Les machines de Turing en temps transfini

Pierre Le Scornet

22 mai - 29 juin 2018

Résumé

L'objet de ce stage est l'étude des machines de Turing en temps transfini (ITTM, Infinite Time Turing Machine). C'est un modèle généralisé des machines de Turing classiques, formalisé par Joel David Hamkins et Andy Lewis à la fin des années 1990 [HL00]. Après des rappels sur les ordinaux, notamment la notion d'admissibilité, nous étudierons quelques propriétés de ces machines.

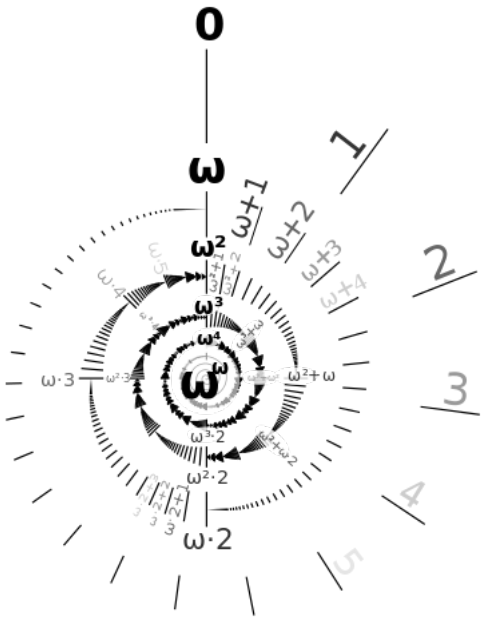

FIGURE 1 – Représentation des premiers ordinaux jusqu'à ω^ω. [1]

1. Source : Wikipedia, `Omega-exp-omega-labeled.svg`, sur la page *Ordinal number* (en anglais)

Table des matières

1 Introduction

1.1 Déroulement du stage

J'ai commencé ce stage par un rappel sur les ordinaux, que j'ai pris dans le cours de complément de la première année. Ensuite, j'ai regardé [HL00] pour apprendre comment fonctionne une ITTM. J'ai ensuite réécrit les preuves classiques sur les ITTM, notamment celle de la sous-section 2.3 : *Toute ITTM s'arrête ou boucle après un temps dénombrable, avec une longueur de boucle dénombrable.*. Ensuite, mes tuteurs m'ont donné plusieurs exercices, notamment sur l'écriture de $\omega_1^{\mathtt{CK}}$ et de $\omega_1^{\mathtt{CK}}.2$. Ces preuves ne sont pas présentes dans la littérature, mais elles s'obtiennent naturellement à partir d'elle. Enfin, mes tuteurs m'ont fait montrer un résultat plus général sur le temps d'écriture d'un ordinal, résultat qui n'a jamais été rédigé. Pour constuire la preuve, il m'a fallu beaucoup interagir avec mes tuteurs et d'autres stagiaires plus âgés, ce qui m'a permis un premier contact concret avec le monde de la recherche.

1.2 Plan du rapport

Ce rapport s'articule en trois parties :

- D'abord, on a une partie de définitions, avec :
  - un rappel sur les ordinaux,
  - la présentation des ITTM,
  - les premières propriétés sur les ordinaux, notamment le dénombrabilité des temps de fin d'exécutions d'ITTM, quelques définitions sur les ordinaux et une preuve de dénombrabilité,
  - la présentation de la notion d'admissibilité, avec des caractérisations pour les ordinaux dénombrables et quelques propriétés.
- Ensuite, une partie contenant quelques outils de calculs pour voir le potentiel des ITTM et qui serviront dans la suite :
  - comment simuler toutes les ITTM en parallèle quasiment en temps réel,
  - comment écrire un ordre total de même type qu'un ordre partiel mis en entrée,
  - comment prouver qu'un produit d'ordinaux calculables est calculable,
  - comment parallèliser une ITTM avec une machine de Turing sous certaines conditions de stabilité.
- Enfin, la dernière partie contient des résultats sur les temps d'écriture d'ITTM, avec
  - des propriétés sur les ordinaux utiles pour la suite,

— des liens entre les notions d'horlogeabilité et d'admissibilité, avec notamment le Lemme Speed-up,
— un premier exercice sur le temps d'écriture de $\omega_1^{\mathtt{CK}}$,
— un second exercice sur l'écriture des récursifs en $\omega_1^{\mathtt{CK}}$,
— une description de la structure des brèches dans les horlogeables, leur nombre et leur longueur notamment
— le résultat principal sur le temps d'écriture de tout ordinal par une ITTM

2 Définitions

2.1 Ordinaux

Un ordinal est un ensemble *transitif* et *bien ordonné par* $\in$. Von Neumann a introduit une construction de ces ordinaux, où 0 est $\varnothing$, 1 est $\{\varnothing\}$, et on peut définir $\alpha + 1 := \alpha \cup \{\alpha\}$. Certains ordinaux ne sont pas sous la forme $\beta = \alpha + 1$, ils sont appelés limite. Le premier ordinal limite est ω, qui est le premier ordinal infini, union de tous les ordinaux finis. On note ω_1 le premier ordinal non dénombrable.
Soient α et β deux ordinaux. On définit leur *somme* par l'induction transfinie suivante :

$$\begin{gathered}\alpha + 0 = \alpha \\ \alpha + (\beta + 1) = (\alpha + 1) + \beta \\ \text{Si } \beta \text{ limite, } \alpha + \beta = \bigcup_{\gamma<\beta} (\alpha + \gamma)\end{gathered}$$

On définit aussi leur *produit* comme suit :

$$\begin{gathered}\alpha.0 = 0 \\ \alpha.(\beta + 1) = (\alpha.\beta) + \alpha \\ \text{Si } \beta \text{ limite, } \alpha.\beta = \bigcup_{\gamma<\beta} (\alpha.\gamma)\end{gathered}$$

Remarquons que ni la somme ni le produit ne sont commutatifs : $2 + \omega = \omega$, alors que $\omega + 2 > \omega$; $2.\omega = \omega$ alors que $\omega.2 = \omega + \omega > \omega$.
Enfin, on définit la *puissance* comme suit :

$$\begin{gathered}\alpha^0 = 1 \\ \alpha^{\beta+1} = \alpha^\beta.\alpha \\ \text{Si } \beta \text{ limite, } \alpha^\beta = \bigcup_{\gamma<\beta} (\alpha^\gamma)\end{gathered}$$

La notion d'ordinal combinée avec l'induction transfinie permet de prouver dans ZF des théorèmes non démontrables dans l'arithmétique de Peano (voir le théorème de Goodstein dans [Goo44] et son développement par Kirby et Paris [KP82]). Elle est particulièrement intéressante pour étudier les niveaux de prouvabilité au-dessus de l'arithmétique de Peano.

2.2 Les ITTM

Nous introduisons dans cette section les ITTM (Infinite Time Turing Machine), généralisations des Machines de Turing qui travaillent sur des durées transfinies.
Dans la suite, nous appellerons *réel* toute suite sur $\{0,1\}$ indexée par $\mathbb{N}$.
Les ITTM sont donc des modèles de calcul généralisant les machines de Turing classiques ; elles ont été formalisées par Hamkins et Lewis dans [HL00].
Nous considérons une machine analogue aux Machines de Turing, à trois bandes : la bande d'entrée sur laquelle se trouve l'entrée $(e_i)_{i\in\mathbb{N}}$, la bande de travail $(t_i)_{i\in\mathbb{N}}$, et la bande de sortie $(s_i)_{i\in\mathbb{N}}$. Ces trois bandes contiennent donc à tout moment chacune un réel.

Tête de lecture	↓									
Entrée	0	1	0	0	1	0	0	0	1	...
Travail	0	0	0	0	0	0	0	0	0	...
Sortie	0	0	0	0	0	0	0	0	0	...

FIGURE 2 – Bandes d'une ITTM sur entrée 010... à l'état initial

Pour s'arrêter, ou passer du temps α au temps $\alpha+1$, elle se comporte comme une machine à trois bandes classique : en fonction de (e_i, t_i, s_i, q) avec i la position de la tête de lecture et q l'état actuel de la machine, on modifie les cases (t_i, s_i), on décale à droite ou à gauche la tête et on passe dans un autre état ; on s'arrête dès qu'on arrive à un état final en renvoyant le réel inscrit sur la bande de sortie. Il suffit maintenant de définir le passage à un temps limite α : pour cela, on ajoute un état à la machine nommé état limite et noté $q_{\mathtt{L}}$. Au temps limite, on fait revenir la tête de lecture à sa position initiale, et on passe les cases de tous les rubans à leur limsup. Cela veut exactement dire que si une case a alterné entre 0 et 1 jusqu'à α, elle contiendra un 1, et si elle est ultimement constante égale à 0 ou 1, alors cette valeur est inchangée au temps α.
On peut définir les ITTM plus formellement par :

Définition 2.1 ([HL00])**.** Une *ITTM* est un sextuplet $(\Sigma, \mathcal{Q}, q_0, \mathcal{Q}_F, q_{\mathtt{L}}, \delta)$ avec :

- $\Sigma = \{0, 1\}$.
- $\mathcal{Q}$ est un ensemble d'états **fini**.
- $q_0 \in \mathcal{Q}$ est dit état *initial*, où commence la machine au temps 0.
- $q_{\mathtt{L}} \in \mathcal{Q}$ est dit état *limite* où se trouve la machine en tout temps limite.
- $\mathcal{Q}_F \subseteq \mathcal{Q}$ est l'ensemble des états finaux.
- δ est une fonction totale $(e, t, s, q) \in \Sigma^3 \times \mathcal{Q} \to (t', s', q', d) \in \Sigma^2 \times \mathcal{Q} \times \{\leftarrow, \rightarrow\}$.

