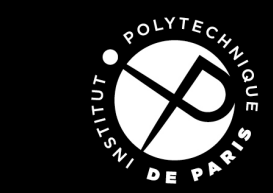
Toward higher-order and many-symbol infinite time Turing machines

Johan Girardot



INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2024IPPAX028

Thèse de doctorat

Toward higher-order and many-symbol infinite time Turing machines

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 4 juillet 2024, par

JOHAN GIRARDOT

Composition du Jury :

Julien Cervelle Professeur, Université Paris-Est Créteil (LACL)	Président
Bruno Durand Professeur, Université Montpellier (LIRMM)	Rapporteur
Joel David Hamkins Professeur, University of Notre-Dame (Department of Philosophy)	Rapporteur
Mirna Džamonja Chargée de recherche, CNRS (IRIF)	Examinateur
Olivier Bournez Professeur, École polytechnique (LIX)	Directeur de thèse
Olivier Finkel Chargé de recherche, CNRS (IMJ-PRG)	Directeur de thèse

Remerciements

Je remercie chaleureusement mes directeurs de thèse, Olivier Bournez et Olivier Finkel, pour leurs conseils avisés, leurs relectures exigeantes ainsi que pour leur bienveillance.

Je remercie également mes rapporteurs, Bruno Durand et Joel D. Hamkins, pour leur lecture minutieuse de mon travail et leurs retours éclairés et encourageants ; ainsi que Julien Cervelle et Mirna Džamonja d'avoir accepté de faire partie de mon jury qui, dans son ensemble, me fait profiter de son expertise.

Je tiens aussi à remercier mes référents, Arnaud Durand et Ludovic Patey pour leur disponibilité et leur écoute.

Toujours dans le domaine des mathématiques, je suis reconnaissant à l'ensemble des utilisateurs et utilisatrices de mathoverflow, et en particulier celles et ceux actives et actifs sur lo.logic, pour leurs explications patientes et leurs réponses détaillées à mes questions parfois naïves.

De l'autre côté du monde, dans le domaine de la philosophie, ce sont mes professeurs de la Sorbonne et la Sorbonne elle-même que je remercie, pour m'avoir abrité si souvent dans cet endroit où tout est calme.

Et finalement, entre toutes ces choses, je remercie ma famille et mes amis pour leur présence, autant que pour leur patience.

Contents

Chapter 1

Introduction

> ‘Θεαίτητος κάθηται.’ μῶν μὴ μακρὸς ὁ λόγος; οὔκ, ἀλλὰ μέτριος.
> STRANGER: “Theaetetus sits.” Is it a long discourse?
> THEAETETUS: No, it is of reasonable length.
> – Plato, *Sophist*

What started as an introduction to computability and mathematics ended up as an introduction to phenomenology and Heidegger's thought. Hopefully, it ends how it should have started. In Chapter 2, starting at page 23, can be found a scientific contextualization, a summary of the results as well as an outline of the document.

1.1 Heidegger and phenomenology

The mathematical, that is everything that pertains in one way or another to mathematics, is a world in itself. It is a world like many with long and tidy alleys, with backwoods, with exotic leaves and mysterious roots. Still, it is a peculiar world. It is a peculiar word inasmuch as it does not appear to us as a world when we enter it but rather when we leave it. We enter it in many ways in our daily lives: when cooking for three instead of four, when trying to get somewhere on time, when doing our everyday work as researchers, etc. And through such mundane pathways, we hardly imagine reaching anything more than a region of our own world. Still, when we leave it after spending there a long enough time–a long day is well enough–, and when we come back to the colorfulness of everything, to the sound of the wind, to the scent of the evening and the gentle freshness of the air, we come to realize that we have been surveying a whole another world: namely, a world where those things are not. And after spending a long day in this vast building where windows can only be slightly opened, we are well justified in asking: what was this world where those things, if not all the things, are not?

As we are here interested in what we called “the mathematical”, this question will be our first step toward it. But if this world is the world where the things are not, what is

left in it? Obviously, there is more than nothing: in there may be left everything that is not a thing. But then, another question arises: what is a thing? This is a second step, now in the wrong direction, but giving the first question the room it needs to unfold.

So, what is a thing? This question, in one way or another, can be found almost anywhere in the history of philosophy: in Plato's analogy of the divided line, in Abû Hâshim's theory of modes, in Descartes' sheet of wax example, etc. And for our purpose, we could trust the philosophers. That is, we could open a well-chosen book at the right page and find, or produce for our usage, a definition of the thing: a thing is that which is material, or a thing is the support for the sensations, or a thing is that which realizes in its unity a given set of properties, or a thing is anything that can be the subject of a proposition, etc. But this handful of definitions should already make us wary: either most of them are false, or we find ourselves in a blurry relativism that may be as complex as the question to clarify: we could assume that the acceptation of the thing is both historical and cultural and then start our way toward a classification of the different meanings of it. Then, once done mapping this history of the concept of thing, we should be able to localize ourselves with our current preoccupation in it. And with this finding ourselves and some luck, be able to eventually produce the definition of the thing that meets our needs. But well before achieving only half of this weary task, resting a bit and looking, through the window that only slightly opens, at the sky and the clouds slowly passing, we would realize that here too, in this new endeavor, the things have left.

In turn, we will as well leave this world where the things are not: if, in order to take a step back from it, we are questioning toward the thing, we should be in one way or another going toward the thing and leaving it. Where, if not in the proximity of the things, should we be able to answer our question? Such a step back is already progress. Namely, we unveiled a new question: how do we reach the things in their proximity? And with it, in our going sideway from the history of philosophy, we realize that we still walk a well-trodden path: that of phenomenology. Founded by Husserl and re-interpreted by Heidegger, its *mot d'ordre* was: "to the things themselves"[1]. So, how, to the things themselves?

Naively but decidedly, toward the things, we consider a thing. A glass, a glass from which we can drink, made of glass, solid but frail at the same time. It stands in front of us, on the wooden table, and we see it. Why do we see it? For this question, science has an established and well-tried answer: photons, which form light, upon reaching the surface of the glass, are reflected with various levels of energy through a game of excitation-emission. Then, reaching our retina, those photons will activate different photoreceptor cells, themselves sending nervous impulses to the brain. With those, the brain is able to construct an image of the glass and we "see" the glass. But does this explanation bring us closer to the glass? Not really: in it, the glass is barely a glass anymore. It is a cluster of atoms with various levels of energy and intriguing micro-reflection properties. Hardly

[1] *Zu den Sachen selbst*, Husserl, *Logische Untersuchungen*, band II.

something a well-mannered individual would want to drink with. Moreover, against this explanation, when someone points toward the table and asks for what we see, our first answer invariably is: "we see a glass". We do not *see* a glimmering of lights that we then manage to crystallize as a glass; what appears to us is a glass. Only after its apparition can we see what it is made of, the details of its production and the intricate patterns that the sunrays describe on it. The glass is, first and foremost, given to us and the previous scientific explanation is itself grounded on this very donation. And this donation of the glass is also a donation *from* the glass: it is the glass showing itself to us. And this is the phenomenon: "the showing-itself-in-itself"[1]. From the greek φαινόμενον, the middle voice (that is at the same time passive and active voice) of φαίνω: to put under the light (φῶς), to make visible. The phenomenon is that which itself brings itself to light, what comes by itself into the open. And with it, with this donation, we are already, before anything, in the proximity of the thing.

So the guiding question once again changes its shape: rather than finding a way into the proximity of the thing, we want to find a way not to leave it. The phenomenon is the thing which itself shows itself as itself. But as we saw, it can easily and inadvertently be concealed: as much as we can't see a glass through a book, the thing in its giving-itself was hidden behind the opaque explanation of science. So, this explanation out of the way, should we then simply look at the glass and try to describe it? But why should we look when already the thing is showing itself? Do we only see when we look? Isn't, on the contrary, the effort of describing it, that is of seizing it in the language, another veiling of the thing? This time not a veiling from too great of a distance, making us lose sight of the thing, but a veiling from the abolishing of the distance and with it, of the proximity. Distance is naturally the condition of proximity: we feel the proximity from the person we love but not from our feet. The nose pressed on the surface of the thing, looking for more than what it gives by itself in its own coming-to-the-world, what appears is not the thing anymore: it lost its thingness. When we scrutinize the glass, it is not what we can drink with anymore, when we scrutinize the walking stick, it is but a strange piece of wood: its usefulness fades away. And this usefulness is part of the Being[2] of those things as things: it comes before the thing itself and the thing itself, in its production–that is in its material apparition–, is grounded in this usefulness.

Then, how should we live with the things and their proximity that surround us if, at the same time, we dispel it wherever we look? Do we have any access toward the thing that is not an intrusion, or should we simply renounce? We look at Heidegger's essay *Der Ursprung des Kunstwerkes* (*The Origin of the Work of Art*). In this work, he is looking

[1] *das Sich-an-ihm-selbst-zeigende*, HEIDEGGER, *Sein und Zeit*, p. 28.

[2] English is an unfortunate language when it comes to philosophy: the infinitive form of a verb can't be used as a substantive. So "Being" translates "das Sein" or "l'être". But then, its gerund, "das Seiende", "l'étant", precisely what should be distinguished from the infinitive form, reads: "being". Hence it is commonplace to write a capitalized "Being" to translate "das Sein" and "being" (often used in its plural form, which helps a bit) to translate "das Seiende".

into the relationship between the work of art, as a thing, and the things themselves. He takes as an example the leather shoes of a peasant, as often painted by Van Gogh: "The peasant woman simply wears the shoes. If only this simply wearing was that simple."[1]: when wearing those shoes, she relies on them with so an unquestioned trust, and yet she never thinks about them, she never *looks* at them, and yet at anytime she knows where they are. They blend in the world of the peasant, and yet they are a support point in this very world. In their reliability, the shoes are part of the world of the peasant, and their proximity is preserved in this world. The thingness of the shoes unfolds in the shoes themselves, and for a moment, they hold the answer to our question. But then again, it means that the answer withdraws further in this holding of the shoe. Still, something new appears: the usefulness of the thing, which helped us toward the thing, is itself grounded on its reliability. And this reliability, thought before and without the usefulness, that is thought as a point of support on the world, before and without what it may support, more generally lifts this thinking up to the things that are not deemed "useful".

But from all of this, the painting by Van Gogh of a pair of shoes stays at first glance quite remote: the shoes are extracted from their world and they stand by themselves on a wooden floor. There, their usefulness hardly stands out. They are exposed and their reliability is forgotten. And yet, when we put ourselves in the presence of the painting:

> The obscure openings of the beaten inner of the shoes are fraught with the hardship of the toilsome footsteps. In the crudely solid heaviness of the shoes accumulates the tenacity of the slow trudge through the far-stretched and ever-identical furrows of the field, over which a crude wind lingers. On the leather lies the dampness and the richness of the soil. Under the soles and through the descending evening stretches out the loneliness of the field paths. In the shoes resonates the silent call of the earth, its still offering of the ripening grain and its unexplained self-refusal in the desert fallow of the wintry field. Through this equipment passes the mute worries regarding the certainty of bread, the wordless joy of again overcoming the need, the tremor in the imminence of the birth and the shiver in the surrounding threat of death. This equipment belongs to the *earth*, and it is sheltered in the *world* of the peasant woman. From this sheltered belonging the equipment itself arises to its resting-in-itself.[2]

[1] *Die Bäuerin dagegen trägt einfach die Schuhe. Wenn dieses einfache Tragen so einfach wäre.* Heidegger, *Der Ursprung des Kunstwerkes*, GA 5, p. 19.

[2] *Aus der dunklen Öffnung des ausgetretenen Inwendigen des Schuhzeuges starrt die Mühsal der Arbeitsschritte. In der derbgediegenen Schwere des Schuhzeuges ist aufgestaut die Zähigkeit des langsamen Ganges durch die weithin gestreckten und immer gleichen Furchen des Ackers, über dem ein rauher Wind steht. Auf dem Leder liegt das Feuchte und Satte des Bodens. Unter den Sohlen schiebt sich hin die Einsamkeit des Feldweges durch den sinkenden Abend. In dem Schuhzeug schwingt der verschwiegene Zuruf der Erde, ihr stilles Verschenken des reifenden Korns und ihr unerklärtes Sichversagen in der öden Brache des winterlichen Feldes. Durch dieses Zeug zieht das klaglose Bangen um die Sicherheit des Brotes, die wortlose Freude des Wiederüberstehens der Not, das Beben in der Ankunft der Geburt und*

In the presence of the painting of Van Gogh, with its blurry contours, its chiaroscuros, its simple and crude strikes and its modest approach of the thing, we see how it preserves the thingness of the peasant shoes. "The being-equipment of equipment was only discovered by bringing ourselves before the Van Gogh painting. It is this that spoke. In proximity to the work we were suddenly somewhere other than we are usually accustomed to be. The artwork let us know what the shoes, in truth, are."[1]

So despite those difficulties in experiencing our relation with the thing, its being-a-thing has to do with the truth of the work of art. Consequently, we do have a privileged relationship with it. After all, it is indeed in our world that the thing becomes the thing that it is: the pair of shoes realizes itself in the world of the peasant woman. The work of art finds its place in the sheltering of the proximity of the pair of shoes. The earth, the sky and the rainclouds themselves are in this world in the proximity of the shoes and the growing wheat. The coming into the open of the shoes opens the very domain in which the things come. And at the same time, this open is enabled by our living in our world. So the initial question resonates even more deeply in this privileged relationship with the things: what is a thing if the thing by itself comes into the open but an open that it opens and of which we are the condition? What is a thing if the thing preserves its thingness in its self-refusal of the meaning that *we* project on it? And what hides behind those questions is that of Being: "The artwork opens up, in its own way, the Being of beings. This opening up, i.e., unconcealing, i.e., the truth of beings, happens in the work."[2]

In all of this, what matters for us is that, through all those difficulties, we realize that the principle of phenomenology, "to the things themselves" is in reality more a grounding question than a working principle. And reaching this question in its very difficulty; that is, for a moment, renouncing the idea of finding an answer and letting the question unfold and come to us in what makes it questioning, this was the task of phenomenology as set by Heidegger. Only then do we reach a solid ground on which the question can be answered. But as this ground was reached by renouncing the answer, the answer can only come back in another form.

> The answer to the question "What is a thing?" has a distinctive character. It is not a proposition but a transformed fundamental stance or–better yet and more cautiously–the initial transformation of the stance we have heretofore taken toward things, a transformation of our questioning and evaluating, of

das Zittern in der Umdrohung des Todes. Zur Erde *gehört dieses Zeug und in der* Welt *der Bäuerin ist es behütet. Aus diesem behüteten Zugehören ersteht das Zeug selbst zu seinem Insichruhen.* Ibid.

[1] *Das Zeugsein des Zeuges wurde gefunden [...] nur dadurch, daß wir uns vor das Gemälde van Goghs brachten. Dieses hat gesprochen. In der Nähe des Werkes sind wir jäh anderswo gewesen, als wir gewöhnlich zu sein pflegen. Das Kunstwerk gab zu wissen, was das Schuhzeug in Wahr-heit ist.* Ibid., p. 21, trans. Young & Haynes.

[2] *Das Kunstwerk eröffnet auf seine Weise das Sein des Seienden. Im Werk geschieht diese Eröffnung, d. h. das Entbergen, d. h. die Wahrheit des Seienden.* Ibid., p. 25, trans. Young & Haynes.

seeing and deciding, in short, of our *Da-sein* in the midst of beings.[1]

This excerpt provides an overview of Heidegger's phenomenology and, more generally, of its view on the task of thinking: it is a preparation, seen as the deconstruction of a tradition that veils the questions, and the very beginning of a transformation, founded on this new cleared ground. But then, what is this transformation? We caught a glimpse: it is a disponibility and a restraint, a being open for the coming-into-the open of beings.

There is still much to think about in this asymmetrical reciprocal relation that we entertain with the things–even if much of it has already been thought through in Heidegger's writings. Indeed, this relation is nothing less than that which is encompassed in the esoteric-looking term of *Da-sein*. Often mistranslated,[2] it should be translated as "being-the-there". "The-there" is nothing like a spatial determination but rather the condition of it: this "there" is nothing else than what we called the open. It is this open which, at the same time, is broken through by the things and of which we phenomenologically assume the role. *Da-sein* expresses our perpetual privilege and responsibility toward beings and the perpetual opening-the-open and inward-looking of the things in their Being: "The human being is not the lord of beings. The human being is the shepherd of Being."[3]

Through this short walk in the shade of the question "what is a thing?", following the cautious steps of Heidegger, we managed to let this question develop itself sufficiently and we are able to return to our initial concern: what was this world, where the things are not? But we have also seen another thing: in the hottest summer ever recorded and in an ever-growing and unrestrained control and exploitation of the world, the need for a new fundamental stance toward the world and the things in it is more urgent than ever. And the key to an environmentalist thought may be found in that of Heidegger: a thought that rebalances the relationship between individuals and that which populates the world and makes it world, and a thought that understands that proximity can only be where distance is respected, whether it is metaphysical or physical.

1.2 Plato and *lógos*

Let us take a look back at the path we took. It involved avoiding many other paths: right from the start, we became wary of the definition, we then steered away from the scien-

[1] *Die Antwort auf die Frage »Was ist ein Ding?« hat einen anderen Charakter. Es ist kein Satz, sondern eine gewandelte Grundstellung oder - noch besser und vorsichtiger - der beginnende Wandel der bisherigen Stellung zu den Dingen, ein Wandel des Fragens und Schätzens, des Sehens und Entscheidens, kurz : des Da-seins inmitten des Seienden.*, Heidegger, *Die Frage nach dem Ding*, GA 41, p. 49, trans. Reid & Crowe modified.

[2] "'Da-sein' is a key word of my work, and it is the victim of serious misinterpretations. 'Da-sein' does not really mean for me 'here I am !' but, if i may express myself in a likely impossible French: being-the-there (être-le-là) and the-there (le-là) is precisely Aléthéia, disclosure — opening." Heidegger, *Questions III et IV*, letter to Jean Beauffret of the 23 novembre 1945, p. 130, trans. from French.

[3] *Der Mensch ist nicht der Herr des Seienden. Der Mensch ist der Hirt des Seins.* Heidegger, *Brief über den »Humanismus«*, GA 9, p. 342.

tific explanation and restrained ourselves from the temptation of a pictorial description. Finally, we had to abandon altogether the idea of reaching the thing in its Being by some kind of proposition. In retrospect, it is easy to see that what we have been avoiding is the language itself, in its most practical form. Indeed: the thing are not found in the language, if only for the very reason that the language points *towards* the things–and consequently outwards. So after the thing, slowly closing on the mathematical, we will now struggle with the language. Is the mathematical to be equated with the language? This would seem a bit unreasonable. However, the mathematical surely has a place of its own in the language.

For a better understanding of this, we now turn to the work of Plato. This immediately prompts a question: while the work of Heidegger is more or less contemporary, why should we resort to two-millennia-old dialogues? Putting aside the possibility of its study being a mere exercise in erudition, there must be something to be found in the thought of Plato. But in this case, after more than thousands of years of study of Plato and as much of a progress in every possible field, this something that may be found in it must have been adequately understood–that is extracted–, furthered, and laid out again in a clearer and sharper way. The assumption that there is something in Plato that we are about to grasp and which has not yet been understood or that no one yet managed to think forward would be for the least bold. As much as the assertion that philosophy has only gone downhill since its antique beginnings would be poor. On the contrary, there is undeniable progress even in such a field where the need for a clean slate can be so often and so strongly felt. But, precisely, what is progress? Progress is answering questions. When we do answer a question, we write down its answer, not to forget it, and we get on our way towards the next question. But this question, that which we just answered, is often simply left behind and forgotten: every going forward comes with a price, and progress is the disappearance of the questions. Only a few famous questions go through the ages–but how largely hollowed out!–and if not for history of philosophy, we would have wholly forgotten the questions that the words *existentia*[1], *voluntas*[2] and *producere*[3]

[1]Root of the word existence, the Latin word *existentia* is built as a substantivation of the verbe *exsistere* (*ex-sistere*): to make something stand, to place something (*sistere*) out of something (*ex-*). Marius Victorinus coined the term in the IV$^{\text{th}}$ century in the theological context of the trinity: the father and the son are one, but at the same time the father is not the son and the son is not the father. How can those two affirmations be conciliated? The answer of Marius Victorinus is the following. The son is *ex-istantia* of the father: he is the manifestation of the father out of the father; that is, he is the generation by the father in which the father ex-ist. As the apparition of the father, the son and the father are one; but the son is not the father and the father is not the son. See V. Carraud, *l'invention de l'existence*, Quaestio, 2003, pp. 3-26.

[2] *Voluntas* gives in French *volonté* (willingness) and was introduced by the theologian Augustine in his *De libero arbitrio*. Augustine, throughout his difficult life, was frequently drawn to the following question: what is the origin of sin in a world created by god, which itself is good? For him, if there is sin, as it can't originate from god, it must come from man. And here comes into play the *voluntas*, for which there was no Greek equivalent: for the human being to act wrongly, he must be given the possibility of doing either the right or the wrong thing; in other word he must be given free will, and with it the muscle of well-doing: the *voluntas*.

[3] *Producere* naturally gives "to produce". *Pro-ducere* means to lead, to bring, to draw (*ducere*) in front

answer.

The question that will interest us, or rather that will give us the necessary inertia, is the following: how can we know something that we did not know? That is, how can we, while leaning over our desk and thinking about a problem, without any outside influences suddenly know something that, the instant before, we did not know? This question is easy enough to ask, but as we have seen, asking is rarely enough and may even conceal the fact that we are not yet in the presence of what makes the question questioning. Moreover, as we have seen in the previous examples, answers are etched in the language, and so the language itself can get in our way towards the question. This is why we will need to delve, as much as this short introduction allows us, in the dialogues of Plato. Doing so, we will also avoid translating some words who, precisely in their ambiguity, hold the questions we are looking for. Such a word and the most important one will be *lógos*. Its translation ranges over: "words", "discourse", "relation", "reason", etc. and its pivotal role in Plato's thought will shed light on its meaning.

For this question, how do we know?, a famous example is taken by Plato in *Meno*. It is that of Socrates discussing with a slave of Meno the possibility of constructing a square that is twice the area of a given square. Socrates wants to show that the slave does not learn but *recollects* (*anámnēsis*) from its soul, which is immortal, the knowledge about this geometrical construction. We put aside the question of the soul to focus on the process of recollection. What interests us primarily is the first essential step in the recollection: knowing that we don't know when we didn't know yet that we don't know. This is the first difficulty that Socrates is faced with: the young slave thinks he knows how to construct such a doubled-area square, hence hiding the fact that he doesn't know. Through a series of questions, he leads him to realize his error. Then he says to Meno:

> Do you think he'd have tried to enquire or learn about this matter when he thought he knew it (even though he didn't), until he'd become bogged down and stuck (ἀπορίαν), and had come to appreciate his ignorance and to long for knowledge?[1]

Obviously, Meno agrees and so do we. Now, we draw our attention to this term, *aporía*. From the greek *á-poros*, it means impasse, somewhere where we are stuck with no way (*poros*) forward. But what exactly is this impasse? That is, what kind of way forward suddenly disappeared for the young slave?

We find again this *aporía* in *Philebus*. There, Socrates discusses with Protarchus about "the identification of one and many" happening in the discourse. It happens when we say

(*pro*). It remarkably is built exactly like the German *hervor-bringen* (to produce) and it holds in itself an answer to the question of the coming into the open that takes place when the craftswoman produces an object: the object must be, in one way or another, say in the mind of the craftswoman, before the object can ex-ist. So the pro-duction answers this issue of antecedence by transforming the creation in a bringing forth of the object out of the mind of the craftswoman.

[1] Plato, *Meno*, 84c, trans. R. Waterfield.

that this person is tall but also beautiful and intelligent: how are three predicates, that is a multiplicity of predicates, crammed together in one person? Should we say that one and many can be identified? This naturally leads again to many aporias:

> In my view the identification of one and many by statements (λόγων) crops up all over the place in everything that is ever said, and it's not a new phenomenon. Indeed, it seems to me, this is an indestructible and unchanging feature of our statements (λόγων), which is not only not new but will always be with us. When a young man first savors it, he is delighted, as if he had found some treasure of wisdom. His delight goes to his head and he loves to worry every statement (λόγον), sometimes rolling it one way and kneading it into a single ball, then unrolling it again and tearing it apart. The result is confusion (ἀπορίαν), first of all for himself, but also for anyone who happens to be by, whatever his age. He has no mercy on his father nor mother, nor on any of his audience.[1]

What appears clearly, is that this aporia is reached by the young man both *in* the *lógos* and *with* the *lógos*. The *lógos* is like a playground with its own set of rules in which the young man can play, guided by those rules, and without having to think by himself.

We experience those rules guiding us in our everyday use of the *lógos* thought as discourse: how often are we caught talking without thinking, letting the power of habit speak for us. But this habit is itself rooted in the rules of this playground that is the language. for example, if someone says: "I see the Jabberwock and a Cheshire cat", without even pondering about the meaning of those words we can affirm that: "he sees the Jabberwock" and "he sees a Cheshire cat". This is an easy example of the "unrolling and tearing apart" described in the previous excerpt which, through no coincidence, reminds us of the logical rules of deductive systems. And in return, this example gives a good idea of what may be the "rolling and kneading into a single ball". It is also that which takes place in the identification of the one and the many: this game of moving forward without thinking forward which, as everybody knows, can quickly lead to a bogging down: the aporia.

The *lógos* is "a thinking machine whose gears are already set up"[2]. It has its own power of development, but this development is not arbitrary: it follows an internal logic, the rules of this playground, that brings to the surface what is still enclosed in it. Plato attests such a coherent development of the *lógos* in the *Sophist*:

> The statement (λόγος) itself will reveal (δηλώσειε) [its contradiction] if we put it to the test (βασανιστεις).[3]

[1] Plato, *Philebus*, 15d, trans. J. C. B. Gosling.
[2] Maldinay, *Cours de philosophie générale*, 1964/65.
[3] Plato, *Sophist*, 237b.

The central word here is *dēlóō*, to make visible: what the *lógos* shows when its gears are put into work, when it is given the impetus it needs to show itself by itself, is simply what was already in it but concealed. What is concealed in it may be contradictory, as in the previous example – or not:

> [The *lógos* "the human being understands"] reveals (δηλοῖ) something about the things that are.[1]

So, the aporia is the revelation of a contradiction regarding the things: the way forward that disappears in the aporia is exactly the one leading to the revelation of the aporia. But if we are strict with our reading, with the idea of the *lógos* revealing something about the things, this something must be in a way closer to the things than it is to the *lógos*; otherwise, if the *lógos* reveals another *lógos*, how do we progress toward the things? But then, if the logos can reveal a contradiction about the things, on the side of things, what would be a contradiction? Things, say my chair and the roof tiles of my neighbor, can hardly contradict each other. We will be brought back to this question.

In any case, the *lógos* has two fundamental aspects: first, it has its own unfolding power that the individual can simply "engage forward" and, second, this unfolding is a revealing of what was concealed–and at the same time hold and sheltered–in it. We see how the many meanings of *lógos* all come into play at once: the *lógos* as a discourse shelters and brings into the open the *lógos* as meaning. But this bringing into the open is only done under the pressure of the individual that operates the gears, that puts the *lógos* to the test, and so, in turn, we see the *lógos* as reason appearing. In other words and through a deeper and more careful examination, we would be able to see that the questions that the term *lógos* encloses are becoming so pressing in Plato's dialogues that, with those, we are on the threshold of the burst of the very concept of *lógos* into its many translations that we know today.

But keeping intact a while longer its integrity and with this two aspects in mind, we circle back to our question. The aporia, reached through the putting in action of the *lógos* according to its own rules, makes someone realize that she does not know something. This is a central prerequisite for the recollection. After it, the recollection, that is the remembering of what is in her soul but forgotten, can take place. Again, this happens with the *lógos*. How does it play out? We had a glimpse in *Meno* between Socrates and the young slave, it plays out through a game of questioning and answering:

> When people are questioned, if you put the questions well, they answer correctly of themselves about everything; and yet if they had not within them some knowledge and right reason (ὀρθὸς λόγος), they could not do this.[2]

[1] Ibid., 262d.
[2] Plato, *Phaedo*, 73a, trans. Fowler.

And this, the game of "questioning and answering"[1], corresponds precisely to the fundamental notion of dialectic. But at the same time, it is nothing less than the putting to test or the putting under the touchstone (*basanos*) of the *lógos* for it to reveal the knowledge it encloses.

So the impetus given by this question, "how do we know?", helped us put into light how the *lógos* and the dialectic, that is the questioning and answering, work hand in hand. In turn, this will give us the hook we need on the particular kind of *lógos* that interests us: the mathematical one. To do this, we now need more content on the dialectic. Plato's *Republic* discusses this subject at length. Again, we want to know what triggers it:

> Some things are apt to summon thought, while others are not, defining as apt to summon it those that strike the sense at the time as their opposites, while all those that do not, are not apt to arouse intellection (νοήσεως).[2]

And intellection (*nóēsis*) is "that which argument (λόγος) itself grasps with the power of dialectic"[3]. So that which provokes the senses and provides contradictory impressions also gives the needed jolt for the individual to rebel against this conflict of the senses and to try to solve this aporia through the dialectic. As an example, Plato comes back to what we encountered in *Philebus*, the conflict of the one and the many.

> The sight, with respect to the one, possesses this characteristic to a very high degree. For we see the same thing at the same time as both one and as an unlimited multitude.[4]

But this time, the example of the one and the many is clearly shifted in the realm of the senses. This begs the question that we already crossed paths with when surveying the possibility of things contradicting each other: how do we *see* something as one and many? How do we even see something as one, or as many? A few lines before, Socrates illustrated this conflictual sighting with our fingers: our hand in front of us, we see the three most central fingers. One of them is smaller than the bigger one and bigger than the smaller one. So this finger is seen, against one of its neighbor, as big and, against the other, as small. In the sighting of the one finger collide many–here two but moreover contradictory–visual impressions. And so comes into play the intellection, "[which was] compelled to see big and little, too, not mixed up together but distinguished, doing the opposite of what sight did"[5]. This example is as simple to describe as it is thorny to unfold: the *lógos* oddly straddles the line between the sighting and the intellection. As seen, through the dialectic, it plays an important part in the process of intellection. But in this example, it

[1] Plato, *Republic*, 487b.
[2] Ibid., 524d, trans. A. Bloom.
[3] Ibid., 511b, trans. A. Bloom.
[4] Ibid., 525a, trans. A. Bloom.
[5] Ibid., 524c, trans. A. Bloom.

also does in the sighting[1], as, for the finger to be seen big or small it must already be put in relation, by the sighting, with those ideas from "big" and "small". A "pure" sighting as science likes to describe would simply gather bare information unrelated to any *lógos*. But such a role for the *lógos* in the sighting is in line with what we described: the sighting is infused with *lógos* and in return the revealing of the *lógos*, that is, at the same time, the revealing through the *lógos* in its mechanical aspect and the revealing *out of* the *lógos* in its veiling and sheltering aspect; this revealing brings into the open *ce sur quoi il porte*, that which it covers about the things. We see through the *lógos* which constantly reveals the things to us. And this is why we can have contradictory sightings, as this revealing, which we previously referred to as the second aspect of the *lógos*, goes hand in hand with its first aspect: a self-going-forward in the language that reveals itself, this going-forward, when it stumbles on–or rather in–the aporia. But in this case, in the sighting, compared to the young boy playing in language, the *lógos* makes its way through the language *à pas de loup*, very quietly putting this march into silent words until it provokes the jolt of the aporia that makes us utter a mute gasp of surprise. After this, the *lógos* can progress in the open through the dialectic and toward the intellection, *nóēsis*.

As an aside, this intricate state of affair–whose complexity can only be measured in facing the difficulty it represents–, that is this going through from the *lógos*, through the frontier between sensation and intellection, is precisely what the concept of *diánoia* can be interpreted to capture[2]. Often over-translated by "thought" or "ability to think"–by lack of better translation in modern languages–, *diá-noia* can be seen in Plato as the going through (*diá*) of a way more fundamental "thought", namely that which is conveyed by the *lógos*, itself in one a more fundamental aspect than that of discourse or of reason: precisely that which we caught a glimpse of in its relation with the perception. But again, this is a questioning that goes through all of Plato's work, and that can too easily be veiled by the answers etched in our language. In *Theatetus*, Theaetetus asks precisely this question to Socrates, "What do you call thinking (διανοεῖσθαι)?". We give the answer of Socrates as translated by C. Rowe:

> A talk (λόγον) that the soul conducts (διεξέρχεται) with itself about whatever it is investigating (σκοπῇ).[3]

Against a more literal translation:

> A *lógos* that the soul itself conducts through itself about whatever it has in his sight.

[1]Which is actually remarkably reminiscing of the previous phenomenological description: for the glass to be *seen* in the way we described, the idea of "glass", or rather something akin to it but more fundamental, must come into play in the sighting. This is actually the pivot around which all of *Sein und Zeit* is structured, providing an access–albeit elusive–to Being through the *phenomenological* sighting of beings.

[2]See Heidegger, *Vom Wesen der Wahrheit*, GA 34.

[3]Plato, *Theaetetus*, 189e, trans. C. Rowe.

And the end of the quote for the reader to make her or his own mind, remembering the peculiar and pre-conceptual role that the *lógos* has in the sighting:

> That's what I'm claiming, at any rate, as someone ignorant about the subject. The image I have of the soul as it is in thought (διανοουμένη) is exactly of it as in conversation (διαλέγεσθαι) with itself, asking itself and answering questions and saying yes to this and no to that. When it fixes on something, whether having arrived at it quite slowly or in a quick leap, and it is now saying the same thing consistently, without wavering, that is what we set down as something it believes. So I for my part call forming and having a belief talking (λέγειν), and belief a talk (λόγον) that has been conducted, not with someone else, or out loud, but in silence with oneself.[1]

We put a halt to this dwelling and come back to our concern: the mathematical. What is his place in this landscape that the *lógos* describes? At first sight, in Plato, mathematical arts is that which "leads toward truth."[2] They do so, as seen, by stimulating the individual through proto-contradictions, which lead to *nóēsis*. The kind of mathematical art that Plato has in mind is that which draws attention to issues like the one about the one and the many which we already encountered a few times. But this would be, at best, distantly related to modern mathematics. The modern approach of mathematics is indeed grounded on an unquestioned trust in its language. For example, we do not question further the unity encompassed by the number 1, neither do we question ourselves about the unity of the number 3, itself made from a multiplicity of the unity of the number 1. And Plato was well aware of something akin to this modern approach which he criticizes in the geometricians:

> They speak as though they were practitioners and were making all their proposition (λόγους) for the sake of their practice, speaking of 'squaring', 'applying', 'adding' and everything of the sort whereas the whole study is surely pursued for the sake of knowing (μάθημα).[3]

This approach that Plato laments–Plato which wasn't a mathematician for the standards of his time–clearly has a deep relation with a specific part of the language. More precisely, this practice grounds itself in the language it uses. And Plato blames it as, according to him, it does not yield intellection. With this, we are finally able to close the loop. Intellection is insistently prompted by the *lógos* reaching, in one way or another, an aporia. This aporia is the revelation of a contradictory state of affairs, toward which precisely the *lógos* points. Why does this not happen in the case of mathematics as practised by the geometricians? Simply because their *lógos* points toward the language itself: the *lógos*

[1] Ibid., trans. C. Rowe.
[2] Plato, *Republic*, 525b, trans. A. Bloom.
[3] Ibid., 527a.

of the geometrician reveals precisely what it is built on, it does not point towards and outwards but inwards itself. The two aspects of their *lógos* are merged together, and its revealing through unfolding keeps on revealing the *lógos* itself. They are glued together and unfold in a new dimension as does the Klein bottle.

And this appears clearly from the point of view of modern mathematics: "aporia" is a word that the mathematicians never use, they do not say that the mathematical machinery brought them to an impasse but rather they say: "*I* made a mistake", that is they say that they derailed this machinery and themselves caused the contradiction. They do not look forward at the lack of paths but backward for their own exiting of the mathematical rails. There is an implicit–because ubiquitous–trust in the language over which the mathematical propositions are built. This trust was the support for the ever-growing care in the usage of the mathematical language that developed itself over the centuries: it gave rise to the mathematical language on which modern mathematics are built. Why did it work so well? What was this trust itself resting on? This may be the question that underlies all philosophy of mathematics, if not all philosophy of knowledge–and that we may never properly reach.

And what is this language exactly? By mathematical language we do not mean the structured collection of words that are used by the mathematician at some point in time, which would itself correspond to the Greek *glōssa* rather than to the Greek *lógos*. It is on the contrary the very possibility of this mundane acceptation of the mathematical language, that is the *logic* of this language. And as such a possibility, it already encloses the mathematical concepts, proofs and propositions that are yet to come. This is why we can understand a new proof involving new objects simply by reading through: we already knew it, only still in a concealed way. The reading through and the grasping was a revealing of what was already known. This language, as the possibility of an enclosing enabled by the human being, is precisely what is revealed by the mathematical practice: it is the forgetful *lógos* forgetting about the things, constantly revealing itself through itself because turned inwards on itself and, for this reason, as simple and poor as it is rich and mysterious.

1.3 Computabiliy and mathematics

Eventually, toward computability theory, let us take a practical example of this mathematical language. What does the *lógos*, "the big tree in my backyard", has in common with the big tree in my backyard? Nothing: this sentence, "the big tree in my backyard", hardly has a trunk or any variety of leaves. On the other hand, what does "seventeen" has to do with seventeen? Everything: mathematics consists in grasping–that is revealing–in the language what is already in the language.

But don't we trivialize the mathematical practice here? Only on the surface: here, "grasping" means bringing down to human understanding. Consider for example the

question of whether there are infinitely many prime numbers. We can imagine a proof that shows this is true by writing down infinitely many of them. That would surely be satisfactory enough for some kind of infinite being. From our point of view, however, we want this infinite stream of information to be gathered, in one way or another, into some finite object. And this is exactly what the usual proof by contradiction does: the other way around, it provides an effective method to produce an infinite list of prime numbers, one at a time. We could say that the list is compressed in the finite description of the algorithm that the proof yields.

Hence, we see two things: mathematics is the privilege of finiteness, and the grasping in the mathematical practice–in the sense of seizing rather than understanding–is inherently a compressing. Again, the other way around: the unfolding of the mathematical language can be seen, in a very pragmatic way, as the uncompressing of the information it encloses. The finite proof of the infinity of prime numbers can be made to talk further and further about greater and greater prime numbers. And those two things go hand in hand, as this compressing is motivated by the humanly finiteness as much as the unfolding makes only sense while there remains something to unfold.

From there, can we study this compressing and uncompressing of the mathematics? That is, can we study this game of inward pointing and of unfolding on which the mathematical practice is grounded? The mathematical practice aside–as mentioned this one is far out of our reach–, this is the object of proof theory. A formal proof, after all, is the unfolding, according to some rules, of a given statement. In logic, a theory T generated by a set of axiom S is the "maximum unfolding" of S according to some deductive system. Quite often, S is finite while T is infinite. But this way of seeing things is still infused with mathematical practice. That is, it observes the unfolding capacity of mathematics through the eyes of a mathematician. As a sensible approach as it sounds, we may want to start with a more agnostic approach. After all, any other field of mathematics or computer science would be reluctant–and rightly so–to start their study, say of spaces or of graphs, from the very singular position of the mathematician.

And so, the mathematician and its practice out of the way, and as long as we manage to keep them at bay, what remains is the study of the relation between the finite and the greater finite–or even more, between the finite and the infinite. And, from a certain angle, this is nothing else than computability. As it was apparent in the example of the prime numbers, an algorithm is the key element for the compression of an infinite amount of data. It is the finite object which speaks of this infinite object, and as such the finite object that names it, the first ply of the mathematical unfolding. If we are interested in the details of this unfolding, we turn ourselves to complexity theory. If we are only interested in the possibility of this unfolding–or rather, dually, in the possibility of the folding of this infinite object–, we turn ourselves to computability.

However, before anything, computability is born from an impossibility. This was the condition for the possibility of compressing to be interesting in itself. This impossibility

has many forms. Among others, it can be encompassed by the halting problem. If we let (p_i) be an enumeration of all programs then the real

$$x = \sum_{i \,|\, p_i \text{ halts}} 2^{-i}$$

is a first example of a real which is not compressible. That is, we can't design a (finite) algorithm which on input i gives us the i^{th} bit of x. On the contrary, as we saw, the real

$$y = \sum_{p \text{ prime}} 2^{-p}$$

is compressible in the previous sense. And so two groups emerge, the group of those infinite objects which are compressible, or equivalently computable, and the group of those which aren't. And with those, the field of computabilty. Now, if we are satisfied with the existence (or rather non-vacuousness) of this second group, we may go back to the study of the first. That is, we now want to know which are those objects that are compressible. To this aim, it is simpler, as often, to start with the end, that is with the compressed form: the algorithm. To do so however, we need to define the space of the algorithms we consider. One way to do it is using the abstract model of Turing machines. A Turing machine is a simple machine-like object that carries out a computation (we will come back in the Chapter 3 to the difficult but rich question of the meaning of "a computation") with some input. It may not halt but if it halts, it does so in finitely many steps, in which case it produces a finite output. And as soon as we have those machine-like objects, we have partial functions from $\mathbb{N}$ to $\mathbb{N}$. Each Turing machine corresponds to one of those functions and the machine m corresponds to the function that maps i to the output of m computing with input i. If we look in particular at the machines inducing total functions from $\mathbb{N}$ to $\{0, 1\}$, it is easy to see that those describe reals in their binary expansion, and those reals are exactly the compressible reals. Alternatively, especially when we start from the machines rather than from the reals, those are called computable reals. This is a prime example of a relation between the finite and the infinite which formalized by computability: computability is the deploying of finitness and those kind of relations, in the usual ubiquitous and veiled fashion, underlie most of our relation to mathematics.

But are we done after putting those under light? That is, did we make a serious attempt at seizing and abstracting the folding and unfolding power of the mathematical language? For the moment, only halfheartedly, if only for the inescapable reason that describing the limit of such a setup at the same time escapes this setup and is itself done in the mathematical language. The real which we introduced, this x that encodes the halting problem, is, once we fix an enumeration of our Turing machines, clearly and unambiguously defined for the mathematician. And so, to some extent, the impossibility

in linking back this real x to finitness–and so in naming it–, forms itself a name for it. And this limit of the formal language, drawn at the frontier with the mathematical language, can in turn be formalized. That is, the mathematical language can be unfolded further and crystallized in the usual way. And with this, this setup, which at the time consists of usual Turing machines, can be enhanced to allow descriptions like that of x. To do this, we simply add to the machines an oracle, which can be queried to know whether some p_i halts. This is the beginnings of the theory of Turing jumps and Turing degrees which, jumping from an impossibility to another, can be extended through the many plies and creases of the mathematical language.

But again, this hierarchy of degrees hits a wall. This wall is linked to the limit in the ability of this setup a particular king of objects which are ordinals. And, as previously, it is through a more liberal use of those objects that this limitation can be overcome, by simply admitting that ordinals are "there", which then allows us to consider computation through the ordinals. But why exactly is it a "more liberal" use? This may sound especially out of place in the context of formalization. But formalization, in the sense of the crystallizing of what unfolds, is precisely an extracting and putting away–away from the mathematician–of what unfolds in the mathematical language. And this putting away is in no way a throwing away: in the formalization, we neatly organize and store parts of this deep and obscure relation that we entertain with mathematics and which is embodied by the–necessarily human–mathematical language. And this putting away naturally creates a distance between the mathematician and its object. This distance frees the mind and the day-to-day mathematical work relies on it. Given an equation, that is given a well-formalized arithmetical equality, the mathematician learned to manipulate it, to unroll it, to cut it half and so on, like the young man in *Philebus* does with the *lógos*: the mathematician tries to multiply by the conjugate, to do a change of variable, a derivation etc. and so many other operation which are, in her phenomenological relation with the equation, not grounded in the mathematical language but now in the formal language. Up until a point, a point at which the mathematical language is queried again, the mathematician is only concerned with symbols almost empty of meaning. When we cross out a common denominator in the two sides of the equation, we hardly *see* it, we just observe that the similar symbols appear on either sides. When we want to integrate a function that looks familiar, we start by trying a few usual integration tricks with, initially, as much distance to the symbols under the integral sign we have to the jigsaw piece that we haphazardly try to fit here or there.

Hence, those liberal usages of mathematical objects, ranging from the symbolical manipulation of equations to the mindless use of ordinals, are structured by the rigor of the formalization and permitted by the distance it creates. This mindless practice, meaning this doing of mathematics without thinking about mathematics, is nothing to be wary of and even a condition of the progress of mathematics: as a reflection of the mathematical language, and so projected, at a distance, in front of her, the formal language is the

laboratory of the mathematician. And so, with ordinals now neatly aligned one after the other in the ambient mathematical world, patiently waiting for us to get closer, we will see in this thesis how we can freely use them to conduct infinite computations, and how we can further unfold a fold of the mathematical language.

But most importantly, at least for this introduction, we saw how a game of back and forth takes place, from the mathematical language, unfolding in the formal hands of the mathematician, and back again in the mathematical language, to the next ply at which this unfolding temporarily halts. And so we see how the formal mathematics are surrounded on either side by the mathematical language, itself a part–maybe the most important one–of the mathematical practice that we previously tried to keep at bay. The other way around, we see again how, that which the formal language eventually uncovers comes from the mathematical language–or even, if we dare, is the mathematical language itself. And if we don't forget what we mean by "mathematical language", the possibility and the grounding of mathematics, we see that we are simply running in instructive circles. But we should also not forget how this mathematical language is itself a privilege of our human finitness, and how this game of back and forth was elucidated years ago.

> We have found a strange footprint on the shores of the unknown. We have devised profound theories, one after another, to account for its origins. At last, we have succeeded in reconstructing the creature that made the footprint. And lo! It is our own.[1]

With all of this in mind, we see how the mathematical progress–this unfolding which we now have been describing for long enough–is a walking toward the unknown, in our own footsteps. And the deeper we glance into this unknown, the more oblivious we become to those footsteps. And it is when he is unaware of those, that the mathematician confuses what he discovers with the world around him. What he discovers, and keeps discovering, is his own self and his own language. What he confuses this mathematical language with, is the world. And so, in turn, a better consciousness of this language and of its place in the mathematical practice is his access to the world.

For this reason, as much as the mathematician should not forget that a number is something we count with, a definition something we rally around and a proof something we agree upon, they should also not forget to look up, before the clouds in the sky have fully gone.

[1] A.S. Eddington, *Space, Time and Gravitation.*

Chapter 2

Summary and outline of the thesis

In this chapter, we give an overview of the mathematical context and of the main contributions of the thesis, followed by a chapter-by-chapter outline.

2.1 Context of ordinal computability

The field of ordinal computabiliy deals with higher-order computation models, where higher-order refers to, in contrast, classical finite computability. Overcoming the natural limits of finitness always involves in one way or another ordinals. Those can be used as a way to organize greater "quantity" of information as with iterated Turing jumps. They can also be used as a way to generate greater computational space, as is done with α-recursion in the constructible hierarchy.

Alternatively, ordinals can simply be seen as a part of ambient mathematics, and so, any kind of operation (well defined for limit ordinals) can be iterated through the ordinals. From there, an idea is to have a machine, say a Turing machine or a register machine, compute through the ordinals. Here, the convenient part is that, taking a finite model of computation already solves the question of defining the transition function at successor stages: if the status of the machine is well defined at some ordinal stage α, its status at stage $\alpha + 1$ is defined as in the finite case. What solely remains is to find a way to define the status–or more canonically the snapshot of the machine at limit stages. One way to do this was introduced in [HL00] by Hamkins and Lewis for the Turing machine: at a limit stage, the value of a cell is the limit superior of its previous values.

