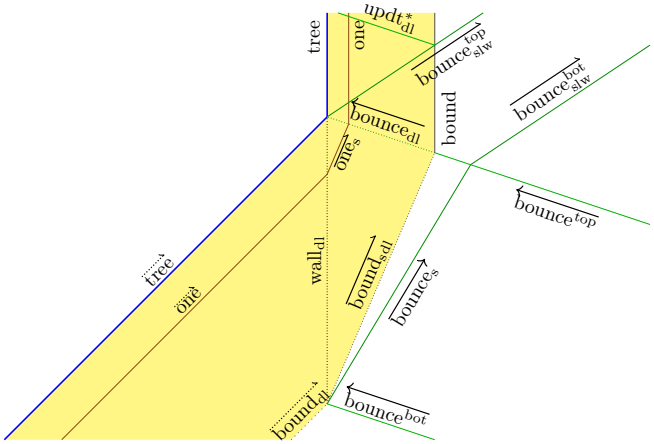


FIGURE 2.14 – *delay* après un *split* droit.

Un *delay* depuis un *split* gauche. Le sommet correspondant à la macro-collision est un fils gauche. Le macro-signal subit d'abord une ***réflexion*** sur wall_{dl}, puis une ***réfraction*** sur $\overleftarrow{\text{bounce}_{\text{dl}}}$, comme montré à la Figure 2.15 (voir aussi Fig. 2.23c). Ici encore, la largeur du signal est diminuée de moitié.

La raison pour l'emploi d'une ***réflexion*** est la suivante : l'information quant au prochain sommet est portée par le signal bordant le macro-signal (par exemple $\overleftarrow{\text{bound}_{\text{dl}}}$), et doit être sur le signal inférieur du macro-signal (i.e. $\overleftarrow{\text{tree}}$ doit être au-dessus). Les macro-signaux issus de fils droit et gauche sont donc miroirs l'un de l'autre, et il se trouve que l'on a choisi ceux issus de fils droit comme étant orientés pareillement à ceux verticaux, c'est-à-dire issus de fils unique.

Après la première ***réflexion***, tout se déroule comme après le première ***réfraction*** venant d'un fils droit. Sans signal $\overleftarrow{\text{bounce}^{\text{top}}}$ pour gérer la seconde ***réfraction***, nous utilisons un signal auxiliaire, $\overrightarrow{\text{dsep}_2}$, orange, venant d'un autre endroit. Un autre signal auxiliaire, $\overleftarrow{\text{dsep}_1}$, orange aussi, est donc lancé depuis la collision initiale pour marquer cet endroit, démarrant $\overrightarrow{\text{dsep}_2}$. Le signal $\overrightarrow{\text{dsep}_2}$ intercepte alors $\overrightarrow{\text{bounce}_{\text{s}}}$ en un lieu et une date permettant non seulement d'initier la seconde ***réfraction***, mais aussi de lancer le signal de rebond inférieur, $\overrightarrow{\text{bounce}^{\text{bot}}_{\text{slw}}}$.

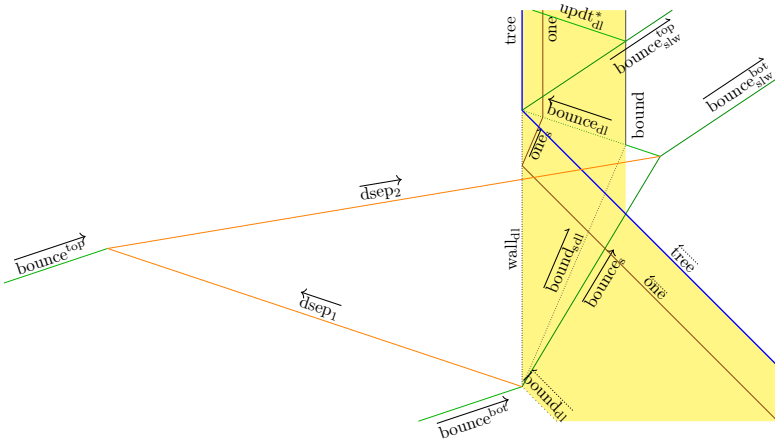


FIGURE 2.15 – ***delay*** après un ***split*** gauche.

Les nouveaux méta-signaux et collisions en jeu sont listés en Figure 2.16.

$s_{\text{shr}} = 3/7$
$s^{\text{fst}}_{\text{shr}} = 3/5$
$s_{\text{cmp}} = 2$

Meta-signal	Speed
tree_{dl}, wall_{dl}	0
$\overleftarrow{\text{dsep}_1}$	-3
$\overrightarrow{\text{dsep}_2}$	6

Meta-signal	Speed
$\overleftarrow{\text{bounce}_{\text{dl}}}$, $\text{updt}^*_{\text{dl}}$	-3
$\overrightarrow{\text{bounce}_{\text{s}}}$	$s^{\text{fst}}_{\text{shr}}$
$\overrightarrow{\text{bound}_{\text{sdl}}}$, $\overrightarrow{\text{one}_{\text{s}}}$, $\overrightarrow{\text{two}_{\text{s}}}$, $\overrightarrow{\text{r}_{\text{s}}}$, $\overrightarrow{\text{l}_{\text{s}}}$	s_{shr}

$$\{\text{bound}_{\text{dl}}, \overleftarrow{\text{bounce}^{\text{bot}}}\} \to \{\overleftarrow{\text{bounce}_{\text{dl}}}, \text{bound}\}$$
$$\{\overleftarrow{\text{bounce}_{\text{dl}}}, \text{tree}\} \to \{\text{tree}_{\text{dl}}, \overrightarrow{\text{bounce}^{\text{bot}}_{\text{slw}}}\}$$
$$\{\text{tree}_{\text{dl}}, \overleftarrow{\text{bounce}^{\text{top}}}\} \to \{\text{tree}, \overrightarrow{\text{bounce}^{\text{top}}_{\text{slw}}}\}$$
$$\{\overleftarrow{\text{bound}_{\text{dl}}}, \overleftarrow{\text{bounce}^{\text{bot}}}\} \to \{\text{wall}_{\text{dl}}, \overrightarrow{\text{bound}_{\text{sdl}}}, \overrightarrow{\text{bounce}_{\text{s}}}\}$$
$$\{\overrightarrow{\text{bounce}_{\text{s}}}, \overleftarrow{\text{bounce}^{\text{top}}}\} \to \{\overleftarrow{\text{bounce}^{\text{top}}}, \overrightarrow{\text{bounce}^{\text{bot}}_{\text{slw}}}\}$$
$$\{\overleftarrow{\text{tree}}, \text{wall}_{\text{dl}}, \overleftarrow{\text{bounce}_{\text{dl}}}\} \to \{\text{tree}, \overrightarrow{\text{bounce}^{\text{top}}_{\text{slw}}}\}$$
$$\{\overrightarrow{\text{bounce}^{\text{bot}}}, \overleftarrow{\text{bound}_{\text{dl}}}\} \to \{\overleftarrow{\text{dsep}_1}, \text{wall}_{\text{dl}}, \overrightarrow{\text{bound}_{\text{sdl}}}, \overrightarrow{\text{bounce}_{\text{s}}}\}$$
$$\{\overrightarrow{\text{bounce}^{\text{top}}}, \overleftarrow{\text{dsep}_1}\} \to \{\overrightarrow{\text{dsep}_2}\}$$
$$\{\overrightarrow{\text{bounce}_{\text{s}}}, \overrightarrow{\text{dsep}_2}\} \to \{\overleftarrow{\text{bounce}^{\text{top}}}, \overrightarrow{\text{bounce}^{\text{bot}}_{\text{slw}}}\}$$
$$\{\text{wall}_{\text{dl}}, \overleftarrow{\text{tree}}, \overleftarrow{\text{bounce}_{\text{dl}}}\} \to \{\text{tree}, \overrightarrow{\text{bounce}^{\text{top}}_{\text{slw}}}\}$$
$$\{\overrightarrow{\text{bounce}^{\text{top}}_{\text{slw}}}, \text{bound}\} \to \{\text{updt}^*_{\text{dl}}, \text{bound}, \overrightarrow{\text{bounce}^{\text{top}}_{\text{slw}}}\}$$
$$\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\overrightarrow{\mu}, \text{wall}_{\text{dl}}\} \to \{\text{wall}_{\text{dl}}, \overrightarrow{\mu_{\text{s}}}\}$$
$$\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\overleftarrow{\mu}, \text{wall}_{\text{dl}}\} \to \{\text{wall}_{\text{dl}}, \overrightarrow{\mu_{\text{s}}}\}$$
$$\{\overrightarrow{\text{bound}_{\text{sdl}}}, \overleftarrow{\text{bounce}^{\text{top}}}\} \to \{\overleftarrow{\text{bounce}_{\text{dl}}}, \text{bound}\}$$
$$\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\overrightarrow{\mu_{\text{s}}}, \overleftarrow{\text{bounce}_{\text{dl}}}\} \to \{\overleftarrow{\text{bounce}_{\text{dl}}}, \mu\}$$

FIGURE 2.16 – Définitions relative à la redirection à l'occasion d'un sommet ***delay***.

Lors d'un sommet ***split***, il est question de dupliquer le macro-signal, pour chacun des fils, ainsi que de générer une nouvelle bordure – border – les séparant.

Un *split* depuis un *split*. La duplication est effectuée de manière symétrique (voir 2.26b and 2.26c), nous présentons donc seulement le cas d'un fils droit. Comme illustré sur la Figure 2.17, la collision initiale (entre $\overleftarrow{\text{bounce}^{\text{bot}}}$ et $\overleftarrow{\text{bound}_{\text{splt}}}$) engendre, entre autres, un signal vertical $\text{wall}_{\text{splt}}$, qui va à la fois réfléchir et diffracter le macro-signal, le dupliquant. Cette duplication, faite avec les bonnes vitesses et combinée

à une ***réfraction*** subséquente sur chacune des branches, permet de réduire de moité, comme il se doit, la taille de des macro-signaux.

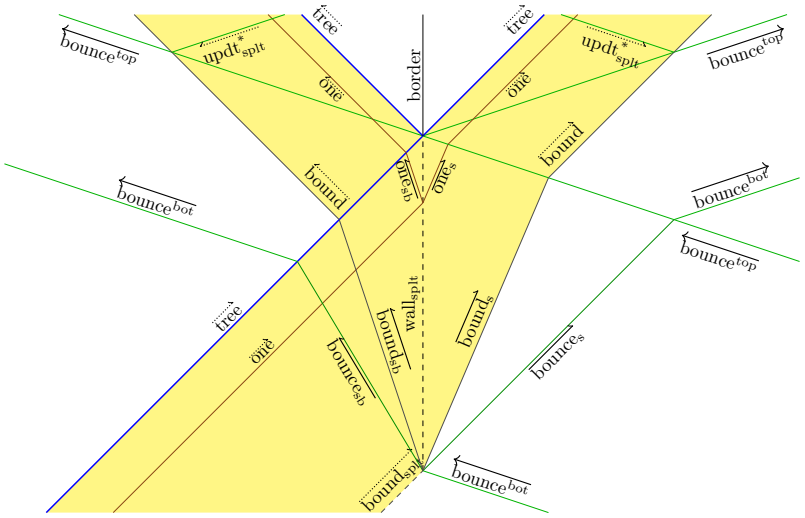


FIGURE 2.17 – ***split*** après un ***split*** droit (reroutage).

Les nouveaux méta-signaux et collisions en jeu sont listés en Fig. 2.18.

$s_{\text{shr}} = 3/7$
$s^{\text{back}}_{\text{shrink}} = 1/3$
$s^{\text{bounce}}_{\text{shrink}} = 1$
$s^{\text{bounceBack}}_{\text{shrink}} = 3/5$

Meta-signal	Speed
$\text{wall}_{\text{splt}}$	0
$\overrightarrow{\text{bounce}_{\text{s}}}$	$s^{\text{bounce}}_{\text{shrink}}$
$\overleftarrow{\text{bounce}_{\text{s}}}$	$-s^{\text{bounce}}_{\text{shrink}}$
$\overrightarrow{\text{bounce}_{\text{sb}}}$	$s^{\text{bounceBack}}_{\text{shrink}}$
$\overleftarrow{\text{bounce}_{\text{sb}}}$	$-s^{\text{bounceBack}}_{\text{shrink}}$

Meta-signal	Speed
$\overrightarrow{\text{bound}_{\text{s}}}$	s_{shr}
$\overleftarrow{\text{one}_{\text{s}}}$, $\overleftarrow{\text{two}_{\text{s}}}$, $\overleftarrow{\text{r}_{\text{s}}}$, $\overleftarrow{\text{l}_{\text{s}}}$, $\overleftarrow{\text{bound}_{\text{s}}}$	$-s_{\text{shr}}$
$\overrightarrow{\text{one}_{\text{sb}}}$, $\overrightarrow{\text{two}_{\text{sb}}}$, $\overrightarrow{\text{r}_{\text{sb}}}$, $\overrightarrow{\text{l}_{\text{sb}}}$, $\overrightarrow{\text{bound}_{\text{sb}}}$	$s^{\text{back}}_{\text{shrink}}$
$\overleftarrow{\text{one}_{\text{sb}}}$, $\overleftarrow{\text{two}_{\text{sb}}}$, $\overleftarrow{\text{r}_{\text{sb}}}$, $\overleftarrow{\text{l}_{\text{sb}}}$, $\overleftarrow{\text{bound}_{\text{sb}}}$	$-s^{\text{back}}_{\text{shrink}}$
$\text{updt}^{*}_{\text{splt}}$	$-s_{\text{cmp}}$
$\text{updt}^{*}_{\text{splt}}$	s_{cmp}

$$\{\overleftarrow{\text{bounce}^{\text{bot}}}, \overrightarrow{\text{bound}_{\text{splt}}}\} \to \{\overleftarrow{\text{bounce}_{\text{sb}}}, \overleftarrow{\text{bound}_{\text{sb}}}, \text{wall}_{\text{splt}}, \overrightarrow{\text{bound}_{\text{s}}}, \overrightarrow{\text{bounce}_{\text{s}}}\}$$
$$\{\overrightarrow{\text{bounce}^{\text{bot}}}, \overrightarrow{\text{bound}_{\text{splt}}}\} \to \{\overrightarrow{\text{bounce}_{\text{sb}}}, \overrightarrow{\text{bound}_{\text{sb}}}, \text{wall}_{\text{splt}}, \overrightarrow{\text{bound}_{\text{s}}}, \overrightarrow{\text{bounce}_{\text{s}}}\}$$
$$\{\overrightarrow{\text{tree}}, \text{wall}_{\text{splt}}, \overleftarrow{\text{bounce}^{\text{top}}}\} \to \{\overleftarrow{\text{bounce}^{\text{top}}}, \overleftarrow{\text{updt}^{\text{lo}}_{\text{splt}}}, \overleftarrow{\text{updt}^{\text{hi}}_{\text{splt}}}, \overleftarrow{\text{tree}}, \text{border}, \overrightarrow{\text{tree}}, \overrightarrow{\text{bounce}^{\text{top}}}\}$$
$$\{\overrightarrow{\text{bound}_{\text{s}}}, \overleftarrow{\text{bounce}^{\text{top}}}\} \to \{\overleftarrow{\text{bounce}^{\text{top}}}, \overrightarrow{\text{bound}}\}$$
$$\{\overrightarrow{\text{tree}}, \overleftarrow{\text{bound}_{\text{sb}}}\} \to \{\overleftarrow{\text{bound}}, \overrightarrow{\text{tree}}\}$$
$$\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\text{wall}_{\text{splt}}, \overrightarrow{\mu}\} \to \{\overrightarrow{\mu_{\text{sb}}}, \text{wall}_{\text{splt}}, \overrightarrow{\mu_{\text{s}}}\}$$
$$\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\overrightarrow{\mu_{\text{s}}}, \overleftarrow{\text{bounce}^{\text{top}}}\} \to \{\overleftarrow{\text{bounce}^{\text{top}}}, \overrightarrow{\mu}\}$$
$$\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\overrightarrow{\text{tree}}, \overleftarrow{\mu_{\text{sb}}}\} \to \{\overleftarrow{\mu}, \overrightarrow{\text{tree}}\}$$
$$\{\overrightarrow{\text{tree}}, \overleftarrow{\text{bounce}_{\text{sb}}}\} \to \{\overleftarrow{\text{bounce}^{\text{bot}}}, \overrightarrow{\text{tree}}\}$$
$$\{\overrightarrow{\text{bounce}^{\text{top}}}, \overrightarrow{\text{bound}}\} \to \{\overrightarrow{\text{updt}^{*}_{\text{splt}}}, \overrightarrow{\text{bounce}^{\text{top}}}, \overrightarrow{\text{bound}}\}$$

$$\{\overrightarrow{\text{bound}_{\text{s}}}, \overrightarrow{\text{bounce}^{\text{top}}}\} \to \{\overrightarrow{\text{bounce}^{\text{top}}}, \overrightarrow{\text{bound}}\}$$
$$\{\overrightarrow{\text{tree}}, \overrightarrow{\text{bound}_{\text{sb}}}\} \to \{\overrightarrow{\text{bound}}, \overrightarrow{\text{tree}}\}$$
$$\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\text{wall}_{\text{splt}}, \overleftarrow{\mu}\} \to \{\overleftarrow{\mu_{\text{s}}}, \text{wall}_{\text{splt}}, \overrightarrow{\mu_{\text{sb}}}\}$$
$$\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\overleftarrow{\mu_{\text{s}}}, \overrightarrow{\text{bounce}^{\text{top}}}\} \to \{\overrightarrow{\text{bounce}^{\text{top}}}, \overleftarrow{\mu}\}$$
$$\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\overleftarrow{\text{tree}}, \overrightarrow{\mu_{\text{sb}}}\} \to \{\overrightarrow{\mu}, \overleftarrow{\text{tree}}\}$$
$$\{\overleftarrow{\text{tree}}, \overrightarrow{\text{bounce}_{\text{sb}}}\} \to \{\overrightarrow{\text{bounce}^{\text{bot}}}, \overleftarrow{\text{tree}}\}$$
$$\{\overleftarrow{\text{bounce}^{\text{top}}}, \overleftarrow{\text{bound}}\} \to \{\overleftarrow{\text{updt}^{*}_{\text{splt}}}, \overleftarrow{\text{bounce}^{\text{top}}}, \overleftarrow{\text{bound}}\}$$

FIGURE 2.18 – Définitions pour la redirection en un sommet ***split*** depuis un sommet ***split***.

Un *split* depuis un *delay*. Dans ce cas, la construction, présentée en Fig. 2.19, est plus exigente, du fait encore un fois de la différence d'orientation entre le macro-signal vertical et le macro-signal gauche. La partie droite de la collision est le fruit d'une simple ***réfraction*** sur le signal $\overleftarrow{\text{bounce}^{1}_{\text{splt}}}$. La partie gauche est obtenue par une ***réfraction*** sur $\overrightarrow{\text{sep}_1}$, une ***réflexion*** sur $\overleftarrow{\text{bound}_1}$ et une ***réfraction*** sur le signal auxiliaire $\overrightarrow{\text{sep}_3}$, lui-même construit à l'aide du signal intermédiaire $\overleftarrow{\text{sep}_2}$.

Les nouveaux méta-signaux et collisions en jeu sont listés en Fig. 2.20.

2.3.4 Mise à Jour des Paramètres

La mise à jour des paramètres a lieu selon l'Algorithme 1 (ou, de manière équivalente, Fig. 2.6). Elle a lieu juste après la macro-collision et dépend de sa nature.

Les techniques de calculs sont présentées en Section 1.2.2.

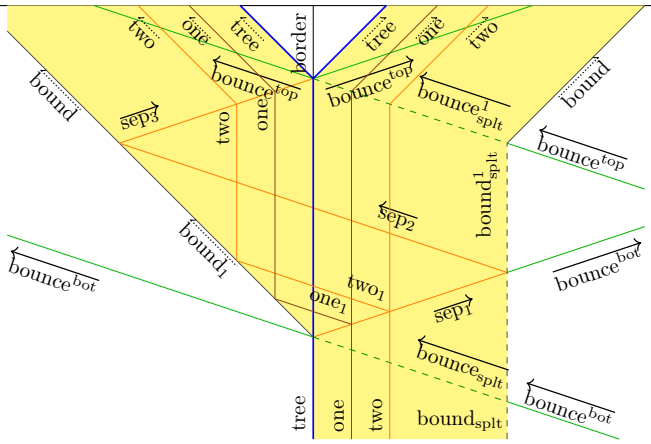


FIGURE 2.19 – Redirection depuis ***split*** après un ***delay***.

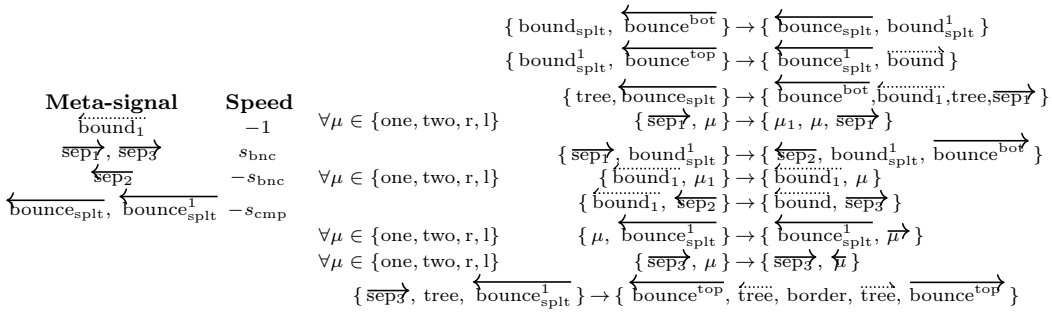


FIGURE 2.20 – Définitions pour la redirection lors d'un ***split*** après un ***delay***.

delay La mise à jour, illustrée par Fig. 2.21 consiste à retrancher 1 de chacun des deux paramètres l et r. À la fin de la mise à jour, le signal $\text{test}_{\text{start}}$ amorce le test de paramètres vu précédemment.

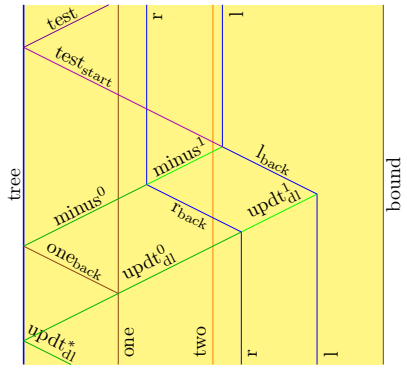


FIGURE 2.21 – Mise à jour de l et r après un ***delay***.

Les nouveaux méta-signaux et collisions utilisés pour la Fig. 2.21 sont listés en Fig. 2.22.

	Meta-signal	**Speed**
	$\text{updt}^0_{\text{dl}}, \text{updt}^1_{\text{dl}}, \text{minus}^0, \text{minus}^1, \text{updt}^0_{\text{dl}}, \text{updt}^1_{\text{dl}}, \text{minus}^0, \text{minus}^1$	s_{cmp}
$s_{\text{cmp}} = 2$	$\text{one}_{\text{back}}, \text{l}_{\text{back}}, \text{r}_{\text{back}}, \text{test}_{\text{start}}, \text{one}_{\text{back}}, \text{l}_{\text{back}}, \text{r}_{\text{back}}, \text{test}_{\text{start}}, \text{updt}^*_{\text{dl}}$	$-s_{\text{cmp}}$
	$\text{updt}^0_{\text{dl}}, \text{updt}^1_{\text{dl}}, \text{minus}^0, \text{minus}^1$	$-s_{\text{cmp}}$
	$\text{one}_{\text{back}}, \text{l}_{\text{back}}, \text{r}_{\text{back}}, \text{test}_{\text{start}}, \text{updt}^*_{\text{dl}}$	s_{cmp}

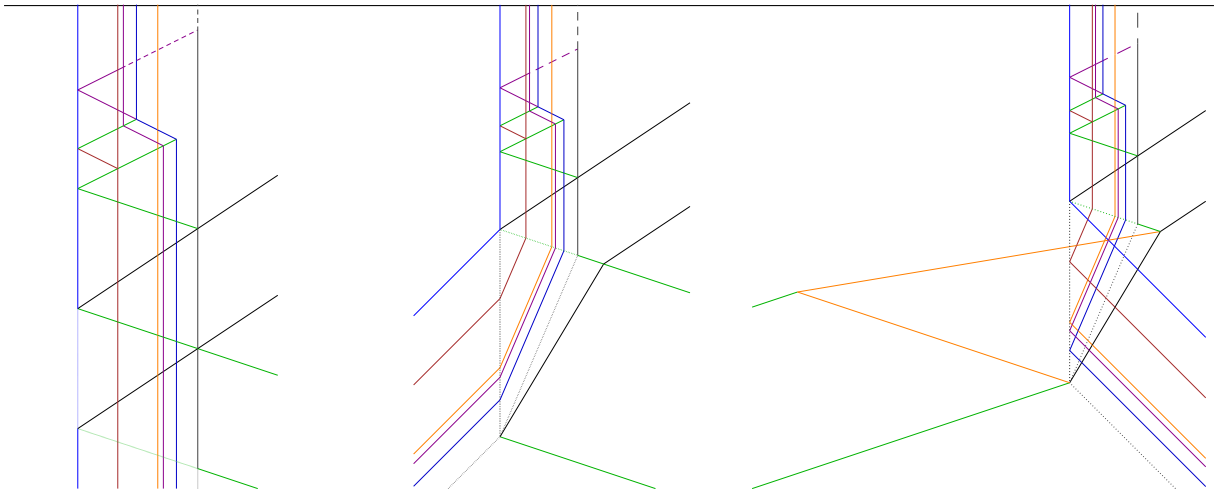
$\{ \text{tree}, \text{updt}^*_{\text{dl}} \} \to \{ \text{tree}, \text{updt}^0_{\text{dl}} \}$
$\{ \text{updt}^0_{\text{dl}}, \text{one} \} \to \{ \text{one}_{\text{back}}, \text{one}, \text{updt}^0_{\text{dl}} \}$
$\{ \text{tree}, \text{one}_{\text{back}} \} \to \{ \text{tree}, \text{minus}^0 \}$
$\{ \text{tree}, \text{test}_{\text{start}} \} \to \{ \text{tree}, \text{test} \}$

$\forall \mu \in \{\text{r}, \text{l}\}$ $\{ \text{updt}^0_{\text{dl}}, \mu \} \to \{ \mu_{\text{ back}}, \text{updt}^1_{\text{dl}} \}$
$\forall \mu \in \{\text{r}, \text{l}\}$ $\{ \text{updt}^1_{\text{dl}}, \mu \} \to \{ \mu_{\text{ back}} \}$
$\forall \mu \in \{\text{r}, \text{l}\}$ $\{ \text{minus}^0, \mu_{\text{ back}} \} \to \{ \mu, \text{minus}^1 \}$
$\forall \mu \in \{\text{r}, \text{l}\}$ $\{ \text{minus}^1, \mu_{\text{ back}} \} \to \{ \mu, \text{test}_{\text{start}} \}$

Les règles de collisions avec flèche sont identiques à celles sans.

FIGURE 2.22 – Définitions pour la mise à jour des paramètres après un ***delay***.

Les trois sous-cas possibles d'une étape ***delay*** sont montrés dans leur entièreté, redirection mise à jour et test, à la Figure 2.23.



(a) `delay` depuis `delay` (b) `delay` depuis `split` droit (c) `delay` depuis `split` gauche

FIGURE 2.23 – Implémentation de `delay`.

`split` La mise à jour dans la branche droite (resp. gauche) d'un `split` est illustrée par la Figure 2.24 et procède comme suit : r (resp. l) est ajouté à chaque paramètre, l et r ; ensuite, 1 est retranché de chaque paramètre, l et r.

Les nouveaux méta-signaux et collisions en jeu sont listés en Fig. 2.25. Le retranchement de 1 et le test des paramètres réutilisent des méta-signaux et règles déjà listés pour le `delay`.

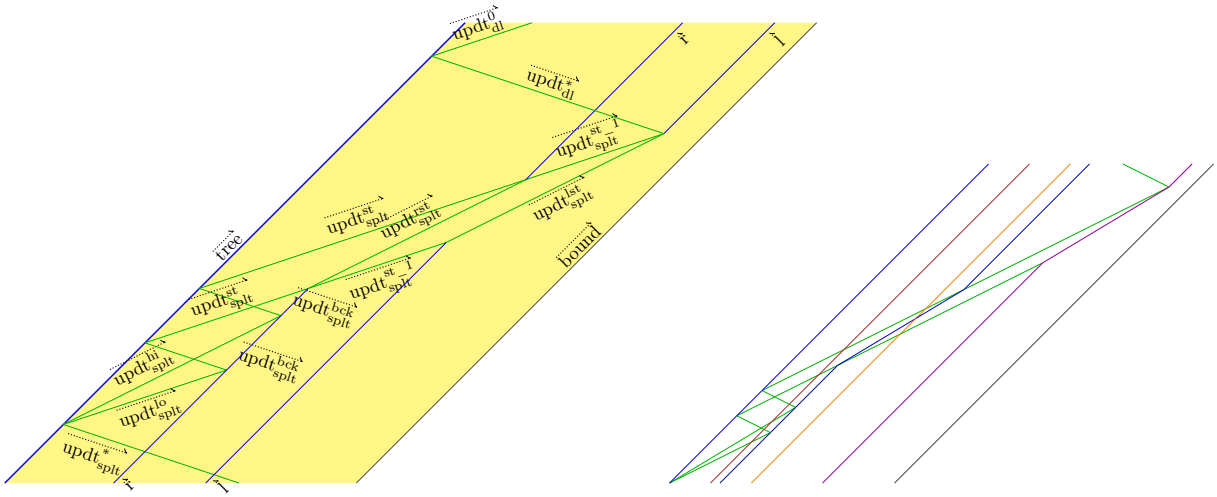


FIGURE 2.24 – Updating l and r after a `split`.

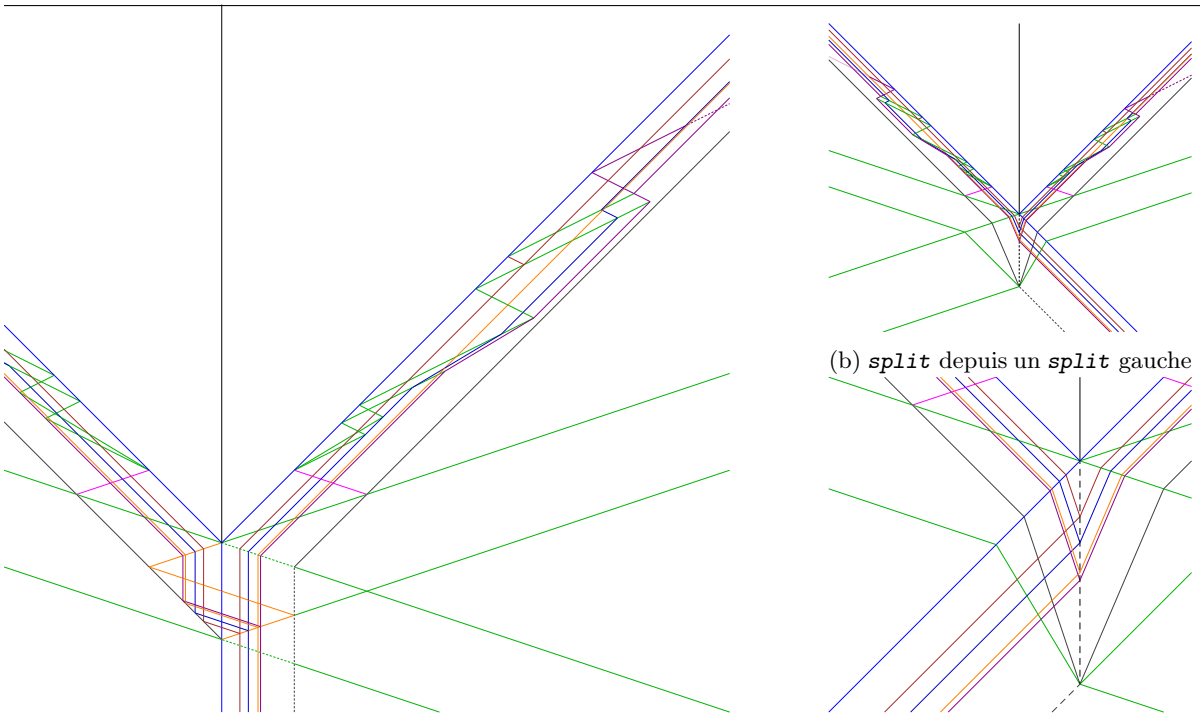
$s^{\text{up}}_{\text{splt}} = 5/3$

	Meta-signal	Speed
	$\overbrace{\text{updt}^{\text{lo}}_{\text{splt}}}, \overbrace{\text{updt}^{\text{st}}_{\text{splt}}}, \overbrace{\text{updt}^{\text{st}\_1}_{\text{splt}}}, \overbrace{\text{updt}^{\text{bck}}_{\text{splt}}}, \overbrace{\text{updt}^{*}_{\text{dl}}}$	s_{cmp}
	$\overbrace{\text{updt}^{\text{lo}}_{\text{splt}}}, \overbrace{\text{updt}^{\text{st}}_{\text{splt}}}, \overbrace{\text{updt}^{\text{st}\_1}_{\text{splt}}}, \overbrace{\text{updt}^{\text{bck}}_{\text{splt}}}, \overbrace{\text{updt}^{*}_{\text{dl}}}$	$-s_{\text{cmp}}$
	$\overbrace{\text{updt}^{\text{hi}}_{\text{splt}}}, \overbrace{\text{updt}^{\text{lst}}_{\text{splt}}}, \overbrace{\text{updt}^{\text{rst}}_{\text{splt}}}$	$-s^{\text{up}}_{\text{splt}}$
	$\overbrace{\text{updt}^{\text{hi}}_{\text{splt}}}, \overbrace{\text{updt}^{\text{lst}}_{\text{splt}}}, \overbrace{\text{updt}^{\text{rst}}_{\text{splt}}}$	$s^{\text{up}}_{\text{splt}}$
†	$\{ \overbrace{\text{updt}^{*}_{\text{splt}}}, \overbrace{\text{tree}} \} \to \{ \overbrace{\text{tree}}, \overbrace{\text{updt}^{\text{hi}}_{\text{splt}}}, \overbrace{\text{updt}^{\text{lo}}_{\text{splt}}} \}$	
$\forall i \in \{\text{lo}, \text{hi}\}$	$\{ \overbrace{\text{updt}^{i}_{\text{splt}}}, \overbrace{\text{r}} \} \to \{ \overbrace{\text{updt}^{\text{bck}}_{\text{splt}}}, \overbrace{\text{r}} \}$	
$\forall i \in \{\text{lo}, \text{hi}\}$	$\{ \overbrace{\text{updt}^{i}_{\text{splt}}}, \overbrace{\text{l}} \} \to \{ \overbrace{\text{updt}^{\text{bck}}_{\text{splt}}}, \overbrace{\text{l}} \}$	
†	$\{ \overbrace{\text{tree}}, \overbrace{\text{updt}^{\text{bck}}_{\text{splt}}} \} \to \{ \overbrace{\text{tree}}, \overbrace{\text{updt}^{\text{st}}_{\text{splt}}} \}$	
† $\forall i \in \{\text{r}, \text{l}\}$	$\{ \overbrace{\text{updt}^{\text{st}}_{\text{splt}}}, \overbrace{\text{i}} \} \to \{ \overbrace{\text{updt}^{\text{st}\_1}_{\text{splt}}}, \overbrace{\text{updt}^{\text{ist}}_{\text{splt}}} \}$	
† $\forall i \in \{\text{r}, \text{l}\}$	$\{ \overbrace{\text{updt}^{\text{st}\_1}_{\text{splt}}}, \overbrace{\text{i}} \} \to \{ \overbrace{\text{updt}^{\text{ist}}_{\text{splt}}} \}$	
† $\forall i \in \{\text{r}, \text{l}\}$	$\{ \overbrace{\text{updt}^{\text{st}\_1}_{\text{splt}}}, \overbrace{\text{updt}^{\text{ist}}_{\text{splt}}} \} \to \{ \overbrace{\text{i}}, \overbrace{\text{updt}^{*}_{\text{dl}}} \}$	

† Chaque ligne commençant par ce symbole définit deux règles : une avec toutes les flèches vers la droite, une avec toutes les flèches vers la gauche.

FIGURE 2.25 – Définitions pour la mise à jour des paramètres après un `split`.

L'étape `split` est montrée dans son entièreté, redirection mise à jour et test, à la Figure 2.26, avec Fig. 2.26c centrée sur la partie redirection.



(a) `split` depuis `delay`

(b) `split` depuis un `split` gauche

(c) `split` depuis un `split` droit

FIGURE 2.26 – Implémentations du `split`.

2.3.5 Configuration Initiale

La configuration initiale consiste en deux signaux bordures border, un macro-signal allant vers la droite, ainsi que deux signaux-rebonds, séparés de 3 fois la largeur du macro-signal de sorte à générer le premier sommet au bon endroit. Le signal $\overrightarrow{\text{bounce}^{\text{top}}}$ est encore à l'intérieur du macro-signal de sorte à déclencher la mise à jour et le test. Les paramètres initiaux du macro-signal peuvent être choisis de sorte qu'ils deviennent des paramètres-cibles valides arbitraires l et r après mise à jour.

La Fig. 2.27 montre une telle configuration initiale, ainsi que la figure qu'elle engendre.

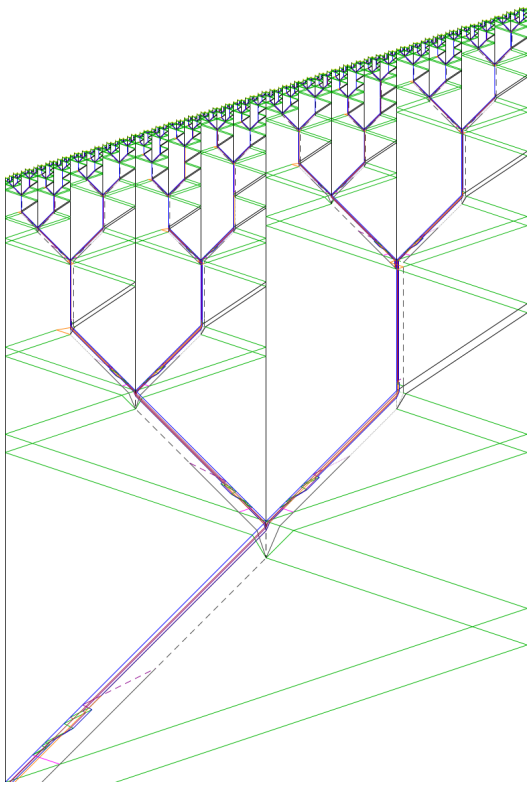


FIGURE 2.27 – Solution du problème de « saccade » d'une ligne de Fusilier pour les machines à signaux.