Fonctionnement :

- Si cette ITTM est dans l'état q avec les bandes (e, t, s) au temps successeur avec la tête de lecture/écriture en n, on remplace (t_n, s_n, q) et on déplace la tête de lecture/écriture selon ce que renvoie δ.
- Pour la passer à un temps limite α, on remet la tête de lecture/écriture en 0, on passe la machine à l'état $q_{\mathtt{L}}$, et on pose $\forall i \in \mathbb{N}, b \in \{t, s\}, b_{i,\alpha} := \limsup_{\gamma<\alpha} b_{i,\gamma}$
- Quand on arrive dans une configuration où l'action suivante nous amène dans un état final, alors on exécute cette dernière action et on renvoie la bande de sortie. [2]

Temps 0	0	0	0	0	0	0	0	0	0	...
10	0	1	1	0	0	0	0	0	1	...
100	0	1	0	1	0	0	1	0	1	...
1000	0	1	1	1	1	0	0	0	1	...
2000	0	1	0	1	0	0	0	0	1	...
5000	0	1	1	1	0	0	0	0	1	...
10000	0	1	0	1	1	0	0	0	1	...
30000	0	1	1	1	0	0	0	0	1	...
					...					
ω	0	1	1	1	1	0	0	0	1	...

FIGURE 3 – Passage d'une bande à la limite

- Si la tête de lecture/écriture se trouve en 0 et que δ essaye de la

2. On peut définir les ITTM et leur fonctionnement de façon différente : à une bande, qui va à droite quand on est tout à gauche et qu'on essaye d'aller à gauche, etc. La plupart du temps, elles ont les mêmes possibilités, à un temps fini près.

déplacer vers la gauche, on reste en 0.

Les ITTM sont capables de simuler n'importe quelle machine de Turing normale en temps inférieur ou égal à ω : on peut même décider du problème de l'arrêt de machines de Turing normales en temps ω, juste en simulant cette machine et en regardant si elle s'arrête avant ω : on simule notre machine, et si elle s'arrête on renvoie oui, sinon quand on passe à l'état limite pour la première fois, on s'arrête et on renvoie non.

On peut donner un exemple de ce que nous permet de faire cette règle de la limsup. D'abord, pour s'arrêter en ω, il suffit de dire que l'état limite, peu importe les cases en 0, envoie vers l'état final. Ensuite, pour s'arrêter en ω^2, il suffit de doubler ce programme pour y arriver. Mais pour s'arrêter en $\omega.2$, utilisons l'ITTM marchant pendant un temps ω en se réservant une case au début de la bande de travail. Lançons la, et à chaque fois qu'elle finit passons à 1 puis à 0 la case. Ainsi, à chaque temps $\omega.k$, cette case passe à 1 : au temps $\omega^2 = \bigcup_{k<\omega} \omega.k$, cette case sera à 1 pour la première fois et on peut s'arrêter. De la même façon, on pourrait trouver ω^ω, ω^{ω^ω}, et même $\omega^{\omega^{\omega^{\cdots}}} = \varepsilon_0$.

2.3 Temps ordinaux

D'abord, nous allons justifier pourquoi nous limitons notre étude aux ordinaux dénombrables, par la propriété suivante :

Proposition 2.1. *Si une ITTM s'arrête, elle s'arrête en un temps qui est un ordinal dénombrable.*

Démonstration. Montrons que si une ITTM ne s'est pas arrêtée avant ω_1, alors il existe un temps dénombrable où l'ITTM s'est trouvée exactement dans la même configuration. Considérons l'état de l'ITTM au temps ω_1 : les bandes sont $(e_i), (t_i), (s_i)$, l'état est $q_\mathtt{L}$ et la tête de lecture/écriture est en position 0. Pour chaque $i \in \mathbb{N}$, la valeur de e_i, t_i ou s_i s'est soit stabilisée en 0 ou 1 avant ω_1, soit a infiniment alterné entre les deux (car ω_1 limite). Notons α_0 le sup des temps où les cases se sont stabilisées : cet ordinal est le sup dénombrable de temps dénombrable, donc il est dénombrable. Ensuite, considérons la suite $(\alpha_n)_{n\in\mathbb{N}}$ telle que en α_{n+1}, toutes les cases qui ne se sont pas stabilisées ont changé au moins une fois depuis α_n. Par le même argument qu'avant, α_n est dénombrable, donc $\beta = \mathtt{sup}_{n\in\mathbb{N}}\alpha_n$ est aussi un ordinal dénombrable. β est le sup d'une suite strictement croissante d'ordinaux, c'est donc un ordinal *limite*. On a donc la machine qui au temps β est à l'état $q_\mathtt{L}$, la tête de lecture/écriture en 0 et dont les cases contiennent 0 si et seulement si elle s'est stabilisée avant α_0, c'est-à-dire exactement la

même valeur qu'en ω_1 : on a donc que la machine est exactement dans la même configuration en β qu'en ω_1. Ainsi, la machine va se comporter de la même manière après ω_1 qu'après β : puisque $\beta + \omega_1 = \omega_1$, on a montré que l'ITTM boucle dans une boucle de longueur ω_1, donc elle ne s'arrêtera jamais. □

Par un raisonnement similaire, on peut dire qu'il existe un temps $\beta+\gamma$ après lequel on retrouve la même configuration qu'en β : il suffit de reconstruire une suite (α_n) dont le sup sera $\beta + \gamma$, en prenant α_{n+1} le premier temps après lequel les cases n'ayant pas stabilisé ont toutes changées de valeur au moins une fois. De même, au temps $\beta + \gamma.2$, $\beta + \gamma.3$ ... puis $\beta + \gamma.\omega$ et tous les $\beta + \gamma.\psi$ avec ψ n'importe quel ordinal non nul, on sera dans la même configuration. On a donc :

Proposition 2.2. *Toute ITTM s'arrête ou boucle après un temps dénombrable, avec une longueur de boucle dénombrable.*

Remarque 2.2.1. Par cofinalité, on sait qu'une ITTM ne s'arrêtant pas avant ω_1 boucle ω_1 fois avant ω_1.

Puisque toutes les configurations qui apparaissent après ω_1 sont déjà apparues ω_1 fois avant ω_1, nous limitons notre étude des ITTM aux temps dénombrables.

Un réel x est dit *récursif* s'il existe une Machine de Turing (normale) qui, prenant en entrée n, donne (en temps fini) x_n. Un réel x est dit *écrivable* s'il existe une ITTM qui, prenant en entrée 0, s'arrête en donnant en sortie x.
Comme les Machines de Turing et les ITTM sont dénombrables, presque tous les réels sont non récursifs et non écrivables.
On utilise des réels pour *coder les ordres partiels* sur $\mathbb{N}$ en fixant le codage suivant : $x_{\langle i,j\rangle} = 1$ si et seulement si $i < j$. On note $i, j \mapsto \langle i, j\rangle$ une bijection [3] de $\mathbb{N}^2$ dans $\mathbb{N}$.
Ainsi, on dira qu'on code un ordinal dénombrable α dans un réel en y codant un (bon) ordre de type ordinal α.
Un ordinal α est dit *écrivable* s'il existe un réel écrivable codant un ordre partiel sur $\mathbb{N}$ de type α. On peut de même définir un ordinal *récursif* si le réel est récursif. On note λ_∞ le sup des écrivables, et ω_1^{CK} est le sup des récursifs, appelé ordinal de Church-Kleene.

Un ordinal β est dit *α-récursif* s'il existe une machine de Turing (normale) qui pour toute entrée x, n, x codant un ordre de type α et $n \in \mathbb{N}$, donne

3. Il faut qu'elle soit calculable, avec ses deux réciproques calculables pour qu'on puisse trouver en temps fini à quel n correspond le couple (i, j), et inversement.

en sortie en temps fini y_n avec y codant β. On peut de même définir α-*écrivable.* [4]
Un ordinal α est dit *horlogeable* s'il correspond au temps de calcul d'une ITTM s'arrêtant sur l'entrée vide (autrement dit sur 0) [5]. Le sup des horlogeables est appelé γ_∞.