More precisely, they introduced the model of infinite time Turing machine (ITTM). An ITTM has the same structure as a three tapes Turing machine. It computes through the ordinals and at any successor stage, the next snapshot of the machine is a function of its machine code and of the actual snapshot, as done in the classical setting. At limit stage, tape heads are back on the first cells of the tapes, the machine state is some distinguished limit state and the value of any cell is set to the limit superior of its previous values. This model, as we will see in Chapter 4, has many great properties and deep links with set theory. From a philosophical point of view, it has the interest of pushing further–which

means opening further–the mathematical language in its mysterious putting-in-relation of finitness and infinitness. But this was already discussed in the introduction. From a mathematical point of view, the properties of ITTMs combined with their links with set theory makes for a great investigation and proving tool. As an example of how those properties can be harvested, Durand and Lafitte devised in [DL19] an algorithmic proof of Sacks Theorem using ITTMs. Further, generalizations of this model already proved to be helpful set theoretical tools as well, as illustrated by the proof of Sacks-Simpson Theorem of Koepke and Seyffert done in [Koe05] and using infinite Turing machines with tapes of ordinal length α.

Indeed, with this idea of limits superior (or, symmetrically, limits inferior), it is possible to define further models of infinite machines. As another example, Koepke defined in [KS06] ordinal register machines: each register contains an ordinal and at any limit stage, the value of any register is set to the limit inferior of its previous values while the active line in the limit inferior of previous active lines. Koepke also generalized the Turing machine further by allowing it to work with a tape of length On in [Koe09]. In this case, at a limit stage, the position of the lecture head is set to the limit inferior of it previous positions. Also, in [Wel00a] and in a fruitful effort toward higher-order machines, Welch came up with a generalisation of ITTMs involving a different limit rule: at a limit stage, the value of a cell is set to the limit inferior, not of all its previous values but of a precise subset of its previous value, and this subset is determined by a "rule tape". A bit more remote, also using limits inferior, Fischbach and Seyfferth designed in [FS13] a model of ordinal lambda calculus.

2.2 Summary of the main results

We see in this brief overview how a good amount of generalizations of usual finite models of computation rely on this idea of defining at a limit stage the state of the system using either limits inferior or limits superior. In this context, the object of this thesis is to study the model of infinite time Turing machines (ITTMs) developed by Hamkins and Lewis' and to envisage an alternative to the limit superior for limit ordinal stages. That is too change the limit rule according to which the value of a cells at limit stages is determined.

A notable aspect of their model of ITTM is that it is almost as simple as the usual finite model of Turing machines, and for this reason it is a very convenient laboratory for putting to the test new limit rules. Moreover, while the choices for the heads and the states at limit stages may appear somewhat canonical, the principal justification for the rule of the lim sup is actually a corroboration: with this rule, Hamkins and Lewis showed how this produces a robust, powerful and structured (we will see in what follows what is meant here by structured) model of infinite time computation. From there, the focus was on the possibility of devising limit rules that yield more powerful but equally structured models of generalized infinite Turing machines.

The approach to this question is somewhat tangent: it starts with the study of the universal ITTM. It happens to be quite straightforward to define a universal ITTM which simulates in parallel all ITTMs. The importance of this machine for the study of ITTMs appears clearly in [Wel00b] where Welch answered positively the open problem left by Hamkins and Lewis, namely whether $\lambda = \gamma$? That is whether the supremum of the ordinal for which a code is writable by an ITTM is equal to that of ordinal stages at which an ITTM halts. This is what we meant by "structured". In this work, we start by introducing a slightly more general structural equality that links Σ, defined in [HL00] as the supremum of the ordinal for which a code appears at any stage in some ITTM (*accidentally writable ordinals*) to T (capital τ) which we define as the supremum of stages at which something is written on one tape for the first time (*accidentally clockable ordinals*). We show with a proof similar to that of Welch, again heavily grounded on the existence of the universal machine, that for the ITTMs the equality $\Sigma = \mathrm{T}$ holds and that it implies the equality $\lambda = \gamma$.

Now, back to the universal machine. Its definition is only fortuitously straightforward: thanks to the limit rule of the ITTMs, that is thanks to the lim sup rule, we can simply take the code of a finite universal machine and it almost immediately yields a universal Turing machine. In other words, this construction rests on quite strong but implicit properties of the lim sup rule. And those properties more generally allow for a machine to simulate another at the "speed" it wants. Take the following example: machine A is simulating machine B for α steps and decides to check whether the output of machine B after those α steps satisfies some property P. It takes β steps to do so and it may happen that P was not satisfied. So machine A goes on with the simulation of B to look at its output at a later stage. Now, what can we say about the history of B *as simulated by* A? After the α first simulation stages, there are β steps during which the history of B is not modified; hence making this history quite different, because somehow stretched, from the true history of B. This is not an issue for the simulation but only because the lim sup rule does not yield different values when the history is stretched in such a fashion. This is one of the important properties of the lim sup rule.

Going further, in the Chapter 5, we exhibit a set of four properties satisfied by the lim sup rule that are implicitly used in the constructions of ITTM that simulate other machines. As the limit rule for a machine can be seen to rests on a function that maps histories of computations to cell values, following the definition of Welch in [Wel00a], we call the functions underlying those rules *limit rule operators* and focus our study on them. We use the letter Γ to refer to operators and write Γ-ITTM for the more general models of infinite Turing machines produced by those operators and Γ-machines for the machines generated by a given operator Γ. This set of four properties, which we call *cell-by-cell*, *asymptotical*, *stable* and *contraction-proof*, allows us to define the concept of *simulational operators*: those are operators that yield models of Γ-ITTMs in which we can explicitly design machines using the simulation of other machines in their computation.

In particular, for Γ a simulational operator, there exists a Γ-universal machine. To this we add a safeguard property called *looping stability* which rules out some pathological operators. The first result, very easy but to some extent striking, is the following.

Result 1 (Theorem 5.4.1)**.** *The* lim sup *and* lim inf *operators are the only* 2*-symbol simulational operators satisfying the condition of looping stability.*

This compels us to consider more generally n-symbol operators and with it n-symbols Γ-ITTMs. The second result, more difficult, is the following, defining for an operator Γ the constants Σ_Γ and T_Γ as was done for ITTMs.

Result 2 (Theorem 5.4.14)**.** *For a* n*-symbol simulational operator* Γ *that satisfies the looping stability condition and that can be defined by a formula of set-theory, we have that* $\Sigma_\Gamma = \mathrm{T}_\Gamma$*.*

And once again, the equality $\lambda_\Gamma = \gamma_\Gamma$ is a corollary of it. The third result of this chapter is the fact, showed through a counter-example, that the condition of looping stability is necessary for this equality to be established, i.e. this result is tight with respect to this condition. That is:

Result 3 (Theorem 5.5.1)**.** *There exists a* 3*-symbol simulational operator* Γ *that does not satisfy the looping stability condition and for which* $\Sigma_\Gamma \neq \mathrm{T}_\Gamma$*.*

Further again, in the Chapter 6: if we consider operator that can be defined by a formula of set-theory, those operator are naturally hierarchized by their level in the Lévy hierarchy. With this point of view, the lim sup operator is a Σ_2 operator. This operator is arguably as powerful as a Σ_2 simulational operator can get having in mind this result from Hamkins and Lewis:

$$L_\zeta \prec_{\Sigma_2} L_\Sigma$$

Hence we design a $\Sigma_3 \wedge \Pi_3$ 3-symbol simulational operator Γ_3. That is an operator which is defined in set theory by a formula $\varphi \wedge \psi$ where φ is a Σ_3 and ψ a Π_3. For this operator, we shorten Σ_{Γ_3}, ζ_{Γ_3} and the other constants by Σ_3, ζ_3, etc. This operator also satisfies the condition of looping stability, hence with the result of the previous chapter, the equality $\Sigma_3 = \mathrm{T}_3$ holds. Then we show how its definition induces a new way of writing reals with those machine, called K-writing, such that K_3, the supremum of the K-writable ordinals is strictly between ζ_3 and Σ_3. Eventually, we show that with this K_3 the analogous of the previous result holds.

Result 4 (Theorem 6.2.49)**.** *For the simulational operator* Γ_3*, we have:*

$$L_{\zeta_3} \prec_{\Sigma_2} L_{\mathrm{K}_3} \prec_{\Sigma_3} \prec L_{\Sigma_3}$$

This shows that this model of Γ_3-machines provides a satisfying simulational generalization of the lim sup machines with respect to power and structure.

2.3 Plan of the thesis

Chapter 1 and Chapter 2 are respectively introductory and summary chapters. Chapter 3 and Chapter 4 provide the mathematical context of this work while Chapter 5 and Chapter 6 present the two main contributions of the thesis. For the reader already familiar with the subject, Chapter 5 and Chapter 6 are written in a self-contained way.

More precisely, Chapter 3 starts with preliminaries in both computability and set theory. It starts with the most basic idea of computation, namely primitive recursion theory, and works its way toward degree theory and higher recursion theory. It ends with the introduction of hyperarithmetical hierarchy and admissible sets.

Chapter 4 introduces the starting point of this work, namely infinite time Turing machines. It presents the main results that have been established as well as the interest of those machines. It also gives a brief overview of generalizations of this model of computation in literature.

Chapter 5 presents the first main result established during this thesis. It starts by reintroducing some key concepts so that that the reader familiar with computability and set theory can skip previous chapters. It tgen shows how we can devise a wide class of model of infinite Turing machines that generalizes the ITTM and for which some of the most interesting results obtained with ITTMs still hold. More precisely, we show how the equality $\lambda = \gamma$ (as well as the stronger and new equality $\Sigma = \mathrm{T}$) can be established for models of infinite machines ruled by other rules than the usual lim sup rule. In this chapter and the next one, results that are not attributed are new results.

Chapter 6 contains the second main result. It builds on the results of the previous chapter and it introduces a model of infinite Turing machine with a $\Sigma_3 \wedge \Pi_3$ rule that again generalizes the model of ITTM and for which we prove strong links with the constructible universe, as was done for the ITTM. Namely, we show that for this model of infinite Turing machine, we can define K_3 a new constant relative to a new way of writing ordinals and such that, with ζ_3 and Σ_3 as usually defined:

$$L_{\zeta_3} \prec_{\Sigma_2} L_{\kappa_3} \prec_{\Sigma_3} \prec L_{\Sigma_3}$$

This corroborates the interest of this model of infinite machine as a generalization of the ITTM.

In Chapter 7, we briefly go through the new results of this work. To this extent, we build some momentum by also going first through the intuitions and bare ideas that lead to those results. This may a good place to start for the reader wanting more insight than hard proofs. Still carried by this momentum, we also discuss the outlooks of this work as we recall some questions left open in it.

Chapter 3

Computability and set-theoretic preliminaries

The aim of this chapter is to introduce the field of classical computability from the crossed perspectives of Turing machines and recursive functions. To do so, it stages the development of computability from the question "what does computable mean ?" For this reason, it starts with well-known objects, like primitive recursive functions, and it shows how those objects help crystallize and confront different intuitions about computation– and with it, how progress in mathematics is often made along a back-and-forth between formalism and intuition. In the same way, it mostly aims at giving intuitions about the usual objects and the proofs rather than formally exposing those.

It then continues to introduce the question of higher recursion theory and with it some of the relevant objects for next chapters. Those objects are mostly set theoretical. They include ordinals, Gödel's constructible universe, set hierarchies and admissible sets.

Definitions and results which are not affiliated are considered to be folklore. All the results on computability, arithmetic and hyperarithmetic can be found in [Rog87]. Those in admissibility theory and α-recursion can respectively be found in [Bar75] and [Sac90].

3.1 Computability theory and Turing machines

Computability theory stems from the following question: "what does it mean for a function (on the natural numbers) to be computable[1]?" Such a question acts as the frontier between, on one side, what could be called the outermost region of the proto-mathematical thinking and on the other side, the origin of the formal mathematical development. Indeed, while this question is inquiring toward the meaning of "computable" with respect to functions, it can only do so and make sense with an already implicit understanding of the term "computable". And this is not contradictory: here "computable" and "computable" are two different words as they are part of two different languages, or regions of thinking.

[1] Or "effectively calculable" as was used by Church.

As such, this question is a fundamental meeting point and a zone of dialogue for those two languages[1].

From there, answering this question is akin to reenacting the dialogue between those two languages, that is to let one, in which we have a pre-concept of the notion of computability, speak through the other. What does the natural or proto-mathematical language says? It says, among other things[2], first, that the simplest functions we know are naturally computable and that any function simply made of two computable functions should be computable as well. And second, it says that if there is a deterministic and automatic procedure that *computes* the function, then it clearly is computable. Let us see how this first affirmation translates.

3.1.1 Primitive recursion theory

What are simple functions over integers? The simplest one may be the constant function, and even more the constant function that maps every natural number to 0. Then, comes the identity function. As we may want to consider n-ary functions, this yield different identity functions, also called projections, each one returning the k^{th} parameters for some k. And the successor function, that maps x to $x+1$ is as well simple enough to be considered computable. So begins the definition of recursive functions.

Definition 3.1.1 (Primitive functions)**.** A *primitive function* is one of the following functions from $\mathbb{N}^n$ to $\mathbb{N}$ for some $n > 0$:

- $C_k^n(x_1, \ldots, x_n) = k$, the constant function with n argument that always return k.
- $P_k(x_1, \ldots, x_n) = x_k$, the identity function with n argument that always return its k^{th} argument for $1 \leqslant k \leqslant n$.
- $S(x) = x + 1$, the successor function.

From there, how does these function interact with each other? The composition of two computable functions is expected to be computable. From there, it it possible to define the function $S_k(x)$ such that $S_k(x) = x + k$:

$$S_k = \underbrace{S \circ S \circ \ldots \circ S}_{k \text{ times}}$$

Now, how is defined a similar function w.r.t. multiplication? That is a function such that $S_k(x) = x \cdot k$. Observe that in the natural definition of multiplication, as an iterated addition, there is a fundamental leap. When we say that $x \cdot k$ is the result of the function S_k iterated x times on 0, there is an ontological shift or even an ascension: x is not

[1]The "dialogue" is not the *lógos* that takes place between two parties. From the greek διάλογος, it is literally the *lógos* that goes through (*dia-*) and as such separates.

[2]Another thing it says is that: "if we can write down a function in some natural way, then it must be computable." This may be seen as the "going through" that led to the development of lambda calculus.

anymore a simple natural numbers that functions can act upon, it is lifted into the realm of functions and may mingle with them; which is clearly needed to speak of the x^{th} iteration of S_k. This ontological shift is encompassed by a recursion scheme that can be seen, on the functions side, as mapping x and S_k to $S_k^x = S_k \circ S_k \circ \ldots \circ S_k$, where S_k is iterated x times. That is, formally:

Definition 3.1.2 (Primitive operators)**.** The *primitive operators* acting on functions over the natural numbers are the following operators:

- The composition operator $\circ$. Given a n-ary function f and n m-ary functions $g_1, \ldots, g_n$, $f \circ (g_1, \ldots, g_m)$ is a m-ary function defined as:

$$f \circ (g_1, \ldots, g_n)(x_1, \ldots, x_m) = f(g_1(x_1, \ldots, x_m), \ldots, g_n(x_1, \ldots, x_m))$$

- The primitive recursion operator. Given g a n-ary function and h a $n+2$-ary function, we define $f = \rho(g, h)$, the recursion operator applied to g and h as the $n+1$-ary function defined as:

$$\begin{aligned} f(0, x_1, \ldots, x_n) &= g(x_1, \ldots, x_n) \\ f(S(y), x_1, \ldots, x_n) &= h(y, f(y, x_1, \ldots, x_n), x_1, \ldots, x_n) \end{aligned}$$

With those two operators, we can define the class of primitive recursive functions.

Definition 3.1.3 (Primitive recursive functions)**.** The class of *primitive recursive functions* is the smallest class of total functions over the natural numbers containing the primitive functions and close under the primitive operators.

Now, as this mathematical definition stems from the natural language, it should be sound with respect to it. That is, the previous computable function are indeed computable w.r.t. the natural pre-concept of computability. But is it complete? That is, does any function that we deem computable is indeed computable in this formal sense? Let us keep this question on hold and see how it compares with the other aspect of computability, namely the operational or automatic aspect of computability, which encompasses the deterministic, finite and effective aspect of computability.

3.1.2 Turing machines

If we want to exploit this operational aspect of computability, which could to some extent be simply called its algorithmic aspect, there needs to be a solid ground on which to develop this point of view. But what exactly needs to be grounded? And where should this ground originates from?

As we have already seen, what needs to be grounded is the development of the mathematical notion of computabilty and its ground is the pre-conceptual notion of computability. But here, more precisely, we are inquiring into one fundamental aspect of computability, namely what we called the operational aspect. Hence, the mathematical development of this aspect must be grounded on the pre-mathematical intuition that links those two terms, operationnal and computability. And the link between those can be embodied by the idea of the machine: the machine is the very thing that acts in a deterministic and automatic kind of way. And understood in this way, the idea of machine is naturally part of the pre-mathematical world. From this, we can piece everything together: this natural idea of machine will be grounding its mathematical definition and it will carry over in the mathematical realm the relation between the notions of operational and computable.

We can imagine many different mathematical definitions of models of machines. There is however a trade-off when it comes to the complexity of the model: a simple model is more easily deemed close to our intuitive idea of a machine and, from the mathematical point of view, it is easy to prove general results on it. On the other hand, a more complex model makes for simpler description of its machines (or rather of the code of its machines). Given this observation, the beginnings of computability naturally calls for simple machine models that may be enhanced latter.

Alan Turing introduced his model of Turing machine (initially "a-machine", for automatic machine) in [Tur+36]. Using it, he was able to give a negative answer[1] to the Entscheidungsproblem: "Is there an algorithmic procedure with which we can decide, given any logical statement, whether this statement is true ?" As such, answering this question shows how this formal definition of machine helps link the operational aspect of the question (can we provide an algorithm for this task?) and its computable aspect (is the underlying function a computable function?). The proof of Turing involves showing that there exists a universal Turing machine, that is a machine that can simulate other machines. We will come back to this point as it is of cardinal importance for the development of infinite time Turing machines.

In it most simple form, a Turing machine is constituted of a tape divided into cell which span infinitely to the right. Each cell contains a symbol and there is a head moving on the tape that can read the symbol of the cell over which it currently is. There is moreover a register that stores a state of the machine, itself belonging to a finite set of state. At each step, reading the symbol under the head and the state in the register and according to a given set of transitions rules, the cell under the head is written over (possibly writing the same symbol that was just read), the head either moves to the left or

[1]Once again, this question holds a peculiar position between the natural and the mathematical language. As such, it cannot be answered from the sole mathematical point of view as it involves terms from the natural language. There needs to link back the mathematical development from the intuitive concepts it stems from. And this does the Church-Turing thesis suggesting that such defined computable functions actually are what the intuitive idea of "computability" encompasses. With this (hypo)thesis, it is possible to close the loop from a region to another and to answer the question.

to the right and the register state is updated (again possibly to the same state). Formally:

Definition 3.1.4 (Turing machine)**.** A *Turing machine* is a 6-uple $\langle \Sigma, b, Q, q_0, \delta, F \rangle$ such that:

- Σ is the finite alphabet of the machine
- $b \in \Sigma$ is a blank symbol
- Q is the finite set of states
- q_0 is the initial state
- $F \in Q$ is a set of final states
- $\delta : \Sigma \times Q \to (\Sigma \setminus \{b\}) \times Q \times \{L, R\}$ is the transition function.

Given a machine, we may want to describe the value of all of its variable aspects, that is of its tape content, its state and its head position. Such a description can be seen as (and will be called) the snapshot of a machine, capturing at some point the complete status of the machine.

Definition 3.1.5 (Snapshot of a machine)**.** The *snapshot* of a Turing machine $\mathcal{M} = \langle \Sigma, Q, q_0, \delta, F \rangle$ is a 3-uple $\langle x, q, k \rangle$ where x describes the content of the tape, q the state of the machine and k the integer position of the head. A snapshot is final if $q \in F$.

Then, observe that the transition function is naturally lifted to a function on snapshots.

Definition 3.1.6 (Successor snapshot)**.** Given a Turing machine $\mathcal{M} = \langle \Sigma, Q, q_0, \delta, F \rangle$ and a snapshot of the machine $s = \langle x, q, k \rangle$ such that $q \notin F$, the successor snapshot s' is the only snapshot produced by applying to it the transition function on the cell k which contains the symbol $x[k]$. That is, writing $\delta(x[k], q) = (a, q', D)$

$$s = \langle x', q', k + d \rangle \text{ with } \begin{cases} x'[k'] = x[k'] \text{ for } k' \neq k \\ x'[k] = a \\ d = 1 \text{ if } D = R \\ d = -1 \text{ if } D = L \text{ and } k > 0 \\ d = 0 \text{ if } D = L \text{ and } k = 0 \end{cases}$$

Definition 3.1.7 (Run of a Turing machine)**.** Given a Turing machine $\mathcal{M} = \langle \Sigma, Q, q_0, \delta, F \rangle$, the *run* of this Turing machine with input x is a possibly infinite sequence (s_i) of snapshots such that, writing $x^\frown b$ for x padded with infinitely many blank symbols to the right if x is finite and x itself otherwise:

- $s_0 = \langle x^\frown b, q_0, 0 \rangle$

- For all i, if s_{i+1} exists, then it is the successor snapshot of s_i. It does not exist if and only if s_i is a final snapshot.

We say that a machine halts when run with input x if its sequence of snapshots is finite. In this case, writing $s_n = \langle y^\frown b, q, k\rangle$ for the last snapshot of the sequence, we say that the machine computed for n steps and that its output is y.

Remark 3.1.8. *Observe that with this definition, the content of the tape x may either be of the form $w \in \Sigma^*$ and then followed by infinitely many blanks (that is b's), or of the form $w \in \Sigma^\omega$, that is with w being an infinite word built from the alphabet Σ. Until Section 3.2, we only work with natural numbers as input, hence that are encoded as finite words $w \in \Sigma^*$. In Section 3.2 we will see the interest of considering reals (that is infinite words) as input.*

Definition 3.1.9 (Turing computable function)**.** A partial function $f : \mathbb{N}^k \mapsto \mathbb{N}$ is Turing computable it there is a machine m such that: for $x_1, \ldots, x_n$, $f(x_1, \ldots, x_n)$ is defined and $f(x_1, \ldots, x_n) = y$ if and only if the computation of m with input $\langle x_1, \ldots, x_n\rangle$ halts and outputs y.

3.1.3 Recursive functions and Turing machines

Now, given this definition, we would like to say that the computable functions are exactly the functions that are Turing computable, that is computable by a Turing machine. Then we may wonder whether this new conceptualization of computation matches with the one defined in Section 3.1.1. And an important discrepancy appears: it is easy to see that primitive recursive function all are total whereas many Turing machines yield partial function. Indeed, if on some input x the Turing machine does not halt, the function it describes is a partial function and is undefined on x. From there stems two questions, each on one side on the frontier: Should a computable function be total? And do all total Turing computable functions are primitive recursive functions?

While tedious to show, it is plausible and even true that Turing machines can compute primitive recursive function. On the other hand, it appears harder at first sight, and rightly, to simulate a (necessarily total) Turing machine with the primitive recursive functions and operator. Indeed, what would be a direct way to do this? It would be to encode (finite) computations of Turing machine using natural numbers and, for a given Turing machine, to devise a primitive recursive function that given some input, produces the code for the computation of the machine with this input.

More formally, it would rest on the following observation: for m a Turing machine that halts on all input, writing $m(x)$ for the input of m when computing from x, $m(x) = y$ if and only if there exists a computation C of m that starts with the snapshot $\langle x^\frown b, q_0, 0\rangle$ and that halts with the snapshot $\langle y^\frown b, q_f, k\rangle$ where $q_f \in F$ and for any k. Hence, if the unique computation C of m on input x can be (uniformly in x) produced by a primitive

recursive function with x as argument, the output y being itself (naturally) encoded in C, the function induced by the machine is easily seen to be primitive recursive.

And it is possible to devise a primitive recursive function that, given some argument C, is equal to 1 if and only if C is the code for a computation of m. However, the problematic part is the finding or the producing of such a C. We indeed see how the previous observation introduces a quantification on all finite computation ("if and only if *there exists* a computation...") This equates for a primitive recursive function to be able to search through the natural numbers for some C satisfying some primitive recursive property. On the other hand, looking at the definition of primitive recursive function, there is not clear way, and actually none, for those to look for a natural number that may be arbitrarily big.

And this resonates with our first question: if we deem such quantification, akin to a search through the natural numbers, computable, then we should allow for partial computable functions. Indeed, it is natural to imagine that the number the functions searches for may not exist and so that it will search for infinitely long and consequently it will never output anything. This brief informal walk on the frontier shows how the mathematical fleshing out of those intuitive notions echoes back and helps probe those ideas that were not yet in words. We summarize this in a more formal way.

Proposition 3.1.10. *There is a total Turing computable function that is not a primitive recursive function.*

Sketch of proof. Observe first that the description of a primitive recursive function is finite and hence that the primitive functions may be effectively enumerated. Let (f_i) be such an enumeration. We can choose f_i so that the function $i \mapsto f_i$ is computable with a Turing machine. In particular, by an usual diagonal argument, the following function is Turing computable and total (as the primitive recursive functions are total), but not primitive recursive:

$$D : i \mapsto f_i(i) + 1$$

□

To bridge this gap between Turing computatble functions and recursive functions, we need to add another operator that will let the recursive function search through the natural numbers.

Definition 3.1.11 (Minimization operator)**.** The minimization operator μ is defined as follow: given a $n+1$-ary total function f, the n-ary partial function $\mu(f)$ is defined as:

$$\mu(f)(x_1, \ldots, x_n) = \text{ the least } z \text{ such that } f(z, x_1, \ldots, x_n) = 0 \text{ if it exists}$$

When no such z exists, $\mu(f)(x_1, \ldots, x_n)$ is undefined.

Definition 3.1.12 (Recursive function)**.** The class of *recursive functions* is the smallest class of partial functions containing the primitive functions and closed under the primitive operators as well as the minimization operator.

Proposition 3.1.13. *The recursive functions are exactly the Turing computable functions on the natural numbers.*

Sketch of proof. As mentioned, the primitive functions are clearly Turing computable and the primitive operators correspond to constructions on the Turing machines that mimic those operators for Turing computable function. Further, given a total Turing computable function, as we can simulate it, we can design a machine that compute the minimization operator applied to this function. This is enough to show, in a brief way, that recursive functions are also Turing computable.

Conversely, the idea is to show we can define a recursive function T such that:

$$T(C, m, x) = 0 \iff C \text{ describes a halting run of } m \text{ with } x \text{ as input}$$

Thanks to the minimization operator, this yields a function $\mu(T)$ that maps m and x to a least run of m with x as input. As this run is unique, it is then possible, given the function $\mu(T)$, to define a function that extracts the output from a run and returns it; that is that returns y, the output of the computation of m on input x. □

This converging of the formalization of different proto-mathematical views[1] of the idea of computability is a strong corroboration of the (necessarily extra-mathematical) Church-Turing thesis, that states the such defined computable functions are exactly the intuitively computable functions.

Now, with the computable functions well delimited, we can prove results on this class of function. First, there are countably many such functions and so they their collection can be written $(\varphi_i)_{i \in \mathbb{N}}$. And from there, we can state the s-m-n theorem as well as Kleene's fixed point theorem which we will need later.

Definition 3.1.14. For two recursive functions f and g, we write $f = g$ when those are defined for the same natural numbers and when they moreover match on this subset.

Definition 3.1.15. For a recursive function f and i such that $f = \varphi_i$, we say that i is an index of f.

Theorem 3.1.16 (S-m-n theorem)**.** *For any integers n and m, natural numbers $x_1, \ldots, x_n$ and for φ_i a recursive function of arity $m+n$, there exists a recursive function φ_j of arity*

[1] As mentioned, there is another way to look at this idea, namely lambda calculus, and this formalization also yields the same class of computable functions.

m such that the functions

$$\begin{cases} y_1, \ldots, y_m \mapsto \varphi_i(x_1, \ldots, x_n, y_1, \ldots, y_m) \\ y_1, \ldots, y_m \mapsto \varphi_j(y_1, \ldots, y_m) \end{cases}$$

are equal in the sense of the previous definition. Moreover, the function S_n^m which maps any $x_1, \ldots, x_n$ and i to some j satisfying previous conditions is itself recursive.

Idea of the proof. This theorem of Kleene says that, taking the point of view of Turing machines, it is possible to "hardcode" some parameters in a machine and, moreover, that a Turing machine, given those parameters and the code for another Turing machine can itself do it. From there the proof is technical but straightforward and can be found in [Dav58]. □

Theorem 3.1.17 (Kleene's fixed point Theorem)**.** *Let $f(x, y)$ be a recursive function with two arguments, then there exists some recursive function φ_i of index i such that for all y for which $f(i, y)$ is defined*

$$\varphi_i(y) = f(i, y)$$

Idea of the proof. The proof relies on the s-m-n theorem which allows to define a recursive function that map an index to the index of another well-thought function. With this function and diagonalization, that is applying a function to its own index, it is possible to show that for any total recursive function F there is a fixed-point i in the sense that

$$\varphi_i = \varphi_{F(i)}$$

Then, the theorem is proved by applying this result to the function F which maps i to the index of the function $y \mapsto f(i, y)$ (this F is clearly total and it is recursive by the s-m-n theorem). A detailed proof can be found in [Rog87]. □

3.2 Turing degrees and the constructible universe

Now, we could question further: what is a non-computable function? We wrote (φ_i) for a natural enumeration of the recursive, that is computable, functions. Given a computable function φ_i and in virtue of the equivalence with Turing computable functions, we write $\varphi_i(x)\downarrow$ when the computation of φ on x halts, $\varphi_i(x)\downarrow y$ when we want to add that it outputs y and $\varphi_i(x)\uparrow$ when it does not halt. It is well known that the halting problem is undecidable; that is the halting function

$$Halt : i \mapsto \begin{cases} 1 \text{ if } \varphi_i(0)\downarrow \\ 0 \text{ if } \varphi_i(0)\uparrow \end{cases}$$

is not computable.

This first dent in the horizon of computability actually draws attention to an extremely fine and potent hierarchy. Indeed, we could say that the halting function is "computationally harder" than the computable functions, for the simple reason that the first is actually not computable while the others are computable. Then, is this function the simplest among the computationally harder functions? And are there even functions computationally harder that the halting problem? But then, what would be harder than impossible? To answer those questions, the idea is simply to allow this impossible, to jump in a new world where that halting function becomes computable and to see whether some functions still are non-computable. We fix a computable bijection $\langle \cdot, \cdot \rangle$ between $\mathbb{N}^2$ and $\mathbb{N}$. Then, observe that any function f on the natural numbers can be coded by an element x of $\{0,1\}^{\mathbb{N}}$ (which we more conveniently write ${}^{\omega}2$): for all i and j, we ask that

$$f(i) = j \longleftrightarrow x[\langle i, j \rangle] = 1$$

where $x[k]$ denotes the k^{th} bit of x. A $x \in {}^{\omega}2$ will be called a real and the set of reals will be naturally identified with $\mathcal{P}(\omega)$. We write h for the real encoding the function $Halt$. Consider a Turing machine that computes with h given as infinite input: this machine can very easily answer the halting problem, that is it can compute $Halt$. Hence, in a self-explanatory way, we could say that $Halt$ is h-computable. But then we can define a new halting function that now deals with h-computable function. And we can show that it is in turn not h-computable; which then shows how the "computational difficulty" levels may stack up one on the other. More formally, we will define oracle computations and a jump operator.

Definition 3.2.1 (Oracle computation)**.** *An oracle machine with real x as an oracle* is a Turing machine with one additional tape (with its own lecture head as other tapes) on which x is written. We write φ_i^x for the function induced by the i^{th} Turing machine with x as an oracle.

Definition 3.2.2 (Computable real)**.** A real x is *computable from a real y*, or *y-computable*, if there is a Turing machine i computing with y as an oracle and such that for all $n \in \omega$

$$\begin{cases} \varphi_i^y(n)\downarrow 1 \longleftrightarrow n \in x \\ \varphi_i^y(n)\downarrow 0 \longleftrightarrow n \notin x \end{cases}$$

Definition 3.2.3 (Recursively enumerable real)**.** A real x is *recursively enumerable from a real y*, or y-r.e., if there is a Turing machine i computing with y as an oracle such that for all $n \in \omega$

$$\varphi_i^y(n)\downarrow \longleftrightarrow n \in x$$

That is, $x = \mathrm{dom}(\varphi_i^y)$.

Proposition 3.2.4. *Equivalently, a y-r.e. real is the image of a y-recursive function.*

Proof. This is straightforwardly proved with dovetailing: by simulating in parallel more and more copies of a Turing machine with different natural numbers as input, it is possible to enumerate those inputs on which it halts. We should however point out that this works because a Turing machine can enumerate $\mathbb{N}$. □

Definition 3.2.5 (Turing jump)**.** Let x be a real. The *Turing jump* of x, written x', is the real defined as follow:

$$x' = \{i \mid \varphi_i^x(0)\downarrow\}$$

Further, we inductively define for any natural number n:

$$\begin{cases} x^{(0)} = x \\ x^{(n+1)} = (x^{(n)})' \end{cases}$$

With the previous definition of the real h, it is clear that, writing 0 for the real coding the empty set, h is $0'$-computable. But $0'$ is h-computable as well. Hence $0'$ encodes the halting problem. Further, the halting problem for the h-computable functions is encoded by the real $h' = 0''$. They are indeed equal being the jumps of two "as powerful" reals. This way of identifying reciprocally computable reals leads to a natural quotient definition.

Definition 3.2.6 (Turing reducibility and Turing equivalence)**.** We say that the real x is *Turing reducible* to the real y, which we write $x \leqslant_T y$ when x is y-computable. When both $x \leqslant_T y$ and $y \leqslant_T x$ we say that x and y are *Turing equivalent*, which we write $x \equiv_T y$. Eventually $x <_T$ means $x \leqslant_T y \wedge x \not\equiv_T y$.

Proposition 3.2.7. *For a real x, $x <_T x'$.*

Proof. With the same proof as for the halting problem, we can show that x' is not computable from x. Then, to compute x from x', observe that given some $i \in \omega$, the code for a machine with x as oracle that halts if and only $i \in x$ is computable from 0. Hence, with x' as oracle, it is enough to look up in x' whether this machines halts to know whether $i \in x$. □

The Turing equivalence is easily an equivalence relation. We define the Turing degrees as classes of equivalence of this relation.

Definition 3.2.8 (Turing degrees)**.** The Turing degrees are the element of ${}^{\omega}2/\equiv_T$. That is the equivalent classes induced by the Turing equivalence. We use bold Latin letters to denote Turing degrees: $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}\ldots$, and $\mathbf{0}$ for $[0]$.

Definition 3.2.9 (Recursively enumerable Turing degrees)**.** A *recursively enumerable Turing degree* is the degree of a r.e. real.

Proposition 3.2.10. *Let x and y be two reals such that $x \equiv_T y$. Then, $x' \equiv_T y'$.*

Proof. We show that y' is computable from x'. To do this, we need to design a machine with x' as oracle such that given some machine index it decides whether $\varphi_i^y\downarrow$. By hypothesis, we know that y is x-computable. Hence, there is a computable code i' for a machine with oracle x such that:

$$\varphi_i^y\downarrow \longleftrightarrow \varphi_{i'}^x\downarrow$$

Hence computing this code i' and using the oracle x', a machine can check whether $\varphi_{i'}^x\downarrow$ and with it decide whether $\varphi_i^y\downarrow$. □

The previous proposition allows us to lift the jump operation to Turing degrees and to write for example $\mathbf{0'} = [0']$. With this, we can reformulate with reals (or degrees) the previous questions regarding computationally hard function: is $\mathbf{0'} = [0']$ a smallest degree above $\mathbf{0}$? Is $\mathbf{0'}$ a greater degree? As seen, the answer to the second question is negative as $\mathbf{0'} <_T \mathbf{0''}$. As for the first question, it was known as the *Post problem* and answered independently and negatively by Friedberg and Muchnik. It won't be used further in the thesis but we nonetheless provide a proof as it can be done in a relatively self-sufficient way. The idea is to build iteratively two r.e. and incomparable reals, keeping track, seen as requirements, of the machines that we can ensure won't compute one real with the other. It is a bit contrived as along the construction, those requirement may be injured, in which case they may need to be satisfied later.

Theorem 3.2.11 (Friedberg–Muchnik theorem)**.** *There exist two incomparable r.e. degrees* $\mathbf{a}$ *and* $\mathbf{b}$*.*

Idea of the proof. We want to construct two r.e. reals x and y such that $x \not\leqslant_T y$ and $y \not\leqslant_T x$. That is, we want that for all index j, there is n and m such that:

$$\begin{cases} \varphi_j^x(n)\downarrow 1 \nleftrightarrow n \in y \\ \varphi_j^y(m)\downarrow 1 \nleftrightarrow m \in x \end{cases} \tag{3.1}$$

which mean that machine j with oracle x does not computes y and that with oracle y it does not compute x.

The proof can get quite technical while the underlying idea is simple enough. Starting with $x_0 = y_0 = 0$, x and y are built iteratively. That is, the algorithm will be adding natural numbers, one after the other, to x and y in order to satisfy the equation (3.1) for all index i. As the algorithm will only be adding elements to x and y, those will clearly be r.e.

Now, when should the algorithm add elements to x and y? At some stage i of the algorithm, we write x_i and y_i for the reals that have been obtained up to this point. Then, taking the point of view of Turing machines, suppose that machine j with oracle x_i and some input n halts and outputs anything other from 1. Then if $n \notin y_i$, we can simply define set

$$\begin{aligned} x_{i+1} &= x_i \\ y_{i+1} &= y_i \cup \{n\} \end{aligned}$$

which ensures that x_{i+1} and y_{i+1} satisfy the first of equation of (3.1) with respect to index j.

However, and the difficulty lies here, we are working now with y_{i+1} instead of y_i. And nothing tells us, for any index k and m, whether the computations $\varphi_k^{y_{i+1}}(m)$ and $\varphi_k^{y_i}(m)$ have anything to do with one another. That is, it is possible that for some index k, equation (3.1) holds for (x_i, y_i) but not anymore for (x_{i+1}, y_{i+1}).

To get around this difficulty, we observe that if the computation $\varphi_k^{y_i}(m)$ halts in less than n steps, where n was the natural number added to y_i to form y_{i+1}, then so does $\varphi_j^{y_{i+1}}(m)$. From there, the idea is to rely on this observation. That is, the algorithm will search for great enough such n's, that is for inputs n greater than the halting time of lower indexes (or of the same index for the other orcacle) and for which the machine i with x_i (or y_i) as oracle halts on anything else than 1. Once such a n is found, adding it to y_i (or x_i) does not injure the requirement (i.e. equation (3.1)) for lower indexes (or for the same index with the other oracle).

Obviously, for a given i, our algorithm can't be blindly looking for an input such that machine i halts (with x_i or y_i as oracle) as this would amount to solving the halting problem. So as always, the idea is to proceed by dovetailing: simulating i steps of the first i machines, $i+1$ steps of the first $i+1$ machines (with great enough inputs, as described in the previous paragraph) and so on. It is important to observe that this way of doing things will involve restarting the algorithm at some point: when, at stage i, equation (3.1) is not yet ensured for some machine $j < i$ and if this machine is just seen to be halting with oracle x_i and some input n (great enough) and output different from 1, we want to add n to y. However adding this n to y may change the computation of machines $k > j$ (but, as seen, not of previous halting computations). And so, the algorithm simply goes back to stage $j+1$ with

$$\begin{cases} x_{j+1} = x_i \\ y_{j+1} = y_i \cup \{n\} \end{cases}$$

and continues. It is clear that stage 0 can only be restarted twice (once for x_0 and once for y_0) and that any stage may only be rerun because either it is restarted or a lower stage

is restarted–and so it may be rerun only a finite amount of times. This ensures that, as required, the algorithm enumerates x and y. □

Corollary 3.2.12. *There exists some degree* $\mathbf{a}$ *such that* $\mathbf{0} <_T \mathbf{a} <_T \mathbf{0}'$

Proof. As the previously defined x and y are r.e., they are $0'$-computable. If $0'$ was computable from one of them, they would be comparable. And as none is the computable from the other, *a fortiori* none is computable from 0. □

This argument can be generalized to show that there is another r.e. degree between any two comparable r.e. degree. And on the other side of $\mathbf{0}$, the degrees $\mathbf{0}^{(n)}$ for $n > 0$ produce a non-collapsing hierarchy. Then, what is next? The next thing is the closest thing [1] beyond the previous ones. In our case, it would be the lowest degree $\mathbf{a}$ such that for all n, $\mathbf{0}^{(n)} <_T \mathbf{a}$. In other words, we would like to extend this hierarchy through its least upper bound which would yield a unequivocal definition of $\mathbf{a}$. And from there, continue with $\mathbf{a}'$, $\mathbf{a}''$ and take again the l.u.b. at each limit stage of the iteration. But this is built on the assumption that there is a l.u.b. for those sets of degrees. And this happens to be false! This is the exact pair theorem of Spector (see [Rog87]) that states that given an ascending sequence of degree $\mathbf{a}_0 < \mathbf{a}_1 < \ldots$, there are actually two incomparable degrees $\mathbf{b}_1$ and $\mathbf{b}_2$ bounding it and such that for all degree $\mathbf{c}$, $\mathbf{c} < \mathbf{b}_1 \wedge \mathbf{c} < \mathbf{b}_2 \implies \exists n\, \mathbf{c} < \mathbf{a}_n$. And so the pair $\{\mathbf{b}_1, \mathbf{b}_2\}$ is at the same time a dent in what could have been the linearity of the order (however we already knew that there existed incomparable degrees), but also a dent in what could have been its lattice-like structure, as this pair $\{\mathbf{b}_1, \mathbf{b}_2\}$ can't have a g.l.b. Moreover, for our purpose, the sequence $(\mathbf{a}_n)$ can't have a l.u.b. either, since it would be less than both $\mathbf{b}_1$ and $\mathbf{b}_2$ and as a consequence not an upper bound. Hence this issue steers the approach to extend this sequence toward a more constructive one. It is indeed fairly easy to construct a real that encodes all the $0^{(n)}$, consider for example:

$$0^{(\omega)} = \left\{ \langle i, j \rangle | i \in 0^{(j)} \right\}$$

And we can see that for any n, $0^{(n)} <_T 0^{(\omega)}$ as given $0^{(\omega)}$ as oracle a machine can compute $0^{(n)}$. But observe that it can do so, only because this encoding is "computationally manageable". Had we used a voluntarily pathological encoding to define $0^{(\omega)}$, a machine might not have been able to retrieve $0^{(n)}$. On the other hand, it is possible to define computable encodings that allow to store even "more" information and to grow this hierarchy further. To do this in a formal setting, we need to define computable ordinals.

3.2.1 Ordinals

Definition 3.2.13 (Well-order)**.** A well-order on a set E is a total order that does not admit infinite strictly decreasing sequences.

[1] This appears more clearly in the French "prochain" and even more in the German "nächster", translation of "next" which is also simply the superlative of "nah", the translation of "close".

Equivalently, well orders may be defined as total orders on some set E for which every non-empty subset of E admits a smallest element. The first definition may be seen as a top-down definition: it says that starting from any element in the well order we can only go down for finitely many steps. The second definition is rather a bottom-up definition: start with E itself, as its well-ordered, there is a smallest element e_0. Continue with $E - \{e_0\}$: if it is not empty, it has in turn a smallest element e_1 with $e_0 \prec e_1$. Then, while E is not exhausted, there is a least e_2 such that $e_0 \prec e_1 \prec e_2$ and so on. Then, this reasoning produces an infinite increasing sequence $(e_i)_{i\in\omega}$ and as $E - \{e_i \mid i \in \omega\}$ has a least element if E was not exhausted, this sequence has a l.u.b. that we can write e_ω and that can be used to continue further this well-ordered chipping away of E until E has been exhausted. We see how this definition allows us to count through E along its well-order. This "counting through" is the basis of transfinite induction and will be used frequently by the algorithm of infinite Turing machines.

Example 3.2.14.

- The usual order on the natural numbers is a well-order.
- The usual orders on $\mathbb{Z}$ or $\mathbb{R}$ are not well-order as there are subsets (actually $\mathbb{Z}$ or $\mathbb{R}$ themselves) which do not admit a smallest element.
- However, $\mathbb{Z}$ is well-ordered by the order $\prec$ defined as

$$0 \prec 1 \prec 2 \prec \ldots \prec -1 \prec -2 \prec \ldots$$

The previous example shows that a given set can have both well-orders and orders which are not well-order. We see how, when studying ordered sets, we may to some extent be more interested in the order than in the underlying set.

Consider now the following two well-orders:

$$\begin{cases} 0 \prec 1 \prec 2 \prec \ldots \prec -1 \prec -2 \prec \ldots \\ 0 \prec 2 \prec 4 \prec \ldots \prec 1 \prec 3 \prec \ldots \end{cases}$$

The first one is an order on $\mathbb{Z}$ while the other is on $\mathbb{N}$. It is clear that those are isomorphic and we may want to identify such ordered sets to further reduce the importance of the underlying sets.

Definition 3.2.15. Two ordered sets $(E, \prec)$ and $(F, \lhd)$ are isomorphic when there is a bijective function $f : E \to F$ such that for any $x, y \in E$

$$x \prec y \longleftrightarrow f(x) \lhd f(y)$$

Further, we also observe that any well-ordered set can be truncated after some element and that the resulting set is still well-ordered. This defines initial segments and will let us

conveniently compare well-order up to isomorphism. The following definition and result will let us define the class of ordinals.

Definition 3.2.16 (Initial segment)**.** Let $(E, \prec)$ be a well-ordered set and $a \in E$. With $E_{<a} = \{x \in E \mid x \prec a\}$, the well-ordered set $(E_{<a}, \prec|_{E_{<a}})$ is called the initial segment of E below a.

Proposition 3.2.17. *Let $(E, \prec)$ and $(F, \lhd)$ be two isomorphic ordered sets and f a bijection between those. For any $a \in E$ and with the notation of previous definition, the ordered sets $(E_{<a}, \prec|_{E_{<a}})$ and $(F_{<f(a)}, \lhd|_{F_{<f(a)}})$ are isomorphic.*

Proof. Simply consider f restricted to $E_{<a}$. □

Definition 3.2.18 (Ordinals)**.** An *ordinal* is an isomorphism class of well-ordered sets. Given a well-ordered set $(E, \prec)$, the ordinal corresponding to $[(E, \prec)]$ is called the *order type* of the set. We write On for the class of all ordinals.

Definition 3.2.19 (Order on ordinals)**.** The class On is itself ordered by the following order. For α and β ordinals, we write $\alpha < \beta$ when an ordered set of order type α is isomorphic to an initial segment of an ordered set of order type β. This yield a well-defined partial order in virtue of Proposition 3.2.17.

Further, we can show that this order is actually a well-order on On.

Proposition 3.2.20. *The class On is well-ordered.*

Proof. We need to show that the order defined in Definition 3.2.19 is total and does not admit infinite strictly decreasing sequence.

For the first requirement, consider two ordinals $[(E, \prec)]$ and $[(F, \lhd)]$. We build an isomorphism g between initial segments of E and F by counting through them in parallel. With e_0 and f_0 the smallest element, resp. of E and F we set $g(e_0) = g(f_0)$. Then, we do the same with e_1 and f_1 the smallest elements of $E - \{e_0\}$ and $F - \{f_0\}$ until one of them is empty. When this occurs, we either have an isomorphism between $[(E, \prec)]$ and $[(F, \lhd)]$ (if both were empty at the same point), or we have an isomorphism between one of the ordered set and an initial segment of the other.