Chapitre 3

Blum-Shub-Smale Linéaire : Modèle et Implémentation

Dans ce chapitre, nous commençons par définir un modèle de calcul permettant de faire des opérations élémentaires sur des nombres réels – ayant la même puissance que Blum-Shub-Smale linéaire [15] – qui soit facile à implémenter en machine à signaux. Ensuite, nous définissons un langage permettant de spécifier ces machines à calculer aisément, puis nous implémentons ce modèle de calcul dans les machines à signaux. Enfin nous donnons quelques exemples.

Le langage et plus généralement sa simulation et son intégration et avec AGC est disponible à la page : `https://github.com/42xel/agc_accumulation_curves`, et dépend de python et de l'interpréteur AGC. Il est expliqué en détails en annexe.

3.1 Définitions des Modèles

3.1.1 Modèle Blum-Shub-Smale "classique"

Voici d'abord une présentation du modèle BSS "traditionnel", tel que présenté dans [15].

Définition

Une machine BSS Linéaire est la donnée des éléments suivants :

— $\mathcal{I}$: un *espace d'entrée*, typiquement $\mathbb{R}^n$;
— $\mathcal{O}$: un *espace de sortie*, typiquement $\mathbb{R}^n$;
— $\mathcal{S}$: un *espace de calcul*, $\mathcal{S} = \mathbb{R}^n$;
— un graphe fini orienté $\mathcal{G}$, qui correspond, par analogie, à l'ensemble des lignes d'un programme ou à l'ensemble des états d'une machine de Turing.

Les nœuds de $\mathcal{G}$ ou *états* sont de quatre types. Si on note η un tel nœud, alors :

— c'est un nœud initial – ou état d'entrée – unique, noté η_0, et associé à la fonction $I : \mathcal{I} \to \mathcal{S}$, qui interprète l'entrée d'un problème donné dans l'espace de calcul ; ce nœud est d'arité entrante nulle ;
— c'est un nœud de calcul, associé à une fonction linéaire $g_\eta : \mathcal{S} \to \mathcal{S}$;
— c'est un nœud de branchement, associé à une fonction linéaire non nulle $h_\eta : \mathcal{S} \mapsto \mathcal{R}$. il est d'arité sortante 2, ses successeurs sont notés γ_η^+ et γ_η^- ;
— c'est un nœud terminal associé à la fonction $O_\eta : \mathcal{S} \to \mathcal{O}$ qui interprète l'état final de l'espace de calcul en une solution du problème donné ; il est d'arité sortante nulle.

Sauf nœud de branchement et terminal, les nœuds sont d'arité sortante 1, et le successeur d'un nœud η est noté γ_η.

Le calcul se déroule en effectuant une marche sur le graphe, tout en suivant les instructions que l'état courant nous donne :

— on commence sur le nœud initial, et transforme l'entrée $i \in \mathcal{I}$ en une valeur de l'espace de calcul $\mathbf{x} \in \mathcal{S}$, puis on avance sur le nœud suivant ;
— lors d'un nœud de calcul, on effectue le calcul via la mise à jour suivante : $\mathbf{x} := g_\eta(\mathbf{x})$, et on avance au nœud suivant ;
— lors d'un nœud de branchement, on va au nœud γ_η^- si $h_\eta(\mathbf{x}) < 0$ et vers γ_η^+ sinon ;
— enfin, un nœud terminal nous somme d'arrêter ; on applique alors sa fonction O_η à la valeur finale de $\mathbf{x}$ pour interpréter celle-ci dans l'espace de sortie $\mathcal{O}$.

Remarque 10. Dans le modèle présenté dans [15], les fonctions g_η sont polynomiales plutôt que linéaires, le cas de fonctions linéaires étant présenté comme une variante. C'est cette variante, BSS Linéaire, qui nous intéresse ici.

Variante : Dimension Infinie

Les espaces d'entrée, de calcul et de sortie peuvent être de dimension infinie (dénombrable), par exemple $\mathcal{S} = \mathbb{R}^{\mathbb{Z}}$. Dans le cas d'un espace de calcul infini, on impose alors que les fonctions de calcul et branchage n'agissent que sur un ensemble fini de dimensions. Un cinquième type de nœud est introduit qui permet de copier la i-ème composante de $\mathbf{x}$ dans la j-ième, où i et j sont stockés à deux endroits de l'espace de calcul, modifiables donc par nœud de calcul, mais sous condition de rester entiers.

Équivalence Machine à Signaux

L'équivalence entre machines à signaux et modèle de calcul Blum-Shub-Smale linéaire est établie dans [16] : une machine à calcul Blum-Shub-Smale linéaire peut encoder une configuration d'une machine à signaux en stockant, pour chaque signal, son type, sa position et sa vitesse, puis peut simuler la dynamique constructive telle que vue en section 4. Inversement, on peut simuler, à l'aide d'un diagramme constructif de machine à signaux, une machine à registres illimités réelle et linéaire (linear $\mathbb{R}$-URM), un modèle de calcul équivalent à Blum-Shub-Smale linéaire qui est plus facile à simuler.

3.1.2 Machines de Turing Réelles Linéaires

De manière informelle, nous avons besoin d'une machine de Turing dont l'alphabet est $\mathbb{R}$, avec des restrictions quant à l'écriture et à la spécification des transitions.

Formellement, nous définissons *une Machine de Turing Réelle Linéaire (MTRL)* comme la donnée des éléments suivants :

— $\mathcal{I}$: un *espace d'entrée*, typiquement $\mathbb{R}^n$;

— $\mathcal{O}$: un *espace de sortie*, typiquement $\mathbb{R}^n$;

— *un ruban* bi-infini de *cellules* noté $\mathbf{x}$, chaque cellule étant capable de contenir un nombre réel : $\mathbf{x} = (x_i)_{i\in\mathbb{Z}} \in \mathbb{R}^{\mathbb{Z}}$;

— *une tête de lecture-calcul-écriture*, aussi appelée simplement *tête*, pointant à un endroit sur le ruban, noté p ; on note aussi h la valeur de la cellule sur laquelle la tête pointe : $h = x_p$.

— *un accumulateur*, utilisé pour les calculs intermédiaires ; la valeur qu'il contient est un nombre réel noté a ;

— un graphe fini orienté $\mathcal{G}$ appelé *graphe d'exécution*, dont les *nœuds* sont encore appelés *nœuds d'exécution*.

On note $\mathcal{S}$ *l'espace de calcul* $\mathbb{Z} \times \mathbb{R} \times \mathbb{R}^{\mathbb{Z}}$ dont les éléments sont des triplets : position de tête, valeur de l'accumulateur, ruban.

Le graphe d'exécution correspond vaguement à l'ensemble des états d'un automate ou d'une machine de Turing avec, comme on va le voir, moins de liberté sur les transitions.

Un nœud η de $\mathcal{G}$ est d'un type parmi les cinq types de nœuds suivants :

— un nœud initial, unique, noté η_0, associé à la fonction $I : \mathcal{I} \to \mathbb{R}^{\mathbb{Z}}$, qui interprète l'entrée d'un problème donné dans l'espace de calcul et qui est d'arité entrante nulle ;

— un nœud de calcul auquel est associé un *calcul élémentaire* g_η ;

— un nœud de mouvement avec la direction associée *gauche* ou *droite* ;

— un nœud de branchement d'arité sortante 3, de successeurs γ_η^+, $\gamma_\eta^=$ et γ_η^- ;

— un nœud terminal, d'arité sortante nulle, qui marque l'arrêt de l'exécution, et qui est associé à la fonction $O_\eta : \mathcal{S} \to \mathcal{O}$, qui permet d'interpréter l'état final de l'espace de calcul dans l'espace de sortie.

Sauf nœuds de branchement et nœuds terminaux, les nœuds sont d'arité sortante 1, et l'unique *successeur* d'un nœud η est noté γ_η.

Un calcul élémentaire est composé des ingrédients suivants :

— un récipient : l'accumulateur ou la cellule courante ;

— une opération : remise à zéro, ajouter à ou retrancher ; dans le cas d'une remise à zéro, rien d'autre n'est nécessaire ;

— une constante multiplicative, strictement positive ;

— une opérande : l'accumulateur, la cellule courante, ou bien l'unité.

L'*exécution* se déroule comme suit : depuis le nœud initial, on effectue une marche sur le graphe d'exécution tout en suivant les instructions données :

— on commence sur le nœud initial, et on encode l'entrée $i \in \mathcal{I}$ dans le ruban $\mathbf{x}$;

— lors d'un nœud de calcul, on effectue le calcul élémentaire suivant : on change la valeur du récipient de la manière prescrite ;

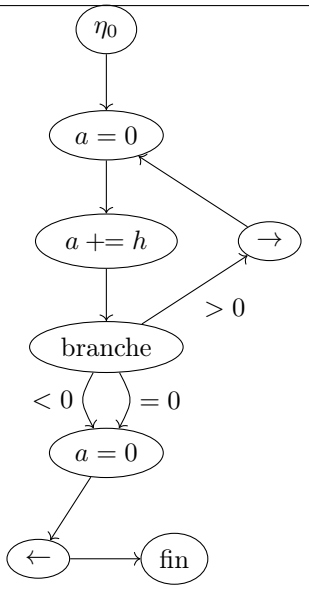


FIGURE 3.1 – Exemple de graphe d'exécution.

— lors d'un nœud de mouvement, la tête se déplace (on incrémente sa position p lors d'un nœud droit, on la décrémente lors d'un nœud gauche) ;

— lors d'un nœud de branchement, on regarde la valeur de l'accumulateur pour décider du nœud suivant : γ_η^- si $a < 0$, $\gamma_\eta^=$ si $a = 0$ et γ_η^+ si γ_η^+ si $a > 0$;

— enfin, un nœud terminal indique la fin du calcul et fournit l'interprétation de l'espace de calcul final dans l'espace de sortie.

Une *Machine de Turing Rationnelle Linéaire*, est une machine de Turing Réelle Linéaire dont tous les nombres en jeu sont rationnels : espace d'entrée et de sortie, ruban, accumulateur, et constantes des fonctions de calcul et de test.

Remarque 11. Le modèle de calcul des Machines de Turing Rationnelle Linéaire est Turing équivalent.

La figure 3.1 montre un exemple de graphe d'exécution d'une MTRL. Depuis sa position initiale, la tête est déplacé jusqu'à la dernière cellule parmi la suite de cellules consécutives contenant des valeurs strictement positives. Le nœud d'étiquette "η_0" est le nœud initial. Son successeur, le premier nœud d'étiquette "$a = 0$", est un nœud de calcul, de récipient l'accumulateur, noté a, est d'opération remise à zéro : $= 0$. Le nœud suivant, d'étiquette "$a \mathrel{+}= h$", est un nœud de calcul de récipient l'accumulateur, d'opération ajout "+=", d'opérande la cellule courante, notée "h" pour "head", et de constante multiplicative 1. Combinés, les deux nœuds dernièrement décrits remplacent la valeur contenue dans l'accumulateur par la valeur de la cellule courante. Le nœud suivant est un nœud de branchement. Son successeur associé à une valeur positive de l'accumulateur est le nœud de mouvement vers la droite, étiqueté par une flèche vers la droite "$\rightarrow$", tandis qu'un nœud de remise à zéro lui sert de successeur pour une valeur nulle et pour une valeur négative. Enfin, le nœud d'étiquette "fin" est le nœud terminal.

Équivalence MTRL BSSL

Les machines MTRL sont équivalentes aux machines Blum-Shub-Smale linéaires de dimension infinie. Les espaces d'entrée et de sortie sont les mêmes.

Pour simuler une machine MTRL à l'aide d'une machine Blum-Shub-Smale linéaire d'espace de calcul $\mathcal{S} = \mathbb{R}^{\mathbb{Z}}$, il suffit par exemple d'encoder le ruban sur les dimensions paires via l'injection $n \mapsto 2n$, d'utiliser les dimensions 1 et 3 pour la copie en dimension arbitraire, 1 pour la source et 3 pour la destination, la dimension 5 pour stocker la position de la tête de lecture-écriture, la dimension 7 pour stocker la valeur de l'accumulateur, ainsi qu'un nombre fini de dimensions d'indice impair pour des calculs intermédiaires.

Ensuite les nœuds MTRL sont encodés de la manière suivante :

— les nœuds initiaux et finaux sont des nœuds initiaux et finaux du modèle BSS, avec les mêmes fonctions d'interprétation ;

— un nœud de mouvement vers la gauche ou la droite correspond à un nœud de calcul qui incrémente ou décrémente la valeur contenue en dimension 5 ;

— un nœud de branchement est traduit par deux nœuds de branchement : un premier nœud η dont la forme linéaire h_η est la valeur à la dimension 7, dont le successeur γ_η^- correspond au successeur négatif du nœud MTRL à traduire, et dont le successeur γ_η^+ est un nœud de branchement de forme moins la valeur à la dimension 7, permettant de traiter le cas strictement positif et celui nul ;

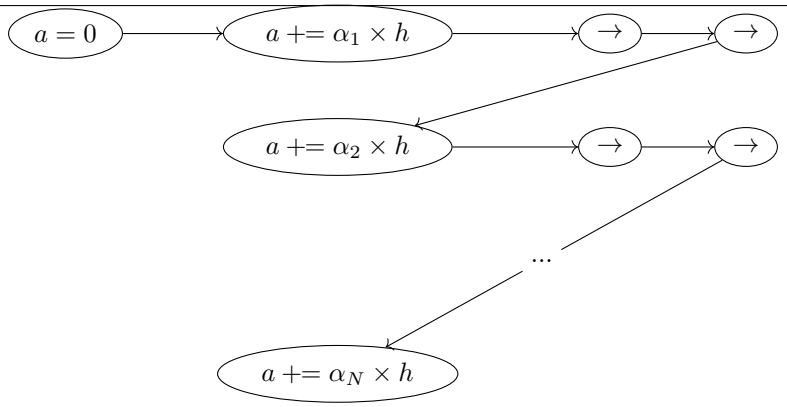


FIGURE 3.2 – Sous-graphe de d'exécution d'un calcul de forme linéaire.

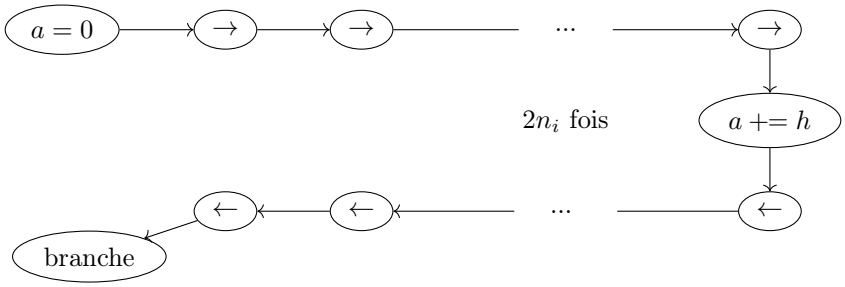


FIGURE 3.3 – Sous-graphe de d'exécution de récupération d'une valeur à une position constante.

— un nœud de calcul est traduit comme suit : un premier nœud de calcul met la valeur 9 à la dimension 3, et le double de la valeur contenue à la dimension 5 dans la dimension 1 ; après un nœud de copie, la valeur de la cellule courante du ruban simulé se retrouve donc à la dimension 9 ; un simple nœud de calcul impliquant seulement les dimensions 7 et 9 permet alors de calculer la nouvelle valeur que doit prendre le récipient, et la mettre directement à la dimension 7 si ce récipient est l'accumulateur, à la dimension 9 sinon ; si le récipient est la cellule courante, on utilise un nœud de calcul pour mettre 9 à la dimension 1 et le double de la valeur de la dimension 5 à la dimension 3, de sorte qu'après un nœud de copie, la valeur calculée se retrouve à sa place dans notre simulation de ruban.

Pour simuler une machine Blum-Shub-Smale finie d'espace de calcul $\mathcal{S} = \mathbb{R}^{[\![0,N]\!]}$ à l'aide d'une machine $MTRL$, on commence par encoder son espace de calcul sur les cellules d'indice pair, de sorte que la dimension n soit stockée dans la cellule $2n$.

La figure 3.2 montre comment calculer une forme linéaire de type $\mathbf{x} \mapsto \sum_{i=0}^{i=N} \alpha_i \times x_i$, en partant de la cellule 0, correspondant à la dimension 0, et en finissant avec la tête en position $2N$, correspondant à la dimension N. Pour simuler un branchement, il suffit donc de calculer de la sorte la forme linéaire associée h_η, de remettre la tête de lecture à zéro à l'aide de $2N$ nœuds de mouvement vers la gauche, et de faire un branchement.

Pour un nœud de calcul, on opère composante par composante, en stockant le résultat de la dimension n dans la cellule $2n + 1$. Une fois toutes les composantes calculées et stockées dans les cellules impaires, on les recopie dans les cellules paires.

Pour simuler une machine BSS linéaire de dimension infinie – $\mathcal{S} = \mathbb{R}^{\mathbb{Z}}$ – on considère, sans perte de généralité, $[\![0, N]\!]$ l'ensemble fini de dimensions sur lesquelles peuvent se faire les calcul et les conditions de branchement. On procède ensuite de la même manière que dans le cas fini, à ceci près que la cellule -1 contient la valeur 1, servant de repère, et toutes les autres cellules impaires la valeur 0. Ce repère n'impacte pas la simulation des nœuds de mouvement ou de branchement telle que déjà prescrite, puisqu'elle ignore les cellules impaires. Pour le traitement des nœuds de calcul, il suffit de nettoyer le contenu des cellules impaires après ou pendant la recopie finale.

Pour la recopie de la i-ième composante dans la j-ième, on procède en deux temps : lecture dans la i-ème, puis écriture dans la j-ième. La figure 3.3 montre comment, depuis la cellule zéro, accéder à la valeur de i stockée dans la composante $n_i \leq N$ puis revenir à la cellule zéro.

La figure 3.4 montre comment, depuis la cellule zéro et avec la position i en mémoire dans l'accumulateur, se déplacer à la cellule correspondante – $2i$ – et récupérer la valeur recherchée. Si i vaut zéro, la tête est déjà à la bonne position, et l'accumulateur étant déjà à zéro, il suffit de lui rajouter la valeur de la cellule courante. Si i est différent de zéro, par exemple si i est strictement positif, on se déplace de 2 cases, on décrémente l'accumulateur et on en teste la valeur. On continue ainsi jusqu'à ce qu'elle soit nulle, lorsque l'on aura parcouru exactement $2 \times i$ cellules et atteint la cellule que l'on doit lire.

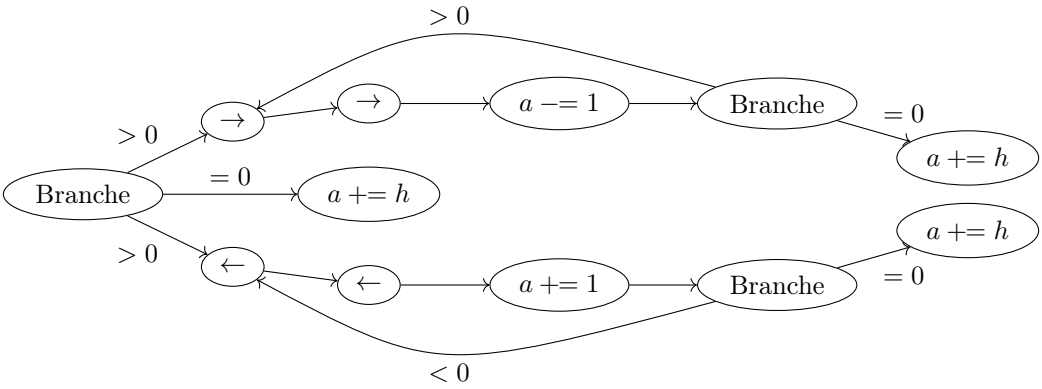


FIGURE 3.4 – Sous-graphe de d'exécution de récupération d'une valeur à une position variable stockée dans l'accumulateur.

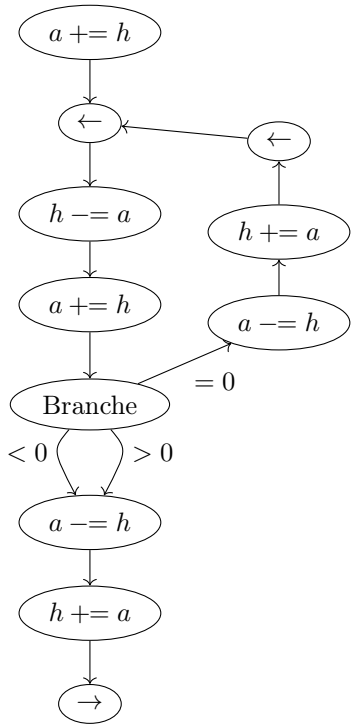


FIGURE 3.5 – Sous-graphe de d'exécution permettant de rapatrier une valeur vers un marqueur.

La figure 3.5 montre comment, depuis une cellule d'indice positif, ramener la tête de lecture à la position zéro tout en conservant la valeur de l'accumulateur. Le tout premier nœud d'étiquette $a \mathrel{+}= h$ correspond, dans la figure 3.4, au dernier nœud de la branche positive, c'est-à-dire au nœud à droite en haut. On commence donc par se déplacer d'une case vers la gauche pour se rendre sur une case impaire tout en allant vers la cellule zéro. Les deux nœuds suivants, effectuant les calculs $h \mathrel{-}= a$ et $a \mathrel{+}= h$, sont un début d'échange sur place – sans utiliser d'espace supplémentaire – de l'accumulateur et de la valeur de la cellule lue par la tête ; si l'accumulateur et la tête sont initialement à la valeur a_0 et h_0 respectivement, leur nouvelle valeur après ces deux nœuds sont $a_1 = h_0$ et $h_1 = h_0 - a_0$, si bien que l'accumulateur contient la valeur de la tête sans pour autant avoir perdu l'information sur sa valeur. On fait ensuite un branchement : si la valeur est non nulle, c'est qu'on vient de trouver la cellule -1, on inverse les deux dernier nœuds de calcul, de sorte à remettre le repère de cellule dans la cellule -1 et la valeur de l'accumulateur dans l'accumulateur, et on se déplace de une cellule vers la droite. Si la valeur est nulle, la cellule -1 est encore à notre gauche, on inverse les calculs, on se déplace de deux cellules vers la gauche, et on recommence le test. Depuis une cellule d'indice négatif, il suffit d'aller vers la droite plutôt que vers la gauche.

Écrire une valeur à la $2j$-ième cellule est un peu plus difficile que de lire à la $2i$-ième, puisque nous devons non seulement nous déplacer, mais aussi conserver la valeur de l'accumulateur. Une manière de faire, en n'utilisant que des outils déjà présentés, et d'aller à la cellule $2j - 1$ sans valeur et d'y déposer un marqueur, puis de revenir avec une valeur sans avoir besoin de connaître j.

Si la valeur à écrire est zéro, on peut directement aller l'écrire. On utilise le même procédé pour lire j que pour lire i – figure 3.3. On utilise le même procédé pour aller à la case j que décrit en figure 3.4, mais en remplaçant la lecture par une remise à zéro, comme montré en figure 3.6. Enfin, puisqu'on sait

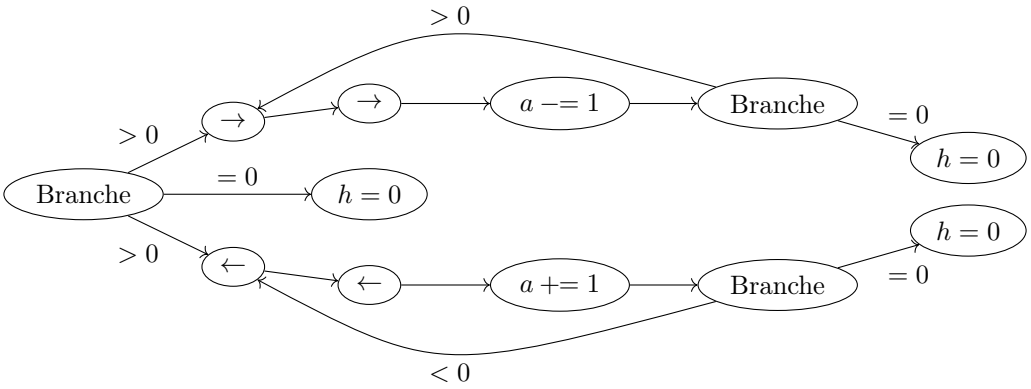


FIGURE 3.6 – Sous-graphe de d'exécution de mise à zéro d'une cellule à une position variable stockée dans l'accumulateur.

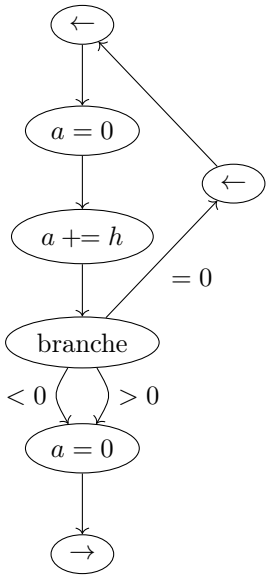


FIGURE 3.7 – Sous-graphe de d'exécution permettant de revenir à l'origine.

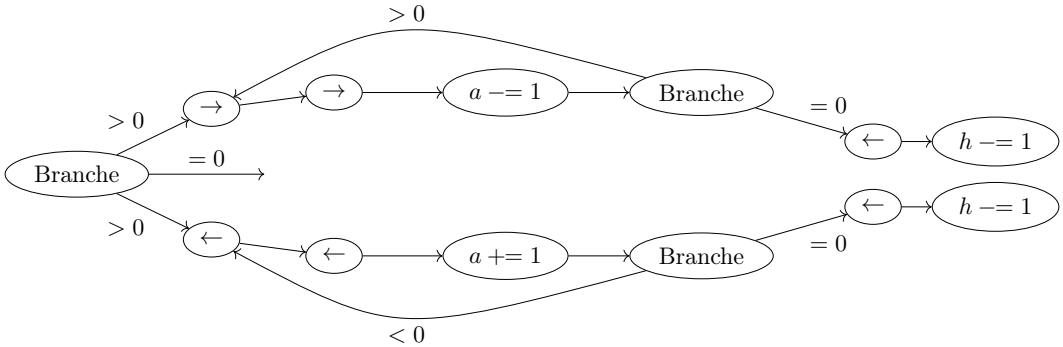


FIGURE 3.8 – Sous-graphe de d'exécution de marquage d'une cellule à une position variable stockée dans l'accumulateur.

revenir vers un marqueur avec une valeur, on sait *a fortiori* revenir vers un marqueur sans contrainte de retenir une valeur – la figure 3.7 montre comment faire.

Si la valeur à écrire est différente de zéro, on la stocke dans la cellule -1. Ensuite, on lit la valeur j – figure 3.3. Puis on procède de manière similaire à la lecture ou à la remise à zéro pour mettre un marqueur. Cela est illustré par la figure 3.8, si $j = 0$, on le traite directement, sinon on traite, dans 2 branches différentes, les cas positif et négatif. Puis on réutilise la même construction qu'à la figure 3.7 pour revenir en zéro. On lit ensuite la valeur en -1, c'est-à-dire qu'on la stocke dans l'accumulateur, on remet la cellule -1 à 1 et on se remet à la cellule zéro. Puis on utilise le même sous-graphe d'exécution qu'en figure 3.5 pour se placer à la case j tout en conservant la valeur de l'accumulateur. Puis on l'écrit, on efface le marqueur, et enfin, on revient à la cellule zéro, comme à la figure 3.7.

Cela finit de montrer comment simuler, dans une machine MTRL, un nœud copie d'une machine BSS linéaire de dimension infinie, et finit donc de montrer que le modèle MTRL est bien équivalent au modèle

BSS linéaire.

3.1.3 Langage MTRL

Un langage a été créé pour faciliter la définition des machines MTRL, ainsi que pour les interpréter et les compiler. Les buts de ce langage sont les suivants :

— définir une machine MTRL rationnelle ;

— spécifier un état initial de l'espace de calcul (ruban, position de la tête et valeur de l'accumulateur) ;

— interpréter un couple machine-état initial ;

— exporter une machine, une configuration initiale, ou un calcul au format AGC.

Le langage se veut assez familier pour pouvoir être lu sans en connaître la spécification. Cette dernière se trouve en annexe A.

La figure 3.9 montre le code correspondant au graphe d'exécution montré en figure 3.1. Le nœud initial est implicite (il précède le premier nœud).

```
loop:
>;
beginning:
a=0;
a+=h;
Branch next, next, loop;
next:
a=0;
<;
end;
```

FIGURE 3.9 – Exemple de code.

L'exemple de code commenté en figure 3.10 effectue un échange de valeur entre le cellule courante et celle à sa droite en utilisant celle à sa gauche comme espace de stockage intermédiaire.

```
a =0;                //remet l'accumulateur à zéro.
// non nécessaire, il est implicite que l'accumulateur commence à zéro
// mais une bonne habitude.
a += 1 * h;         //stocke la première valeur, de x_0, dans l'accumulateur
<;                        //va à gauche, à x_-1
h += 1 * a;         //copie de la première valeur à échanger dans x_-1
>;
>;
//deux déplacements à droite
a =0;                //mise à zéro de l'accumulateur
a += 1 * h;         //stocke la seconde valeur, de x_1, dans l'accumulateur
<;                        //on va à gauche
h =0;                //mise à zéro de la cellule courante
h += 1 * a;         //écriture de la seconde valeur dans x_0
<;                        //on va à gauche récupérer la première valeur
a =h;
//on écrit la valeur de la cellule courante
//(la première valeur à échanger) dans l'accumulateur.
//Le compilateur transforme automatiquement cette ligne en deux instructions
// élémentaires :
//remise à 0 "a =0;" suivie d'ajout "a += 1 * h;"
h =0;
//On efface x_-1,
>>;                        //un double déplacement, concis,
//de la tête de calcul de deux cellules vers la droite.
h = a;                //on écrit la première valeur dans x_1
<;
a =0;
//on revient sur la cellule x_0 et on remet l'accumulateur à zéro
end;
//Noeud Final
```

FIGURE 3.10 – Exemple de code codant l'échange de deux valeurs.

L'exemple de code en figure 3.11 définit une machine à signaux effectuant l'algorithme d'Euclide, et une configuration initiale. Cet algorithme appliqué à des nombres rationnels donne encore le PGCD

```
create_lbss_machine > machine_euclide <<FIN
//créer une machine à signaux de nom "machine_euclide"
//sa définition commence ci-après et se termine à la ligne contenant "FIN"
//cette machine calcule le PGCD de x_0 et x_1, et met le résultat dans x_0
beginning:        //déclaration d'un label
a = h;
>;
labelTest:
Branch labelSwap, labelEnd, labelNext;
//Branche : continue en labelSwap, labelEnd, labelNext,
//selon que a<0, a=0 ou a<y respectivement
labelNext:
a -= h; goto labelTest //un goto.
labelSwap:
a += h;
h -= a;
a += h;
<;
h = a;
a=0; goto beginning
labelEnd:
a += h;
h = 0;
<;
h = a;
a = 0;
end;
FIN
create_lbss_configuration > conf_euclide <<FIN
//créer une configuration
x = [33/5, 24/7]
FIN
toAGC_machine machine_euclide me.agc
//compile la machine en une machine agc
toAGC_run me.agc conf_euclide run_e.agc run_e.pdf beginning 0 -1 0 1/10
//créer un fichier agc faisant un calcul de la machine RTLM
//Les paramètres optionels à partir de "beginning" sont, dans l'ordre:
//le noeud initial
//un facteur d'échelle en temps
//une limite au nombre de pas de calculs
//un facteur d'échelle en espace
run machine_euclide configuration_euclide Euclide run -1 beginning
//créer (la trace d') un calcul d'une machine RTLM
//et le retient sous le nom Euclide
print Euclide
//affiche le run (la trace de calcul) appelé Euclide
```

FIGURE 3.11 – Exemple de code codant l'algorithme d'Euclide.

de deux nombres pour une définition généralisée, par exemple $\mathrm{PGCD}(33/5, 24/7) = 3/35$; les détails de ce que fait l'algorithme et comment importent peu, le but ici est d'illustrer le langage. Celui-ci fait davantage que coller au modèle MTRL : s'il est toujours possible de coder un nœud de calcul avec la syntaxe "récipient, opération, constante positive, fois, opérande", il est aussi possible d'écrire, directement et de manière plus naturelle, des lignes comme $a = 3$ ou $h = a$. La simplicité de conception du modèle, qui en facilite l'étude théorique telle qu'en section 3.1.2 ainsi que la compilation vers le langage AGC – section 3.2, diffère de la simplicité d'usage, à laquelle répond le langage.

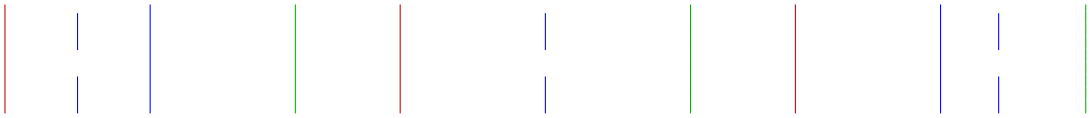
3.2 Implémentation Machine à Signaux

Dans cette section, nous montrons comment *simuler* une machine MTRL par une machine à signaux, et ce en un temps borné. Nous nous limitons aux machines d'espace d'entrée fini, $\mathbb{R}^n$. Un espace d'entrée infini

3.2.1 Représentation

Registres

Comme montré dans en Fig. 3.12, une valeur est encodée comme la distance (orientée) d'un signal valeur (en pointillés bleus dans la figure) à un signal zéro (bleu). La valeur est bornée par deux signaux appelés *marges de registre*, rouge et vert, pour détecter les dépassements. Une telle structure composée de quatre signaux (hors superposition) est appelée un *registre*.



(a) valeur négative (b) valeur nulle (c) valeur positive

FIGURE 3.12 – Des registres simples.

La Tête de Lecture-Calcul-Écriture

La Figure 3.13 montre une *tête de lecture-calcul-écriture*. Celle-ci est composée, de gauche à droite, d'un *signal programme* (en pointillés noirs sur la figure), d'un *signal horloge* (violet), puis d'une constante unité (demi-registre, c-à-d registre sans borne gauche), puis d'un registre accumulateur.

FIGURE 3.13 – Une tête de lecture-calcul-écriture.

Il y a typiquement un méta-signal par nœud de calcul de la machine MTRL pour le signal programme, cela sera détaillé à la sous-section flux 3.2.6.

Pour l'instant, il suffit de savoir que, lors de l'exécution du programme, un signal fait l'aller-retour entre les données (unité, accumulateur et cellule), et le signal du programme. Les signaux allant du programme aux données, qui servent à transmettre, une à une, les instructions élémentaires, sont appelés *signaux aller*. Les signaux allant des données au programme, qui servent à marquer la fin d'une instruction et, parfois, dans le cas d'un branchement, à transmettre de l'information, sont appelés *signaux retour*. Chaque primitive commence donc par un signal aller et se termine par un signal retour.

Le signal horloge, quant à lui, fait partie d'une horloge qui sera présentée plus tard dans cette sous-section.

Le Ruban et Accumulateur

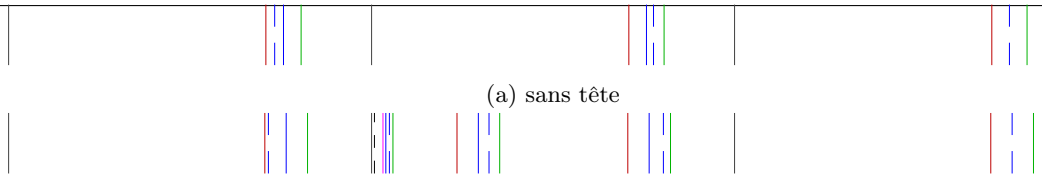
FIGURE 3.14 – Une cellule simple.

Une *cellule* est un ensemble, de gauche à droite, d'un marqueur – un signal noir, une tête ou bien la place pour la contenir, et un registre valeur de cellule. La Figure 3.14 montre une cellule simple tandis que la Figure 3.15 montre une cellule sur laquelle pointe la tête.

FIGURE 3.15 – Une cellule sur laquelle la tête pointe.

La Figure 3.16 montre trois cellules côte-à-côte. La Figure 3.16a montre ces trois cellules sans tête, tandis que la Figure 3.16b les montre avec une tête pointant sur la cellule du milieu.

Le *ruban* consiste en une juxtaposition d'un certain nombre de cellules, dites *cellules internes*, bordées par des cellules vierges (initialisées à zéro) dites *cellules de marge* qui, si elles sont entrées, déclenchent la création de nouvelles cellules, comme détaillé plus tard dans cette sous-section. La Figure 3.17 montre



(a) sans tête

(b) la tête pointe sur la cellule du milieu

FIGURE 3.16 – Trois cellules juxtaposées.

un ruban muni de telles cellules de marge. De chaque côté, le marqueur de cellule de la première cellule de marge est d'un type spécial, dénoté par des signaux en pointillés.

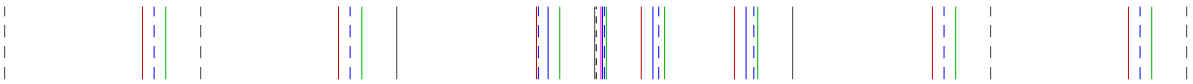


FIGURE 3.17 – Un ruban avec trois cellules internes et des marges de taille 1.

Des signaux spéciaux, situés aux endroits où auraient été le prochain marqueur de cellule, dénotent les extrémités du ruban.

3.2.2 Primitives

Déplacement de la Tête

Le déplacement de la tête est effectué par des parallélogrammes, en utilisant les marqueurs de cellule, comme montré dans la Figure 3.18.