Proposition 2.3. *$\omega_x^{\mathtt{CK}}, \forall x \in \mathbb{R}$, γ_∞ et λ_∞ sont dénombrables.*

Démonstration. Pour $\omega_x^{\mathtt{CK}}$, les machines de Turing sont dénombrables, donc l'ensemble des sorties possibles sur entrée x sont dénombrables, donc l'ensemble des x-récursifs est dénombrable. Or un ordinal récursif est dénombrable par définition, donc $\omega_x^{\mathtt{CK}}$ est un sup dénombrable de dénombrables, donc $\omega_1^{\mathtt{CK}}$ dénombrable.
Pour γ_∞ et λ_∞, il suffit de remarquer que les ITTM sont dénombrables, donc les ITTM qui s'arrêtent sont dénombrables. Ainsi, il n'existe qu'un nombre dénombrable de temps d'arrêt et de sortie différents. On a donc que γ_∞ et λ_∞ sont des sup dénombrables d'ordinaux dénombrables, donc γ_∞ et λ_∞ sont dénombrables. □

2.4 Ordinaux admissibles

Définition 2.2 (Constructibles de Gödel, voir [Bar90])**.** On définit L_α pour α un ordinal par :

$$L_0 = \emptyset$$
$$L_{\alpha+1} = Def(L_\alpha)$$
$$\text{Pour } \beta \text{ limite, } L_\beta = \bigcup_{\alpha<\beta} L_\alpha$$

Avec $Def(X)$ l'ensemble des sous-parties de X définies par une formule du premier ordre.

Définition 2.3 ([Bar90])**.** Un ensemble est dit *admissible* s'il est transitif s'il est un modèle des axiomes de Kripke-Platek, ie :

— Extensionnalité : deux ensembles sont égaux si et seulement s'ils ont les mêmes éléments.

4. On met bien une seule machine pour n'importe quel x codant α. En effet, dans un réel codant l'ordre ω, on peut parfaitement coder à l'intérieur n'importe quel réel : par exemple, pour coder x, on définit $2 \times n > 2 \times n + 1$ si et seulement si $x_n = 1$, et sinon on applique l'ordre canonique sur $\mathbb{N}$. Dans la littérature, on appelle α un oracle par analogie au calcul avec oracle sur Machine de Turing.[Pap93].

5. Le temps de calcul d'une ITTM ne tient pas compte de la dernière action passant la machine en état final. Ainsi, 0 ou les temps limites peuvent être horlogeables.

— Induction : pour toute formule du premier ordre, on a $[\forall x, \forall y \in x, \phi(y) \Rightarrow \phi(x)] \Rightarrow \forall x \phi(x)$.
— Paire : $\forall x, \forall y, \exists z$ dont les éléments sont exactement x et y.
— Ensemble vide : il existe un ensemble sans aucun élément.
— Σ_0-séparation : pour tout ensemble E et formule ϕ Σ_0, il existe un sous-ensemble de E des éléments de X tel que $x \in E \Leftrightarrow \phi(x)$.
— Σ_0-collection : Pour une formule Σ_0 $\phi(x, y)$ avec $\forall x, \exists y, \phi(x, y)$, on a $\forall X, \exists E, [e \in E \Leftrightarrow \exists x \in X, \phi(x, e)]$.

Définition 2.4. Un ordinal α est dit *admissible* si L_α est admissible.

Une autre propriété permet de voir un peu mieux ce que peut faire une ITTM jusqu'à un certain temps :

Proposition 2.4. *Pour β admissible, la sortie d'une ITTM sur entrée 0 s'arrêtant au temps $\alpha < \beta$ est contenue dans L_β.*

Démonstration. [CDLO17] nous propose cette idée de démonstration : par Σ_1-récursivité dans L_β, toute configuration d'une ITTM au bout d'un temps $< \beta$ est contenue dans L_β, donc sa sortie y est aussi. □

Remarque 2.4.1. Il existe un temps où la machine s'arrête : cette propriété s'exprime avec des quantificateurs finis, donc Σ_1, et on se base sur ça pour la démonstration. Cette idée fait partie du folklore des outils liés aux ITTM.

Proposition 2.5 (de Sacks, [Sac76])**.** *Les admissibles (dénombrables) sont exactement les $\omega_x^{\mathtt{CK}}$ pour x un réel, où on note $\omega_x^{\mathtt{CK}}$ le premier ordinal non récursif en x.*

La preuve de ce théorème est très originale : elle s'appuie notamment sur du *forcing* subtil, et on l'admet ici.

On peut aussi définir $\omega_\beta^{\mathtt{CK}}$ comme le premier ordinal non β-récursif. Cependant, la définition de β-récursif est un peu différente qu'x-récursif : en effet, si x ne peut coder qu'un seul type d'ordre, plusieurs réels peuvent coder le même ordinal.

Définition 2.5. On dit que α est β-récursif par le fait qu'il existe une machine de Turing qui sur entrée (y, n), y codant β sort x_n avec x codant α.[6]

Notons α^+ le premier admissible strictement supérieur à α.

6. Ici, x dépend du codage de β choisi.

Proposition 2.6. $\alpha^+ = \omega_\alpha^{\mathtt{CK}}$

Démonstration. D'abord, $\omega_\alpha^{\mathtt{CK}} \geq \alpha^+$, car $\omega_\alpha^{\mathtt{CK}}$ est admissible par la propriété de Sacks.
Supposons qu'il existe $\alpha < \beta < \omega_\alpha^{\mathtt{CK}}$ admissible. On a donc β récursif en α. Cela veut dire qu'il existe une machine de Turing qui, prenant en entrée x codant α et $n < \omega$ et sortant y codant β. On a donc une fonction surjective calculable de α vers β. Or cette bijection est en contradiction avec la propriété suivante, démontrée par [Bar90] : α est admissible ssi α est limite et il n'existe pas de $\gamma < \alpha$ tel qu'il existe une surjection $\Sigma_1(L_\alpha)$ de γ vers α. β n'est pas admissible, ce qui nous donne une contradiction. On a donc $\omega_\alpha^{\mathtt{CK}} = \alpha^+$. □

Proposition 2.7. *Un ordinal α dénombrable est admissible ou limite d'admissibles si et seulement si pour tout $\beta < \alpha$ il n'est pas β-récursif.*

Démonstration. Dans le cas où α est un admissible non limite d'admissibles, cela veut dire qu'il existe un $\beta < \alpha$ tel que $\alpha = \beta^+ = \omega_\beta^{\mathtt{CK}}$. Supposons qu'il existe $\gamma < \alpha$ tel que α soit γ-récursif : or $\gamma < \omega_\beta^{\mathtt{CK}}$ donc γ β-récursif (il n'y a pas de brèches dans les β-récursifs) donc on a α est β-récursif, ce qui nous donne la contradiction. On a donc bien α non β-récursif $\forall \beta < \alpha$.
Supposons maintenant que α est limite d'admissible, et qu'il existe $\beta < \alpha$ tel que α est β-récursif. Il existe donc un admissible $\beta < \gamma < \alpha$. Or α β-récursif implique que γ est β-récursif, ce qui est impossible. Donc si α est une limite d'admissibles, $\forall \beta < \alpha$, α non β-récursif.
Supposons maintenant que $\forall \beta < \alpha$, α non β-récursif. Prenons le sup des admissibles strictement inférieurs ou égaux à α, noté γ. Si il n'est pas égal à α, alors on a $\gamma < \alpha < \omega_\gamma^{\mathtt{CK}}$: la première inégalité vient de $\alpha \neq \gamma$ et que γ est un sup d'ordinaux inférieurs à α, donc $\gamma \leq \alpha$, et la seconde vient du fait que sinon on aurait eu $\gamma \geq \omega_\gamma^{\mathtt{CK}}$. On en déduit que α est γ-récursif, ce qui nous donne une contradiction. On a donc $\gamma = \alpha$, c'est-à-dire α admissible ou limite d'admissibles. □

3 Outils de calcul

3.1 Exécuter toutes les ITTM en parallèle

Lemme 3.1. Il existe une ITTM $\mathcal{U}$ simulant en parallèle toutes les ITTM sur entrée nulle, telle que pour tout ordinal limite α, les ITTM se trouvent (dans la simulation) dans leur configuration après un temps α (ou leur configuration finale si elles ont déjà terminé).

Démonstration. La bande d'entrée et de sortie de $\mathcal{U}$ est et reste nulle tout au long de l'exécution.
On sait que les ITTM sont énumérables. Notons-les donc $(\mathcal{M}_i)_{i\in\mathbb{N}}$. Simulons d'abord une seule ITTM $\mathcal{M}_i$.
Pour cela, il faut arriver à diviser en trois la bande de travail en une partie où sont stockés $\mathcal{Q}$ et δ, une partie où est stockée la configuration de la machine, et une partie servant à faire la simulation. On peut le faire en réservant les cases $3\times\mathbb{N}$ au codage de $\mathcal{Q}$ et δ, les $3\times\mathbb{N}+1$ à la configuration de la machine et les $3\times\mathbb{N}+2$ au traitement des deux premières parties. Sur les $3\times\mathbb{N}$, on alterne des espaces de 3 cases où on met (e_i, t_i, s_i), et un espace vide de taille fixe $1+k$. La première case de cet espace vide va servir de drapeau, qui va indiquer où se trouve la tête de lecture/écriture : on va faire en sorte que le premier drapeau allumé correspond à la position de la tête de lecture. Les cases suivantes ne vont être intéressantes qu'à la position de la tête de lecture, où elles vont contenir l'état courant.