Suppose now there is an infinite strictly decreasing sequence of ordinals:

$$[(E_0, \prec_0)] > [(E_1, \prec_1)] > [(E_2, \prec_2)] > \dots$$

Then, for any $i > 0$, E_i is isomorphic to an initial segment of E_0. For $i > 0$, we write x_i the element of E_0 such that E_i is isomorphic to the initial segment of E_0 below x_i. By proposition 3.2.17, for $i < j$, we have $x_j \prec_0 x_i$ and this yield an infinite strictly decreasing sequence $(x_i)_{i>0}$ in E_0 which contradicts the fact that it was well-ordered.

□

The set of every ordinals smaller than a given ordinal α is itself well-ordered by inclusion and its order type is α itself. Canonical representatives of those equivalence classes can be choose according to the Von Neumann definition of ordinals. They are inductively defined in [Neu23] as follows: "each ordinal is the set of all ordinals preceding it". From there, $\emptyset$ is the least ordinal (there is a least ordinal as On is well-ordered). $\{\emptyset\}$ is the next one, then comes $\{\{\emptyset\}, \emptyset\}$ and so on. They are consequently ordered by inclusion and this yields a natural way to define natural number in set-theory: the natural number i is represented by the i^{th} (starting at 0) set defined in this way and i is a set of cardinal i and $i + 1 = i \cup \{i\}$. And so "$i < j$" is expressed by $i \in j$. Following this logic, we see ω as the unions of every natural numbers, hence bigger than all of them and himself a well-order. And more generally we often identify an ordinal (in the sense of Definition 3.2.18) with its representative constructed in the definition of Von Neumann.

Definition 3.2.21 (Successor ordinal)**.** The *successor* of an ordinal α is the smallest ordinal greater than α, denoted $\alpha + 1$. That is, with the Von Neumann ordinals, $\alpha + 1 = \{\alpha\} \cup \alpha$.

Definition 3.2.22 (Limit ordinal)**.** An ordinal is a *limit ordinal* if it is neither zero nor the successor of another ordinal.

Now, if we want to see ordinals as a generalization of natural numbers, we would like to be able to do basic arithmetic on those. How to define the sum of two ordinals? To do this, on way is to stick with the definition: given two well-order set $(E, \prec)$ and $(F, \lhd)$, their sum is a well-order that first counts through E and after it exhausted it, that counts through the other. That is, writing $G = E \sqcup F$ their disjoint union and defining $<_G$ as:

$$\begin{aligned} a <_G b \longleftrightarrow\ & (a \in E \wedge b \in E \wedge a \prec E) \\ & \vee (a \in F \wedge b \in F \wedge a \lhd E) \\ & \vee (a \in E \wedge b \in F) \end{aligned}$$

then $(G, <_G)$ can be seen as the sum of those well-orders and the sum of the order types of E and F is defined as the order type of G. This definition has the interest of being rather "manual" and so of showing what happens with the underlying sets. It is however rather impractical if we hope to develop a symbol-orientated arithmetic.

On the other hand, with the definition of Von Neuman, the successor operator is naturally defined: seeing both α as the ordinal and the set of its predecessors, $\alpha + 1$ is the order type of the well-ordered set $\alpha \cup \{\alpha\}$. It leads to an inductive definition of the sum: $\alpha + (\beta + 1) = (\alpha + \beta) + 1$; that is, pushing further the set and ordinal identification, $\alpha + (\beta + 1) = (\alpha + \beta) \cup \{\alpha + \beta\}$. More generally, this yield the following definition, that match the previous suggested definition.

Definition 3.2.23 (Ordinal arithmetic)**.** For two ordinals α and β their sum $\alpha + \beta$ is

inductively defined as:

$$\alpha + (\beta + 1) = (\alpha + \beta) + 1$$
$$\alpha + \beta = \bigcup_{\delta<\beta}(\alpha + \delta) \text{ when } \beta \text{ is a limit ordinal.}$$

and their product, $\alpha \cdot \beta$ is defined as:

$$\alpha \cdot (\beta + 1) = \alpha \cdot \beta + \alpha$$
$$\alpha \cdot \beta = \bigcup_{\delta<\beta}(\alpha \cdot \delta) \text{ when } \beta \text{ is a limit ordinal.}$$

The operator $+$ and $\cdot$ are associative but not commutative.

Definition 3.2.24 (Additively closed ordinals)**.** An ordinal α is additively closed if for all $\beta, \beta' < \alpha$ we have $\beta + \beta' < \alpha$.

Definition 3.2.25 (Multiplicatively closed ordinals)**.** An ordinal α is multiplicatively closed if for all $\beta, \beta' < \alpha$ we have $\beta \cdot \beta' < \alpha$.

Definition 3.2.26 (Countable ordinals and ω_1)**.** A *countable ordinal* is the order type of a countable well-ordered set. We write ω_1 the first uncountable ordinal.

Let $\alpha = [(E, \prec)]$ be a countable ordinal. Then, $E \simeq \mathbb{N}$ with some bijection f which induces an order $\prec'$ on $\mathbb{N}$ with $n \prec' m \longleftrightarrow f^{-1}(n) \prec f^{-1}(m)$ and such that $\alpha = [(\mathbb{N}, \prec')]$. Hence, a countable ordinal can be described by the sole data of a relation (describing a well order) $\prec' \subset \mathbb{N} \times \mathbb{N}$. As such a relation can be encoded in $\mathbb{N}$, this yields a natural way for the encoding of ordinals with reals.

Definition 3.2.27 (Encoding of an ordinal)**.** We say that a real x is a *code for a countable ordinal* α if the relation $\prec \subset \mathbb{N} \times \mathbb{N}$ restricted to the subset of the natural numbers that are related to at least one other natural number and defined as

$$i \prec j \longleftrightarrow \langle i, j \rangle \in x$$

is a well-order of order type α.

Definition 3.2.28 (Recursive ordinals and ω_1^{CK})**.** A *recursive ordinal* is a countable ordinal which admits a code x which is computable (in the sense of Definition 3.2.2.) The first non recursive ordinal is written ω_1^{CK}.

Proposition 3.2.29. *The recursive ordinals form an initial segment of On. That is, if $\alpha < \beta$ and β is recursive, then so is α.*

Proof. Let $\alpha < \beta$ and x a computable code for β. Then x describes a well order $\prec$ on the natural numbers. For i in the domain of $\prec$, we write $\mathbb{N}_i = \{j \in \mathbb{N} \mid j \prec i\}$. Then $\mathbb{N}_i$ is

naturally ordered by (the restriction) of $\prec$ and we can write $\beta_i = [(\mathbb{N}_i, \prec|_{\mathbb{N}_i})]$. As there is a proper injective homomorphism from $(\mathbb{N}_i, \prec|_{\mathbb{N}_i})$ to $(\mathrm{dom}(\prec), \prec)$, we have $\beta_i < \beta$. Further, the map $i \mapsto \beta_i$ induces an isomorphism between $(\mathrm{dom}(\prec), \prec)$ and $(\beta, \in)$. Hence there is some i such that $\beta_i = \alpha$ and this yields the following straightforward way to compute a real coding α: writing m_β for the machine computing x, given j and k, it first simulates m_β to decide whether both $j \prec i$ and $k \prec i$. If it is not the case, it outputs 0; else it uses m_β again to decide if $j \prec k$ and outputs the value returned by m_β. □

We see how the concept of recursive ordinals will help us iterate the Turing jumps through the transfinite. We can now define the real $0^{(\alpha)}$ for a computable α.

Definition 3.2.30 (Transfinite Turing jump)**.** The transfinite Turing jumps of a real x are inductively defined through the recursive ordinals as follows:

$$x^{(0)} = x$$
$$x^{(\alpha+1)} = \left(x^{(\alpha)}\right)'$$

And when α is a computable limit ordinal, writing y for a code for α, $\prec_y$ for the well-order described by y and $(\alpha_j)_{j \in \mathrm{dom}(\prec_y)}$ the enumeration of α induced by this well-order:

$$x^{(\alpha)} = \left\{ \langle i, j \rangle \mid i \in x^{(\alpha_j)} \right\}$$

As is, this definition is still very precarious since the definition of $x^{(\alpha)}$ heavily depends on the choice of the code for α. We now aim to show that the hierarchy of degree induced by this construction does not depend on those codes and is indeed a hierarchy (that is non-collapsing and monotonous). Just before this, let us point out that for a limit α and as was explained earlier for the case $\alpha = \omega$, $x^{(\alpha)}$ is not the lower upper bound of $(x^{(\beta)})_{\beta<\alpha}$, which sequence by the exact pair Theorem does not admit any lower upper bound.

Proposition 3.2.31. *Let two recursive ordinals $\beta < \alpha$. If $x^{(\beta)}$ and $x^{(\alpha)}$ are defined with the same sequence of code until stage β, then $x^{(\beta)} <_T x^{(\alpha)}$.*

Proof. We do this by transfinite induction on α. We have seen that $x^{(\beta)} <_T x^{(\beta+1)}$. For a limit ordinal α, we show that we can compute $x^{(\beta)}$ using $x^{(\alpha)}$. Indeed, let y be the code for a well order $\prec_y$ of order-type α used to define $x^{(\alpha)}$. As seen in the proof of Proposition 3.2.29, there is some j such that $\beta = \alpha_j$, with α_j defined as in Definition 3.2.30. Moreover, under our hypothesis regarding the ordinal code, the β^{th} real appearing in the construction of $x^{(\alpha)}$ is $x^{(\beta)}$ itself. Hence given some i to decide whether $i \in x^{(\beta)} = x^{(\alpha_j)}$, it is enough to decide whether $\langle i, j \rangle \in x^{(\alpha)}$, which is straightforward with $x^{(\alpha)}$ as oracle. □

Proposition 3.2.32. *The degrees induced by this definition do not depend on the choice for the codes of the ordinals.*

Sketch of the proof. The proof is based on Kleene fixed-point theorem. Let Y and Z, two sets of codes for the ordinals $\beta \leqslant \alpha$. We write $x_Y^{(\alpha)}$ and $x_Z^{(\alpha)}$ for the α_{th} jump built respectively using codes from Y and Z. We show by induction that $x_Y^{(\alpha)} \equiv_T x_Z^{(\alpha)}$: when α is not limit, the construction does not rely on codes and the equivalence is obtained by induction hypothesis. Now suppose that α is limit and that for all $\beta < \alpha$, $x_Y^{(\beta)} \equiv_T x_Z^{(\beta)}$. We show that $x_Y^{(\alpha)} \leqslant_T x_Z^{(\alpha)}$ and the other inequality is symmetric. Given some $\langle i, j\rangle$, we want to decide with $x_Z^{(\alpha)}$ as oracle whether $\langle i, j\rangle \in x_Y^{(\alpha)}$. That is, whether $i \in x_Y^{(\alpha_j^Y)}$. We write α_j^Y as the enumeration of the ordinals below α depends on Y. Then, there is some k such that $\alpha_j^Y = \alpha_k^Z$. And by induction hypothesis, $x_Y^{(\alpha_j^Y)} \equiv_T x_Z^{(\alpha_k^Z)}$. As by the previous proposition $x_Z^{(\alpha_k^Z)} <_T x_Z^{(\alpha)}$, given k and with $x_Z^{(\alpha)}$ as oracle, we can compute $x_Z^{(\alpha_k^Z)}$. From this, given k and the index of a function reducing $x_Z^{(\alpha_k^Z)}$ to $x_Y^{(\alpha_j^Y)}$ we can decide whether $i \in x_Y^{(\alpha_j^Y)}$, which is our objective.

We see that the difficulty is two-fold:

- Given j, we want to find k such that $\alpha_j^Y = \alpha_k^Z$
- Given j and k, we want to find the index of a function operating the reduction between $x_Y^{(\alpha_j^Y)}$ and $x_Z^{(\alpha_k^Z)}$

We show how to deal with the first problem. Consider the following inductive function (that for the moment may not be computable) $f(j, k)$ defined such that:

- If $\alpha_j^Y = 0$ and $\alpha_k^Z = 0$, then $f(j, k) = 1$.
- Else, if there is j' and k' such that $\alpha_{j'}^Y + 1 = \alpha_j^Y$ and $\alpha_{k'}^Z + 1 = \alpha_k^Z$, $f(j, k) = f(j', k')$.
- Else, if there is $(\alpha_{j_i}^Y)_i$ and $(\alpha_{k_i}^Z)_i$ two computable sequences unbounded resp. in α_j^Y and α_k^Z and such that for all i, $f(\alpha_{j_i}^Y, \alpha_{k_i}^Z) = 1$, then $f(j, k) = 1$.
- Else, $f(j, k) = 0$.

Then, by induction, for j and k, $f(j, k) = 1 \iff \alpha_j^Y = \alpha_k^Z$. We want to show that such a function f is actually computable with $x_Z^{(\alpha)}$ as oracle. First we define $h(e, j, k)$ with the same definition as f with the only difference that we replace all calls of the function f by calls of φ_e. Then we explain why h is computable with $x_Z^{(\alpha)}$ as oracle: as α is a limit ordinal, for any n, $0^{(n)} <_T x_Z^{(\alpha)}$ and we can decide the n^{th} halting problem with it. This let us decide whether $\alpha_j^Y = 0$ simply by asking whether the machine that looks for a predecessor of j in the computable code $y \in Y$ of α halts. Similarly, we can decide whether α_j^Y is a successor ordinal. And for the third bullet point, with $0''$ as oracle, we can enumerate the machines and test for each of them whether they generate the wanted sequence.

Now, and that is the brilliant use by Kleene of his second recursion theorem, we can find e such that: $\varphi_e(j, k) = h(e, j, k)$ and now, when h applies φ_e in its definition, it actually applies itself and provide the required (transfinite! but grounded) induction.

As for the second problem, finding the machine that reduces $x_Y^{(\alpha_j^Y)} \leqslant_T x_Z^{(\alpha_k^Z)}$, observe that this induction is constructive: at a given stage, given a function g_j that gives the index of the machines operating the reductions for previous stages, we can compute the index of the machine that operates the reduction at this stage. Hence, with the same technique, this yields a grounded inductive and computable function (with oracle) that yields the index of the machine operating the reduction. □

Finally, this let us unequivocally define the degrees $\mathbf{x}^{(\alpha)} = [x^{(\alpha)}]$ for any real and for all recursive ordinals α. How far does the hierarchy of the $\mathbf{0}^{(\alpha)}$ now extends? By definition the recursive ordinals are countable, but as stated in the next proposition, those are bounded in ω_1 and the hierarchy is again put to an halt.

Proposition 3.2.33. *There are countable ordinals that are non-recursive i.e.* $\omega_1^{\mathrm{CK}} < \omega_1$.

Proof. There are countably many Turing machines, hence countably many recursive ordinals, therefore their union is countable. □

3.2.2 Gödel's constructible universe, gaps and master-codes

We see where the previously defined hierarchy hits a wall: its in the effective naming of the ordinals which were used to define new reals of higher degrees. Once stage ω_1^{CK} is reached, ordinal can't be effectively named (with the machines that compute them) and this method can't produce higher order reals. Still, as ω_1^{CK} is countable and as recursive ordinals admit only countably many codes, only countably many reals may be produced in the hierarchy starting from 0. And we would like to believe that since there are uncountably many other reals, one of them is able to extend this hierarchy. However, there now needs a way to define those reals other than by relying on codes for ordinals. Obviously, all reals can be defined as subsets of ω. Still, this is not very helpful as this defines all the real at once rather than gradually and hence does not discriminate one or another. To do this, to define reals in a finer and more gradual way, we need to step in set theory.

Set theory is the study of sets. As a first approximation, sets are collection of other sets, or of other objects. We can start by ruling out the existence of other objects and set are informally understood as collection of sets. This induces a relation $\in$ on sets such that $a \in b$ if the set a is part of the set b, seen as a collection of set. Moreover those collections are well-founded, that is for a given set, again seen as collection of sets, there is a minimal element with respect to $\in$. Now, there needs to be some constraint on which collections of sets define a set as otherwise this construction is met with inconsistencies like the well-known Russel paradox. The axioms of ZFC may be seen, still in a first approximation, as instructions for the construction of new sets with previous sets. This is akin to the back-and-forth that takes place between our intuition and formalism in the definition of recursive functions. In our pre-conception of the idea of set, the empty collection has

everything it takes to be a set. Hence, $\emptyset$ is a set in the formal sense. Then, given a and b, two sets in the formal, we would like to say that the collection $\{a, b\}$ is a set: this yields another way to formally construct sets. And so on, until we feel that we have exhausted intuitively valid ways to build sets upon sets. This gives the following list of axiom, or, with this approximation which we will see can be misleading, of set construction schemes.

Definition 3.2.34 (ZFC)**.** In what follows, free variable are implicitly universally quantified. $x = \emptyset$ is a shortening for $\forall z\,(z \notin x)$ and $\emptyset \in x$ is a shortening for $\exists z\,(z \in x \wedge z = \emptyset)$. Similarly, $\forall x \in a\,\phi$ means $\forall x(x \in a \implies \phi)$ and $a \subset b$ mean $\forall x \in a\,(x \in b)$. Eventually, a function f is described by its graph seen as a set.

$$
\begin{aligned}
\text{Extensionality} &: \forall x\,(x \in a \leftrightarrow x \in b) \to a = b \\
\text{Pair} &: \exists a\,\forall z\,(z \in a \iff z = x \vee z = y) \\
\text{Union} &: \exists B\,\forall a\,\forall x[(x \in a \wedge a \in A) \Rightarrow x \in B]. \\
\text{Powerset} &: \exists y\,\forall z(z \subseteq x \Rightarrow z \in y) \\
\text{Schema of Separation} &: \exists y\,\forall x[x \in y \Leftrightarrow x \in z \wedge \varphi(x, w_1, w_2, \ldots, w_n)] \\
\text{Schema of Replacement} &: \forall x \in A\,\exists! y\,\varphi(x, y, w_1, \ldots, w_n) \Rightarrow \exists B\ \forall x \in A\,\exists y \in B\,\varphi(x, y, w_1, \ldots, w_n) \\
\text{Foundation} &: \forall x\,(x \neq \emptyset \Rightarrow \exists y \in x\,(y \cap x = \emptyset)) \\
\text{Infinity} &: \exists x\,(\emptyset \in x \wedge \forall y \in x\,(y \cup \{y\} \in x) \\
\text{Choice} &: \forall y \in x\,(y \neq \emptyset) \implies \exists f\,\forall y \in x(f(y) \in y)
\end{aligned}
$$

As said, we might want to see those axioms as set-construction schemes. That is, a set would be inductively defined as a collection of set that can be defined using those axiom, keeping in mind that $\emptyset$ and ω can respectively be built *ex nihilo* from the schema of Separation and the axiom of Infinity. While this would be rather innocuous and well-defined for the first steps (that is the finite steps hence using natural numbers, supposing that those are clear enough for us) of this construction, it quickly gets challenging. Indeed, we want this inductive definition to go further than stage ω, as otherwise we can see that most of ordinals would not be deemed as sets. But then, we would need ordinals (that may not be shown to be set yet!) to continue this inductive construction. And those ordinals would indeed not be set yet (with respect to the stage of the constructions they correspond to) and so we would define sets using collections that may not be sets–but that will become sets once the induction carried on until this stage. That is, more precisely: if the definition is carried on up to some ordinal α (as a collection), then α is an ordinal (as a set now). Which makes sense, as carrying this construction up to stage α implicitly assumes that α is a set. Then it seems like the construction loses its purpose as it does not tell us anymore what is a set; we are rather on our own assuming what is a set.

But, looking back, we observe the mathematical practice is itself grounded on this assumption, that we know what a set is. This come from the inherently human aspect of

mathematics that we tried to point out and describe in the introduction. An object is a set when it has been deemed as such in the mathematical practice, whether of the mathematical community or of the individual. So, rather than trying to conduct a bottom-up construction which hopelessly tries to circumvent the self-grounded aspect of mathematics (which, at times, feels like a lack of ground), we are rather drawn to a top-down approach. That is, we accept and build on this implicit assumption which grounds mathematics and we write V for the collection of sets–now working under the hypothesis that the previous axioms are true in this collection and so, in particular, that V is closed under those axioms seen as set-construction schemes. Now that we fixed V, at the same time nothing changed and it feels like we have a steadier ground. It is like a leap in what appears groundless–and which leap magically reaches a ground[1]. And where are we after this leap? Exactly where we already were, on the self-grounding of mathematics, only now more attentive to this surrounding and at the same time more modest in our approach as this newfound ground still eludes us. This leap is remarkably similar to the leap described by Heidegger as required to re-enter the reciprocal relation with Being:

> How can such an entry come about? By our moving away from the attitude of representational thinking. This move is a leap in the sense of a spring. The spring leaps away, away from the habitual idea of man as the rational animal who in modern times has become a subject for his objects. [...] What a curious leap, presumably yielding us the insight that we do not reside sufficiently as yet where in reality we already are. Where are we? In what constellation of Being and man?[2]

But, one thing after the other, we will leave the question of Being there.

From now on, now that we fixed the collection V, we call an object x a set if and only if $x \in V$. Similarly, we call natural numbers the finite Von Neumann ordinals of V. A collection of elements of V which is not a set is called a proper class. For example, V itself is a proper class. And we will see how, despite having to renounce rebuilding and formalizing set theory from nothing, we can do so inside V which acts as a ground and a frame. As a ground, we will be able to formally define the concept of model and as a frame it will allow to us to define the hierarchies of the V_α's and the L_α's. From there,

[1] Some authors write $V = \{x \mid x = x\}$ which may, for some readers, comfortingly hide this leap under the blurriness of the distinction between the formalism of mathematical practice and formal mathematics, themselves contained in the mathematical practice. As a result, this may also tone down the grasp that this leap forcingly provides regarding this distinction. There would be much to say between the relation between mathematical formalism and "representational thinking", as used in the next quote of Heidegger.

[2] *Wie aber kommt es zu einer solchen Einkehr? Dadurch, daß wir uns von der Haltung des vorstellenden Denkens absetzen. Dieses Sichabsetzen ist ein Satz im Sinne eines Sprunges. Er springt ab, nämlich weg aus der geläufigen Vorstellung vom Menschen als dem animal rationale, das in der Neuzeit zum Subjekt für seine Objekte geworden ist. [...] Seltsamer Sprung, der uns vermutlich den Einblick erbringt, daß wir uns noch nicht genügend dort aufhalten, wo wir eigent-lich schon sind. Wo sind wir? In welcher Konstellation von Sein und Mensch?*, Heidegger, *Identität und Differenz*, trans. Stambaugh.

we could also look, in V, at different models of ZFC, study its non-standard models and so on. In some way, V acts as a formal mirror of the humanly mathematical practice and so, as a frontier between the two, between the formal mathematics and the mathematical practice–which we have seen to be in perpetual talks.

So we from here onward, we consider ZFC to be our meta-theory and V to be our meta-mathematical world.

Definition 3.2.35. A set x is transitive if:

$$\forall y \in x \, \forall z \in y \, (z \in x)$$

Definition 3.2.36. Ordinals in V correspond to transitives sets well-ordered by $\in$.

Hence, as seen before, $\emptyset$ is the $\in$-least ordinal. Then comes $\{\emptyset\}$, $\{\{\emptyset\}, \emptyset\}$, etc.

Proposition 3.2.37. *For α an ordinal, $\alpha + 1 = \alpha \cup \{\alpha\}$ is the least ordinal greater than α.*

Proof. We need first to show that $\alpha \cup \{\alpha\}$ is a transitive set well-ordered by $\in$. By the axiom of Extensionality, $\{\alpha\} = \{\alpha, \alpha\}$ and the latter is a set by the axiom of Pair applied to α and α. Then, by the axiom of Pair again, $\{\alpha, \{\alpha\}\}$ is a set and by the axiom of Union, so is $\alpha \cup \{\alpha\}$. It also inherits the transitivity and well-orderdness of α. As $\alpha \in \alpha + 1$ and as there is no β such that $\alpha \in \beta \in \alpha + 1$, $\alpha + 1$ is the successor of α. □

The following property links this definition back to the definition of ordinals as equivalence classes of well-ordered sets, as was introduced in Definition 3.2.18.

Proposition 3.2.38. *In V, for a well-ordered set $(E, \prec)$, there is a unique ordinal α such that $(\alpha, \in) \simeq (E, \prec)$.*

Proof. We proceed by induction. The empty well-order is an ordinal. Then, we consider $(E, \prec)$. For $x \in E$, we write $E_x = \{y \in E \mid y \prec x\}$. By induction for all $x \in E$, there is a unique α_x such that $(\alpha_x, \in) \simeq (E_x, \prec)$. This induces a function f (seen as a predicate) that maps any $x \in E$ to α_x. Now, consider the collection of ordinals $f(E)$. By Replacement and Separation, it is a set. Moreover can show that it is a transitive set of ordinals, and as such, that it is an ordinal. And by construction, as wanted, $(f(E), \in) \simeq (E, \prec)$. Further, for any α such that $(\alpha, \in) \simeq (E, \prec)$, we have $(\alpha, \in) \simeq (f(E), \in)$ and by induction $\alpha = f(E)$.

□

Proposition 3.2.39. *The collection of all ordinals, On, is not a set.*

Proof. Suppose than On is a set. Then, as it is transitive and well-ordered by $\in$, it is an ordinal and so $On \in On$ which contradicts Foundation. □

In what follows, when we speak of ordinals, it is always ordinals that are sets. Now, let us come back to the definition of reals as subset of ω. We remarked that this was to powerful to induce some kind structure among the reals. However, it does induce some structure among sets.

Definition 3.2.40. The *Von Neumann hierarchy of sets* is defined inductively as follow:

$$\begin{aligned} V_0 &= \emptyset \\ V_{\alpha+1} &= \mathcal{P}(V_\alpha) \\ V_\alpha &= \bigcup_{\beta<\alpha} V_\beta \text{ for } \alpha \text{ limit} \end{aligned}$$

Proposition 3.2.41. *For $\alpha \in On$, V_α is a set.*

Proof. As the construction uses the axiom of Powerset and is done along ordinals that are set, we can show by induction using Separation and Replacement that each V_α is a set. □

Proposition 3.2.42. *$V = \bigcup_{\alpha\in On} V_\alpha$ where V is as previously fixed.*

Proof. As for any ordinal α, V_α is a set we have $\bigcup_{\alpha\in On} V_\alpha \subset V$. In the other direction, suppose that $\bigcup_{\alpha\in On} V_\alpha \subsetneq V$. Then, by Foundation, there is a minimal element (w.r.t. $\in$) $x \in V - \bigcup_{\alpha\in On} V_\alpha$. Hence, by $\in$-minimality, for all $y \in x$, there is a least α_y such that $y \in V_{\alpha_y}$. By Replacement and Separation, there is some $\alpha = \bigcup_{y\in x} \alpha_y$ such that $x \subset V_\alpha$ and so $x \in V_{\alpha+1}$ which contradict the characterization of x. □

In this construction, all finite ordinals n are in V_ω and $\omega \subset V_\omega$. Consequently, $\omega \in V_{\omega+1}$ and $\mathcal{P}(\omega) \subset V_{\omega+1}$. But further, V_ω is actually the set of hereditarily finite sets (i.e. set whose transitive closure is finite). This means that all reals that are not finite subset of ω appear at once in this hierarchy between stage ω and $\omega + 1$. As said, this hierarchy of set pays its potency by the fact that its very coarse. Still, when it comes to sets, they are slowly built along the ordinals. This hints toward a similar hierarchy of sets in which the powerset is replaced by a finer operation. To do this, we introduce basic definitions of logic and model theory. See [Hod93] for a complete presentation.

Definition 3.2.43 (Signature)**.** A *first-order signature* $\sigma = (\mathcal{F}, \mathcal{R}, a)$ is the data of a set of function $\mathcal{F}$ and a set of relation $\mathcal{R}$ as well as a function $a : \mathcal{F} \cup \mathcal{R} \to \mathbb{N}$ that maps any symbol to its arity.

Definition 3.2.44 (Signature of set theory)**.** The *signature of set theory* is $(\emptyset, \{\in\}, a)$ with $a(\in) = 2$. (Note that the equality symbol is often considered a logical symbol and hence omitted from the signature.)

Definition 3.2.45 (Language of a signature)**.** The *language L of a signature* σ, is the set of well formed sentences built over σ.

Definition 3.2.46 (Formula of set theory)**.** A *formula of set theory* is a well formed sentence built over the signature of set theory.

Definition 3.2.47 (Theory)**.** Given a signature σ, a *theory T over σ* is a set of sentences built over σ. For a sentence φ in the same language, we write $T \vdash \varphi$ when φ can be proved with the sentences of T as axioms.

Definition 3.2.48 (Structure)**.** Given a signature σ, a *σ-structure* $M = (D, I)$ is a set constituted of the data of a domain D as well as an interpretation I that assigns a function to each element of σ. That is, each $f \in \mathcal{F}$ are mapped to some $f^I : D^{a(f)} \to D$ while each $r \in \mathcal{R}$ are mapped to some $r^I : D^{a(r)} \to \mathbb{B}$. This induces a map over L that maps any well formed and closed sentence φ to its interpretation $\varphi^M \in \mathbb{B} = \{\top, \bot\}$, where $\top$ denotes the truth value and $\bot$ the false value.

Definition 3.2.49 (Model)**.** For a signature σ and T a theory over σ, a σ-structure M is a *model* for T, written $M \models T$, if for all $\varphi \in T$, $\varphi^M = \top$. Further, we write $M \models \varphi$ when M is a model for $\{\varphi\}$.

Now, we remember that we fixed ZFC as our meta-theory. To take advantage of those definitions, we need to formalize those inside ZFC.

To do so, we will first need to formalize integers. This can be done by introducing constants into our meta-language. We write those constants $\ulcorner 0 \urcorner$, $\ulcorner 1 \urcorner$, etc. and we introduce axioms[1] for those constants :

$$\begin{aligned} \forall x \, \neg(x \in \ulcorner 0 \urcorner) \\ \forall x \, (x \in \ulcorner 1 \urcorner \iff x = \ulcorner 0 \urcorner) \\ \dots \end{aligned}$$

And using those integers, we can then encode finistic objects like formulas. So for a formula φ, we can write $\ulcorner \varphi \urcorner$ for it associated constant in our enhanced language of set theory.

With this, formalizing basic model theory in ZFC means that we want it to be possible, in ZFC, to recognize and manipulate the objects we defined. That is, for example (and for simplicity restricting ourselves to the language of set theory), there should be a formula Ψ with one free variables such that for an integer n in our meta-mathematical world,

$\text{ZFC} \vdash \Psi(\ulcorner n \urcorner) \longleftrightarrow n$ encodes a well formed formula in the language of set theory

[1] Observe that adding those constants and axioms of definitions does not change the proving power of ZFC. First, any formula using those constants can be rewritten in the original language of ZFC and, second, nothing new can be proved in those extensions. See for example [Kun80, I, §13].

Further, the concept of satisfaction should be formalized as well, in the sense that there should be a a formula Ψ with three free variables such that for a formula φ with n free variables

$$\mathrm{ZFC} \vdash \forall M \, \forall x_1, \ldots, x_n \, \left[\varphi^M(x_1, \ldots, x_n) \iff \Psi(M, \ulcorner \varphi \urcorner, \langle x_1, \ldots, x_n \rangle) \right]$$

And as we would except, such meta-formulas can actually be constructed and more generally all those definitions of model theory can be formalized in ZFC. This is further discussed in [Kun80].

Hence, this mean that we can do model theory inside any model of ZFC. In other words, in some model W of ZFC (which can be built inside V), we can define the concepts of formula, sentence, models etc. And the finistics objects like formulas will be represented by the integers of W. So this gets particularly interesting when W is a non-standard model of ZFC, that is when its least ordinal ω^W is different from that of our meta-mathematical world, i.e. different from ω^V. In this case, there may also be non-standard formulas, that is non-standard integers which, in W, are deemed to be formulas according to the meta-formula Ψ we introduced.

Now, for our purpose, we can formalize the concept of "definability" inside ZFC.

Definition 3.2.50 (Definability)**.** Given a formula φ in our meta-theory with n free variables and a set A, we say that φ defines the following subset of A^n,

$$\{(x_1, \ldots, x_n) \in A^n \mid (A, \in) \models \varphi(x_1, \ldots, x_n)\}$$

and this subset is said to be *definable over* A. We write $\mathrm{Def}(A)$ the set of definable subsets over A.

Proposition 3.2.51. *The previous definition can be formalized inside ZFC.*

Sketch of proof. We previously outlined the formalization of satisfaction. Hence we can use Ψ to formalize in ZFC the fact that some $B \subset A^n$ is definable by a formula.

Then, since the syntax of first-order logic is formalized as well in ZFC, we can define by collection $\mathrm{Def}(A)$ as the set of definable sets over A, i.e. of sets B for which there exists a formula (in the object theory) which defines B over A. □

Eventually, this brief outline of formalization of definability allows us to define Gödel's constructible universe in our meta-theory.

Definition 3.2.52 (Gödel's constructible universe)**.** We define inductively *Gödel's con-*

structible universe L, as follow

$$\begin{aligned} L_0 &= \emptyset \\ L_{\alpha+1} &= \mathrm{Def}(L_\alpha) \\ L_\alpha &= \bigcup_{\beta<\alpha} L_\beta \text{ for limit ordinals.} \\ L &= \bigcup_{\alpha\in On} L_\alpha \end{aligned}$$

As previously ω appears in this hierarchy at stage $L_{\omega+1}$. It is indeed definable by a first-order formula over L_ω that is true for the y's that are hereditarily transitive sets (i.e. transitive and whose element are hereditarily transitive sets). Indeed, considering

$$\mathrm{Tran}(y) = \forall a \in y \, \forall b \in a \, b \in y$$

we have

$$\omega = \{y \in L_\omega \mid L_\omega \models \mathrm{Tran}(y) \wedge \forall a \in y \ \mathrm{Tran}(a)\}$$

We now prove some simple results on L.

Proposition 3.2.53. *For any ordinal α, $L_\alpha \subset L_{\alpha+1}$ and $L_\alpha \in L_{\alpha+1}$.*

Proof. For the first one, consider $\psi_x(y) \coloneqq y = x$ for all $x \in L_\alpha$ and for the second one, consider $\psi(y) \coloneqq y = y$. □

Proposition 3.2.54. *For any ordinal α, L_α is a set. However, L is a proper class.*

Proof. This comes from the fact that the operator Def can be defined inside ZFC as outlined above. L is a proper class as it contains On. □

Proposition 3.2.55. *For any α, $\alpha \in L_{\alpha+1}$. That is, the Von Neumann representative of the ordinal α is in $L_{\alpha+1}$.*

Proof. We prove this by induction: $0 \in L_1$ and for any $\alpha > 0$, by induction hypothesis $\alpha \subset L_\alpha$ and as previously α is definable over L_α with $\psi(y) := \mathrm{Tran}(y) \wedge \forall a \in y \ \mathrm{Tran}(a)$. □

Proposition 3.2.56. *For α countable, L_α is countable.*

Proof. By transfinite induction, at any stage where α is countable, there are only countably many formulas with parameters in L_α, hence only countably many sets defined over L_α. □

Hence with this hierarchy, it gets more interesting than with V as reals appear "slowly", at most countably many at a time through the countable levels of the transfinite construction of L. This yield a way to compare the complexity of reals: a real can be considered "more complex" than another when it appears later in the constructible hierarchy. Through a simple improvement of this idea, we show how this yields a well-order on L.

Proposition 3.2.57. *L can be well-ordered.*

Proof. We define $<_L$ inductively as follows. First we fix (φ_i) an enumeration of the first-order formulas. Then, for $x \in L_{\beta+1} - L_\beta$ and $y \in L_{\alpha+1} - L_\alpha$ with $\beta < \alpha$, we set $x <_L y$. For $x \neq y$ appearing at the same stage $\alpha + 1$: they are defined over L_α respectively using some least formulas (w.r.t. the previously fixed enumeration) φ_i and φ_j. If $i < j$, we set $x <_L y$, and if $j < i$, we set $y <_L x$. Eventually, if $i = j$, that is, if x and y are both defined using the same least formula, we write $a_1, \ldots, a_n$ the $<_L$-lexicographical least sequence of element of L_α used to defined x with φ_i and $b_1, \ldots, b_n$ the sequence used to define y. Observe that by induction the element of L_α are well-ordered by $<_L$. As $x \neq y$, both sequences are distinct. Hence either the first is $<_L$-lexicographical smaller than the second and $x <_L y$, either it is greater and $y <_L x$. □

Remark 3.2.58. *One may wonder: does $V = L$? both V and L are proper classes and (as classes) are models of* ZFC*. Those being so big, it seems natural that* ZFC *can't grasp them or prove statement about them. Hence, this equality, called the axiom of constructibility, is independent of* ZFC *and can only be answered either in a more powerful theory, either from a philosophical point of view. While it may be quite convincing (why would we postulate the existence of set that can't be constructed?), it should not be forgotten that the power of definition of the operator* Def *does not encompass that of the mathematical consensus, which itself underlies all of mathematical practice.*

Indeed it is actually possible to come up with philosophical intuitions regarding set theory and V, which, once formalized, are shown to be incompatible with the axiom of constructibility. Those intuitions are related to large cardinals. Among those is the existence of $0^\#$, which is a real describing the set of true sentences in L with ordered indiscernibles as parameters (see [Jec03, §18.] for the formal definition of $0^\#$ and its link with axioms of large cardinals).

And the idea of $0^\#$ with its related assumptions, while outside of the scope of the operator Def*, are easy enough to convey in the day-to-day mathematical language; which mean that it has already been given flesh to in the mathematical practice. In the end, this may tip the philosophical scale against the axiom of constructibility. This is one of the moments where the philosophical thinking is caught up and further guided by the mathematical formalism.*

As V may be (tremendously) greater than L, this imposes some caution when stating formal statements involving objects which, to us, may look like unambiguous constants.

Definition 3.2.59. We write ω_1^L for the least uncountable ordinal in L. That is, ω_1^L is the least ordinal in L such that, with $\varphi(I, X, Y)$ the predicate true when I is an injection from X into Y:

$$L \models \forall I \, \neg\varphi(I, \omega_1^L, \omega)$$

If $V \neq L$, then there are set in V that are not in L. In particular, there may be some set $I \in V - L$ that describes an injection from ω_1^L in ω. In other words, V would see that ω_1^L is countable using injection I while L wouldn't, as it can't grasp the fact, in the guise of I, that it is countable. That is, as I (as well as any other suitable injection) is not an element of L, from its point of view, ω_1^L is uncountable. But ω_1^L would then be countable in V and so strictly smaller than ω_1.

3.2.2.1 Gaps on the constructible universe

We have seen how reals appear somewhat slowly in the constructible universe. Another peculiar fact is that at some level, and even for long sequences of level (any countable length actually), no new reals appear. This is called a gap.

Definition 3.2.60 (Gap in the constructible universe, [LP71])**.** Ordinal α corresponds to a *gap in the constructible universe* if

$$(L_{\alpha+1} - L_\alpha) \cap \mathcal{P}(\omega) = \emptyset$$

That is if no real appears between stages $L_{\alpha+1}$ and L_α.

To explain the existence of those gaps, we need a bit more set-theoretic definitions and results.

Definition 3.2.61 (Substructure)**.** For a signature σ and two structures $M = (D, I)$ and $N = (E, J)$, we say that N is a *substructure* of M, written $N \subseteq M$ when $E \subseteq D$ and for all $f \in \mathcal{F}$ and $r \in \mathcal{R}$, f^J and r^J are respectively the restriction of f^I and r^I to E.

Now, what does this definition yields in the case of models of set theory? In such a structure $M = (D, I)$ of set theory, for $a \in D$, we can look at elements $b \in D$ such that $b \in^I a$. This induces a set $a^I = \left\{b \in D | b \in^I a\right\}$. In a satisfying definition of set theory, we except a to be equal to a^I, that is a to be equal to the set made of the elements of a in M. However, consider now $N = (E, J) \subseteq M$. This similarly induces a set $a^J = \left\{b \in E \mid b \in^J a\right\}$. But now, it is wholly possible that some $b \in^I a$ be simply absent of E. In which case, a^I would be different from a^J, and one of them would be different from a. To palliate this issue, we need to introduce a stronger definition.

Definition 3.2.62 (End extension)**.** For two structure $M = (D, I)$ and $N = (E, J)$ in the language of set theory, we say that M is an *end extension* of N (or N an *initial substructure* of M), written $N \subseteq_{end} M$ when $N \subseteq M$ and for all a, with previous notation, $a^I = a^J$

Definition 3.2.63 (Elementary extension)**.** For two σ-structures $M = (D, I)$ and $N = (E, J)$ such that $N \subseteq M$, we say that M is an *elementary extension* of N, written

$N \prec M$ when, for all $\varphi(x_1, \ldots, x_n)$ in the language of σ with free variables $x_1, \ldots, x_n$ and for $a_1, \ldots, a_n \in E$:

$$N \models \varphi(a_1, \ldots, a_n) \longleftrightarrow M \models \varphi(a_1, \ldots, a_n)$$

For two structures M and N of set theory, when M is both an end extension and an elementary extension of N, we speak of elementary end extension (e.e.e.). We may want to restrict the definition of elementary extension to some class of formulas to obtain a weaker definition. To do this, we introduce the Lévy hierarchy.

Definition 3.2.64. In the signature of set-theory a formula $\varphi(x_1, \ldots, x_n)$ is Δ_0 (and Σ_0 and Π_0) if all its quantifiers are bounded (that is of the form $\exists x \in y$ or $\forall x \in y$). A formula $\varphi(x_1, \ldots, x_n)$ is Σ_{n+1} if it is of the form

$$\exists y_1, \ldots, \exists y_m \, \psi(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

where $\psi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is a Π_n formula. Π_{n+1} formulas are defined in the same way from Σ_n formulas.

Definition 3.2.65 (Absolute and persistent formulas)**.** Let φ be a formula in the language of set theory. We say that φ is *persistent* when for all structures of set theory such that $N \subseteq_{end} M$ and with $a_1, \ldots, a_n \in A$, we have

$$N \models \varphi(a_1, \ldots, a_n) \implies M \models \varphi(a_1, \ldots, a_n)$$

And we say that φ is *absolute* when, with the same notations,

$$N \models \varphi(a_1, \ldots, a_n) \iff M \models \varphi(a_1, \ldots, a_n)$$

Proposition 3.2.66. *Σ_1 formulas are persistent and Δ_0 formulas are absolute.*

Sketch of proof. This is done by induction on the formula with the end extension hypothesis providing the base case for the atomic formulas $x \in a$. □

Definition 3.2.67 (Σ_n-elementary extension)**.** For two σ-structures $M = (D, I)$ and $N = (E, J)$ such that $N \subseteq M$, we say that M is an *Σ_n-elementary extension* of N, written $N \prec_{\Sigma_n} M$ when, for all Σ_n formulas $\varphi(x_1, \ldots, x_n)$ in the language of σ with free variables $x_1, \ldots, x_n$ and for $a_1, \ldots, a_n \in E$:

$$N \models \varphi(a_1, \ldots, a_n) \longleftrightarrow M \models \varphi(a_1, \ldots, a_n)$$

Lemma 3.2.68 (Condensation Lemma)**.** *Let α be a limit ordinal and M a transitive structure such that $M \prec_{\Sigma_1} L_\alpha$ for some α. Then $M = L_\beta$ for some $\beta \leqslant \alpha$.*

Proof. First M is well-founded as L_α is (by Σ_1-elementarity a decreasing sequence in M would be decreasing as well in L_α). Hence, we can write β for the well-founded transitive part of M, that is $M \cap On$. As α is a limit ordinal, for all ordinal $\gamma \in M$, $L_\alpha \models \exists \gamma' \, \gamma' > \gamma$. By Σ_1-elementarity, all those sentences are true in M. Hence, β is a limit ordinal as well. Furthermore, again as α is limit:

$$\begin{cases} \text{for all } a \in M \, L_\alpha \models \exists \gamma \, a \in L_\gamma \\ \text{for all } \gamma \in M \, L_\alpha \models \exists a \, L_\gamma = a \end{cases}$$

And those sentences are also true in M. The first sentence implies that $M \subseteq \bigcup_{\delta<\beta} L_\delta$. And the other sentence implies that $\bigcup_{\delta<\beta} L_\delta \subseteq M$. As β is limit, $\bigcup_{\delta<\beta} = L_\beta$, which shows as wanted that $M = L_\beta$. □

Remark 3.2.69. *If M is not transitive, it can still be shown that $(M, \in)$ is isomorphic to some $(L_\beta, \in)$ using the Mostowski collapse lemma. See [Dev84, p. 80] for a proof.*

Proposition 3.2.70 (Tarski-Vaught criterion)**.** *Let M and N be two structures of some signature σ. Then $N \prec M$ is equivalent to the following fact: for all formulas $\varphi(x, y_1, \ldots, y_n)$ and for all $a_1, \ldots, a_n \in N$, $M \models \exists x \, \varphi(x, a_1, \ldots, a_n)$ if and only if there exists $a_0 \in N$ such that $M \models \varphi(a_0, a_1, \ldots, a_n)$.*

Proof. Suppose that the stated fact holds, we show by induction on φ that $N \prec M$. If φ is Σ_0, then it is absolute by Proposition 3.2.66 and it follows from it that $\varphi(a_1, \ldots, a_n)$ being satisfied in N is equivalent to it being satisfied in M. If φ is Σ_n with $n > 0$, then $\varphi(a_1, \ldots, a_n) = \exists x \psi(x, a_1, \ldots, a_n)$ for some Π_{n-1} formula ψ. Suppose that $M \models \varphi(a_1, \ldots, a_n)$. Then by our hypothesis, this is equivalent to having $a_0 \in M$ such that $M \models \psi(a_0, a_1, \ldots, a_n)$. Further, by induction hypothesis, this is also equivalent to having a_0 such that $N \models \psi(a_0, a_1, \ldots, a_n)$. Which is in turn the same as saying that $N \models \varphi(a_1, \ldots, a_n)$. Now, if φ is Π_n, $\neg\varphi$ is Σ_n and saying that

$$M \models \neg\varphi(a_1, \ldots, a_n) \longleftrightarrow N \models \neg\varphi(a_1, \ldots, a_n)$$

is equivalent to say that

$$M \models \varphi(a_1, \ldots, a_n) \longleftrightarrow N \models \varphi(a_1, \ldots, a_n)$$

Conversely, if $N \prec M$, we show in a similar fashion, by induction on φ, that the fact in the statement of the proposition holds. □

This criterion gives a good idea of what may be missing in some $N \subseteq M$ to be an initial substructure of M: for each φ and $a_1, \ldots, a_n \in N$ such that $M \models \exists x \varphi(x, a_1, \ldots, a_n)$, for $N \models \exists x \varphi(x, a_1, \ldots, a_n)$ to be true, there may be missing in N some $a_0 \in M$ as described in the previous theorem. So we could simply add those a_0's to N, for every formula and

tuple of parameters. But then, with those new elements in N there will be new tuples $a_1, \ldots, a_n$ in N and those may again require the existence in N of others elements, and so on. This gives the basis for the proof of the (downward) Löwenheim-Skolem theorem.