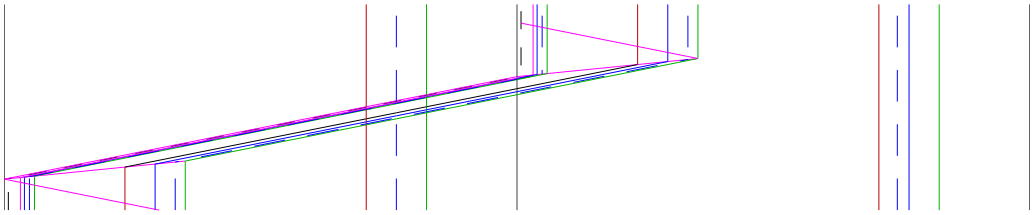


FIGURE 3.18 – Déplacement de la tête d'une cellule vers la droite.

Remise à Zéro

La remise à zéro d'un registre est effectuée simplement avec un signal aller qui interrompt le signal valeur et marque le signal zéro comme valeur, comme montré à la Figure 3.19.

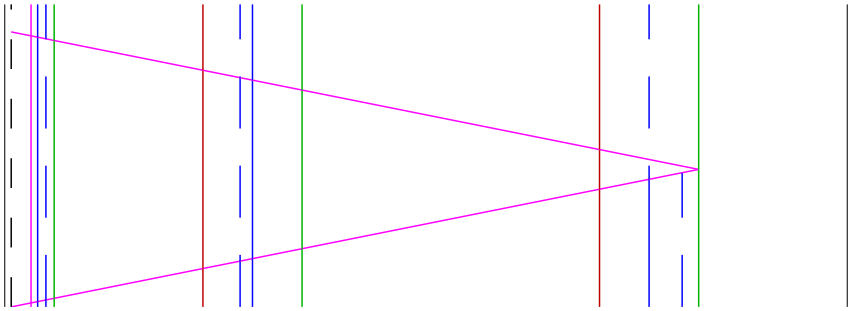


FIGURE 3.19 – Remise à zéro.

Lecture d'une Donnée

L'ajout et la soustraction à un registre sont faits en deux étapes élémentaires.

La première étape, objet de la présente sous-section, consiste à lire une valeur multipliée par une constante positive et à la stocker dans un registre dit *temporaire*. Cette première étape s'appelle *lecture*, ou *get*.

Par exemple, dans :

```
a += 1/2 * h;
```

Si on note le registre temporaire 't', la lecture correspond à :

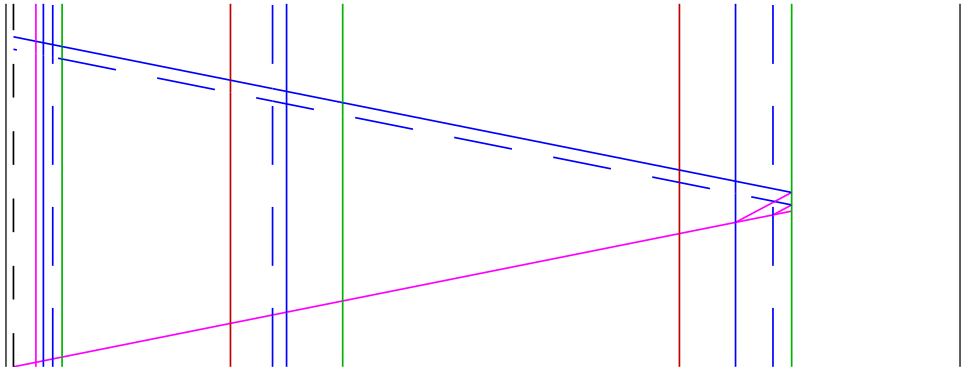
```
t <- 1/2 * h;
```

et la deuxième étape correspond à :

```
a <- a + t;
```

Ce registre temporaire consiste en deux signaux-retours parallèles, dont l'écart est proportionnel à la valeur lue. Ces deux signaux sont de types différents, l'un correspondant au zéro, l'autre à la valeur, et leur position relative permet de signer la valeur, zéro au-dessus signifiant positif.

La Figure 3.20 illustre la lecture de registre. À la Figure 3.20a, on lit une fois une valeur positive de la cellule, à la Figure 3.20b on lit une fois une valeur négative de l'accumulateur, tandis qu'à la Figure 3.20c on lit cette même valeur multipliée par 7/2.



(a) lecture de la cellule

(b) lecture de l'accumulateur

(c) lecture de la valeur de l'accumulateur multiplié par 7/2

FIGURE 3.20 – Lecture de données.

Addition et Soustraction

On suppose que le registre temporaire contient une valeur positive, encodée par l'écart entre deux signaux aller.

La Figure 3.21 montre l'addition de 45/4 (stocké dans le registre temporaire) à l'accumulateur. La construction utilise les techniques expliquées à la Section 1.2.2. Une fois l'addition terminée, un signal retour est émis.

FIGURE 3.21 – Ajout de 45/4 à l'accumulateur.

De la même manière, la Figure 3.22 montre la soustraction d'une valeur à la cellule.

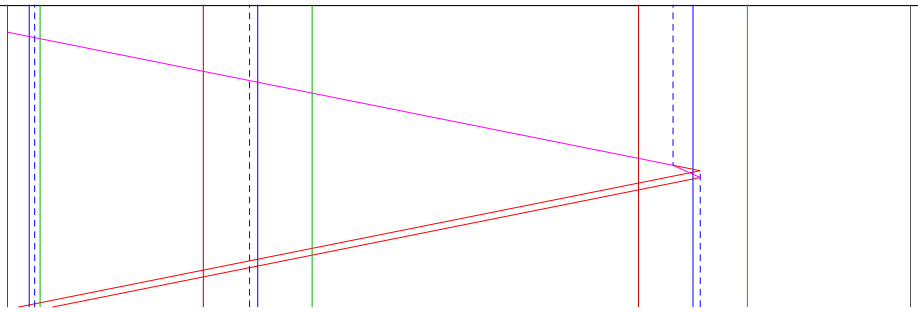


FIGURE 3.22 – Soustraction.

Test de signe d'une donnée

Le test de signe est fait comme montré par la Figure 3.23, selon la technique expliquée en Section 1.2.2 : un signal aller va vers le registre, et regarde quel signal est rencontré en premier.

FIGURE 3.23 – Test.

3.2.3 Débordement Registre

Divers événements peuvent nécessiter de mêler, entre les primitives correspondant aux nœuds de calculs, des opérations de régulation s'appliquant au ruban tout entier. Pour ce faire, la tête contient un signal dit de régulation, à droite du signal programme, dont le rôle est d'intercepter les signaux retour le temps de l'opération de régulation, avant de les libérer. Ce signal est utilisé à cette section 3.2.3 et aux deux suivantes, 3.2.4 et 3.2.5.

Il y a *débordement de registre* lorsque la valeur d'un registre se retrouve – après un nœud de calcul – en dehors des marges.

Dans ce cas, toutes les valeurs sont réduites de sorte à faire rentrer la valeur de registre débordant dans les marges, tout en conservant les proportions relatives entre les valeurs, notamment relative à l'unité. Ainsi, les valeurs encodées sont inchangées, elles sont juste encodées de manière plus compacte.

Pour ce faire, le plus grand coefficient multiplicatif, noté α_{max}, est utilisé :

— l'échelle initiale est choisie de sorte que l'unité ainsi que toute les valeurs du ruban rentrent dans les marges ;

— le ratio entre le rayon d'un registre – de zéro à la marge – et la distance entre deux registres – de marge à marge – est α_{max} ;

— le coefficient par lequel les valeurs sont réduites est $\alpha_{max} + 1$.

Cela permet d'assurer que :

— si une valeur est initialement dans les marges, elle ne déborde pas sur un autre registre après une opération ;

— si une valeur déborde des marges, elles y est ramenée en une contraction.

En effet, si on note Δx le rayon d'un registre, alors, en partant de valeurs dans les marges – leur distance au signal zéro étant strictement inférieure à Δx, la distance à zéro après une opération est strictement inférieure à $\Delta x + \alpha_{max} * \Delta x$.

La Figure 3.24 montre un débordement de registre après l'addition d'une grande valeur à l'accumulateur.

3.2.4 Débordement Mémoire

Lorsqu'une cellule de marge est entrée, il y a *débordement mémoire*. Le ruban est alors manipulé de sorte que la cellule courante redevienne une cellule interne, que les cellules qui sont internes le restent, et que le ruban reste de même longueur. Cette manipulation nécessite la création de cellules.

Lors d'un débordement à droite, l'ensemble des cellules du ruban est réduit de moitié en taille, placé à la gauche du ruban, et est juxtaposé à un ensemble de cellules vierges de taille égale. Le nombre de cellules par marge demeure constant.

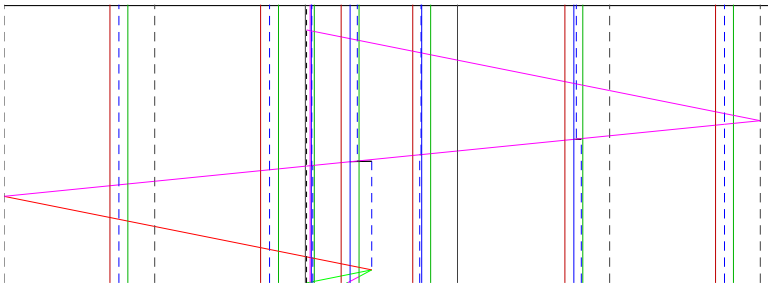


FIGURE 3.24 – Débordement de registre.

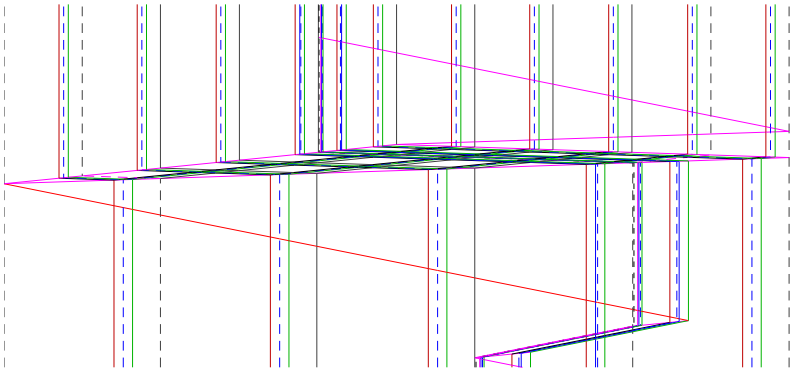


FIGURE 3.25 – Débordement mémoire à droite.

Cette manipulation est illustrée par la Figure 3.25.

Le débordement à gauche est traité de manière analogue.

Si on note m_{max} le plus grand mouvement de tête et si on le prend comme taille de marge, on a les garanties suivantes :

— un mouvement de tête depuis une cellule interne reste dans le ruban ;

— lorsque, après un mouvement, la tête se retrouve dans une cellule de marge, elle est ramenée dans une cellule interne par la manipulation ;

— les cellules internes restent internes lors de la manipulation.

En effet, lors par exemple d'un débordement à droite, la cellule interne la plus à gauche demeure à la même position relativement à la marge gauche, et la cellule courante, strictement à droite de toute cellule interne avant manipulation, se retrouve à gauche du milieu du ruban après manipulation.

3.2.5 Contraction en Temps

L'horloge sert à garantir qu'un calcul prenne un temps fini. Elle consiste en un signal horloge stationnaire et un signal pendule oscillant entre lui et le signal programme. À intervalles réguliers, le pendule marque l'horloge, qui alors intercepte le prochain signal retour marquant la fin d'une opération élémentaire, démarre une contraction du ruban, puis libère le signal retour intercepté.

Cette contraction, de ratio constant 2, est illustrée par la Figure 3.26.

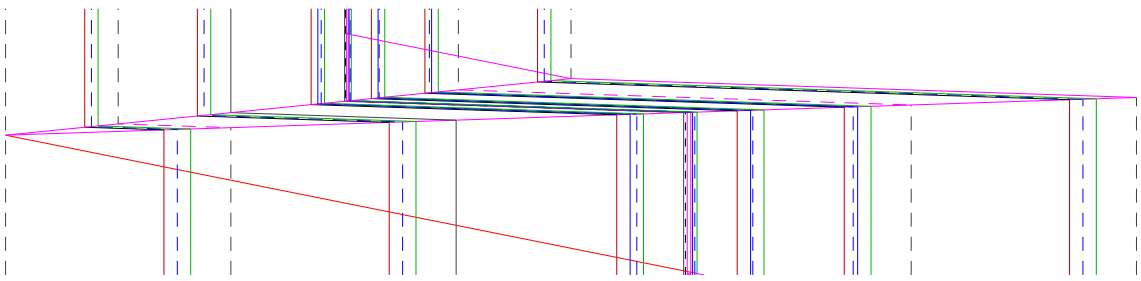


FIGURE 3.26 – Débordement en temps.

Si l'intervalle de temps entre deux contractions est borné et que la contraction est de ratio fixe, la construction est bornée dans le temps, et tout calcul fini arbitrairement long en nombre d'étapes est garanti de finir en un temps fini – temps au sens du diagramme espace-temps.

Notons Δt_h le temps d'oscillation du pendule, Δt_e le temps maximal d'exécution d'une opération élémentaire (déplacement, addition, test), Δt_o le temps de maximal de traitement d'un débordement (registre ou mémoire), et Δt_c le temps de contraction du ruban.

Le temps maximal s'écoulant entre deux contractions est $\Delta t_h + \Delta t_e + \Delta t_o + \Delta t_c$. Le temps maximal du calcul est donc borné par $2(\Delta t_h + \Delta t_e + \Delta t_o + \Delta t_c)$

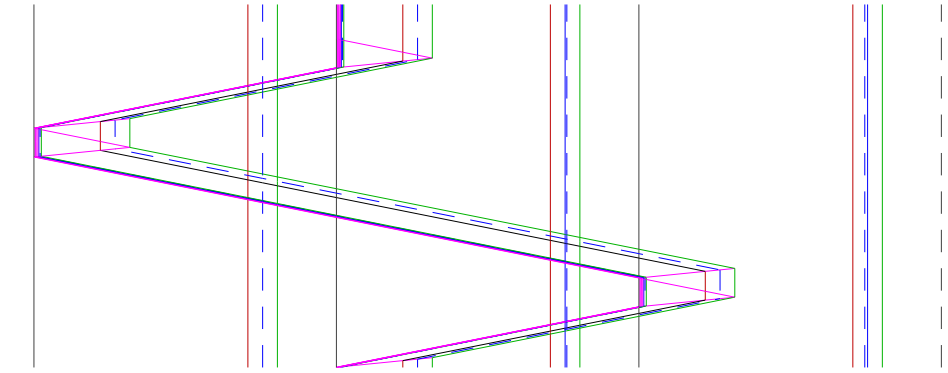
3.2.6 Flux de Contrôle

Le *flux de contrôle*, ou *flux* de la machine MTRL, c'est-à-dire la manière dont les instructions s'enchaînent, est géré par le signal programme de la tête.

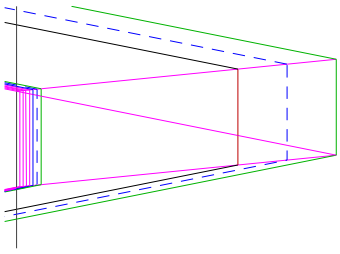
À chaque primitive correspond *un ensemble de méta-signaux initiaux*, qui démarre l'opération et correspond à un membre droit de collisions, un méta-signal programme, dénotant l'opération en cours, et un ou plusieurs *ensembles de méta-signaux finaux*, qui correspondent à des membres gauches de collisions. Un ensemble de méta-signaux initiaux est composé d'un méta-signal programme et d'un méta-signal aller. Un ensemble de méta-signaux finaux est composé d'un méta-signal programme et d'un méta-signal retour.

Les ensembles de méta-signaux initiaux et finaux sont décrits dans les sous-sections décrivant chacune des primitives. Par exemples, les primitives de type test ont trois ensembles de méta-signaux finaux, un par type de signal retour.

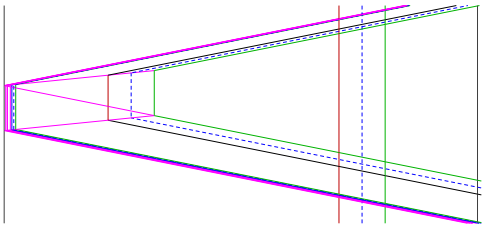
Pour chaque arête du graphe d'exécution, il y a une règle de collision dont le membre gauche est l'ensemble de méta-signaux finaux du nœud début de l'arête, et dont le membre droit est l'ensemble de méta-signaux initiaux du nœud fin de l'arête. Cela suffit à assurer le flux du programme.



(a) vue d'ensemble



(b) 1re halte



(c) 2e halte

FIGURE 3.27 – Déplacements consécutifs.

Les figures 3.27 et 3.28 montrent la simulation de la machine suivante :

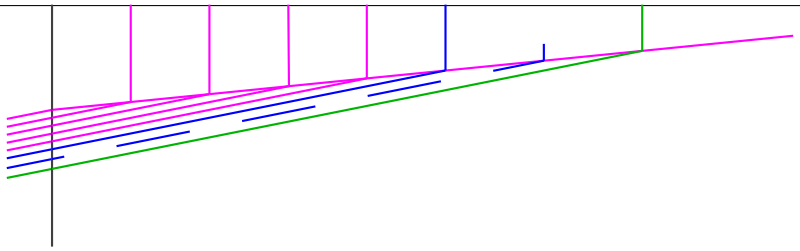
```
>;
<<;
>;
```

qui déplace la tête de la manière suivante : droite-gauche-gauche-droite. La figure 3.27a montre une vue d'ensemble. Les figures 3.27b et 3.27c montrent, respectivement, le premier et le second changement de direction.

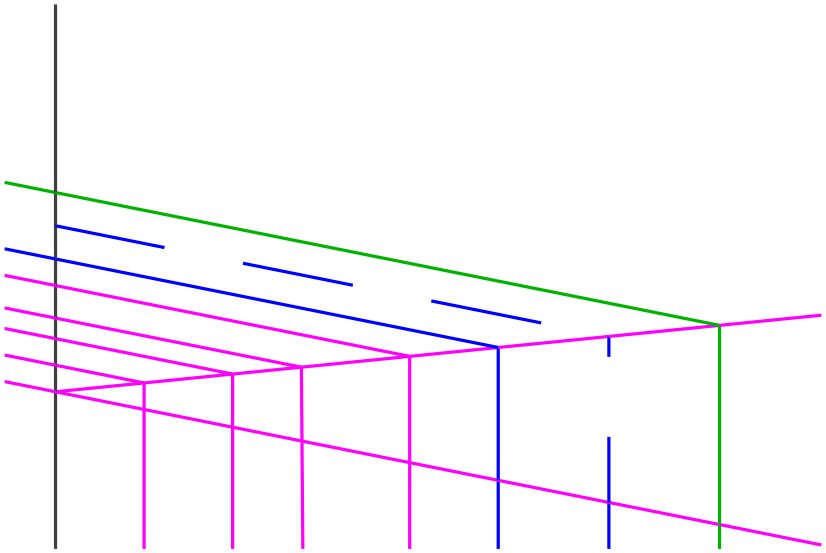
La figure 3.28 se focalise sur le premier changement de direction. La figure 3.28a montre la fin d'un mouvement, comme déjà présenté à la figure : 3.18. En haut de figure, on peut observer un début de tête complet : signal marqueur noir, puis viennent quatre signaux violets : signal programme, signal régulation, signal pendule (en très lent, imperceptible mouvement), et signal horloge. Enfin il y a les trois signaux du demi registre-unité : zéro, valeur et borne max.

La figure 3.28b montre la transition entre la fin d'un mouvement et le début d'un autre mouvement. Précisément, la collision entre le signal retour et le signal programme correspond à l'arrête entre le premier nœud mouvement "$>$;" et le suivant. La partie entrante de cette collision est la fin d'une primitive mouvement telle que vue en section 3.2.1, et la partie sortante de cette collision est le début d'une primitive mouvement.

Les nœuds de calculs requièrent l'enchaînement de deux primitives, lecture, suivi d'addition ou de soustraction, qui sont présentées en section 3.2.1. On peut voir la primitive de lecture comme ayant quatre ensembles de méta-signaux finaux : deux pour chacun des deux signaux retours, selon lequel est

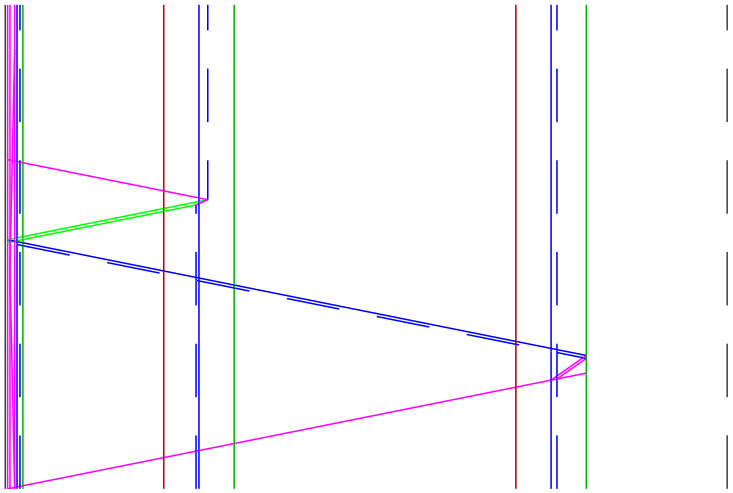


(a) fin d'un mouvement

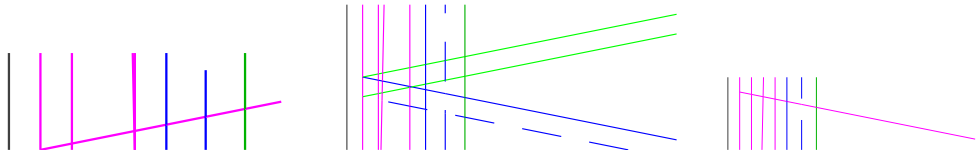


(b) début d'un mouvement : 1re redirection

FIGURE 3.28 – Vues rapprochées de la transition entre deux mouvements.



(a) vue d'ensemble



(b) 1re collision programme (c) 2e collision programme (d) 3e collision programme

FIGURE 3.29 – Un nœud de calcul de type ajout $a+ = 2 * h;$.

lu en premier (c'est-à-dire selon le singe de la valeur lue). À ces ensembles correspondent deux ensembles de méta-signaux initiaux de la primitive d'addition et deux pour la soustraction, qui ensemble forment les règles des collisions intermédiaires d'un nœud de calcul telles qu'illustrées Fig. 3.29c.

La figure 3.29 montre un nœud de calcul, de type ajout, spécifiquement $a+ = 2 * h;$. Ce nœud est décomposé en une primitive de lecture de la valeur $2 * h$ puis en une primitive addition ou soustraction de la valeur à l'accumulateur. Le choix entre ajout et retrait d'une valeur positive pour la seconde primitive s'effectue au niveau du signal programme, en fonction du signe de la valeur lue–dénoté par l'ordre des signaux retour–et de l'opération à effectuer. La figure 3.29a montre une vue d'ensemble, les figures 3.29b,

3.29c et 3.29d détaillent chacune des collisions entre signaux retour et signaux programmes.

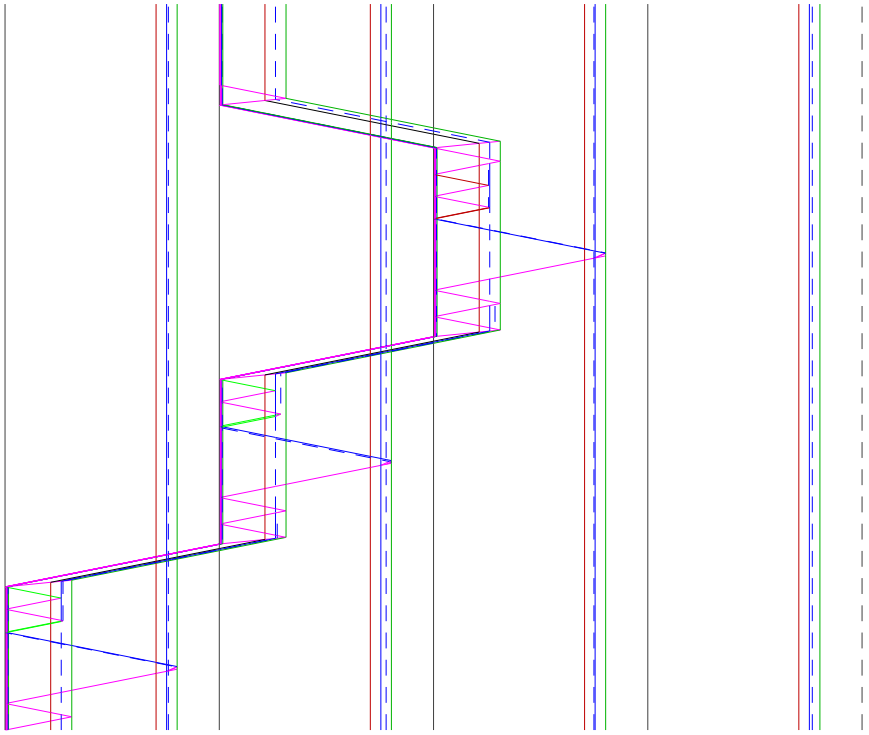
3.3 Exemples

```
Warning: could not resolve as a node ID:  _#0  , trying with the next label:  loop_#0
Initial configuration:
[..., 0, 0, >3<, 9, -2, 5, 0, 0, ...] focus (>3<) at 0

loop_#0 :       >;
        [..., 0, 0, 3, >9<, -2, 5, 0, 0, ...] focus (>9<) at 1  a -> 0
beginning_#0 :  a =0 ;  a -> 0
beginning_#1 :  a += 1 * h;     a -> 9
beginning_#2 :  Branch next_#0, next_#0, loop_#0;       a ->  9 > 0
loop_#0 :       >;
        [..., 0, 0, 3, 9, >-2<, 5, 0, 0, ...] focus (>-2<) at 2 a -> 9
beginning_#0 :  a =0 ;  a -> 0
beginning_#1 :  a += 1 * h;     a -> -2
beginning_#2 :  Branch next_#0, next_#0, loop_#0;       a ->  -2 < 0
next_#0 :       a =0 ;  a -> 0
next_#1 :       <;
        [..., 0, 0, 3, >9<, -2, 5, 0, 0, ...] focus (>9<) at 1  a -> 0
next_#2 :       end;    terminating node:  next_#2

Final configuration:
[..., 0, 0, 3, >9<, -2, 5, 0, 0, ...] focus (>9<) at 1
```

(a) interpréteur console



(b) simulation AGC

FIGURE 3.30 – Exemple de programme MTRL.

La figure 3.30 montre la trace d'un calcul de machine MTRL définie précédemment en figure 3.1, qui déplace la tête de lecture jusqu'à la dernière cellule contenant une valeur positive. L'état initial du ruban est $[3, 9, -2, 5]$. La figure 3.30a montre une interprétation console ; on y observe notamment que la ligne de code $a = h$ est décomposée en deux nœuds $a = 0$ et $a+ = 1 * h$. La figure 3.30b en montre une simulation par une machine AGC.

La figure 3.31 montre le diagramme d'une machine à signaux tel que défini précédemment 3.11, qui effectue un algorithme d'Euclide, sur les nombres 33/5 et 24/7, traduit dans la configuration initiale. Cet algorithme peut nécessiter un nombre arbitrairement long d'étapes – voire infini si l'entrée est deux nombres réels de ratio irrationnel, mais peut être simulé par une machine AGC en un temps fini, à l'aide de contractions.

FIGURE 3.31 – Simulation AGC d'une MTRL effectuant l'algorithme d'Euclide.

Chapitre 4

Méthode de tracé par Arbres Unaires Binaires

Dans ce chapitre, nous voyons d'abord des primitives structurelles pour tracer des arbres unaire-binaires. Cela est fait de manière très similaire à la machine vue au Chapitre 2 Section 2.3. Puis, nous voyons comment incorporer des machines MTRL pour dicter ce tracé. Ensuite nous voyons une série d'exemples. Enfin, nous caractérisons la puissance d'une telle machine.

Intégration MTRL à UB

Conceptuellement, l'état d'arrêt dans lequel se trouve une machine MTRL après un calcul peut être utilisé pour choisir la nature (degré) d'un sommet d'un arbre unaire-binaire, ainsi que spécifier un calcul subséquent à effectuer dans chacun de ses enfants. Pour définir précisément comment l'entrée d'une machine est choisie en fonction de la sortie de son père, on adjoint aux machines MTRL deux nouveaux types de nœud :

— nœud ***delay***, d'arité sortante 1 ;

— nœud ***split***, d'arité sortante 2 ; si le nœud est noté η, ses successeurs sont notés γ_η^l et γ_η^r.

Une machine MTRL s'articule avec le tracé d'un arbre unaire-binaire de la manière suivante : initialement la machine est lancée depuis son nœud de calcul initial.

Lorsqu'elle entre un nœud de type arrêt, cela correspond à un sommet de degré sortant 0 : le tracé depuis ce sommet s'arrête.

Lorsqu'elle entre un nœud η de type ***delay***, cela correspond à un sommet de degré sortant 1 : le calcul s'arrête et le sommet courant possède un unique fils. Au niveau de ce fils est effectué un calcul en utilisant la même machine MTRL, l'état final du ruban de son père est utilisé comme état initial du ruban, et le nœud γ_η est utilisé comme nœud initial.

Lorsqu'elle entre un nœud η de type ***delay***, cela correspond à un sommet de degré 2. Le calcul s'arrête et le sommet courant possède deux fils. Au niveau de chacun des fils, est effectué un calcul en utilisant la même machine MTRL, une copie de l'état final du ruban du père comme état initial du ruban des fils. Pour son fils gauche – respectivement droit, le nœud de calcul initial est γ_η^l– respectivement γ_η^r.

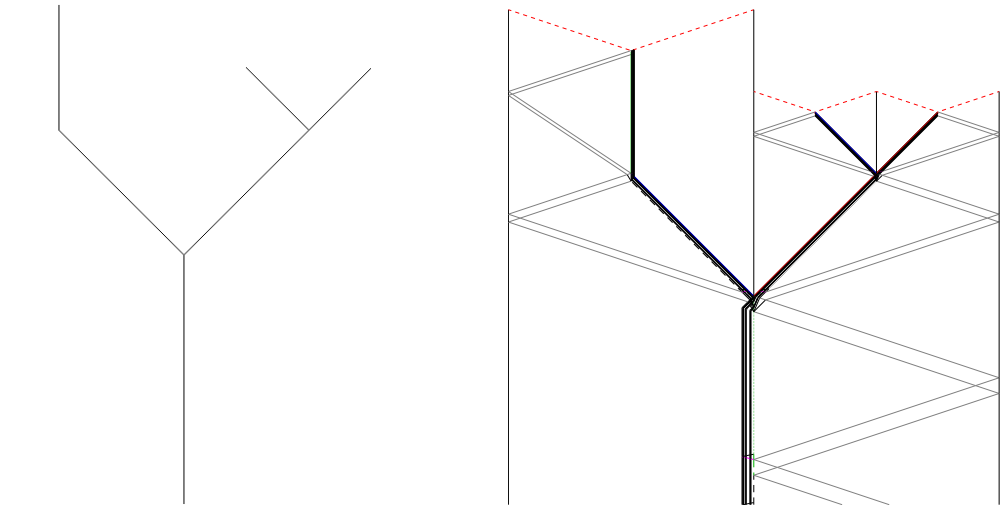
Enfin, lorsque la machine MTRL ne s'arrête pas, on convient que le tracé est indéterminé.

La figure 4.2 montre le tracé d'un arbre unaire-binaire tel que défini par le code à la figure 4.1. La figure 4.2a montre un tracé effectué directement à l'aide d'un interpréteur python (module tkinter) tandis que la figure 4.2b montre une simulation AGC de la même machine, traçant le même arbre.

Le tracé d'un arbre unaire-binaire par l'interpréteur python s'effectue simplement à l'aide d'une fonction récursive prenant en paramètre la position courante et un facteur d'échelle – la 2-profondeur du sommet courant. Simuler ce tracé à l'aide de machine AGC est l'objet de la première section de ce chapitre 4.1.

```
Delay LBLd;
LBLd:
Split LBLdl, LBLdr;
LBLdl:
Delay LBLend;
LBLdr:
Split LBLend, LBLend;
LBLend:
end;
```

FIGURE 4.1 – machine MTRL UB traçant un arbre fini (7 sommets)



(a) tracé de l'arbre unaire binaire (b) simulation agc

FIGURE 4.2 – Arbre unaire-binaire fini défini par une machine MTRL.

On appelle *machine MTRL UB* une machine MTRL utilisant des nœuds `delay` et `split`, définissant ainsi le tracé d'un arbre unaire-binaire en fonction d'une entrée. On appelle *machine MTRL UB AGC* une machine AGC dont le but est de tracer un arbre unaire-binaire en fonction des résultats d'une machine MTRL.

4.1 Construction AGC

4.1.1 Structure

Les terminologies `delay` et `split` sont reprises, pour désigner la construction d'un sommet de l'arbre unaire-binaire de degré respectif 1 ou 2 dans un diagramme AGC.

Généraux

Un *général* est une sous partie d'une configuration composé de :

— un signal principal ;

— un signal borne ;

— une machine MTRL entre le signal principal et le signal bordure.

La *largeur* d'un général est simplement la distance de son signal principal à sa borne.

Un *général vertical* est un général dont le signal principal et le signal borne sont de types respectifs tree et bound et de vitesse nulle.

Un *général vertical droit* est un général dont le signal principal est à droite du signal borne. Un *général vertical gauche* est un général dont le signal principal est à gauche du signal borne, et dont la machine MTRL est construite en miroir.

Un *général oblique droit* est un général dont le signal principal et le signal borne sont de types respectifs tree et bound et de vitesse 1 ; de plus, toutes les vitesses de sa machine MTRL sont augmentées de 1.

Un *général oblique gauche* est un général dont le signal principal et le signal borne sont de types respectifs tree et bound et de vitesse -1 ; son signal borne est à droite de son signal principal ; ses signaux et méta-signaux, y compris sa machine MTRL, sont construits en miroir du général oblique droit.

Dans le reste de cette sous-section, les machines MTRL des généraux ne sont pas représentées, pour la clarté des figures. Elles sont supposées à l'arrêt et redirigées de la même manière que le reste des signaux.

Configuration Initiale

La configuration initiale d'une machine MTRL UB AGC contient :

— deux signaux bordure – border, délimitant spatialement l'intervalle de travail ;

— un général vertical droit, dont le signal principal se situe exactement au milieu des bordures.

— deux signaux rebonds s'apprêtant à intercepter le général, en laissant tout de même le temps à la machine MTRL d'effectuer un premier calcul.

Délai

La Figure 4.3 montre comment un ***delay*** est initié depuis un général vertical droit. Le signal principal est de couleur verte, pour dénoter que c'est un signal principal vertical ; il est aussi en pointillés longs pour dénoter qu'il contient l'information ***delay***. Le signal bordure, à sa gauche, est vertical noir. Des signaux de rebond auxiliaires, à droite du signal principal et de types $\overleftarrow{\text{bounce}^{\text{bot}}}$ et $\overleftarrow{\text{bounce}^{\text{top}}}$, démarrent la manipulation ; aucune redirection n'étant nécessaire, ils rebondissent simplement – en des signaux de types $\overrightarrow{\text{bounce}^{\text{bot}}_{\text{slw}}}$ et $\overrightarrow{\text{bounce}^{\text{top}}_{\text{slw}}}$– vers les bordures pour effectuer le ***delay***. Un ***delay*** depuis un général vertical gauche se fait de manière symétrique.

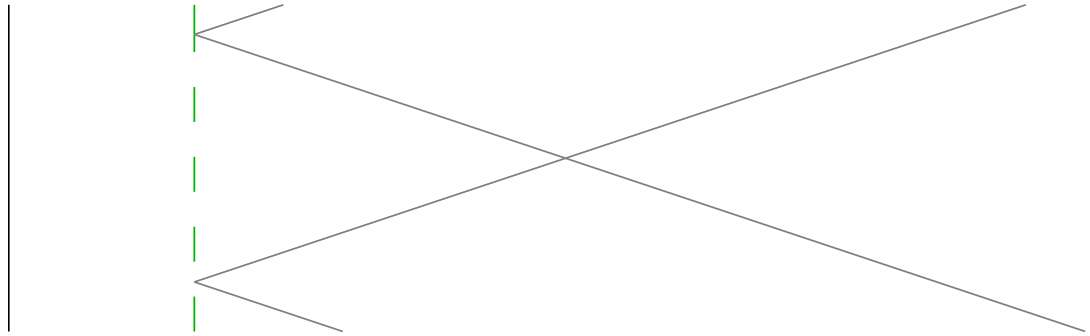


FIGURE 4.3 – Un ***delay*** depuis un général vertical droit.

La Figure 4.4 montre comment un ***delay*** est initié depuis un général oblique droit. Le signal principal est de couleur rouge, pour dénoter que c'est un signal principal allant vers la droite. Le signal bordure, à sa droite, est noir, et en pointillés long pour dénoter qu'il contient l'information ***delay***. Des signaux de rebond auxiliaires, à droite et de types $\overleftarrow{\text{bounce}^{\text{bot}}}$ et $\overleftarrow{\text{bounce}^{\text{top}}}$, démarrent la manipulation ; ils rebondissent vers les bordures – en des signaux de types $\overrightarrow{\text{bounce}^{\text{bot}}_{\text{slw}}}$ et $\overrightarrow{\text{bounce}^{\text{top}}_{\text{slw}}}$– pour effectuer le ***delay***. La machine est redirigée via une ***réfraction*** de sorte à croiser le signal principal. Le général de sorti est donc un général vertical droit. Les pentes sont choisies de sorte à réduire de moitié la largeur du général.

Un ***delay*** depuis un général oblique gauche se fait de manière symétrique.