Entrée	0	1	0	0	1	0	0	0	1	...
Travail	0	0	0	0	0	0	0	0	0	...
Sortie	0	0	0	0	0	0	0	0	0	...

On a codé les bandes (e, t, s) de $\mathcal{M}_i$, et ici la machine se trouve à l'état q et la tête de lecture/écriture est à la position 1.

e_0	t_0	s_0	○	q_0	e_1	t_1	s_1	●	q	e_2	t_2	s_2	...

FIGURE 4 – Codage de la configuration

Ainsi, exécuter une action consiste d'abord à parcourir les $3\times\mathbb{N}$ pour trouver le drapeau, puis lire le triplet (e, t, s) et le q correspondant, modifier (e, t, s) selon δ stocké dans les $3\times\mathbb{N}+1$ et enfin éteindre le drapeau, se décaler à gauche ou à droite selon δ et enfin allumer le drapeau et inscrire le nouvel état dans la nouvelle position. Remarquons ici que la valeur de la case $b_i, b \in \{e, t, s\}$ de la machine simulée correspond exactement à la valeur d'une case du ruban de travail de notre machine simulante : ainsi, quand on passe à la limite, les valeurs de ces cases seront exactement ce qu'elles sont censées être, c'est-à-dire la limsup. On n'a pas le problème que l'on aurait à la limite si, par exemple, 2 cases codaient 2 bits de la machine simulée pas par une copie parfaite de ces bits : en effet, si 01 code 00 et 10 code 01, et que la machine simulée est censée alterner entre 00 et 01, la limite

est censée être 01 alors que le code va alterner entre 01 et 10 et donner à la limite 11. Quand on arrive à la limite, on peut avoir beaucoup de drapeaux allumés : en effet, la tête de lecture peut être passée une infinité de fois sur une case, allumant une infinité de fois le drapeau. Quand on arrive à un état limite, on allume donc le premier drapeau et on place l'état limite à sa suite. Alors, on peut continuer la simulation : puisqu'on éteint le drapeau dès qu'on bouge la tête de lecture, le premier drapeau allumé correspondra toujours à la position de la tête de lecture, même si d'autres sont allumés après. Avec cette méthode, on peut simuler une machine. Remarquons ici qu'il nous faut ω étapes pour simuler ω actions sur $\mathcal{M}_i$, puisque chaque action prend un temps fini. Puisque tout temps limite α s'écrit $\omega.\beta$, la machine simulée pendant un temps α limite se trouve dans sa configuration au temps α (ou a déjà terminé).

Maintenant, on veut simuler toutes les ITTM en parallèle. Pour cela il faut résoudre deux problèmes : comment stocker toutes les ITTM avec leurs configurations, et comment les simuler en parallèle :

— Pour les stocker en parallèle, on va stocker $\mathcal{M}_0$ sur les pairs, puis $\mathcal{M}_1$ sur les cases $4\mathbb{N}+1$, et de la même manière caser la machine $\mathcal{M}_k$ sur les cases de numéro $2^{k+1}\mathbb{N}+2^k-1$.

$\mathcal{M}_0$	0	1	0	0	0	1	0	1	0	0	1	1	0	1	...
$\mathcal{M}_1$	0	0	1	1	1	0	0	1	0	1	1	1	0	0	...
$\mathcal{M}_2$	1	0	0	1	1	1	0	1	1	1	0	1	0	0	...
$\mathcal{M}_3$	0	0	0	1	1	1	0	1	0	1	0	0	0	1	...
$\mathcal{M}_4$	0	0	1	1	1	0	1	0	1	0	1	0	1	1	...

Se code en :

0	0	1	1	0	0	0	0	0	1	1	0	0	1	1	0	0	1	0	...

FIGURE 5 – Stockage de toutes les ITTM

Remarquons ici qu'à une case correspond une seule ITTM : on n'a donc pas de problème à la limite. Il faut enfin dire que les ITTM sont énumérables : il existe donc une ITTM capable d'écrire la i^e ITTM en temps fini.

— Pour les simuler en parallèle, on va simuler une fois $\mathcal{M}_0$, puis une fois $\mathcal{M}_0$ et deux fois $\mathcal{M}_1$, puis une fois les machines $\mathcal{M}_0$ à $\mathcal{M}_{k-1}$ et $k+1$

fois $\mathcal{M}_k$. Au bout d'un temps ω, on a donc simulé ω actions pour chaque ITTM. Enfin, remarquons que la création de la i^e ITTM peut se faire la première fois qu'on l'exécute. On en déduit qu'au bout d'un temps α limite, on a simulé α actions de toutes les ITTM.

L'algorithme est donc :

Algorithm 1 Machine $\mathcal{U}$

On remplit la bande de travail des ITTM, de leurs configurations initiales et des cases d'information sur chacune d'elles (se fait en parallèle des ω premières exécutions, par exemple en codant notre ITTM avant de l'exécuter pour la première fois).
Supposons que l'on soit au temps limite α :
for $k \in \mathbb{N}$ **do**
  for $l \in [0, k-1]$ **do**
    if $\mathcal{M}_l$ n'a pas déjà terminé **then**
      On exécute une fois $\mathcal{M}_l$
    end if
  end for
  if $\mathcal{M}_k$ a déjà terminé **then**
    Passer $\mathcal{M}_k$ dans son état limite, puis l'exécuter k fois
  end if
end for

□

Une fois qu'une ITTM est en état final, on n'a plus besoin de l'exécuter : on peut l'effacer (ce qui peut se faire en parallèle du reste, donc sans retarder l'exécution). Il est aussi possible de noter où s'arrête chacune des ITTM : ainsi, si on inscrit un réel x codant α sur les ITTM s'arrêtant en 0 et un réel y codant β sur les ITTM s'arrêtant en 1, l'ordre sur les ITTM s'arrêtant en temps 0 ou 1 est $\alpha + \beta$. Enfin, on peut utiliser cette machine pour savoir, avec un temps de retard de ω, si on est dans une brèche : il suffit d'allumer et d'éteindre un drapeau à chaque fois qu'une machine s'arrête : à un temps limite, il ne sera éteint que si on est dans ou à la fin d'une brèche.

3.2 Ordre partiel et total

Proposition 3.1. *Il existe une ITTM qui sur entrée un réel codant un ordre partiel donne en temps ω un ordre total de même type ordinal.*

Démonstration. Dans la suite, nous ne sortirons que des ordres partiels sur $\mathbb{N}$. Cependant, on peut en temps ω transformer un ordre partiel écrit sur la bande d'entrée en un ordre total, écrit sur la bande de sortie. D'abord, rappelons qu'un ordre est codé par un réel, tel que un 1 dans la case $\langle i, j\rangle$ correspond à $i \leq j$ et que un 0 peut vouloir dire que i ou j ne sont pas dans l'ordre ou que $i > j$.
Nous allons donc utiliser une machine qui en entrée prend un réel codant un ordre partiel et qui donne en sortie un réel codant un ordre total de même type ordinal. Nous construisons cet ordre total en construisant des ordres totaux successifs sur $[0, k]$ en rajoutant les relations d'ordre dès qu'on les trouve.
On utilise la bande de travail pour y mettre les entiers de l'ordre partiel qu'on a déjà mis dans l'ordre total. Par exemple, on peut dire que l'on sépare des suites de 1 par un 0, et la k^e suite contient n 1 avec n le k^e entier de l'ordre partiel que l'on aura trouvé. On parcourt la bande d'entrée, supposons que la m^e case contient un 1 et que l'on récupère les $i_m \leq j_m$ qu'ils représentent. On peut savoir en parcourant la bande de travail (jusqu'à la première fois que l'on rencontre un 00) si i_m et j_m sont dans la bande de travail. Il suffit donc de rajouter si besoin i_m et j_m à l'ordre en les codant par le prochain entier disponible, puis de noter leur ordre sur la sortie.
On a construit une bijection conservant l'ordre entre le support de l'ordre partiel et $\mathbb{N}$. L'ordre de sortie est donc bien un ordre total du même type que l'ordre d'entrée. □

3.3 Calculabilité et produit

Supposons que α et β récursifs, ie $\mathcal{M}_\alpha, \mathcal{M}_\beta$ deux machines de Turing telles que $\forall n \in \mathbb{N}, M(n)$ donne le n^e bit de α et β. Notre but est de créer une machine $\mathcal{M}_{\alpha.\beta}$ donnant le n^e bit de $\alpha.\beta$. Nous pouvons voir aisément qu'il est équivalent de travailler une machine qui prenant en entrée $(n, m) \in \mathbb{N}^2$ nous dit si $n \leq m$ selon l'ordre-type de $\alpha.\beta$. Supposons qu'en entrée, on dispose d'un ordre total sur $\mathbb{N}$ de type α et β.