Theorem 3.2.71 (Downward Löwenheim-Skolem theorem)**.** *Given σ a signature and M an infinite σ-structure, for all infinite cardinal number κ such that $|\sigma| \leqslant \kappa$ and $\kappa < |M|$, there is a σ-structure N such that $|N| = \kappa$ and such that N is an elementary substructure M.*

Proof. Let M be an infinite σ-structure. We define N inductively. We start with an arbitrary $N_0 \subseteq M$ and we will refine this choice latter. At each stage i, we consider all formulas $\varphi(x, y_1, \ldots, y_n)$ and sets $a_1, \ldots, a_n \in N_i$ such that:

$$M \models \exists x \varphi(x, a_1, \ldots, a_n)$$

This induces a map $A : \varphi, a_1, \ldots, a_n \mapsto a_0$ such that $M \models \varphi(a_0, a_1, \ldots, a_n)$. Observe that the existence of this map relies on the axiom of choice. We define $N_{i+1} \supseteq N_i$ as follows:

$$N_{i+1} = \{A(\varphi, a_1, \ldots, a_n) \mid \varphi \text{ has } n+1 \text{ free variables and } a_1, \ldots, a_n \in N_i\}$$

Eventually, we consider $N = \bigcup N_i$ and by construction, N respect the Tarski-Vaught criterion w.r.t. M. Hence $N \prec M$. Now, what can we say about $|N|$? Through the induction, at each stage, $|N_i + 1|$ depends both on the cardinal of N_i, in which the a_j are quantified, and on that of σ, according to which are built the well formed formulas. Hence, $|N_i + 1| \leqslant |N_i| \cdot \max(\aleph_0, |\sigma|)$. Under the hypothesis that $|\sigma| \leqslant \kappa < |M|$, taking N_0 of infinite cardinality κ yields by a direct induction $|N_i| = \kappa$ for all i and eventually $|N| = \kappa$, as wanted. □

We can now prove the following result from Putnam (see [LP71]).

Proposition 3.2.72 ([LP71])**.** *There are unboundedly big ordinals α in ω_1^L such that α is a gap ordinal.*

Proof. Consider stage ω_1^L of the constructible hierarchy. We can show that $L_{\omega_1^L} \models \mathrm{ZF}^-$, that is $L_{\omega_1^L}$ is a model for ZF minus the axiom of Powerset. By the Löwenheim-Skolem Theorem 3.2.71 there is a countable structure of set theory N such that $N \prec L_{\omega_1^L}$. Moreover, as seen in the proof, for any countable $N_0 \subset L_{\omega_1^L}$, we can suppose that $N_0 \subset N$. So for any $\alpha_0 \in \omega_1^L$, we can suppose that $\alpha_0 \in N$. By the condensation lemma, $(N, \in) = (L_\alpha, \in)$ for some countable $\alpha > \alpha_0$. Hence, $L_\alpha \models \mathrm{ZF}^-$ and α can be chosen unboundedly big in ω_1^L. Now, suppose that a new real x was defined over L_α by some formula φ with parameters in L_α:

$$x = \{j \in \omega \mid L_\alpha \models \varphi(j, a_1, \ldots, a_n)\}$$

Then, the formula φ yields a function f_φ over L_α that maps $i \in \omega$ to the least $j > i$ such that $\varphi(j, a_1, \ldots, a_n)$ (there is always one as otherwise x is finite and so is already in L_α). By the axiom of Comprehension, as $\omega \in L_\alpha$, so is $f_\varphi(\omega)$. But $f_\varphi(\omega)$ is simply x, which is a contradiction. □

Those gaps are strongly linked with master codes as introduced by Jensen in [Jen72] (in a slightly finer way).

Definition 3.2.73 (Master code, [Jen72])**.** A real x is a *master code* for some ordinal α if:

$$L_{\alpha+1} \cap \mathcal{P}(\omega) = \{y \in \mathcal{P}(\omega) \mid y \leqslant_T x\}$$

Clearly, two master codes for a same ordinal are Turing equivalent and for $\alpha < \beta$, any master code of α is computable from any master code of β. So to each ordinal α admitting a master code, we can associate the Turing degree of its master code. Now the question is: are there enough such α's to speak of a hierarchy? And does it yield a satisfying structure for a hierarchy? The following theorem from Jensen answers the first question.

Theorem 3.2.74 ([Jen72])**.** *If α is not a gap ordinal, it admits a master code.*

Sketch of proof. As α is not a gap ordinal, there is some real $x \in L_{\alpha+1} - L_\alpha$. By definition of $L_{\alpha+1}$, x is defined as the set of elements of L_α satisfying some formula ψ with parameters $p_1, \ldots, p_n$ in L_α. Suppose w.l.o.g. that this formula is Σ_n for $n > 1$. We look at $M \in L_\alpha$ defined as the structure of all elements in L_α definable with Σ_n functions and from some fixed parameters $p_1, \ldots, p_n$. M is called the Σ_n Skolem hull of $p_1, \ldots, p_n$. Then with the Tarski-Vaught criterion (Proposition 3.2.70), it can be shown that $M \prec_{\Sigma_n} L_\alpha$: if there is a Σ_n predicate $\exists a\, \varphi(a)$ true in L_α then some a witness of $\varphi(a)$ in L_α is Σ_n-definable in L_α, and so it is in M. By the condensation lemma, $M = L_\beta$ for some $\beta \leqslant \alpha$. Now observe that x is definable over L_β as it is over L_α and those agree on Σ_n formulas. Hence, $\beta = \alpha$ as x appeared at stage $\alpha + 1$. In the end, L_α has been defined using some parameters and a set of Σ_n functions. We write E_α for the real describing this set of Σ_n functions. Observe that E_α is a code for L_α seen as an isomorphic subset of ω. Now, the definition of E_α heavily depends on the enumeration of the Σ_n functions; we need a well-behaved enumeration for E_α to be useful. Jensen showed that there exists a uniform Σ_n Skolem function for which it can be shown that E_α is definable over L_α and that E_α is a master code for α. □

Now, in L, at the same time, the set of (constructible) reals is uncountable and only countably many reals appear at any stage (in the previous proof we actually showed how to build an injection from $L_{\alpha+1}$ into ω if at least one real appeared). This implies that there are ω_1^L stages of the constructible universe where new reals appear and, with previous

theorem, ω_1^L stages admitting master codes. This justifies the following definition from Hodes in [Hod80].

Definition 3.2.75 (Transfinite Turing jump via master code, [Jen72])**.** Given some ordinal $\alpha < \omega_1^L$, $\mathbf{0}^{(\alpha)}$, the α^{th} *transfinite Turing jump via master code* from $\mathbf{0}$, is the degree of the master codes of the α^{th} ordinal admitting a master code.

And we can further show that this notion coincides with the previously defined transfinite Turing jump for ordinals below ω_1^{CK}, hence extending the previous hierarchy.

As for the second question: does this produces a non-collapsing and monotonous hierarchy, the answer is two-fold. First, as mentioned, as a master code for α is in $L_{\alpha+1}$ but not the converse, this ensures that $\mathbf{0}^{(\alpha)} <_T \mathbf{0}^{(\alpha+1)}$. However, if we generalizes this construction for any $\mathbf{a}$ (which can be done in a straightforward manner, defining $L_{\alpha+1}[a]$ as the definable set on $L_\alpha[a]$ using a as a parameter), Hodes shows in [Hod80, Theorem 8] that there exists some degree $\mathbf{a}$ and ordinals $\alpha < \beta$ such that:

$$\mathbf{0}^\alpha = \mathbf{a}^\beta$$

which is a strikingly unsatisfying result for this generalization of the hierarchy. Moreover, another discrepancy highlighted by Hodes is the fact that, contrary to what happened below ω_1^{CK}, there is now no canonical jump operation on the reals ($x \mapsto x'$) that correspond to the jump operation between degrees as induced by Definition 3.2.75.

3.3 Analytical hierarchy and admissible sets

Now, let us go back to the fundamental idea of computation as encompassed by Turing machines. As mentioned in Definition 3.1.7, a (successful) computation of a Turing machine is by definition finite and as seen in Proposition 3.1.13, this finiteness yields the following logical way of apprehending Turing computations: there is an arithmetical formula T such that for a machine m, input i and output j,

$$m(i) = j \longleftrightarrow \exists C\, T(C, m, i, j)$$

where the variable C ranges over the natural number as they can be seen as encoding finite computations. And here, switching back to arithmetic after a long walk in set theory, we see how in order not to get tangled between one and another we need to tread formally.

3.3.1 Arithmetical and analytical hierarchies

Definition 3.3.1 (Signature of first-order arithmetic)**.** The signature of first-order arithmetic is $(\{+, \times, 0, 1\}, \{<\})$ where $+$, $\times$ and $<$ are of arity 2 while the symbols 0 and 1 are constants.

Definition 3.3.2 (Arithmetical hierarchy)**.** In the signature of first-order arithmetic a formula $\varphi(x_1, \ldots, x_n)$ is Δ_0^0 (and Σ_0^0 and Π_0^0) if all its quantifiers are bounded (that is of the form $\exists n < m$ or $\forall n < m$). A formula $\varphi(x_1, \ldots, x_n)$ is Σ_{n+1}^0 if it is of the form:

$$\exists y_1, \ldots, \exists y_m \, \psi(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

where $\psi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is a Π_n^0 formula. And Π_{n+1}^0 formulas are defined in the same way from Σ_n^0 formulas.

Definition 3.3.3 (Σ_0^n real)**.** A real $x \subset \omega$ is Σ_n^0 if it is defined by a Σ_n^0 formula of arithmetic. That is if there exists a Σ_n^0 formula $\varphi(x)$ such that for all $i \in \omega$:

$$i \in x \longleftrightarrow \mathbb{N} \models \varphi(i)$$

We can already establish deep connections between the arithmetical hierarchy and the previous study of Turing degrees: a real is recursively enumerable if and only if it is Σ_1^0. And the Turing jump on one side can be shown to correspond to an added quantifier alternation on the other side. And with this idea, the following theorem can be proved.

Theorem 3.3.4 (Post's theorem)**.** *A real is Σ_{n+1}^0 if and only if it is recursively enumerable with $0^{(n)}$ as oracle.*

Sketch of the proof. A complete proof can be found in [Rog87, p. 314].
We show this by induction. For $n = 0$, we need to show that x is Σ_1^0 if and only if it is r.e. If x is r.e. by some machine m, as seen, there is a Δ_0^0 formula such that

$$m(i) = j \longleftrightarrow \mathbb{N} \models \exists C \, T(C, m, i, j)$$

which shows that x is Σ_1^0. Conversely, given a Σ_1^0 formula $\varphi(x_1) = \exists n \, \psi(x_1, n)$, a machine can semi-decide whether $i \in x$ by looking for a n such that $\psi(i, n)$.

For $n > 0$: suppose that x is r.e. with $0^{(n)}$ as oracle. That is, there is some machine index m such that:

$$i \in x \iff \varphi_m^{0^{(n)}}(i)\downarrow$$

The real $0^{(n)}$ is r.e. with $0^{(n-1)}$ as oracle so by induction, it is defined by a Σ_n^0 formula ζ_n. As previously, the affirmation "$\varphi_m^{0^{(n)}}(i)\downarrow$" can be transformed into an arithmetic formula that quantifies over the computations seen as natural numbers. Moreover, to ensure that said computation was run with $0^{(n)}$ as oracle, the formula also ensures that for those computations, all cells of offset k of the input and on which a part of the oracle is written (with k less than the computation length as only those may have been reached) read 1 if and only $\mathbb{N} \models \zeta_n(k)$. Hence, this yields as required a Σ_{n+1}^0 formula describing x.

Conversely, if x is Σ^0_{n+1}, it is defined by $\varphi(x_1) = \exists n \psi(x_1, n)$. By induction, for any i, a machine can semi-decide $\mathbb{N} \models \psi(i, n)$ with $0^{(n-1)}$ as oracle; so it can decide it with $0^{(n)}$ as oracle. Hence, it can semi-decide $\mathbb{N} \models \varphi(i)$ with $0^{(n)}$ as oracle, as wanted. □

To go further, we consider second-order arithmetic.

Definition 3.3.5 (Second-order arithmetic)**.** The signature of second-order arithmetic is that of first-order arithmetic to which we add the relation symbol $\in$.

Definition 3.3.6 (Sentences of second-order arithmetic)**.** A sentence of second-order arithmetic is composed of both first order ($\exists x_1$, $\forall y_1$) and second order quantifiers ($\exists X_1$, $\forall Y_1$). It is well-formed under the usual constraints to which we add that the symbol $\in$ be only used between a first-order variable and a second-order variable.

Definition 3.3.7 (Interpretation of second-order arithmetic)**.** The intended interpretation of the first-order objects is as natural numbers while that of the second-order object is as set of natural numbers. We write $\mathbb{N} \models \varphi$ when a second-order arithmetic sentence φ is satisfied in the intended interpretation.

Definition 3.3.8 (Analytical hierarchy)**.** In the signature of second-order arithmetic a formula $\varphi(x_1, \ldots, x_n)$ is Δ^1_0 (and Σ^1_0 and Π^1_0) when it has no second order quantifier. A formula $\varphi(x_1, \ldots, x_n)$ is Σ^1_{n+1} if, once curated from its first-order quantifiers, it is of the form

$$\exists Y_1, \ldots, \exists Y_m \, \psi(x_1, \ldots, x_n, Y_1, \ldots, Y_m)$$

where $\psi(x_1, \ldots, x_n, Y_1, \ldots, Y_m)$ is a Π^1_n formula. Π^1_{n+1} formulas are defined in the same way from Σ^1_n formulas.

Definition 3.3.9. A set of natural numbers $x \subset \omega$ is Σ^1_n if it is definable by a Σ^1_n formula of arithmetic. That is if there exists a Σ^1_n formula $\varphi(x_1)$ such that for all $i \in \omega$:

$$i \in x \longleftrightarrow \mathbb{N} \models \varphi(i)$$

A set of real $A \subset \mathcal{P}(\omega)$ is Σ^1_n if it is definable by a Σ^1_n formula of arithmetic. That is if there exists a Σ^1_n formula $\varphi(X_1)$ such that for all $x \in \mathcal{P}(\omega)$:

$$x \in A \longleftrightarrow \mathbb{N} \models \varphi(x)$$

In the analytical hierarchy, the first level is of particular interest and has deep connections with what we have seen on Turing degrees. We call hyperarithmetical the Δ^1_1 sets and we show how with those sets we can generalize Post's theorem. This is mostly due to Kleene's work.

Definition 3.3.10. We write $T^1(x_1, x_2, x_3, x_4)$ for the Δ^1_0 formula such that:

$$T^1(m, x, k, i) \longleftrightarrow \varphi^x_m(i)\downarrow \text{ in less than } k \text{ steps.}$$

which can be defined with the same idea as for T.

Theorem 3.3.11 (Kleene normal-form theorem)**.** *Let φ be a Σ^1_n formula. Then there is some machine m such that for all reals x_1, ..., x_m:*

$$\mathbb{N} \models \varphi(x_1, \ldots, x_m) \iff \exists X_1 \forall X_2 \ldots Q X_n \, \exists k \, T^1(m, \langle x_1, \ldots, x_m, X_1, \ldots, X_n\rangle, k, i)$$

where $Q \in \{\exists, \forall\}$ depending on the parity of n.

Proof. This is shown by induction. See again [Rog87, §16] for a complete presentation. □

Observe that in the previous normal form, it is somehow redundant to give the complete real $\langle x_1, \ldots, x_m, X_1, \ldots, X_n\rangle$ as parameter to T^1 as, looking for an halting computation with less than k steps, it only needs the k first bits of $\langle x_1, \ldots, x_m, X_1, \ldots, X_n\rangle$. So, for a real x we write $\bar{x}^k$ for the truncation of x up to its k first bits. And with it we can define the formula $\bar{T}^1$ that avoid this redundancy.

Definition 3.3.12. We write $\bar{T}^1(x_1, x_2, x_3, x_4)$ for the Δ^1_0 formula such that:

$$\bar{T}^1(m, t, k, i) \longleftrightarrow \exists x \, \bar{x}^k = t \wedge \varphi^x_m(i)\downarrow \text{ in less than } k \text{ steps.}$$

Theorem 3.3.13 (Kleene alternative normal-form theorem)**.** *For all reals $x_1, \ldots, x_m$:*

$$\mathbb{N} \models \exists X_1 \forall X_2 \ldots Q X_n \, \exists k \, T^1(m, \langle x_1, \ldots, x_m, X_1, \ldots, X_n\rangle, k, i)$$
$$\iff$$
$$\mathbb{N} \models \exists X_1 \forall X_2 \ldots Q X_n \, \exists k \, \bar{T}^1(m, \overline{\langle x_1, \ldots, x_m, X_1, \ldots, X_n\rangle}^k, k, i)$$

Proof. Follows from previous explanations. □

Proposition 3.3.14. *The set of reals describing a well-order,* WO*, is a Π^1_1 set of reals.*

Proof. We want a formula that given x as parameters decides whether x encodes well-order. So let R be the relation described by some real x in the usual $\langle i, j\rangle \in x$-fashion. First, to check whether R is a total order on some subset of ω it is enough for the formula to check if for all natural numbers i, j and k, the conditions for being a total order, seen as a Δ^1_0 predicates of i, j,k and x are satisfied. Hence, as i, j and k are quantified with a first order "$\forall$", this also yields a Δ^1_0 predicate $\mathrm{LO}(x)$ which is true when x describes a total order. Then, a total order R is a well-order if and only if every non-empty subset admits a minimal element. So:

$$x \in \mathrm{WO} \iff LO(x) \wedge \forall y \neq \emptyset \, \exists i \in y \, \forall j \in y \, (\langle j, i\rangle \notin x)$$

which is a Π^1_1 predicate. □

Proposition 3.3.15. *The set of reals describing a well-order,* WO*, is Π^1_1-complete. That is for all Π^1_1 set A, there is a recursive function e such that for all real x:*

$$x \in A \longleftrightarrow \{i \in \omega \mid \varphi^x_e(i) = 1\} \in \mathrm{WO}$$

Proof. Let A be a Π^1_1 set. That is, there is a predicate $\psi(x) = \forall y\, \psi_0(x, y)$ where y is quantified over the reals and ψ_0 is Σ^0_n for some n and such that:

$$x \in A \longleftrightarrow \psi(x)$$

By the alternative normal-form theorem, there is some machine m such that:

$$x \in A \longleftrightarrow \forall y\, \exists k\, \bar{T}^1(m, \overline{\langle x, y\rangle}^k, k, 0)$$

Then, we consider the set B of elements of the form $\overline{\langle x, y\rangle}^k$ for some y and k and such that $\neg\bar{T}^1(m, \overline{\langle x, y\rangle}^k, k, i)$. B can be totally ordered by the Kleene-Brouwer ordering. This order is usually defined on finite sequences but it is easily adaptable to elements of B seen as finite sets of natural numbers. For two sets a and b we write $a\Delta b$ for their symmetrical difference. And for $a, b \in B$, we write $a <_{KB} b$ when either $a \subset b$ or $a\Delta b \neq \emptyset$ and the least element of $a\Delta b$ is in b.

Now, $x \notin A$ if and only if there is some y such that $\forall k \neg\bar{T}^1(m, \overline{\langle x, y\rangle}^k, k, 0)$. That is, if for some y the sequence $(\overline{\langle x, y\rangle}^k)_k$ is in B. Which implies that there is a strictly increasing sequence: $\overline{\langle x, y\rangle}^0 <_{KB} \overline{\langle x, y\rangle}^1 <_{KB} \overline{\langle x, y\rangle}^2 \ldots$ Hence that B is not well-ordered w.r.t. the inversed relation $<^{-1}_{KB}$.

Conversely, we show that if B is not well-ordered by $<^{-1}_{KB}$, then $x \notin A$. Let $a_1 <_{KB} a_2 <_{KB} a_3 \ldots$ be a decreasing sequence (w.r.t. $<^{-1}_{KB}$) in B. Then, for any j, we look at the sets $a^j_i = \{n < j \mid n \in a_i\}$. Looking at the definition of $<_{KB}$ it is clear that for any j, the sequence $(a^j_i)_i$ converges and that for $j < j'$, the stabilization value of $(a^j_i)_i$ is a subset of that of $(a^{j'}_i)_i$. Hence, the limit of those stabilization values as j grows to ω yield some real y such that the sequence $\overline{\langle x, y\rangle}^0 <_{KB} \overline{\langle x, y\rangle}^1 <_{KB} \overline{\langle x, y\rangle}^2 \ldots$ is in B. This implies that $\forall k \neg\bar{T}^1(m, \overline{\langle x, y\rangle}^k, k, 0)$, that is that $x \notin A$.

Eventually, the real describing the relation $<^{-1}_{KB}$ over B is uniformly computable from x. Indeed, given two natural numbers a and b, a machine using x can decide whether both are in B and then whether $a <_{KB} b$. □

From this, we can define the hyperarithmetical hierarchy.

Theorem 3.3.16. *A real x is Δ^1_1 if and only if x is recursive with $0^{(\alpha)}$ as oracle for some recursive ordinal α.*

Sketch of proof. Suppose that x is recursive with $0^{(\alpha)}$ as oracle for α recursive. By definition of x, there is some machine index e such that for all $i \in \omega$:

$$i \in x \longleftrightarrow \varphi_e^{0^{(\alpha)}}(i)\downarrow 1 \longleftrightarrow \exists k\, T_o^1(e, 0^{(\alpha)}, k, i, 1)$$

where T_o^1 is a slight modification of T^1 that checks whether the output is equal to the last parameter (here 1). Then, to obtain an hyperarithmetical formula to express "$i \in x$", what is left is to find a way to define $\overline{0^{(\alpha)}k}$. If $\alpha = \alpha' + 1$, then:

$$e \in 0^{(\alpha)} \longleftrightarrow \varphi_e^{0^{(\alpha')}}\downarrow$$

If α is limit, $e \in 0^{(\alpha)}$ then:

$$e \in 0^{(\alpha)} \longleftrightarrow e = \langle f, g\rangle \wedge f \in 0^{(\alpha_g)}$$

where $\alpha_g < \alpha$ is the ordinal of index g in the encoding of α. We see how $e \in 0^{(\alpha)}$ is naturally defined by induction on α, which induction can be encoded by a real using the code for α, itself recursive. This yields a Σ_1^1 definition of x. And as there can only be one real encoding this induction, this also yields a Π_1^1 definition and so x is Δ_1^1-definable.

Conversely, suppose that x is Δ_1^1. Then, it can defined by some Π_1^1 formula φ. By Proposition 3.3.15, seeing natural numbers as finite reals, there is a machine e such that:

$$i \in x \longleftrightarrow \left\{j \in \omega \mid \varphi_e^i(j)\right\} \in \mathrm{WO}$$

Hence, this induces a function on x, $i \mapsto \alpha_i$ such that for all i, $\alpha_i \in \omega_1^{\mathrm{CK}}$ (as i is finite). We consider $\alpha = \bigcup_i \alpha_i$. By Σ_1^1 bounding (see [Rog87, p. 400]), we can show that $\alpha \in \omega_1^{\mathrm{CK}}$. Hence, $i \in x$ if and only if φ_e^i describes a well-order of order-type less than α. And for a machine, deciding whether φ_e^i describes a well-order is recursive using $0'$ and deciding whether this well-order is of order type less than α is recursive using the machine that describes α and $0^{(\alpha)}$, hence showing that x is $0^{(\alpha)}$-recursive. □

We see how this approach is firmly anchored in the natural numbers: quantifiers are ranging either over ω or $\mathcal{P}(\omega)$ and even when we consider reals, they are only considered one finite prefix at a time, as is embodied by Theorem 3.3.13. This comes with many advantages: anything finite is easily encoded with natural numbers and the set of the natural numbers come with very strong structural properties that are naturally unnoticed in their everyday usage.

But rather than piling up the quantifiers, a way to generalize this idea of computation might be to altogether start with more general sets than that of natural numbers. A questions arises: what would the computation of a Turing machine encoded by an infinite set, for example an infinite ordinal, look like? We will postpone this question until next chapter that precisely develops this idea. For the time, we forget about machines. What

is left is the logic aspect of the computation and this approach lead to the study of admissible sets and with it to the development of α-recursion.

3.3.2 Admissible sets and α-recursion

The idea developped by Sacks and others (see [Sac90]) is the following: by analogy with the usual computation over ω, some set A will play the role of ω. Hence a set B is said A-finite when $B \in A$. And B is A-recursively enumerable when $B \subset A$ and B is definable by a Σ_1 predicate over A. From there, B is A-recursive when both B and $A-B$ are A-r.e., that is if and only if B is Δ_1-definable over A. Still, if we hope to generalize some of the results of the finite recursion theory, we need to impose some structure on A. First come natural expectations for the set A: before anything, it should be well-founded. Then it should be closed by taking pairs; this allows for example to define recursive functions as functions whose graph (so made of pairs) is recursive. Among other things, we can also ask that if B is A-finite, then so should be the union of the subsets of B. And further, in a less straightforward but still fundamental way, we except the following statement to hold: if f is a A-r.e. function and there is some A-finite a such that $a \subset \mathrm{dom}(f)$, then $f(a)$ should be A-finite as well.

Those expectations can be met when A is a transitive model for Kripke-Platek set theory, KP. Kripke and Platek sought (independently) a formalization of what should we require from a set that we want to see as a domain of computation. It happened that the axioms needed to describe such sets where weaker than those of the classical set theory, ZF, and it gave rise to KP which is an interesting and more parsimonious set theory. KP also plays a role outside of computation theory. Set theorists lead fine enquiries about the nature of its models that are plentiful and come in many shapes and sizes. In contrast classical set theory is in this aspect (that is of having interesting models) deemed "in some ways too strong" (see [Bar75] for an extensive study of KP and admissible sets). As an illustration, we have seen how the axiom of Powerset can itself be in some ways to powerful and a consequence of it is that any model of ZF is inhumanly big.

Definition 3.3.17 (Kripke-Platek set theory)**.** KP is defined by the following axioms on the language of set theory, where the free variables are implicitly universally quantified

and with the usual notations.

$$\begin{aligned}
\text{Extensionality} &: \forall x\,(x \in a \leftrightarrow x \in b) \implies a = b \\
\text{Foundation (or induction)} &: \exists x\,\varphi(x) \implies \exists x\,(\varphi(x) \wedge \forall y \in x\,\neg\varphi(y)) \\
\text{Pair} &: \exists a(x \in a \wedge y \in a) \\
\text{Union} &: \exists b\,\forall y \in a\,\forall x \in y\,(x \in b) \\
\Delta_0\text{-separation} &: \exists b\,\forall x\,(x \in b \implies x \in a \wedge \varphi(x)) \text{ for all } \Delta_0 \text{ formulas.} \\
\Delta_0\text{-collection} &: \forall x \in a\,\exists y\,\varphi(x,y) \implies \exists b\,\forall x \in a\,\exists y \in b\,\varphi(x,y) \text{ for all } \Delta_0 \text{ formulas.}
\end{aligned}$$

When a transitive set A is a model of KP, we say that A is an *admissible set.*

Let us first show some basic results on KP.

Definition 3.3.18. For a formula φ and a set a, we write $\varphi^{(a)}$ for the formula φ in which every unbounded quantifier is bounded by a.

Definition 3.3.19 (Σ formula)**.** A Σ *formula* is a formula without negation and in which all universal quantifiers are bounded. A Π *formula* is the negation of a Σ formula. A Δ *formula* is a formula equivalent in KP to both a Σ formula and a Π formula.

Observe that whether Σ formulas are equivalent to Σ_1 formulas is not immediately clear. Take for example anything of the form: $\exists a\,\forall b \in a\,\exists c\,\psi(a,b,c)$. It is Σ but may not be equivalent in KP to any Σ_1 formula. The following proposition bridges this gap.

Lemma 3.3.20. *For a set A, a Σ formula ψ and two sets $a, b \in A$, the two following sentences are true in A:*

$$\begin{cases} \psi^{(a)} \wedge a \subseteq b \implies \psi^{(b)} \\ \psi^{(a)} \implies \psi \end{cases}$$

Proof. This is done by induction and it rests on the fact that the boolean operator induced by the bounded existential quantifier is monotonous in the bound. □

Proposition 3.3.21 (Σ-reflection)**.** *Let φ be a Σ formula. We can show in KP that φ is equivalent to a Σ_1 formula with a single unbounded existential quantifier. That is, more precisely:*

$$\text{KP} \vdash \varphi \iff \exists a\,\varphi^{(a)}$$

Proof. We show this by induction on φ. If φ is Δ_0, then $\varphi^{(a)} = \varphi$ and it is directly equivalent to $\exists a \varphi^{(a)}$.

If $\varphi = \psi_1 \wedge \psi_2$, then by induction:

$$\begin{aligned} \mathrm{KP} \vdash \psi_1 &\iff \exists a_1 \, \psi_1^{(a_1)} \\ \mathrm{KP} \vdash \psi_2 &\iff \exists a_2 \, \psi_2^{(a_2)} \end{aligned}$$

Hence, supposing that $\mathrm{KP} \vdash \psi_1 \wedge \psi_2$, we can say in KP that there are a_1 and a_2 such that respectively $\psi_1^{(a_1)}$ and $\psi_2^{(a_2)}$. Now consider $a = a_1 \cup a_2$ which exists by virtue of the pairing and union axioms. By the previous lemma, in any model of KP we have both $\psi_1^{(a)}$ and $\psi_2^{(a)}$. Hence $\mathrm{KP} \vdash \exists a \, \psi_1^{(a)} \wedge \psi_2^{(a)}$. Conversely, if $\mathrm{KP} \vdash \exists a \, \psi_1^{(a)} \wedge \psi_2^{(a)}$ then with the other half of the lemma, we have that $\mathrm{KP} \vdash \psi_1 \wedge \psi_2$. And the case where $\varphi = \psi_1 \vee \psi_2$ is similar.

If $\varphi = \forall b \in a \, \psi(b)$, then by induction for all $b \in a$:

$$\mathrm{KP} \vdash \psi(b) \iff \exists c \, \psi(b)^{(c)}$$

Supposing that, $\mathrm{KP} \vdash \forall b \in a \, \psi(b)$, this induces a mapping of all $b \in a$ to some c_b such that $\psi(b)^{(c_b)}$ is true in KP. By Δ_0-collection, there is some d that collects those c_b and with the same reasoning as previously, $\forall b \in a \, \psi(b)^{(d)}$. Hence $\mathrm{KP} \vdash \exists d \, \varphi^{(d)}$. And the other direction again holds in virtue of the second part of the lemma.

If $\varphi = \exists a \, \psi(a)$, then by induction:

$$\mathrm{KP} \vdash \psi(a) \iff \exists b \, \psi(a)^{(b)}$$

Now, suppose that $\mathrm{KP} \vdash \exists a \, \psi(a)$. Then, in KP there is a and b such that $\psi(a)^{(b)}$. Take $c = b \cup \{a\}$ and we have $\mathrm{KP} \vdash \exists c \, \exists a \in c \, \psi(a)^{(c)}$. The converse case is again direct. □

Proposition 3.3.22 (Σ-collection)**.** *Σ-collection is a theorem of KP. That is: for a set a and a Σ formula $\varphi(x, y)$ such that $\forall x \in a \, \exists y \, \varphi(x, y)$, there exists b such that*

$$\begin{cases} \forall x \in a \, \exists y \in b \, \varphi(x, y) \\ \forall x \in b \, \exists y \in a \, \varphi(x, y) \end{cases}$$

Proof. Let $\varphi(x, y)$ a Σ formula such that, working in KP: $\forall x \in a \, \exists y \, \varphi(x, y)$. Then, by Σ reflection, there exists c such that $\forall x \in a \, \exists y \in c \, \varphi^{(c)}(x, y)$. By Δ_0 separation, we can consider $b = \{y \in c \mid \exists x \in a \, \varphi^{(c)}(x, y)\}$. Then by Lemma 3.3.20, we have both

$$\forall x \in a \, \exists y \in b \, \varphi(x, y)$$

and

$$\forall x \in b \, \exists y \in a \, \varphi(x, y)$$

□

Proposition 3.3.23 (Δ separation)**.** *Δ separation is a theorem of KP. That is, for a Δ formula $\varphi(x)$ and a set a, there exists $b = \{x \in a \mid \varphi(x)\}$.*

Proof. As $\varphi(x)$ is Δ, we can suppose w.l.o.g. that it is written as a Σ formula and that it is equivalent to some Π formula $\psi(x)$. Then, the sentence $\forall x \in a \, (\varphi(x) \vee \neg\psi(x))$ is true in KP and it is Σ. By Σ-reflection, there is some c such that $\forall x \in a \, (\varphi^{(c)}(x) \vee \neg\psi^{(c)}(x)$. Then, by Δ_0-separation, we can consider $b = \{x \in a \mid \varphi^c(x)\}$. By Lemma 3.3.20, this is a subset of $\{x \in a \mid \varphi(x)\}$. Now, $\bar{b} = a - b = \{x \in a \mid \neg\psi^c(x)\}$ is again definable by Δ_0-separation and by Lemma 3.3.20, it is a subset of $\{x \in a \mid \neg\psi(x)\}$. Hence, b is the desired set. □

Now, we can show how the fact that some domain of computation A is admissible helps us to establish the results we expect to hold in a satisfying domain of computation. In what follows we let A be an admissible set. We give again some of the previous definitions.

Definition 3.3.24. $B \subset A$ is definable by a Σ_1 predicate over A if there is some Σ_1 formula φ such that

$$x \in B \iff A \models \varphi(x)$$

Definition 3.3.25 ([Sac90])**.** B is *A-finite* when $B \in A$. B is *A-r.e.* when $B \subset A$ and B is definable by a Σ_1 predicate over A. B is *A-recursive* when $A \subset B$ and B is definable by a Δ_1 predicate over A.

Proposition 3.3.26 ([Sac90])**.** *Let f be a A-r.e. function such that for some A-finite a we have $a \subset \mathrm{dom}(f)$. Then $f(a)$ is A-finite.*

Proof. We need to show that $f(a) = \{f(b) \mid b \in a\}$ is in A. Let φ_f be the Σ predicate defining f. That is,

$$f(x) = y \iff \varphi_f(x, y)$$

Then, $a \subset \mathrm{dom}(f)$ translates into: $\forall b \in a \, \exists y \, \varphi_f(b, y)$ is true in A. By Σ collection, there is some set $c \in A$ that collects those $f(b)$ and only those. Hence $c = f(a)$ is in A. □

This shows how taking an admissible set as an higher domain of computation satisfies such basic expectations as the image of a "finite" (that is A-finite) set being "finite" (again A-finite). From there we can develop the theory of α-recursion in the domain A and prove

some results like the following, further showing the analogy between finite and A-finite sets or r.e. functions and A-r.e. functions.

Proposition 3.3.27 (Σ_1-recursion, [Bar75])**.** *Let $h(x, y, z)$ and $g(x)$ two A-r.e. functions and consider the function f defined as:*

$$\begin{cases} f(0, y) = g(y) \\ f(\alpha, y) = h\left(\alpha, y, (f(\beta, y))_{\beta\in\alpha}\right) \end{cases}$$

Then f is A-r.e. as well.

Proof. We need to show that the function f is Σ_1 definable over A. That is, we need to produce a Σ_1 predicate φ_f such that:

$$f(\alpha, y) = x \iff \varphi_f(\alpha, y, x)$$

In order to "unroll" the inductive definition of f, we need to consider a wider setting in which all steps of the inductive definition are given. That is we consider:

$$\begin{aligned} \psi(\alpha, y, x, f) \coloneqq\ & f \text{ is a function} \wedge \operatorname{dom}(f) = \alpha + 1 \\ & \wedge f(0) = g(y) \wedge \forall \beta \in]0, \alpha]\ f(\beta) = h(\beta, y, f|_\beta) \\ & \wedge f(\alpha) = x \end{aligned}$$

where $f|_\beta = \{\langle \delta, x\rangle \in f \mid \delta < \beta\}$. Now as g and h are Σ_1-definable as A-r.e. functions, ψ is a Σ_1 predicate. What is left to prove is that:

$$f(\alpha, y) = x \iff \exists f\, \psi(\alpha, y, x, f)$$

To do this, we prove that:

$$\forall \alpha, y\, \exists! f, \exists x\, \psi(\alpha, y, x, f)$$

First, the unicity of x will come from that of f and from the fact that f must be a function to satisfy ψ. Then, for the unicity of f: suppose that for some α and y, there are $f \neq f'$ and $x \neq x'$ such that both $\psi(\alpha, y, x, f)$ and $\psi(\alpha, y, x', f')$. Then $f(0) = g(y) = f'(0)$ and by transfinite induction $f = f'$.

Now, for the existence of f. We proceed by induction. For $\alpha = 0$, $f = \langle 0, g(y)\rangle$ is a set of A by the axiom of Pair. Then, for $\alpha > 0$, by induction hypothesis:

$$\forall \beta < \alpha, y\, \exists! f_\beta\, \exists x\, \psi(\alpha, y, x, f_\beta)$$

By Σ-collection, the set $F = \{f_\beta \mid \beta < \alpha\}$ is in A and $f_\alpha = \{f_\beta[\beta] \mid f_\beta \in F\}$ as well. □

Proposition 3.3.28 ([Sac90]). *Suppose that $A = L_\alpha$ for some ordinal α. Then, the A-finite elements are enumerable along α with an A-recursive function. That is, there is a bijective A-recursive function f from α onto L_α. It is the analogy of the fact that we can enumerate the finite sets along ω.*

Sketch of proof. We consider the well-order of L defined in 3.2.57. L_α is also well-order by the restriction of $<_L$ and we can define by Σ_1 recursion a bijective Σ_1 function f such that for $\beta \leqslant \alpha$

$$f(\beta) = x \iff x \text{ is the } \beta^{th} \text{ element in } L_\alpha \text{ w.r.t. } <_L$$

Now we want to show that all x in L_α is the image of some β. It isn't the case if and only if there is some $x \in L_\alpha$ such that $f(\alpha) = x$. In this case, we consider $X = \{y \in L_\alpha \mid \leqslant_L x\}$. It is in $L_{\alpha+1}$ as it is definable with x as parameter over L_α. And by Σ_1 collection, $\{\beta \mid \exists y \in X\, f(\beta) = y\}$ is also in $L_{\alpha+1}$. But this set is simply $\alpha+1$, which is a contradiction. Hence $f(\alpha) = L_\alpha$ and f is a bijective A-r.e. function from α onto L_α. As such, it is also A-recursive. □

Now a question may be, how and where do we find interesting admissible sets? As seen in the previous result, when the admissible domain of computation is of the form L_α, we can harvest the structure of L_α to obtain finer results. In this direction, the following results and also Proposition 3.3.34 show the interest of again considering the Gödel hierarchy in the study of admissible sets.

Proposition 3.3.29. *Let α be a limit ordinal. Then L_α is admissible if and only if L_α satisfies Δ_0-collection.*

Proof. We just need to show that L_α satisfies all other axioms of KP. As L_α is a set, Extensionality and Foundation hold. Then we show that Pair holds: let x and y in L_α. As α is limit, there is some $\beta < \alpha$ such that $x, y \in L_\beta$. Then, $\{x, y\}$ is definable over L_β by the formula $\varphi(z) \coloneqq z = x \vee z = y$ with x and y as parameters. Hence $\{x, y\} \in L_{\beta+1} \subset L_\alpha$. Union is shown in the same way. Eventually, for Δ_0-separation, let a some set in L_α and φ a Δ_0 formula. As previously, there is some $\beta < \alpha$ such that $a \in L_\beta$. Now, consider the formula $\varphi(z) \coloneqq z \in a \wedge \varphi(x)$. It defines the wanted set over L_β. □

Definition 3.3.30 (Admissible ordinal). An ordinal α is admissible if L_α is admissible.

Definition 3.3.31 (Cardinal). An ordinal α is a *cardinal* if for all $\beta < \alpha$ there is no surjection from β onto α. For α an ordinal, we write α^+ the *next cardinal*, that is the least cardinal strictly greater that α. The *cardinality* of set is the least cardinal in bijection with this set.

Definition 3.3.32 (Cofinality). The *cofinality* of an ordinal α, $\mathrm{cf}(\alpha)$, is the the least order-type ν of a sequence $(\alpha_\iota)_{\iota<\nu}$ unbounded in α. A limit ordinal α is said to be *regular* if $\mathrm{cf}(\alpha) = \alpha$.

Proposition 3.3.33. *For κ an infinite cardinal, κ^+ is regular.*

Proof. Suppose the contrary. Then, there is some sequence $(\alpha_\iota)_{\iota<\nu}$ unbounded in κ^+ with $\nu < \kappa^+$. Then, ν has cardinality κ and for any $\nu' < \nu$, α_ι has cardinality κ as well. Hence, the sum of the α_ι which is equal to κ^+ has cardinality κ which is a contradiction. □

Proposition 3.3.34. *Admissible ordinals are unbounded in ω_1 and in On.*

Proof. We show first that ω_1 is admissible. To do this, it is enough to show that L_{ω_1} satisfies Δ_0-collection. Let $a \in L_{\omega_1}$ such that $\forall x \in a\, \exists y \varphi(x, y)$ for some Δ_0 formula φ. By definition of L_{ω_1+1}, it must contain a $<_L$-least b such that $\forall x \in a\, \exists y \in b \varphi(x, y)$. We want to show that b is in L_{ω_1}. First as a is countable, so is b. Further, observe that $\omega_1 = \omega^+$. Hence, ω_1 is regular. Now, as $b \subset L_{\omega_1}$, for all $x \in b$, there is some $\alpha_x < \omega_1$ such that $x \in L_{\alpha_x}$. This induces a countable sequence of ordinals below ω_1. By regularity of ω_1 this sequence is bounded and both $a \in L_\beta$ and $b \subset L_\beta$ for some $\beta < \omega_1$. Hence, as b is definable with a as parameters, $b \in L_{\beta+1}$.

Then, by the Löwenheim-Skolem Theorem 3.2.71, for any $\alpha < \omega_1$ we can find some $M \supset L_\alpha$ such that $M \prec L_{\omega_1}$. By the condensation lemma $M = L'_\alpha$ for some $\alpha' > \alpha$ and it is easy to see that admissibility is reflected down to M by elementarity; hence that α' is admissible. This shows that admissible ordinals are unbounded in ω_1.

To see that they are also unbounded in On, observe that with the same reasoning as in ω_1, for any cardinal κ, κ^+ is admissible. □

Hence, the admissible sets of the form L_α come in any size. And the fact that they all have a very structured shape, as level of Gödel's universe, make them particularly convenient domains for α-recursion. From there, many techniques and results can be generalized to α-recursion on admissible L_α's. For example the priority argument, as used for the proof of the Friedberg-Muchnik theorem, can be generalized to an infinite priority argument to study the α-degrees. The main drawback of this generalization of recursion theory is that it altogether leaves behind the idea of computation as encompassed by Turing machines. However, as shown by Koepke in [Koe05], it is possible to devise a model of Turing machines, with a tape made of α cells and that computes for at most α steps, that provides the missing machine-like part of α-recursion. And this kind of model of α-Turing machines is based, to some extent, on the more fundamental infinite time Turing machine developed by Hamkins and Lewis in [HL00], which is the main object of study of this thesis.

Chapter 4

Infinite time Turing machines

In this chapter we present the theory of infinite time Turing machines (ITTMs) as introduced by Hamkins and Lewis in [HL00]. We then give an overview of the state-of-the-art generalizations of this model of infinite machines. Those generalizations, with the one we introduce in Chapters 5 and 6, constitute a step toward higher-order infinite time Turing machines.

In Chapter 3, we introduced three fundamental objects: the finite Turing machine (which can be used to ground the theory of recursion and the theory of Turing degrees), arithmetical logic and set theory. In particular, in Theorem 3.3.4 and Theorem 3.3.16, we have seen the deep links between the ω_1^{CK} first Turing degrees and the of arithmetical and hyperarithmetcial sets. In the presentation of admissible sets and α-recursion, we strayed away from the peaceful realm of natural numbers to study admissible set, which are from a computability point of view a generalizations of the set of natural numbers. Proposition 3.3.28 and Proposition 3.3.34 shown the interest of considering the constructible hierarchy and with it, On. With this, we saw how the hierarchy of Turing degrees may be continued after ω_1^{CK}. And to complete the picture, with the correspondence between finite Turing machine and arithmetical logic in mind, one may feel like that there misses a fourth fundamental concept, namely a computational model that somehow deals with admissible set or with ordinals.

And the model of infinite time Turing machines is a relatively simple model of computation that can compute through the ordinals and that can be seen to bridge this gap. Such machines may either halt at some ordinal stage or "compute endlessly as the ordinals fall one after another through the transfinite hourglass", as put in [HL00]. This way, they have a privileged relation with set theory and in particular with the constructible universe. After the introduction of this model, we will start in this chapter with its study from the point of view of the analytical hierarchy, as done in [HL00]. While this already gives a good idea of the power of those machines, we will see how their power sits astride strictly between two levels of this hierarchy and how the constructible hierarchy provides

a more adequate point of view.

4.1 Hamkins and Lewis' infinite time Turing machine

The ITTM (infinite time turing machine) is a model of computation that computes through the hierarchy of the ordinals. It is built upon the classical finite Turing machine. Imagine a finite Turing machine m that somehow managed to compute up to some limit ordinal stage α. As it computed up to this stage, it is in some state q, has some real x (an element of $\{0,1\}^{\mathbb{N}}$, which we more conveniently write ${}^{\omega}2$, as in the previous chapter) written on its tape and its head is on some cell i. Following [HL00], we call the tuple of those data the *snapshot* of the machine at this computation stage. Given this snapshot, the computation step from stage α to $\alpha+1$ is simply determined by the program or the code of m. And so is the computation up to stage $\alpha+k$ for any finite k.

We see that the question left out in this description is: how does the machine "somehow computes" up to a limit stage? And it is crucial to observe that this question is the most important one, if not the only one, that needs to be answered to have an effective infinite model of computation. The following definition is Hamkins and Lewis' proposition of an answer. This defines the model of ITTMs.

Definition 4.1.1 ([HL00])**.** An *infinite time Turing machine (ITTM) m* is a machine which has the same structure as a three tapes finite Turing machine with a distinguished state q_{limit} and with an input, an output and a working tape such that:

- At any stage $\alpha+1$, the snapshot of the machine M is determined by the snapshot of M at stage α and by the program of the machine m, as for classical finite TMs.
- At any limit stage α, the head of M is on the first cell, its state is set to the distinguished state q_{limit} and for all cell i, writing C_i the function that maps an ordinal stage to the value of the cell i at this stage:

$$C_i(\alpha) = 0 \longleftrightarrow \exists \beta < \alpha \, \forall \delta \in [\beta, \alpha[\, C_i(\delta) = 0 \tag{4.1}$$

  That is the value of the cell i at limit stage α is set to 0 if and only if there is some stage $\beta < \alpha$ at which the value of cell i stabilized on 0 up to stage α.

Observe that with this definition, an ITTM is simply given by the code of a finite Turing machine. Hence, most of the finite definitions generalize in a natural way. Observe moreover that for this reason, the sole data of a limit rule (as we may have imagined different limit rules) is enough to define a model of infinite machines. We will come back to this possibility of having different limit rules later on. The fact that the structure of an ITTM is the same as that of a classical finite Turing machine explains why the concept of snapshot and successor snapshot of an ITTM (Definitions 4.1.2 and 4.1.3) are defined as in the classical Turing machine setting (Definitions 3.1.5 and 3.1.6).

Definition 4.1.2 (Snapshot of an ITTM, [HL00])**.** Given an ITTM $\mathcal{M} = \langle \Sigma, Q, q_0, \delta, F \rangle$, a snapshot of $\mathcal{M}$ at some computation stage is a 3-uple $\langle x, q, k \rangle$ where x describes the content of the tape, q the state of the machine and k the integer position of the head. A snapshot is final if $q \in F$.