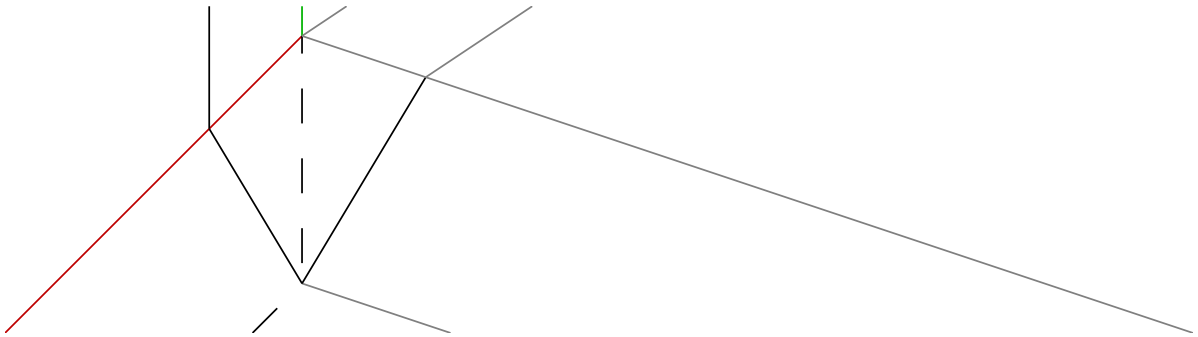


FIGURE 4.4 – Un ***delay*** depuis un général oblique droit.

Branchement

La Figure 4.5 montre comment un ***split*** est initié depuis un général oblique droit. Le signal principal initial est de couleur rouge, le signal bordure correspondant est noir et en pointillés courts pour dénoter qu'il porte l'information ***split*** pour le sommet à venir. Des signaux de rebond auxiliaires, de types $\overleftarrow{\text{bounce}^{\text{bot}}}$ et $\overleftarrow{\text{bounce}^{\text{top}}}$, démarrent la manipulation.

Celle-ci consiste en :

— une ***réflexion*** et une ***réfraction*** du général initial pour le dupliquer ;

— une ***réfraction*** supplémentaire de chaque côté permettant de réduire de moitié la largeur des généraux sortants – obliques droit et gauche, relativement au général entrant ;

— un signal rebond supérieur de chaque côté $\overleftarrow{\text{bounce}^{\text{top}}}$ et $\overrightarrow{\text{bounce}^{\text{top}}}$ démarrés au point d'intersection principal ;

— une construction de chaque côté permettant de créer les signaux rebond inférieurs $\overleftarrow{\text{bounce}^{\text{bot}}}$ et $\overrightarrow{\text{bounce}^{\text{bot}}}$ à la bonne hauteur, c'est-à-dire de sorte a correspondre à la largeur des nouveaux généraux réduite de moitié par rapport à l'ancien ;

— de chaque côté, un signal généré par le croisement du rebond et de la borne signalant que les opérations de duplication et redirection sont terminés – signal utilisé pour démarrer les calculs suivants.

Un ***split*** depuis un général vertical gauche se fait de manière symétrique.

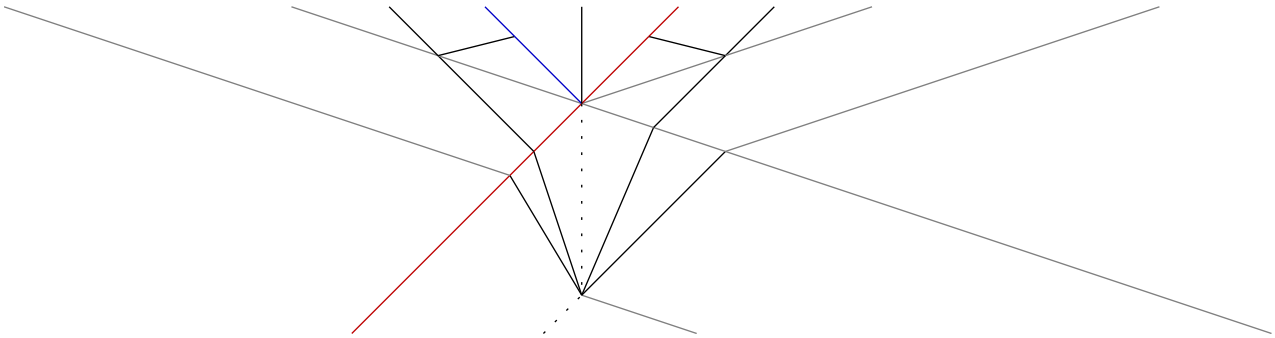


FIGURE 4.5 – Un ***split*** depuis un général oblique droit.

La Figure 4.6 montre comment un délai est initié depuis un général vertical gauche : Le général est réfracté en un général oblique gauche, puis tout se passe comme lors d'un ***split*** depuis un général oblique. Un ***split*** depuis un général vertical droit se fait de manière symétrique.

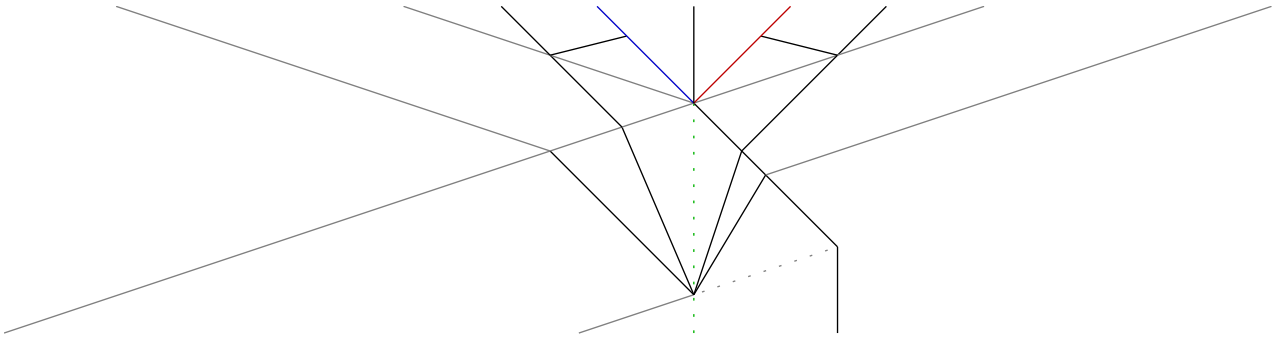


FIGURE 4.6 – Un ***split*** depuis un général vertical gauche.

Arrêt

Lors d'un nœud terminal, la machine doit s'arrêter. A cet effet deux signaux auxiliaires d'arrêt sont émis, un de chaque côté. On peut voir ces signaux sur la figure 4.2b, en pointillés roses. Ces signaux d'arrêt interrompent tous les autres signaux qu'ils interceptent, jusqu'au premier signal bordure, auxquels ces signaux d'arrêt stoppent. De plus un signal bordure frappé par deux signaux d'arrêts s'arrêtent eux aussi, puisqu'ils n'ont plus rien à border. Les signaux bordures initiaux peuvent être pris comme ayant déjà été chacun frappé une fois, puisqu'ils n'ont qu'un seul côté à border.

Incrustation

Initialement, la machine MTRL UB AGC d'un général est lancée depuis son nœud d'exécution initial.

Lorsqu'elle entre un nœud η de type arrêt, ***delay*** ou ***split***, elle s'arrête, et envoie une information au général. Cette information est stockée sur le signal rencontrant le signal rebond en premier, c'est-à-dire sur le signal principal pour les généraux verticaux et sur le signal borne pour les généraux obliques ; elle est dénotée par le style de ligne, en pointillés courts ou longs. Utilisant cette information, l'opération correcte parmi *arrêt*, ***delay*** ou ***split*** est effectuée.

Au sortir d'une opération de type ***delay*** ou ***split***, un signal est envoyé à la machine pour redémarrer. Les derniers signaux traversant chaque général que l'on peut observer aux figures 4.5 et 4.6 servent précisément à envoyé un signal pour redémarrer la machine MTRL suffisamment tard pour ne pas croiser de signal rebond. Celle-ci redémarre comme attendu depuis le nœud γ_η, γ_η^l ou γ_η^r (γ_η pour un général vertical après un ***delay***, γ_η^l pour une général oblique gauche après un ***split***, γ_η^r pour une général oblique droit).

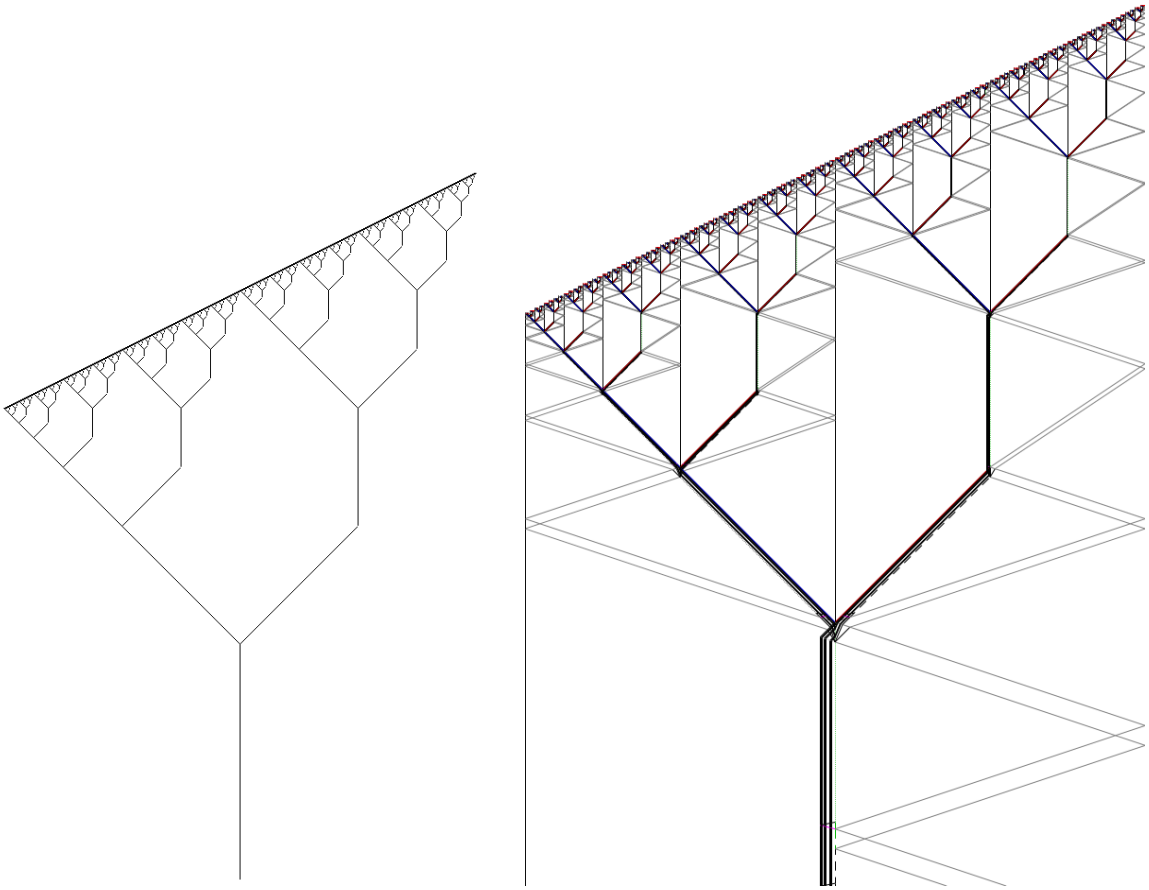
```
left:
Split left, right;
right:
Delay left;
machine_ubtTest_rec
```

FIGURE 4.7 – Code définissant récursivement un arbre UB infini.

4.2 Exemples

4.2.1 Premier Exemple

La machine dont le code est donné en figure 4.7 peut être décrite de la manière suivante : après chaque branche gauche s'ensuit un *split*, après chaque branche droite s'ensuit un *delay* puis un *split*. La figure 4.8 montre l'arbre produit : la figure 4.8a est dessinée directement par l'interpréteur, tandis que la figure 4.8b est le diagramme de la machine AGC correspondante.



(a) tracé de l'arbre unaire binaire (b) simulation agc

FIGURE 4.8 – Synchronisation en saccade particulière d'une ligne de fusilier résolut à l'aide d'une machine MTRL.

4.2.2 Paraboles

L'algorithme suivant 1 permet de tracer une parabole.

La paramétrisation par les valeurs $f(-1)$, $f(0)$ et $f(1)$ permet une actualisation simple et linéaire des paramètres après redécoupage. Si $f(x) = ax^2 + bx + c$, alors $a = \frac{f(-1)-2f(0)+f(1)}{2}$, ce qui permet de calculer l'accélération de manière linéaire. Lors du changement d'échelle, l'accélération est divisée par deux. En effet, temps et espace sont multipliés par deux, et l'accélération est homogène à une distance sur le carré d'un temps : en posant par exemple $g(x) = 2 \times f((x+1)/2)$, on a $g''(x) = f''(x)/2$. Enfin, on peut inverser la formule donnant l'accélération en fonction de 3 valeurs pour trouver une valeur à l'aide

Algorithme 1 Paraboles

Résultat
tracé du graphe d'une fonction $f : [-1\,;1] \to \mathbb{R}$, polynomiale de degré au plus 2 et telle que $f[-1\,;1] \geq 1$

Entrée
m $f(-1)$
z $f(0)$
p $f(1)$

fonction PARABOLE(m, z, p) :
$a \leftarrow (m - 2z + p)/2$ ▷ calcul de l'accélération
Si $a \leq 0$ **alors**
minoration $\leftarrow \min(m, p)$
Sinon, **si** $0 < a$
minoration $\leftarrow \min(2z - p, 2z - m)$
Fin si
Si minoration ≥ 2 **alors**
`delay`(PARABOLE$(m - 1, z - 1, p - 1)$)
Sinon
`split`(PARABOLE$(2m - 1, m + z - a - 1, 2z - 1)$, PARABOLE$(2z - 1, z + p - a - 1, 2p - 1)$)
Fin si
Fin fonction

de l'accélération et de deux autres valeurs :

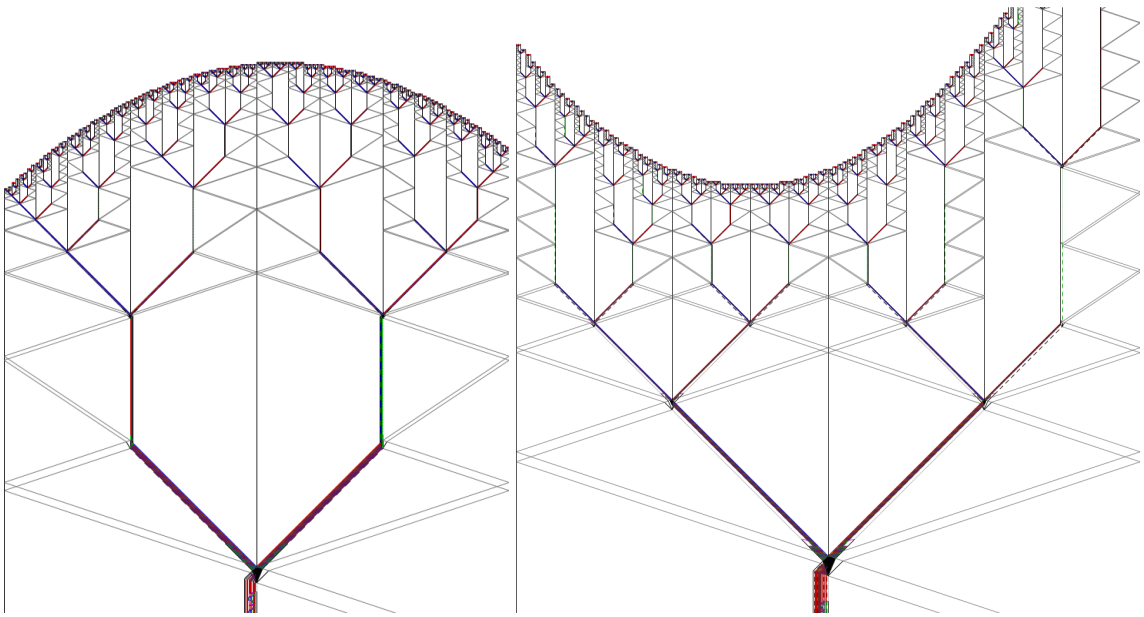
$$g(0) = \frac{g(-1) + 2a + g(1)}{2} \tag{4.1}$$

ce qui permet d'exprimer $g(0)$, la valeur à interpoler après mise à l'échelle, en fonction des anciens paramètres $f(0)$ et $f(1)$.

Le principe de fonctionnement est le même que pour le tracé d'un segment vu au chapitre 2 : on utilise des étapes `delay` tant que l'on peut, c'est à dire tant que la fonction cible est au dessus de 2, et des étapes `split` sinon.

Une parabole concave – d'accélération négative – est située au dessus de sa corde sur le segment $[-1\,;1]$; son minimum sur ce segment se situe donc à l'une des extrémités de celui-ci. Dans le cas d'une parabole convexe, la valeur minimale atteinte de la parabole ne peut pas de manière général être calculée linéairement, mais elle peut être minorée de manière suffisamment proche.

Pour traduire cet algorithme en une machine MTRL, on suppose par exemple que les valeurs m, z, p sont initialement sur le ruban aux positions $-1, 0, 1$, et on utilise la cellule 2 pour stocker l'accélération.



(a) tracé d'une parabole concave (b) tracé d'une parabole convexe

FIGURE 4.9 – Paraboles.

4.2.3 Courbe du Blanc-Manger/ Fonctions de Takagi

Une adaptation infime de l'algorithme de la parabole permet d'obtenir la courbe du blanc-manger : l'équation 4.2 suffit à définir cette fonction, qui se trouve être fractal, continue en tout point et dérivable en aucun.

$$f(0) = \frac{f(-1) + 4a + f(1)}{2} \tag{4.2}$$

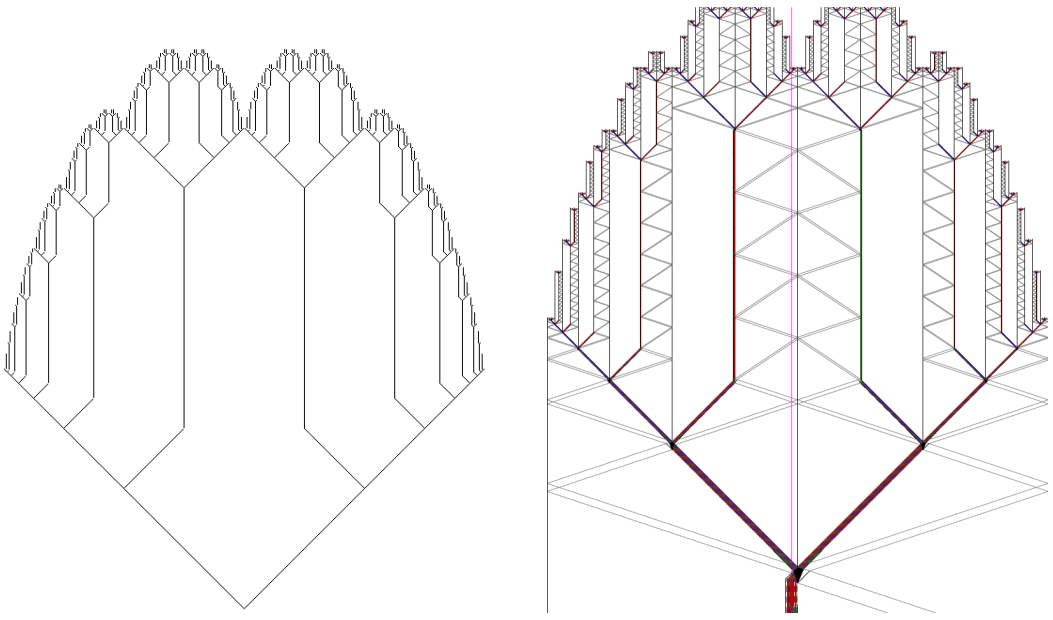
Plus généralement, en jouant sur le coefficient par lequel on multiplie la pseudo accélération a, on obtient des fonctions de Takagi paramétrées [34].

Algorithme 2 Blanc-manger

Résultat
tracé du graphe d'une fonction $f : [-1\,;1] \to \mathbb{R}$, polynomiale de degré au plus 2 et telle que $f[-1\,;1] \geq 1$

Entrée
m $f(-1)$
z $f(0)$
p $f(1)$

fonction PARABOLE(m, z, p) :
$a \leftarrow (m - 2z + p)/2$ ▷ calcul de l'accélération
Si $a \leq 0$ **alors**
minoration $\leftarrow \min(m, p)$
Sinon, **si** $0 < a$
minoration $\leftarrow \min(2z - p, 2z - m)$
Fin si
Si minoration ≥ 2 **alors**
`delay`(PARABOLE$(m - 1, z - 1, p - 1)$)
Sinon
`split`(PARABOLE$(2m - 1, m + z - 2a - 1, 2z - 1)$, PARABOLE$(2z - 1, z + p - 2a - 1, 2p - 1)$)
Fin si
Fin fonction



(a) tracé par interpréteur MTRL (b) simulation agc

FIGURE 4.10 – Tracé du graphe de la fonction blanc-manger.

4.3 Une Surjection Effective de l'Ensemble des Réels sur l'Ensemble des Arbres Unaires-Binaires

Nous montrons dans cette section qu'il est possible de tracer n'importe quel arbre unaire-binaire à l'aide d'une machine à signaux encodant l'arbre dans la configuration initiale.

4.3.1 Notations pour Manipulations de Nombres Ternaires

On commence par identifier un nombre réel entre 0 inclus et 1 exclu à la suite des chiffres de son écriture ternaire propre. Dans le reste de cette section, les nombres à virgules sont supposés écrits en base ternaire.

Pour tout x de $[0\,;1[$, on note x_i son i-ème chiffre après la virgule, de sorte que :

$$x = \sum_{1}^{\infty} x_i \times 3^{-i}$$

On note $a.b$ la concaténation, où a et b sont des chiffres ou des suites de chiffres, a étant fini. Par associativité, on note $a.b.c$ la suite $a.(b.c) = (a.b).c$. Par exemple, si a est un chiffre ternaire (0, 1 ou 2, identifiable à une suite finie de longueur 1) et x un nombre de $[0\,;1[$:

$$a.x = \frac{a+x}{3} = a \times 3^{-1} + \sum_{2}^{\infty} x_{i+1} \times 3^{-i}$$

Pour tout nombre x de $[0\,;1[$ et tout entier k, on note $x_{>k}$ le décalage de k chiffres vers la gauche de x, c'est à dire le nombre dont les décimales sont celles de x en ignorant les k-premières ; formellement, on a :

$$x_{>k} = \sum_{1}^{\infty} x_{i+k} \times 3^{-i}$$

ou encore, de manière équivalente : $x_{>0} = x$ et $x_{>k} = x_k.x_{>k+1}$.

Pour tout couple de nombres x et y dans $[0\,;1[$, on note $x \ltimes y$ l'opération d'enchevêtrement, c'est à dire le nombre dont les chiffres de positions impairs sont, dans l'ordre, ceux de x, et ceux de positions pairs sont ceux de y ; formellement :

$$x \ltimes y = \sum_{1}^{\infty} x_i \times 3^{-2\times i+1} + \sum_{1}^{\infty} y_i \times 3^{-2\times i}$$

L'enchevêtrement est prioritaire sur la concaténation : $a.x \ltimes y = a.(x \ltimes y)$ désigne la concaténation de $x \ltimes y$ à a tandis que $(a.x) \ltimes y$ désigne l'enchevêtrement de $a.x$ avec y. Une autre manière encore de définir l'enchevêtrement est via la relation de récurrence suivante, où a est un chiffre :

$$(a.x) \ltimes y = a.(y \ltimes x)$$

4.3.2 Encodage

Définition 17. Soit F un fonction de $[0\,;1[$ dans UB. On dit que F est *calculable* s'il existe une machine de Turing à quatre rubans :

- un premier ruban, en lecture seule et utilisant trois symboles, encode l'entrée en écriture ternaire ;
- un deuxième ruban, en lecture seule et unidirectionnel avec deux symboles, correspondant à gauche et droite ;
- un troisième ruban, en écriture seule et unidirectionnel, avec trois symboles `stop`, `delay` et `split` ;
- un quatrième ruban de calcul auxiliaire, utilisable librement ;

cette machine écrivant sur la sortie (ruban trois) la nature des sommets rencontrés en descendant un chemin de l'arbre image par F en suivant, aux sommets de degrés 2, les directions prescrites par le deuxième ruban.

Theorème 10. *Il existe une surjection calculable de* $[0\,;1[$ *dans UB.*

L'objet de cette partie est d'expliciter une telle surjection.

On pose UB^e l'ensemble des arbres unaire-binaires finis de feuilles étiquetées dans $[0\,;1[$. On note $G \approx G'$ lorsque les arbres G et G' on la même structure, mais diffèrent possiblement sur leurs étiquettes.

Soit x un nombre de $[0\,;1[$. On va définir, par récurrence, une suite $(T_i(x))_{i\in\mathbb{N}}$ de UB^e– dont les étiquettes vont encoder l'arbre qu'il reste à développer depuis chaque feuille.
$T_0(x)$ est l'arbre trivial à un seul sommet, que l'on étiquette par x.
On définit ensuite $T_1(x)$ de la manière suivante :

- si $x_1 = 2$, on s'arrête : $T_1(x) = T_0(x)$;
- si $x_1 = 1$, on attend (`delay`) : $T_1(x)$ est l'arbre à deux sommets, la racine et son fils, ce dernier que l'on étiquette par $x_{>1}$;
- si $x_1 = 0$, on branche (`split`) : soit y et z tels que $x_{>1} = y \ltimes z$, $T_1(x)$ est l'arbre constitué d'une racine ayant deux enfants, le gauche étant étiqueté par y, le droit par z.

$T_{n+1}(x)$ est l'arbre obtenu à partir de $T_n(x)$ en remplaçant chaque feuille d'étiquette y par $T_1(y)$. On dit encore qu'on *développe* chaque feuille.

On appelle *feuille terminale* une feuille d'un arbre $G \in UB^e$ d'étiquette commençant par 2 (les feuilles inchangée par développement).
Enfin, on définit $T(x)$ comme l'union croissante des $T_i(x)$ – en oubliant les étiquettes terminales, ce qui permet d'associer, à tout nombre de $[0\,;1[$, un arbre unaire-binaire.

$T(x)$ vérifie la relation de récurrence suivante :

— si $x_0 = 2$, $T(x)$ est l'arbre trivial à un sommet ;

— si $x_0 = 1$, la racine de $T(x)$ a un fils, dont part le sous-arbre $T(x_{>1})$;

— si $x_0 = 0$, étant donnés y et z tels que $x_{>1} = y \ltimes z$, la racine de $T(x)$ a deux fils, un gauche et un droit, dont partent, respectivement, les sous-arbres $T(y)$ et $T(z)$.

Remarque 12. Intuitivement, tout les $T_n(x)$ pour n entiers contiennent toute l'information pour restituer x. La suite de cette section formalise et prouve cette assertion, dans l'optique de prouver que tous les arbres unaire-binaires sont atteignables.

Soit ϕ la fonction de UB^e dans UB^e qui à un arbre G associe l'arbre obtenu en remplaçant dans G chaque feuille de profondeur maximale, par $T_1(x)$ où x est l'étiquette de la feuille.

Soit ψ la fonction de UB^e dans UB^e qui à un arbre G associe l'arbre binaire obtenu en remplaçant dans G chaque sous-arbre de hauteur 1 de profondeur maximale, comme suit :

— si le sous-arbre est une racine et une feuille d'étiquette x, il est remplacé par une feuille d'étiquette $1.x$.

— si le sous-arbre est une racine et deux feuilles, d'étiquettes y et z, respectivement à l'ordre gauche droite, il est remplacé par une feuille d'étiquette $0.y \ltimes z$.

Soit $E_n \subset UB^e$ l'ensemble des arbres unaire-binaires étiquetés de hauteur inférieure ou égale à n dont les feuilles non terminales sont de profondeur exactement n.
Soit E l'union de ces ensembles :

$$E = \bigcup_{n\in\mathbb{N}} E_n$$

Remarque 13. L'intersection $E_n \cap E_{n+1}$ est une manière de noter l'ensemble des arbres unaire-binaires étiquetés de hauteur au plus n dont toutes les feuilles sont terminales ; ses éléments sont inchangés par ϕ :

$$\{G \in UB^e | \phi(G) = G\} = \bigcup_{n\in\mathbb{N}} E_n \cap E_{n+1}$$

Remarque 14. Les fonctions ϕ et ψ sont stables sur E ; précisément :

$$\forall n \in \mathbb{N} \quad \phi(E_n) \subseteq E_{n+1}$$

et

$$\forall n \in \mathbb{N} \quad \psi(E_{n+1}) \subseteq E_n$$

Remarque 15.

$$\forall n \in \mathbb{N}\, \forall G \in E_n, \phi(G) \neq G \quad \psi(\phi(G)) = G$$

$$\forall G \in E, \psi(G) \neq G \quad \phi(\psi(G)) = G$$

Démonstration. Soit $n \in \mathbb{N}$ et $G \in E_n$ tel que $\phi(G) \neq G$. Comme $\phi(G) \neq G$, l'arbre G possède au moins une feuille non terminale, il est de hauteur exactement n, et l'arbre $\phi(G)$ est de hauteur exactement $n+1$. Les sous-arbres de hauteur 1 de $\phi(G)$ sont donc de profondeur n, et correspondent exactement au développement des feuilles de G. On a donc bien $\psi(\phi(G)) = G$.

Soit $G \in E$ tel que $\psi(G) \neq G$. Soit n la hauteur de G, de sorte qu'on a $G \in E_n$. Comme $\psi(G) \neq G$, on a $n \geq 1$ et $\psi(G)$ est de hauteur $n-1$ exactement. Les feuilles non terminales de $\psi(G)$ sont donc de hauteur $n-1$ et correspondent exactement au remplacement de sous-arbre effectué par ψ. On a donc bien $\phi(\psi(G)) = G$ □ □

Remarque 16. On a :

$$\forall x \in [0\,;1[\, \forall n \in \mathbb{N} \quad T_n(x) \in E_n$$

$$\forall x \in [0\,;1[\, \forall n \in \mathbb{N} \quad T_{n+1}(x) = \phi(T_n(x))$$

Démonstration. Par récurrence : soit $x \in [0\,;1[$. L'arbre $T_0(x)$ ayant une seule feuille, on a bien :

$$T_0(x) \in E_0$$

On a aussi, par définition :

$$T_1(x) = \phi(T_0(x))$$

Soit un entier n tel que : $T_n(x) \in E_n$ et $T_{n+1}(x) = \phi(T_n(x))$

D'après la remarque 14, $T_{n+1}(x) \in E_{n+1}$

$T_{n+2}(x)$ est obtenu en développant toutes les feuilles de $T_{n+1}(x)$, tandis que ϕ consiste en développer les feuilles de profondeur maximales. Puisque $T_{n+1}(x)$ est dans E, cela revient au même, et on a bien $T_{n+2}(x) = \phi(T_{n+1}(x))$. □ □

Remarque 17 (Cas particulier d'arbres unaire-binaires finis). Soit G un arbre unaire-binaire fini, de hauteur n. En notant G' l'arbre obtenu en étiquetant les feuilles de G par $0, 2$ (dénotant des feuilles terminales) on a :

$$G' \in E_n \cap E_{n+1}$$

$$G' = \phi^n(\psi^n(G'))$$

de plus, l'arbre $\psi^n(G')$ est trivial, et si on note $T_0^{-1}(\psi^n(G'))$ son étiquette, on a bien :

$$G' = \phi^n(T_0(T_0^{-1}(\psi^n(G')))) = T_n(T_0^{-1}(\psi^n(G'))$$

et même :

$$G' = \phi^m(T_0(T_0^{-1}(\psi^n(G')))) = T_m(T_0^{-1}(\psi^n(G'))$$

pour tout nombre m supérieur à n. On a donc encore :

$$G = T(T_0^{-1}(\psi^n(G'))$$

Ce qui prouve que l'image de T contient les arbres de hauteur finie.

Soit G un arbre unaire-binaire. Soit $G_n \in UB^e$ l'arbre préfixe de G constitué des sommets de profondeur n ou moindre, aux feuilles duquel sont adjointes les étiquettes $\overline{0,2}^3$, de sorte que :

$$G = \bigcup_1^\infty \uparrow G_n$$

et de sorte que toutes ses feuilles soient terminales, c'est-à-dire :

$$\forall n \in \mathbb{N} \quad G_n \in E_n \cap E_{n+1}$$

À l'aide de la remarque 15, on peut montrer que :

$$G_n = \phi^n(\psi^n(G_n))$$

L'arbre étiqueté $\psi^n(G_n)$ est trivial, on peut donc en extraire l'étiquette $T_0^{-1}(\psi^n(G_n))$ et écrire :

$$G_n = \phi^n(T_0(T_0^{-1}(\psi^n(G_n)))) = T_n(T_0^{-1}(\psi^n(G_n))$$

Notons $x^n = T_0^{-1}(\psi^n(G_n))$. On observe qu'à tout indice i fixé, la suite $(x_i^n)_{n\in\mathbb{N}^*}$ peut changer au plus deux fois de valeur : le terme initial de cette suite est $x_i^0 = 0$ – excepté $x_0^0 = 2$ qui rentre dans le cas suivant. Cette suite prend la valeur 2 lorsque cet indice correspond pour la première fois à un sommet de G_n ; dans ce cas, cet indice i correspondra au même sommet dans tous les arbres préfixes subséquents G_m avec $m > n$. Enfin entre G_n et G_{n+1}, cette suite peut prendre n'importe quelle valeur 0, 1 ou 2 lorsque le sommet auquel elle correspond est traité (ses enfants éventuels apparaissent), et ne change plus.

Cela implique que la suite $(x_i^n)_{n\in\mathbb{N}^*}$ converge, et donc que la suite de suites de chiffres $(x^n)_{n\in\mathbb{N}^*}$ elle aussi converge, simplement, point à point, vers une suite de chiffres x. Le choix de ne pas utiliser le chiffre 2 pour les nœuds non terminaux implique que x n'est pas un développement ternaire impropre et peut donc même être interprété comme un nombre. Aussi, du fait que chaque nœud est traité aussitôt, on peut même assurer la vitesse de convergence : $\forall n \in \mathbb{N}^*, x_n = x_n^n$.

On remarque que G_n et $\psi(G_{n+1})$ ont la même structure. Il en va de même pour $T_n(x^n)$ et $T_n(x^{n+1})$, et plus généralement $T_n(x^n) \approx T_n(x)$

Pour conclure, on a donc :

$$G = \bigcup_1^\infty \uparrow G_n = \bigcup_1^\infty \uparrow T_n(T_0^{-1}(\psi^n(G_n))) \approx \bigcup_1^\infty \uparrow T_n(x) = T(x)$$

Donc G est bien dans l'image de T qui est donc surjective, d'inverse à droite :

$$G \mapsto \lim_{n\to\infty} T_0^{-1}(\psi^n(G_n))$$

où la limite a lieu point à point dans le développement ternaire et, a fortiori, aussi dans $\mathbb{R}$ muni de la topologie usuelle.

Calculabilité de l'encodage

Pour calculer un certain $T(x)$, il suffit de garder trace du nombre d de sommets de degré 2 traversés puis de suivre et de recopier les instructions des deux premiers rubans, en ne lisant du premier qu'un symbole sur 2^d. T est donc bien calculable.

4.3.3 Machine de Tracer d'Arbre Unaire-Binaire Universelle

Machine Universelle

Theorème 11. *Il existe une machine MTRL UB universelle, c'est-à-dire capable de tracer n'importe quel arbre unaire-binaire, en fonction de son entrée.*

En effet, la fonction calculable T peut être convertie en machine MTRL. La décomposition d'un nombre en son développement ternaire propre peut s'effectuer à l'aide de multiplications par 3 et de comparaisons avec des constantes. L'Algo. 3 détaille l'algorithme de calcul de T.

Algorithme 3 Un algorithme de tracé universel.

```
1: Entrée
2:     x       nombre réel de [0 ; 1[
3:
4: Résultat
5:     tracé de l'arbre T(x)
6:
7: d ← 1
8: fonction PRINCIPAL :                          ▷ on saute les chiffres ne nous concernant pas
9:     Pour k = 1, d − 1, faire :
10:        x ← 3 ∗ x
11:        Tant que 1 ≤ x, faire :
12:            x ← x − 1
13:        Fin tant que
14:    Fin pour
15:    x ← 3 ∗ x
16:    Si 2 ≤ x alors                                ▷ arrêt
17:        x ← x − 2 renvoyer
18:    Sinon, si 1 ≤ x                               ▷ delay
19:        x ← x − 1
20:        delay(PRINCIPAL)
21:    Sinon                                         ▷ split
22:        d ← 2 ∗ n
23:        split(GAUCHE,PRINCIPAL)
24:    Fin si
25: Fin fonction
26: fonction GAUCHE :
27:    Pour k = 1, d/2, faire :
28:        x ← x/3
29:    Fin pour
30:    PRINCIPAL
31: Fin fonction
```

Machine Rationnelle Universelle

La machine universelle du théorème 11 de la section précédente repose sur le codage, via un nombre réel, d'une quantité d'information infinie en entrée. Ce peut être vu comme recevoir la solution d'un problème au lieu de la calculer, et n'avoir qu'à la lire. Une manière de limiter la quantité d'information en entrée est d'imposer que l'entrée ne contienne que des nombres rationnels, en nombre fini, et que les constantes de la machine MTRL utilisée soient aussi rationnelles. Cela correspond à se limiter aux machines AGC rationnelles, avec une correspondance entrée-entrée et constantes-vitesses. On parle alors de *MTRL rationnelle*, et de *MTRL UB rationnelle*.

Theorème 12. *Les arbres traçables par des MTRL UB rationnelles sont exactement les $T(x)$ pour x calculable – au sens $n \mapsto x_n$ calculable. De plus, il existe une machine MTRL UB universelle capable de tracer n'importe lequel de ces arbres unaire-binaires, en fonction de son entrée.*

En effet il suffit, en reprenant la machine de la section précédentes, au lieu de lire x, de le calculer au fur et à mesure :

Algorithme 4 Un algorithme de tracé universel pour les machines rationnelles.