En fait, il nous suffit de décrire un ordre total sur $\mathbb{N}^2$ de type $\alpha.\beta$, puis de le passer dans $\mathbb{N}$.
On va donc devoir comparer (i, j) et (k, l) :

- si $i \neq j$, alors on définit l'ordre entre (i, j) et (k, l) par l'ordre entre i et k selon β.
- sinon, on définit l'ordre entre (i, j) et (k, l) par l'ordre entre j et l selon α.

Si l'on regarde bien, les (i, j) à i fixé sont triés selon l'ordre $\alpha.\beta$. Alors, il suffit juste d'utiliser une bijection calculable de $\mathbb{N}^2$ dans $\mathbb{N}$ comme $(i, j) \to \frac{(i+j)(i+j+1)}{2} + i$ pour avoir un ordre total sur $\mathbb{N}$ de type $\alpha.\beta$.

De manière similaire, on pourrait calculer $\alpha.\beta.\gamma$ ou encore α^β.

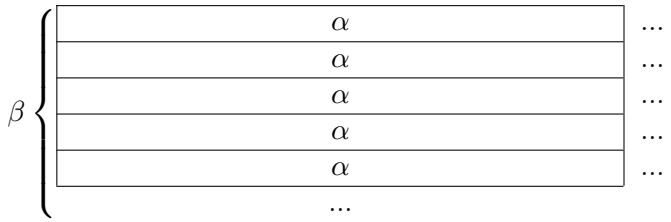

FIGURE 6 – Écriture de $\alpha.\beta$: plus on va bas puis à droite, plus on est haut dans l'ordre

3.4 Parallélisation avec une machine de Turing

Imaginons que l'on possède une ITTM $\mathcal{M}$ donnant en sortie un réel x en temps α limite. On veut écrire un autre réel y récursif en x. On peut l'écrire en temps $\alpha+\omega$: puisqu'il existe une machine de Turing (normale) prenant x en entrée et $n \in \mathbb{N}$ et donnant en sortie y_n, il suffit de l'exécuter pour tout n, ce qui va nous prendre un temps ω. Cependant, il est possible dans certains cas de l'écrire en temps α.

Proposition 3.2. *Si $\forall n \in \mathbb{N}, \exists \beta_n < \alpha$ tel que s_n (s la bande de sortie) n'est plus modifiée après β_n, alors on peut écrire y en temps α.*

Démonstration. Notons $\mathcal{MT}$ la machine de Turing (normale) obtenant y_n à partir de la sortie de $\mathcal{M}$.
On va simuler $\mathcal{M}$ sur la première partie. Quand on simule la $\sigma + n^e$ action de $\mathcal{M}$ avec σ limite, on exécute $\mathcal{MT}$ sur entrée $0..n$ en mettant les $y_0...y_n$ obtenus sur la bande de sortie. Ainsi, $\forall \beta < \alpha$, la n^e case est mise à jour au moins une fois par $\mathcal{MT}$ (et même une infinité de fois).

Or on a supposé que $\forall n \in \mathbb{N}, \exists \beta_n < \alpha$ tel que s_n (s la bande de sortie de $\mathcal{M}$ n'est plus modifiée après β_n. Si on donne en entrée de $\mathcal{MT}$ la sortie de $\mathcal{M}$, elle ne va avoir besoin que d'un temps fini pour terminer, elle ne va

donc visiter qu'un nombre k fini de cases. Ainsi, ce nombre fini de cases va stabiliser un temps infini avant α (car α limite et $\forall n \in \mathbb{N}, \beta_n < \alpha$). On a donc qu'après le sup de ces k β_n (qui est donc $< \alpha$), la sortie va être mise à jour par $\mathcal{MT}$ dans le bon résultat et ne plus jamais changer avant α. Ainsi, au temps α, la sortie va bien être (y_n). □

Par exemple, si on a une machine écrivant un ordre partiel, on pourra souvent sortir un ordre total du même type.

4 Temps d'écriture de tout ordinal

4.1 Propriétés des ordinaux récursifs, écrivables et horlogeables

Proposition 4.1. *$\forall \alpha < \omega_1^{\mathtt{CK}}$, α est récursif, et $\forall \alpha < \lambda_\infty$, α est écrivable.*

Démonstration. Soit β récursif codé par x, et $\alpha < \beta$. Cela veut dire que il existe un entier k tel que le sous-ordre dans x sur les entiers strictement inférieurs à k est de type α. Il suffit donc pour l'entrée n de renvoyer 1 si la case n correspond à un couple (i, j) tels que $(i < j) \wedge (i < k) \wedge (j < k)$, et 0 sinon (cela se fait en temps fini). On a dont une machine de Turing qui pour n donne y_n avec y codant α, d'où α récursif, donc $\forall \alpha < \omega_1^{\mathtt{CK}}$, α est récursif. □

Soit β écrivable codé par x, et $\alpha < \beta$. Ici aussi, il existe un entier k tel que α est codé par le sous ordre dans x sur les entiers strictement inférieurs à k. Il suffit donc d'écrire β sur la bande de sortie, puis en temps ω de visiter toutes les cases et à la n^e case correspondant à (i, j) la passer à 0 si $i > j \vee i \geq k \vee j \geq k$. On a donc si α écrivable, d'où $\forall \alpha < \lambda_\infty$, α écrivable. □

Dans le cas récursif, on peut démontrer exactement de la même manière qu'il n'y a pas de brèche dans les $\omega_x^{\mathtt{CK}}$-récursif, pour x un réel, ou dans les écrivables sur entrée x.

Proposition 4.2 (voir [HL00])**.** *α récursif $\Rightarrow$ α horlogeable. α horlogeable $\Rightarrow$ α écrivable.*

Remarque 4.2.1. Il n'y a donc pas de brèches dans les récursifs ou les horlogeables : $\alpha < \omega_1^{\mathtt{CK}}$ (resp. λ_∞) est équivalent à α récursif (resp. écrivable). On a donc que $\omega_1^{\mathtt{CK}}$ est l'ensemble des récursifs, et λ_∞ l'ensemble des écrivables.

Proposition 4.3 ([Wel09])**.** $\lambda_\infty = \gamma_\infty$

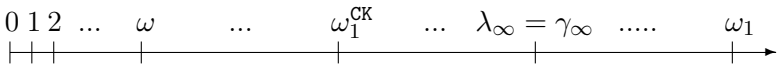

FIGURE 7 – Comparaison entre les sups des horlogeables, écrivables, récursifs

Remarque 4.3.1. Si leurs sups sont égaux, cela ne veut pas dire que les notions d'horlogeabilité et d'écrivabilité soient équivalentes. En effet, il existe des ordinaux inférieurs à γ_∞ non horlogeables.

4.2 Horlogeabilité et admissibilité

Proposition 4.4 ([HL00])**.** *α admissible $\Rightarrow$ α non horlogeable.*

Remarque 4.4.1. Cela nous montre qu'il existe des brèches dans les horlogeables, c'est-à-dire des intervalles de temps ordinaux non horlogeables, la première commençant en ω_1^{CK}.

Proposition 4.5 (Lemme Speed-Up [HL00])**.** *Si $\alpha + n$ est horlogeable, avec $n < \omega$, alors α est horlogeable*

Démonstration. On va prouver cette propriété pour α limite, puisque si β est horlogeable, alors $\forall n < \omega$, $\beta + n$ est horlogeable. Soit $\mathcal{M}$ une ITTM qui sur entrée 0 s'arrête au temps $\alpha + n$. Au temps α, les n premières cases des bandes sont dans une certaine configuration $\mathcal{C}$ qui à partir de l'état limite et au bout d'un temps n va aller en état final. Puisque le calcul a continué, on en déduit qu'elles n'ont jamais été dans cette configuration en temps limite. On veut donc savoir dès le temps α si les n premières cases sont dans cette configuration ou pas. Pour cela, on va décaler la bande de travail et de sortie d'un cran vers la droite pour se réserver la première case. On va simuler $\mathcal{M}$ sur ces cases $\mathbb{N} + 1$, et mettre des drapeaux sur les deux cases libérées.
À chaque fois que l'on exécute une action de la machine simulée, on regarde les n premières cases et on les compare à la bande d'entrée : si il y a une case contenant un 1 alors que $\mathcal{C}$ y contient un 0, alors on allume le premier drapeau, et si une case contient un 1 alors que $\mathcal{C}$ y contient un 1, on allume le deuxième drapeau. Ainsi, à un temps limite, le premier drapeau est allumé si et seulement s'il existe une case contenant un 1 où $\mathcal{C}$ contient un 0, et le deuxième est allumé si et seulement si toutes les cases contenant des cases à 1 dans $\mathcal{C}$ contiennent un 1 dans la simulation. Ainsi, le premier drapeau est éteint et le deuxième allumé si et seulement si les n premières cases de la simulation correspondent à $\mathcal{C}$. Or la tête de lecture se situe au-dessus de ces deux drapeaux aux temps limite : on peut donc s'arrêter dès que le premier

drapeau est éteint, le deuxième allumé et qu'on est dans l'état limite, et ça arrive au temps α. □

Nous allons utiliser cette propriété pour adapter $\mathcal{U}$ en $\mathcal{U}'$ afin de l'arrêter au temps $\omega_1^{\mathtt{CK}} + \omega$.