Definition 4.1.3 (Successor snapshot, [HL00])**.** Given an ITTM $\mathcal{M} = \langle \Sigma, Q, q_0, \delta, F \rangle$ and a snapshot of the machine $s = \langle x, q, k \rangle$ such that $q \notin F$, the successor snapshot s' is the only snapshot produced by applying the transition function on the cell k which contain the symbol $x[k]$. That is, writing $\delta(x[k], q) = (a, q', D)$

$$s = \langle x', q', k + d \rangle \text{ with } \begin{cases} x'[k'] = x[k'] \text{ for } k' \neq k \\ x'[k] = a \\ d = 1 \text{ if } D = R \\ d = -1 \text{ if } D = L \text{ and } k > 0 \\ d = 0 \text{ if } D = L \text{ and } k = 0 \end{cases}$$

Definition 4.1.4 (Limit snapshot, [HL00])**.** Given an ITTM $\mathcal{M} = \langle \Sigma, Q, q_0, \delta, F \rangle$ and an ordinal stage α, the limit snapshot of the machine at stage α is $s_\alpha = \langle x_\alpha, q_{limit}, 0 \rangle$ where x_α is defined as follow:

$$x_\alpha[i] = 0 \longleftrightarrow \exists \beta < \alpha \forall \delta \in [\beta, \alpha[\, x_\delta[i] = 0$$

Definition 4.1.5 (Run of an ITTM, [HL00])**.** Given a Turing machine $\mathcal{M} = \langle \Sigma, Q, q_0, \delta, F \rangle$, the *run* of this Turing machine with input x is an ordinal-indexed sequence $(s_\iota)_\iota$ (possibly of length On) of snapshot such that, writing $x^\frown b$ for x padded with infinitely many b to the right if x is finite, else for x itself:

- $s_0 = \langle x^\frown b, q_0, 0 \rangle$
- For all ι, if $s_{\iota+1}$ exists, then it is the successor snapshot of s_i. It does not exist if and only if s_i is a final snapshot.
- For all limit ι, s_ι is the limit snapshot produced by the sequence of snapshots $(s_{\iota'})_{\iota' < \iota}$.

We say that a machine *halts when run with input* x if its sequence of snapshots has length $\nu + 1$ for some $\nu \in On$ (there must be a final snapshot which is also the last one, so the length must be a successor ordinal.) In this case, writing $s_\nu = \langle y^\frown b, q, k \rangle$ the last snapshot of the sequence, we say that the machine computed for ν steps and that its output is y. A *partial run* of a machine is any initial segment of its run sequence.

One almost obvious but very important fact is that it is possible to "run" a machine inside L_α for some limit ordinal α. This means that L_α is an adequate domain of definition for the α first steps of an ITTM. More formally:

Proposition 4.1.6 ([HL00]). *There exists a Σ_2 formula $\varphi_{\sup}(x_1, x_2, x_3)$ such that, for a limit ordinal α, a machine m, a cell i and an input y,*

$$C_i(\alpha) = 1 \longleftrightarrow L_\alpha[y] \models \varphi_{\sup}(i, m, y)$$

Proof. While Equation 4.1 provides a natural way of defining $C_i(\alpha)$, its inductive nature hinders it direct translation as a formula of set theory.

To translate it into a formula of set theory, we need first to provide the formula for a function that given some δ, defines in L_α the history (seen as a set) of the machine up to stage δ (stage δ included), if it exists. Given this formula, we can show by induction on α that this history indeed exists in L_α for all $\delta < \alpha$. We show more thoroughly in the proof of Proposition 5.3.37 in Chapter 5 that such a formula exists and that it is Σ_1. Writing it $\eta(m, y, \delta)$, it corresponds to the whole history of the computation of machine m with input y up to stage δ, $\eta(m, y, \delta)[i]$ corresponds to the history of the cell i and $\eta(m, y, \delta)[i][\delta]$ to $C_i(\delta)$. With it we can define $\varphi_{\sup}$:

$$\varphi_{\sup}(i, m, y) := \exists\alpha\, \forall\delta\, \forall h^\delta\, (\alpha < \beta \wedge h_i^\delta = \eta(m, y, \delta)[i] \implies h_i^\delta[\delta] = 0)$$

And it is clear that this formula is Σ_2 and satisfies the relation stated in the proposition. □

We will see in Proposition 4.1.9, which deals with the asymptotic behavior of the machines, how the previous proposition can be useful. Indeed, machines that do not halt may have different asymptotic behaviors. The most general one, and unavoidable for machines that do not halt (this is Proposition 4.1.9), is the fact that they will be looping at some point.

Definition 4.1.7 (Looping, [HL00]). For an ITTM m, we say that m is *looping* when the machine never halts and some interval of computation $[\alpha, \beta[$ repeats itself through the whole computation after stage α. That is, writing $\beta = \alpha + \delta$, for any stage $\nu \geqslant \alpha$, writing $\nu = \alpha + \delta \cdot \mu + \nu'$ with μ maximal (and so $\nu' < \delta$), the snapshot of m at stage ν is the same as that of m at stage $\alpha + \nu'$. In this case we say that m started looping at stage β.

This definition, as is, is quite unpractical as it involves inspecting the run of the ITTM through all of On. The following result fortunately allows us to only look at some intervals of computation.

Proposition 4.1.8 ([HL00]). *An ITTM is seen to be looping when there are two limit stages α and β sharing the same snapshot such that between those two stages, cells that are set to 0 in the two limit stages stay so between those. That is, such that*

$$\forall i\, \forall\delta \in [\alpha, \beta](C_i(\alpha) = 0 \implies C_i(\delta) = 0)$$

Proof. Suppose there are α and β as described. As the snapshot at stage α and stage β are the same, starting from stage β, the computation behaves as it did from stage α. Writing $\beta = \alpha + \delta$, this will yield again the same snapshot at stage $\alpha + \delta \cdot 2$ and again at any stage $\alpha + \delta \cdot n$. Now, what happens at stage $\alpha + \delta \cdot \omega$? At this stage, the history gets slightly more complicated. Still, if $C_i(\alpha) = 0$, then for all $\alpha' \in [\alpha, \alpha + \delta \cdot \omega[$, $C_i(\alpha') = 0$ and by the limit rule, $C_i(\alpha + \delta \cdot \omega) = 0$. Conversely, if $C_i(\alpha) = 1$, there are cofinally 1's in the cell i up to stage $\alpha + \delta \cdot \omega$, at least at stages $\alpha + \delta \cdot n$ for $n \in \omega$. Hence, in this case, $C_i(\alpha + \delta \cdot \omega) = 1$. As $\alpha + \delta \cdot \omega$ is also a limit stage, it shares the same snapshot with stages α and β. This provides the ingredients for an inductive proof through On: for a limit μ such that α and $\alpha + \delta \cdot \mu$ share the same snapshot, the segment will repeat until stage $\alpha + \delta \cdot \mu\omega$ and with the same reasonning on cells set to 0, it will produce again the same snapshot. This describes the entire machine behavior through On and shows that is it looping. □

4.1.1 Power and limit of ITTMs

As the structure of an ITTM is exactly that of a classical Turing machine, many usual results are directly inherited from it. Among those are for example the s-m-n theorem or Kleene's recursion theorem or the fact that the (infinite) halting problem is (infinite time) undecidable. And these three results are proved with the same proof as in the classical setting. The fact that the halting problem is undecidable, combined with the following proposition, already gives an upper bound for the power of ITTMs.

Proposition 4.1.9 ([HL00])**.** *An ITTM either halts or starts looping stage before stage* ω_1.

Proof. Let m be an ITTM that does not halt before stage ω_1. We show that it starts looping before stage ω_1. By the Löwenheim-Skolem Theorem 3.2.71, we can find two countable stages α and β such that $L_\alpha \prec L_\beta$. By proposition 4.1.6, this means that running m in L_α and L_β produces the same limit snapshot as for all i, $L_\alpha \models \varphi_{\text{sup}}(i, m, y) \longleftrightarrow L_\beta \models \varphi(i, m, y)$. That is, stages α and β have the same snapshot. Now, let some cell i such that $C_i(\alpha) = 0$. As the snapshot are the same, we also have $C_i(\beta) = 0$. We want to show that between stages α and β the cell i is always set to 0. Suppose it false. Then, for any $\alpha' < \alpha$, the formula

$$\exists \delta > \alpha' \, C_i(\delta) = 1$$

with α' as parameter holds in L_β. As $\alpha' \in L_\alpha$, this also holds in L_α by elementarity. This show that 1's in the cell i are cofinal up to stage α, which contradict the fact that $C_i(\alpha) = 0$. □

This implies that to look whether a machine halted, it is enough to look at countable ordinal stages. So a machine halts if and only if there exists a real encoding a well-ordered

sequence of snapshot itself describing a countable run of the machine. Equivalently, a machine halts if and only every real encoding a partial run of a machine does not show the machine to be looping. Hence, this yield Proposition 4.1.10.

Proposition 4.1.10 ([HL00])**.** *There is a Δ^1_2 set which is not decidable with ITTMs.*

Proof. Simply consider the real $h = \{m \mid \text{the ITTM } m \text{ halts on the empty input}\}$ describing the halting problem. As explained in the previous paragraph:

$$\begin{cases} m \in h \longleftrightarrow \exists x\,(x \text{ describes a well-ordered sequence of snapshots forming a run of } m) \\ m \in h \longleftrightarrow \forall x\,(x \text{ describes a well-ordered sequence of snapshots forming a partial run of } m \\ \qquad\qquad \Longrightarrow m \text{ is not seen to be looping in the partial run } x) \end{cases}$$

As WO is a Π^1_1 predicate and checking whether a well-ordered sequence of snapshot is a run or, given a run, whether a machine is seen to be looping are Δ^1_0 predicate, h is Δ^1_2. □

On the other side, that is looking for an arithmetical lower-bound on the power of ITTMs, we can first establish the following result.

Proposition 4.1.11 ([HL00])**.** *The Σ^0_n sets of natural numbers are infinite time decidable, that is decidable by an ITTM.*

Proof. Let A be a Σ^0_n defined by some formula $\varphi(x_1, x_2)$ with a parameter $p \in \omega$. That is $i \in A$ iff $\varphi(i, p)$. If $n = 0$, then $\varphi(i, p)$ is Δ^0_0 and the machine can directly evaluate φ. Else, $n > 0$ and $\varphi(i, p) = \exists j \psi(i, j, p)$. By induction on the number of quantifiers in the formula, a machine can iterate through all $j \in \omega$ and decide whether $\psi(i, j, p_1, \ldots, p_n)$ in order to decide whether $i \in a$. □

The simplicity of this result may give us an inkling of how much more than classical Turing machines can ITTMs achieve. To go further, Hamkins uses the following result in tandem with the well-known fact that WO is Π^1_1-complete, shown in Proposition 3.3.15.

Proposition 4.1.12 ([HL00])**.** WO *is infinite time decidable.*

Proof. We describe an ITTM that, given as input some real x, decides whether x encodes a well-order.

First, the machine needs to decide whether x is a total order. As seen in the proof of Proposition 3.3.14, deciding this is Δ^1_0. With the same reasoning as in the proof of Proposition 4.1.11, for a Δ^1_0 formula φ and a real x, a machine with x as input can inductively decide whether $\varphi(x)$.

Then, what is left for the machine is to decide whether the total order described by x is a well-order. We write $<_x$ this total order and dom_x for its domain. To check whether it is a well order it is enough to check whether dom_x can be "counted through". That is

if dom_x has a least element, i_0, and if $\mathrm{dom}_x - \{i_0\}$ has a least element, i_1, ... and then whether $\mathrm{dom}_X - \{i_0, i_1, i_2 \ldots\}$ has a least element and so on until dom_X is exhausted.

So we now show that an ITTM can decide, given a total order x, whether dom_x has a least element: it goes through the i's in dom_x (that is such that $\langle i, j\rangle \in x$ or $\langle j, i\rangle \in x$ for some j). Each time it encouters such a i, it first flashes (i.e. turns on and off) some distinguished cell c_0, acting as a flag. Then, for this i, it turns on some flag c_1 and checks whether there is j such that $j <_x i$: that is, it again enumerates through the natural numbers and for each j checks whether $\langle j, i\rangle \in x$. If it finds one, i is not the least element and it turns off c_1 and goes on with the first enumeration, along the i's. If it does not find one: first, how does the machine realizes it? If it does not find one, it computed for ω steps looking for one. So, it reached the next limit step and c_1 is still on but c_0 is off, which is enough to distinguish this particular stage. In this case, the machine knows that this i is the least element in $<_x$ and so that dom_x has a least element. As for the last case, what happens when dom_x has no least element? Again the enumeration on the i's will exhaust ω and the machine will have computed for ω steps (as for each i it finds a $j <_x i$ in finitely many steps). However, at this limit stage, both c_0 and c_1 are turned on, which again let the machine distinguish this last case in which it answers negatively.

Now, an ITTM can decide WO as follows: if $<_x$ is the empty total order, then $x \in \mathrm{WO}$. Else, the machine looks whether dom_x has a least element, deletes it (in ω steps) and goes on until dom_x is either empty (in which case it answers positively) or does not have a least element (in which case it answers negatively). Observe that this needs a third flag c_2, flashed each time a least element is removed, to further distinguish limit stages happening when the machine deleted some limit ordinal amount of least elements in x. When c_2 is activated, the machine knows that c_0 and c_1 being actived are "false flag" and it simply continues its counting through of x. □

Corollary 4.1.13. *Π^1_1 sets (and consequently Σ^1_1 sets) are infinite time decidable.*

Proof. Let A be a Π^1_1 set of reals and e the index of the recursive function of Proposition 3.3.15. Then given some real x, a machine can compute the real $\{i \in \omega \mid \varphi^x_e(i) = 1\}$ and decides whether it encodes a well-order to decides the membership of x in A. □

However, this lower-bound is not tight. One way to see this is the following: in the next section, we will consider ordinals α such that an ITTM can output a real describing well-order of order type α. In particular, we will be able to show that the least α which is limit of admissible ordinals is writable. This α is also the least ordinal such that $L_\alpha \cap \mathcal{P}(\omega)$ is a model of Π^1_1-comprehension (see [Sim09, Theorem VII.1.8 on p. 246 and theorem VII.5.17 on p. 292 and notes to §VII.5 on p. 293], as referenced in [Mad17]). As such, we can show that $L_\alpha \cap \mathcal{P}(\omega)$ is not Π^1_1 decidable: suppose it were, then using the Π^1_1 predicate normal form, there is some m such that $x \in L_\alpha \cap \mathcal{P}(\omega)$ if and only if $\forall y\, \exists k\, T(m, \overline{\langle x, y\rangle}^k, k, 0)$. Then, we can consider the real d constructed bit after bit as follows: suppose that $\bar{d}_k$,

the truncation of d to its k first bit was defined then: $d[k+1] = 1$ if there exists a real starting with those $k+1$ first bits for which $\varphi_m^x \uparrow$, and otherwise $d[k+1] = 0$. Writing $d = \langle d', d'' \rangle$, this produces a Σ_1^1-definable real d' such that:

$$\begin{cases} \forall k \neg T(m, \overline{\langle d', d'' \rangle}^k, k, 0) \text{ (by construction)} \\ d' \in L_\alpha \cap \mathcal{P}(\omega) \text{ (by } \Pi_1^1\text{-comprehension)} \end{cases}$$

However, since α is writable, a machine can decide $L_\alpha \cap \mathcal{P}(\omega)$ in the following fashion: given some x, it writes a code for α, then uses it to produce a code for L_α and then simply checks whether x, seen as a real, is in L_α.

This shows that the power of the ITTMs lies somewhere strictly between Π_1^1 and Δ_2^1. Moreover, the previous argument hints at the interest of now taking a set-theoretic point of view to better understand the power of those machines.

4.1.2 Writing, eventually writing and accidentally writing

Previous section was focused on deciding sets from an arithmetical perspective. This is the direct generalization of what was doable with classical Turing machine and classical Turing machines computing with oracle. However, a new aspect appears with ITTMs, namely the possibility to write, i.e. to output, reals. Indeed, as a machine may go on for way more than ω steps before halting, it has more than enough time to write a proper real on its output. We define three ways for a real to be writable.

Definition 4.1.14 (Writable reals, [HL00])**.** A real x is *writable* if there is an ITTM which, when computing from the empty input, halts with x written on its output.

Definition 4.1.15 (Converging, [HL00])**.** For m an ITTM computing on some input y we say that m *converges* or *converges definitively* when it never halts and when after some stage its output tape is never modified again. We say that m *converges up to some limit ordinal* α when it does not halt before stage α and after some stage $\beta < \alpha$ its output tape is never modified before stage α.

Definition 4.1.16 (Eventually writable reals, [HL00])**.** A real x is *eventually writable* (e.w.) if there is an ITTM which converges when computing with the empty input and which has x written on its output when the content of its output tape stabilized.

Definition 4.1.17 (Accidentally writable reals, [HL00])**.** A real x is *accidentally writable* (a.w.) if there is a machine computing with the empty input that has written x at some stage on any of its tapes.

Further, as we can encode ordinal with reals, in the sense of Definition 3.2.27, an ordinal can also be said to be writable, e.w. or a.w.

Definition 4.1.18 (Writable ordinals, [HL00])**.** An ordinal α is *writable* (resp. *eventually writable* and *accidentally writable*) if some real x that encodes α is writable (resp. e.w. and a.w.). We write λ, ζ and Σ respectively for the supremum of the writable, e.w. and a.w. ordinals.

We give the table of Figure 4.1 to help the reader. The notion of "clockable" ordinal will be introduced in Definition 4.1.24.

	simply	eventually	accidentally
writable	λ	ζ	Σ
clockable	γ		

Figure 4.1: Greek letters associated to each definition.

It is clear that a writable, e.w. or a.w. ordinal is countable and so that that λ, ζ and Σ are well-defined ordinals and at most equal to ω_1. Further, λ and ζ are countable ordinals: there are countably many machines, a machine may write or e.w. at most one ordinal and the ordinal it writes is countable as its encoded in ω; hence the supremum of those ordinals is countable. For Σ, a single computation may a.w. more than one ordinal. But by Proposition 4.1.9 it writes at most countably many ordinals and the same argument holds.

Manipulating those three different concepts, we will soon need to consider more and more involved computations of ITTMs simulating other ITTMs. The question of the simulation will be discussed thoroughly in the next chapter. Still, to begin in this direction, observe that it is rather straightforward for an ITTM to simulate another: as we know that classical finite Turing machines can simulate another finite TMs, the only tricky part may be limit stages. But if one cell of the simulated machine is simulated by one cell of the simulating machine, the lim sup rule applied in the simulating machine directly simulates itself. That is, it behaves as it would have done in the simulated machine. Lastly, at limit stages, cells of the simulating machine used to keep track of the simulation (as in the classical model) may often be set to wrong or incoherent values. However, as the simulated machines knows that is reached a limit stage (remember that at limit stages, the machine is in a distinguished state q_{limit}), it can simply tidy up those cells (which may take up to ω steps) and then go on with the simulation.

We will see how those simulation can be used in the next proposition. While it could be shown without simulations (by simply modifying the machines that we will consider), this offers a more level and coherent presentation of the question of writable ordinals w.r.t. that of simulation.

Proposition 4.1.19 ([HL00])**.** *The writable ordinal form an initial segment of On. And so do the e.w. and the a.w. ordinals.*

Proof.

– For the first case, we need to show that if α is writable, any $\beta < \alpha$ is writable as well. Let m be the machine writing a real x describing a well-order $<_x$ of order-type α. As already seen, $\beta < \alpha$ implies that there is some $i \in \mathrm{dom}_x$ such that the set $\mathrm{dom}_x^i = \{j \in \mathrm{dom}_x \mid j <_x i\}$ ordered by (the restriction of) $<_x$ has order-type β. We consider the following machine m': it simulates m and when m halts it copies the order-type on the output of m, truncated below i, to its own output. As i is an natural number it can be hardcoded in m'. Once the truncated order-type has been copied, m' halts and it has written a code for β.

– If α is e.w. and $\beta < \alpha$, we proceed similarly: we write m for the machine that e.w. α and i for the order of β in the code of α and we consider the following machine: it simulates m and each time the output of m is modified and a well-order appears in it (this can be checked by Proposition 4.1.12) with i in its domain, the machine copies this well-order, truncated below i on its output tape, possibly erasing a previous written well-order. Once m stabilized in its simulation inside m', it has written on its output tape a well-order for α and m' writes its truncation on its own output tape and does not modify it further as the simulated output tape of m is not modified further. Hence m' eventually writes β.

– The a.w. case is similar to the e.w. case: the simulating machine simply copies and truncates any well-order that may appear in the simulation of m.

□

Further, the following inequality shows that those constants are actually meaningful for the study of ITTMs. Now, to prove it, we need a slightly more involved way of simulating other machines. We consider the universal machine $\mathcal{U}$ that simulates in parallel every other machines. We will come back to the existence of such a machine in the next chapter, but once we are convinced that it is possible to simulate a single machine, this is only a small step further. Indeed, the universal machine works as follows: it separates its working tape into ω virtual tapes (taking for example cells $(3^i)_i$ for the first, cells $(5^i)_i$ for the second and so on) and on each virtual tape it simulates one ITTM. By dovetailing, that is simulating k steps of the k first machines, $k+1$ steps of the $k+1$ first machine etc. it simulates ω steps of all ω machines in ω steps, which is a rather satisfying "parallel" computation. And further again, as $\mathcal{U}$ is an ITTM, its most frequent use will be being simulated it in another computation.

Proposition 4.1.20 ([HL00])**.** $\lambda < \zeta < \Sigma$

Proof.

$\lambda < \zeta$: As already seen, there is some ordinal stage at which every machine either halted or is looping. Hence, at this stage every writable ordinal as been written. Now

consider the following computation: a machine simulates $\mathcal{U}$ and at every stages of $\mathcal{U}$ it looks at all machines that halted and whose output is an ordinal and writes on its output the sum (ordered by the indices of the machines) of those ordinals. When all machines halted, this sum is at least as great as λ and moreover it never changes as no other machine will halt. Hence an ordinal at least greater than λ is e.w. Invoking Proposition 4.1.19 if the ordinal e.w. by the previous computation is strictly greater than λ, this shows that λ itself is e.w.

$\zeta < \Sigma$: With the same reasonning, at some point all converging machine converged and after this point all e.w. ordinal appear in the tape of some converging machine. So, simulating $\mathcal{U}$ and writing the sum of any ordinal appearing on the output of any machine is enough to a.w. at some point an ordinal greater than ζ.

□

Those constants help hierarchizing the different ordinals that may appear in ITTMs with respect to how good a grasp those machines have on them. We will shows later that this classification naturally extend to sets. Let us first remind how sets are handled in this setting.

Definition 4.1.21 (Encoding of sets, [HL00])**.** We say that a real x encodes as set a when x describes a transitive relation E on ω such that $(\omega, E) \simeq (\mathrm{TC}(\{a\}, \in)$, where TC denotes the transitive closure. As with ordinals, a set a is writable (resp. e.w. and a.w.) by an ITTM when a code for a is writable (resp. e.w. and a.w.) by an ITTM.

This definition and Proposition 4.1.22 allow us to manipulate set in a convenient way as in the proof of Proposition 4.1.23. This latter proposition also gives an idea of the kind of high closure properties that those constant have.

Proposition 4.1.22 ([HL00])**.** *Any set $a \in L_\lambda$ is writable. Similarly, any set L_ζ is e.w. and any set in L_Σ is a.w.*

Proof. We show this for the first case. Both other cases are obtained by adapting the technique of the first case, as in previous proofs.

Let $a \in L_\lambda$. First, λ is a limit ordinal (as if some ordinal α is writable, so is $\alpha + k$ for any integer k.) Hence, there is some $\alpha < \lambda$ such that $a \in L_\alpha$. As the writable ordinals form an initial segment, α is writable. So we can design a machine that writes α and then that uses it to split its working tape into α virtual tapes. Then we explain how it can use those α virtual tapes to inductively build L_α: at any stage, suppose that it stored L_β for $\beta < \alpha$ on the β^{th} virtual tape. Using the code for L_β, it can compute the truth value in L_β of any set-theoretic formula in the language of L_β. Hence, it can compute a code for all the element of $L_{\beta+1}$. Using $\beta + 1$ (a code for $\beta + 1$ can be extracted from the code of α), it then organizes those codes into a code for $L_\beta + 1$, as per the encoding described

in Definition 4.1.21. At some limit stage δ, it uses a code for δ and the codes previously written for all the L_β's for $\beta < \delta$ to write a code for L_δ. At the α^{th} step of this inductive process, the machine produced a code for L_α. As $a \in L_\alpha$, the code for a correspond to the truncation of the code of L_α below some i: the code of L_α can be seen as a partial order on ω describing a tree, so below each i in this order spans another tree describing a set. Thus, with this i hardcoded in the machine, a is writable as well. □

Proposition 4.1.23 ([HL00])**.** *The ordinal λ is admissible and a limit of admissible ordinals.*

Proof. As already seen, to show that λ is admissible, it is enough to show that $L_\lambda \models \Delta_0$-collection. So suppose that there is some set $a \in L_\lambda$ such that $\forall x \in a \exists y\, \varphi(x, y)$ and that those y's can't be collected inside L_λ. This implies that the ranks (i.e. least β such that $y \in L_\beta$) of the y's are unbounded in L_λ. This induces a sequence of ordinals unbounded in L_λ that we will us to reach a contradiction.

First, as $a \in L_\lambda$, by the previous result, a code for a is writable. Then we consider the following computation: it writes a on some part of its working tape and then it simulates $\mathcal{U}$ on another part. As all machines are simulated in $\mathcal{U}$, all a.w. ordinal appear at some point in the simulation of $\mathcal{U}$. Hence, it can be used to enumerate the a.w. ordinals. So, using $\mathcal{U}$, the machine we are describing looks for an ordinal β such that for each x in a, there is $y \in L_\beta$ such that $\varphi(x, y)$. Such an ordinal exists as λ itself satisfies those requirements. And as said, λ is supposed to be the least such. Hence, at some point, this machines finds such a suitable ordinal $\beta \geqslant \lambda$ and writes it on it output and then halts, which is a contradiction. This shows that λ is admissible.

Further, observe that given some limit ordinal α, deciding whether α is admissible is doable with an ITTM: it computes a code for L_α and enumerate all formulas with parameters in L_α to see whether the axiom of Collection holds for each formula. That is whether the code for a set that each formula produces by collection corresponds (by relation isomorphism) to the code of a set in L_α. Once it checked whether it holds for all formulas, it can decide whether α is admissible. Hence, a machine can look for the first admissible ordinal ω_1^{CK} and write it. So $\omega_1^{\text{CK}} < \lambda$. But it can also look for the second one, ω_2^{CK}, the third one, ω_3^{CK} etc. and all of those are writable, and so strictly less than λ. Further again, for a writable β, it can write it down and look for ω_β^{CK}, the β^{th} admissible ordinal (or limit of the β first), and write it. Hence for all $\beta < \lambda$, $\omega_\beta^{\text{CK}} < \lambda$. In particular, $\omega_\lambda^{\text{CK}} = \lambda$ and λ is an admissible ordinal limit of admissibles. □

Naturally, Proposition 4.1.22 begs the question of whether the converse holds. That is, for the first case, is a writable set or a writable real in L_λ? Observe that if we manage to show that any writable real x is written before stage L_λ then, as those machine can be run in L_λ this x would be definable at some constructible stage below λ and so it would be in L_λ. So the question reduces to: are writable reals writable before stage L_λ? And it

is clear that this question may be (and actually is) closely linked with another question: do all machine that halt, halt before stage λ? In case some machine halts after stage λ, it very well may also write after this stage a real that isn't written earlier by any other halting machines. This drives us toward the question of clockable ordinals.

4.1.3 Clockable ordinals

Definition 4.1.24 (Clockable ordinals, [HL00])**.** An ordinal α is *clockable* if there is an ITTM computing with the empty input that halts at stage α. That is it computes for α steps and then, on its next transition, the machine reaches one of its halting states, $q_{halt} \in F$. We write γ for the supremum of the clockable ordinals.

A first observation is the following: the clockable ordinal do not form an initial segment of On. Indeed, there are gaps: that is, there are ordinals (and even segment of ordinals) below γ that are not clockable.

Proposition 4.1.25 ([HL00])**.** *For $k, k' \in \omega$, the ordinal $\omega \cdot k + k'$ is clockable.*

Proof. If $k = 0$, a machine that computes for $k' + 1$ steps clocks k'. If $k > 0$, observe that a machine can detect limit stages by waiting for the stage q_{limit}. Hence, it can count for k limit stages by turning on cell $k - 2$ after the first limit stage, cell $k - 3$ after the second, ... and cell 0 after the $k - 1^{th}$ stage. Doing this, stage $\omega \cdot k$ is the first limit stage at which cell 0 is turned on. Then, counting $k' + 1$ steps before halting allows it to clock stage $\omega \cdot k + k'$. □

Proposition 4.1.26 ([HL00])**.** *If α and β are clockable, then so is $\alpha + \beta$.*

Proof. Let m_α and m_β be the machine clocking resp. α and β. Suppose first that $\beta \geqslant \omega^2$. In this case we can consider the machine built from m_α and m_β that behaves as m_α until stage α, and then, instead of halting, that erases its tapes in ω stages and after that behaves as m_β; hence clocking $\alpha + \omega + \beta = \alpha + \beta$ for $\beta \geqslant \omega^2$. If $\beta < \omega^2$, it is enough either to count for a finite amount of steps or to erase only the first k cells for some finite k and to apply the technique of the previous proposition. □

Proposition 4.1.27 ([HL00])**.** *Every recursive ordinal is clockable.*

Proof. Let α be a recursive ordinal. If $\alpha < \omega^2$, we can invoke Proposition 4.1.25.

If α is a multiple of ω^2, as α is a recursive ordinal, using the finite classical TM that decides a well-order of order type α, an ITTM can write a code for α in ω steps. Then, it can count through this code as in the proof of Proposition 4.1.12. However, it always takes $\omega \cdot 2$ steps to look for the least element of the well-order and then to erase it from the domain of the well-order. And when the domain of the well-order is empty, it takes another ω steps in order to check that it is empty. We can remove those last ω steps with this trick from [HL00]: the machine keeps track of the least bit (w.r.t. the natural order

on ω) activated in the encoding of the well-order and each time this least bit is turned off, it flashes a flag on the first cell of its working tape. This flag is activated at a limit iff it was flashed ω times iff the well-order is the empty order. Hence, this allows us to clock $\omega \cdot 2 \cdot \alpha$ which is equal to α under the hypothesis that α was a multiple of ω^2.

Eventually, if $\alpha > \omega^2$ and α is not a multiple of ω^2, then $\alpha = \beta + \beta'$ with β multiple of ω^2 and $\beta' < \omega^2$ and α is clockable by Proposition 4.1.26. □

Proposition 4.1.28 ([HL00])**.** *The ordinal ω_1^{CK} is not clockable.*

Proof. As ω_1^{CK} is an admissible ordinal, $L_{\omega_1^{\text{CK}}}$ satisfies Σ_1-collection. We shows that this makes it impossible for ω_1^{CK} to be clockable.

Suppose that some machine clocks ω_1^{CK}. By definition, it means that at stage ω_1^{CK}, given its state (which is q_{limit}) and the content of the first cell of each of the three tapes (that is those under the heads at a limit stage), it decides to halt. This means that ω_1^{CK} is the first limit stage at which those three first cells have this content. First there must be a 1 among those cells as otherwise, the three 0's stabilized before stage ω_1^{CK} and so appear at some limit stage between this stabilization stage and ω_1^{CK}. So, at least one cell reads 1. Further, with the same reasoning one of those stage reading 1 did not stabilize on 1 before stage ω_1^{CK} but rather was cofinally alternating between 0 and 1 before stage ω_1^{CK}. So, we write $\beta < \omega_1^{\text{CK}}$ for the stage at which all cells that stabilize among those three cells do stabilize. And we write $\beta_1 > \beta$ the stage at which each of the cell that does not stabilize among those three cells (so among those reading 1) are turned to 1 at least once after stage β. Then $\beta_2 > \beta_1$ is defined in the same way w.r.t. β_1. Let $\beta_\omega = \bigcup \beta_i$. β_ω is a limit stage less or equal to ω_1^{CK} and by construction its first three cells have the same value as that of stage ω_1^{CK}. So $\beta_\omega = \omega_1^{\text{CK}}$.

But the application $i \mapsto \beta_i$ is easily seen to be Σ_1 and its range is ω_1^{CK} itself, which contradicts Σ_1-collection. □

Proposition 4.1.29 ([HL00])**.** *However, the ordinal $\omega_1^{\text{CK}} + \omega$ is clockable.*

Proof. One way to show this is simply to look for the first non-clockable limit ordinal as done in [DL19]. Indeed, we saw with the two previous propositions that ω_1^{CK} was the first non-clockable ordinal. Hence, an ITTM which simulates the universal machine can use this description of stage ω_1^{CK} in order to look for it.

More precisely, we consider the following machine $\mathcal{M}$: it simulates the universal machine $\mathcal{U}$ and at any limit stage α of $\mathcal{U}$, it checks whether at least one simulated ITTM stops at this stage α. That is, whether one of those ITTM reaches a final state q_f on its transition from stage α to $\alpha + 1$. If one is found, it is found in finitely many steps and $\mathcal{M}$ goes on with the simulation of $\mathcal{U}$ just after it is found–which ensures that $\mathcal{U}$ is never more that finitely many stages late. If at some limit stage α of $\mathcal{U}$, no halting machine is found, $\mathcal{M}$ halts. And this happens when $\alpha = \omega_1^{\text{CK}}$. But then, $\mathcal{M}$ still needs ω steps to observe that none does stop–and so it halts at stage $\omega_1^{\text{CK}} + \omega$.

Observe moreover that, by keeping track of the order of the halting time of the machine, as done in [DL19], $\mathcal{M}$ could also have written down a code for the ordinal ω_1^{CK} at stage ω_1^{CK} before halting at stage $\omega_1^{\mathrm{CK}} + \omega$.

□

Hence, a first gap on the clockable ordinals starts at stage ω_1^{CK} and can be shown (see [HL00, Lemma 3.3]) to be actually of length ω, that is no ordinal between ω_1^{CK} included and $\omega_1^{\mathrm{CK}} + \omega$ not included is clockable. See [HL00, pp. 579–582] for more results on the gaps.

Still, beyond this difference, that is beyond the fact there are gaps in the clockable ordinals and not in the writable ordinals, those objects and the constants λ and γ are closely linked.

Proposition 4.1.30 ([HL00])**.** *The set of the writable ordinals and that of the clockable ordinals have the same order-type.*

Proof. The writable ordinals form an initial segment. So the length of this segment corresponds to the supremum of the writable ordinals, λ. So we need to show that the clockable ordinals form a set of order type λ as well. We write OT_{clock} for the order-type of the clockable ordinals.

We show first that $\lambda \leqslant \mathrm{OT}_{clock}$. Suppose that there were only $\alpha < \lambda$ clockable ordinals. In this case, we could consider the following machine: it writes α on some part of its tape (as $\alpha < \lambda$, it is writable) and then it simulates the universal machine $\mathcal{U}$. At each stage of $\mathcal{U}$ at which a machine halts, it decrements α. When α is down to 0, it halts. But α down to 0 means that the machine computed through all clockable ordinal stages. So it is a contradiction for this machine to halt after this point.

Now, suppose that $\lambda < \mathrm{OT}_{clock}$. This means that there is some clockable ordinal α such that β ordinals are clockable below α with $\lambda < \beta < \mathrm{OT}_{clock}$. We write m_α for the machine that clocks α. We consider the following machine that uses the halting time of the machines to write a well-order of order-type α. It computes as follows: it begins with the empty well-order and simulates $\mathcal{U}$. Each time a machine i halts (if there are more than one, it only considers the least indexed one), it add them to the well-order: that is, it writes $j \prec i$ for all machine j that already halted. Under our assumptions, when m_α halts, the order type written by this computation is β. Hence, if we ask the machine we described to halt when m_α does in the simulation of $\mathcal{U}$, it writes $\beta > \lambda$ which is a contradiction. □

Encouraged by such a result, we may be optimistic and think that $\lambda = \gamma$. That is, that the supremum of the clockable ordinals and that of the writable ordinals match. This was proved left open in [HL00] and proved in [Wel00b].

Theorem 4.1.31 ([Wel00b])**.** $\lambda = \gamma$.

Proof. We give a proof for a slightly more general result in Proposition 5.2.1 at the beginning of next chapter. The proof is based on the same technique as used in [Wel00b]. □

We will see in the next section the usefulness for the study of ITTMs of linking those two objects (writable and clockable) of quite different nature. This also motivates Theorem 5.4.14 which generalizes this equality to a wide range of machines and which will be the subject of a good part of the next chapter.

4.1.4 Writable and clockable ordinals

Let us come back to the question that followed Proposition 4.1.22. Is it equivalent for a set to be writable (in the sense of Definition 4.1.21) and to be in L_λ ? We can now answer this question using Theorem 4.1.31.

Proposition 4.1.32 ([Wel00b])**.** *A set a is writable (resp. e.w. and a.w.) by an ITTM if and only if $a \in L_\lambda$ (resp. $a \in L_\zeta$ and $a \in L_\Sigma$.)*

Proof. As seen in Proposition 4.1.22, a being in L_λ means that a is writable.

Conversely, if a is writable, it is written before stage γ as no machine halts after stage γ. But as $\gamma = \lambda$ it means that a code for a is written at some stage $\alpha < \lambda$. Writing x_a for this code, it means that x_a is definable over L_α and that $x_a \in L_{\alpha+1} \subset L_\lambda$. Moreover, this code describes a relation E on ω such that $(\omega, E) \simeq (\mathrm{TC}(\{a\}, \in)$. We write f for the associated isomorphism. Now, by Proposition 4.1.23, λ is admissible. So the fact that $x_a \in L_\lambda$ and that L_λ is admissible allows us to define f by Σ_1 recursion with x_a as parameter. And by definition of f, there is some $i \in \mathrm{dom}\, x_a$ such that $f(i) = a$. And by an induction relying on Σ_1-collection, $f(i) \in L_\lambda$, as wanted.

The two other cases are similar. □

From there, we can prove one of the main theorem involving those constants. It shows how deeply connected ITTMs are with the constructible universe and set theory.

Theorem 4.1.33 (λ-ζ-Σ Theorem, [Wel00a])**.** *$L_\lambda \prec_{\Sigma_1} L_\zeta \prec_{\Sigma_2} L_\Sigma$. That is, L_ζ is a Σ_1-e.e.e. of L_λ and L_Σ is a Σ_2-e.e.e. of L_ζ. Moreover, (λ, ζ, Σ) is the lexicographical least triplet yielding this chain of e.e.e.*

Proof.

$L_\lambda \prec_{\Sigma_1} L_\zeta$: Let $\varphi(y) = \exists x\, \psi(x, y)$ with $y \in L_\lambda$ and ψ Δ_0. Suppose that $L_\zeta \models \varphi(y)$. We design the following infinite machine $\mathcal{M}$. First it writes y on some part of its working tape (which it can do by Proposition 4.1.32, as $y \in L_\lambda$.) Then using $\mathcal{U}$ it enumerates through the a.w. ordinals and for each α it encounters, it computes a code for L_α and checks whether there is some $x \in L_\alpha$ satisfying $\psi(x, y)$; this is doable as ψ is Δ_0 and y is written on some part of the working tape. If it doesn't find it, $\mathcal{M}$ goes

on with the iteration of the a.w. ordinals in $\mathcal{U}$. However, at some point $\alpha \geqslant \zeta$ and under the assumptions that $L_\zeta \models \varphi(y)$, there is some $x \in L_\alpha$ such that $\psi(x, y)$. When it is found by $\mathcal{M}$, the latter writes it on its output tapes and halts. Hence, such a x is writable. By Proposition 4.1.32, this time invoked in the other direction, this x (as a set) is in L_λ and $L_\lambda \models \exists x\, \psi(x, y)$, as wanted.

Conversely, if $L_\lambda \models \varphi(y)$ then so does L_ζ by persistency of Σ_1 formulas.

$L_\zeta \prec_{\Sigma_2} L_\Sigma$: Let $\varphi(z) = \exists x \forall y\, \psi(x, y, z)$ be a Σ_2 formula with the constant z in L_ζ such that $L_\Sigma \models \varphi(z)$. We consider the following machine $\mathcal{M}$. It simulates the machine m_z that e.w. z and each time the output of this machines changes, it starts with this output and from the beginning the subcomputations $\mathcal{S}$ that we describe at the next paragraph (hence potentially halting and erasing the previous instance of this subcomputation that was run). We point out that at some point in the simulation of m_z, z is e.w. and not modifier thereafter. Hence, at some point this subcomputation is run with the correct z and not interrupted before it decides to halt, if it does. So w.l.o.g. we can describe this subcomputation with the correct z.

This subcomputation $\mathcal{S}$ does the following: As previously, it simulates $\mathcal{U}$ in order to look for candidates x such that $L_\Sigma \models \forall y\, \psi(x, y, z)$. Observe however that $L_\Sigma \models \forall y\, \psi(x, y, z)$ is only semi-decidable. Indeed, $\mathcal{S}$ can launch another fresh simulation of $\mathcal{U}$ in order to iterate through the y's in L_Σ and for each of them check whether $\psi(x, y, z)$. If it fails for one y, the machine can decide that $L_\Sigma \not\models \forall y\, \psi(x, y, z)$. In this case, this means that this particular x wasn't the right candidate and $\mathcal{S}$ goes on with its enumeration of the $x \in L_\Sigma$. However, if it is true that $L_\Sigma \models \forall y\, \psi(x, y, z)$, the enumeration on the y never finds a y for which $\psi(x, y, z)$ fails. But it also never knows when it went saw all possible a.w. y's. Hence when $\mathcal{M}$ find the right x (and there is one under the assumption that $L_\Sigma \models \varphi(z)$), the subcomputation $\mathcal{S}$ never halts. So, having the main machine $\mathcal{M}$ copy the candidate x used by $\mathcal{S}$ before it begins makes it e.w. a x such that $L_\Sigma \models \forall y\, \psi(x, y, z)$. In virtue of Proposition 4.1.32, this set x is in L_ζ and $L_\zeta \models \exists x \forall y\, \psi(x, y, z)$.

Conversely, if $L_\zeta \models \varphi(z)$, then there exists $x \in L_\zeta$ such that $\forall y \in L_\zeta\, \psi(x, y, z)$. So if there were some $y \in L_\Sigma - L_\zeta$ such that $\neg\psi(x, y, z)$, a machine we would be able to e.w. it with the same technique, which would contradict the fact that its not part of L_ζ. Hence $L_\Sigma \models \forall y\, \psi(x, y, z)$.

As for the second assertion regarding the triplet (λ, ζ, Σ), it is more conveniently proved using Proposition 5.2.1 which generalizes Theorem 4.1.31. This is done in Proposition 5.2.2 □

This theorem remarkably describes the power and the limit of ITTMs. It can also naturally act as a criterion to evaluate attempts at generalizing ITTMs. Given a generalization of ITTMs, we can ask ourselves: does a variation of this theorem exists? And does

this variations shows that this generalization provides a more powerful or more interesting model? Reaching a variation of this theorem for a new model of infinite machine will be the main aim of Chapter 6.

4.1.4.1 The theory machine

Friedman and Welch introduced the idea of a "theory machine" in [FW07]. It is an ITTM which successively writes down the Σ_2 theories of the levels J_α of the Jensen hierarchy (which is very close to the L hiearchy) for α below Σ_0; Σ_0 being the least ordinal such that L_{Σ_0} is the Σ_2-elementary extension of some previous level, which we in turn write ζ_0. At this point we do know that Σ_0 and ζ_0 actually are Σ and ζ, but it is worth noting that the results regarding the theory machine can be established prior to high-level results like the $\lambda - \zeta - \Sigma$ Theorem. More precisely, they show the following.

Proposition 4.1.34 ([FW07])**.** *There exists an ITTM which, when computing with the empty inputs, does not halt and such that, for any $\alpha < \Sigma_0$, the Σ_2 theory of $(J_\alpha, \in)$ is written on its output tape at stage $\omega^2(\alpha + 1)$. At stage $\Sigma_0 + \omega^2$, the theory on the output of the machine is the Σ_2 theory of J_{ζ_0} and the machine starts looping.*

To define such a machine, the harder part to overcome is, as often with ITTMs, the limit part. If we want the machine to conduct such a sharp and structured computation, it clearly should not, while computing up to some limit stage, lose too much information about what it has previously done. But if, when computing up to some limit stage λ, it has successively written all the Σ_2 theories of the $(J_\alpha, \in)$ for $\alpha < \lambda$ on some tape, the content of this tape at limit stage λ, that is the content produced by the limit rule, more than likely won't be the theory of $(J_\lambda, \in)$. This comes from the simple fact that a Σ_2 sentence may be cofinally true in the models $(J_\alpha, \in)$ for $\alpha < \lambda$, while not true in $(J_\lambda, \in)$.

The way to overcome this issue is the following. Friedman and Welch write T_α the theory of Σ_2 sentences defined as

$$T_\alpha = \{\varphi \mid \exists \beta < \alpha \, \forall \delta < \alpha \, (\beta < \delta \implies J_\delta \models \varphi\}$$

which is the theory of Σ_2 sentences true in J_δ for big enough $\delta < \alpha$.

From there, two key observations are as follows. First at a limit stage λ, the limit of the codes of the Σ_2 theory of the J_α's (note that Friedman and Welch chose to work with the lim inf limit rule) below λ actually produce a code for T_λ. This is not a coincidence since we clearly recognize the definition of the lim inf rule in the definition of T_α. Second, in more set-theoretic involved way, for any ordinal α, the Σ_2 theory of J_α is r.e. in T_α. Hence, at a limit stage and in less than ω steps, the theory of J_α can actually be retrieved with what was produced by the limit rule, and the computation can go on. This outlines how the complexity of limit stages can be overcome.. We will deal again with the idea of

the theory machine in the more advanced setting of hypermachines, also introduced by Friedman and Welch, which we briefly present in Section 4.2.2.

The existence of such a machine immediately tells us a few things which we had to prove "by hand". Among those are the fact that machines can start looping as late as Σ_0 (which, by Theorem 4.1.33, we know to be equal to Σ), or that any real in L_{Σ_0} is a.w.

But more than this broad pictures on the behavior of ITTMs, its sharpness allows us to get much finer details, notably regarding the time at which writable reals can actually be written. This result from Welch is a prime example.

Proposition 4.1.35 ([Wel09])**.** *If $\alpha > \omega$ is clockable, then it is writable in at most α steps.*

The machine theory is here a more than valuable tool, since trying to come up with results akin to this "quick writing" and without the machine theory can be especially difficult. When trying to show similar results from scratch, it promptly appears that the behavior of machines like the universal machine (e.g. the time at which it produces given ordinals) is, before obtaining said results, far too erratic or at least unknown. And this results then allows for a fine study of gaps.

Proposition 4.1.36 ([Wel09])**.** *If α starts a gap in the clockable ordinals, then α is admissible.*

Proposition 4.1.37 ([BDL23])**.** *If α ends a gap, then it is writable in a write-only way (i.e. the cells of the output tape are not written on more than once) in α steps.*

Hence, it is clear that, after defining higher-order model of ITTMs as in Chapter 6, trying to generalize the idea of the theory machine to those new models ought to be a fruitful endeavor. This has unfortunately not been attempted in this thesis by lack of time. And that said, we see in the context of the hypermachines, presented in Section 4.2.2, how things get rather complicated when dealing with Σ_3 formulas or higher. One of the reasons is that we now can't rely on very convenient results, like the existence of a uniform Σ_1 Skolem function for the levels of the Gödel's hierarchy.

4.2 Rule-wise and machine-wise generalizations

Here, we introduce a brief panorama of the state-of-the-art of models of infinite machines – from other kind of infinite machines to generalizations and restrictions of the ITTM model of Hamkins and Lewis.

First, we can distinguish two types of nature of generalizations of ITTMs. Observe indeed that in the description of the ITTM, two fundamental aspects can be distinguished : its structural or machinal aspect (in particular it has the same structure as a classical finite TM) and its computational or rule aspect (in this case, the lim sup rule). The structural

aspect of a machine defines its components and is analogous to a spatial description while its computational aspect defines the behavior of those components throughout the computation and is analogous to a temporal description.

Hence, this points toward two directions for generalizations of ITTMs: those can either be *machine-wise*, that is structural, or *rule-wise*, that is, in the above sense, computational. In this work, we are mostly interested with rule-wise generalizations as, given their role in the temporal aspect of the computation, they entertain a closer relationship with ordinals and the constructible hierarchies in set theory.