Entrée
    x une fonction calculable de $NaturalSet \to \{0,1,2\}$ qui n'est pas constante égal à 2 après un certain rang – on exclut les développements impropres

Résultat
    tracé de l'arbre $T(x)$

$p \leftarrow 0$
$d \leftarrow 1$
fonction PRINCIPAL : ▷ on saute les chiffres ne nous concernant pas
    $p \leftarrow p + d$
    $xp \leftarrow x(p)$
    Si $xp = 2$ **alors renvoyer** ▷ arrêt
    Sinon, **si** $xp = 1$ ▷ `delay`
        `delay`(PRINCIPAL)
    Sinon ▷ `split`
        $d \leftarrow 2 * d$
        `split`(GAUCHE,PRINCIPAL)
    Fin si
Fin fonction
fonction GAUCHE :
    $p \leftarrow p - d/2$
    PRINCIPAL
Fin fonction

Puisque le modèle des machines MTRL est Turing complet et qu'il existe une machine de Turing universelle, il existe une machine qui non seulement implémente cet algorithme mais aussi peut, seule, calculer n'importe quel x calculable.

C'est aussi le mieux que l'on puisse faire avec des machines MTRL UB. En effet, les MTRL rationnelles sont Turing équivalentes. Si on note G un arbre tracé par une MTRL UB rationnelle et G_n l'arbre préfixe de G composé des sommets de profondeur au plus n, G_n est calculable, donc $T^{-1}(G_n)$ est calculable, et $T^{-1}(G)$ est calculable, autrement dit, il existe x calculable tel que $G = T(x)$.

4.4 Tracés d'Arbres Unaire-Binaires

4.4.1 Caractérisation des Accumulations des Tracés d'Arbres Unaire-Binaires sur des Fonctions Continues

Fonctions en Escalier Rationnelles Dyadiques et Arbres Unaire-Binaires

On rappelle que la *2-profondeur* d'un sommet d'un arbre unaire-binaire est son nombre d'ancêtres de degré sortant 2. La *2-hauteur* d'un arbre est la 2-profondeur maximale de ses sommets.

Définition 18 (fonction en escalier rationnelle dyadique)**.** On appelle *fonction en escalier* les fonctions constantes par morceaux. De plus on qualifie une telle fonction de *rationnelle dyadique* si les positions et hauteurs des marches sont des nombres rationnels dyadiques et les marches sont de largeur non nulle.

Formellement, f est une fonction en escalier rationnelle dyadique d'un intervalle $[a\,;b]$ s'il existe un nombre entier N et une subdivision finie

$$a = s_0 < s_1 < s_2 < ... < u_N = b$$

telle que :

- $\forall n \in [\![1\,;N-1]\!]\ s_n \in D$
- f est constante à valeur dyadique sur les $]s_n\,;s_{n+1}[$ $(n \in [\![0\,;N-1]\!])$
- $f(s_n) \in f(]s_{n-1}\,;s_n[) \cup f(]s_n\,;s_{n+1}[)$ pour $n \in [\![1\,;N-1]\!]$, et $f(a)$ a la même valeur que sur $]s_0\,;s_1[$, et $f(b)$ que sur $]s_{N-1}\,;s_N[$.

La condition pas de marche de largeur nulle est arbitraire, mais est choisi pour coller au plus près de ce que peuvent faire les arbre unaire-binaires.

Définition 19. Un diagramme de machine à signaux ou un tracé d'arbre unaire-binaire *s'accumule* sur une fonction en escalier rationnelle dyadique f lorsque l'ensemble d'accumulation du diagramme ou du tracé est la fermeture du graphe de f.

Cette nouvelle définition d'accumulation sur une fonction permet d'inclure des fonctions non continues malgré le fait qu'un ensemble d'accumulation soit fermé et que les graphes de fonctions fermés sont exactement les graphes de fonctions continues. En échange, l'ensemble d'accumulation ne suffit pas a déterminer la fonction en escalier rationnelle dyadique en jeu. Toutefois, il la détermine à l'intérieur des marches – les $]s_n\,;s_{n+1}[$, il la détermine donc de manière unique aux valeurs des bordures de marche près. Ces bordures – les s_n – sont en nombre fini et le choix de leur valeur est binaire : on rattache la bordure à la marche d'après ou d'avant – $f(s_n) = f((s_{n-1}+s_n)/2) \in f(]s_{n-1}\,;s_n[)$ ou $f(s_n) = f((s_n+s_{n+1})/2) \in f(]s_n\,;s_{n+1}[)$.

Définition 20. Par convention, lorsque l'on doit resituer une fonction en escalier rationnelle dyadique à l'aide d'un ensemble d'accumulation, on prend la plus petite possible – $\forall n, f(s_n) = \min(f(]s_{n-1}\,;s_n[) \cup f(]s_n\,;s_{n+1}[))$. On dit qu'une une telle fonction en escalier est *à valeurs de bordures de marche minimales* ou encore *à bordures de marche minimales.*

Remarque 18. Soit G un arbre dont le tracé s'accumule sur une fonction de $[-1\,;1]$ dans $\mathbb{R}$. G n'a pas de sommet d'arité sortante nulle – une telle feuille empêche l'accumulation sur tout un segment.

Lemme 13 (intervalles d'influences)**.** *Soit un arbre unaire-binaire G infini sans feuille et de tracé borné.*

Il existe d'une part une bijection entre l'ensemble des sommets de G d'arité sortante 2 et les nombres rationnels dyadiques $D \cap]-1\,;+1[$.

D'autre part, on associe à ces sommets un intervalle d'influence *$]\frac{2n}{2^d}\,;\frac{2n+2}{2^d}[$, où $\frac{2n+1}{2^d}$ est l'abscisse du sommet. L'injection de l'ensemble des sommets de G d'arité sortante 2 dans l'ensemble des segments de $]-1\,;+1[$ ainsi obtenue est strictement croissante pour les relations de descendance et d'inclusion. Autrement dit, un sommet est descendant d'un autre (si et) seulement si son intervalle associé est inclus dans celui de l'autre sommet. Si on associe encore les sommets de n'importe quelle arité à leur intervalle d'influence, l'application n'est plus injective mais est encore strictement croissante.*

Démonstration. Le travail a déjà été fait au chapitre 2, remarque 9 et au début de la preuve 2.1.2. □

Remarque 19. Soit un arbre unaire-binaire G infini sans feuille et de tracé borné et y un nombre dyadique. L'arbre G admet un nombre fini de sommets de tracé d'abscisse y.

Définition 21. Pour un $G \in UB$ un arbre fini de profondeur maximale p. On note $\phi^\infty(G)$ l'arbre obtenu en remplaçant les feuilles de G par des arbres binaires complets (tous les sommets sont d'arité sortante 2). Autrement dit, en notant $G(0)$ l'arbre obtenu en étiquetant toutes les feuilles par 0 :

$$\phi^\infty(G) = \bigcup_{n=0}^{\infty} \uparrow \phi^n(G(0)) = T(T_0^{-1}(\psi^p(G(0))))$$

Lemme 14. *Soit G un arbre unaire binaire fini. $\phi^\infty(G)$ s'accumule sur une fonction en escalier rationnelle dyadique f. De plus, les coordonnées (y,t) des sommets de $\phi^\infty(G)\backslash G$ de 2-profondeur d vérifient la condition :*

$$f(y) = t + \frac{1}{2^d}$$

En effet, $T(0)$ s'accumule sur la fonction constante égale à 1, et par construction, pour obtenir $\phi^\infty(G)$, on colle une copie réduite et translatée de $T(0)$ pour chaque feuille de G ; comme de plus ces réductions et translations sont rationnelles dyadiques, l'arbre $\phi^\infty(G)$ s'accumule donc bien sur une fonction en escalier rationnelle dyadique. La relation entre y, t et f est une simple quantification de ce collage.

Lemme 15. *Soit G un arbre unaire binaire fini, f la fonction accumulation de $\phi^\infty(G)$, et y un nombre de l'intervalle $[-1\,;1]$.*

Considérons les intervalles d'influence des feuilles de G. On sait que le nombre y soit appartient à un unique de ces intervalles, ou bien se trouve à la frontière de deux d'entre eux.

Dans le premier cas, notons (y',t') les coordonnées de la feuille à l'intervalle d'influence de laquelle y appartient : $y \in]y' - \frac{1}{2^{d'}}\,;y' + \frac{1}{2^{d'}}[$, où d' est sa 2-profondeur. On a :

$$f(y) = t' + \frac{1}{2^{d'}}$$

Dans second cas, notons (y',t') et (y'',t''), $y' \leq y''$, les coordonnées de la feuille à la frontière des intervalles d'influence desquels y se trouve : $y' + \frac{1}{2^{d'}} = y = y'' - \frac{1}{2^{d''}}$, où d' et d'' sont leur 2-profondeur respective. On a :

$$f(y) = \min(f(y'), f(y'')) = \min\left(t' + \frac{1}{2^{d'}}, t'' + \frac{1}{2^{d''}}\right)$$

Dans tous les cas, si on note d la 2-profondeur minimale des feuilles de G, il existe une feuille de G, de coordonnées (y',t') et de 2-profondeur d', telle que

$$|y - y'| \leq \frac{1}{2^d} \text{ et } f(y) = f(y') = t' + \frac{1}{2^{d'}}$$

Démonstration. Le premier cas est une conséquence directe du lemme précédent 14.

Le second cas est une conséquence directe du lemme précédent 14 et des bordures de marche minimales de f. □

Définition 22 (arbres unaires-binaires 2-complets)**.** On dit d'un arbre unaire-binaire qu'il est *2-complet* si toutes ses feuilles sont de même 2-profondeur.

Lemme 16. *Soit G un arbre unaire-binaire fini, et g la fonction en escalier rationnelle dyadique sur laquelle $\phi^\infty(G)$ s'accumule. Soit h une fonction en escalier rationnelle dyadique telle que $g \leq h$.*

Alors il existe un arbre unaire-binaire fini H tel que :

— *G est un arbre préfixe de H ;*

— *$\phi^\infty(H)$ s'accumule sur h.*

De plus, on est libre de choisir H 2-complet et de 2-hauteur strictement supérieure à celle de G.

Soit un entier d un entier supérieur strictement à la hauteur de G tel que les positions et hauteurs des marches de g et h soient des multiples entiers de $\frac{1}{2^d}$.

On obtient H de la manière suivante : on remplace chaque feuille de G de 2-profondeur inférieure à d par un sommet et ses deux enfants, et jusqu'à ce que toutes les feuilles soit de 2-profondeur d.

L'arbre ainsi obtenu, qui n'est pas encore H, se trouve entre G et $\phi^\infty(G)$ pour la relation préfixe, et est de 2-profondeur d strictement supérieure à celle de G. Les intervalles d'influence de ses feuilles induisent une division plus fine que la subdivision de h. Précisément, si (y, t) sont les coordonnées d'une telle feuille, alors h est constant sur $]y - \frac{1}{2^d}\,;y + \frac{1}{2^d}[$. De plus, d'après le lemme 14, on a :

$$t = g(y) - \frac{1}{2^d} \leq h(y) - \frac{1}{2^d}$$

De cet arbre, on remplace chaque feuille dont le tracé est de coordonnées (y, t) par un sommet et son unique enfant, jusqu'à ce que $t = h(y) - \frac{1}{2^d}$. Ceci est possible car on avance par pas de $\frac{1}{2^d}$ dont t et $h(y)$ sont des multiples entiers.

L'arbre obtenu H satisfait nos critères : il est 2-complet de 2-hauteur d strictement supérieure à celle de G, G en est bien un préfixe, et $\phi^\infty(H)$ s'accumule bien sur h.

Lemme 17. *Soit $(f_n)_{n\in\mathbb{N}}$ une suite croissante bornée de fonctions en escalier rationnelles dyadiques.*

Il existe une suite $(G_n)_{n\in\mathbb{N}}$ d'arbres unaires-binaires finis, croissante pour la relation préfixe, telle que :

$$\forall n \in \mathbb{N}, \phi^\infty(G_n) \text{ s'accumule sur } f_n$$

En outre, on peut imposer que les (G_n) soient 2-complets et de 2-hauteur croissante.

Une récurrence simple suffit, en utilisant le lemme 16 pour l'hérédité.

Fonctions d'Accumulation des Tracés d'Arbre Unaire-Binaire

Lemme 18 (les fonctions continues sont réglées par des fonctions en escalier rationnelles dyadiques)**.** *Toute fonction continue d'un compacte est limite uniforme d'une suite croissante de fonctions en escalier rationnelles dyadiques.*

Démonstration. Le raisonnement est sensiblement le même que pour la preuve du théorème de Heine, toute fonction continue d'une compacte est uniformément continue, et de sont corollaire : toute fonction continue d'un compacte est limite uniforme d'une suite de fonctions en escalier.

Soit f une fonction continue d'un compact, $[a\,;b]$ sans perte de généralité.

Soit $\epsilon > 0$. Pour tout y dans $[a\,;b]$, l'ensemble $f^{-1}(]f(y) - \epsilon\,;f(y) + \epsilon[)$, est, par continuité, ouvert, et contient y. On choisit, pour chaque y, une boule ouverte O_y contenant y et contenue dans $f^{-1}(]f(y) - \epsilon\,;f(y) + \epsilon[)$.

La famille $(O_y)_{y\in[a\,;b]}$ est un recouvrement d'ouvert de l'intervalle $[a\,;b]$. Par compacité, on peut en extraire un sous-recouvrement fini $(O_y)_{y\in I}$ où I est un partie finie de $[a\,;b]$.

On définit deux fonctions g^- et g^+ de $[a\,;b]$ dans $\mathbb{R}$ comme suit :

$$g^- : u \mapsto \min_{y\in I | u\in O_y} (f(y)) - 2\times\epsilon$$

$$g^+ : u \mapsto \max_{y\in I | u\in O_y} (f(y)) - \epsilon$$

Ce sont des fonctions en escalier.

Soit $u \in]a\,;b[$. Soit $y \in I$ tel que $u \in O_y$ et $g^+(u) = f(y) - \epsilon$, et soit $z \in I$ tel que $u \in O_z$ et $g^-(u) = f(z) - 2\epsilon$.

Par définition, $g^-(u) \leq f(y) - 2\epsilon = g^+(u) - \epsilon < g^+(u)$.

Comme $u \in O_y \subset f^{-1}(]f(y)-\epsilon\,;f(y)+\epsilon[)$, on a $f(u) \in]f(y)-\epsilon\,;f(y)+\epsilon[$. En particulier, $f(u) > f(y)-\epsilon = g^+(u)$

De manière, similaire, comme $u \in O_z \subset f^{-1}(]f(z)-\epsilon\,;f(z)+\epsilon[)$, on a $f(u) \in]f(z)-\epsilon\,;f(z)+\epsilon[$. En particulier, $f(u) < f(z)+\epsilon$ et $g^-(u) = f(z)-2\epsilon > f(u)-3\epsilon$.

On a $f(u)-3\epsilon < g^-(u) < g^+(u) < f(u)$, et comme u a été pris quelconque,

$$f - 3\epsilon < g^- < g^+ < f$$

En particulier, g^- et g^+ sont à distance uniforme au plus 3ϵ de f, et chacune suffit à démontrer le caractère réglée de f. En effet, "f est limite uniforme d'une suite de fonction en escalier" est équivalent à "il existe des fonctions en escalier arbitrairement proche de f au sens de la distance uniforme". Le fait de pouvoir prendre ces fonctions inférieures à f permet de prendre la suite de fonction convergeant vers f croissante si bon nous semble.

Pour terminer la preuve, il suffit de trouver une fonction en escalier rationnelle dyadique suffisamment proche de f, et pour ce faire, nous allons l'insérer entre g^- et g^+.

Observons tout d'abord que g^- et g^+ sont toutes deux semi-continues, g^- supérieurement et g^+ inférieurement ; à toutes fins utiles, la semi continuité supérieure de g^- en y s'exprime ainsi :

$$\forall \epsilon > 0, \exists \eta, \forall z \in [a\,;b], |z-y| < \eta \implies g^-(z) < g^-(y) + \eta$$

et la semi continuité inférieure de g^+ en y :

$$\forall \epsilon > 0, \exists \eta, \forall z \in [a\,;b], |z-y| < \eta \implies g^+(z) > g^+(y) - \eta$$

En effet, Soit $y \in [a\,;b]$. Soit $z \in I$ tel que $y \in O_z$ et $g^-(y) = f(z) - 2\epsilon$. Par définition, $g^- \leq f(z) - 2\epsilon$ sur O_z, autrement dit, g^- est majorée par $g^-(y)$ sur un voisinage de y, autrement dit, g^- admet des maxima locaux en tout point, et, a fortiori, est semi continue supérieurement. Pareillement, g^+ admet des minima locaux en tout point, et, a fortiori, est semi-continue inférieurement.

Si g^- et g^+ sont constantes, il suffit pour conclure de prendre une fonction constante de valeur rationnelle dyadique comprise entre g^- et g^+. Sinon, soit $a = u_0 < u_1 < u_2 < u_3 < ... < u_N = b$, $N \in \mathbb{N}$ et $N \geq 2$, une subdivision de l'intervalle $[a\,;b]$ suffisamment fine pour décrire g^- et g^+. Pour $n \in [\![1\,;N-2]\!]$, on choisit un nombre dyadique v_n dans l'intervalle $]u_n\,;u_{n+1}[$. On pose encore $v_0 = a$ et $v_{N-1} = b$.

On définit une fonction en escalier rationnelle dyadique h sur la subdivision $a = v_0 < v_1 < v_2 < ... < v_{N-1} = b$ de la manière suivante : pour $n \in [\![1\,;N-1]\!]$, $g^-(u_n) < g^+(u_n)$, et on peut prendre une valeur rationnelle dyadique h_n dans $]g^-(u_n)\,;g^+(u_n)[$, et définir, pour $y \in]v_{n-1}\,;v_n]$, $h(y) = h_n$ – on prend $h(a) = h_1$.

Soit $n \in [\![0\,;N-2]\!]$. La fonction g^- est constante sur $[u_n\,;v_n[$ et $]v_n\,;u_{n+1}]$, et admet un maximum local en v_n. L'abscisse v_n est donc position de maximum de g^- sur tout l'intervalle $]v_{n-1}\,;v_n]$. De même, sur $]v_{n-1}\,;v_n]$, g^+ a un minimum en u_n. Donc h est encadrée par g^- et g^+ sur $]v_{n-1}\,;v_n]$, et donc sur $[a\,;b]$, ce qui suffit à conclure. □

Theorème 19. *Un arbre unaire-binaire peut s'accumuler sur n'importe quel fonction continue. Formellement,*

$$\forall f \in [1\,;\infty[^{[-1\,;1]}, f \text{ continue}, \exists G \in UB, G \text{ s'accumule sur } f$$

Démonstration. Soit f une fonction continue de $[-1\,;1]$ dans $\mathbb{R}$ supérieure à 1. D'après le lemme 18, il existe une suite croissante $(g_n)_{n\in\mathbb{N}}$ de fonctions en escalier rationnelles dyadiques convergeant uniformément vers f.

Prenons, grâce au lemme 17, une suite $(G_n)_{n\in\mathbb{N}}$ d'arbres unaire-binaires finis, 2-complets, croissante pour la relation préfixe, et de 2-hauteur strictement croissante, telle que $\forall n \in \mathbb{N}, \phi^\infty(G_n)$ s'accumule sur g_n Considérons l'union croissante des G_n, que l'on note G. Comme les G_n sont 2-complets et de 2-hauteur croissante, G est infini sans feuille.

Soit y un nombre de l'intervalle $[-1\,;1]$ et soit une suite de sommets de G, tous distincts, de coordonnées $(y_m, t_m)_{m\in\mathbb{N}}$, et tel que $(y_m)_{m\in\mathbb{N}}$ est convergente et de limite y. On cherche à montrer que $(t_m)_{m\in\mathbb{N}}$ est convergente, de limite $f(y)$.

On pose $n(m)$ le nombre tel que (y_m, t_m) appartienne à $G_{n(m)+1} \backslash G_{n(m)}$. Parce-que tous les (y_m, t_m) sont distincts et chaque G_n est fini, on a $n(m) \to \infty$.

On considère (y'_m, t'_m) la feuille de $G_{n(m)}$ dont descend (y_m, t_m).

On pose $d(m)$ et $d'(m)$ les 2-profondeurs respectives de (y_m, t_m) et (y'_m, t'_m) dans G. Parce-que les G_n sont 2-complets et de 2-hauteur strictement croissante, et que $n(m) \to \infty$, on a $d'(m) \to \infty$ et donc $d(m) \to \infty$; par voie de conséquence, on a aussi $y'_m \to y$, puisque $y'_m \to y$ et $|y'_m - y_m| \leq 1/2d'(m)$

D'une part,

$$t_m \leq g_{n(m)+1}(y_m) \leq f(y_m) \to f(y)$$

La limite vient de la continuité de f en y, et la seconde inégalité vient de $\forall n, g_n \leq f$. La première inégalité découle du lemme 13 : tout sommet de G d'abscisse dans $]y_m - \frac{1}{2^{d(n)}}\,;y_m + \frac{1}{2^{d(n)}}[\setminus \{y_m\}$ est un

descendant de y_m, et en particulier toutes les feuilles de $G_{n(m)+1}$ d'abscisse dans $]y_m - \frac{1}{2^{d(n)}} ; y_m + \frac{1}{2^{d(n)}}[$ sont des descendants de y_m ; tous les sommets de $\phi^\infty(G_{n(m)+1})$ sont donc d'abscisse dans $]y_m - \frac{1}{2^{d(n)}} ; y_m + \frac{1}{2^{d(n)}}[\setminus \{y_m\}$ sont des descendants de y_m ; leur ordonnée est donc supérieure à t_m, et on conclut en passant à la limite le long d'une suite de tels sommets de $\phi^\infty(G_{n(m)+1})$ convergeant vers $(y_m, g_{n(m)+1}(y_m))$.

D'autre part,

$$f(y) - t_m = (f(y) - f(y'_m)) + (f(y'_m) - g_{n(m)}(y'_m)) + (g_{n(m)}(y'_m) - t'_m) + (t'_m - t_m)$$

Par continuité de f en y et convergence de y'_m vers y, on a $f(y) - f(y_m) \to 0$. Par convergence uniforme de g_n vers f, $f(y'_m) - g_{n(m)}(y'_m) \to 0$. D'après le lemme 14, $g_{n(m)}(y'_m) - t'_m = \frac{1}{2^{d'(m)}} \to 0$. Enfin, $t'_m - t_m \leq 0$. On a donc :

$$f(y) - t_m \leq (f(y) - f(y'_m)) + (f(y'_m) - g_{n(m)}(y'_m)) + (g_{n(m)}(y'_m) - t'_m) \to 0$$

t_m est donc encadré entre deux valeurs qui tendent vers $f(y)$, donc tend vers $f(y)$.

On vient de terminer de montrer que, étant donnés y un nombre de l'intervalle $[-1\,;1]$ et une suite de sommets de G, tous distincts, d'ordonnées $(y_m, t_m)_{m\in\mathbb{N}}$, telle que $(y_m)_{m\in\mathbb{N}}$ est convergente et de limite y, la suite $(t_m)_{m\in\mathbb{N}}$ est convergente, de limite $f(y)$.

En particulier, toute accumulation de G de coordonnées (y, t) vérifie $t = f(y)$ d'une part. D'autre part, pour toute abscisse y de $[a\,;b]$, puisqu'il existe une suite une suite de sommet de G d'abscisses convergeant vers y – par densité des nombre dyadique et en appliquant le lemme 13 sur l'arbre infini sans feuille G, $(y, f(y))$ est un point d'accumulation du tracé de G.

En d'autres termes, le tracé de l'arbre G s'accumule sur f. □

Corollaire 20. *Il existe une machine AGC universelle pour le tracé de fonctions continues, c'est-à-dire capable, en fonction de son entrée, de s'accumuler sur n'importe quelle fonction continue supérieure à* 1.

En effet, il suffit de reprendre la machine MTRL UB du théorème 11, de la traduire en machine AGC, et d'encoder en entrée un arbre unaire-binaire encodant la fonction continue à atteindre.

Corollaire 21. *Il existe une machine AGC rationnelle universelle pour le tracé d'arbres unaire-binaires calculables, c'est-à-dire que pour tout nombre x de $[0\,;1[$ calculable – au sens $n \mapsto x_n$ calculable, cette machine peut, en codant le calcul de x en entrée, tracer l'arbre $T(x)$; le cas échéant, le diagramme AGC obtenu s'accumule sur une fonction continue.*

En effet, il suffit de reprendre la machine MTRL UB rationnelle du théorème 12, de la traduire en machine AGC.

4.4.2 Caractérisation des Accumulations des Tracés d'Arbres Unaire-Binaires sur des Fonctions Semi-Continues

Une des principales raisons pour lesquelles cette thèse s'intéresse aux fonctions continues et que les ensembles d'accumulations sont fermés, et que le graphe d'une fonction est fermé si et seulement si celle-ci est continue.

On peut néanmoins se poser la question de lever cette restriction. Intuitivement, si on a un arbre unaire-binaire infini sans feuille G qui est union croissante d'une famille d'arbres finis (G_n), chaque $\phi^\infty(G_n)$ s'accumule sur une fonction en escalier rationnelle dyadique g_n. On a envie de dire que le tracé de G s'accumule sur la limite croissante, quand elle existe, de la suite (g_n). Puisque les g_n sont semi-continues inférieurement – les bordures de marche ont valeur minimum – leur limite croissante l'est encore.

Pour cette raison, cette partie, ainsi que le dernier chapitre 5, sont biaisés en faveur de l'utilisation de fonctions semi-continues inférieurement.

On peut imaginer qu'un autre choix pour la valeur par défaut aux bordures de marches des fonctions en escalier rationnelles dyadiques entraînerait un biais différent, mais seul le choix de la valeur minimale semble donner des résultats simples.

Le choix de la valeur maximale par exemple, résulte en des fonctions en escalier rationnelles dyadiques semi-continues supérieurement – car borne supérieure de combinaisons linéaires positives d'indicatrices de fermés. Il n'y a pas grand chose à dire des bornes supérieures de fonctions semi-continues supérieurement.

Définition 23 (accumulations et droite réelle achevée)**.** On étend la notion d'accumulation de la définition 8 ainsi que les notions en découlant de la section 1.1.2 du chapitre 1 en étendant le temps à la demi-droite réelle achevée : $\mathbb{T} = \mathbb{R}^+ \cup \{+\infty\}$. Un point $(y, +\infty)$ est donc encore un point d'accumulation si tout voisinage de ce point contient une infinité de collision. On rappelle une base de voisinages de $(y, +\infty) \in \mathbb{R} \times \mathbb{R}^+ \cup \{+\infty\}$ pour la topologie naturelle — topologie produit de la topologie usuelle de $\mathbb{R}$ et de la topologie induite par celle de l'ordre de $\mathbb{R} \cup \{+\infty, -\infty\}$:

$$]y - \epsilon\,; y + \epsilon[\times]A\,; +\infty], \epsilon > 0, N \in \mathbb{R}^+$$

Cette topologie peut encore se définir séquentiellement, à savoir que la suite $(y_n, t_n)_n$ converge vers $(y, +\infty)$ lorsque la suite $(y_n)_n$ converge vers y et que la $(t_n)_n$ diverge de limite $+\infty$.

Remarque 20. Les ensembles d'accumulations de $\mathbb{R} \times \mathbb{R}^+ \cup \{+\infty\}$ sont encore des fermés.

Définition 24 (accumulation hâtive sur une fonction)**.** Un diagramme de machines à signaux ou un tracé d'arbre unaire-binaire *s'accumule hâtivement* sur une fonction $f, f \in \mathbb{R}^{[-1\,;1]}$ si pour tout y dans $[-1\,;1]$, le point de coordonnées $(y, f(y))$ est un point d'accumulation du diagramme ou tracé, et qu'il est d'ordonnée minimale parmi les points d'accumulations d'abscisse y.

Définition 25 (accumulation tardive sur une fonction)**.** Un diagramme de machine à signaux ou un tracé d'arbre unaire-binaire *s'accumule tardivement* sur une une fonction $f, f \in (\mathbb{R} \cup \{+\infty\})^{[-1\,;1]}$ si pour tout y dans $[-1\,;1]$, le point de coordonnées $(y, f(y))$ est un point d'accumulation du diagramme ou tracé, et qu'il est d'ordonnée la borne supérieure des points d'accumulations d'abscisse y. Cette borne supérieure définissant $f(y)$ existe et est atteinte pour la topologie ordre.

Remarque 21. Les ensembles d'accumulations étant fermés, l'ensemble des points d'accumulation d'un diagramme ou d'un tracé ayant une certaine abscisse est lui aussi fermé, et dès lors que cet ensemble est non vide, il possède une ordonnée minimale, et s'il est aussi borné, il possède une ordonnée maximale finie.

Remarque 22. Ces nouvelles définitions d'accumulation sont reliées au précédentes comme suit :

- L'accumulation sur les fonctions continues telle que définie au chapitre 1, sous-section 1.3.1, est équivalente à la conjonction d'une accumulation hâtive et d'une accumulation tardive sur une même fonction continue.
- De manière générale, la définition 19 de l'accumulation sur des fonctions en escalier rationnelles dyadiques ne correspond ni à l'accumulation tardive, ni à celle hâtive.
- En revanche, l'accumulation sur les fonctions en escalier rationnelles dyadiques à bordure de marches minimale —respectivement maximale — correspond à l'accumulation hâtive — respectivement tardive.
- Les lemmes 14, 16, 17 et 18, sont encore vrais lorsque l'on utilise l'accumulation hâtive — respectivement tardive — pour les fonctions en escalier rationnelles dyadiques et que l'on impose des bordures de marches minimales — respectivement maximale.
- La remarque 18 est encore vrai pour l'accumulation hâtive et tardive.

Theorème 22. *Les fonctions qui sont accumulations hâtives, respectivement tardives, d'un diagramme ou d'un tracé sont semi-continues inférieurement, respectivement supérieurement.*

Démonstration. On rappelle qu'une fonction f à valeur dans $\mathbb{T} = \mathbb{R}$ ou $\mathbb{T} = \mathbb{R} \cup \{-\infty, +\infty\}$ est semi-continue inférieurement si et seulement si son épigraphe $\{(y,t) \in [-1\,;1] \times \mathbb{T} : t \geq f(y)\}$ est fermé. De manière similaire, une fonction f est semi-continue supérieurement si et seulement si son hypographe $\{(y,t) \in [-1\,;1] \times \mathbb{T} : t \leq f(y)\}$ est fermé.

Soit f une fonction qui est accumulation hâtive — respectivement tardive — d'un diagramme ou d'un tracé. Soit A l'ensemble d'accumulation correspondant.

L'ensemble A est un sous-ensemble fermé du compact $[-1\,;1] \times \mathbb{R} \cup \{-\infty, +\infty\}$ donc est lui-même compact.

L'épigraphe — respectivement l'hypographe — de f est $A + \{0\} \times [0\,;\infty]$ — respectivement $A + \{0\} \times [-\infty\,;0]$. C'est un compact de la droite achevée, car somme de compacts ; en effet, la fonction somme de $([-1\,;1] \times (\mathbb{R} \cup \{-\infty, +\infty\}))^2$ dans $[-1\,;1] \times (\mathbb{R} \cup \{-\infty, +\infty\})$ est continue sur son ensemble de définition, le produit cartésien de deux compact est compact et enfin l'image par une fonction continue d'un compacts est compact.

Cet épigraphe — respectivement hypographe — est donc encore un fermé de la droite réelle achevée, ainsi que de la droite réelle pour la topologie usuelle, ce qui conclut la preuve. □

Démonstration. Une preuve alternative n'utilisant pas la topologie de la droite réelle achevée est aussi possible.

Soit f une fonction qui est l'accumulation hâtive d'un diagramme ou d'un tracé. Soit (y,t) un point d'adhérence de l'épigraphe. Donnons nous $((y_n, t_n))_{n\in\mathbb{N}}$ une suite de l'épigraphe convergeant vers (y,t).

La suite (t_n) est convergente donc bornée. La suite $f(y_n)$ est donc bornée – par 1 à gauche et $\sup t_n$ à droite. Soit $(f(y_{\sigma(n)}))$ une sous-suite convergente de $f(y_n)$, et soit t' sa limite.

La suite $(y_{\sigma(n)}, f(y_{\sigma(n)}))$ est une suite convergente – de limite (y,t') – de points d'accumulations. C'est donc encore un point d'accumulation, et, comme f est hâtive, $f(y) \leq t'$. De plus, on a, pour tout n et par appartenance de (y_n, t_n) à l'épigraphe : $f(y_n) \leq t_n$. En passant à la limite le long de σ, $t' \leq t$.

On a $f(y) \leq t' \leq t$, donc (y,t) appartient à l'épigraphe de f, et ce pour tout point (y,t) dans l'adhérence de cet épigraphe. Ce dernier est donc fermé, et f est semi-continue inférieurement.

La démonstration pour les fonctions accumulations tardives est analogue. □

Theorème 23. *Soit G un arbre unaire-binaire.*

Si le tracé de G s'accumule hâtivement sur une fonction f, alors il s'accumule tardivement sur la fonction à valeur dans $\mathbb{R} \cup \{+\infty\}$ définie comme suit :

$$y \mapsto \lim_{\substack{h \to 0 \\ h>0}} (\sup f([y-h\,;y+h]))$$

Démonstration. On note $\limsup f$ la fonction : $y \mapsto \lim_{\substack{h \to 0 \\ h>0}} (\sup f([y-h\,;y+h]))$

Soit un point d'accumulation du tracé de G de coordonnées (y,t), $y \in [-1\,;1], t \in \mathbb{R}$. Soit $(y_n,t_n)_n$ une suite de coordonnées de sommets de G convergente de limite (y,t). Sans perte de généralité, les y_n peuvent être pris tous distincts de y –remarque 19. Soit un entier n. D'après le lemme 13, le sommet de coordonnées (y_n,t_n) possède un intervalle d'influence centré sur y_n de sorte que tout sommet de G d'abscisse dans cet intervalle est descendant de (y_n,t_n). En particulier $f(y_n) \geq t_n$. En prenant $h_n = |y-y_n|$, on a $t_n \leq f(y_n) \leq \sup f([y-h_n\,;y+h_n])$ Au passage à la limite, $t \leq \limsup f(y)$, et ce pour un point d'accumulation (y,t) quelconque. Pour tout y, $\limsup f(y)$ est donc un majorant des ordonnées des points d'accumulation d'abscisse y.

Soit $t, t \in \mathbb{R} \cup \{\infty\}$ un majorant des ordonnées des points d'accumulation d'abscisse y. Pour n entier, si $\sup f([y-1/n\,;y+1/n]) = \infty$ on choisit un y_n dans $[y-1/n\,;y+1/n]$ de sorte que $f(y_n) \geq n$; sinon on le choisit de sorte que $f(y_n) \geq \sup f([y-1/n\,;y+1/n]) - \frac{1}{n}$. Deux choses l'une, $\limsup f(y) = \infty$ ou bien $\limsup f(y) < \infty$.

Si $\limsup f(y) = \infty$, a fortiori, pour tout n, $\sup f([y-1/n\,;y+1/n]) = \infty$. La suite $(y_n, f(y_n))$ est une suite de coordonnées d'accumulations convergeant vers $(y,+\infty)$, qui est encore des coordonnées d'une accumulation. On a donc $t \geq +\infty = \limsup f(y)$

Si $\limsup f(y) < \infty$ alors, à partir d'un certain rang, $\sup f([y-1/n\,;y+1/n]) < \infty$. La suite $(y_n, f(y_n))$ est une suite bornée de coordonnées d'accumulations distinctes. Soit t' tel que (y,t') soit une valeur d'adhérence de cette suite. La paire (y,t') est encore les coordonnées d'une accumulation. On a donc $t' \leq t$. D'autre part, comme $f(y_n) \geq \sup f([y-1/n\,;y+1/n]) - \frac{1}{n}$, on a aussi $t' \geq \limsup f(y)$, donc $\limsup f(y) \leq t$.

Le nombre $\limsup f(y)$ est donc bien le plus petit majorant des ordonnées des points d'accumulation d'abscisse y. Autrement dit, le tracé de G s'accumule tardivement sur f. □

Corollaire 24. *Soit G un arbre unaire-binaire dont le tracé s'accumule hâtivement sur une fonction f, et tardivement sur une fonction g.*

$$\forall y \in [-1\,;1], f(y) = g(y) \iff f \text{ continue en } y$$

En particulier, f et g diffèrent sur un espace maigre.

Démonstration. L'égalité $f(y) = \limsup f(y)$ signifie "f est continue supérieurement". Par ailleurs, f est semi-continue donc de classe de Baire 1 donc l'ensemble de ses points de continuité est comaigre.

□

Remarque 23. Si le théorème 23 signifie que les accumulations hâtives d'arbres unaires-binaires détermine celles tardives, l'inverse n'est pas vrai :

Par exemple, $\limsup(1 - I_{\{0\}}) = 1 = \limsup(1)$.

Theorème 25. *Soit G un arbre unaire-binaire. Soit $(G_n)_{n\in\mathbb{N}}$ une suite de préfixes finis de G, croissante et de limite G – une telle suite existe toujours — remarque 8. Notons, pour n entier, g_n la fonction en escalier rationnelle dyadique sur laquelle s'accumule $\phi^\infty(G_n)$.*

Le diagramme de G s'accumule hâtivement sur une fonction si et seulement si G est infini sans feuille et la suite de fonctions $(g_n)_n$ est majorée par une fonction.

Le diagramme de G s'accumule alors sur la fonction $\sup_{n\in\mathbb{N}} g_n$, qui ne dépend pas du choix de la suite (G_n).

Démonstration. Soit G un arbre unaire-binaire et $(G_n)_{n\in\mathbb{N}}$ une suite de préfixes de G croissante et de limite G.

Sens direct : supposons que le tracé de G s'accumule sur une fonction f.

G est infini sans feuille d'après la remarque 18.

Supposons, par l'absurde, qu'il y ait un réel y et un entier n tels que $f(y) < g_n(y)$.

Soit d tel que $2 \times \frac{1}{2^d} \leq g_n(y) - f(y)$. Soit m un entier supérieur à n tel que les feuilles de G_m soient de 2-profondeur au moins d.