Réservons la première case de la bande de travail pour y mettre un drapeau global. Quand une machine arrive dans un état final, on fait passer sa valeur à 1 puis on le repasse à 0. Quand $\mathcal{U}$ est dans son état limite au temps α limite, on regarde le drapeau : s'il a une valeur de 1, alors il existe des ITTM s'étant arrêtées indéfiniment avant α, et on continue l'exécution de $\mathcal{U}$ après avoir passé le drapeau à 0.

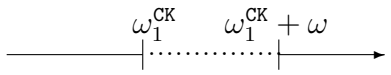

FIGURE 8 – Brèche entre $\omega_1^{\mathtt{CK}}$ et $\omega_1^{\mathtt{CK}} + \omega$

Or, le premier ordinal non horlogeable est $\omega_1^{\mathtt{CK}}$, donc les ordinaux de $[\omega_1^{\mathtt{CK}}, \omega_1^{\mathtt{CK}} + \omega)$ sont tous non horlogeables : on aura donc le drapeau à la valeur 0 au temps $\omega_1^{\mathtt{CK}} + \omega$ et jamais avant, puisque $\omega_1^{\mathtt{CK}}$ est le premier ordinal non horlogeable. Donc $\mathcal{U}$ s'arrête en $\omega_1^{\mathtt{CK}} + \omega$.

4.3 Temps d'écriture de $\omega_1^{\mathtt{CK}}$

On sait déjà s'arrêter au bout d'un temps $\omega_1^{\mathtt{CK}} + \omega$. On va pouvoir adapter la machine $\mathcal{U}$ pour écrire $\omega_1^{\mathtt{CK}}$ sur la bande de sortie, en temps $\omega_1^{\mathtt{CK}} + \omega$.

L'idée naïve serait ici d'écrire l'ordre canonique sur ω puis de le faire évoluer : par exemple, pour écrire $\omega + \omega$, on fait passer l'ordre précédent sur les pairs, puis on définit tous les impairs comme étant supérieurs aux pairs, et enfin on donne l'ordre naturel aux impairs. De même, on peut écrire $\omega.3$, voire tous les $\omega.k, k < \omega$. Cependant, cette méthode n'est pas utilisable : en effet, quand on arrivera à ω^2, les cases correspondant aux nombres impairs auront alternés indéfiniment entre 0 et 1, ce qui va nous faire perdre l'ordre en mettant toutes ces cases à 1. Ici, notre façon de faire évoluer notre ordre n'est pas assez "stable", dans le sens où une case du codage de cet ordre n'est pas stationnaire. L'idéal serait de réussir à ce que le code ne bouge plus dès qu'on l'a codé. C'est possible en introduisant un ordre sur les ITTM.

Nous allons trier les ITTM par ordre de temps d'arrêt : c'est un semi-ordre partiel, puisque certaines ITTM ne s'arrêtent jamais, et que certaines ITTM s'arrêtent au même temps. Pour obtenir un (bon) ordre partiel, il faut donc sélectionner une ITTM par temps d'arrêt.

À chaque fois que nous trouverons une ITTM $\mathcal{M}_i$ s'arrêtant en un temps $\alpha + n$, nous vérifierons s'il existe une ITTM $\mathcal{M}_k, k < i$ s'y étant arrêtée. Si oui, nous pourrons l'ignorer pour le reste de l'exécution de $\mathcal{U}$. Si non, nous pourrons ajouter cette ITTM à l'ordre inscrit en sortie. Notons bien que cet ordre va dépendre de l'ordre dans lequel on énumère les ITTM, et il faut faire attention à ce que $\forall k, l \in \mathbb{N}$ avec $k < l$, alors pour un certain temps ordinal on a simulé la k^e ITTM avant la l^e.

Cependant, on a ici un problème : dès lors qu'on a dépassé le temps ω, on a un nombre ω d'ITTM dans l'ordre. Rajouter une ITTM dans cet ordre prend donc un temps ω.
Il va donc falloir faire ces mises à jour de l'ordre en parallèle des autres tâches, en lançant des mises à jour à deux moments : quand une machine finit et qu'aucune machine n'a fini au même temps, ou alors quand on passe sur une machine qui a déjà terminé.
On a donc besoin de savoir à quel temps $\alpha + n$ avec α limite l'ITTM a terminé. Pour cela, il faut savoir si la machine a terminé il y a un temps infini (ie $< \alpha$) ou fini : il suffit d'activer un drapeau quand une machine termine et qu'on l'intègre dans l'ordre puis le désactiver quand on y repasse. Il nous faut aussi noter n, afin de pouvoir mettre à jour l'ordre.

Nous avons ici tous les éléments pour écrire $\omega_1^{\mathtt{CK}}$ en ω étapes, par la machine exécutant cet algorithme :

Algorithm 2 Machine $\mathcal{U}$

Ensure: $\omega_1^{\mathtt{CK}}$ en temps $\omega_1^{\mathtt{CK}} + \omega$

On remplit la bande de travail des ITTM, de leurs configurations initiales et des cases d'information sur chacune d'elles (se fait en parallèle des ω premières exécutions).
On initialise le drapeau global à 1.
while drapeau global à 1 **do**
Supposons que l'on soit au temps limite α :
Passer le drapeau global à 0
for $k \in \mathbb{N}$ **do**
for $l \in [0, k-1]$ **do**
if $\mathcal{M}_l$ n'a pas déjà terminé **then**
On exécute une fois $\mathcal{M}_l$
if Cette exécution termine $\mathcal{M}_l$ et que personne ne s'est arrêté en $\alpha + k$ **then**
Mettre à jour l'ordre entre l et $j, \forall j < l$ grâce aux informations sur les temps d'arrêt des $\mathcal{M}_j$, et noter son temps d'arrêt passer son drapeau local à 1 et passer le drapeau global à 1.
end if
end if
end for
if $\mathcal{M}_k$ a déjà terminé **then**
Passer son drapeau local à 0
for $l \in [0, l-1]$ **do**
if $\mathcal{M}_l$ a terminé depuis un temps fini et est dans l'ordre **then**
Noter $k \leq l$
end if
end for
else
Passer $\mathcal{M}_k$ dans son état limite, puis l'exécuter k fois
if Une machine $\mathcal{M}_k$ s'est arrêtée au temps $\alpha + n, n \in \mathbb{N}$, et que les $\mathcal{M}_l, l < k$ ne s'y sont pas arrêtés **then**
Noter où la machine s'est arrêtée sur la ligne et mettre à jour l'ordre avec les machines parmi les $\mathcal{M}_l, l < k$ qui font déjà partie de l'ordre, grâce aux informations sur le temps d'arrêt des $\mathcal{M}_l$, et faire passer le drapeau local de $\mathcal{M}_k$ et le drapeau global à 1.
end if
end if
end for
end while

Nous avons vu plus haut qu'il nous faut un temps $\omega_1^{\mathtt{CK}} + \omega$ pour écrire $\omega_1^{\mathtt{CK}}$. Il est donc légitime de se demander quel temps il faut pour écrire des ordinaux plus grands.

4.4 Temps d'écriture au-dessus de $\omega_1^{\mathtt{CK}}$

Non seulement on peut écrire $\omega_1^{\mathtt{CK}}$ en temps $\omega_1^{\mathtt{CK}} + \omega$, mais dans le même temps on peut aussi écrire $\omega_1^{\mathtt{CK}}.2$.

Proposition 4.6. *Le temps d'écriture de $\omega_1^{\mathtt{CK}}.2$ est exactement $\omega_1^{\mathtt{CK}} + \omega$.*

Démonstration. Pour cela, reprenons la machine précédente, et remarquons que quand on écrit $i < j$ avec elle, sa valeur ne changera jamais : en effet, à chaque fois que l'on visitera à nouveau ces deux ITTM, on laissera le 1 en place. Aussi, si on écrit un ordre de type α sur $\mathbb{N}$, il est facile d'écrire un ordre de type $\alpha.2$: il suffit de dire que quand pour le premier ordre on a $i < j$, pour le deuxième ordre on pose $2i < 2j$, $2i + 1 < 2j + 1$, $2i < 2i + 1$ et $2j < 2j + 1$. Ainsi, on a posé les pairs inférieurs aux impairs et on met l'ordre α sur les pairs et les impairs : ainsi l'ordre obtenu est de type $\alpha.2$. □

On peut aussi écrire $\alpha.3$, en prenant les cases $3\mathbb{N}$, $3\mathbb{N}+1$ et $3\mathbb{N}+2$, ou même n'importe quel $\alpha.k$. En fait, on peut même écrire n'importe quel ordinal récursif en $\omega_1^{\mathtt{CK}}$ en temps $\omega_1^{\mathtt{CK}} + \omega$.