We start by giving a quick overview of some other infinite machines which are based on the same idea of a lim sup rule as well of machine-wise generalizations and restrictions which have been studied. A complete presentation of those can be found in [Car19]. After this, we present the rule-wise generalization introduced by Friedman and Welch in [FW11].

4.2.1 Machine-wise generalizations

We follow the presentation order suggested by Carl in [Car19], from the less powerful model to the more powerful one.

4.2.1.1 Infinite time register machines (ITRMs and α-ITRMs)

Definition 4.2.1 ([Koe09])**.** For an limit ordinal α, an *α-ITRM* has the structure of a classical finite register machine (see [Car19, p. 19] for a description) and is defined by the following limit rule, writing l^α the active program line at limit stage α and r_i^α the content of the i^{th} register at limit stage.

- $l^\alpha = \liminf_{\beta<\alpha}(l^\beta)$
- $r_i^\alpha = \begin{cases} \liminf_{\beta<\alpha}(r_i^\beta) \text{ if } \liminf_{\beta<\alpha}(r_i^\beta) < \alpha \\ 0 \text{ else} \end{cases}$

Moreover, ω-ITRMs are simply called *ITRM*.

In this definition we see how the main difference between α-iTRMs and ITTMs comes from the structure of the machines rather the from the rule. Obviously, a different structure forces a different description of the limit rule, but the underlying idea is here still that of a lim inf.

Now, one main aspect of the model of α-ITRM is the fact that its has finitely many registers and whose value are bounded; namely by α. Obviously, the choice of α has direct consequences on the power of an α-ITRM: it can for example easily count for α steps using only one register. So we now consider ITRMs (that is ω-ITRMs) that can be

compared with ITTMs and on an equal footing with them. The first remarkable result is the following:

Proposition 4.2.2 ([KM08])**.** *There is no universal ITRM. That is, there is no ITRM that simulate in parallel all ITRMs.*

Sketch of the proof. This comes from the fact that the halting problem for n-registers ITRM is decidable by an ITRM. The idea behind this fact is that with finitely many registers the configurations of the machine can be encoded as integers. Hence, it is possible to consider a machine with more register that n register that simulate machines with n integers while keeping track of its successive configurations, and doing so until it sees it halt or it detects a looping pattern in the succession of its configurations (as with ITTMs, an ITRM loops if and only if at some point it exhibits a looping pattern.) □

Still, as for ITTMs a lower bound for their arithmetical power can be established.

Proposition 4.2.3 ([KM08])**.** WO *is ITRM-decidable.*

Sketch of the proof. The idea is the same as for the ITTM case: the machine ensures that the real given in input describes a partial order and then it inductively decides whether its a well-order by deciding first whether its empty or whether there is a least element and then whether the partial order obtained by removing this least element is a well-order. □

Hence, from an arithmetical point of view, we may be tempted to see ITRMs and ITTMs as equally powerful. However, from a set-theoretic point of view, the former model is remarkably less powerful.

Proposition 4.2.4 ([Koe09])**.** *The halting time of ITRMs are bounded by* $\omega_\omega^{\mathrm{CK}}$.

Sketch of the proof. This is shown by studying the looping condition of ITRM and showing that such a looping pattern must appear before stage $\omega_\omega^{\mathrm{CK}}$. □

On the other hand, it is relatively easy to see that $\omega_\omega^{\mathrm{CK}}$ is strictly smaller than γ: this comes from the fact that $\lambda = \gamma$ and that by Proposition 4.1.23, λ is an admissible ordinal limit of admissible ordinals (and even limit of limit of limit...) Still, akin the the equality $\lambda = \gamma$, the following results shows that ITRM are well-behaved when it comes to what corresponds to their writing and clocking abilities.

Proposition 4.2.5 ([Koe09])**.** *A real x is ITRM-computable if and only if $x \in L_{\omega_\omega^{\mathrm{CK}}}$.*

Sketch of proof. The left-to-right direction come from the previous proposition. The other direction is proved by showing how the set of ITRMs computing well-orders recursive in ω_i^{CK} is ITRM-decidable and how this implies that a code for $\omega_{i+1}^{\mathrm{CK}}$ is ITRM-computable and then, in turn, how it implies that the set of ITRMs which compute well-orders recursive in $\omega_{i+1}^{\mathrm{CK}}$ is ITRM-decidable and so on for $i \in \omega$. □

As said, such relations may act as a criterion to judge of the practicality(in its usage in set theory) of an infinite machine. Alternatively, it can be seen as a criterion of how natural or well-behaved it is (the former if we believe in some sort of mathematical natural arrangement). Consequently the following result, regarding α-ITRMs and not as satisfying as the previous one, may be the sign that something deeper is yet to be understood with this idea of α-powered computation models.

Theorem 4.2.6 ([Car19])**.** *Under technical conditions on α described in [Car19, p. 115], we have the following: if β has an α-ITRM computable code and $x \in \mathcal{P}(\alpha) \cap L_\beta$ then x is α-ITRM computable.*

4.2.1.2 α-Turing machines (α-ITTMs)

Even if both were introduced at the same time, the model of α-ITRM can be seen as generalization of that of ITRM. Similarly, we can define α-ITTMs.

Definition 4.2.7 ([KS09])**.** α-ITTMs are defined as ITTMs with the onmy difference that their tapes are of length α. That is, each tape with its content induces a function $C : \alpha \to 2$ that maps some ordinal $\iota \in \alpha$ to the content of the ι^{th} cell of the tape. Hence, the heads also move on those tapes of length α and at limit stage ν, writing h^ν for the position of some head at this limit stage, we have:

$$h^\nu = \begin{cases} \liminf\limits_{\nu'<\nu} h^{\nu'} \text{ if } \liminf\limits_{\nu'<\nu} h^{\nu'} < 0 \\ \text{else } 0 \end{cases}$$

Moreover, the following convention is added: if one of the heads moves to the left while on a cell whose index is a limit ordinal, it goes to cell 0.

First, α-ITTMs are an extension of α-IRMs in the sense that the former can simulate the latter (see [Car19, Lemma 3.5.38]). Second, while this definition is very close to that of ITTMs, a difficulty arises. Namely, the fact that the tape has length α makes it difficult for the head of some tape to "know where it is". Indeed, if at some point of the computation of an α-ITTM, the head of the working tape is on the cell β, the machine has no practical mean to apprehend this fact or to behave w.r.t. this ordinal β. Or, from another point of view, the tape may be too big for the machine to purposely move to some β^{th} cells – whereas in the ITTM model it can easily and purposely move to some i^{th} cell .

For a concrete example, suppose that α is multiplicatively closed and that a well-order on α of order-type ν is written on the working tape. As in the proof of 4.1.19, we would like to say that for any $\nu' < \nu$ we can "cut" this order type below some $\beta \in \alpha$ in order to obtain a well-order of order type ν'. In order to do this, the machine needs to identify all $\beta' \in \alpha$ such that $\beta \prec_\nu \beta'$ and remove them from the well-order. That is it need to look at

cells of offset $\langle \beta, \beta' \rangle$ (where $\langle \cdot, \cdot \rangle$ is some pairing function from $\alpha \times \alpha \to \alpha$) that reads 1. But as said, it may not be able to do so without being given, in one way or another, this β. And for such an ubiquitous proposition like Proposition 4.1.19 to be missing would make a big gap in the theory of α-ITTMs. So, we need to allow finitely many parameters $p_i \in \alpha$ to be used in the computation of α-ITTMs. With those (and only with those, see [CRS20]) we can prove the following two fundamental results.

Proposition 4.2.8 ([Car19])**.** *Let $\alpha < \beta$. Then β-ITTMs can simulate α-ITTMs.*

Sketch of proof. We write $\alpha = \alpha' + k$ where α' is limit. Suppose first that $k \neq 1$. Given α as parameter in a β-ITTM computation, it is easy to write a 1 in the cell α. Then, observe that any α-ITTM can be modified to never use cells that are multiple of k (i.e. cells of offset $\nu + k \cdot i$ for any limit ν and finite $i > 0$.) Hence, from any α-ITTM, this let us design a β-ITTM that can detect that it reached cell α (and does not go past it) as it is the only cell of offset multiple of k whose value is 1.

If $k = 1$, we write a 1 in cell $\alpha + 1 = \alpha' + 2$ and then we avoid using cells whose offset is even and simply tell the machine to look, after each step, one cell to the right to see whether it actually reached cell α. □

Proposition 4.2.9 ([Car19])**.** *Let α be multiplicatively closed. Then α-writable ordinals form an initial segment of On.*

Sketch of proof. An ordinal ν is α-writable if an α-ITTM writes the code for a well-order on α of order type ν. Hence, any $\nu' < \nu$ is the order type of this well-order truncated below some $\beta \in \alpha$. With the right β given as parameter, it is easy to truncate the well-order below this β

□

This new feature however, the fact that we authorize parameters in order to prove such basic and excepted results, can be philosophically motivated. It echoes with the idea of α-recursion in which any $x \in L_\alpha$ is deemed L_α-finite: adding finitely many parameters in α to an α-machine is the natural extension of being able to hardcode finitely many natural numbers, that is ordinals in ω, in the code of an ITTM. And further, α-machines can only be run in a natural way in some set in which α is included, typically L_β for $\alpha \leqslant \beta$. From this point of view, admitting parameters in α for an α machine run in such a L_β is as natural as allowing parameters in α or L_α for a formula which is to be evaluated in L_β.

And despite this difficulty and with this feature–and on the contrary to what happened in the case of ITRMs–there exists (for α multiplicatively closed) a universal α-ITTM.

Proposition 4.2.10 ([Car19])**.** *For α a multiplicatively closed ordinal, there exists a universal α-ITTM that simulates simultaneously every computation of the form $m(\nu)$ for m an α-ITTM and $\nu \in \alpha$ a parameter.*

Sketch of proof. The idea is the same as that used for the universal ITTM. It is slightly more involved as with all parameters it simulates way more machines; but also, it now has way more room to do so. See [Car19, Theorem 3.5.15] for a thorough description. □

With such a universal machine, it is relatively easy to reproduce some of the proofs done in the ITTM setting. This yield the following result which shows that the naturally extended constants, $\lambda(\alpha)$, $\zeta(\alpha)$ and $\Sigma(\alpha)$ stay relevant in the context of α-ITTM.

Proposition 4.2.11 ([Car19])**.** $\lambda(\alpha) < \zeta(\alpha) < \Sigma(\alpha)$.

Proof. Same proof as for Proposition 4.1.20. □

Then again, we can prove that the there is a deep connection between the concept of clockable and that of writable ordinal, even for α-ITTM.

Theorem 4.2.12 ([Car19])**.** $\lambda(\alpha) = \gamma(\alpha)$.

Proof. This can be established with the same technique as done by Welch in [Wel00b]. This technique is demonstrated in the proof of Proposition 5.2.1 which generalizes this type of result. □

Corollary 4.2.13 ([Car19])**.** *A code for a set $x \subset \alpha$ is α-writable (resp. α-e.w. and α-a.w.) if and only $x \in L_{\lambda(\alpha)}$ (resp. $x \in L_{\zeta(\alpha)}$ and $x \in L_{\Sigma(\alpha)}$).*

Proof. This can be established as a corollary of the previous theorem as in the proof of Proposition 4.1.32. Alternatively, this can be proved directly; in which case the previous theorem becomes the corollary. With what we have seen in the previous chapter and with the proximity between ITTMs and α-ITTMs, it is relatively clear that those two results are equivalent. See [Car19, Section 4.6.3] for such a presentation. □

And as stated above, at the end of Section 3.3.2, α-ITTMs are the, to this point missing, computational counterpart of α-recursion. By considering α-machines whose computation time is bounded by α, Koepke established the following theorem.

Theorem 4.2.14 ([Koe05])**.** *Let α be closed under ordinal exponentiation. Then $x \subset \alpha$ is α-recursive if and only if x is α-ITTM computable in less than α steps.*

Sketch of the proof. As already seen, $x \subset \alpha$ is α-recursive if and only if it is $\Delta_1(L_\alpha)$ (i.e. Δ_1-definable over L_α with parameters in L_α).

Then, one direction of the equivalence expresses the fact that if an α-ITTM computes some $x \subset \alpha$, then this x is $\Delta_1(L_\alpha)$. And indeed, it is $\Sigma_1(L_\alpha)$ as x is computed by some machine if and only if there exists a halting computation of this machine on less than α steps such that x is on the output when the machine halts. Also, it is $\Pi_1(L_\alpha)$ as x is computed by this machine if and only any computation of this machine (observe that with fixed inputs, a machine only gives rise to one computation) halts and outputs x.

As for the other direction, the idea is the following: a Σ_1 predicate is true in L_α if and only it is true in some L_β for $\beta < \alpha$. So an α-ITTM can go through all $\beta < \alpha$, compute a code for L_β and test whether the predicate is true in L_β. This let α-ITTM semi-decide Σ_1 predicates in L_α. So if a predicate is Δ_1, it is decidable by an α-ITTM, and consequently any α-recursive x is computable. □

4.2.1.3 Ordinal register machine and ordinal Turing machine (ORM and OTM)

Further again, Koepke defined *ordinal register machines* (ORM) and *ordinal Turing machine* (OTM). Those are obtained by simply providing registers of size On for the ORMs and tapes of length On for the OTMs.

For the ORMs, if we apply the same reasonning as previously, we may want to allow parameters. But then, it means parameters in On; which may in turn seem a bit excessive. In this setting, as could be expected, every constructible set of ordinal X is decidable with some $\alpha \in On$ as parameter (see [KS06]). Without parameters, it can decide any set of ordinal X such that $X \in L_\sigma$ where σ is the least stable ordinal (i.e. least ordinal such that $L_\sigma \prec L$). This come from the fact that cofinally many ordinals below σ are the least ordinal satisfying some Σ_1 property (see [SS12]). Hence those ordinal can be recognized without being given as parameters. Interestingly, it can be shown that there is exists a universal ORM (see [KS08]).

For OTMs, as previously, they are more powerful as they can simulate ORMs (see [Car19, Lemma 3.5.45]). From this, we obtain the analog results for OTMs: a set $X \subset On$ is OTM-writable (with parameters) if and only it is constructible ([KS09]) and it is OTM writable without parameters if and only if it is in L_σ ([SS12]).

Hence, α-ITTM and OTM give a computational and practical tool for the study of admissible sets and the constructibe universe. As an example, we can cite the proof of the Sacks-Simpson theorem (for an admissible ordinal α, there exists two α-r.e. sets that are incomparable w.r.t. weakly α-recursivity) using α-ITTMs presented in [Koe05].

4.2.2 Rule-wise generalization: Friedman and Welch's hypermachine

In [FW11], Friedman and Welch introduced a rule-wise generalization of the classical ITTM. The idea is the following: at limit stages, the value of a cell is still determined by a lim sup or a lim inf of the previous values of the cell (in particular, they chose the lim inf here), with the difference that the lim inf is computed only from a specific subset, depending of the history of the machine, of the previous values of this cell.

Definition 4.2.15 ([FW11])**.** A *hypermachine* has the structure of a 4-tape Turing machine. We call those tapes input tape, working tape, output tape and rule tape. It computes like an ITTM with only a difference as to how the limit value of the cells are computed.

At any limit stage α, the rule tape determines (in a way that we will describe just after) a cofinal subset $E^1_\alpha \subseteq \alpha$ and, writing $C_i(\beta)$ for the value of the cell i at some stage β, $C_i(\alpha)$ is defined as follow.

$$C_i(\alpha) = 1 \longleftrightarrow \exists \beta < \alpha \, \forall \delta \in [\beta, \alpha[\; (\delta \in E^1_\alpha \implies C_i(\delta) = 1)$$

That is, in the computation of the machine up to some stage α, there is some stage β such that at every ulterior stage $\delta \in E^1_\alpha$, $C_i(\delta) = 1$. In a way, we could say that the cell i stabilized before stage α w.r.t. the subset E^1_α.

It is clear that those machines are rule-wise generalizations of the ITTMs: the fact that a tape is added is more for convenience and clarity of the exposition, as it would be doable to merge the working tape with the rule tape. However, Friedman and Welch show that this model is strictly more powerful than that of ITTM. This shows that the change of the limit rule couldn't be simulated by classical ITTM and so that the essential aspect of this generalization is the change of the rule itself.

We now describe how E^1_α is defined. The stages chosen for the lim inf, that is the stages in E^1_α, are called by the authors "1-correct in α". So the limit operator can be written like this, for a limit α:

$$C_i(\alpha) = 1 \longleftrightarrow \exists \beta < \alpha \, \forall \delta \in [\beta, \alpha[\; (\delta \text{ is 1-correct in } \alpha \implies C_i(\delta) = 1)$$

To define the 1-correct ordinals in α, they first define the 1-stable ordinals in α. In what follows, we write 1^n for the word of length n made only of the letter 1 and $1x$ for the (possibly ordinal-indexed) word made of the word 1 concatenated to the left of the word x.

An ordinal $\beta \in \alpha$ is 1-stable in α when it is a multiple of ω^ω and when the following holds: if any pattern of the form $0^n 1x$, for $x \in {}^\omega 2$ and $n \in \omega$, appeared as the content of the rule tape before stage α and if $1x$ appeared as the content of the rule tape before stage β, then the pattern $0^n 1x$ also appeared before β. That is:

$$\forall x \in {}^\omega 2 \, \forall n \, \forall \delta < \alpha \, \forall \delta' < \beta \, (C(\delta) = 0^n 1x \wedge C(\delta') = 1x \implies \exists \delta'' < \beta \, C(\delta'') = 0^n 1x)$$

The authors write $S^1_\alpha \subseteq \alpha$ for the 1-stable ordinals below α. From there, they define E^1_α, the 1-correct ordinals below α, as follows.

$$\begin{cases} E^1_\alpha = \{\alpha' < \alpha \mid S^1_\alpha \cap \alpha' = S^1_{\alpha'}\} \text{ if } \alpha \text{ is a limit of multiple of } \omega^\omega \\ E^1_\alpha = \alpha \text{ else} \end{cases}$$

So, an ordinal α' is 1-correct in α if it is correct (w.r.t. α) about the ordinals $\beta < \alpha'$ it thinks are 1-stable.

This definition is tailored to allow the construction of a *theory machine*. That is a

machine that will iteratively write on its output a coden as well as the Σ_3-theory, of the successive levels of the Jensen's hierarchy J_α. For our purpose, it is enough to know that if $\alpha = \omega\alpha$ then $J_\alpha = L_\alpha$. As defined in [FW11], we write $(\zeta(3), \Sigma(3))$ for the least couple such that $\zeta(3) < \Sigma(3)$ and

$$L_{\zeta(3)} \prec_{\Sigma_3} L_{\Sigma(3)}$$

Theorem 4.2.16 ([FW11])**.** *There exists a theory hypermachine that iteratively writes codes for the J_α as well as, in parallel, the complete theories of the J_α for all α below $\Sigma(3)$. Moreover, at stage $\Sigma(3)$ it reproduces stage $\zeta(3)$ and starts looping.*

Sketch of proof. As done in [FW11], we write G (for "good") for the set of ordinals that are multiple of ω^ω and G^* for ordinals that are limit of ordinals in G.

Then, observe first that if the rule tape is not used, that is if nothing is written in the rule tape, then for any α, the 1-stable in α are simply the elements of $\alpha \cap G$. So in this case, for any $\alpha' < \alpha$, we have:

$$S^1_\alpha \cap \alpha' = \alpha \cap G \cap \alpha' = \alpha' \cap G = S^1_{\alpha'}$$

This means that if the rule tape is not used, the hypermachine behaves like a classical lim inf ITTM. In other words, ITTMs can be seen as particular hypermachines.

This allows for the following observations, writing T^n_α for the Σ_n-theory of J_α, T^ω_α for its complete theory and $l(\alpha)$ for a code of J_α.

- Given $l(\alpha)$ and T^ω_α an ITTM (and so a hypermachine) can compute $l(\alpha+1)$, a code for $J_{\alpha+1}$, by computing from the input the next stage of the Jensen hierarchy.
- Given $l(\alpha+1)$, a machine can compute $T^\omega_{\alpha+1}$ by evaluating every possible sentence in $J_{\alpha+1}$, using $l(\alpha+1)$.

With this, we see how successor stages of the theory machine can be relatively easily dealt with.

It gets tricky at limit stages as nothing tells us that what the limit rule will produce with previously written codes and theories won't simply be gibberish. To deal with limit cases, we can define Σ_n stable ordinals:

Definition 4.2.17. For a limit ordinal λ, an ordinal α is *Σ_n-stable in λ* when:

$$\alpha < \lambda \wedge J_\alpha \prec_{\Sigma_n} J_\lambda$$

We write $\widehat{S}^n_\lambda$ for the Σ_n-stable ordinals in λ.

Then, Friedmann and Welch distinguish two limit cases in the computation of the theory machine, writing $\bar{\lambda} = \sup(\widehat{S}^1_\lambda)$.

$\bar{\lambda} < \lambda$: Then, considering the $\bar{\beta}$'s as β grows below λ, those $\bar{\beta}$'s eventually reach $\bar{\lambda}$ (using persistency of Σ_1 formulas) and the sequence does not grow further. That is, $\bar{\lambda}$ is e.w. below λ. Also, observe that, by persistency of Σ_1 formulas, $T^1_\lambda(\bar{\lambda})$ (the theory of J_λ with $\bar{\lambda}$ as parameter) is the union of the $T^1_\beta(\bar{\lambda})$ for $\bar{\lambda} < \beta < \lambda$. Hence, $T^1_\lambda(\bar{\lambda})$ can be produced by the usual lim sup rule from the previous $T^1_\beta(\bar{\lambda})$.

Then, by a set-theoretic argument, it is possible to show that for each n there is a uniform and parameter-free Σ_n Skolem function. In particular and in our case, using $T^1_\lambda(\bar{\lambda})$, this allows to compute the Σ_1-Skolem hull of $\left\{\bar{\lambda}\right\}$ inside J_λ. This is a Σ_1-elementary submodel of J_λ (this can be seen using the Tarski-Vaught criterion) in which $\bar{\lambda}$ belongs. By the definition of $\bar{\lambda}$, this is J_λ itself. Hence, this yield a uniform (in λ and $\bar{\lambda}$) map from $\bar{\lambda}$ onto J_λ which let the machine compute a code for J_λ using that for $\bar{\lambda}$.

$\bar{\lambda} = \lambda$: Observe that this implies $\lambda \in G^*$ and that this limit stage is ruled by the lim inf on E^1_λ. We define:

$$\widehat{E}^1_\lambda = \left\{\alpha < \lambda \mid \widehat{S}^1_\alpha = \widehat{S}^1_\lambda \cap \alpha\right\}$$

The similarity in the notations is naturally intentional as with the right usage of the rule tape, we can show that for $\lambda \in G^*$, $\widehat{E}^1_\lambda \cap G = E^1_\lambda \cap G$. Briefly the idea of how to use the rule tape is the following: it can be used to write information about reals. At some stage β, writing $1x$ on the rule tape means that the real $x \in J_\beta$ has been considered. Then, writing 0^n1x means that $J_\beta \models \varphi^1_n(x)$ for $(\varphi^1_n)_n$ an enumeration of the Σ_1 formulas with one parameter. Hence, by construction, 1-stability in λ translates into a statement about the Σ_1-theory of J_β with reals as parameters. More precisely, looking at the rule, we see that it requires J_β to have the same Σ_1-theory as J_λ with reals as parameters. At the levels we are working at, everything is seen to be countable and this is equivalent to a statement about the Σ_1-theories without any parameter and this is enough to show that $\beta \in S^1_\lambda$ if and only if $J_\beta \prec_{\Sigma_1} J_\lambda$ if and only if $\beta \in \widehat{S}^1_\lambda$.

This in mind, there is in the machine a flag F that keeps track of when the value of $\bar{\beta}$ changes, for β's below λ. Using it, it can detect that $\bar{\lambda} = \lambda$. Then, at those stages, using the equality $\widehat{E}^1_\lambda \cap G = E^1_\lambda \cap G$ and the information carried by this particular lim inf, Friedman and Welch show that for $\lambda < \Sigma(3)$ the machine can compute T^3_λ. They show that for a Σ_3 formula ϕ, $J_\lambda \models \varphi$ is equivalent to say that all J_γ in $\widehat{E}^1_\lambda$ satisfy some Σ_1 formula; which is exactly the information carried on by the lim inf on $E^1_\lambda = \widehat{E}^1_\lambda$ of the T^1_γ (that is for $\gamma \in \widehat{E}^1_\lambda$).

Hence, T^3 is produced at those limit stages $\lambda < \Sigma(3)$ for which $\bar{\lambda} = \lambda$ and with the same Skolem hull argument, this enables the machine to compute a code for J_α and to continue further with the computation.

□

The existence and the behavior of this theory machine shows that all sets in $L_{\Sigma(3)}$ are a.w. by a hypermachine. To see this, for any x in $L_{\Sigma(3)}$, we can simply consider the theory machine slightly modified so that it outputs, each time it writes the code of a new J_α, the truncation of the code of J_α below some i. With a well chosen i, for the right J_β, it will produce a code for x. Further, while the authors do not give a clear looping condition, it seems plausible that, more generally, all hypermachine start looping at stage $\Sigma(3)$, repeating the stage $\zeta(3)$. This would conversely show that all a.w. sets are in $L_{\Sigma(3)}$ and similarly that being hypermachine-e.w. and being in $L_{\zeta(3)}$ is equivalent for a set. This naturally justifies the use of the constants $\zeta(3)$ and $\Sigma(3)$, which were originally simply defined as the least couple for which $L_{\zeta(3)} \prec_{\Sigma_3} L_{\Sigma(3)}$ holds. This proves indeed the following theorem for the hypermachines, which was naturally the aim of this generalization:

Theorem 4.2.18. *$\Sigma(3)$ is the supremum of the hypermachine-a.w. ordinals and $\zeta(3)$ is the supremum of the hypermachine-e.w. ordinals and $L_{\zeta(3)} \prec_{\Sigma_3} L_{\Sigma(3)}$.*

Observe however that to reach the diverse assertions of the previous paragraph, we did not use the universal machine. And this is for a good reason: it is not clear whether there exists a universal hypermachine. We will come back better equipped for this question in Remark 5.3.34 but for the moment it is enough to observe that it would be difficult to simulate a rule tape in a straightforward way: as the limit rule looks at the whole rule tape, it does not seem possible to split it into two to "share" between the simulating machine and the simulated machine. And the absence of a universal machine may both hinder their study, as most of high-level proof techniques for ITTMs use universal machines, as well as their practicality. For this last point, we should add that, for the same reasons, the possibility of simulating a single other hypermachine in an hypermachine is not granted as well.

Chapter 5

Toward higher-order machines: simulational Γ-machines

Wo aber Gefahr ist, wächst das Rettende auch.
But where the danger is, also grows what saves.
– Hölderlin, *Patmos*

In this chapter we present the more general theory of simulational Γ-machines. Γ-machines provide a wide generalization of the previous model of ITTM: namely a model that accounts for infinite Turing machines whose limit behavior is ruled by any rule. In turn, simulational Γ-machines form a restriction of this generalization to model of machines in which a universal machine can be defined in a straightforward way. More precisely, simulational Γ-machines are defined as Γ-machines whose limit rule satisfy a set of 4 simple constraints–which we show is enough for a model of Γ-machine to admit a universal machine. To this set of constraint we add a safeguard constraint and a definability constraint which ensures tight links with the constructible universe. The three main results of this chapter are the following. First, Theorem 5.4.1 surprisingly shows there are only two 2-symbol limit rules, namely the lim sup and the lim inf rule, that satisfy the 4 simulational constraints as well as the safeguard one. This shows how, more generally, n-symbol machines must be considered. Second, the most central result, for models of Γ-machine satisfying the 6 constraints we mentioned, Theorem 5.4.14 establishes the equality between Σ_Γ, the supremum of the ordinals that are accidentally writable with a given model of Γ-machine, and T_Γ the supremum of the ordinals that are accidentally clockable with the same model. This result is the generalization to a wide class of models of machine of Proposition 5.2.1. Thirdly, Theorem 5.5.1 shows through a counter-example that the safeguard condition cannot be omitted in previous theorem. Moreover, we discuss of the hypermachines developed by Friedman and Welch in the light of those results in Remark 5.3.34.

5.1 Results and organization of the chapter

As mentionned, ITTMs simply are finite Turing machines to which we add a limit rule that ensures that the machine snapshots are unambiguously defined at limit stages. Hence an ITTM has the same structure as a classical finite Turing machine and what actually does the (transfinite) work is the limit rule. In other word, with this point of view, the structure of the classical finite Turing machine only misses a limit rule to become a model of infinite machine. Once we realize this, the model of ITTM introduced in [HL00] can immediately be seen as a "plug and play" structure in which it is enough to plug a limit rule in in order to obtain a model of infinite computation ready for use or study. This is the basis of what was refereed to as "rule-wise generalization" in Section 4.2.

Now it is clear that not any limit rule will give rise to as a well-behaved and interesting model as that of ITTMs. And we will see how those feats of the ITTM are rooted in the existence of a universal machine. So, working our way toward the possibility of devising interesting rule-wise generalizations, our first leading question will be: how and why does the universal ITTM work?

To answer this question, we first look at the universal ITTM and the way it is commonly used in order to exhibit sufficient conditions for a universal machine to exist in the model produced by a given rule. This is done in Section 5.2. We begin by proving a slightly new structural equality (which generalizes the equality $\lambda = \gamma$ proved in [Wel00b]) for the usual ITTMs. This proof shows how the universal machine is canonically used in this kind of state-of-the-art proofs developped by Welch in [Wel00b]. Then we try to understand what is implicitly needed in this proof regarding the universal machine and more generally regarding the possibility of simulating other machines. At the end of this discussion, we exhibit five critical characteristics satisfied by the usual lim sup limit rule. The first four of those form a sufficient condition for a given limit rule to yield a model of ITTM in which a universal machine exists.

In Section 5.3, we provide general definitions for the chapter as well as a formal definition of what a rule is. It can indeed be seen an operator that maps ordinal words representing the history to symbols in the alphabet of the machine. With this formal definition we can formalize the five conditions of the previous section.

In Section 5.4, we start by showing in Theorem 5.4.1 that the lim sup and the lim inf operator are, surprisingly, the only two-symbol limit operators satisfying this set of five conditions. This steers the search for higher order machine toward a yet unexplored direction: n-symbol infinite Turing machine. We then show in the main theorem, Theorem 5.4.14, that for any n-symbol Turing machine that satisfies those five condition and definable by a set-theoretic formula we can establish many of the results established for the classical ITTMs. The most important one, whose proof takes a large part of the section, is a strong constant equality relating the clockable ordinals and the writable ordinals (In the case of Γ-machines, those constants are introduced in Section 5.2). A corollary of this

equality is that for those models, the generalization to Γ-machines of the equation $\lambda = \gamma$ also holds.

In Section 5.5 we eventually show in Theorem 5.5.1 that the fifth condition, acting as a safeguard and not needed for an operator to generate a model of machine with a universal machine, is however needed for the main theorem. This shows that this result is sharp with respect to this fifth hypothesis.

5.2 Eventually clocking and accidentally clocking

As already seen, a real can be *writable* if there is a machine that halts (by reaching its distinguished state q_{halt}) and such that when it halts x is written on its output tape. It can also be *eventually writable* (e.w. in what follows) if there is a machine that never stops and that, at some point, has x written on its output tape and never modifies it afterward. It can be *accidentally writable* (a.w. in what follows) if there is a machine in which x is written on any of its tapes and at any stage, even if it is modified or erased just after. It is easy to see that writable implies e.w. implies a.w.

From there, we explained how an ordinal can also be writable (resp. e.w. and a.w.) when a code for it is writable (resp. e.w. and a.w.) This induces three constants: Σ, *the supremum of the a.w. ordinals*, ζ, *the supremum of the e.w. ordinals* and λ, *the supremum of the writable ordinals.*

On the other hand, when it comes to clockable ordinals, only one constant was defined: γ, *the supremum of the clockable ordinals.* But we can actually also devise three ways for an ordinal to be clockable. To this extent, we introduce the two following definitons: α is *eventually clockable* when the output of some converging machine m stabilizes at stage α. And α is *accidentally clockable* when there is a computation in which at stage α some real x appears for the first time in the computation. Then we can write η *the supremum of the eventually clockable ordinals* and T *(capital τ) the supremum of the accidentally clockable ordinals.* And with the latter, we can show the following result, generalizing Theorem 4.1.31 (that is yet to be proved in this document).

Proposition 5.2.1 ([Wel00b])**.** $\Sigma = \mathrm{T}$. *That is the supremum of the accidentally writable ordinal is equal to that of the accidentally clockable ordinals.*

This proposition, while stated here in a new way, is very close to [Wel00b, Corollary 3.5] which states that the collection of sets encoded by accidentally writable reals is actually L_Σ. In this article from Welch, it was a corollary of the fact that $\lambda = \gamma$ and this fact is itself a corollary of Theorem 5.2.1 here, as we'll see in a more general setting in Corollary 5.4.31. Another corollary is the fact that, with previous notations, $\zeta = \eta$. So this chain of consequences can be seen to justify this new phrasing. Moreover, it should be noted that the proof of this slightly more general result uses the same technique as that of the main proposition of [Wel00b]. Finally, we take for granted in this proof that there exists a

universal ITTM $\mathcal{U}$ and that we can design in a natural way machines that simulate other machines. This is actually true but we will come back to this.

Proof. First, suppose that $\Sigma > \mathrm{T}$. This implies that T is a.w. by some machine m_{T}. So we consider the following machine $\mathcal{M}$. It will use the code of T that appears at some point in m_{T} to write something for the first time after stage T, hence reaching a contradiction. To do this, first, it keeps the first cell of its working tape always set to 0 and the first cell of its output tape always set to 1. So when using its working tape, it uses cells $i > 0$. Then, it simulates m_{T} on some part of its working tape (again, without using cell $i = 0$) and each time an ordinal α is written in the simulation of m_{T}, it counts for at least α steps. Remember that by Proposition 4.1.12 it can recognize reals describing well-orders. This is done by trying to count through it, so it takes at least α steps to recognize that a real encodes a well-order of order type α. Then, it copies α on its output tape, without using the first cell that it keeps to 1. Now, when T appears for the first time at stage α_{T}: as T appears for the first time in m_{T} and T is the supremum of the a.c., that is of the stages of first appearance in a given machine, we have $\alpha_{\mathrm{T}} < \mathrm{T}$. Then, $\mathcal{M}$ counts for at least T steps and after those writes T on its output tape. This is the first time that T appears on this tape. Also, the whole content of this tape, with the first cell set to 1 cannot have appeared earlier in the working tape whose first cell is left on 0. Hence the real it writes was not written before and it writes it for the first time at least at stage $\alpha_{\mathrm{T}} + \mathrm{T} \geqslant \mathrm{T}$. And $\alpha_{\mathrm{T}} + \mathrm{T}$ is consequently a.c. which is a contradiction. So $\Sigma \leqslant \mathrm{T}$.

For the other direction, to show that $\Sigma \geqslant \mathrm{T}$, we will show that in any computation that does not halt, for any cell on any tape, the values of this cell at stages ζ and Σ match and moreover that a cell set to 0 at stage ζ stays forever so. This will prove that nothing new is ever written after stage Σ, as the machine will be repeating indefinitely the segment of computation $[\zeta, \Sigma[$ and so that $\Sigma \geqslant \mathrm{T}$. Let m be a machine and $i \in \omega$ be some cell on one of its tape.

- Suppose $C_i(\Sigma) = 0$. Then, by the limsup rule, the cell must have converged to 0 at some least ordinal stage $\alpha < \Sigma$. That is $C_i(\alpha) = 0$ and stays so up to Σ. We show that $\alpha < \zeta$ (this is actually Main Proposition of [Wel00b]). We design a machine $\mathcal{M}$ that does the following: it launches a copy $\mathcal{U}_1$ of the universal machine and for each ordinal β appearing in $\mathcal{U}_1$, that is accidentally written in a machine simulated by $\mathcal{U}_1$, $\mathcal{M}$ writes this ordinal on its output tape and simulates m up to this ordinal β. If $C_i^m(\beta) = 0$, it launches a new copy $\mathcal{U}_2$ of the universal machine and each time some ordinal $\beta' > \beta$ is appears in $\mathcal{U}_2$, it simulates a fresh copy of m up to stage β' and looks whether the cell i was set back to 1 between β and β'. If it did, $\mathcal{M}$ goes back to its simulation of the universal machine $\mathcal{U}_1$ and looks for the next such ordinal β, that is such that $C_i^m(\beta) = 0$. When $\beta > \alpha$, the machine never find 1's anymore in the history of cell i after this β, as all $\beta' > \beta$ generated are strictly between α and Σ. Hence, β is written on the output and $\mathcal{M}$ looks indefinitely for $\beta' > \beta$ at which

a 1 appear on cell i. And so $\mathcal{M}$ actually eventually writes some $\beta > \alpha$. As such, β is eventually writable and we have $\alpha < \beta < \zeta < \Sigma$ and $C_i(\zeta) = 0$ as well.

- Suppose now that $C_i(\zeta) = 0$. Then, the value of the cell converged at some $\alpha < \zeta$ and there exists a machine m_α that e.w. α. We consider the following computation from a machine that we call $\mathcal{W}$: It simulates m_α on some part of its working tape, which will eventually write α. Each time an ordinal β is written on the simulated output tape of m_α, it does the following: while β does not change (that is the simulation of m_α is intertwined by dovetailing with the rest of the computation) it simulates $\mathcal{U}$ and, each time $\mathcal{U}$ (accidentally) writes an ordinal $\beta' > \beta$, $\mathcal{W}$ starts by simulating m up to β and then up to β'. If it finds a 1 in the cell i between stages β and β', it writes β' on the output tape and stops this part of the computation. Note that as the simulation of m_α is still going on, it might start again from the beginning later, if its output value changes. However, at some point α is written in m_α and does not change anymore. At this point, if $\mathcal{W}$ finds a 1 between stages $\beta = \alpha$ and some $\beta' > \alpha$, this 1 must appear after ζ (as the value of the cell converges to 0 at stage α and up to stage ζ) and then β', which it has consequently eventually written is greater that ζ, which is a contradiction. Hence, the value of C_i never changes after stage ζ and $C_i(\Sigma) = 0$ and this conclude the proof.

□

Let us now show how Theorem 4.1.31, stating that $\lambda = \gamma$ and first proved in [Wel00b], is a corollary of this proposition.

Proof of Theorem 4.1.31. First $\lambda \leqslant \gamma$ as otherwise γ would be writable and, writing γ, it would be easy to clock an ordinal greater than it.

Then, if a machine halts, it must do so before stage Σ in virtue of Proposition 5.2.1. But then, it is easy to simulate this machine along the a.w. ordinals, using $\mathcal{U}$ to look for and write the a.w. ordinal stage at which this machine halts; which proves that every halting stage is also writable. □

Similarly, the second part of Theorem 4.1.33 is now more easily shown with this equality.

Proposition 5.2.2. (λ, ζ, Σ) *is the lexicographical least triplet yielding the chain of elementary end-extension*

$$L_\lambda \prec_{\Sigma_1} L_\zeta \prec_{\Sigma_2} L_\Sigma$$

Proof. Let $(\alpha, \beta, \delta) <_{lex} (\lambda, \zeta, \Sigma)$ such that $L_\alpha \prec_{\Sigma_1} L_\beta \prec_{\Sigma_2} L_\delta$. To start, we can show that this implies that any machine is seen to be looping between stages β and δ: first, by Σ_2-elementarity, the snapshot of any machine match at stages β and δ. Second, if

$C_i(\beta) = 0$, there exists $\beta' < \beta$ at which the value of cell i stabilized on 0 and, again by elementarity, this also holds in L_δ and so $C_i(\delta) = 0$. But by definition of T, the machines cannot all start looping before some stage strictly less T. And as $\Sigma = \mathrm{T}$, we also have that $\delta = \Sigma$.

Then, if $\beta < \zeta$, this means that β is e.w. Using it, a machine can do the following: e.w. β, simulate $\mathcal{U}$ until stage β, save its snapshot at stage β and continue with the simulation of $\mathcal{U}$ until this snapshot appears again. When it does, it means that $\mathcal{U}$ was simulated up to stage T. Using the same idea as for the proof of Proposition 5.2.1, it can arrange to write for the first time a real as this stage T, which would be a contradiction.

Eventually, if $\alpha < \lambda$: α is writable and as $L_\alpha \prec_{\Sigma_1} L_\delta = L_\mathrm{T}$ the models L_α and L_T agree on which machines halt. But any halting machine does so before stage T and consequently before stage α. In this case, the halting problem would be decidable, which is a contradiction.

□

The proof of Proposition 5.2.1 is somewhat of a classical proof in the theory of ITTMs, at least in its usage of simulations, and in particular in that of the simulation of the universal machine $\mathcal{U}$. Hence, the question that we postponed: how does the universal ITTM works? Does it actually exist and how can we describe it?

To answer this, observe that u, the classical three-tapes finite universal machine, gives almost immediately raise to an infinite time universal machine that simulates a single other ITTM. Indeed in the finite setting, u is such that given the code of some finite machine m, it simulates n steps of m in less than Cn^2 steps for some constant C, supposing that u works in a straightforward fashion. How does it work? Without going into the finicky details, every cell i of m is simulated by some cell I of u. That is, for any finite stage k, the value of I in u at stage Ck^2 is the value of i in m at stage k, and the next value to be written in I is that of the cell i at stage $k+1$. Outside of those cells I, the other cells of u are used to keep track of the simulation, in particular of the position of the head and of the state of the simulated machine m.

Now consider the ITTM $\mathcal{U}_0$ whose code is the same as u. Remember indeed that the structure of an ITTM is exactly that of a three-tapes classical Turing machine. We can also see m as an ITTM, which we denote by M. What happens when $\mathcal{U}_0$ computes from the code of M (viz., that of m)? By construction, for any finite stage k, the snapshot of $\mathcal{U}_0[\langle M\rangle]$ at stage k will be the same as that of $u[\langle M\rangle]$. Moreover, this also holds for the computations of M and m. Hence, for the first ω steps, the computation $\mathcal{U}_0[\langle M\rangle]$ is the same as $u[\langle m\rangle]$. In other words, the ITTM $\mathcal{U}_0$ with the code of M as input simulates the ITTM M through all the finite stages. But what happens a stage ω? That is, what is the snapshot of $\mathcal{U}_0[\langle M\rangle]$ at stage ω? Take some cell i of M. It is simulated by some cell I of $\mathcal{U}_0$. Hence, suppose i in M stabilizes on 0 before stage ω. That is, with the lim sup rule, $C_i^M(\omega) = 0$. Then, so does I in $\mathcal{U}_0[\langle M\rangle]$ and $C_I^{\mathcal{U}_0}(\omega) = 0$ and same goes for the

cells i' such that $C^M_{i'}(\omega) = 1$. Hence, the simulation of the limit rule for the simulated cell i comes directly from the limit rule itself in the simulating cell I. This mean that at stage ω in the computation $\mathcal{U}_0[\langle M\rangle]$, the content of the cells I describes exactly that of the cells i in M at stage ω. From there, to simulate the ITTM M further through the ordinals, only a few things are missing at limit stages: (1) the auxiliary cells used for the simulation will likely all be set to 1 and those need to be tidied up, (2) the simulated head of M should be back on the first simulated cell and (3) the simulated state of M should be q_{limit}. But as at any limit stage $\mathcal{U}_0$ reaches the distinguished state q_{limit}, it's easy from there to modify the machine $\mathcal{U}_0$ for it to tidy up its auxiliary cells and to set the correct head position and state for the simulated machine. This takes a finite amount of steps and after those the simulation goes on. This description yields a universal infinite time machine $\mathcal{U}'_0$ that simulates a single ITTM. From it we can simply enough describe the universal machine $\mathcal{U}$ that simulates all ITTMs at once. $\mathcal{U}$ works with its working tape as if it were ω different tapes in which it simulates in parallel all computations $\mathcal{U}'_0[m]$ for all machines $m \in \omega$. For $\mathcal{U}$ to split its working tape, this can be done as follows: first, all even cells (that is cells on the workings tape whose index i is even) are kept aside and will constitute a virtual working tape of their own that the machine uses to keep track of its simulations of all the $\mathcal{U}_0[m]$'s. The idea being that its easy for it to "stay" on this virtual tape while it reads it as it just needs to shift its head twice to the left or to the right of the real tape in order to shift it on this virtual tape. And using this virtual tape it may easily split the odd cells into ω tapes, e.g. the virtual tapes of some machine m are constituted of the cells $(f(m,i))_i$ for any computable function (in the finite sense) f that partitions the odd integers into ω unbounded sets. Then, when $\mathcal{U}$ begins, it initializes all its virtual tapes m with as would $\mathcal{U}'_0[m]$ initializes its working tape, this take ω steps. Then begins the simulation itself. By dovetailing, that is simulating the machines m in cascade, k steps of the first k machines, $k+1$ steps of the first $k+1$ machines, etc. it simulates ω steps of all of those machines in ω steps of itself; and this concludes the description of $\mathcal{U}$.

Now, why does the universal machine works? That is, what is implicitly used in this explanation? First, as a cell i of some machine m (we now drop the capital m and only consider ITTMs) will likely be simulated by a cell $I \neq i$, it uses the fact that every cell are governed by the same limit rule. Also, it uses the fact that the limit value of the cell I only depends of the history of this cell. Indeed, for a limit ordinal α, in order to say that $C^m_i(\alpha) = C^{\mathcal{U}}_I(\alpha)$, where $C^m_i(\alpha)$ denotes the value of cell i in m at stage α, we use the fact that h_i and h_I, the history of respectively the cell i in m and the cell I in $\mathcal{U}$ are *somewhat* the same and that the limit rule only looks at the histories h_i in m and h_I in $\mathcal{U}$. When a rule only looks at the history of a given cell to define the limit value of this cell, we will say that the rule is *cell-by-cell*. But what was meant by "*somewhat* the same"? Are the histories h_i and h_I not actually equal? Suppose that m is a very simple machine that continuously blinks its cell i. First the head of m needs i steps to reach i. As the cell i is initially set to 0, its history begins with 0^i, that is the word of length i. At stage

$i+1$, the cell i in m is set to 1. Its history is now 0^i1. And then back to 0, then again to 1 and so on. Hence, at any limit stage α, its history reads $0^i(10)^\alpha$. Now what does the history of cell I in $\mathcal{U}$ read? We can merge in $\mathcal{U}$ the ω steps of initialization with the actual simulation, but in any case, the machine will begin the actual simulation of m after some k strictly greater than i. Hence, h_I begins with 0^k. At stage $k+1$, $\mathcal{U}$ writes a 1 in cell I and the history of H_i reads 0^k1. But, and this is one of the most important point, after having simulated this step from k to $k+1$ in m, $\mathcal{U}$ will then be simulating some steps in the machine $m+1$ and so on. And it won't be coming back to m before some finite amount of steps k_2. In the meantime, during those k_2 steps, the cell I stays set to 1. So, after those k_2 steps, H_i reads $0^k1^{k_2}$. And as the computation goes on, H_i will look like: $0^k1^{k_2}0^{k_3}\dots1^{k_\alpha}0^{k_{\alpha+1}}\dots$ where each k_ν are finite and greater than 0. So, to some extent, h_i and H_i are not quite alike. Still we could say that h_i is the *contraction* of H_i, that is the word obtained by contracting any 1^k or $0^{k'}$ respectively to 1 and 0. And $\mathcal{U}$ operates as it should because, as we could say it, the lim sup rule is *contraction-proof*. The limit value of a cell does not change if we contract its history. And here, we could say that the dilatation from h_i to H_i is only finite as all k_ν are finite, but in practice, when we use $\mathcal{U}$ or when we simulate any other machine, it very common to keep it "on hold" for extensive periods of computation, as in the first part of the proof of proposition 5.2.1. In this case the dilatation factors of the history, that is the factors k_ν, may be infinite ordinals and *a priori* unboundedly big.