D'après le lemme 15, de deux choses l'une : y est dans l'intervalle d'influence d'une feuille de G_m de coordonnées (y',t') et de 2-profondeur d' ; ou bien y est à la frontière entre les intervalles d'influences de deux feuilles de G, de coordonnées et 2-profondeurs respectives (y',t') et d', et (y'',t'') et d'', $y' \leq y''$. Quitte à poser $(y'',t'',d'') = (y',t',d')$, on a :

$$y \in]y' - \frac{1}{2^{d'}}\,;y'' + \frac{1}{2^{d''}}[$$

et

$$g_m(y) = \min(g_m(y'), g_m(y''))$$

Il existe donc un voisinage $\mathcal{V}$ de y, à savoir $]y' - \frac{1}{2^{d'}} ; y'' + \frac{1}{2^{d''}}[$, tel que tout sommet d'abscisse dans $\mathcal{V}$ différente de y se trouve dans l'intervalle d'influence du sommet (t', y') ou de celui (t'', y''). Tout sommet d'abscisse dans $\mathcal{V}$ est donc d'ordonnée supérieure ou égale à $\min(t', t'')$

Par ailleurs, on a : $g_m(y') = t' + \frac{1}{2^{d'}} \leq t' + \frac{1}{2^d}$ et $g_m(y'') = t'' + \frac{1}{2^{d''}} \leq t'' + \frac{1}{2^d}$ – lemme 14, donc

$$\min(t', t'') \geq \min(g_m(y'), g_m(y'')) - \frac{1}{2^d} = g_m(y) - \frac{1}{2^d} \geq g_n(y) - \frac{1}{2^d} \geq f(y) + \frac{1}{2^d}$$

.

Donc tout sommet d'abscisse dans $\mathcal{V}$ est d'ordonnée supérieure ou égale $f(y) + \frac{1}{2^d}$. En particulier, le point $(y, f(y))$ qui est limite d'une suite de sommets doit vérifier $f(y) \geq f(y) + \frac{1}{2^d}$, ce qui est absurde.

On vient donc de montrer que $g_n \leq f$, et ce pour un entier n quelconque, ce qui termine la démonstration du sens directe.

Sens indirect : supposons que G soit infini sans feuille et que la suite de fonctions $(g_n)_n$ soit majorée par une fonction.

La suite $(g_n)_n$ est majorée donc convergente (simplement), on note sa limite g. Pour tout entier n, on note $d(n)$ la 2-profondeur minimale des feuilles de G_n. Comme G est infini sans feuille et que chacun des G_n est fini, $d(n) \to \infty$.

Soit y un nombre de $[-1\,;1]$. Pour chaque entier n et grâce au lemme 15, il existe une feuille de G_n, de coordonnées (y'_n, t'_n) et de 2-profondeur $d'_{y,n}$, telle que :

$$|y - y'_n| \leq \frac{1}{2^{d(n)}} \text{ et } g_n(y) = t'_n + \frac{1}{2^{d'_{y,n}}}$$

On a donc d'une part $y'_n \to y$. D'autre part :

$$g_n(y) - \frac{1}{2^{d(n)}} \leq g_n(y) - \frac{1}{2^{d'_{y,n}}} = t'_n \leq g_n(y) \leq g(y)$$

donc $t'_n \to g(y)$.

Donc le point $(y, g(y))$ est un point d'accumulation du tracé de G. A l'aide du lemme 21, on peut conclure que le tracé de G s'accumule hâtivement sur une fonction f.

Fin : la preuve du sens direct montre que $\sup g_n \leq f$. La preuve du sens indirecte montre que tout point de type $(y, \sup g_n(y))$ est point d'accumulation, donc, par accumulation hâtive sur f, $f \leq \sup g_n$. On a donc bien $f = \sup g_n$ □

Corollaire 26. *Réciproquement à la remarque 18, un arbre unaire-binaire G infini sans feuille et de tracé borné s'accumule sur une fonction.*

Lemme 27. *Toute fonction semi-continue inférieurement est la borne supérieure d'une suite fonctions en escalier rationnelles dyadiques de $[-1\,;1]$ à bordures de marches minimales.*

Démonstration. Soit f une fonction semi-continue inférieurement. Il existe une suite croissante de fonctions continues $(f_n)_{n\in\mathbb{N}}$ convergeant simplement vers f [35]. Pour chaque entier n, on prend, grâce au lemme 18, une fonction en escalier rationnelle dyadique g_n à distance uniforme au plus $\frac{1}{n}$ de $f_n - \frac{1}{n}$.

Soit $y \in [-1\,;1]$.

$$(g_n(y) - (f_n(y) - \frac{1}{n})) + (f_n(y) - f(y)) + f(y) \leq g_n(y) \leq f(y)$$

Or $|f(y) - f_n(y)| \to 0$ par convergence simple de f_n vers f, et $|g_n(y) - (f_n(y) - \frac{1}{n})| \leq \frac{1}{n}$, donc $g_n(y) \to f(y)$. □

Theorème 28. *Soit f une fonction semi-continue inférieurement.*
Il existe un arbre unaire-binaire G s'accumulant hâtivement sur f.

Démonstration. D'après le lemme 27, on peut se donner une suite $(g_n)_{n\in\mathbb{N}}$ croissante de fonctions en escalier rationnelles dyadiques, de borne supérieure f.

D'après le lemme 17, il existe une suite croissante d'arbres $(G_n)_{n\in\mathbb{N}}$ telle que, pour tout entier n, $\phi^\infty(G_n)$ s'accumule sur g_n ; on peut imposer de plus que ces arbres soit 2-complets et de 2-hauteur croissante, de sorte que leur union croissante, que l'on nomme G, est infinie sans feuille.

D'après le théorème 25, le tracé de l'arbre G s'accumule sur $\sup g_n = f$. □

Corollaire 29. *Il existe une machine AGC universelle, capable, en fonction de son entrée, de s'accumuler sur n'importe quelle fonction semi-continue inférieurement supérieure à* 1.

Il suffit de reprendre la machine MTRL UB du théorème 11, de la traduire en machine AGC, et d'encoder en entrée un arbre unaire-binaire encodant la fonction semi-continue inférieurement à atteindre.

Si toutes les fonctions semi-continues inférieurement sont hâtivement traçables par des arbres unaires-binaires, le résultat analogue et faux pour les fonctions semi-continues supérieurement .

Remarque 24. La fonction $I_{\{0\}} + 1$ est semi-continue supérieurement mais n'est accumulation tardive d'aucun arbre unaire-binaire.

On peut toutefois construire une machine AGC dont elle est l'accumulation tardive.

Chapitre 5

Caractérisations des Fonctions Accumulations de Diagrammes de Machines à Signaux

Au chapitre 4, notamment la dernière section 4.4, nous avons vu deux facteurs clés dans l'étude des ensembles d'accumulations et des courbes d'accumulations des diagrammes de machines à signaux : la quantité d'information disponible à l'instant initial, et la définition précise de l'accumulation d'un diagramme sur une fonction.

Nous allons faire deux choix pour chacun de ces facteurs : information réelle ou rationnelle (finie) et accumulation hâtive où tardive.

5.1 Accumulations Hâtives Réelles de Diagramme Machine à Signaux

Theorème (rappel)**.** *Toute fonction continue de* $[1\,;\infty[^{]-1\,;1[}$ *est l'accumulation d'un diagramme de machine à signaux 1.3.1.*

De plus, il existe une machine à signaux universelle dont les diagrammes, en faisant varier la configuration initiale, peuvent s'accumuler sur n'importe quelle fonction continue.

C'est le corollaire 20.

Theorème (rappel)**.** *L'ensemble des fonctions de* $[1\,;\infty]^{]-1\,;1[}$ *qui sont l'accumulation hâtive d'un diagramme de machine à signaux sont exactement les fonctions semi-continues* $[1\,;\infty]^{]-1\,;1[}$.

De plus, il existe une machine à signaux universelle dont les diagrammes, en faisant varier la configuration initiale, peuvent s'accumuler sur n'importe quelle fonction continue.

Le théorème 22 énonce que les fonctions qui sont accumulations hâtives sont nécessairement semi-continues inférieurement, tandis que le corollaire 29 stipule la machine universelle.

5.2 Accumulations Hâtives Rationnelles de Diagramme Machine à Signaux

Le corollaire 21 stipule que tous les arbres unaires-binaires calculables sont traçables par une machine à signaux, et implique donc qu'une large classe de fonctions semi-continues sont l'accumulation d'un diagramme de machines à signaux rationnelle. Dans cette section, nous proposons une réciproque partielle, et montrons que la puissance d'accumulation sur un graphe de fonction d'une large classe de machines à signaux rationnelles est exactement l'ensemble des suites croissantes de fonctions en escaliers calculables.

5.2.1 Diagramme Partiels

Définition 26 (diagrammes partiels)**.** Un *diagramme espace-temps partiel* $\mathcal{D}$ est une collection de configurations au cours du temps vérifiant les conditions suivantes :

1. la configuration du temps zéro, aussi appelée *configuration initiale*, est finie ;
2. tout point de l'espace-temps associé à un méta–signal par $\mathcal{D}$ appartient, sinon à la configuration initiale, à un signal (définition 6) de $\mathcal{D}$;
   - un signal de type μ est de pente l'inverse de $S(\mu)$ (le temps s'écoule vers le haut, les signaux ne peuvent pas être horizontaux) ;

- l'origine d'un signal de type μ est un point de la configuration initiale ou un point associé à une règle ρ tel que $\mu \in \rho^+$;
- la fin d'un tel signal, s'il en est, est associée ou bien à une règle ρ telle que $\mu \in \rho^-$ ou bien à la valeur $\oslash$, ou bien à une valeur d'accumulation ✺ ;

3. tout point de l'espace temps associé à une règle de collision ρ est situé sur la configuration initiale ou bien la fin d'un signal de type μ pour chaque μ dans ρ^-.
4. *traçage* : pour tout signal, tout chemin dont il est le dernier élément est fini ;
5. *accumulations* : un point est associé à ✺ si et seulement si c'est un point d'accumulation.

En d'autres termes, un diagramme partiel est un diagramme dont les collisions peuvent manquer tout ou partie de leur signaux sortant.

Remarque 25. Un diagramme défini de manière implicite est un diagramme partiel. En particulier, l'unique diagramme défini de manière constructive est, après ajout des valeurs d'accumulation aux points d'accumulation, un diagramme partiel.

Définition 27 (collision, signal entamé)**.** Une *collision entamée* de type ρ d'un diagramme partiel est un point du diagramme qui :

— est associé à une règle ρ, et

— il existe un élément μ de ρ^+ tel que ce point ne soit pas le début d'un signal de type μ.

Intuitivement, c'est une collision dont la sortie n'est pas achevée.

Un *signal entamé* de type μ d'un diagramme partiel $\mathcal{D}$ est un point du diagramme qui

— est associé à une règle ρ telle que $\mu \in \rho^+$ ou bien est associé au méta-signal μ et appartient à la configuration initiale, et

— n'est pas le début d'un signal de type μ.

Intuitivement, c'est un point qui devrait être le début d'un signal si le diagramme était complet mais qui ne l'est pas.

Remarque 26. Les signaux entamés sont isolés, c'est à dire qu'il en existe un voisinage ne contenant que la collision correspondante, ses signaux entrant, et une partie de ses signaux sortant.

En effet les points non isolés sont des points d'accumulation.

Remarque 27. Étant donné une configuration initiale c_0, la collection de configuration constituée de c_0 au temps 0 et associant n'importe quel autre point de l'espace-temps à la valeur vide $\oslash$ est un diagramme partiel.

Celui-ci possède un signal entamé pour chaque point de c_0 associé à un méta-signal, ainsi que pour méta-signal sortant de chaque point associé à une collision.

Remarque 28. Les notions de signal, chemin, collision, profondeur de signal et profondeur de collision ont été définis pour les collections de configurations 6, 10, et s'appliquent donc encore aux diagrammes partiels. En particulier, les signaux entamés sont des points de la configuration initiale ou des collisions, et on peut y étendre la notion de profondeur.

Par ailleurs, la remarque 4 et sa preuve s'appliquent encore aux diagrammes partiels : la profondeur des signaux et collisions entamés sont finis.

Remarque 29 (Diagramme comme collection de signaux et de collisions). La condition 2 de la définition 26 stipule : tout point de l'espace-temps associé à un méta–signal par $\mathcal{D}$ appartient à un signal ou à la configuration initiale.

On en déduit que les diagrammes partiels sont entièrement décrits par la collection de leur signaux et collisions - quitte à rajouter des collisions fictives ou à considérer des signaux ponctuels.

Définition 28. On note $\mathcal{C}ol$ l'ensemble $\mathbb{R}\times\mathbb{T} \times R$ des collisions possibles d'un diagramme espace-temps. On note $\mathcal{S}ig$ l'ensemble :

$$\{([(x,t)\,;(x',t')],\mu) \in \mathcal{P}(\mathbb{R}\times\mathbb{T}) \times M | S(\mu)(t'-t) = x'-x\}$$

des signaux bornés possibles d'un diagramme espace-temps. On note $\mathcal{S}ig'$ l'ensemble

$$\{[(x,t)\,;(\Delta x,\Delta t)[|S(\mu)\times\Delta t = \Delta x\}\mathbb{R}\times\mathbb{T}\times M$$

des signaux non bornés – demi-droites – possibles d'un diagramme espace-temps.

On peut voir les diagrammes partiels comme des sous-ensembles particuliers de $\mathcal{C}ol \sqcup \mathcal{S}ig \sqcup \mathcal{S}ig'$.

Définition 29. Un *diagramme partiel fini* est un diagramme partiel ayant un nombre fini de signaux et de collisions (et donc de signaux entamés).

Définition 30 (relation d'ordre entre diagrammes)**.** On munit les diagrammes partiels de la relation d'ordre suivante : $\mathcal{D} \subseteq \mathcal{D}'$ si et seulement si l'ensemble des signaux et collisions de $\mathcal{D}$ est inclus dans celui de $\mathcal{D}'$.

Définition 31 (transitions)**.** Les diagrammes et diagrammes partiels ont au plus un nombre dénombrable de signaux et collisions (remarque 5). Tout diagramme partiel peut donc être vue comme une suite de signaux et collisions.

On groupe ensemble chaque collision avec ses signaux entrants, et on appelle les groupes ainsi formés des *transitions*. Une *transition* τ est ou bien un signal sans fin ou bien une collision ainsi que ses signaux entrants : $\tau \in \mathcal{S}ig' \sqcup (\mathcal{C}ol \times \mathcal{S}ig^{(\mathbb{N})})$. La profondeur d'un transition est la *profondeur* ou bien de son unique signal ou bien de sa collision. Tout diagramme partiel peut donc être vue comme une suite de transitions.

Définition 32 (suite de transitions)**.** Soit $N \in \mathbb{N} \cup \{\infty\}$. On dit d'une suite de transition $(\tau_n)_{n<N}$ qu'elle est un *représentant* d'un diagramme partiel $\mathcal{D}$ lorsqu'elle satisfait les conditions suivantes :

— le diagramme partiel $\mathcal{D}$, vu comme ensemble de signaux et de collisions, est égal à l'union des transitions τ_n – chacune vue comme un ensemble d'au plus une collision et d'au moins un signal – à laquelle on ajoute les valeurs d'accumulation pertinentes ;

— les signaux et collisions de la suite $(\tau_n)_{n<N}$ apparaissent par ordre d'influence, c'est à dire :

$$\forall m, n \in \mathbb{N}\ \forall a \in \tau_m\ \forall b \in \tau_n \text{ il existe un chemin de } a \text{ vers } b \implies m \leq n$$

.

Si de plus les éléments de $(\tau_n)_{n<N}$ apparaissent par ordre croissant de profondeur, temps et espace, cette suite est dite *représentant canonique* de $\mathcal{D}$.

Le nombre d'éléments d'un diagramme d'une certaine profondeur étant fini (remarque 5), tout diagramme possède un représentant canonique.

Pour toute suite de transitions $(\tau'_n)_{n<N}$, on note, quand il existe, $\mathcal{D}((\tau'_n)_{n<N})$ le diagramme partiel qu'elle représente.

Remarque 30. Soit $\mathcal{D}$ un diagramme partiel de représentant $(\tau'_n)_{n<N}$, $N \in \mathbb{N} \cup \{\infty\}$, tel qu'aucun signal borné n'ai pour fin une valeur d'accumulation ✺. Toute sous-suite $(\tau'_m)_{m\leq n}$, avec $n < N$, représente un diagramme partiel fini.

Remarque 31. Soient $\mathcal{D}$ et $\mathcal{D}'$ deux diagrammes partiels infinis, et soient $(\tau_n)_{n\in\mathbb{N}}$ le représentant canonique de $\mathcal{D}$. Si $\mathcal{D}$ et $\mathcal{D}'$ sont distincts, alors il existe un entier N tel que la suite finie $(\tau_n)_{n\leq N}$ ne soit pas préfixe du représentant canonique de $\mathcal{D}'$.

Un des intérêts de la notion de représentant canonique est donc de différencier différents diagrammes à l'aide de préfixes finis.

Définition 33 (diagramme partiel calculable)**.** Une suite de transitions rationnelles $(\tau_n)_{n<N}$ est dite *calculable* si, pour un certain encodage des transitions rationnelles en mots finis, la fonction $n \mapsto \tau_n$ est calculable.

Un diagramme partiel est dit *semi-calculable* s'il admet un représentant calculable. Un diagramme partiel est dit *calculable* lorsque son représentant canonique est calculable.

Définition 34 (diagramme calculables)**.** Un diagramme défini de manière constructive ou implicite est dit *semi-calculable* lorsqu'il est semi-calculable en temps que diagramme partiel.

Un diagramme défini de manière constructive ou implicite est dit *calculable* si son représentant canonique $(\tau_n)_{n<N}$ est calculable et si de plus il existe une fonction $\sigma : \mathbb{N} \to \mathbb{N}$, calculable, telles que : pour tout $n < N$, les collisions entamées de $(\tau_m)_{m\leq n}$ ne sont plus collisions entamées de $(\tau_m)_{m\leq\sigma(n)}$

Remarque 32. Les diagrammes définis de manière constructive sont calculables.

5.2.2 Espace d'Influence

Définition 35 (cônes d'influences émises, reçues – [12] chapitre 3)**.** Étant donnée une machine à signaux $\mathcal{A}$ et ses vitesses de méta-signaux minimale et maximale S_{min} et S_{max}, on définit, pour une position (x, t), les grandeurs suivantes :

$$\begin{aligned} J(x,t) &= \{(x',t') | 0 \leq (S_{max}(t \in J^-(b) - t') - x + x').(x - x' - S_{min}(t' - t))\} \\ J^-(x,t) &= J(x,t) \cap \mathbb{R} \times [0\,;t] \\ J^+(x,t) &= J(x,t) \cap \mathbb{R} \times [t\,;\infty[\end{aligned}$$

On appelle $J(x,t)$ le *cône d'influences issues* de la position (x,t), $J^-(x,t)$ le *cône d'influences reçues* et $J^+(x,t)$ le *cône d'influences émises*.

Remarque 33. Soient deux positions a et b. On a la suite d'équivalences :

$$J^-(a) \subseteq J^-(b) \iff a \in J^-(b) \iff b \in J^+(a) \iff J^+(b) \subseteq J^+(a)$$

De plus, si a influence b, c'est-à- dire s'il existe un chemin de signaux reliant a à b, alors $a \in J^-(b)$.

Définition 36 (espaces d'influences émises et reçues d'un diagramme partiel)**.** Soit $\mathcal{D}$ un diagramme partiel et soit (x,t) les coordonnées d'une un collision entamée de $\mathcal{D}$. *L'espace d'influences émises* par la collision entamée de coordonnées (x,t) , noté $\mathcal{I}^+_{\mathcal{D}}(x,t)$, est défini comme l'espace des points potentiellement atteignables, depuis cette collision, à l'aide d'un chemin fini de signaux n'intersectant pas $\mathcal{D}$:

$\mathcal{I}^+_{\mathcal{D}}(x,t) = \{(x',t') | \exists N \in \mathbb{N}, \exists([(x_n,t_n)\,;(x_{n+1},t_{n+1})])_{n\leq[\![1\,;N]\!]} \in \mathcal{S}ig^{[\![1\,;N]\!]},$
$\forall n \in [\![1\,;N]\!] \mathcal{D}(](x_n,t_n)\,;(x_{n+1},t_{n+1})]) = \{\oslash\}\}$

L'espace d'influences émises d'un diagramme $\mathcal{D}$, noté $\mathcal{I}^+(\mathcal{D})$, est l'union des espaces d'influence de ses collisions entamées.

Remarque 34. les espaces d'influences relatifs à un diagramme partiel sont inclus dans les cônes d'influences :

$\forall \mathcal{D}, x, t\ \mathcal{I}^+_{\mathcal{D}}(x,t) \subseteq J^+(x,t)$

5.2.3 Caractérisation Partielle

Theorème 30. *Soit $\mathcal{A}$ une machine à signaux, et soient S_{min} et S_{max} les vitesses respectivement minimale et maximale de méta-signaux. On suppose que l'ensemble des vitesses est non trivial, c'est-à-dire $S_{min} \neq S_{max}$.*

Soit $\mathcal{D}$ un diagramme infini implicite de $\mathcal{A}$ dont aucun signal n'a pour fin une valeur d'accumulation. Soit $(\tau_n)_{n\in\mathbb{N}}$ une suite de transitions le représentant. On pose, pour n entier, $\mathcal{D}_n = \mathcal{D}((\tau_m)_{1\leq m\leq n})$.

Si $\mathcal{D}$ s'accumule hâtivement sur une fonction f de $[-1\,;1]$ dans $\mathbb{R}$, alors l'épigraphe de f, à savoir $\{(x,t)|f(x)\leq t\}$, est inclus dans $\bigcap_{n\in\mathbb{N}} \overline{\mathcal{I}^+(\mathcal{D}_n)}$, où $\overline{\mathcal{I}^+(\mathcal{D}_n)}$ signifie la fermeture de $\mathcal{I}^+(\mathcal{D}_n)$.

Si de plus f est $1/\max(|S_{min}|,|S_{max}|)$-lipschitzienne, alors $\bigcap_{n\in\mathbb{N}} \overline{\mathcal{I}^+(\mathcal{D}_n)}$ est l'épigraphe de f.

Démonstration. Par traçabilité, les signaux et collisions restants de $\mathcal{D}$ après considération des n premières transitions sont dans l'influence de $\mathcal{D}_n$, c'est-à-dire :

$$\forall n \in \mathbb{N},\ \mathcal{D}^{-1}(\{\oslash\}^c) \backslash \mathcal{D}_n^{-1}(\{\oslash\}^c) \subseteq \mathcal{I}^+(\mathcal{D}_n)$$

L'ensemble d'accumulation de $\mathcal{D}$ restant inchangé en enlevant un nombre fini de collisions et de signaux, il est bien inclus dans $\mathcal{I}^+(\mathcal{D}_n)$, et ce pour tout entier n. A fortiori, l'épigraphe de f est inclus dans $\bigcap_{n\in\mathbb{N}} \overline{\mathcal{I}^+(\mathcal{D}_n)}$.

Supposons maintenant que f soit $1/\max(|S_{min}|,|S_{max}|)$-lipschitzienne, et soient (x,t) les coordonnées d'un point de $\bigcap_{n\in\mathbb{N}} \overline{\mathcal{I}^+(\mathcal{D}_n)}$ telles que $x \in [-1\,;1]$. On note $k = 1/\max(|S_{min}|,|S_{max}|)$.

Pour tout entier n, il existe une collision entamée au point (x_n,t_n) telle que $(x,t) \in \mathcal{I}^+(\mathcal{D}_n)$. La suite $(x_n,t_n)_{n\in\mathbb{N}}$ est à valeurs dans le compact $J^-(x,t)$, elle y possède donc une valeur d'adhérence (x',t'). Cette valeur d'adhérence est encore dans le compact $J^-(x,t)$, ce qui implique $t' \leq t - k|x'-x|$.

La valeur d'adhérence (x',t') est une accumulation de $\mathcal{D}$, telle que $x' \in [-1\,;1]$, on a donc, par accumulation hâtive, $f(x') \leq t'$.

Par caractère k-lipschitzien de f, on a $f(x) \leq f(x') + k|x'-x|$

On a donc

$$\begin{aligned} f(x) &\leq f(x') + k|x'-x| \\ &\leq t' + k|x'-x| \\ &\leq t \end{aligned}$$

En d'autres termes, (x,t) appartient à l'épigraphe de f. □

Theorème 31. *Soit $\mathcal{A}$ une machine à signaux rationnelle et soient S_{min} et S_{max} les vitesses respectivement minimale et maximale de méta-signaux. On suppose que l'ensemble des vitesses est non trivial, c'est-à-dire $S_{min} \neq S_{max}$. Soit $\mathcal{D}$ un diagramme infini implicite de $\mathcal{A}$, s'accumulant hâtivement sur une fonction f.*

S'il existe une suite $(\tau_n)_{n\in\mathbb{N}}$ de transitions, calculable – au sens où $n \mapsto \tau_n$ est calculable – représentant $\mathcal{D}$ et que f est $1/\max(|S_{min}|,|S_{max}|)$-lipschitzienne, alors il existe un nombre calculable y de $[0\,;1[$ tel que f est l'accumulation de l'arbre $T(y)$, où T est la fonction définie au chapitre 4 section 4.3.2.

Ce théorème est une généralisation très partielle du théorème 25. Intuitivement, la condition $1/\max(|S_{min}|,|S_{max}|)$-lipschitzienne implique que les accumulations ne sont pas utilisées dans le calcul de la fonction.

Démonstration. D'après le théorème 30, l'épigraphe de f est $\bigcap_{n\in\mathbb{N}} \overline{\mathcal{I}^+(\mathcal{D}((\tau_m)_{m\leq n}))}$.

L'ensemble $\overline{\mathcal{I}^+(\mathcal{D}((\tau_m)_{m\leq n}))}$ est l'épigraphe d'une fonction continue (affine par morceaux rationnelle) f_n. Puisque l'épigraphe de f est $\bigcap_{n\in\mathbb{N}} \overline{\mathcal{I}^+(\mathcal{D}((\tau_m)_{m\leq n}))}$, la suite de fonctions $(f_n)_{n\in\mathbb{N}}$ est croissante de limite simple f.

Il existe donc une fonction en escalier rationnelle dyadique g_n à distance uniforme de f_n inférieure à $\frac{1}{n}$ (lemme 18), de sorte que $\lim g_n = \lim f_n = f$ (limite simple). Puisque $\bigcap_{n\in\mathbb{N}} \overline{\mathcal{I}^+(\mathcal{D}((\tau_m)_{m\leq n}))}$ est calculable et l'information pour définir f_n et g_n finie, on peut prendre $n \mapsto g_n$ calculable, par exemple en

énumérant les fonctions rationnelles dyadiques et en s'arrêtant quand inévitablement on en trouve une suffisamment proche (la distance uniforme entre deux fonctions affines par morceaux rationnelles étant encore calculable). Quitte à remplacer g_n par $\max_{m\leq n} g_m$, on peut supposer que la suite $(g_n)_{n\in\mathbb{N}}$ est croissante.

Grâce au lemme 17, il existe une suite $(G_n)_{n\in\mathbb{N}}$ d'arbres unaires-binaires finis, croissante pour la relation préfixe, telle que :

$$\forall n \in \mathbb{N}, T(G_n(0)) \text{ s'accumule sur } g_n$$

En outre, on peut imposer que les (G_n) soient 2-complets et de 2-hauteur croissante. Par ailleurs, toutes les opérations effectuées dans la preuve du lemme 16 sont calculables, donc la suite $(G_n)_{n\in\mathbb{N}}$ est encore calculable, et donc traçable par une machine à signaux.

En reprenant le schéma de la preuve du théorème 19 et en posant $G = \bigcup_{n\in\mathbb{N}} G_n$, on a bien que l'arbre G s'accumule sur f. □

5.3 Accumulation Tardive de Diagramme Machine à Signaux

On a établi au chapitre 4 que les accumulations tardives étaient des fonctions continues supérieurement.

On donne ici un contre-exemple de fonction continue supérieurement qui n'est accumulation tardive d'aucune machine à signaux.

Remarque 35. La fonction semi-continue supérieurement $f : [-1\,;1] \to \mathbb{R}$ définie par :

$$\begin{aligned} 1/n &\mapsto 1 + 1/n & n &\in \mathbb{N} \\ x &\mapsto 1 & &\text{sinon} \end{aligned}$$

n'est accumulation d'aucun diagramme de machine à signaux.

Démonstration. Par l'absurde, supposons qu'il existe un diagramme $\mathcal{D}$ d'une machine à signaux $\mathcal{A}$ s'accumulant tardivement sur f. Soient S_{min} et S_{max} les vitesses minimale et maximale de méta-signaux de $\mathcal{A}$.

Soit n un entier supérieur à $2 \times \max(|S_{min}|, |S_{max}|, 2)$. Le point $(1/n, 1 + 1/n)$, est un point d'accumulation du diagramme, donc il existe une collision à la position (x', t') à distance au plus $1/(2n)$ de $(1/n, 1 + 1/n)$.

Le cône d'influences reçues $J^-(x', t')$ est coupé par la ligne d'équation $t = 1$ qui a pour valeur ✺ par le diagramme $\mathcal{D}$. La collision de coordonnées (x', t') ne peut donc pas être retracée à la configuration initiale, ce qui est absurde. □

Ce contre-exemple est toutefois à modérer : il dépend fortement de la définition choisie pour les diagrammes de machine à signaux, en particulier du choix d'attribuer une valeur aux accumulations. Une démonstration de la conjecture 6 (ou d'un résultat similaire ou contradictoire) permettrait d'apprécier la validité et la canonicité des définitions choisies ou bien de les raffiner, auquel cas la preuve de remarque précédente pourrait ne plus être valide.

Conclusion

Résultats

Nous nous sommes posé la question du tracé de graphe de fonctions par ensemble d'accumulations de machine à signaux. L'enjeu est triple : tout d'abord, des diagrammes semblables aux diagrammes de machines à signaux sont utilisés dans la conception d'un certain nombres de machines ou système pour d'autres modèles de calculs, notamment pour la conception d'automates cellulaires. L'étude de tracé de fonctions à l'aide de machine à signaux peut fournir des idées et des outils plus généraux pour d'autres modèles de calculs, y compris discrets. Le second enjeu est celui de géométrie du calcul. Calculer sur le plan dans le plan est une chose non standard, en particulier l'espace de calcul, et l'espace de sortie sont de grande cardinalité, et permette de définir des problèmes uniques, tel que le tracé de fonction. Le troisième enjeu est celui de la notion de puissance de calcul des machines à signaux.

Nous nous sommes d'abord intéressés aux fonctions affines. Nous avons construit une machine à signaux dont les diagramme s'accumulent, selon la configuration initiale, sur n'importe quelle fonction affine [30].

Nous avons ensuite développé un modèle de calcul, aisément simulé par des diagrammes constructifs de machines à signaux. Ce modèle est équivalent à la variante linéaire du modèle de calcul Blum-Shub-Smale, et peut simuler les diagrammes constructifs d'une machine à signaux.

En réutilisant l'idée d'arbre unaire-binaire utilisée lors du tracé de fonctions affines, on construit une machine à signaux dont les diagrammes peuvent s'accumuler sur n'importe quelle fonction continue. On obtient ce résultat en faisant correspondre aux arbres unaire-binaires finis des fonctions en escalier, et on montre qu'un arbre unaire-binaire infini sans feuille s'accumule sur la limite simple de la suite croissante de fonctions en escalier correspondant à ses sous-arbres préfixes. Cela permet d'élargir la notion de tracé de fonction par accumulation de sorte à atteindre toutes les fonctions semi-continues inférieurement, et uniquement celles-ci.

Nous nous sommes ensuite intéressés aux machines à signaux rationnelles. Au lieu de pouvoir coder n'importe quel arbre unaire binaire dans la configuration initiale, on ne peut plus tracer que ceux qui sont calculables. On peut alors atteindre n'importe quelle fonction qui est la limite simple d'une suite croissante de fonctions en escalier rationnelles calculables, au sens ou la subdivision et les valeurs prises par chacune de ces fonctions sont rationnelles, et leur suite est calculable.

Réciproquement, nous avons montré que sous certaines conditions, les fonctions tracées par des diagrammes de machines à signaux rationnelles sont des limites croissantes de fonctions en escalier rationnelles calculables. Ces conditions sont : ces diagrammes peuvent être décrits par une liste calculable d'éléments – signaux ou collisions – et la fonction cible doit être lipschitzienne de constante de Lipschitz inférieure à l'inverse de toute vitesse de méta-signal. Intuitivement, cette condition signifie que les accumulations ne sont pas utilisées dans le calcul.

Pistes

Cette thèse a suscité des questions qui sont encore ouvertes et que nous présentons ici.

Lien avec les Automates Cellulaires

Puisque les machines à signaux tirent leur origine des automates cellulaires, on peut se demander si il est possible de discrétiser notre travail. Par exemple, est-il possible de créer un automate cellulaire ou une famille d'automates cellulaires approximant le graphe d'une fonction continue arbitraire à l'aide d'arbres unaire-binaires ?

Existence et Unicité de Diagramme

Étant donnée une machine à signaux et une configuration initiale, il existe un unique diagramme constructif correspondant.Nous pouvons nous demander s'il en est de même pour le diagrammes définis de manière implicite comme conjecturé au premier chapitre 6. La réponse dépend fortement de comment

sont traitées les accumulations. Nous n'avons pas, selon la définitions de cette thèse, trouvé de contre-exemple de couple machine-configuration qui n'aurait pas de diagramme associé ou qui en aurait plusieurs.

Construire explicitement un tel diagramme est a priori difficile, puisque les diagrammes de machines à signaux ne sont prouvés calculables que dans le cas constructif. Par exemple, l'ensemble des collisions (règles et positions) d'un diagramme n'est a priori pas décidable, ni même énumérable.

Une idée de preuve est de définir des diagrammes partiels où les collisions ne sont pas nécessairement origine de méta-signaux, puis de définir une relation d'ordre telle qu'un élément maximal soit un diagramme, mais aussi tel que l'ensemble des diagrammes partiels soit inductif.

L'inclusion, en voyant les diagrammes partiels comme des ensembles de signaux et de collisions, est bien une relation d'ordre mais les éléments maximaux ne sont pas des diagrammes, intuitivement parce qu'il existe beaucoup de diagramme partiels inconsistants avec ne serait-ce qu'un diagramme défini de manière constructive.

Un autre idée consiste à construire une suite transfinie de diagrammes partiels tel qu'un élément indicé par un ordinal successeur soit le diagramme obtenu en appliquant la dynamique constructive de machine à signaux au diagramme précédent, et un élément indicé par un ordinal limite soit l'union croissante des éléments précédents. Le principal écueil est qu'il n'y a aucune garanti de pouvoir appliquer la dynamique constructive –ou une modification de celle-ci – au delà de la première accumulation. Mais même si cette idée ne permet pas d'exhiber un diagramme pour tous les couples machine-configuration initiale, elle peut potentiellement en traiter davantage que les diagrammes constructifs seuls.

Les Diagrammes Suffisamment Lipschitziens sont Calculables

Une idée pour montrer l'existence d'un diagramme de machine à signaux et de modifier la dynamique constructive. Celle-ci peut être reformulée comme le tracé des collisions – et de leur signaux afférents – dans l'ordre chronologique. Une manière de formuler pourquoi cette méthode fonctionne, c'est-à-dire pourquoi les collisions tracées sont toujours consistantes avec le diagramme partiel tracé jusqu'à présent, est de raisonner sur les cônes d'influence : la prochaine collision dans l'ordre chronologique n'est dans le cône d'influence d'aucune collision entamée autre que celle dont ses signaux afférents sont issues. En précisant cette condition sur les collisions potentielles, on peut considérer une dynamique où l'on trace par exemple la collision de profondeur minimum ou encore toutes les collisions potentiels « assurées » à la fois.

Cela ne suffit pas a montrer l'existence d'un diagramme dans le cas général, mais devrait suffire suffit à reconstruire au moins un diagramme constructif, parfois davantage, et ce en utilisant un procédé calculable. En particulier, il se peut que le théorème 31 puisse être prouvé en laissant tombé la condition calculable, qui découlerait de la condition $1/\max(|S_{min}|, |S_{max}|)$-lipschitzienne.

Puissance de Calcul

Nous nous sommes aussi posé la question de la puissance de calcul des machines à signaux. Par exemple le problème d'apparition d'un méta-signal en un nombre fini de collisions est Σ_1^0-complet [12]. Le même problème d'apparition d'un méta-signal, sans la condition du nombre fini de collisions, est a priori plus haut dans la hiérarchie arithmétique. Nous pouvons nous poser la question d'où exactement.

La plupart des résultats de calculabilité relatifs aux machines à signaux vont jusqu'à la première accumulation. Pour les tracer de fonctions, il est nécessaire d'aller au delà.

Les machines à signaux sont capables d'effectuer une infinité (dénombrable) de calculs d'une machine de Turing en un temps et espace fini. On pourrait donc altérer légèrement le résultat précédemment cité et imposer de plus que le méta-signal apparaissent en temps fini borné, voir imposer l'apparition d'un signal ou d'une collision en une position particulière, l'intérêt étant de pouvoir réutiliser, à l'intérieur d'un diagramme de machine à signaux, le résultat du problème d'apparition de signal.

On peut de plus observer que les constructions de diagrammes de machines à signaux prenant un temps et un espace fini peuvent être répétées une infinité dénombrable de fois en temps et espace fini dans un diagramme de machine à signaux. Cette observation devrait permettre de montrer que le problème d'apparition de signal est $\Sigma^0_{<\omega}$-difficile, c'est à dire Σ^0_n-difficile et Π^0_n-difficile pour tout entier n.

Nous pouvons nous poser la question de si ce problème est dans $\Sigma^0_{<\omega}$ ou encore plus haut.

Par rapport à cette thèse, la question naturelle suivante est : comment cela affecte-t-il les fonctions traçables par accumulation de machine à signaux ? On peut commencer par se poser la question des positions de points d'accumulation. On sait déjà que l'ordonnée (le temps) des points d'accumulations isolés est semi-calculable [20]. La question « est-ce qu'un nombre semi-calculable est plus petit qu'une constante donnée ? » est dans Π^0_1. Ce type de problème est équivalent à « est-ce qu'un nombre entier moins un nombre semi-calculable est plus grand qu'une constante donnée ? » ; en utilisant des techniques similaire à [20], il devrait donc être possible de tracer une accumulation d'ordonnée un nombre entier moins un nombre semi-calculable. En particulier on peut s'accumuler sur des points qu'on ne pourrait atteindre avec des accumulations isolées.

Cette accumulation n'étant pas isolé, cela soulève la question de distinguer les accumulations en sortie, qui servent à dessiner ce que l'on s'est donner pour but de tracer, des accumulations auxiliaires, nécessaires au calcul. Le tracé de courbe par accumulation tardive peut constituer une approche possible.