Proposition 4.7. *Si $\alpha > \omega_1^{\mathtt{CK}}$ est $\omega_1^{\mathtt{CK}}$-récursif, alors son temps d'écriture est exactement $\omega_1^{\mathtt{CK}} + \omega$.*

Démonstration. Il suffit d'utiliser la sous-section 3.4 : on a une ITTM sortant $\omega_1^{\mathtt{CK}}$ de manière stable (puisque la seule chose que fait l'ITTM de la sous-section 4.3 est d'ajouter des 1 à la sortie) et qu'on veut écrire un récursif en $\omega_1^{\mathtt{CK}}$, il suffit d'écrire □

4.5 Longueur de brèche

Pour l'instant, on ne connaît que des brèches de longueur ω, et on n'a pas parlé de beaucoup d'autres admissibles que $\omega_1^{\mathtt{CK}}$.

Proposition 4.8 ([Wel09])**.** *Tous les débuts de brèche sont admissibles.*

Démonstration. On suppose γ début de brèche et non admissible : il est donc limite d'horlogeables et $\forall n < \omega, \exists \alpha < \gamma, \psi(n, \alpha, \xi)$, telle que cette fonction ψ définit une fonction $F : \omega \to \gamma$ croissante telle que l'union des $F(n)$ soit exactement γ. Alors, on peut l'utiliser pour s'arrêter en γ. Pour le détail de la preuve, voir [Wel09]. □

Proposition 4.9. *Soit α horlogeable. Alors α^+ est le début d'une brèche de longueur ω.*

Démonstration. Il existe une ITTM qui sur entrée 0 termine en temps α : lançons-la, puis lançons la machine de la sous-section 4.2 qui va chercher le prochain admissible pour l'écrire. Ici, le prochain admissible est exactement le premier non horlogeable après α, puisque tout récursif en α est horlogeable : comme pour la sous-section 4.2, nous pouvons donc nous arrêter au temps $\alpha^+ + \omega$. Ainsi, la prochaine brèche après α horlogeable est de longueur ω. □

On peut donc toujours trouver des brèches de longueur ω jusqu'à γ_∞. Cependant, nous pouvons aussi trouver des brèches de longueur plus grande qu'ω. En fait, on peut même avoir des brèches tellement grandes qu'elles contiennent le prochain admissible.

Théorème 4.1 ([CDLO17])**.** *Il existe des brèches qui contiennent strictement (pas uniquement en leur début) des admissibles.*

Les auteurs proposent plusieurs preuves, et en voici une qui prouve une version un peu plus forte (déjà connue) dans la lignée de la construction de notre rapport.

Proposition 4.10. *La première brèche contenant strictement un admissible s'arrête au temps ω après cet admissible.*

Démonstration. Quand on arrive dans une brèche commençant par α, en reprenant la machine de la sous-section 4.4, on dispose de l'écriture d'un codage de α. On lance alors le calcul de α^+ comme pour $\omega_1^{\mathtt{CK}}$ mais sur l'entrée α : si on arrive à le terminer avant de sortir de la brèche alors on s'arrête, sinon on réinitialise sur entrée α $\mathcal{U}_{\mathtt{calc}}$ et on relance le calcul. Plus précisément, on attend que la machine $\mathcal{U}_{\mathtt{calc}}$ s'arrête : quand elle s'arrête, c'est qu'on a réussi à écrire α^+ au plus quand on sort de la brèche : en effet, il suffit de dire que $\mathcal{U}_{\mathtt{calc}}$ s'arrête au temps $\alpha^+ + \omega$, où on va donc arrêter la machine. Remarquons qu'on lance $\mathcal{U}_{\mathtt{calc}}$ en $\alpha + \omega$, et qu'il faut un temps $\alpha^+ + \omega$ pour le trouver, on s'arrête donc en $\alpha + \omega + \alpha^+ + \omega = \alpha^+ + \omega$ car $\alpha^+ \geq (\alpha + \omega).\omega$.
L'existence est donnée par ce même programme, car si cette brèche n'existait pas, il s'arrêterait en temps $\gamma_\infty + \omega$ ce qui n'est pas possible puisque γ_∞ est le sup des horlogeables. □

Les brèches peuvent être très grandes, comme le montre cette propriété :

Proposition 4.11. *Pour tout horlogeable, il existe une brèche de longueur supérieure à lui.*

Démonstration. Posons $\mathcal{M}$ une ITTM s'arrêtant en temps α horlogeable. Dans la suite on va le supposer limite (quitte à remplacer α par $\alpha + \omega$ ce qui revient au même). Nous allons chercher les brèches de taille supérieure ou égale à α, en exécutant $\mathcal{M}$ en parallèle de $\mathcal{U}$: dès qu'on détecte que l'on est dans ue brèche, on lance $\mathcal{M}$, et si l'on voit que $\mathcal{M}$ s'arrête alors on s'arrête. Supposons qu'il n'existe pas de telles brèches : alors arrivé à γ_∞ le programme va détecter qu'on rentre dans une brèche en $\gamma + \omega$, mais jamais qu'on en sorte. On va donc arriver en $\gamma_\infty + \omega + \alpha$ et s'arrêter, ce qui est impossible car γ_∞ est le sup des horlogeables. On obtient donc l'existence d'une brèche de taille au moins α. □

En fait, Hamkins et Lewis ont prouvé plus : les brèches sont aussi grandes qu'on veut, et aussi nombreuses que l'on veut :

Proposition 4.12 ([HL00])**.** *Pour α écrivable, il existe au moins α brèches de taille au moins α.*

On adapte la preuve précédente en écrivant α, puis en comptant α brèches de taille au moins α.

Ils se sont aussi intéressés au nombre de brèches de taille plus grande qu'un écrivable :

Proposition 4.13 ([HL00])**.** *Pour α écrivable, le type ordinal de brèches de taille supérieure ou égale à α n'est pas écrivable.*

Démonstration. Supposons que le nombre β de brèches de taille supérieure ou égale à α est écrivable.
Ècrivons-le donc, puis, exécutons une machine qui, à chaque fois qu'elle trouve une brèche de longueur au moins α retire le plus petit élément de l'ordre. Quand cet ordre est vide, il n'existe plus de brèche de longueur plus grande qu'α. La prochaine fois que l'on va trouver α ordinaux non horlogeables à la suite sera donc après γ_∞ : on arrête la machine dès qu'on en compte α puis on s'arrête, ce qui est impossible car γ_∞ est le sup des horlogeables. On a donc que β non écrivable. □

On peut améliorer ceci et obtenir des tailles de brèches plus exactes.

Proposition 4.14. *Pour α un ordinal limite écrivable, le type ordinal des brèches de taille exactement α est γ_∞.*

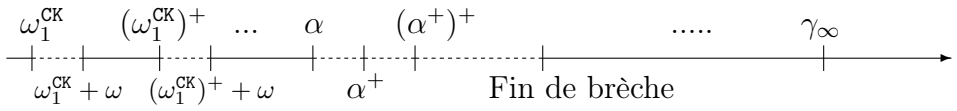

FIGURE 9 – Brèches dans les ordinaux horlogeables

Démonstration. D'abord on écrit α. Puis on recherche les brèches comme dans la preuve précédente et dès qu'on a trouvé une brèche de taille au moins α on s'arrête, ce qui fait que la taille est exactement α. □

On peut tirer de cette construction un corollaire intéressant sur la structure des brèches.

Corollaire 4.14.1. Pour α un ordinal limite écrivable, au dessus du temps d'écriture de α, les premières apparitions de brèches de taille β limite, $\beta < \alpha$ sont croissantes.

Démonstration. Il faut utiliser ici que le temps d'écriture de $\beta < \alpha$ est inférieur ou égal au temps d'écriture de α : voir la sous-section 4.6. □

4.6 Temps d'écriture de tout ordinal

On peut se demander s'il existe une machine écrivant ω_1^{CK}, $\omega_1^{CK}.2$ ou n'importe quel récursif en ω_1^{CK} plus vite que $\mathcal{U}$. La réponse est négative : non seulement on ne peut pas l'écrire plus vite qu'en temps $\omega_1^{CK} + \omega$, mais on peut avoir besoin d'un temps affreusement plus grand qu'α pour écrire α.

Théorème 4.2. *Un ordinal écrivable non récursif s'écrit en temps la borne supérieure des fins de brèche commençant avant lui (avant étant compris au sens large). S'il est récursif, alors ce temps est ω.*

Remarque 4.2.1. Par exemple, si β est dans une brèche, alors son temps d'écriture est la fin de cette brèche.

Démonstration. On va écrire $\alpha < \lambda$.
Supposons d'abord que α est une limite d'admissible horlogeable.