Other than that, keeping a simulated machine "on hold" for some infinite ordinal amount of steps α, implicitly makes use of the fact that the cells that aren't written on (i.e. whose content is not modified) won't change their content at limit stages. We can say the the lim sup rule is *stable* as the content of a cells which stabilized up to at limit stage does not change at the limit stage.

Eventually, there is a last feature of the Hamkins and Lewis ITTMs that is used implicitly. Take the computation of some machine m up to some limit stage α. Then consider a fresh computation of m in which the initial snapshot (that is the content of the tapes, the state and the head position) is set to match that of m after α steps. In this second computation, stage 0 corresponds to stage α in the first computation. Still in both cases it leads to the same subsequent computation. Hence, for any stage β we'd like that for all i, writing s_α the snapshot of m at stage α and $C_i^{m,s}$ for the function that maps an ordinal to the content of cell i at this ordinal stage in the computation of m that starts with s as its snapshot:

$$C_i^m(\alpha+\beta) = C_i^{m,s_\alpha}(\beta)$$

This clearly holds in the classical model of ITTM for finite α and β. For α and β limits, it means that even if the history h_i of length $\alpha+\beta$ is truncated to the final segment h'_i of length β, it still yields the same limit value w.r.t. the lim sup rule. This mean more

generally that for any histories h and h' such that h' is a final segment (or suffix) of h, then h and h' yields the same limit value. We can say that the lim sup is *asymptotical*, as the limit value that an history yields does not change when we take a final segment of this history.

This quick glance into the characteristics of the lim sup or lim inf rules that are implicitly used in the universal machine leads us to distinguish four characteristics (formalized below): those limit rules are *cell-by-cell*, *contraction-proof*, *asymptotic* and *stable*. And, as we will see, those conditions are sufficient for the universal machine to be easily described and, may even be necessary conditions for it to exists in a satisfying form. To those, we can add the *looping stability* which somehow equates to the fact that a machine can be seen to be looping without looking at its entire computation through On. For example, in the lim sup ITTM, a machine is seen to be looping if there are two stages sharing the same snapshot such that cells that are set to 0 at both of those stages are also set to 0 for the whole segment of computation that spans between those two stages. Without it, we could easily conceive a pathological machine that repeats for some gigantic ordinal amount of time after which, thanks to its rule, that escapes the repeating pattern.

As stated, the remarkable result, which we show at the beginning of Section 5.4 is that the lim sup and lim inf rules are the only 2-symbol rules that satisfy this set of five characteristics. In other words, they are the only 2-symbol rules that allow for natural simulations and that have a natural universal machine, as well as satisfying the condition of looping stability. Further, again as mentioned, the main theorem of this chapter shows the following strong result: for a n-symbol rule definable by a set-theoretic formula and that satisfies those five conditions, the supremum of the a.w. ordinals matches with that of the a.c. ordinals, that is $\Sigma = \mathrm{T}$ for this particular rule. In the last section, we provide a counter-example to show that this theorem is tight. Namely, we show that there exists a limit rule that satisfies all those conditions but that of looping stability and such that for the machines produced by this rule, the equality $\Sigma = \mathrm{T}$ does not hold.

5.3 General definitions and conditions on operators

In order to formaly study limit rules, we need first to define what a limit rule is. In [Wel00a], Welch introduces the concept of operator. A n-symbol limit rule can be seen as an operator $\Gamma : {}^{<On}({}^{\omega}n) \to {}^{\omega}n$, that is a function that given any computation history seen as an ordinal-indexed limit sequence of n-symbol reals (corresponding to tape contents) yields a n-symbol real that will represent the tape content at the next limit stage. As noted, the data of an operator Γ is enough to produce a model of infinite time Turing machine whose limit stage behavior is ruled by this operator. Such machines will be called n-symbol Γ-machines. We start by introducing some general concepts.

5.3.1 General definitions on ordinal words

Definition 5.3.1 (Ordinal word)**.** An *ordinal word* w of length λ on some alphabet A is a function from λ to A that maps every ordinal $\iota < \lambda$, seen as a *position* or *index* in the word, to a letter in A. We write $w[\iota]$ for the letter at position ι in w and $w[\iota, \kappa[$ the subword of w staring at position $\iota < \kappa$ up to position κ non included. We write $|w|$ for the length of the word w and when $|w|$ is finite, this coincides with the definition of words in the ordinary theory of formal languages. When $|w|$ is a limit ordinal, we say that w is a limit word. Eventually, we write ${}^{\alpha}\mathrm{A}$ for the set of ordinals words of length α on the alphabet A and we write ${}^{<On}\mathrm{A}$ for the class of ordinal words of any length on the alphabet A. That is:

$${}^{<On}\mathrm{A} = \bigcup_{\alpha \in On} {}^{\alpha}\mathrm{A}$$

Example 5.3.2. Consider an infinite Turing machine m with two symbols, 0 and 1, that computes with any limit rule and whose computation reaches some stage α. For any cell i, this induces a function C_i that maps $\beta \leqslant \alpha$ to the value of the cell i at stage β. This also induces an ordinal word h_i of length $\alpha + 1$ on the alphabet $2 = \{0, 1\}$ such that for any $\beta \leqslant \alpha$, $h_i[\beta] = C_i(\beta)$. Also, the content of the three tapes at any stage β can be described, as an ordinal word $w_\beta \in {}^{\omega}2$, that is a word of length ω on the alphabet 2. This can simply be done using an usual encoding to describe the three tapes of length ω in a single word of length ω. Combining both those points of view, this computation induces an ordinal word $W \in {}^{\alpha+1}({}^{\omega}2)$, that is a word of length $\alpha + 1$ on the alphabet ${}^{\omega}2$ such that for any stage $\beta \leqslant \alpha$, $W[\beta] = w_\beta$.

Definition 5.3.3 (Stutter-free)**.** Let h be an ordinal word on some alphabet A. We say that h is stutter-free when for all $\alpha + 1$ index of h, $h[\alpha] \neq h[\alpha + 1]$; that is the α^{th} letter of h is different from the $\alpha + 1^{th}$.

Definition 5.3.4 (Suffixes and prefixes)**.** Given u and v two ordinal words, we say that u is a *prefix* or an *initial segment* of v, written $u \sqsubseteq v$ when $|u| \leqslant |v|$ and for all $\iota < |u|$, $u[\iota] = v[\iota]$. We say that u is a *suffix* or a *final segment* of v, written $v \sqsupseteq u$ when $|u| \leqslant |v|$ and there exists α such that $\alpha + |u| = |v|$ and for all $\iota \in [\alpha, |v|[$, that is for all $\iota = \alpha + \iota' < |v|$, we have $u[\iota'] = v[\iota]$.

Definition 5.3.5 (Operations on words)**.** Given two words u and v and α an ordinal, we write uv for the concatenation of u and v of ordinal length $|u| + |v|$ and u^α the word made from u concatenated α times to itself, of length $|u| \cdot \alpha$.

5.3.2 General definitions on machines

Definition 5.3.6 ([Wel00a])**.** For a natural number n, a *n-symbol limit rule operator* Γ is a (class) function $\Gamma : {}^{<On}({}^{\omega}n) \to {}^{\omega}n$ that maps ordinal words on the alphabet ${}^{\omega}n$, seen as

computation histories of n-symbol machines, to element of ${}^{\omega}n$, seen as tape contents. A Γ-*machine* is an infinite Turing machine whose limit rule is that induced by the operator Γ.

We now introduce basic definitions regarding Γ-machines. Naturally, as Γ-machines and Hamkins' ITTM have the same structure, most of those definitions are identical to the definition given in the specific context of Hamkins' ITTM.

Definition 5.3.7. We write Γ_{sup} and Γ_{inf} the operators associated respectively to the lim sup and the lim inf rule.

Definition 5.3.8 (Looping, [HL00])**.** For Γ a limit operator and m a Γ-machine, we say that m is *looping* when the machine never halts and some interval of computation $[\alpha, \alpha + \beta[$ repeats itself through the whole computation of m after stage α. That is for any stage $\nu \geqslant \alpha$, writing $\nu = \alpha + \beta \cdot \delta + \nu'$ with δ maximal, the snapshot of m at stage ν is the same as that of m at stage $\alpha + \nu'$.

Definition 5.3.9 (Writable reals, [HL00])**.** For Γ a limit operator, a real x is Γ-*writable* if there is a Γ-machine which, when computing from the empty input, halts with x written on its output.

Definition 5.3.10 (Converging, [HL00])**.** For Γ a limit operator and m a machine computing on some input y we say that m *converges* or *converges definitively* when it never halts and when after some stage its output tape is never modified again. We say that m *converges up to some ordinal* α when it does not halt before stage α and after some stage $\beta < \alpha$ its output tape is never modified before stage α. Observe that the definition of converging involves only the output tape: often, converging machines will continue to modify their working tape through the whole computation.

Definition 5.3.11 (Eventually writable reals, [HL00])**.** For Γ a limit operator, a real x is Γ-*eventually writable* if there is a Γ-machine which converges when computing from the empty input and has x written on its output tape when the content of its output tape stabilized.

Definition 5.3.12 (Accidentally writable reals, [HL00])**.** For Γ a limit operator, a real x is Γ-*accidentally writable* if there is a Γ-machine which, when computing from the empty input, at some stage has x written on any of its tapes

As with ITTMs, we want to encode ordinals using reals. We give again the encoding which we use.

Definition 5.3.13. We say that a real x *encodes* some ordinal α when the real encodes a relation $\prec$ on ω of order type α in the the following fashion.

$$i \prec j \longleftrightarrow x[\langle i, j \rangle] = 1$$

Definition 5.3.14 (Writable ordinals, [HL00])**.** For Γ a limit operator, an ordinal α is Γ-*writable* (resp. Γ-*eventually writable* and Γ-*accidentally writable*) if some x that encodes α is Γ-writable (resp. Γ-e.w. and Γ-a.w.) We write λ_Γ, ζ_Γ and Σ_Γ respectively for the supremum of the Γ-writable, Γ-e.w. and Γ-a.w. ordinals.

Definition 5.3.15 (Clockable ordinals, [HL00])**.** An ordinal α is Γ-*clockable* if there is a Γ machine computing from the empty input that halts a stage α. That is, it computes for α steps and then, on its next transition, it reaches the state q_{halt}. We write γ_Γ for the supremum of the Γ-clockable ordinals.

With those definitions, what Hamkins and Lewis asked in [HL00] is whether $\lambda_{\Gamma_{\text{sup}}} = \gamma_{\Gamma_{\text{sup}}}$. As noted, Welch answered this question positively in [Wel00b]. More generally, we may wonder under which conditions on Γ does the equality $\lambda_\Gamma = \gamma_\Gamma$ holds. As done in the discussion of Section 5.2, we can get a better insight on the question by introducing the following extensions of the concept of clockable, akin to that of the concept of writable.

Definition 5.3.16 (Eventually clockable ordinals)**.** An ordinal α is Γ-*eventually clockable* if there is a Γ-machine which, computing from the empty intput, never halts and whose output tape stabilizes at stage α; that is, it changes its content upon reaching this stage and it never subsequently changes it. We write η_Γ for the supremum of the Γ-eventually clockable ordinals.

Definition 5.3.17 (Accidentally clockable ordinals)**.** An ordinal α is Γ-*accidentally clockable* if there is an Γ-machine which, when computing from the empty input, writes at stage α on one of its tapes a real that wasn't written at any previous stage of this computation on any tape. We write T_Γ, capital τ, for the supremum of Γ-accidentaly clockable ordinals.

In Figure 5.1, we update the table of Figure 4.1 with η and T.

	simply	eventually	accidentally
writable	λ	ζ	Σ
clockable	γ	η	T

Figure 5.1: Greek letters associated to each definition.

Now, as before, given some limit operator Γ, the question becomes: does $\Sigma_\Gamma = \mathrm{T}_\Gamma$? This is answered positively in Theorem 5.4.14 for a wide range of operators. And we show in Corollary 5.4.30 and Corollary 5.4.31 that $\Sigma_\Gamma = \mathrm{T}_\Gamma$ implies both the equalities $\zeta_\Gamma = \eta_\Gamma$ and $\lambda_\Gamma = \gamma_\Gamma$.

5.3.3 Conditions on operators

5.3.3.1 Simulational operators

Definition 5.3.18 (History of computation)**.** Given a n-symbol operator Γ and a Γ-machine m that does not halt before stage α, the *history H of the computation of m up to α* is an element of ${}^{\alpha}({}^{\omega}n)$, that is, it is an ordinal word of length α on the alphabet made from n-symbol reals, and such that for all $\beta < \alpha$, $H[\beta] = C^m(\beta)$, meaning that the β^{th} letter of H is the content of the tapes of m at stage β (representing the three tapes with a single real). For a given cell i, the *cell history h_i of the computation of m up to α* is an element of ${}^{\alpha}n$ such that for all $\beta < \alpha$, $h_i[\beta] = C^m_i(\beta)$. Further, $h_i = H|_i$, the restriction of H to the cell i. When α is a limit ordinal, we say that H (resp. h_i) is a *limit history* (resp. a *limit cell history*).

Definition 5.3.19 (Suitable operator, [Wel00a])**.** An operator Γ is a *suitable* n-symbol operator if there is a set-theoretic first-order formula $\varphi(x_1, x_2, x_3, x_4)$ such that for any machine m, input y, cell i, symbol k and stage ν, writing H_ν the history of m up to stage ν and L_ν the ν^{th} level of the constructible hierarchy, we have

$$\Gamma(H_\nu)[i] = k \iff L_\nu[y] \models \varphi(i, m, y, k)$$

When φ is a Σ_n formula, it is a Σ_n *operator* and it produces Σ_n *machines*.

Remark 5.3.20. *We may want to restrict the symbols produced at limit stages to* 0 *and* 1*, that is* Γ *would be an operator which acts from* ${}^{<On}({}^{\omega}n)$ *to* ${}^{\omega}2$*. In this case, it is enough to provide a first-order formula* $\varphi(x_1, x_2, x_3)$ *such that, with the same notations,*

$$\Gamma(H_\nu)[i] = 0 \iff L_\nu[y] \models \varphi(i, m, y)$$

From there, we can formalize the different conditions exhibited and informally defined in the previous section.

Definition 5.3.21 (Stable)**.** An operator Γ is *stable* when for any real x and limit ordinal α, with x^α denoting the word made from the real x, seen as a letter, repeated α times, we have

$$\Gamma(x^\alpha) = x$$

Definition 5.3.22 (Cell-by-cell)**.** An operator Γ is *cell-by-cell* when there exists $\gamma : {}^{<On}n \to n$ such that, given any limit history H and for all cell i, we have

$$\gamma(H|_i) = \Gamma(H)|_i$$

where $|_i$ denotes the restriction of the tape history H, possibly made of a single tape

content, to a single cell; hence yielding a single cell history, that is an element of ${}^{<On}n$. We say that γ is the cell restriction of Γ.

Definition 5.3.23 (Asymptotic)**.** A limit rule operator Γ is *asymptotic* when for any limit history H of length α and any non-empty final segment H' (with previously introduced notation, $H \sqsupseteq H'$), we have $\Gamma(H) = \Gamma(H')$

Example 5.3.24. Let us see how a Γ-machine m behaves when Γ is asymptotic. For the sake of simplicity, we suppose that m only has a working tape. We write $H(\alpha)$ for the history word of length α of the working tape up to some stage α. Again to make thing simpler, suppose that at stage 0 some real x is on the working tape and the machine is in the state q_{lim}. Suppose further that there is a limit stage $\alpha > 0$ such that $\Gamma(H(\alpha)) = x$. Then, we look at the computation of m after this stage α and we show how this gives rise to a repetition in the computation; but a repetition that may not be a loop.

By definition of the model of ITTMs, at any stage $\beta + k$ for a finite k, the snapshot of m depends only of its code and of its snapshot at stage β. In this case, as only the working tape is used, in the computation of m stages 0 and α share the same snapshot. This mean that, for any finite k, the snapshots at stage k and $\alpha+k$ match. That is, $H(\alpha+\omega) \sqsupseteq H(\omega)$. Then, what happens at limit stage $\alpha+\omega$? By asymptoticity, $\Gamma(H(\alpha+\omega)) = \Gamma(H(\omega))$. And so, snapshots at stages ω and $\alpha + \omega$ also match. Further, inductively, we can show that $\Gamma(H(\alpha \cdot 2)) = \Gamma(H(\alpha)) = x$. And further that $\Gamma(H(\alpha \cdot k)) = \Gamma(H(\alpha)) = x$ for any finite k. Is then m actually looping? Consider stage $\alpha \cdot \omega$. This stage is the first stage after α whose ordinal number is additively closed. Consequently, by additive closeness, any non-empty final segment of $H(\alpha \cdot \omega)$ has itself $H(\alpha \cdot \omega)$ as final segment. Hence applying asymptoticity in this case only yields that $\Gamma(H(\alpha \cdot \omega)) = \Gamma(H(\alpha \cdot \omega))$ and without more assumptions, the limit rule may very well make m exit the repeating pattern at stage $\alpha \cdot \omega$.

For the next condition, we need a prior definition. What we are aiming to define is the contraction of a word w, that is the stutter-free word which is made by squeezing any repeated a symbol in w into a single symbol, and that until no more repetition can be found. For example, the contraction of the word $aabc^\omega$ would be abc. We define by transfinite induction the operator ctr, which maps a word to its contraction.

Definition 5.3.25 (Contraction)**.** We define ctr for non-limit ordinal words as follows. Let w be a word on some alphabet A and $a \in A$. ε denotes the empty word.

$$\text{ctr}(\varepsilon) = \varepsilon$$

$$\text{ctr}(wa) = \begin{cases} \text{ctr}(w) \text{ if } w \text{ is not limit and ends with a } a \\ \text{ctr}(w) \text{ if } w \text{ is limit and has a final segment constituted only of } a\text{'s} \\ \text{ctr}(w)a \text{ else} \end{cases}$$

For words whose ordinal length is limit, ctr is defined as a limit of prefixes. Let w be such a word over A.

$$\mathrm{ctr}(w) = \lim_{u \sqsubseteq w} \mathrm{ctr}(u)$$

That is, the contraction of w is defined as the limit of the contraction of its prefixes. For this definition to be licit, we need to show that this limit is well-defined. This is the object of the next proposition.

Proposition 5.3.26. *The previous limit is well-defined and with it the definition of* ctr *is licit.*

Proof. We show by induction on the length of v that ctr is well-defined and that if $u \sqsubseteq v$ then $\mathrm{ctr}(u) \sqsubseteq \mathrm{ctr}(v)$.

First, $\mathrm{ctr}(\varepsilon)$ is well-defined and the second condition is immediate.

Then let u be an ordinal word and a a letter. Looking at the definition of ctr it is clear that if $\mathrm{ctr}(u)$ is well-defined, then so is $\mathrm{ctr}(ua)$ and further that $\mathrm{ctr}(u) \sqsubseteq \mathrm{ctr}(ua)$. For two words u and v such that $u \sqsubset v$, if v is not a limit word, it can be written as $v'a$. Hence, in this case, by induction and by the previous observation, ctr is well-defined for all of those and $\mathrm{ctr}(u) \sqsubseteq \mathrm{ctr}(v') \sqsubseteq \mathrm{ctr}(v)$.

In the other case, if v is a limit word, then for any prefixes v' and v'' of v and such that $v' \sqsubseteq v''$, by induction, their contraction is well-defined and $\mathrm{ctr}(v') \sqsubseteq \mathrm{ctr}(v'')$. Hence, $\mathrm{ctr}(v)$ is also well-defined, as a growing limit of prefixes. And by construction, for any $v' \sqsubseteq v$, $\mathrm{ctr}(v') \sqsubseteq \mathrm{ctr}(v)$, as required. □

Definition 5.3.27. We say that two words u and v are *equal after contraction*, written $u \simeq_{ctr} v$, whenever $\mathrm{ctr}(u) = \mathrm{ctr}(v)$

Below are a few examples of contraction of words and words equal up to contraction. This wholly coincides with the intuitive idea of contracting every sequence of a symbol which repeats itself into a single symbol.

Example 5.3.28.

- $aaabcccc \simeq_{ctr} abbbc$.
- The contraction of the limit word b^ω, that is of the limit word in which the letter b is repeated ω times, is the single-letter word b, made from the letter b.
- For $\alpha, \beta > 0$ and any δ, $(b^\beta a^\alpha)^\delta \simeq_{ctr} (ba)^\delta$.

Definition 5.3.29 (Contraction-proof)**.** A operator Γ is *contraction-proof* when for any limit histories H and H' such that $H \simeq_{ctr} H'$, we have $\Gamma(H) = \Gamma(H')$.

Remark 5.3.30. *As the contraction of a limit word may be a non limit word (take for example b, the contraction of b^ω), general operators should not only be defined on limit words. However, any non limit word w finishing with some letter a is equal up to contraction to wa^ω. So, when considering contraction-proof operators, as we will do, it is enough for those to be defined only on limit words.*

Remark 5.3.31. *Observe that for a cell-by-cell operator, those three characteristics, namely being stable, asymptotic or contraction-proof, are naturally refined to cell-wise properties. For example, if we take the case of the property of a cell-by-cell operator Γ being contraction-proof and write γ its associated cell restriction. Take h and h' two limit cell histories such that $h \simeq_{ctr} h'$. Then we can consider H and H' such that for all i, $H|_i = h$ and $H'|_i = h'$. By the properties of being cell-by-cell and contraction-proof, we have:*

$$\gamma(h) = \gamma(H|_i) = \Gamma(H)|_i = \Gamma(H')|_i = \gamma(H'|_i) = \gamma(h')$$

which is simply the fact that γ is contraction-proof as well, in a cell-by-cell fashion. Conversely, if γ satisfy this cell-by-cell contraction-proof property, it is clearly carried over to Γ.

Definition 5.3.32 (Simulational operator)**.** We say that a limit rule operator Γ is a *simulational* limit rule operator when it is stable, cell-by-cell, asymptotic and contraction proof.

Proposition 5.3.33. *Let Γ be a simulational limit rule operator. Then there exists a universal machine U_Γ and more generally, a Γ-machine can be designed to compute using the simulation of any other Γ-machine.*

Proof. We use the same reasoning as in the description of the lim sup universal machine. It is easy, in virtue of the fact that Γ is cell-by-cell and given the code of the finite universal machine u, to describe a universal Γ-machine $\mathcal{U}_0$ that simulates a single other Γ-machine. From there the description of U_Γ from $\mathcal{U}_0$ is as previously: we can arrange to virtually split the working tape into ω virtual workings tapes in each of which, for $m \in \omega$, we run $\mathcal{U}_0$ that simulates m. We can also run those machines in cascade (that is n steps of the n^{th} first machine, $n+1$ steps of the $(n+1)^{th}$ first machine and so on) so that in ω steps, we run ω steps of each of those machines, and more generally at any α limit the machine simulated α steps of each simulated machine. And this produces the desired computation, that is at limit stages, the tapes of the simulated machine have the same content as the machines would have because Γ is contraction-proof.

Now for the second part. What is a natural way for a machine to use the simulation of another? As briefly sketched, it should first be able to run the simulation of any machine from any snapshot and for as long it wants. And second, it should be able to put this

simulation on hold also for as long as it wants. As Γ is cell-by-cell, asymptotical and contraction-proof, a machine can run simulations of other machine from any snapshot. As it is stable and contraction-proof, it can put the simulation on hold for as long as it wants. □

Remark 5.3.34. *Let us now look at the hypermachines developped by Welch and Friedman in [FW11]. We give a brief overview of the definition of those machines in order to put them under the scrutiniy of those new notions. A more thorough description of those machines was done in Section 4.2.2.*

The structure of the hypermachine is akin to that of a 2-symbol ITTMs with the difference that it has a new tape called a rule tape. At a limit stage α, the value of the cells is computed by a discretionary $\liminf$ *operator, that is a* $\liminf$ *operator that considers only a precise subset of the previous stages over which it takes the* $\liminf$*. Those stages chosen for the* $\liminf$ *are called by the authors 1-correct in α. Hence, the limit operator can be written like this, for a limit α:*

$$C_i(\alpha) = 1 \longleftrightarrow \exists \beta < \alpha \, \forall \delta \in [\beta, \alpha[\ (\delta \text{ is 1-correct in } \alpha \implies C_i(\delta) = 1)$$

To define the 1-correct ordinals in α, the authors first define the 1-stable ordinals in α. With the previous notations, 1^n stands for the words of length n made only of the letter 1 and $1x$ stands for the (possibly infinite) word made of the word 1 concatenated to the left of the word x. Then, some ordinal $\beta \in \alpha$ is 1-stable in α when it is a multiple of ω^ω and when the following holds: if any pattern of the form $0^n 1x$ for $x \in {}^\omega 2$ appeared as the content of the rule tape before stage α and if $1x$ appeared as the content of the rule tape before stage β, then the pattern $0^n 1x$ also appeared before β. That is:

$$\forall x \in {}^\omega 2 \, \forall n \, \forall \delta < \alpha \, \forall \delta' < \beta \, (C(\delta) = 0^n 1x \wedge C(\delta') = 1x \implies \exists \delta'' < \beta \, C(\delta'') = 0^n 1x)$$

The authors write $S^1_\alpha \subseteq \alpha$ for the 1-stable ordinals below α. From there, they define E^1_α the 1-correct ordinals below α as follows:

$$\begin{cases} E^1_\alpha = \{\alpha' < \alpha \mid S^1_\alpha \cap \alpha' = S^1_{\alpha'}\} \text{ if } \alpha \text{ is a limit of multiple of } \omega^\omega \\ E^1_\alpha = \alpha \text{ else} \end{cases}$$

Now, as this operator needs to look at the whole rule tape, it is not cell-by-cell. However, it is easily seen to be stable. Then, if it was asymptotic, at some limit stage $\alpha \cdot 2$ such that $\alpha \cdot 2$ is a limit of multiple of ω^ω, the 1-correct ordinals would not depend on the first half of the history. However, for all 1-stable $\beta \in [\alpha, \alpha \cdot 2[$, if there is x such that $0^n 1x$ appears for the first time at some stage $\alpha' \in]\beta, \alpha' \cdot 2[$, then modifying the first half of the history so that $1x$ appears before stage $\alpha' < \alpha$ would change the status of β w.r.t. 1-stability. Hence, we could design a carefully tailored history for which the 1-stability of

cofinally many β's depend on the first half of the history. And so, for each β there would be some α' between β and α for which we have $\beta \in S^1_{\alpha'} - S^1_\alpha$. That is, the status of α' w.r.t. 1-correctness in α would be modified when we modify the first half of the history of the rule tape. Eventually, considering a cell outside of the rule tape which is set to 1 only at those stages α' (which are cofinals) would be enough to contradict asymptoticity. Finally, as there exists ordinals limit of multiple of ω^ω that are not additively closed (e.g. $\omega^\omega(\omega^2+\omega)$), a contraction or a dilatation of the historic may change the length of the historic and so the way 1-correct ordinals in α are determined, and with it the whole behavior of the limit rule at limit stage α.

This shows that is not possible to define a universal machine for the hypermachines with the technique used in the previous proposition. It may still exist but the presence of the rule tape makes a constructive approach seem difficult.

The following proposition can be seen as a closure property. It is of the utmost importance and is ubitquitous in what follows, often under the guise of the formulation given in Remark 5.3.36.

Proposition 5.3.35. *Let Γ be a simulational limit rule operator. If a real x is Γ-e.w. and y is x-Γ-writable, then y is Γ-e.w.*

Proof. For this proposition, we show how we can apply a dovetailing technique that will be used extensively in what follows. The real x being e.w. means that there is a machine m_x which, when computing from the empty input, does not halt and at some point has written x on its output and does not modify it anymore. The real y being x-Γ-writable means that there is a machine m_y which, when computing with x as input, halts and has y written on its output when it does. From there, it would seem natural to consider the following computation that e.w. y: some machine simulates m_x and when x appeared in m_x, it computes y from it using m_y, which allows it to e.w. y from the empty input. However the difficulty lies in the fact that this machine can't decide when x actually appeared. To circumvent this difficult, we can use the a dovetailing technique.

Consider the following machine: it simulates m_x. This is possible given that Γ is a simulational operator. At each simulation stage of m_x, it does the following: if the output of m_x was modified, writing $\tilde{x}$ for its new output, it initializes a fresh simulation of $m_y[\tilde{x}]$ (eventually discarding a previous simulation of it), that is of m_y on the input $\tilde{x}$. while the output of m_x is not modified, the machine run in parallel the simulation m_x and the simulation $m_y[\tilde{x}]$. If the output of m_x is modified, we go back to the beginning of the description with the new output of m_x. Observe that the simulation $m_y[\tilde{x}']$ may also halt at some point, in which case the parallel simulation continues with the convention that running a step of m_y simply does nothing. Moreover, at each step of the simulation of m_y, the main machine copies the output of m_y to its own output.

Now, what happens in this computation? Until m_x reaches x, the output of m_x regularly changes and so simulation of m_y is restarted again and again, even if it halted,

each time discarding what was done and starting from a new input. When x appears not to be modified again, observe that this is the last time the simulation of m_y is restarted. It now computes from input x and by our assumptions, the output of m_y eventually reads y and m_y halts. However m_x does not halt–as it never does– and also never changes. As the output of m_y was copied on the of the main machine, y has been written on the main machine output not to be changed again; and so was e.w. □

Remark 5.3.36. *Proposition 5.3.35 justifies the following formulation widely used in the following proofs:*

> *To show that y is e.w., we consider the following machine: first it simulates m_x to e.w. x on some part of its working tape. Then using x it conducts the following computation to write y on its output...*

What is hidden behind this slightly abusive de-interlacing or sequencing of the different writing operations is the more complex but sound dovetailing technique presented in the previous proof.

Proposition 5.3.37. *Let $\Gamma_{\sup}$ be the operator of the limsup rule. $\Gamma_{\sup}$ is a suitable and simulational limit rule Σ_2-operator.*

Proof. $\Gamma_{\sup}$ is a suitable Σ_2-operator: We first define by recursion a Σ_1 function η uniformly in α such that in L_α for a machine m, an input y and an ordinal $\nu < \alpha$, $\eta(m, y, \nu) = \langle h_i^\nu \rangle_{i\in\omega}$ where h_i^ν is the history of cell i up to stage ν. Observe first that as L_α may not be admissible, we can't use Σ-recursion as was developed in [Bar75]. However, with what follows we can show by induction that $\langle h_i^\nu \rangle$ is definable over L_ν, and so in $L_{\nu+1}$ and with it in L_α. This patches the use of Δ_0-collection (see [Bar75, p. 27]). Indeed, observe that for a successor ordinal ν, writing $\nu = \alpha + k$ where α is a maximal limit ordinal (possibly 0) and k an finite ordinal, $\eta(m, y, \nu)$ is easily definable from $\eta(m, y, \alpha)$: using the definition of the lim sup rule and the history $\eta(m, y, \alpha)$, it is possible to define the snapshot of m at stage α over L_α. And then, from this snapshot and $\eta(m, y, \alpha)$, the code of m is enough to define the whole history $\eta(m, y, \nu)$ over L_ν. For a limit ordinal ν, $\eta(m, y, \nu)$ is simply the limit of the $\eta(m, y, \nu')$ for $\nu' < \nu$.

With this, we can define $\varphi_{\sup}$. As the history up to stage ν is too big to be in L_ν, we can't use our function η this way. So this is not as straightforward as we might want but it is still relatively easy as we can use η with any $\nu' < \nu$. This yields the following Σ_2 definition (as the Σ_1 function η is applied in the left-hand part of the implication):

$$\varphi_{\sup}(i, m, y) := \exists\alpha\, \forall\beta\, \forall h_i^\beta\, (\alpha < \beta \wedge h_i^\beta = \eta(m, y, \beta)[i] \implies h_i^\beta[\beta] = 0)$$

And with this definition, we have, as wanted, for m a $\Gamma_{\sup}$-machine and H_ν the history of m up to some stage ν:

$$\Gamma_{\sup}(H_\nu)[i] = 0 \longleftrightarrow L_\nu[y] \models \varphi_{\sup}(i, m, y)$$

And $\Gamma_{\sup}$ is a simulational operator: it is stable, asymptotical and cell-by-cell by definition. Then let h and h' be two cell histories and write $\gamma_{\sup}$ the restriction of $\Gamma_{\sup}$ on a single cell. Suppose that $h \simeq_{ctr} h'$. Then $\gamma_{\sup}(h) = 0$ means that a final segment of h is of the form 0^α. As $h \simeq_{ctr} h'$, same goes for h' and $\gamma_{\sup}(h') = 0$. By symmetry, the other direction also holds and this proves that $\Gamma_{\sup}$ is contraction-proof as well. □

5.3.3.2 Looping stability

The possibility for an infinite machine to exit a repeating pattern, as presented in Example 5.3.24, is in a way what enables those machines to compute for so long and to give rise to intricate computations. At the same time it may lead to pathological patterns from the point of view of those being computation models. As presented in Remark 5.3.39, this could result in a Γ-machine which repeats for a very long time (very long when compared to the time it took to enter this repeating pattern) and then which suddenly exits this pattern. To avoid those cases, we introduce in Definition 5.3.38 the notion of *looping stability*. It is a simple assumption that let us develop a rather general and sound subclass of the simulational operators for which we can prove strong results like that of Theorem 5.4.14.

Definition 5.3.38 (Looping stability)**.** A limit rule operator Γ satisfies the *looping stability* condition when for any limit history H and limit ordinals α and α', writing H^α for the word in which H is repeated α times, we have:

$$\Gamma(H^\alpha) = \Gamma(H^{\alpha'})$$

Equivalently, in a formulation that will be used more often, for any limit history H and limit ordinal α:

$$\Gamma(H^\alpha) = \Gamma(H^\omega)$$

Remark 5.3.39. *As seen in Example 5.3.24, the asymptoticity of an operator implies the looping stability for some α's, e.g. for $\alpha = \beta + \omega$, by asymptoticity $\Gamma(H^\alpha) = \Gamma(H^\beta \cdot H^\omega) = \Gamma(H^\omega)$. However, it gets trickier when α is additively closed: any final segment of H^α has the same length as H^α itself and is of the form $H_f \cdot H^\alpha$ where H_f is some final segment of H. In this case, asymptoticity only yields: $\Gamma(H^\alpha) = \Gamma(H_f \cdot H^\alpha) = \Gamma(H^\alpha)$.*

Now why do we need looping stability in the general case? Without it, it seems impossible to establish a general looping condition as was established in the Σ_2 case and as we'll do in Proposition 5.3.40, for the operators satisfying the condition of looping stability. By "looping condition", we mean a criterion that let us decide, by looking at at some point the history of a machine (so looking at a given and fixed past of the computation), whether the machine is at this point seen to be looping (and so producing a statement about the future of the computation). In other word, it is a criterion that allows us to decide whether a

machine is looping without having to look at its whole computation through On. To give an idea of what would be possible without this hypothesis and the difficulty it poses, imagine γ a cell-by-cell operator acting as the $\limsup$ *operator with the only difference being that for some gigantic additively closed τ, $\gamma((01)^\tau) = 0$. This operator would be stable and asymptotic. We could also arrange for it to be contraction proof. And with $\tau > \Sigma_{\sup}$, it would be repeating from stage $\mathrm{T}_{\sup} = \Sigma_{\sup}$ onward and up to stage τ where it would magically exit this loop.*

In particular, refining this idea, Theorem 5.5.1 shows that the main theorem of this section, that is that under some conditions on Γ, $\Sigma_\Gamma = \mathrm{T}_\Gamma$, does not hold without the hypothesis of looping stability.

As suggested, the first benefit of the condition of looping stability is that it gives a general looping condition for the operators of the class.

Proposition 5.3.40. *Let Γ be an operator which is looping stable and asymptotic. Then the following looping condition holds: a Γ-machine is looping if and only if there are two ordinals α, β such that stages α, $\alpha + \beta$ and $\alpha + \beta \cdot \omega$ share the same snapshot.*

Proof. In the first direction, suppose that m is looping. Then, after some point, some word H repeats through the whole rest of the computation. This means that after repeating ω time, that is after H^ω appearing in the history, H itself comes next. Taking α the stage at which H begins to repeat and β the length of H yield the wanted ordinal stages.

In the other direction, let H be the history segment that spans between stages α and $\alpha + \beta$. As those stages share the same snapshot, by asymptoticity the computation behaves similarly starting from α or $\alpha+\beta$. So H is found again between $\alpha+\beta$ and $\alpha+\beta\cdot 2$ and so on until stage $\alpha + \beta \cdot \omega$. By our assumption, this stage also has the same snapshot as do stages α and $\alpha+\beta$. So, again by a asymptoticity, the computation behaves similarly starting from this stage. Moreover, writing x the real appearing at those stage, it means that $\Gamma(H^\omega) = x$. Hence, by looping stability, for any limit ordinal δ, $\Gamma(H^\delta) = x$ and, by induction, any stage $\alpha + \beta \cdot \delta$ has the same snapshot and the machine is looping, endlessly repeating the segment H. □

The following corollary will offer a bit more flexibility when using Proposition 5.3.40.

Corollary 5.3.41. *Let Γ be an operator which is looping stable and asymptotic. Then the following looping condition holds: a Γ-machine is looping if and only if there are two ordinals α, β such that α and $\alpha + \beta$ share the same snapshot and such that the snapshot $\alpha + \beta \cdot \omega$ also appears between stages α and $\alpha + \beta$.*

Proof. Let α' be the least stage between stages α and $\alpha+\beta$ at which the snapshot of stage $\alpha + \beta \cdot \omega$ also appears. As α and $\alpha + \beta$ share the same snapshot, by asymptoticity, the computation segment $[\alpha, \alpha + \beta[$ is the same as the computation segment $[\alpha + \beta, \alpha + \beta \cdot 2[$. In particular, the snapshot of stage $\alpha + \beta \cdot \omega$ also appears at some stage $\alpha' + \beta'$ between

stages $\alpha + \beta$ and stages $\alpha + \beta \cdot 2$ and such that $\alpha' + \beta' \cdot \omega = \alpha + \beta \cdot \omega$. And so, α' and β' satisfy the looping condition of Proposition 5.3.40. □

Further, Proposition 5.3.40 justifies the following definition.

Definition 5.3.42. Let Γ be an asymptotic operator that satisfies the looping stability and m a Γ-machine. We say that m is *seen to be looping at some stage* δ if there are two ordinal α and β such that stages α, $\alpha + \beta$ and $\alpha + \beta \cdot \omega$ share the same snapshot and such that $\alpha + \beta \cdot \omega = \delta$.

Proposition 5.3.43. *Let Γ be an operator that satisfy the looping stability as well as being asymptotic and m be a Γ-machine that does not halt. Then at any stage α, if the machine is not seen to be looping before stage $\alpha \cdot \omega^\omega$, there is a snapshot that did not appear strictly before stage α that appears strictly before stage $\alpha \cdot \omega^\omega$.*

Proof. Let m be some machine and α an ordinal such that the machine is not seen to be looping before stage $\alpha \cdot \omega^\omega$. We look at its computation starting from stage α onwards. We consider the snapshot s_0 that appears in m at stage α. Either this snapshot appears for the first time, and we're done. Either this snapshot appears at some earlier stage $\alpha_0 < \alpha$. This means that the snapshot s_0 (at stage α_0) leads to another occurrence of the snapshot s_0 (at stage α). By asymptoticity of Γ, the snapshot s_0 (at stage α) must also lead to an ulterior occurrence of s_0 and so on, for at least ω repetitions. After those ω repetitions, we write s_1 the snapshot of m. Observe first that $s_1 \neq s_0$ as otherwise m would be seen to be looping by Proposition 5.3.40. And even stronger: by Corollary 5.3.41, s_1 does not appear in the whole segment that spans between the two occurrences of s_0 and that repeats itself. Hence, again, if s_1 does not appear for the first time, there is some least stage $\alpha_1 < \alpha_0$ at which the snapshot s_1 occurs.

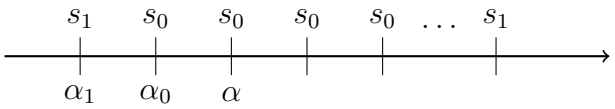


Figure 5.2: The snapshot s_0 repeats ω times and if the snapshot s_1 does not appear for the first time after this repetition it must have appeared before stage α_0.

With the same reasoning, as long as the repetition of the segments between the occurrences of s_i does not yield a new snapshot, it yields a snapshot s_{i+1} that appeared earlier, at some least ordinal stage $\alpha_{i+1} < \alpha_i$ (as, by hypothesis, the machine isn't seen to be looping yet). This yields a decreasing sequence of ordinals (α_i). Consequently, by well-orderdness, there must be some snapshot s_k that appears for the first time at some stage α_k for a finite k.

And how late can this snapshot occur? At worse, the first repeating segment between the occurrences of s_0 has length α. Then the second repeating segment, between the

occurrences of s_1 would have length $\alpha \cdot \omega$. And, more generally, the repeating segment between the occurrences of s_i would have length $\alpha \cdot \omega^i$. This yields the bound

$$\sum_{i<\omega} \alpha \cdot \omega^{i+1} = \alpha \cdot \omega^\omega$$

for the appearance of a new snapshot. □

Proposition 5.3.44. *Let Γ be a suitable operator that satisfy the looping stability as well as being asymptotic. Then there is some countable stage α such that any Γ-machine either stopped or is seen to be looping before stage α.*

Proof. Consider ordinal ω_2. By the downward Löwenheim-skolem theorem, we can find limit ordinals $\alpha_1 < \alpha_2 < \alpha_3$ of cardinality $\aleph_1$ and such that the L_{α_i} are elementary substructures of L_{ω_2}. Moreover we can arrange for α_3 to be strictly greater than $\alpha_2 \cdot \omega$. Further, by a result of Devlin [Dev84, II. 5.5], we can show in L that no new reals appear in the constructible universe after construction stage ω_1. Hence, we have:

$$\begin{aligned} L_{\omega_2} \models \exists \alpha_1 < \alpha_2 < \alpha_3 \; & L_{\alpha_1} \prec L_{\alpha_2} \prec L_{\alpha_3} \\ & \wedge \mathrm{Lim}(\alpha_1) \wedge \mathrm{Lim}(\alpha_2) \wedge \mathrm{Lim}(\alpha_3) \\ & \wedge \alpha_2 \cdot \omega < \alpha_3 \\ & \wedge \text{no new real appear after construction stage } \alpha_1 \end{aligned}$$

Again by the downward Löwenheim-Skolem theorem there exists some countable ordinal β such that $L_\beta \prec L_{\omega_2}$. This yields three limit countable ordinals $\beta_1 < \beta_2 < \beta_3$ that satisfy the previous formula in L_β. We show that any Γ-machine m is seen to be looping before stage β_3.

Let m be a Γ-machine computing from the empty input. First, as Γ is a suitable operator there is some predicate $\varphi(i, m, 0, k)$ that defines in L_α, for any limit α, the value of any cell i of m at limit stage α (the 0 in φ stands for the empty input). As $L_{\beta_1} \prec L_{\beta_2}$ (and as those are limit stages), whether in L_{β_1} or L_{β_2}, the computation of the limit rule for some cell i using φ yields the same value. Hence, the snapshots of m at stages β_1 and β_2 match. For convenience, we let $\beta_0 < \beta_2$ be the least ordinal stage that share the same snapshot as β_2 and we write x_2 the real appearing at those two stages. As seen in Example 5.3.24, by asymptoticity of Γ this gives rise to a repeating pattern where the segment between stages β_0 and β_2 repeats at least ω stages. Let β_2' be the limit stage directly after those ω repetitions of the segment of computation $[\beta_0, \beta_2[$ and x' be the real appearing at this stage. As $\beta_2' \leqslant \beta_2 \cdot \omega$, we also have that $\beta_2' < \beta_3$. This state of affairs is represented in Figure 5.3.

By Corollary 5.3.41, m is seen to be looping if x' appears at a limit stage in the segment of computation $[\beta_1, \beta_2[$ (as it would then produce the same snapshot as the one appearing at stage β_2'.) As x' appears as a snapshot in m at stage β_2', it can be defined

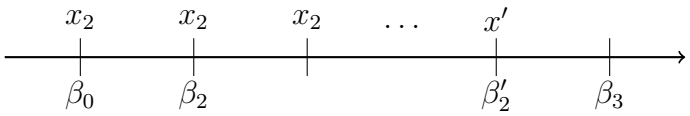


Figure 5.3: The segment that spans between β_0 and β_2 repeats ω times and produces x'.

in $L_{\beta'_2}$. Hence $x' \in L_{\beta'_2+1}$. As $\beta'_2 < \beta_3$, x' is also in L_{β_3}. So, with x_2 being the snapshot that appeared at stage β_2 and with $C^m(\alpha)$ the function that maps some ordinal stage α to the real that describes the snapshot at stage α in the computation of m:

$$L_{\beta_3} \models \exists \beta_2 < \beta'_2 \ \mathrm{Lim}(\beta_2) \wedge \mathrm{Lim}(\beta'_2) \wedge C^m(\beta_2) = x_2 \wedge C^m(\beta'_2) = x'$$

Now, and that is the corner stone of this proof, L_{β_3} sees that no new reals appeared in the constructible universe after stage β_1. This means that x' (as well as x_2) are in L_{β_1} and in L_{β_2} and that, by elementarity, we can reflect the previous sentence down to L_{β_2}. This means that, as depicted in Figure 5.4, in the computation of m up to stage α_2, the real x' appeared at some limit stage δ'_2 such that there is an earlier limit stage $\delta_2 < \delta'_2$ at which the real x_2 appeared. By definition of β_0, it must be less than δ_2 and x' appeared at a limit stage in the segment of computation $[\beta_0, \beta_2[$. Hence, by Corollary 5.3.41, m is seen to be looping before stage β_3.

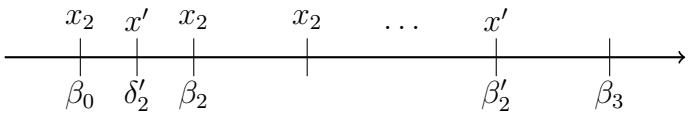


Figure 5.4: By elementarity, there is $\delta'_2 < \beta_2$ such that $C^m(\delta'_2) = x'$

□

5.4 Toward higher-order and many-symbol ITTMs

Theorem 5.4.1. $\Gamma_{\sup}$ *and* $\Gamma_{\inf}$ *are the only simulational operators with two symbols that satisfy the looping stability condition.*

Proof. First, as seen, $\Gamma_{\sup}$ is a suitable and simulational operator. With the same reasoning, so is its symmetric, $\Gamma_{\inf}$. Then, let Γ be a two-symbol suitable and simulational operator. As Γ is cell-by-cell we can write γ the cell restriction of Γ. We now want to show that γ is either $\gamma_{\sup}$ or $\gamma_{\inf}$. So let $h \in {}^{<On}2$ a limit cell history on two symbols. We distinguish two cases

– If there is $s = 0$ or $s = 1$ and some α such that the word s^α is a final segment of h then by asymptoticy $\gamma(h) = \gamma(s^\alpha)$ and by stability $\gamma(h) = s$.