Par ailleurs, la puissance de calcul des machines à signaux peut possiblement être mise en rapport avec la difficulté de montrer, de manière générale, l'existence d'un diagramme.

Complexité

De manière similaire aux ordinateurs analogiques[10], la complexité en temps n'a guère de sens pour les machines à signaux. On peut se poser la question de définir une notion de complexité particulière aux machines à signaux. Par exemple, la complexité en temps pourrait s'exprimer en nombre de signaux ou de collisions, celle en espace en nombre de signaux concourants. Sur des problèmes algorithmiques classiques, ces notions devraient pouvoir se rattacher à des notions usuelles, présumément de celles des machines de Turing.

Si les machines à signaux s'avèrent effectivement capables de supers calculs – Sec. 5.3 – et si l'approche de construction de diagramme par suite transfinie de diagramme partielle – Sec. 5.3 – s'avère fructueuse, le plus petit ordinal nécessaire pour tracer les collisions une par une de manière chronologique ou topologique serait un bon candidat pour définir la complexité d'un diagramme de machine à signaux.

Bibliographie

[1] C. Corge, *Machines de Turing et automates cellulaires : du trait gravé au très animé*. Ellipses, 2008.

[2] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. s2-42, pp. 230–265, 01 1937.

[3] A. Church, "A set of postulates for the foundation of logic," *Annals of Mathematics*, vol. 33, no. 2, pp. 346–366, 1932.

[4] A. M. Turing, "Computability and λ-definability," *Journal of Symbolic Logic*, vol. 2, no. 4, p. 153–163, 1937.

[5] P. Bernays, "Alonzo church. an unsolvable problem of elementary number theory. american journal of mathematics, vol. 58 (1936), pp. 345–363.," *Journal of Symbolic Logic*, vol. 1, pp. 73–74, jun 1936.

[6] S. Aaronson, "Introduction to quantum information science, lecture notes," Fall 2018.

[7] S. Arora and B. Barak, *Computational complexity : a modern approach*. Cambridge University Press, 2009.

[8] M. B. Pour-El and N. Zhong, "The wave equation with computable initial data whose unique solution is nowhere computable," *Mathematical Logic Quarterly*, vol. 43, no. 4, pp. 499–509, 1997.

[9] D. Graça, M. Campagnolo, and J. Buescu, "Computability with polynomial differential equations," *Advances in Applied Mathematics*, vol. 40, pp. 330–349, 03 2008.

[10] O. Bournez, D. S. Graça, and A. Pouly, "Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length : The general purpose analog computer and computable analysis are two efficiently equivalent models of computations," 2016.

[11] H. Christos, "Papadimitriou : Computational complexity," *Addison-Wesley*, vol. 2, no. 3, p. 4, 1994.

[12] J. Durand-Lose, *Calculer géométriquement sur le plan – machines à signaux*. Habilitation à Diriger des Recherches, École Doctorale STIC, Université de Nice-Sophia Antipolis, 2003.

[13] J. Durand-Lose, "Abstract geometrical computation: Turing computing ability and undecidability," in *New Computational Paradigms, 1st Conf. Computability in Europe (CiE 2005)* (B. S. Cooper, B. Löwe, and L. Torenvliet, eds.), no. 3526 in LNCS, pp. 106–116, Springer, 2005.

[14] L. Blum, M. Shub, and S. Smale, "On a theory of computation over the real numbers ; NP completeness, recursive functions and universal machines (extended abstract)," in *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pp. 387–397, IEEE Computer Society, 1988.

[15] L. Blum, M. Shub, and S. Smale, "On a theory of computation and complexity over the real numbers : *np*- completeness, recursive functions and universal machines," *Bull. Amer. Math. Soc. (N.S.)*, vol. 21, pp. 1–46, 07 1989.

[16] J. Durand-Lose, "Abstract geometrical computation and the linear Blum, Shub and Smale model," in *Computation and Logic in the Real World, 3rd Conf. Computability in Europe (CiE 2007)* (B. S. Cooper, B. Löwe, and A. Sorbi, eds.), no. 4497 in LNCS, pp. 238–247, Springer, 2007.

[17] J. Durand-Lose, "Abstract geometrical computation 6: a reversible, conservative and rational based model for black hole computation," *Int J Unconventional Computing*, vol. 8, no. 1, pp. 33–46, 2012.

[18] J. Durand-Lose, "Abstract geometrical computation 1: embedding black hole computations with rational numbers," *Fund Inf*, vol. 74, no. 4, pp. 491–510, 2006.

[19] J. Durand-Lose, "Abstract geometrical computation 3: black holes for classical and analog computing," *Nat Comput*, vol. 8, no. 3, pp. 455–472, 2009.

[20] J. Durand-Lose, "Abstract geometrical computation 7: Geometrical accumulations and computably enumerable real numbers," *Nat Comput*, vol. 11, no. 4, pp. 609–622, 2012. Special issue on Unconv. Comp. 2011.

[21] J. Durand-Lose, "Abstract geometrical computation 5: embedding computable analysis," *Nat Comput*, vol. 10, no. 4, pp. 1261–1273, 2011. Special issue on Unconv. Comp. 2009.

[22] P. Arrighi, S. Martiel, and S. Perdrix, "Reversible causal graph dynamics," in *Reversible Computation* (S. Devitt and I. Lanese, eds.), (Cham), pp. 73–88, Springer International Publishing, 2016.

[23] W. Hordijk, C. R. Shalizi, and J. P. Crutchfield, "Upper bound on the products of particle interactions in cellular automata," *Physica D : Nonlinear Phenomena*, vol. 154, p. 240–258, Jun 2001.

[24] A. Waksman, "An optimum solution to the firing squad synchronization problem," *Inf. Control.*, vol. 9, no. 1, pp. 66–78, 1966.

[25] J. Mazoyer, "On optimal solutions to the firing squad synchronization problem," *Theor. Comput. Sci.*, vol. 168, no. 2, pp. 367–404, 1996.

[26] H. Umeo, M. Hisaoka, and T. Sogabe, "A survey on optimum-time firing squad synchronization algorithms for one-dimensional cellular automata," *Int. J. Unconv. Comput.*, vol. 1, no. 4, pp. 403–426, 2005.

[27] M. Delorme, J. Mazoyer, and L. Tougne, "Discrete parabolas and circles on 2d cellular automata," *Theor. Comput. Sci.*, vol. 218, no. 2, pp. 347–417, 1999.

[28] J. Durand-Lose, "Irrationality is needed to compute with signal machines with only three speeds," in *CiE 2013, The Nature of Computation* (P. Bonizzoni, V. Brattka, and B. Löwe, eds.), no. 7921 in LNCS, pp. 108–119, Springer, 2013. Invited talk for special session *Computation in nature*.

[29] T. Besson and J. Durand-Lose, "Exact discretization of 3-speed rational signal machines into cellular automata," in *Cellular Automata and Discrete Complex Systems* (M. Cook and T. Neary, eds.), (Cham), pp. 63–76, Springer International Publishing, 2016.

[30] J. Durand-Lose and A. Emmanuel, "Abstract geometrical computation 11 : Slanted firing squad synchronisation on signal machines," *Theoret Comp Sci*, vol. 894, pp. 103–120, 2021. arXiv 2106.11176.

[31] J. Durand-Lose, "The coordinates of isolated accumulations [includes] computable real numbers," in *Programs, Proofs, Processes, 6th Int. Conf. Computability in Europe (CiE 2010) (abstracts and extended abstracts of unpublished papers)* (F. Ferreira, H. Guerra, E. Mayordomo, and J. Rasga, eds.), pp. 158–167, CMATI, U. Azores, 2010. Contains a flaw, corrected in [36].

[32] F. Becker, T. Besson, J. Durand-Lose, A. Emmanuel, M.-H. Foroughmand-Araabi, S. Goliaei, and S. Heydarshahi, "Abstract geometrical computation 10 : An intrinsically universal family of signal machines," *ACM Trans. Comput. Theory*, vol. 13, no. 1, pp. 1–31, 2021. arXiv 1804.09018.

[33] J. Durand-Lose, "A reversible and conservative model based on rational signal machines for black hole computation," in *HyperNet 10: The Unconventional Computation 2010 (UC 202010) Hypercomputation Workshop* (M. Stannett, ed.), pp. 48–59, 2010.

[34] J. C. Lagarias, "The takagi function and its properties," 2011.

[35] R. Engelking, *General Topology*. Sigma series in pure mathematics, Heldermann, 1989.

[36] J. Durand-Lose, "Geometrical accumulations and computably enumerable real numbers (extended abstract)," in *Int. Conf. Unconventional Computation 2011 (UC 2011)* (C. S. Calude, J. Kari, I. Petre, and G. Rozenberg, eds.), no. 6714 in LNCS, pp. 101–112, Springer, 2011.

Annexe A

Langage MTRL, notice d'utilisation

Une introduction au langage AGC ainsi que les ressources pour l'utiliser sont disponibles :
`https://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose/Recherche/AGC/intro_AGC.html`

Le code et la documentation du langage MTRL est disponible :
`https://github.com/42xel/agc_accumulation_curves`

Le langage est dénoté par l'extension `.mrtl`. Il permet 4 choses :

— définir une MTRL rationnelle ;

— spécifier un état initial de l'espace de calcul (ruban, position de la tête et valeur de l'accumulateur) ;

— interpréter une machine ;

— exporter une machine, une configuration initiale, ou un calcul en une machine à signaux, une configuration initiale ou un calcul le simulant, au format AGC.

A.1 Commandes Shell

Pour exécuter un fichier `my_lbss_file.lbss`, il est nécessaire d'avoir python. Cela se fait via une commande :

`python3 <lbss.py_path>/lbss.py my_lbss_file.lbss [<agc_files_path>]`

où `<lbss.py_path>` est l'emplacement du fichier `lbss.py`, qui est le point d'entrée de l'interpréteur python, spécifié relativement à l'emplacement de la commande. Le chemin optionnel `<agc_files_path>` désigne l'emplacement où mettre les fichier AGC en sortie.

A.2 Principes généraux

Des *variables* sont utilisées pour stocker et référencer des configurations, des machines ou des exécutions de machines. Un nom de variable est une suite de caractères alphanumériques, hors mots-clefs.

Le langage fonctionne par *mots-clef*, des mots réservés. Certains mots-clefs permettent, entre autres choses, de délimiter des définitions de configurations et de machines.

D'autres mots-clefs agissent comme des fonctions ou des procédures : ils agissent en fonction de leur arguments, listés à la suite séparés par des espaces. Il n'existe pas de valeur fonction ou de manière de définir de nouvelles fonctions dans le langage. Par soucis de concision et de clarté, nous appellerons de tel mots clés des *commandes*.

Comme annoncé au chapitre 1, les nombres maniés sont des rationnels. On peut les rentrer par exemple : numérateur, barre de fraction, dénominateur, avec possibilité d'omettre la barre de fraction et le dénominateur si celui-ci vaut 1. On peut rentrer les valeurs décimales au format : partie entière point partie décimale.

Par exemple, `22/7`, `11/4` et `5` sont des manières valide d'écrire un nombre.

A.3 Définition de la Configuration Initiale

La *configuration initiale*, c'est-à-dire l'état initial de l'espace de calcul, est spécifiée comme suit : le mot-clef `create_lbss_configuration`, suivi de 2 chevrons '<' puis du nom, en caractères alphanumériques, de la configuration dénote le début de la définition ; la fin de cette dernière est marquée par une ligne contenant uniquement le nom de la configuration.

La définition est faite en python, dans un environnement où `x` est le ruban, une liste bi-infinie dont les valeurs par défaut sont 0, où `p` est la position de la tête de la machine de calcul, et où `a` est la valeur de l'accumulateur, par défaut à 0.

Par exemple :

```
create_lbss_configuration << example_configuration
x[-1] = -4/3
x[0] = 3/5
x[2] = -7
p = -1
a = 15
example_configuration
```

A.4 Encodage d'une machine MTRL

A.4.1 Définition

La création d'une machine MTRL se fait par l'usage du mot clé `function_create_lbss_machine` suivi de 2 chevrons '<<' puis du nom de la machine a créer. Ce nom, répété seul sur une ligne, dénote la fin de la définition de la machine

Par exemple, le code suivant :

```
create_lbss_machine << simplistic_example
end;
simplistic_example
```

crée une machine nommée `simplistic_example`, dont la définition tient sur une ligne, expliquée dans la sous-section suivante.

A.4.2 Définition des nœuds

À chaque nœuds correspond une ligne de code. Un point virgule '`;`' marque la fin de la définition d'un nœud (et rien d'autre).

Un nœud terminal est simplement écrit :

```
end;
```

Les nœuds de mouvement gauches et droits sont notés avec des chevrons ; la machine dont la définition est la suivante :

```
>;
<;
end;
```

Se déplace d'un pas à droite, d'un pas à gauche, puis s'arrête.

À cela s'ajoute le sucre syntaxique suivant : plusieurs nœuds de mouvement dans la même direction peuvent s'écrire comme une suite de chevrons, de sorte que :

```
>>;
```

est équivalent à :

```
>;
>;
```

Les nœuds de calcul sont notés par leur opération de calcul élémentaire, au format :

`<recipient> <opération> (<constante> * <opérande> | <constante> | <opérande>)`

où :

- `<récipient>` est `a` (accumulateur) ou `h` (cellule courante) ;
- `<opération>` est `=` (assignement), `+=` (ajout) ou `-=` (retranchement) ;
- `<constante>` est un nombre (suite de caractères numériques, éventuellement précédée d'un signe moins) ;
- `<opérande>` est `a` (accumulateur), `h` (cellule courante) ou `u` (unité).

Par exemple, le code :

```
a = 0;
a += 1 * h;
a += 2 * a;
a += 1/2 * u;
end;
```

définit une machine qui met $3h+1/2$ dans l'accumulateur, via les étapes suivantes : effacer l'accumulateur, le rendre égal à la valeur de la cellule courante, l'ajouter deux fois à lui-même, puis ajouter une demie unité.

Le code précédent peut encore être écrit :

```
1 a = 0;
2 a += h;
3 a += 2 * a;
4 a += 1/2;
5 end;
```

ou encore :

```
1 a = h;
2 a += 2 * a;
3 a += 1/2;
4 end;
```

Les labels sont utilisés pour contrôler le flot du programme ; en particulier ils sont utilisés par les nœuds de branchement dans leur définition. Abstraitement, un label est un pointeur vers un nœud. Un label est défini en écrivant un séquence de caractères alphanumériques suivie de deux points ':', et pointe vers le prochain nœud défini. Le code qui suit définit un label, puis un nœud de mouvement, vers lequel pointe le label.

```
LabelX:
<;
```

Les doubles labels ne font rien ; tous les labels d'une suite consécutive de labels pointeront vers le même prochain nœud défini. Ce n'est pas très utile en soit, mais peut faciliter la méta-programmation.

Les nœuds de branchement sont notés "Branch", suivi par trois noms de label, correspondant aux trois nœuds γ_η^-, $\gamma_\eta^=$ or γ_η^+. Par exemple, le code :

```
1 a = h;
2 Branch LabelNegative, LabelZero, LabelPositive;
3 LabelNegative:
4 end;
5 LabelZero:
6 end;
7 LabelPositive:
8 end;
```

définit une machine s'arrêtant à différent nœuds selon le signe de la valeur de la cellule courante.

A.4.3 Flot/transition/arrêt

Par défaut, le nœud suivant d'un nœud de calcul ou de mouvement est le prochain nœud défini. Les "goto" permettent d'altérer ce comportement ; Ils sont placés après le point virgule. Par exemple, le code suivant :

```
1 h =0; goto labelEnd
2 h = 1;
3 labelEnd:
4 end;
```

définit une machine qui met la cellule courante à zéro et s'arrête. Le nœud '`h =0;`' de la ligne 1 est lié à (a pour successeur) le nœud terminal de la ligne 4, ignorant ainsi le nœud '`h = 1;`' de la ligne 2 (qui n'a pas de nœud entrant, et est donc du code mort).

A.5 Exécution d'une machine

L'exécution d'un machine se fait via le mot-clef `run` Ce mot-clef possède 6 arguments. Les trois premiers, obligatoires, sont : la machine, la configuration initiale, et le nom de l'exécution à produire, par exemple :

```
run machine_swap configuration_swap run_swap
```

Les trois derniers, optionnels et à fournir sont : le nombre maximal d'étapes, le nœud initial, et la valeur initiale de l'accumulateur, à donner dans l'ordre (si l'on veut spécifier le nœud initial, on doit spécifier le nombre d'étapes). Par exemple :

```
run machine_move configuration_move run_move -1 _#0 45
```

Le nombre d'étapes est le nombre maximal d'arêtes du graphe d'exécution traversées. La valeur par défaut du nombre maximale d'étape est -1, et signifie l'infini.

Le nœud initial peut être spécifié par un label, ou un *identifiant de nœud*, au format `labelName_#<number>`, qui signifie `<number>` nœuds après le label `labelName`, en comptant à partir de 0 ; sa valeur par défaut est `_#0`, qui est l'identifiant du premier nœud défini dans le code de la machine : pour l'identification des nœuds, tout se passe comme si le code commence par le mot vide comme label.

La valeur de l'accumulateur remplace celle définie dans la configuration initiale.

A.6 Export au format agc

Les mot-clefs présentés dans cette sous-section `toAGC_machine`, `toAGC_config`et `toAGC_run`servent à générer les fichier AGC des objets correspondant Typiquement, les machines et les configurations respectivement générées par `toAGC_machine`et `toAGC_config`vont être utilisés par le fichier de calcul généré par `toAGC_run`. Ce dernier est à exécuter via l'interpréteur AGC.

Pour transpiler un une machine dans son équivalent agc, utiliser :

```
toAGC_machine machine_swap machine_swap.agc
```

Pour transpiler une configuration :

```
toAGC_config configuration_swap configuration_swap.agc 3
```

L'argument numérique, optionnel et nul par défaut, est la valeur initiale de l'accumulateur.

Pour transpiler un calcul :

```
toAGC_run machine_swap.agc configuration_swap.agc test_machine_swap.agc
↪  test_machine_swap.pdf _#0 1/6 1000
```

- La machine et la configuration peuvent être soit des noms de fichiers agc, ou bien directement des nom de machine et de configuration défini dans MTRL. En l'absence de nom de fichier agc, les machines et configurations sont quand même transpilées et écrites dans un fichier, ayant le même nom pour tous.
- Le nom du pdf est optionnel, de valeur par défaut `"lbssagc_out.pdf"`.
- L'argument optionnel suivant est le nœud initial, de valeur par défaut `_#0`.
- L'argument suivant est l'échelle en lien avec l'épaisseur des tracés de signaux (défaut : 1/3).
- Le dernier argument est le nombre maximum de pas de calcul de la machine agc ; c'est le nombre de collision de signaux et n'est que vaguement (linéairement ? d'un facteur 10 au doigt mouillé) en lien avec le nombre de pas de simulation direct (python) de machine MTRL.

Le résultat est un fichier qui peut être exécuter par le logiciel agc pour générer un pdf. Dans l'exemple précédent, la commande idoine serait :

```
java -jar agc_2_0.jar <agc_path>/test_machine_swap.agc
```

Les fichiers agc sont à exécuter depuis l'emplacement où ils ont été générés. Ils dépendent d'une librairie agc situé à un endroit précis relativement au chemin de l'interpréteur/transpilateur python.

Annexe B

Langage AGC, Portées Dynamiques et Programmation Orienté Objet

La page `https://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose/Recherche/AGC/Sommaire.pdf` documente de manière satisfaisante le langage agc. Celui-ci y est décrit comme impératif non typé (typage dynamique faible, sans possibilité de création de type). La programmation orientée objet y est considérée comme une évolution possible du langage.

La présente thèse s'appuie sur une certaine quantité de code AGC. J'ai donc employé un certain nombre de techniques pour organiser ce code. Des cas d'usage propres à la thèse sont détaillés dans l'annexe C. Dans cette section, j'explique des techniques pour elles-mêmes, et les poussent plus loin qu'utile pour cette thèse, et entends montrer que le langage AGC répond d'ores et déjà aux critères habituels de programmation orientée objet.

B.1 Contextes/Blocs

B.1.1 Localisation de Variables

Le langage AGC fonctionne par *contextes* délimités par des accolades `{}`. Ces contextes permettent la localisation des variables. Par exemple :

```
{
        name .= "Harley";
        {
                println(name);
                //$ Harley
                println(.name);
                //$ undef
                .name .= "Johnny";
                println(name);
                //$ Johnny
                println(..name);
                //$ Harley
        };
        println(name);
        //$ Harley
}

println(name);
//$ undef
```

Si une variable est absente d'un contexte, on regarde dans les contextes parents successifs. Il est possible de commencer cette recherche dans le contexte parent via le préfixe double points : `..momsVar`. Si on ne trouve rien, une rvalue sera évaluée `undef` tandis qu'une lvalue donnera lieu à la création d'une variable dans le contexte dans lequel on a commencé la recherche. Pour forcer la création ou l'usage d'une variable locale, sans remonter dans les contextes parents, on peut préfixer la variable par un simple point : `.myVar`.

B.1.2 Blocs de Codes comme Objets

Ces blocs de codes sont en fait des objets, des dictionnaires dont les variables locales sont les entrées. On peut par exemple écrire :

```
block .= {
        a := "Hello";
        b := "World";
        sep := " ";
};
println(block.a _ bock.sep _ bock.b);
//$ Hello World
```

Ces contextes peuvent être ré-ouverts en étant suffixés par `::` ou par `.`, en plus et par contraste avec une syntaxe `nom.champ` :

```
block.a := "Bonjour"
block::sep := "        ";
block.{
        b := "Monde";
}
block::{
        println(a _ sep _ b);
};
//$ Bonjour        Monde
```

Contrairement à des langages comme C++, Rust ou JavaScript, où les accolades peuvent noter aussi bien des blocs de code que des objets littéraux ou des structures, ici, les accolades dénotes *simultanément* des blocs de code et des objets.

Par exemple, l'opérateur de contrôle de flot `if` renvoie une valeur (combiné avec `else`, c'est l'opérateur ternaire), et cette valeur est un contexte si (et seulement si) l'expression suivant est un bloc délimiter par des accolades.

```
.name .= "Harley";
{
        .name .= "Johnny";

        v .= if (true)
        println(..name);
        //$ Harley
        println(v);
        //$ void

        o .= if (true) {
                println(..name);
                //$ Johnny
        };
        println(o);
        //$ {}
};
```

B.1.3 Fonctions

Les fonctions dans AGC sont des valeurs de première classe. Cela signifie qu'elles sont associées à des variables et qu'elles sont déclarées comme telles.

Dès lors qu'il y a des accolades, il y a un contexte. La réciproque n'est pas vrai, dans la mesure ou les fonctions introduisent leur contexte de manière implicite. Dans cet exemple :

```
f .= (a, b) -> a + b;
res := f(1, 2);
```

Tout se passe comme si on avait écrit :

```
f .= (a, b) -> a + b;
{
        .a := 1;
        .b := 2;
        ..res := a + b;
}
```

Si l'expression de retour est un contexte littéral, il est possible de ré-ouvrir ce contexte directement après l'appel de la fonction :

```
f .= (a, b) -> { .sum := a + b; };
res := f(1, 2) { println(sum); };
```

est équivalent à

```
f .= (a, b) -> { .sum := a + b; };
{
        .a := 1;
        .b := 2;
        ..res := { .sum := a + b; };
}
res::{
        println(sum);
}
```

Cela a des implications sur la portée des variables dans le corps de la fonction :

```
.a := 4;
f .= (a) -> println(..a);
g .= (a) -> { println(..a); };

f(5);
//$ 4
g(6)
//$ 6
.{println(..{});};
//$ { .a := 6; }
```

La portée des variables est statique. En d'autres termes, l'environnement contenant le contexte d'exécution d'une fonction est l'environnement de définition de la fonction. En particulier, en toute rigueur, les équivalences utilisant `..res:=` ne valent que lorsque l'environnement d'appel (auquel réfère `res`) coïncide avec celui de définition. Autre conséquence, les fonctions capturent leur environnement.

B.2 Portée Dynamique

Si la possibilité de déclarer une variable dans un contexte parent depuis un contexte enfant n'est pas commune, la gestion de la portée des variables demeure statique, ce qui le standard des langages de programmation modernes.

L'existence ou non d'une variable dans un contexte est connue au temps d'exécution, comme pour les variables global en lua ou en python. La résolution de nom en une variable du contexte ou de l'un de ses parents est aussi connue au temps d'exécution. Le langage AGC fait donc parti des langages illustrant le caractère approximatif des choix des qualificatifs statique et dynamique dans le contexte des portées.

Les noms de variable en revanche, eux, sont statiques, par contraste avec les environnements globaux de lua ou python, qui sont des dictionnaires entièrement manipulables à l'exécution, génération de clées comprise.

Dans cette section, nous voyons le préfixe de variable point d'interrogation (`?var`), qui permet de gérer des portées de manière dynamique.

B.2.1 Signification

A l'intérieur du corps d'une fonction, préfixer une variable d'un point d'interrogation signifie commencer la recherche dans le contexte appelant.

```
.x .= 10;

// Called by g()
f .= () -> ?x ;

// g() has its own variable
// named as x and calls f()
g .= () -> {
        .x .= 20;
        := f();
};

println(g());
//$ 20
```

Dans cet exemple, sans le point d'interrogation, la variable `x` dans le contexte d'exécution de `f` serait de portée statique, et en partant de son usage dans la définition de la fonction, on en déduit qu'elle réfère à la variable de niveau maximal qui est constante égale à 10.

Avec le point d'interrogation en revanche, sa portée est dynamique, et dépend de l'endroit où la fonction est appelée, ici réfère à la constante égale à 20 créée dans le contexte d'exécution de `g`.

Contrairement à Perl, qui permet aussi de manipuler à la fois des portées statiques et dynamiques, le type de portée est préciser à l'usage d'une variable plutôt qu'à sa déclaration. Cela est moins flexible, moins verbeux, et davantage prédictible.

B.3 Programmation Orientée Objet

Il n'y a pas dans le langage AGC de classes à proprement parler, pas de mot-clef `private`, pas de système de typage, et a fortiori pas de constructeur de classes, pas d'intégration des classes aux système de typage, et a priori pas de mécanisme d'héritage.

Cependant, la programmation orientée objet ainsi que ses vertus ne se limitent pas à l'existence et à l'usage de classes parmi les mots-clefs ou les systèmes optimisés du langage. Pour faire simple, nous suivons les quatre piliers de la programmation orientée objet, à savoir abstraction, encapsulation, héritage et polymorphisme, et montrons qu'ils sont réalisables dans le langage AGC en l'état.

B.3.1 Abstraction

Le pilier probablement le plus important, mais aussi le plus commun et moins exclusif de la programmation orientée objet. Dans la mesure où il y a des variables, des fonctions, et des structures, AGC présente un degré d'abstraction suffisant. L'absence de module ou d'ensemble de nom est notable, mais pas gênante ; en outre, les contextes peuvent servir de modules de fortune, combinant la syntaxe `context::variable` et le fait que les fonctions sont des valeurs comme les autres.

B.3.2 Encapsulation

Le manque de contrôle sur la visibilité des champs du langage, ainsi que son relativement haut degré d'introspection, notamment la facilité à remonter les piles, aussi bien d'appel que de définition, n'est pas prometteur quant à l'encapsulation.

Par exemple, il est toujours aisé de remonter au parent d'un contexte.

```
proxy .= (source) -> source::{
:= {};
};

t .= proxy ({
.a .= 4;
});

println(t.a);         //$ 4
println(t);         //$ {}

t::.a := 3;
println(t.a);         //$ 3
println(t);         //$ { .a := 3 ; }


f .= (() -> t.{
?source := ..{};
});
f ();
println(source);
//$ { .a := 4 ; }
```

L'absence d'encapsulation ne serait pas en soit rédhibitoire. On peut laisser l'utilisateur libre de ne pas utiliser ce qu'il ne veut pas utiliser, sans lui forcer l'usage d'un système de visibilité s'il n'en a pas besoin. C'est le choix de lua, et initialement de JavaScript par exemple, au moins dans leur fonctionnement par défaut.

Enfin, il est tout à fait possible de faire de l'encapsulation dans le langage AGC. Partons du principe que l'utilisateur a accès au contexte racine et à toute l'arborescence des contextes nommés qui en découle. Pour masquer des données, il est donc nécessaire de se munir de contextes anonymes, qui ne sont pas non plus ancêtres de contextes nommés. Cela s'avère aussi satisfaisant.

```
{
        .c := 0;
        ..incr .= () -> { c := c + 1;
                := void;
        };
        ..read .= () -> c;
};

println(read());
//$ 0
incr();
println(read());
//$ 1
incr();
println(incr());
//$ void
```

Dans l'exemple ci-dessus, on se munit d'un contexte anonyme dans lequel on définit une variable privée `c` et on s'y réfère à l'aide de fonction qu'on exporte dans le contexte parent. On fait bien attention à ce que les fonctions renvoient autre chose que leur contexte d'exécution (ou tout autre contexte permettant de remonter au contexte anonyme), et voilà ! On ne peut plus interagir avec la variable `c` qu'avec les fonctions `read` et `incr`.

B.3.3 Classe comme Constructeur

Une manière de définir une classe est de définir son constructeur :

```
create_A .= (a) -> .{}; // .{} ici réfère à l'environnement où sont définis
↪  les arguments de la fonction.
instance_a := create_A(5);
println(instance_a.a);
//$ 5
println(instance_a);
//$ { .a := 5; }
```

C'est ainsi que sont utilisables les fonctions `create_signal_machine` et `create_configuration` (implémentées en java).

Parce que les fonctions capturent leur environnement, le plus simple pour définir une méthode, c'est-à-dire une fonction rattachée à l'instance d'un objet, est de créer ces méthodes comme fonctions membres à l'instanciation de l'objet. C'est ce qui est fait ici avec `get_a` et `set_a`.

```
create_A .= (a) -> .{
        .double_a .= () -> 2 * ..a;
};
instance_a := create_A(4);
println(instance_a.double_a());
//$ 8
```

Une alternative est d'utiliser des fonctions qui ne sont pas vraiment des méthodes, tel que `double_a .= (o) -> 2 * o`
Si on veut de l'encapsulation :

```
create_A .= () -> .{
        // define methods here for look-up
        .get_a := undef ;
        .set_a := undef ;
        .double_a := undef ;
        {
                .a := undef; // define `a` for look-up.
                ..get_a .= () -> a;
                ..set_a .= (val) -> { a := val; := void; };
                ..double_a .= () -> 2 * a ;
        }
};
```

Un problème de taille demeure : les instances sont descendantes du contexte parent du constructeur. Pour le résoudre, il suffit de construire l'instance dans le contexte appelant :

```
{
        .ctx .= "classes";
        ..create_A .= () -> {
                .double_a .= () -> 2 * a ;
        };
        ..create_B .= () -> ?{
                := {
                        .double_a .= () -> 2 * a ;
                };
        };
}
{
        .ctx .= "instances";
        instance_a := create_A();
        println(instance_a.ctx);
        //$ classes
        instance_b := create_B();
        println(instance_b.ctx);
        //$ instances
}
```

L'initialisation de données membres et plus généralement l'échange de données entre contexte appelant et contexte de définition des arguments est fastidieux, mais possible. Il est cependant plus idiomatique ici de ré-ouvrir le contexte, et de ne définir à la création que les valeurs "statiques" (communes à toutes les instances, comme, par exemple, les méthodes) :

```
make_A .= (a) -> {
        ?{
                .a := undef;
                .double_a .= () -> 2 * .a;
        }
        ?a := a ;
        := void;
};
i1 := { make_A(5); };

create_A .= () -> ?{ := { .double_a .= () -> 2 * .a; }; };
i2 := create_A { .a := 5; };
```

B.3.4 Prototype

Les prototypes sont une alternative minimaliste aux classes. Pour créer un objet, il suffit de le faire :

```
instance_a := {
        .a := 5 ;
        .double_a .= () -> 2 * ..a;
};
```

Avec encapsulation :

```
instance_a := {
        .get_a := undef ;
        .set_a := undef ;
        .double_a := undef ;
        {
                .a := 5 ;
                ..get_a .= () -> a;
                ..set_a .= (val) -> { a := val; := void; };
                ..double_a .= () -> 2 * ..a;
        }
};
```

B.3.5 Héritage Simple pour les Classes

Composer les constructeur est une manière de faire de l'héritage.

```
create_A .= (a) -> .{
        .double_a .= () -> 2 * ..a ;
};

create_B .= (a, b) -> {
        .res .= create_A(a) ;
        res.b := b;
        := res ;
};

instance_b .= create_B(2, 7);
println(instance_b.double_a()); //$ 4
println(instance_b);        //$ { .a := 2; .b := 7 ; .double_a := () -> 2 * ..
↪  a ; }
```

B.3.6 Héritage de Prototypes par Portée

Le mécanisme de portée peut être utilisé pour gérer l'héritage d'un prototype.

```
foo .= {
        .name .= "foo"; .one .= 1; .two .= 2;
        .say_my_name .= () -> println(name);
} ;
println(foo.one);         //$ 1
println(foo.two);         //$ 2
println(foo.three);         //$ undef
foo.say_my_name();         //$ foo

foo.{
        // create a new context inside of foo
        ..bar .= { .two .= "two"; three .= 3; };
}
println(bar.one);         //$ 1
println(bar.two);         //$ two
println(bar.three);         //$ 3
bar.say_my_name();         //$ foo
```

Un problème avec l'exemple ci-dessus est que l'on peut vouloir prototyper un objet depuis n'importe où, pas seulement un environnement facilement accessible depuis le prototype. Une manière de le résoudre est de se munir d'une méthode de construction :

```
foo .= {
        .name .= "foo"; .one .= 1; .two .= 2;
        .say_my_name .= () -> println(name);
        // create a new context inside of foo
        .create .= () -> {};
} ;

bar .= foo.create(){ .two .= "two"; three .= 3; };
```

Ou encore, avec une fonction constructeur générique :

```
like .= (prototype) -> prototype::{
        // create a new context inside of prototype
        := {};
};
foo .= {
        .name .= "foo"; .one .= 1; .two .= 2;
        .say_my_name .= () -> println(name);
} ;
bar .= like(foo){ .two .= "two"; three .= 3; };
```

Comme vu précédemment, les méthodes ont besoin d'un lien à leur objet

```
like .= (prototype) -> prototype::{        := {}; };
foo .= {
        .name .= "foo";
```

```
        .say_my_name .= () -> println(name);
} ;
bar .= like(foo){ .name .= "bar"; };
bar.say_my_name();        //$ foo
```

Une alternative à créer une copie des méthodes pour chaque instances est d'utiliser la portée dynamique et l'opérateur ? pour dénoter `self`.

```
like .= (prototype) -> prototype::{ := {}; };
foo .= {
        .name .= "foo";
        .say_my_name .= () -> println(?name);
};
bar .= like(foo){ .name .= "bar"; };

foo.{ say_my_name(); } //$ foo
bar.{ say_my_name(); } //$ bar
bar.{ println(.say_my_name); }        //$ undef
```

Cela utilise les contextes davantage pour ce qu'il sont, mais rend particulièrement difficile l'injection et l'extraction de donnée, tout particulièrement hors contexte

```
like .= (prototype) -> prototype::{        := {}; };
foo .= {
        .name .= "foo";
        .my_name_is .= (guess) -> guess == ?name;
} ;
bar .= like(foo){ .name .= "bar"; };

guess .= "slim";
res := undef;
bar.{
        ..res := my_name_is(..guess);
}
println(res);        //$ false
```

L'exemple ci-dessus marche seulement parce que le contexte d'exécution est ancêtre de notre instance. Si les méthodes ne travaillent que sur des champs de l'objet, et n'ont ni entrée ni sortie, pas de problème.

Une des limites de cette approche est l'impossibilité de lier et relier arbitrairement les contextes entre eux, comme ce peut être le cas avec les prototypes et les fonctions dans JavaScript. Plus simplement, un sucre syntaxique à la lua `obj:(self, ...) == obj.(obj, ...)` permettrait de singer à moindre frais l'usage de méthodes. Au passage, utiliser de simple fonctions, pourquoi pas membres, prenant l'objet en argument, est probablement l'approche la plus saine :

```
like .= (prototype) -> prototype::{        := {}; };
foo .= {
        .name .= "foo";
        .my_name_is .= (self, guess) -> guess == self.name;
} ;
bar .= like(foo){ .name .= "bar"; };

guess := "slim";
println(bar.my_name_is(bar, guess));        //$ false
```

B.3.7 Héritage multi-Niveaux

Pour faire de l'héritage multi-niveaux, il suffit de chaîner les héritages simples. On peut tout de même faire l'observation suivante : l'héritage par prototype nécessite une seule fonction, `like` (`Object` en JavaScript) en tout et pour tout, alors que le chaînage de constructeurs nécessite et permet d'écrire chaque constructeur.

L'approche suivante permet d'avoir le meilleur des deux mondes : un héritage prototypal par portée automatique pour les membres statiques :

```
make .= () -> ?{
        // bound methods and mandatory fields to create upon object
        ↪  initialisation
        .create .= () -> { make(); };
```

```
                .my_name_is .= (guess) -> guess == ..name;
                := void;
        };

        foo .= { make(); .name .= "foo"; } ;
        println(foo.my_name_is("foo"));         //$ true

        bar .= foo.create{} ;
        println(bar.my_name_is("bar"));         //$ false

        ciz .= bar.create{ .name .= "ciz"; };
        println(ciz.my_name_is("ciz"));         //$ true
```

et l'option de préciser un constructeur pour les classes intermédiaires :

```
        make .= () -> ?{
                .create := () -> { := ..{ make(); }; };
                .my_name_is := (guess) -> guess == ..name;
                := void;
        };

        foo .= { make(); .name .= "foo"; } ;

        bar .= foo.create{} ;
        bar.{
                .make_super .= make;
                .make .= () -> ?{
                        make_super();           // inheritance
                        .say_my_name := () -> println(..name);
                        .name := "bar";
                        := void;
                };
                make();
        }
        bar.say_my_name(); //$ bar

        ciz .= bar.create{ .name .= "ciz"; };
        println(ciz.my_name_is("ciz"));         //$ true
        ciz.say_my_name(); //$ ciz
```

B.3.8 Héritage Multiple

En utilisant des fonctions d'initialisation plutôt que des constructeurs, il est aisé de faire de l'héritage multiple.

```
make_A .= () -> ?{
        .a := 0 ;
};
make_B .= () -> ?{
        .b := 0 ;
}

obj .= { make_A(); make_B(); };
```

B.3.9 Polymorphisme

Par défaut, AGC dispose de fonctions membres et de "duck typing", et est donc capable d'un modicum de polymorphisme. Via le chargement de librairie java, il est aussi capable de surcharge d'opérateur et de faire du dispatch dynamique.