Remarque 4.2.2. Soient α et β tels que $\alpha.\omega \leq \beta$, alors on a que $\alpha + \beta = \beta$.

Démonstration. Posons $\beta = \nu + n$ avec ν limite et n fini. On a alors $\alpha.\omega \leq \beta \Rightarrow \alpha.\omega \leq \nu$. De plus, $\alpha + \nu = \bigcup_{\gamma<\nu} (\alpha + \gamma)$. Or $\forall \gamma < \nu, \alpha + \gamma + n < \beta$, donc $\alpha + \beta \leq \beta$, et $\alpha + \beta \geq \beta$, d'où $\alpha + \beta = \beta$. □

On a donc :

Remarque 4.2.3. Soit $(\delta_n)_{n\in\mathbb{N}}$ une suite croissante telle que $\forall n \in \mathbb{N}, \delta_n.\omega < \delta_{n+1}$, alors la limite de $\sum_{n=0}^{N} \delta_n$ est égale au sup des δ_n.

Nous allons exécuter trois ITTM en parallèle :

- $\mathcal{U}_{\mathtt{g}}$ qui va reprendre la machine de la sous-section 3.1 pour savoir si l'on se trouve dans ou à la fin d'une brèche, mais aussi à définir un horlogeable où s'arrêter.
- $\mathcal{U}_{\mathtt{calc}}$ qui va nous servir à calculer les admissibles successifs,
- $\mathcal{U}_{\mathtt{stock}}$ qui va nous servir à stocker les admissibles successifs.

Posons β le nombre d'admissibles inférieurs ou égaux à α. $\mathcal{U}_{\mathtt{stock}}$ va exécuter la machine de la sous-section 4.3, à part que l'on ne va pas s'arrêter au $\omega_1^{\mathtt{CK}}$ ième horlogeable (ie en $\omega_1^{\mathtt{CK}} + \omega$), mais au β^e horlogeable. En plus de construire l'ordre d'arrêt des ITTM, $\mathcal{U}_{\mathtt{stock}}$ va noter où chaque ITTM s'arrête, par exemple en notant quand une ITTM s'arrête le numéro de l'ITTM s'arrêtant au même temps et dans l'ordre. On va écrire l'ordre du σ^e admissible sur les ITTM s'arrêtant au σ^e horlogeable. Par la dernière remarque, on a que l'ordre sur les ITTM s'étant arrêtées jusqu'au β^e horlogeable contient le β^e admissible. Remarquons qu'il suffit qu'à chaque fois qu'on trouve un admissible, on calcule un pas de toutes les ITTM pour trouver toutes les ITTM s'arrêtant au prochain horlogeable disponible, et cela se fait en temps ω.

$\mathcal{U}_{\mathtt{calc}}$ prendra en entrée le dernier admissible trouvé γ et cherchera le prochain admissible. Pour cela, on reprend le principe de l'ITTM de la sous-section 4.2, qui sur entrée 0, cherche $\omega_1^{\mathtt{CK}} = 0^+$, et qui sur entrée γ va en fait chercher $\omega_\beta^{\mathtt{CK}} = \beta^+$, ce qui prend un temps β^+. Dès qu'on trouve un admissible, on le rajoute à $\mathcal{U}_{\mathtt{stock}}$.

La structure de notre algorithme est donc la suivante :

- On lance $\mathcal{U}_{\mathtt{g}}$, $\mathcal{U}_{\mathtt{calc}}$ et $\mathcal{U}_{\mathtt{stock}}$ sur entrée 0.
- Quand on trouve le prochain admissible (en temps ω après, on l'inscrit sur le prochain horlogeable disponible de $\mathcal{U}_{\mathtt{stock}}$, on réinitialise $\mathcal{U}_{\mathtt{calc}}$ sur entrée cet admissible (se fait en temps ω).
- Quand $\mathcal{U}_{\mathtt{g}}$ nous dit de s'arrêter, on renvoie la sortie de $\mathcal{U}_{\mathtt{stock}}$.

On inscrit donc un admissible en temps $\omega.2$ après l'avoir atteint : un temps ω pour reconnaître qu'on y est arrivé, et un temps ω pour le marquer. On peut même supprimer ce dernier temps ω : pour cela, il suffit qu'en parallèle de l'exécution de $\mathcal{U}_{\mathtt{calc}}$, on fasse en sorte que sa bande de sortie soit toujours la même que l'ordre sur le prochain horlogeable disponible de $\mathcal{U}_{\mathtt{stock}}$ (par exemple, à chaque fois que $\mathcal{U}_{\mathtt{calc}}$ modifie sa sortie, on modifie de même

l'ordre de $\mathcal{U}_{\texttt{stock}}$). Ainsi, quand $\mathcal{U}_{\texttt{calc}}$ nous indique qu'on a trouvé notre prochain admissible, il est déjà marqué sur la bande de sortie de $\mathcal{U}_{\texttt{stock}}$.

On exécute donc $\mathcal{U}_{\texttt{calc}}$ pour trouver les admissibles dont α est le sup. La bande de sortie de $\mathcal{U}_{\texttt{stock}}$ contient leur somme, qui est donc exactement leur sup, ie α. On a donc que quand $\mathcal{U}_{\texttt{g}}$ nous dit d'arrêter, la sortie contient α.

On sait donc écrire un sup d'admissibles horlogeable. Malheureusement, un sup d'admissibles peut ne pas être horlogeable. En fait, elle peut même être admissible [HL00], tous à l'intérieur de la même brèche ou non, ou non admissible.
Cependant, ce cas nous suffit. En effet, prenons α n'importe quel écrivable, et notons β le sup des fins de brèches commençant avant lui. Comme β est supérieur à tous les admissibles inférieurs ou égaux à α, il est supérieur à leurs sup. Ainsi, α est récursif en le sup des admissibles inférieurs à β.

On vient de voir une façon d'écrire le sup des admissibles inférieurs à β, c'est-à-dire le sup de tous les admissibles dans une brèche commeçant avant α. Or α est récursif pour β, et écrire ce sup se fait d'une façon extrêmement stable : chaque case n'est modifiée au plus qu'une fois, et ceci bien avant le sup des admissibles (puisqu'on écrit un admissible en temps cet admissible). Il nous suffit donc d'utiliser la propriété de la sous-section 3.2 : il suffit de parallèliser la machine donnant ce sup avec une machine de Turing normale pour obtenir une machine écrivant α. □

Remarque 4.2.4. La fin de la brèche peut se situer bien après son début, puisqu'elle peut non seulement contenir le prochain admissible, mais aussi en contenir beaucoup, comme on l'a vu dans la partie précédente. Ainsi, il faut parfois un temps bien plus long qu'α pour écrire α, mais aussi parfois un temps plus court.

5 Problèmes ouverts

Il existe plusieurs autres champs des ITTM à étudier :

- Le premier est très liée au théorème de la sous-section 4.6 : combien d'états faut-il au minimum pour écrire un ordinal donné, ce qui a un début de réponse dans [DDL17]
- Un autre problème serait de savoir quelles sont exactement les longueurs de brèches possibles, quand elles arrivent et combien il y en a. Bien qu'on en ai déjà une petite idée dans la sous-section 4.5, il y a plus de choses à trouver et compiler dans la littérature.
- Il existe un autre champ des ITTM, développé notamment dans [HL00], sur la notion d'oracle : l'idée d'un oracle est d'être une boîte noire qui donne immédiatement une réponse à une question compliquée. Par exemple, on peut utiliser un oracle, noté $0^{\blacktriangledown}$, qui nous dit si une ITTM s'arrête sur une certaine entrée.

Références

[Bar90] John Barwise. *Admissible Set Theory*. Springer Verlag Publishing Company, 1990.

[CDLO17] Merlin Carl, Bruno Durand, Grégory Lafitte, and Sabrina Ouazzani. Admissibles in gaps. In *Proceedings of the Computability in Europe*, 2017.

[DDL17] Oscar Defrain, Bruno Durand, and Grégory Lafitte. Infinite time busy beavers. In *Proceedings of the Computability in Europe*, 2017.

[Goo44] Reuben Goodstein. On the restricted ordinal theorem. *The Journal of Symbolic Logic*, 9 :33–41, 1944.

[HL00] Joel David Hamkins and Andy Lewis. Infinite time turing machine. *The Journal of Symbolic Logic*, 65(2) :567–604, Jun 2000.

[KP82] Laurence Kirby and Jeff Paris. Accessible independance results for Peano arithmetic. *Bulletin of the London mathematical society*, 12 :285–293, 1982.

[Pap93] Christos Papadimitriou. *Computational Complexity*. Addison-Wesley, 1993.

[Sac76] Gerald Enoch Sacks. Countable admissible ordinals and hyperdegrees. *Advances in Mathematics*, 19 :213–262, 1976.

[Wel09] Philip Welch. Characteristics of discrete transfinite time Turing machine models : Halting times, stabilization times, and normal form theorems. *Theoretical Computer Science*, 410 :426–442, Février 2009.