Annexe C

Langage MTRL, note technique

Le langage MTRL est minimal et peu soigné. Idéalement, il faudrait le réimplémenter, par exemple en Java en utilisant antlr, pour une interopérabilité maximale avec agc, et en profiter pour le garnir de quelques fonctionnalités de plus haut niveau.

La présente annexe ne se veut pas une explication exhaustive de l'implémentation des MTRL, mais plus comme un tour de jardin des curiosités techniques les plus intéressantes rencontrées. Cette implémentation mêle python, langage avec lequel le lecteur est supposé familier, et AGC.

C.1 Parsage et Représentation Interne

C.1.1 Parsage top-level

Au plus haut niveau, le parsage est une simple pile de `elif line.startswith("mot_clef"):`, avec un dictionnaire en guise d'environnement global, et déléguant le parsage des blocs configurations et machines.

C.1.2 Configuration

Le ruban bi-infini est représenté par une classe `Tape`. Il se comporte comme un tableau fainéant (il n'est pas vraiment de taille infinie, ses cases non explorées sont ajoutées juste à temps et renvoient une valeur par défaut). Il est manipulé comme un tableau, à l'aide de surcharge d'opérateurs.

```
class Tape(object):
    #a class to manipulate biinfinite tape.
    def __init__(self, pos = [], neg = [], center = 0):
        self.__positiveIndexes = [toFraction(i) for i in pos]
        #nonnegative atually
        self.__negativeIndexes = [toFraction(i) for i in neg]
        self.center = center

    def __getitem__(self, index):
            #returns the relevant value, or 0 if none found
            #self[ie]
            if index < 0:
                return self.__negativeIndexes[-1 - index] if -1 - index <
                ↪  len(self.__negativeIndexes) else 0
            else:
                return self.__positiveIndexes[index] if index <
                ↪  len(self.__positiveIndexes) else 0

    def __setitem__(self, index, value):
        # snip

# snip

    def read(self):
        return self[self.center]

    def write(self, item):
        self[self.center] = item

    def move(self, delta):
```

```
        self.center += delta

    def moveTo(self, p):
        self.center = p
```

Cette représentation est exposée à l'utilisateur, puisque la définition d'un ruban n'est autre qu'une exécution de code python avec un environnement particulier :

```
def fun_create_configuration(configuration_def):
    tape = Tape()
    local = {'x' : tape, 'a' : 0}
    exec(configuration_def, {'Fraction':Fraction, 'ONE':Fraction(1)}, local)
    if type(local['x']) != Tape:
        #if tape is entered as an array
        local['x'] = Tape(local['x'])
    if 'p' in local:
        local['x'].moveTo(local['p'])

    return local
```

C.1.3 Nombre, nombres flottants et fractions

Dans un contexte de calcul géométrique reposant sur des intersections précises, il apparaît naturel de travailler en précision parfaite avec des fractions. C'est exactement ce que nous faisons avec le bien nommé module fractions.

Un problème de taille se pose alors : comme vu à la section précédente, la définition des configuration est un script python. Cela signifie que `22/7`, par exemple, n'est pas interprété comme 22/7, mais comme un nombre flottant qui l'approche. Deux solutions s'offrent à nous :

— écrire un interpréteur qui n'évalue pas les nombres en flottants ;

— requérir une syntaxe plus lourde de la part de l'utilisateur, comme `Fraction(22, 7)` par exemple.

La seconde solution ne requiert aucun travail de notre part, si ce n'est que d'assurer que Fraction est disponible dans l'environnement.

Nous optons pour une troisième solution : retrouver, à partir d'un nombre flottant, la fraction dont il est (probablement) issue. Pour trouver, étant donné un nombre en virgule flottante, le nombre rationnel le plus simple qui, évalué, vaut ce nombre flottant, nous utilisons le développement en fraction continues.

```
class ContinuedFraction(list):
    """a class to handle finite continued fractions"""
    #The point is to help convert float into the simplest fraction yielding them.
    #It's a lazy yet lengthy hack to not write a proper interpreter for configurations
    #it's bad and should be removed once proper interpreter for configurations is
    ↪  done.
    def __init__(self, x = [0]):
        #snip
            # if x is float
            x0 = x
            self.append (floor(x))
            x = x - floor(x)
            while x != 0:
                #copy self
                cf = ContinuedFraction(self)
                f = cf.toFraction()
                if (f.numerator/f.denominator == x0):
                    return

                #just as a precaution, try continued fraction with the last term
                ↪  negative.
                cf = ContinuedFraction(self)
                cf.append(1+cf.pop())
                f = cf.toFraction()
                if (f.numerator/f.denominator == x0):
                    self.append(1+self.pop())
```

```
                    return

                x = 1/x
                self.append(floor(x))

                x = x - floor(x)

    def toFraction(self):
        if len(self) == 1:
            return Fraction(self[0])
        else:
            b = self.pop ()
            a = self.pop ()
            self.append(a + 1 / Fraction(b))
            return self.toFraction()
```

C.1.4 Compilation Machines

Le parsage d'une machine en langage MTRL se fait très simplement, ligne à ligne, avec une batterie de regex. La compilation en un graphe d'état est tout aussi simple.

On peut noter que la correspondance instruction - nœud n'est pas tout à fait parfaite. Par exemple, `a = 3 * h` est décomposé en `a =0` puis `a = 3*h` . Ceci est dans le but de simplifier la transpilation vers agc, en se rapprochant au plus près du fonctionnement de la simulation agc.

C.2 Transpilation AGC

C.2.1 Configuration

Le parallèle entre MTRL et machine à signaux concernant le schéma configuration, machine et calcul, est superficiel, et la correspondance très imparfaite.

Par exemple, une configuration initiale de machine à signaux ne fait sens que par rapport à un ensemble de méta-signaux, et donc par rapport à une machine à signaux. Par contraste, toutes les configurations de machines de Turing réelles ou rationnelles linéaires sont de même type, des rubans bi-infinis, et peuvent être définis indépendamment d'une machine. Pour surmonter cet écueil et tout de même transpiler configuration, machine, et calcul indépendamment, nous combinons deux techniques de méta programmation et compilation : chargement de code et fonctions rappels.

```
/*
configuration file automatically generated

usage :
myConfiguration.{
        load ".anonymous_config.agc";
};
*/

add_left_margin(MaxMove);

foreach(v:[0, 5/3])
        add_cell(v);
add_accu_cell(0, 5/4);
foreach(v:[5/2, 0])
        add_cell(v);

add_right_margin(MaxMove);
```

Par "chargement", `load`, nous entendons un mécanisme qui permet de charger le contenu d'un fichier tel quel à l'emplacement ou l'instruction load est appelée. Tout se passe comme si on avait copié le contenu du fichier chargé et collé à la place de l'instruction `load`.

L'instruction `load` est souvent utilisable comme une manière rudimentaire d'organiser du code en plusieurs fichiers, de charger des sortes de librairies ou modules. Ici elle permet de transpiler une configuration sans avoir le contexte nécessaire pour créer une configuration de machine à signaux. L'instruction load permettra de mettre cette configuration en contexte.

Pour donner ses instructions de comment simuler une configuration MTRL dans une configuration de machine à signaux, on utilise des fonction de rappel (`add_left_margin`, `add_cell`, `add_accu_cell`,

`add_right_margin`) ; ces fonctions sont définies dans un contexte (création de la machine à signaux) différent de celui dans lequel elles sont appelées (création de la configuration).

C.3 Machine

C.3.1 Commun

Les machines à signaux produites dépendent en partie de méta-signaux et de schémas structurels, communs à toutes, définis dans des libraires. C'est par exemple d'une telle librairie que viennent les fonctions de rappel vu précédemment.

```
Create_configuration_tape .= (Cdist, MaxR) ->
{
    := ..create_configuration()
    {
        .add_cell .= (value) ->
        {
            "Cell_marker" ..@ ..end;
            // ...
        };
        // ...
    }
}
```

Ce code est voué à être chargé dans une machine, à laquelle `..create_configuration` réfère.

En l'essence, `Create_configuration_tape` est le constructeur d'une classe qui étend celle construite par `create_configuration`.

C.3.2 Particulier

Les machines ont aussi des méta-signaux qui leur sont propres, et dépendent de la définition de leur machine MTRL sous-jacente. La transpilation des nœuds du graphe d'état de la machine MTRL se fait en trois étapes :

— une première passe permet de créer tous les méta-signaux stockant l'état, qui est matérialisé par un signal (au moins un méta-signal par état) ;

— une deuxième passe permet de générer les signaux internes à chacun de ces nœuds, ainsi que de collecter autant d'information que possible sur les collisions à construire reliant un nœud à l'autre ; spécifiquement, le traitement d'un nœud commence par un ensemble de signaux donnant des instructions et correspondant à la moitié sortante d'une règle de collision ; ce traitement se termine par un retour d'information, correspondant à la moitié entrante d'une règle de collision.

— une troisième phase permet, une fois toutes ces moitiés établies, de les rassembler, de former les collisions permettant de passer d'un état à un autre.

Table of symbols

Symbol	Definition	Page
$\mathbb{R}$	Real set	3
$\mathbb{N}$	L'ensemble des entiers naturels	17
$\mathbb{Z}$	L'ensemble des entiers relatifs	17
$\mathbb{Q}$	Rational set	17
D	L'ensemble des rationnels dyadiques	17
M	Ensemble des méta-signaux d'une machine à signaux $\mathcal{A}$	17
S	Vitesses des méta-signal d'une machine à signaux $\mathcal{A}$	17
R	Ensemble des règles de collisions d'une machine à signaux $\mathcal{A}$	17
ρ	Une règle d'une machine à signaux $\mathcal{A}$	17
ρ^-	Signaux entrant d'une règle de collision ρ	17
ρ^+	Signaux sortant d'une règle de collision ρ	17
ρ'	Une autre règle d'une machine à signaux $\mathcal{A}$	17
ρ'^-	Signaux entrant d'une autre règle de collision ρ'	17
$\overrightarrow{\text{Bleu}}$	Méta-signal $\overrightarrow{\text{Bleu}}$	17
$\overleftarrow{\text{Rouge}}$	Méta-signal $\overleftarrow{\text{Rouge}}$	17
Vert	Méta-signal Vert	17
c	Configuration d'une machine à signaux	17
$\oslash$	Vide (appartient à V)	17
$\circledast$	Accumulation (appartient à V)	17
x	Une coordonnée spatiale	18
$\mathbb{T}$	Un ensemble de temps, un intervalle de $\mathbb{R}^+$	18
$\ltimes$	La relation "être issue de"	18
μ	Un méta-signal d'une machine à signaux $\mathcal{A}$	18
t	Une coordonnée temporelle	18
t'	Une autre coordonnée temporelle	18
$\mathcal{D}$	Un diagramme espace temps	18
$\overleftarrow{\text{zag}}$	Méta-signal $\overleftarrow{\text{zag}}$	19
$\overleftarrow{\text{ri}}$	Méta-signal $\overleftarrow{\text{ri}}$	19
$\overrightarrow{\text{le}}$	Méta-signal $\overrightarrow{\text{le}}$	19
$\overrightarrow{\text{zig}}$	Méta-signal $\overrightarrow{\text{zig}}$	19
$\mathbb{R}\times\mathbb{T}$	L'espace-temps, produit cartésien de l'espace et du temps	19
c^+	La vitesse de méta signaux maximale	21
c^-	La vitesse de méta signaux minimale	21
$\mathcal{D}'$	Un autre diagramme espace temps	21
UB	L'ensemble des arbres unaire-binaires	23
main	Méta-signal main	24
x_i^0	Méta-signal x_i^0	24
x_i^v	Méta-signal x_i^v	24
x_2^v	Méta-signal x_2^v	24
x_2^0	Méta-signal x_2^0	24
$x_i^{v=0}$	Méta-signal $x_i^{v=0}$	24
$x_3^{v=0}$	Méta-signal $x_3^{v=0}$	24
x_3^0	Méta-signal x_3^0	24
x_3^v	Méta-signal x_3^v	24
x_0^0	Méta-signal x_0^0	24
x_0^v	Méta-signal x_0^v	24
x_1^0	Méta-signal x_1^0	24
x_1^v	Méta-signal x_1^v	24
	Méta-signal	24
	Méta-signal	24
test	Méta-signal test	25
test+	Méta-signal test+	25
test-	Méta-signal test-	25
réfraction	a refraction	25
réflexion	a reflection	25

Symbol	Definition	Page
aux	Méta-signal aux	26
réfractions	refractions	26
bound	Méta-signal bound	27
aux0	Méta-signal aux0	27
aux1	Méta-signal aux1	27
aux2	Méta-signal aux2	27
aux3	Méta-signal aux3	27
f	une fonction	27
l	Paramètre d'entré de l'algorithme SFSS (ordonnée de l'extrémité gauche)	29
r	Paramètre d'entré de l'algorithme SFSS (ordonnée de l'extrémité droite)	29
delay	Étape d'attente d'une machine (SFSS ou UB)	29
split	Étape de branchement d'une machine (SFSS ou UB)	29
l_0	paramètre l initial	29
r_0	paramètre r initial	29
d	La profondeur ou 2-profondeur d'un sommet	30
α	La pente du segment cible pour le problème SFSS	30
Δt	Un intervalle de temps	31
border	Méta-signal border	31
$\overrightarrow{\text{delay}}_{l-1}^{r-1}$	Méta-signal $\overrightarrow{\text{delay}}_{l-1}^{r-1}$	32
$\overrightarrow{\text{bounce}}_{\text{slw}}$	Méta-signal $\overrightarrow{\text{bounce}}_{\text{slw}}$	32
$\overleftarrow{\text{bounce}}$	Méta-signal $\overleftarrow{\text{bounce}}$	32
$\overrightarrow{\text{bounce}}$	Méta-signal $\overrightarrow{\text{bounce}}$	32
$\overleftarrow{\text{split}}_l^r$	Méta-signal $\overleftarrow{\text{split}}_l^r$	32
delay_l^r	Méta-signal delay_l^r	32
$\overrightarrow{\text{split}}_l^r$	Méta-signal $\overrightarrow{\text{split}}_l^r$	32
$\overleftarrow{\text{split}}_{l+l-1}^{r+l-1}$	Méta-signal $\overleftarrow{\text{split}}_{l+l-1}^{r+l-1}$	32
$\overrightarrow{\text{split}}_{l+r-1}^{r+r-1}$	Méta-signal $\overrightarrow{\text{split}}_{l+r-1}^{r+r-1}$	32
$\overrightarrow{\text{bounce}}^{\text{top}}$	Méta-signal $\overrightarrow{\text{bounce}}^{\text{top}}$	33
$\overrightarrow{\text{bounce}}^{\text{bot}}$	Méta-signal $\overrightarrow{\text{bounce}}^{\text{bot}}$	33
$\overrightarrow{\text{bounce}}_{\text{slw}}^{\text{bot}}$	Méta-signal $\overrightarrow{\text{bounce}}_{\text{slw}}^{\text{bot}}$	33
$\overrightarrow{\text{bounce}}_{\text{slw}}^{\text{top}}$	Méta-signal $\overrightarrow{\text{bounce}}_{\text{slw}}^{\text{top}}$	33
$\overleftarrow{\text{bounce}}^{\text{bot}}$	Méta-signal $\overleftarrow{\text{bounce}}^{\text{bot}}$	33
$\overleftarrow{\text{bounce}}^{\text{top}}$	Méta-signal $\overleftarrow{\text{bounce}}^{\text{top}}$	33
$\overrightarrow{\text{bounce}}_{\text{slw}}^{i}$	Méta-signal $\overrightarrow{\text{bounce}}_{\text{slw}}^{i}$	33
$\overleftarrow{\text{bounce}}^{i}$	Méta-signal $\overleftarrow{\text{bounce}}^{i}$	33
$\overrightarrow{\text{bounce}}^{i}$	Méta-signal $\overrightarrow{\text{bounce}}^{i}$	33
$\overleftarrow{\text{bounce}}^{j}$	Méta-signal $\overleftarrow{\text{bounce}}^{j}$	33
$\overrightarrow{\text{bounce}}^{j}$	Méta-signal $\overrightarrow{\text{bounce}}^{j}$	33
h_b	Height of the bouncing pairs of signals	33
w_t	Width of the tree macro-signal	33
$h_{b'}$	Height of the bouncing pairs of signals after a step	33
$w_{t'}$	Width of the tree macro-signal after a step	33
tree	Méta-signal tree	33
one	Méta-signal one	33
two	Méta-signal two	33
l	Méta-signal l	34
r	Méta-signal r	34
$\overleftarrow{\text{tree}}$	Méta-signal $\overleftarrow{\text{tree}}$	34
$\overleftarrow{\text{test}}$	Méta-signal $\overleftarrow{\text{test}}$	34
ő̈n̈ë	Méta-signal ő̈n̈ë	34
$\overleftarrow{\text{two}}$	Méta-signal $\overleftarrow{\text{two}}$	34
$\overleftarrow{\text{l}}$	Méta-signal $\overleftarrow{\text{l}}$	34
$\overleftarrow{\text{r}}$	Méta-signal $\overleftarrow{\text{r}}$	34
$\overleftarrow{\text{bound}}$	Méta-signal $\overleftarrow{\text{bound}}$	34
$\overrightarrow{\text{tree}}$	Méta-signal $\overrightarrow{\text{tree}}$	34
$\overrightarrow{\text{test}}$	Méta-signal $\overrightarrow{\text{test}}$	34
ö̀n̈ë	Méta-signal ö̀n̈ë	34
$\overrightarrow{\text{two}}$	Méta-signal $\overrightarrow{\text{two}}$	34
$\overrightarrow{\text{l}}$	Méta-signal $\overrightarrow{\text{l}}$	34
$\overrightarrow{\text{r}}$	Méta-signal $\overrightarrow{\text{r}}$	34
$\overrightarrow{\text{bound}}$	Méta-signal $\overrightarrow{\text{bound}}$	34
test_{dl}	Méta-signal test_{dl}	34
$\text{test}_{\text{splt}}$	Méta-signal $\text{test}_{\text{splt}}$	34
bound_{dl}	Méta-signal bound_{dl}	34
$\text{bound}_{\text{splt}}$	Méta-signal $\text{bound}_{\text{splt}}$	34

Symbol	Definition	Page
$\overleftarrow{\text{bound}_{\text{dl}}}$	Méta-signal $\overleftarrow{\text{bound}_{\text{dl}}}$	34
$\overleftarrow{\text{test}_{\text{dl}}}$	Méta-signal $\overleftarrow{\text{test}_{\text{dl}}}$	34
$\overrightarrow{\text{bound}_{\text{splt}}}$	Méta-signal $\overrightarrow{\text{bound}_{\text{splt}}}$	34
$\overrightarrow{\text{test}_{\text{splt}}}$	Méta-signal $\overrightarrow{\text{test}_{\text{splt}}}$	34
s_{cmp}	speed of computation	35
s	Speed (usually comes with a subscript like s_1) of a meta-singal	35
$\overrightarrow{\text{test}_{\text{dl}}}$	Méta-signal $\overrightarrow{\text{test}_{\text{dl}}}$	35
$\overleftarrow{\text{test}_{\text{splt}}}$	Méta-signal $\overleftarrow{\text{test}_{\text{splt}}}$	35
$\overrightarrow{\text{bound}_{\text{dl}}}$	Méta-signal $\overrightarrow{\text{bound}_{\text{dl}}}$	35
$\overleftarrow{\text{bound}_{\text{splt}}}$	Méta-signal $\overleftarrow{\text{bound}_{\text{splt}}}$	35
$\overleftarrow{\text{bounce}_{\text{dl}}}$	Méta-signal $\overleftarrow{\text{bounce}_{\text{dl}}}$	35
tree_{dl}	Méta-signal tree_{dl}	35
$\text{updt}^{*}_{\text{dl}}$	Méta-signal $\text{updt}^{*}_{\text{dl}}$	35
$\overrightarrow{\text{wall}_{\text{dl}}}$	Méta-signal $\overrightarrow{\text{wall}_{\text{dl}}}$	35
$\overrightarrow{\text{bound}_{\text{sdl}}}$	Méta-signal $\overrightarrow{\text{bound}_{\text{sdl}}}$	35
$\overrightarrow{\text{bounce}_{\text{s}}}$	Méta-signal $\overrightarrow{\text{bounce}_{\text{s}}}$	35
$\overrightarrow{\text{one}_{\text{s}}}$	Méta-signal $\overrightarrow{\text{one}_{\text{s}}}$	35
$\overrightarrow{\text{dsep}_2}$	Méta-signal $\overrightarrow{\text{dsep}_2}$	36
$\overleftarrow{\text{dsep}_1}$	Méta-signal $\overleftarrow{\text{dsep}_1}$	36
s_{shr}	speed of intermediate signals when shrinking into delay	36
$s^{\text{fst}}_{\text{shr}}$	speed of the intermediate wannabe bounceRSlow signal when shrinking into delay	36
$\overrightarrow{\text{two}_{\text{s}}}$	Méta-signal $\overrightarrow{\text{two}_{\text{s}}}$	36
$\overrightarrow{\text{r}_{\text{s}}}$	Méta-signal $\overrightarrow{\text{r}_{\text{s}}}$	36
$\overrightarrow{\text{l}_{\text{s}}}$	Méta-signal $\overrightarrow{\text{l}_{\text{s}}}$	36
$\text{wall}_{\text{splt}}$	Méta-signal $\text{wall}_{\text{splt}}$	36
$\overrightarrow{\text{bounce}_{\text{s}}}$	Méta-signal $\overrightarrow{\text{bounce}_{\text{s}}}$	37
$\overleftarrow{\text{bounce}_{\text{sb}}}$	Méta-signal $\overleftarrow{\text{bounce}_{\text{sb}}}$	37
$\overrightarrow{\text{bound}_{\text{s}}}$	Méta-signal $\overrightarrow{\text{bound}_{\text{s}}}$	37
$\overleftarrow{\text{bound}_{\text{sb}}}$	Méta-signal $\overleftarrow{\text{bound}_{\text{sb}}}$	37
$\overrightarrow{\text{one}_{\text{sb}}}$	Méta-signal $\overrightarrow{\text{one}_{\text{sb}}}$	37
$\overrightarrow{\text{updt}^{*}_{\text{splt}}}$	Méta-signal $\overrightarrow{\text{updt}^{*}_{\text{splt}}}$	37
$\text{updt}^{*}_{\text{splt}}$	Méta-signal $\text{updt}^{*}_{\text{splt}}$	37
$s^{\text{back}}_{\text{shrink}}$	speed of intermediate signals when shrinking from leftward to right ward	37
$s^{\text{bounce}}_{\text{shrink}}$	speed of intermediate signals when shrinking from leftward to rightward	37
$s^{\text{bounceBack}}_{\text{shrink}}$	speed of intermediate signals when shrinking from leftward to right ward	37
$\overleftarrow{\text{bounce}_{\text{s}}}$	Méta-signal $\overleftarrow{\text{bounce}_{\text{s}}}$	37
$\overrightarrow{\text{bounce}_{\text{sb}}}$	Méta-signal $\overrightarrow{\text{bounce}_{\text{sb}}}$	37
$\overleftarrow{\text{one}_{\text{s}}}$	Méta-signal $\overleftarrow{\text{one}_{\text{s}}}$	37
$\overleftarrow{\text{two}_{\text{s}}}$	Méta-signal $\overleftarrow{\text{two}_{\text{s}}}$	37
$\overleftarrow{\text{r}_{\text{s}}}$	Méta-signal $\overleftarrow{\text{r}_{\text{s}}}$	37
$\overleftarrow{\text{l}_{\text{s}}}$	Méta-signal $\overleftarrow{\text{l}_{\text{s}}}$	37
$\overleftarrow{\text{bound}_{\text{s}}}$	Méta-signal $\overleftarrow{\text{bound}_{\text{s}}}$	37
$\overrightarrow{\text{one}_{\text{sb}}}$	Méta-signal $\overrightarrow{\text{one}_{\text{sb}}}$	37
$\overrightarrow{\text{two}_{\text{sb}}}$	Méta-signal $\overrightarrow{\text{two}_{\text{sb}}}$	37
$\overrightarrow{\text{r}_{\text{sb}}}$	Méta-signal $\overrightarrow{\text{r}_{\text{sb}}}$	37
$\overrightarrow{\text{l}_{\text{sb}}}$	Méta-signal $\overrightarrow{\text{l}_{\text{sb}}}$	37
$\overrightarrow{\text{bound}_{\text{sb}}}$	Méta-signal $\overrightarrow{\text{bound}_{\text{sb}}}$	37
$\overleftarrow{\text{two}_{\text{sb}}}$	Méta-signal $\overleftarrow{\text{two}_{\text{sb}}}$	37
$\overleftarrow{\text{r}_{\text{sb}}}$	Méta-signal $\overleftarrow{\text{r}_{\text{sb}}}$	37
$\overleftarrow{\text{l}_{\text{sb}}}$	Méta-signal $\overleftarrow{\text{l}_{\text{sb}}}$	37
$\text{updt}^{\text{lo}}_{\text{splt}}$	Méta-signal $\text{updt}^{\text{lo}}_{\text{splt}}$	37
$\text{updt}^{\text{hi}}_{\text{splt}}$	Méta-signal $\text{updt}^{\text{hi}}_{\text{splt}}$	37
$\overleftarrow{\text{bounce}^{1}_{\text{splt}}}$	Méta-signal $\overleftarrow{\text{bounce}^{1}_{\text{splt}}}$	37
$\overrightarrow{\text{sep}_1}$	Méta-signal $\overrightarrow{\text{sep}_1}$	37
$\overleftarrow{\text{bound}_1}$	Méta-signal $\overleftarrow{\text{bound}_1}$	37
$\overrightarrow{\text{sep}_3}$	Méta-signal $\overrightarrow{\text{sep}_3}$	37
$\overleftarrow{\text{sep}_2}$	Méta-signal $\overleftarrow{\text{sep}_2}$	37
$\overleftarrow{\text{bounce}_{\text{splt}}}$	Méta-signal $\overleftarrow{\text{bounce}_{\text{splt}}}$	38
$\text{bound}^{1}_{\text{splt}}$	Méta-signal $\text{bound}^{1}_{\text{splt}}$	38
one_1	Méta-signal one_1	38
two_1	Méta-signal two_1	38
s_{bnc}	absolute speed of bouncing signals	38
$\text{test}_{\text{start}}$	Méta-signal $\text{test}_{\text{start}}$	38
$\text{updt}^{0}_{\text{dl}}$	Méta-signal $\text{updt}^{0}_{\text{dl}}$	38
$\text{updt}^{1}_{\text{dl}}$	Méta-signal $\text{updt}^{1}_{\text{dl}}$	38
one_{back}	Méta-signal one_{back}	38

Symbol	Definition	Page
$\mathrm{r}_{\mathrm{back}}$	Méta-signal $\mathrm{r}_{\mathrm{back}}$	38
$\mathrm{l}_{\mathrm{back}}$	Méta-signal $\mathrm{l}_{\mathrm{back}}$	38
minus^{0}	Méta-signal minus^{0}	38
minus^{1}	Méta-signal minus^{1}	38
$\mathrm{updt}^{0}_{\mathrm{dl}}$	Méta-signal $\mathrm{updt}^{0}_{\mathrm{dl}}$	38
$\mathrm{updt}^{1}_{\mathrm{dl}}$	Méta-signal $\mathrm{updt}^{1}_{\mathrm{dl}}$	38
$\grave{\mathrm{minus}}^{0}$	Méta-signal $\grave{\mathrm{minus}}^{0}$	38
$\grave{\mathrm{minus}}^{1}$	Méta-signal $\grave{\mathrm{minus}}^{1}$	38
$\grave{\mathrm{one}}_{\mathrm{back}}$	Méta-signal $\grave{\mathrm{one}}_{\mathrm{back}}$	38
$\grave{\mathrm{l}}_{\mathrm{back}}$	Méta-signal $\grave{\mathrm{l}}_{\mathrm{back}}$	38
$\grave{\mathrm{r}}_{\mathrm{back}}$	Méta-signal $\grave{\mathrm{r}}_{\mathrm{back}}$	38
$\grave{\mathrm{test}}_{\mathrm{start}}$	Méta-signal $\grave{\mathrm{test}}_{\mathrm{start}}$	38
$\mathrm{updt}^{*}_{\mathrm{dl}}$	Méta-signal $\mathrm{updt}^{*}_{\mathrm{dl}}$	38
$\mathrm{updt}^{0}_{\mathrm{dl}}$	Méta-signal $\mathrm{updt}^{0}_{\mathrm{dl}}$	38
$\mathrm{updt}^{1}_{\mathrm{dl}}$	Méta-signal $\mathrm{updt}^{1}_{\mathrm{dl}}$	38
$\acute{\mathrm{minus}}^{0}$	Méta-signal $\acute{\mathrm{minus}}^{0}$	38
$\acute{\mathrm{minus}}^{1}$	Méta-signal $\acute{\mathrm{minus}}^{1}$	38
$\acute{\mathrm{one}}_{\mathrm{back}}$	Méta-signal $\acute{\mathrm{one}}_{\mathrm{back}}$	38
$\acute{\mathrm{l}}_{\mathrm{back}}$	Méta-signal $\acute{\mathrm{l}}_{\mathrm{back}}$	38
$\acute{\mathrm{r}}_{\mathrm{back}}$	Méta-signal $\acute{\mathrm{r}}_{\mathrm{back}}$	38
$\acute{\mathrm{test}}_{\mathrm{start}}$	Méta-signal $\acute{\mathrm{test}}_{\mathrm{start}}$	38
$\mathrm{updt}^{*}_{\mathrm{dl}}$	Méta-signal $\mathrm{updt}^{*}_{\mathrm{dl}}$	38
$\mathrm{updt}^{\mathrm{lo}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{lo}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{hi}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{hi}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{bck}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{bck}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{st}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{st}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{rst}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{rst}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{st\_1}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{st\_1}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{lst}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{lst}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{*}_{\mathrm{dl}}$	Méta-signal $\mathrm{updt}^{*}_{\mathrm{dl}}$	39
$s^{\mathrm{up}}_{\mathrm{splt}}$	speed of the upper signal involved in updating parameters after a split	39
$\mathrm{updt}^{\mathrm{bck}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{bck}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{*}_{\mathrm{dl}}$	Méta-signal $\mathrm{updt}^{*}_{\mathrm{dl}}$	39
$\mathrm{updt}^{\mathrm{st}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{st}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{st\_1}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{st\_1}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{lst}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{lst}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{rst}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{rst}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{*}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{*}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{hi}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{hi}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{lo}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{lo}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{i}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{i}_{\mathrm{splt}}$	39
$\mathrm{updt}^{i}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{i}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{bck}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{bck}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{st}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{st}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{\mathrm{st\_1}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{\mathrm{st\_1}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{i\mathrm{st}}_{\mathrm{splt}}$	Méta-signal $\mathrm{updt}^{i\mathrm{st}}_{\mathrm{splt}}$	39
$\mathrm{updt}^{*}_{\mathrm{dl}}$	Méta-signal $\mathrm{updt}^{*}_{\mathrm{dl}}$	39
$\mathcal{I}$	L'espace d'entrée d'une machine BSSL ou TMRL	41
$\mathcal{O}$	L'espace de sortie d'une machine BSSL ou TMRL	41
$\mathcal{S}$	L'espace de calcul d'une machine BSSL ou TMRL	41
$\mathcal{G}$	Le graphes des états d'une machine BSSL ou TMRL	41
η	Un état, un nœud de $\mathcal{G}$	41
g_η	La fonction de calcul associé au nœud η	41
h_η	La fonction de calcul associé au nœud η	41
γ^{+}_{η}	Un successeur de η	41
γ_η	Un successeur η, c'est à dire qu'il y a une arête allant de γ_η à η	41
γ^{-}_{η}	Un successeur de η	41
$\mathbf{x}$	le ruban de calcul d'une machine de Turing	41
p	la position sur le ruban de calcul	42
h	La valeur présentement lue sur le ruban	42
a	La valeur de l'accumulateur	42
$\gamma^{=}_{\eta}$	Un successeur de η	42
$+=$	Opération d'ajout	43
$-=$	Opération de retrait	45
α_{max}	La plus grande contante multiplicative	52

Symbol	Definition	Page
α	Une constante multiplicative	52
Δx	Un intervalle spatiale	52
m_{max}	Le plus grand movement de tête	53
γ_η^l	Un successeur de η	59
γ_η^r	Un successeur de η	59
`stop`	Étape d'arrêt d'une machine (UB)	66
UB^e	L'ensemble des arbre unaire-binaires de feuilles étiqueté	66
ϕ	Fonction servant à développer un arbre partiel	67
ψ	Fonction servant à réduire un arbre partiel	67
$\mathcal{C}ol$	L'ensemble des collisions possibles d'un diagramme	80
$\mathcal{S}ig$	L'ensemble des signaux bornés possibles d'un diagramme	80
x'	Une autre coordonnée spatiale	80
$\mathcal{S}ig'$	L'ensemble des signaux non bornés possibles d'un diagramme	80
τ	Une transition	81
τ'	Une transition	81
$\mathcal{A}$	Machine à signaux	81
J	Cône de lumière dans un diagramme espace-temps	81
J^-	Cône de lumière arrière dans un diagramme espace-temps	81
J^+	Cône de lumière avant dans un diagramme espace-temps	81
V	Ensemble des valeurs étendues d'une machine à signaux $\mathcal{A}$ pour définir des diagrammes espace-temps	109

Aurélien EMMANUEL

Courbes d'accumulations des machines à signaux

Cette thèse s'inscrit dans l'étude d'un modèle de calcul géométrique : les machines à signaux.
Nous y montrons comment tracer des graphes de fonctions à l'aide d'arbres unaire-binaires.
Dans le monde des automates cellulaires, il est souvent question de particules ou signaux : des structures périodiques dans le temps et l'espace, autrement dit des structures qui se déplacent à vitesse constante. Lorsque plusieurs signaux se rencontrent, une collision a lieu, et les signaux entrant peuvent continuer, disparaître ou laisser place à d'autres signaux, en fonctions des règles de l'automate cellulaire.
Les machines à signaux sont un modèle de calcul qui reprend ces signaux comme briques de base. Visualisées dans un diagramme espace-temps, l'espace en axe horizontal et le temps vertical s'écoulant vers le haut, ce modèle revient à calculer par le dessin de segments et demi-droites colorés. On trace, de bas en haut, des segments jusqu'à ce que deux ou plus s'intersectent, et l'on démarre alors de nouveau segments, en fonctions de règles prédéfinies.
Par rapport aux automates cellulaires, les machines à signaux permettent l'émergence d'un nouveau phénomène : la densité des signaux peut être arbitrairement grande et même infinie, y compris en partant d'une configuration initiale de densité finie. De tels points du diagramme espace-temps, des points au voisinage desquels se trouvent une infinité de signaux, sont appelés points d'accumulation. Ce nouveau phénomène permet de définir de nouveau problèmes, géométriquement. Par exemple : quels sont les points d'accumulations isolés possibles en utilisant des positions initiales et des vitesses rationnelles ? Peut-on faire en sorte que l'ensemble des points d'accumulation forment un segment ? un ensemble de Cantor ?
Dans cette thèse, nous nous attelons à caractériser des graphes de fonctions qu'il est possible de dessiner par un ensemble d'accumulation. Elle s'inscrit dans l'exploration de la puissance de calcul des machines à signaux, qui s'inscrit plus généralement dans l'étude de la puissance de calcul de modèles non standards.
Nous y montrons que les fonctions d'un segment compact de la droite réelle dont le graphe coïncide avec l'ensemble d'accumulation d'une machine à signaux sont exactement les fonctions continues. Nous montrons plus généralement comment les machines à signaux peuvent dessiner n'importe quel fonction semi-continue inférieurement. Nous étudions aussi la question sous des contraintes de calculabilité, avec le résultat suivant : si un diagramme de machine à signaux calculable coïncide avec le graphe d'un fonction suffisamment lipschitzienne, cette fonction est limite calculable d'une suite croissante de fonctions en escalier rationnelles.

Mots clés : automates cellulaires, modèles de calcul non standards, points d'accumulation, calcul analogique, calcul fractal, machines à signaux.

Accumulation curves of signal-machines

This thesis studies a geometric computational model : signal machines.
We show how to draw function graphs using-binary trees.
In the world of cellular automata, we often consider particles or signals : structures that are periodic in time and space, that is, structures that move at constant speed. When several signals meet, a collision occurs, and the incoming signals can continue, disappear, or give rise to new signals, depending on the rules of the cellular automaton.
Signal-machines are a computational model that takes these signals as basic building blocks. Visualized in a space-time diagram, with space on the horizontal axis and time running upwards, this model consists of calculating by drawing segments and half-lines. We draw segments upwards until two or more intersect, and then start new segments, according to predefined rules.
Compared to cellular automata, signal-machines allow for the emergence of a new phenomenon : the density of signals can be arbitrarily large, even infinite, even when starting from a finite initial configuration. Such points in the space-time diagram, whose neighborhoods contain an infinity of signals, are called accumulation points.
This new phenomenon allows us to define new problems geometrically. For example, what are the isolated accumulation points that can be achieved using rational initial positions and rational velocities ? Can we make so the set of accumulation points is a segment ? A Cantor set ?
In this thesis, we tackle the problem of characterizing the function graphs that can be drawn using an accumulation set. This work fits into the exploration of the computational power of signal-machines, which in turn fits into the study of the computational power of non-standard models.
We show that the functions from a compact segment of the line of Real numbers whose graph coincides with the accumulation set of a signal machine are exactly the continuous functions.
More generally, we show how signal machines can draw any lower semicontinuous function. We also study the question under computational constraints, with the following result : if a computable signal-machine diagram coincides with the graph of a Lipschitz-function of sufficiently small Lipschitz coefficient, then that function is the limit of a growing and computable sequence of rational step functions.

Keywords : celullar automata, computation model, limit points, analog computing, fractal computing, signal machines.

Laboratoire d'Informatique Fondamentale d'Orléans
rue Léonard de Vinci 45067 ORLEANS Cedex 2