– Otherwise, this means that both 0 and 1 are cofinal in the word h. We show that all such h are mapped to the same symbol. Take h_1 and h_2 two such history who have cofinally 0's and 1's. From h_1 and h_2 we can construct by transfinite induction h'_1 and h'_2 such that h'_1 and h'_2 are stutter-free and $h'_1 \simeq_{ctr} h_1$ and $h'_2 \simeq_{ctr} h_2$. As γ is contraction-proof, $\gamma(h'_1) = \gamma(h_1)$ and $\gamma(h'_2) = \gamma(h_2)$. As h'_1 and h'_2 are stutter-free, they are simply the regular alternation of 0 and 1 : $010101\ldots$ or $101010\ldots$ Hence, there are two ordinals α and β such that $h'_1 = (01)^\alpha$ (or $h'_1 = (10)^\alpha$) and $h'_2 = (01)^\beta$ (or $h'_2 = (10)^\beta$). In any case, by asymptoticity and looping stability (which can be applied cell-by-cell as Γ is cell-by-cell; see Remark 5.3.31.), $\gamma(h'_1) = \gamma((01)^\omega) = \gamma(h'_2)$. and from there $\gamma(h_1) = \gamma(h_2)$. Hence those histories form a single equivalence class w.r.t. γ: either they are all mapped to 1 and $\gamma = \gamma_{\sup}$. Either they are all mapped to 0 and $\gamma = \gamma_{\inf}$.

□

First, observe that this result is really proper to 2-symbol Γ-machines. With 3 symbols or more, there are many intricate ordinal words that can't be simplified by contraction. Hence this theorem shows how, on the contrary to what we are used to in computability, working with only two symbols is actually here a real constraint. This shows the necessity to consider n-symbol machines. This will let us consider more complex rules that will still satisfy basic operator constraints, like being simulational. Still, we will want to build on the previous result established for the 2-symbol operators lim sup and lim inf. To this extent, we will be interested in operators *enhancing* other operators.

Definition 5.4.2. Let Γ_n a n-symbol operator and Γ_k a k-symbol operator with $k < n$. We say that Γ_n *enhances* Γ_k when for any $H \in {}^{<On}({}^\omega k)$:

$$\Gamma_k(H) = \Gamma_n(H)$$

Remark 5.4.3. *Enhancement is a powerful feature as it allows an operator to behave like an operator it enhances by simply sticking to a subset of symbols. In the case of enhancement of the operators $\Gamma_{\sup}$ or $\Gamma_{\inf}$ is it moreover* almost *ubiquitous in the setting of well behaved operators. Indeed, take Γ a n-symbol simulational and looping stable operator with $n > 2$. We further suppose that a limit stages Γ does not produce symbols that were not cofinal in the cell history. That is, writing γ for the associated cell operator, $\gamma(h) = k$ implies that k is cofinal in h. Hence, if, say, only 0 and 1 are cofinal in h, $\gamma(h)$ is either 0 or 1. This behavior, which could be called* strong stability *(observe that it implies stability), is not a consequence of the operator being simulational or looping stable but it is natural enough to be worth considering. What is convenient with this additional hypothesis of strong stability is that the restriction of Γ to a subset of symbols is safely defined–as once the domain is restricted to words on this subset of symbols, the image is restricted automatically restricted to this same subset.*

So we can consider the restriction of Γ *to* 0 *and* 1*. Now it is clear that the operator that we thus obtain is still simulational and looping stable. But it is also a* 2*-symbol operator! So, by Theorem 5.4.1, it is either* $\Gamma_{\sup}$ *and* $\Gamma_{\inf}$*. Which mean, the other way around, that such an operator* Γ *must enhance one of the classical* 2*-symbol operators. In particular, for this operator, a* Γ*-machine can simulate any classical* lim sup *ITTM.*

Still, as much as such an enhancement is convenient we may not want to impose this hypothesis of strong stability to our operators. One principal reason being that even without this hypothesis, a simulational and looping stable operator can still act almost as if it was classical ITTMs in a fairly straightforward and faithful way. We will call this behavior emulation*.*

Definition 5.4.4. Let Γ_n and Γ_k be respectively a n-symbol operator and a k-symbol operator with $k < n$. We say that Γ_n *emulates* Γ_k when for each Γ_k-machine m_k there exists a Γ_n-machine m_n such that:

- at any limit stage the snapshots of m_k and m_n match.
- all snapshots of m_k appearing between two limit stages also appear between the same two limit stages and in the same order in m_n but they may be separated by a finite amount of other snapshots.
- all snapshot of m_n that are not from m_k have a symbol $s \notin k$ written in one of their cell.

With this definition, it is clear that any emulation of a machine will produce a very similar computation. In particular, with the notations of Definition 5.4.4, m_k writes (or e.w.) x if and only if m_n does. While m_n may a.w. more reals, this will be enough for our purpose as we will mostly want to harvest the deciding and writing power of lim sup machines (deciding when a real is a code for an ordinal, writing a code for L_α given a code for α, ...) Proposition 5.4.6 motivates this definition and Proposition 5.4.5 links it with previous definition of enhancement.

Proposition 5.4.5. *If* Γ_n *enhances* Γ_k *then it also emulates it.*

Proof. With the notations of Definition 5.4.4, it is enough for m_n to take m_k seen as a Γ_n-machine. □

Proposition 5.4.6. *Let* Γ *be a* n*-symbol simulational and looping stable operator. Then* Γ *emulates the operator* $\Gamma_{\sup}$*.*

Proof. What may be the main difficulty in emulating a classical 2 symbols ITTM with a simulational and looping stable operator Γ? As suggested in the previous remark, it is the fact that Γ may not be "strongly stable" and so that Γ applied to an history like $(01)^\omega$ may produce, say, 2. But is it really an issue? For a Γ-machine that wants to behave like

a classical ITTM, it could just replace the 2's at a limit stage by 1's–as they come from the history $(01)^\omega$–and continue forward with the simulation. It would work for the first few limit steps but it gets tricky at the ω^{th} limit step. Indeed, at limit stage ω^2, the end segment of the history of a blinking cell would read

$$(2(10)^\omega)^\omega$$

And this may very well yield yet another symbol in a Γ-machine, as neither asymptoticity nor looping stability can be applied in this case.

To circumvent this issue, we use a simple enough trick. Consider the word $w = 012\dots n-1$ made from all letters in n. Let $k \in n$ such that $\Gamma(w^\omega) = k$. Now, writing $w_k = k(k+1)\dots(n-1)01\dots(k-1)$, the shift of w by k letters to the left, w_k^ω is a final segment of w^ω. So, by asymptoticicty, $\Gamma(w_k^\omega) = \Gamma(w^\omega) = k$. And w.l.o.g., as those are just symbols, we can suppose that $k = 1$.

So given m a classical ITTM, we describe the emulating machine m_Γ of Definition 5.4.4. It behaves like m with the only difference that when it writes a 0 over a 1, it write, one after the other in the same cell, 2, 3, …, $n-2$, $n-1$ and finally 0. In other words, the Γ-machine writes a 0 over a 1 by writing w_1, one letter after the other, in the cell. This way, the history $(10)^\omega$ in the classical ITTM we want to simulate corresponds to the history $(w_1)^\omega$ in the Γ-machine. And as wanted, the operator Γ maps this history to 1. So at after the limit, it reads $(w_1)^\omega 1$ which corresponds to $(10)^\omega 1$. Then $(10)^\omega 10$ corresponds to $(w_1)^\omega w_1$. And further, $(10)^{\omega^2}$ corresponds to $(w_1)^{\omega^2}$ to which we can now apply looping stability: it yields again 1 and, by induction, the simulation carries on in a faithful way and respecting the conditions of Definition 5.4.4. □

We will see that a large part of the results for the lim sup machines involving either only writable or only clockable ordinals are true and proved with the exact same proof for n-symbol machines able to emulate the classical 2-symbol machine. The following few results illustrate this.

The first two results also ensure that the writing and clocking constants are well-behaved and meaningful, as they are in the ITTM setting. Note that when possible we include the hypothesis that Γ emulates $\Gamma_{\sup}$ (or indifferently $\Gamma_{\inf}$) rather the stronger hypothesis of looping stability.

Proposition 5.4.7. *Let Γ be a simulational n-symbol operator that emulates the operator $\Gamma_{\sup}$. Then the constants related to the three different kinds of Γ-writable ordinals are distinct. That is: $\lambda_\Gamma < \zeta_\Gamma < \Sigma_\Gamma$*

Proof. As Γ emulates the lim sup operator, this proof works like that of Proposition 4.1.20. This comes from the fact that, for this proof, the specificity of a Γ-machine when compared to a lim sup-machine is hidden in $\mathcal{U}_\Gamma$, which is used instead of $\mathcal{U}_{\limsup}$. $\lambda_\Gamma < \zeta_\Gamma$: At some point, all machines that write an ordinal have written their ordinal. So we consider the

following machine: it simulates $\mathcal{U}_\Gamma$ (which exists as Γ is simulational) and at each step it writes the sum of all ordinals that have been written by any of the machines that halted. This can be done by emulation of a Γ_{sup}-machine. When all writable ordinals have be written, i.e. when their respective machine stopped, the machine we are describing has eventually written their sum (ordered by the code of the machine) which is by definition greater or equal to λ_Γ.

$\zeta_\Gamma < \Sigma_\Gamma$: With the same reasoning, at some point all e.w. ordinal appeared in $\mathcal{U}_\Gamma$. Computing the sum of all ordinals appearing in $\mathcal{U}_\Gamma$ at the same time (we can't distinguish between those which are e.w. and a.w.) will at some point a.w. an ordinal greater or equal to ζ_Γ. □

Proposition 5.4.8. *Let Γ be a simulational n-symbol operator that emulates the operator Γ_{sup}. Then the supremum of any kind of writable is smaller than its respective kind of clockable. That is, $\lambda_\Gamma \leqslant \gamma_\Gamma$, $\zeta_\Gamma \leqslant \eta_\Gamma$, $\Sigma_\Gamma \leqslant \mathrm{T}_\Gamma$.*

Proof. Again, the technique in this proof is akin to that used in the proof of Proposition 4.1.30. Suppose that $\lambda_\Gamma > \gamma_\Gamma$, that is that γ_Γ is writable and consider the following machine toward a contradiction: it writes a code for γ_Γ, then it emulates the Γ_{sup}-machine that counts through an ordinal with this code to count for γ_Γ steps after which is stops. The emulations takes at least γ_Γ steps. So this machine effectively clocks some ordinal greater or equal to γ_Γ, which is a contradiction.

Suppose that $\zeta_\Gamma > \eta_\Gamma$, that is that η_Γ is e.w. and consider the following machine toward a contradiction: it simulates the machine that e.w. a code for η_Γ and, again using the emulation of a lim sup machine, each time a code for an ordinal is written on its output, it counts through this ordinal and then copies the output of this machine to its own output. When the simulated machine stabilizes on a code for η_Γ, the main machine copies it on its output after at least η_Γ steps, hence eventually clocks an ordinal greater or equal to η_Γ, which is again a contradiction.

As for the last inequality, it is proved exactly as in the first part of the proof of Proposition 5.2.1, replacing Σ and T respectively by Σ_Γ and T_Γ. Moreover, albeit slightly trickier, it is the same technique as used with the two previous inequalities. □

Proposition 5.4.9. *Let Γ be a simulational n-symbol operator that emulates the operator Γ_{sup}. Then λ_Γ, ζ_Γ and Σ_Γ are multiplicatively closed.*

Proof. We show this for Σ_Γ. Let α and β two a.w. ordinals. We want to show that $\alpha \cdot \beta$ is a.w. as well. Observe first that the product of two ordinals is computable with a Γ-machine. This comes from the fact that Γ emulates the lim sup operator: there is a lim sup machine that computes this function, and this machine can be emulated by a Γ-machine. Now suppose w.l.o.g. that (a code for) α appears earlier than (a code for) β on one of the tapes of the machines simulated in $\mathcal{U}_\Gamma$ (and $\mathcal{U}_\Gamma$ exists as Γ is simulational.) We consider the following computation: the machine simulate $\mathcal{U}_\Gamma^1$, a copy of $\mathcal{U}_\Gamma$, and at each

step of $\mathcal{U}^1_\Gamma$, writing s^1 for its snapshot, it launches $\mathcal{U}^2_\Gamma$ a fresh copy of $\mathcal{U}^2_\Gamma$ and it simulates this copy until it reaches the snapshot s^1. Meanwhile, it computes the product of any two ordinals appearing in s^1 and in $\mathcal{U}^2_\Gamma$. When $\mathcal{U}^1_\Gamma$ computed far enough so that β appears in s^1, α appeared earlier and it will appear again in the computation of $\mathcal{U}^2_\Gamma$ until it reaches snapshot s^1; hence the ordinal $\alpha \cdot \beta$ will be a.w. by this machine before $\mathcal{U}^2_\Gamma$ reaches the snapshot s^1.

For λ_Γ and ζ_Γ, it relies in the same way on the fact that the product of two ordinals is computable, and it is easier as α and β can be multiplied once e.w. (resp. written.) □

Remark 5.4.10. *While, as illustrated above, many results on ITTMs are readily generalized to* Γ*-machines (provided basics hypothesis on* Γ*, e.g. being simulational), there is one interesting result which does not easily lend itself to generalization. As kindly pointed out by Bruno Durand, this is the case of the Speed-up Lemma of [HL00]. This lemma reads: "if* $\alpha + k$ *is clockable for* $k \in \omega$*, then so is* α*" and its proof is a clever argument which relies on combinatorial aspects of the* lim sup *rule. It is clear that this argument can't be ported as is to* Γ*-machines and unclear whether it can be proved differently for those.*

Thankfully, the consequences of this fact are limited. One reason is that when it comes to the behavior of infinite machines from a macro perspective (as mostly done her where the ordinals we consider are limit ordinals closed under many different kind of operations), this lemma is not needed. It is however useful when conducting a finer analysis of gaps and particularly of their beginning and ends. While I reckon most of the analysis regarding the beginning of gaps (see e.g. [BDL23] and [Wel09]) and their links to admissible ordinals may be extended to Γ*-machines, the absence of the Speed-up Lemma makes it less clear when it comes to the end of gaps. In particular, this leads to the following question: does there exist a simulational operator* Γ *for which a gap ends at a non-limit ordinal stage?*

To specify the relationship between writable reals and the level of the constructible hierarchy, we can encode sets with reals. This is done in a similar fashion as with ordinals. Observe however that, without consequences, it does not yield the exact same encoding for ordinals seen as well-orders and ordinals seen as sets.

Definition 5.4.11 (Encoding of sets)**.** We say that a real x encodes as set a when x describes a transitive relation E on ω such that $(\omega, E) \simeq (TC(\{a\}), \in)$. As with ordinals, a set a is writable (resp. e.w. and a.w.) by a Γ-machine when a code for a is Γ-writable (resp. e.w. and a.w.) by a Γ-machine.

Proposition 5.4.12. *Let* Γ *be a simulational* n*-symbol operator that emulates the operator* Γ_{sup}*. If a set* a *is in* L_{λ_Γ} *(resp. in* L_{ζ_Γ} *and in* L_{Σ_Γ}*) then it is writable (resp. e.w. and a.w.) by a* Γ*-machine.*

Proof. This proposition is the generalization of one implication of Proposition 4.1.32. The following remark has to do with the other implication. If $a \in L_{\Sigma_\Gamma}$, there is an a.w. ordinal

α such that $a \in L_\alpha$. Hence, a code for a is computable from a code for L_α. Now consider the following computation that will a.w. a under the assumption that a is in L_{Σ_Γ}: as Γ is simulational, the machine can enumerate all ordinals below Σ_Γ. For each of those ordinals ν, as Γ emulates the lim sup operator, the machine can write a code for L_ν on its working tape and try to extract the subcode that would be a code for a. Before α appears, what it thus produces is likely gibberish. When α appears, a code for a is accidentally written. This works similarly if a is in L_{ζ_Γ} or L_{λ_Γ}: the machine simply e.w. or writes the right L_α and computes a code for a from it.

□

Remark 5.4.13. *We may be tempted, under the hypothesis that Γ emulates the operator Γ_{sup}, to try and immediatly establish the converse implication of Proposition 5.4.12 (which will be a corollary of Theorem 5.4.14). That is to show, for the first case, that if x is a.w. then $x \in L_{\Sigma_\Gamma}$. It would simply build on the work which was already done for the operator Γ_{sup}. Suppose that x is a.w. by Γ, then as Γ emulates Γ_{sup} a Γ-machine could use this x to a.w. greater and greater ordinals in $\Sigma^x_{\Gamma_{\text{sup}}}$ (which is $\Sigma_{\Gamma_{\text{sup}}}$ relativised to computations with x as input). And as we might think that $x \in L_{\Sigma^x_{\Gamma_{\text{sup}}}}$ (after all, x is x-writable), this would mean,* a fortiori, *that $x \in L_{\Sigma_\Gamma}$. But it happens that $x \in L_{\Sigma^x_{\Gamma_{\text{sup}}}}$ simply does not hold in the general case (the fact that x is x-writable simply says that $x \in L_{\Sigma^x_{\Gamma_{\text{sup}}}}[x]$). Welch first considered in [Wel00a] the (analogous) set*

$$F_0 = \left\{ x \in {}^\omega 2 \mid x \in L_{\lambda^x_{\Gamma_{\text{sup}}}} \right\}$$

which he explains is akin to the set

$$Q = \left\{ x \in {}^\omega 2 \mid x \in L_{\omega_1^{\text{CK},x}} \right\}$$

Both of those sets are thin (i.e. they do not admit non-empty and non-unit closed subsets) which implies that uncountably many reals x's are not in F_0 or Q. A proof of which we give the idea can be found in [Kec75]. Consider F_0 and suppose that it is not thin. That is, it contains a perfect subset P (i.e. closed subset with not isolated points). P is continuously isomorphic to ${}^\omega 2$, so for our purpose we can assume that $P = {}^\omega 2$. We then consider the application $f : x \mapsto \lambda^x$. It induces a pre-well-ordering $\preceq$ on ${}^\omega 2$:

$$x \preceq y \iff \lambda^x \leqslant \lambda^y$$

This pre-well-order has order type ω_1 (the λ^x are unbounded in ω_1) and is Σ^1_1 (under our the assumption that $P \subset F_0$ and modulo the isomorphism, which implies that $x \in \lambda^x$, we have that $\lambda^x \leqslant \lambda^y$ is equivalent to x being y-computable.) Hence, this yields a Lebesgue measurable (as a subset of ${}^\omega 2 \times {}^\omega 2$) pre-well-order of the reals which can be shown with Fubini's theorem to yield a contradiction.

We now state the main theorem of this chapter.

Theorem 5.4.14. *Let Γ be a n-symbol suitable, looping stable and simulational operator. Then $\Sigma_\Gamma = \mathrm{T}_\Gamma$.*

We will prove this result step by step. For all that follows in this subsection, we let Γ be a n-symbol suitable, looping stable and simulational operator. By Theorem 5.4.6, we can suppose w.l.o.g. that it emulates the 2-symbol operator Γ_{sup}.

Proposition 5.4.15. *The supremum of the e.c. ordinals is strictly less than that of the a.c. ordinals. That is, $\eta_\Gamma < \mathrm{T}_\Gamma$.*

Proof. We show that in $\mathcal{U}_\Gamma$ a new real appears at stage η_Γ, that is that $\mathcal{U}_\Gamma$ a.c. stage η_Γ.

For this, observe that once a machine converges, that is once its output stabilizes definitely, it computes through a set of snapshot that never occurred in the computation of this machine before it converged. Indeed, suppose it did earlier, then by asymptoticity the machine would already have converged at this earlier stage. Hence this defines two disjoint sets of snapshot: those before the machine converged and those after, that we can call the converging snapshots. Moreover, as a machine e.c. the stage at which it converges, η_Γ is the first stage at which all converging machines actually converged. Hence it is the first stage at which all the snapshots of those machine belong to their respective set of converging snapshots. This naturally stays true in their simulation done by the universal machine. Hence the real on the working tape of the universal machine at stage η_Γ appeared for the first time. □

Proposition 5.4.16. *If the supremum of the e.c. is smaller or equal to that of the a.w., then the supremum of the a.w. and that of the a.c. match. That is, if $\eta_\Gamma \leqslant \Sigma_\Gamma$ then $\Sigma_\Gamma = \mathrm{T}_\Gamma$.*

Proof. We suppose that $\eta_\Gamma \leqslant \Sigma_\Gamma$. We show that, in this case, in any Γ-machine m, nothing new appears after stage Σ_Γ. This will show that $\Sigma_\Gamma \geqslant \mathrm{T}_\Gamma$ and, by Proposition 5.4.8, that $\Sigma_\Gamma = \mathrm{T}_\Gamma$.

Suppose that there is some Γ-machine m, a real x_m and a least ordinal stage $\alpha_m \geqslant \Sigma_\Gamma$ such that x_m appears for the first time on a tape of m at stage α_m. We consider the following computation: it simulates m and at each step of m, writing x the real written on the relevant tape, it simulates a fresh instance of $\mathcal{U}_\Gamma$. For each ordinal α a.w. by $\mathcal{U}_\Gamma$, its simulate a fresh copy of m for α steps and looks whether x appears in m during those α steps. If x appeared for the first time in m before stage Σ_Γ, this subcomputation will eventullay find an ordinal below Σ_Γ great enough so that x is found in the simulation of m. Actually, x_m is the first real a.w. by m that can't be found by this subcomputation. Effectively, the computation we designed e.w. x_m and e.c. some ordinal greater or equal to α_m. By our assumptions, α_m is itself greater than Σ_Γ and so greater than η_Γ. Hence the machine e.c. an ordinal greater than η_Γ which is a contradiction. □

Now, in virtue of Proposition 5.4.16, to prove Theorem 5.4.14, it suffices to prove that the case $\zeta_\Gamma < \Sigma_\Gamma < \eta_\Gamma < \mathrm{T}_\Gamma$ leads to a contradiction. This case is represented in Figure 5.5. In the rest of the proof, spanning almost until the end of the section, we suppose toward a contradiction that the constants of Γ give rise to this situation.

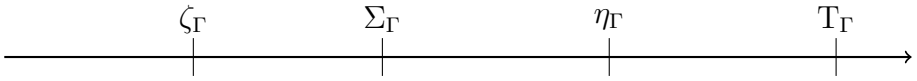


Figure 5.5: Situation of the main constants of the operator Γ if $\Sigma_\Gamma < \eta_\Gamma$.

Proposition 5.4.17. *In the computation of $\mathcal{U}_\Gamma$, there are Σ_Γ distinct snapshots appearing before stage Σ_Γ.*

Proof. As $\Sigma_\Gamma \leqslant \mathrm{T}_\Gamma$ by Proposition 5.4.8, $\mathcal{U}_\Gamma$ is not seen to be looping before stage Σ_Γ. Then for any $\alpha < \Sigma_\Gamma$, a code for $\alpha \cdot \omega^\omega$ is writable from a code for α. Hence $\alpha \cdot \omega^\omega < \Sigma_\Gamma$ and by Proposition 5.3.43 for any $\alpha < \Sigma_\Gamma$ there is a new snapshot (that is distinct from all previous snapshots) that appears after stage α and before stage $\alpha \cdot \omega^\omega$. As Σ_Γ is multiplicatively closed, we can repeat this reasoning with $\alpha \cdot \omega^\omega$ and so on; which yields Σ_Γ distinct snapshots. □

Proposition 5.4.18. *Let x_α be a code for some ordinal $\alpha < \Sigma_\Gamma$. Then the α^{th} distinct snapshot appearing in the computation of $\mathcal{U}_\Gamma$ is x_α-Γ-writable.*

Proof. We show that there is a machine that, given (the code of) some ordinal α as input, can write the α^{th} distinct snapshot appearing in $\mathcal{U}_\Gamma$ if it exists and that never halt if it does not. More precisely it will use α to write down the $\alpha + 1$ first distinct snapshots of $\mathcal{U}_\Gamma$. The machine works as follows: using α given as input to arrange the information (that is it splits the working tape or some part of it into α virtual tape, each of those being able to store a real describing a snapshot), it will inductively look for the β first distinct snapshots for all $\beta < \alpha$. For $\beta = 0$, the first distinct snapshot of $\mathcal{U}_\Gamma$ is its initial snapshot. Then, for any $\beta < \alpha$ such that the machine saved the β first distinct snapshots, that is such that every $\delta < \beta$ correspond to a distinct saved snapshot, the machine simulates $\mathcal{U}_\Gamma$ and, at each stage of this simulation, checks whether this snapshot is one of the snapshots saved as the δ^{th} distinct snapshot for some $\delta < \beta$. If the machine finds one that isn't part of those and consequently is a new snapshot, it saves it as the β^{th} distinct snapshot. If the machine eventually finds α distinct snapshots, then the next distinct snapshot it finds (if it finds it) with the same procedure is the α^{th} snapshot it was looking for and it can stop. If at any point the machine does not find the wanted next distinct snapshots, it never halts. □

Proposition 5.4.19. *In the computation of $\mathcal{U}_\Gamma$, there are at least $\Sigma_\Gamma+1$ distinct snapshots and the Σ_Γ^{th} distinct snapshot (that is the last of the $\Sigma_\Gamma + 1$ first snapshots) is e.w.*

Proof. As, by Proposition 5.4.16, the working hypothesis $\Sigma_\Gamma < \eta_\Gamma$ implies that $\Sigma_\Gamma < \mathrm{T}_\Gamma$, there is some machine computation in which a real appears for the first time after stage Σ_Γ. As this machine is simulated by the universal machine $\mathcal{U}_\Gamma$, there is a least stage $\alpha \geqslant \Sigma_\Gamma$ such that a snapshot s appears for the first time at this stage on the working tape of $\mathcal{U}_\Gamma$. By Proposition 5.4.17 this snapshot is the Σ_Γ^{th} distinct snapshot (that is the last of the $\Sigma_\Gamma + 1$ first snapshots) appearing in the computation of $\mathcal{U}_\Gamma$.

We show that this snapshot s is e.w. Consider this machine: it simulates $\mathcal{U}_\Gamma$ in a simulation that we call $\mathcal{U}_\Gamma^1$. For each snapshot s^1 of $\mathcal{U}_\Gamma^1$, it first copies it on its output and launches a new simulation $\mathcal{U}_\Gamma^2$ to enumerate the ordinals $\alpha \in \Sigma_\Gamma$. More precisely, using $\mathcal{U}_\Gamma^2$, the main machine looks for an ordinal α such that s^1 is the α^{th} distinct snapshot of $\mathcal{U}_\Gamma$. This can be done using the machine described in the proof of Proposition 5.4.18. If such a α is found, the computation goes on with the simulation of $\mathcal{U}_\Gamma^1$. Now, when s^1, the snapshot of $\mathcal{U}_\Gamma^1$, is the β^{th} distinct snapshot of $\mathcal{U}_\Gamma^1$ for $\beta \in \Sigma_\Gamma$, the simulation $\mathcal{U}_\Gamma^2$ obviously yields at some point the correct α, that is such that $\alpha = \beta$. On the other hand, when $\beta \geqslant \Sigma_\Gamma$ the simulation of $\mathcal{U}_\Gamma^2$ never finds an α big enough and the machine never halts. This happens for the first time with the Σ_Γ^{th} snapshot of $\mathcal{U}_\Gamma^1$ and this snapshot is actually e.w., as wanted. □

Proposition 5.4.20. *For the operator Γ, there exists some stage $\mathrm{A} < \eta_\Gamma$ such that for each a.w. ordinals, there is a machine in which this ordinal appears before stage* A.

Proof. We use the previously proven fact that there is some e.w. snapshot that appears after Σ_Γ distinct snapshots appeared in $\mathcal{U}_\Gamma$ to study the time of first appearance of ordinals in any computation. Let s_α be this e.w. snapshot, as defined in the proof of Proposition 5.4.19, and $\alpha \geqslant \Sigma_\Gamma$ that stage at which it appears.

We show using this snapshot s_α that there exists some stage $\mathrm{A} < \eta_\Gamma$ such that all a.w. ordinals appear in $\mathcal{U}_\Gamma$ before this stage A. While this is slightly different than the statement of the proposition, remember that at any limit stage ν, inside of the computation of the universal machine $\mathcal{U}_\Gamma$, all machine also reached stage ν of their simulation. In other words, in its simulations of other machines, the universal machine is never delayed by more than ω steps. Hence, as η_Γ is additively closed, it is equivalent to prove this result for $\mathcal{U}_\Gamma$ or, as initially stated, for all machines individually.

So we consider the following computation of a machine that we call $\mathcal{M}$. It simulates $\mathcal{U}_\Gamma^1$ a copy of $\mathcal{U}_\Gamma$, and at each step of $\mathcal{U}_\Gamma^1$, writing s for its snapshot, $\mathcal{M}$ writes s on its own output tape and launches $\mathcal{U}_\Gamma^2$, a fresh copy of $\mathcal{U}_\Gamma$, in order to enumerate the ordinals below Σ_Γ. For each ordinal β appearing in this fresh copy of $\mathcal{U}_\Gamma$, the main machine launches $\mathcal{U}_\Gamma^3$, a third fresh copy of $\mathcal{U}_\Gamma$ and checks whether the snapshot s of the simulation of $\mathcal{U}_\Gamma^1$ appears in the β first steps of $\mathcal{U}_\Gamma^3$. If it does, the simulation of $\mathcal{U}_\Gamma^1$ carries on and this procedure is repeated with the next snapshots appearing in $\mathcal{U}_\Gamma^1$. If it does not, the simulation of $\mathcal{U}_\Gamma^2$ continues with its enumeration of a.w. ordinals. Hence, when the simulation of $\mathcal{U}_\Gamma^1$ reaches stage α, the snapshot s_α is written on the output tape of the main machine. As

this snapshot does not appear before stage Σ_Γ, it will never be found by the simulation of $\mathcal{U}_\Gamma^3$ that is only conducted through a.w. stages and $\mathcal{M}$ effectively e.w. s_α. Now the interesting part is: which ordinal is then e.c. by this computation?

Let $\mathrm{A} \leqslant \mathrm{T}_\Gamma$ be the least ordinal stage such that for each a.w. ordinal, there is a machine in which it appears before stage A. We claim that if $\mathrm{A} > \eta_\Gamma$, that is if there is some a.w. ordinal σ that does not appear before stage η_Γ in any machine, then the previous computation e.c. some ordinal greater than η_Γ. First, if such an ordinal σ exists, all ordinals greater than σ also appears after stage η_Γ. Then, as σ is a.w., $\sigma < \Sigma_\Gamma$ and by Proposition 5.4.17 there is some snapshot t that appears for the first time in $\mathcal{U}_\Gamma$ after stage σ. Consequently, when t appears in the first simulation of $\mathcal{U}_\Gamma$ in $\mathcal{M}$, $\mathcal{U}_\Gamma^2$, the second simulation of $\mathcal{U}_\Gamma$, won't stop until it finds some ordinal greater than σ. And to do this it will compute for at least η_Γ steps. Consequently, in such a case, when $\mathcal{M}$ converges, it has computed for more than η_Γ steps and it e.c. an ordinal greater than η_Γ; which is a contradiction. Hence $\mathrm{A} \leqslant \eta_\Gamma$. What is left to show is that A can't be equal to η_Γ.

Suppose now that $\mathrm{A} = \eta_\Gamma$: by minimality of A, for any $\alpha < \eta_\Gamma$, there is some a.w. ordinal σ that does not appear in $\mathcal{U}_\Gamma$ before stage α. Moreover, by the definition of η_Γ, for any $\alpha < \eta_\Gamma$, there are machines that converge after stage α. In particular there is some stage $\alpha_\zeta < \eta_\Gamma$ such that ζ_Γ appears for the first time at stage α_ζ and some machine m that converges at stage $\alpha_m > \alpha_\zeta + \omega$, as depicted in figure 5.6. With it, we design the following computation $\mathcal{N}$: using a first simulation of $\mathcal{U}_\Gamma$, that we call $\mathcal{U}_\Gamma^1$, $\mathcal{N}$ enumerate the a.w. ordinals and for every ordinal α it finds, it does the following: it simulates in parallel a copy of m and $\mathcal{U}_\Gamma^2$, a new copy of $\mathcal{U}_\Gamma$, until ordinal α or greater appears in $\mathcal{U}_\Gamma^2$. As α appeared in the simulation $\mathcal{U}_\Gamma^1$, it must appear at some point in $\mathcal{U}_\Gamma^2$. When α has been found in $\mathcal{U}_\Gamma^2$, it is written on the output of $\mathcal{N}$ and then the simulation of m is carried on until there is a proof that m had not yet converged. That is until the output of m is modified for the first time after α has been found. When and if this happens, the main computation carries on with simulation of $\mathcal{U}_\Gamma^1$ and starts this procedure again with the next α' it produces.

Now, as m is a converging machine, it is definitely converging (that is converging through all of On) after some α has been found when, before reaching the point at which this α was found, the parallel computation of m and $\mathcal{U}_\Gamma^2$ was carried for at least α_m steps. For this parallel simulation of $\mathcal{U}_\Gamma^2$ and m to be carried for at least α_m steps, it must have been looking for an ordinal greater than ζ_Γ; as ζ_Γ appeared before, at stage α_ζ. Observe also that ζ_Γ appears at stage α_ζ implies that any ordinal $\beta < \zeta_\Gamma$ appears before stage $\alpha_\zeta + \omega$. Moreover, by the assumption that $\mathrm{A} = \eta_\Gamma$, ordinals great enough (that is greater than ζ_Γ and further, great enough to appear for the first time after stage α_m) are indeed found after stage α_m. Consequently, at this point, m is definitely converging and so is $\mathcal{N}$. And this is only possible because such an ordinal greater than ζ_Γ has been encountered in $\mathcal{U}_\Gamma^1$, that $\mathcal{U}_\Gamma^2$ then found at some computation stage greater than α_m. So $\mathcal{N}$ has eventually written this ordinal greater than ζ_Γ, which is a contradiction.

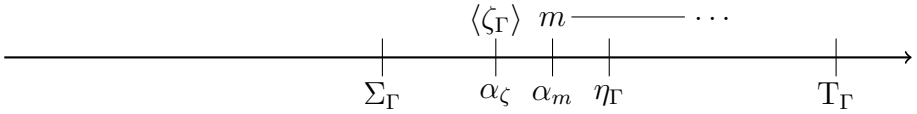


Figure 5.6: Case where $\eta_\Gamma = \mathrm{A}$. A code for ζ appears at stage α_ζ and m converges at an ulterior stage α_m.

□

Now, by Proposition 5.4.20, we obtain the state of affairs described in figure 5.7 for the Γ-machines.

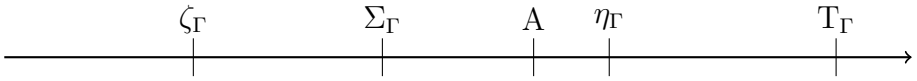


Figure 5.7: Situation of the main constants of the operator Γ. In this case, all a.w. ordinal appeared in some machine before stage A.

Proposition 5.4.21. *In the situation described in Figure 5.7 there is an e.w. snapshot s_Σ and a machine m_Σ such that given as input s_Σ, any set theoretic formula φ and any parameter $p \in L_{\Sigma_\Gamma}$, this machine decides whether $L_{\Sigma_\Gamma} \models \varphi(p)$.*

Proof. Let m be a machine that converges after all a.w. ordinal appeared in $\mathcal{U}_\Gamma$. Such a machine exists as, by Proposition 5.4.20, $A < \eta_\Gamma$.

First we show that we can e.w. the snapshot at which m converges (while the output of m stabilizes at some point, its whole snapshot may not stabilize.) Consider the following computation: it simulates m and at each stage where the output appears to have stabilized (that is the output did not change for at least one step), the main machine saves the snapshot of m. When m starts to converge definitely, its output is never modified anymore and consequently neither does its saved snapshot. We call s_Σ this particular snapshot. What is now interesting is that when this snapshot appears in m, itself simulated in $\mathcal{U}_\Gamma$, all ordinals appeared in $\mathcal{U}_\Gamma$. Indeed, it does not appear earlier as otherwise, by asymptoticity of Γ, the machine would also be converging earlier.

Now we describe the machine m_Σ that decides whether $L_\Sigma \models \varphi(p)$ with φ, p and s_Σ given as input. As s_Σ is given as input, by looking at the simulation of m inside U_Γ, m_Σ knows when m reaches snapshot s_Σ and so, it knows when all ordinal below Σ_Γ appeared. Moreover, as the operator Γ emulates the usual Σ_2-rule, given a code for any ordinal α, a code for L_α is computable from it. With this in mind, m_Σ can use $\mathcal{U}_\Gamma$ to enumerate L_{Σ_Γ} and it will know when all elements of L_{Σ_Γ} appeared in the iteration. Using this, it works inductively as follows: If $\varphi(p)$ is Δ_0, it is absolute and the machine can directly evaluate it. Then, when $\varphi(p) = \exists x \psi(x, p)$ is a Σ_n formula: it starts by simulating $\mathcal{U}_\Gamma$ and it looks at the ordinals that appear in it until the simulation of m inside $\mathcal{U}_\Gamma$ reaches snapshot s_Σ.

When this occurs, all ordinals below Σ_Γ appeared and the machine effectively enumerated all a.w. ordinals. Meanwhile, before s_Σ appears in m, for all ordinal $\alpha < \Sigma_\Gamma$ appearing in $\mathcal{U}_\Gamma$, it enumerates L_α. For each x coding an element of L_α it inductively (on ψ) decides whether $L_{\Sigma_\Gamma} \models \psi(x, p)$. If it does, that is if $L_{\Sigma_\Gamma} \models \psi(x, p)$, then $L_{\Sigma_\Gamma} \models \varphi(p)$ and m_Σ halts and outputs true. If it doesn't, the iteration of L_Σ goes on. If at the end of the iteration, no x such that $L_\sigma \models \psi(x, p)$ has been found, then it outputs false. And the case where φ is Π_n is dealt with by considering $\neg\varphi$. Observe also, as the induction is finite, that there is no difficulties organizing the information from the different recursive calls.

□

Proposition 5.4.22. *L_{Σ_Γ} is an end-elementary extension (e.e.e) of L_{ζ_Γ}. That is, $L_{\zeta_\Gamma} \prec L_{\Sigma_\Gamma}$.*

Proof. We show this by induction. By absoluteness $L_{\zeta_\Gamma} \prec_{\Sigma_0} L_{\Sigma_\Gamma}$. Then let φ be a Σ_n formula such that for some $p \in L_{\zeta_\Gamma}$, $L_{\Sigma_\Gamma} \models \varphi(p)$. We show that $L_{\zeta_\Gamma} \models \varphi(p)$. By Proposition 5.4.12, p is e.w. Writing $\varphi(p) = \exists x \psi(x, p)$, by Proposition 5.4.21, given p, some x and s_Σ, m_Σ can decide whether $L_{\Sigma_\Gamma} \models \psi(x, p)$. As p and s_Σ are e.w., $\widetilde{x}$, the first x appearing in $\mathcal{U}_\Gamma$ such that $L_{\Sigma_\Gamma} \models \psi(x, p)$ is e.w. as well. So we consider $\psi(\widetilde{x}, p)$, true in L_{Σ_Γ}. As both $\widetilde{x}$ and p are e.w., they are in L_{ζ_Γ} and we can apply the induction hypothesis: $L_{\zeta_\Gamma} \models \psi(\widetilde{x}, p)$. And so, $L_{\zeta_\Gamma} \models \varphi(p)$.

Conversely, if $L_{\Sigma_\Gamma} \not\models \varphi(p)$, then $L_{\Sigma_\Gamma} \models \forall x \neg\psi(x, p)$. Hence for all $\widetilde{x} \in L_{\zeta_\Gamma}$, $L_{\Sigma_\Gamma} \models \neg\psi(\widetilde{x}, p)$ with $\neg\psi(\widetilde{x}, p)$ Σ_{n-1}. So by the previous implication, $L_{\zeta_\Gamma} \models \neg\psi(\widetilde{x}, p)$. That is, $L_{\zeta_\Gamma} \models \forall x \neg\psi(x, p)$, i.e. $L_{\zeta_\Gamma} \not\models \varphi(p)$. □

Corollary 5.4.23. *Given as input s_Σ, any set theoretic formula φ and any parameter $p \in L_{\zeta_\Gamma}$, the machine m_Σ decides whether $L_{\zeta_\Gamma} \models \varphi(p)$.*

Proof. By elementarity it is enough to decide whether $L_{\Sigma_\Gamma} \models \varphi(p)$. □

In this proof toward a contradiction we now have obtained fairly strong results. The fact that $L_{\zeta_\Gamma} \prec L_{\Sigma_\Gamma}$ is clearly way too powerful to be established for any Σ_n operator Γ and the fact that we can decide whether $L_{\Sigma_\Gamma} \models \varphi(p)$ seems a direct contradiction of the intuition that L_{Σ_Γ} is a bound on what the Γ-machines may apprehend. Still, reaching a conclusion from there is still somewhat involved. To harvest the previous result, Corollary 5.4.23, we show how we can use a system of notations to virtually work with $L_{\zeta_\Gamma \cdot 2}$ in L_{ζ_Γ}.

Definition 5.4.24 (Notation)**.** We define a system of notation in L_{ζ_Γ}. We write $\top$ for some distinguished tautological sentence. A set a is a notation if and only if:

- Either $a = \langle \top, x, 0, \beta \rangle$ for $x \in L_\beta$ and $\beta \in \zeta_\Gamma$.
- Or $a = \langle \varphi, b_1, \ldots, b_n, \alpha, \beta \rangle$ where φ is a set-theoretic formula (encoded by a set) without the symbol of equality, $\alpha > 0$ and β are ordinals in ζ_Γ and the ordered pair (α, β) is called the *rank* of a, written $\mathrm{rk}(a)$, and $b_1, \ldots, b_n$ are notations of rank strictly less (w.r.t. lexicographical order) than (α, β).

We now use those notations to encode sets in $L_{\zeta_\Gamma^2}$, the level $(\zeta_\Gamma)^2$ of the constructible universe.

Definition 5.4.25 (Notation of a set)**.** For $x \in L_{\zeta_\Gamma^2}$, the *least notation of the set* x, $\overline{x}$, is a notation inductively defined as follows.

- If $x \in L_\beta$ for some least $\beta \in \zeta_\Gamma$:

$$\overline{x} \coloneqq \langle \top, x, 0, \beta \rangle$$

- If $x \in L_{\zeta_\Gamma \cdot \alpha + \beta + 1} - L_{\zeta_\Gamma \cdot \alpha + \beta}$, with $\alpha > 0$, then x was defined over $L_{\zeta_\Gamma \cdot \alpha + \beta}$ by some least formula φ (w.r.t. some fixed order on formulas) and least parameters $p_1, \ldots, p_n \in L_{\zeta_\Gamma \cdot \alpha + \beta}$ (w.r.t. $<_L$). And by extensionality we can w.l.o.g. suppose that the symbol of equality is not used in φ. So we define:

$$\overline{x} \coloneqq \langle \varphi, \overline{p_1}, \ldots, \overline{p_n}, \alpha, \beta \rangle$$

Definition 5.4.26 (Set of a notation)**.** For a notation $a \in L_{\zeta_\Gamma}$, the *set of the notation* a, $\widehat{a}$, is inductively defined as follows:

- If $a = \langle \top, x, 0, \beta \rangle$:

$$\widehat{a} \coloneqq x$$

- If $a = \langle \varphi, b_1, \ldots, b_n, \alpha, \beta \rangle$ with $\alpha > 0$:

$$\widehat{a} \coloneqq \left\{ y \in L_{\zeta \cdot \alpha + \beta} \mid L_{\zeta \cdot \alpha + \beta} \models \varphi(y, \widehat{b_1}, \ldots, \widehat{b_n}) \right\}$$

We draw the attention of the reader to some clear but important features of this system of notations.

Proposition 5.4.27.

- *The operator $x \mapsto \overline{x}$ defined on $L_{\zeta_\Gamma^2}$ is injective, $\widehat{\overline{x}} = x$ and the operator $a \mapsto \widehat{a}$ is surjective onto $L_{\zeta_\Gamma^2}$. That is, every set in $L_{\zeta_\Gamma^2}$ has a notation.*
- *A notation only uses finitely many notations in its definition.*
- *For a notation a of rank (α, β), $\widehat{a} \in L_{\zeta_\Gamma \cdot \alpha + \beta + 1}$.*
- *For $x \in L_{\zeta_\Gamma \cdot \alpha + \beta + 1}$ with $\alpha, \beta < \zeta_\Gamma$, x has a notation of rank less than (α, β).*

Proof.

- By definition, it is clear that $x \mapsto \overline{x}$ is injective and that $\widehat{\overline{x}} = x$. Hence $a \mapsto \widehat{a}$ is surjective onto $L_{\zeta_\Gamma^2}$.

– Observe that the inductive definition of a notation induces a tree in which each node is a notation. In this tree, every node has finitely many children and by well-orderdness every path is finite. Hence the tree itself is finite.

– By induction: notations of rank $(0, \beta)$ clearly yield elements of L_{ζ_Γ}. For a notation $a = \langle \varphi, b_1, \dots, b_n, \alpha, \beta \rangle$ with $\alpha > 0$, by induction $b_1, \dots, b_n$ are notations for some sets $p_1, \dots, p_n$ (i.e. $\widehat{b_i} = p_i$) of rank strictly less than (α, β) and so they are in $L_{\zeta \cdot \alpha + \beta}$. Hence, as a set defined over $L_{\zeta \cdot \alpha + \beta}$, $\widehat{a}$ is in $L_{\zeta \cdot \alpha + \beta + 1}$.

– Simply consider $\overline{x}$, the least notation of x.

□

Proposition 5.4.28. *There exists a Γ-machine such that: given as input s_Σ, a formula φ, two ordinals $\alpha, \beta \in \zeta_\Gamma$ with $\alpha > 0$ and notations $p_1, \dots, p_n$ of rank strictly less than (α, β), it decides whether*

$$L_{\zeta \cdot \alpha + \beta} \models \varphi(\widehat{p_1}, \dots, \widehat{p_n})$$

Proof. We show this by induction on (α, β). We provide a description of a machine $\mathcal{M}$ that uses the inductive definition of the notations to inductively decide whether $L_{\zeta \cdot \alpha + \beta} \models \varphi(\widehat{p_1}, \dots, \widehat{p_n})$. Through this description we will actually describe two recursive subroutines relying on each other. Namely:

- A subroutine that given φ, two ordinals $\alpha, \beta \in \zeta_\Gamma$ with $\alpha > 0$ and notations $p_1, \dots, p_n$, decides whether $L_{\zeta \cdot \alpha + \beta} \models \varphi(\widehat{p_1}, \dots, \widehat{p_n})$.
- A subroutine that given notations a and b, decides whether $\widehat{a} \in \widehat{b}$.

The main difficulty will be the fact that we can't decide whether some set a is a notation. Deciding whether a is of the form $\langle \varphi, b_1, \dots, b_n, \alpha, \beta \rangle$ for some sets $b_1, \dots, b_n$ is fairly straightforward if we forget the condition on α and β being in L_{ζ_Γ}. However deciding this, that is whether $\alpha, \beta \in \zeta_\Gamma$, is clearly not doable with what we have established so far. And even if α and β are known to be in ζ_Γ, taking a set that looks like a notation of lower rank, that is of rank $(\alpha', \beta') <_{lex} (\alpha, \beta)$, clearly does not guarantee with the lexicographical order that β' is in ζ_Γ.

Still, despite this difficulty, let us first suppose that there is some machine $m_{\langle\rangle}$ that, given the code of some set $a \in L_{\Sigma_\Gamma}$, decides whether a is a notation. We now describe $\mathcal{M}$ using $m_{\langle\rangle}$. It works as a triple imbricated induction on (α, β) for the outermost one, the rank of φ for the next one and the rank of its parameters for the innermost one. We begin with the induction on (α, β).

- Base case is $(\alpha, \beta) = (1, 0)$. In this case, by hypothesis, the rank of any p_i is $(0, \beta_i)$ with $\beta_i \in \zeta_\Gamma$. Hence, for any of those, $p_i = \langle \top, x_i, 0, \beta_i \rangle$ with $\widehat{p_i} = x_i$ and $\mathcal{M}$ can extract

a code for x_i given a code for p_i. And by Corollary 5.4.23, using s_Σ which is given as input, it can decide whether $L_\zeta \models \varphi(x_1, \dots, x_n)$.

- Then, when $(\alpha, \beta) >_{lex} (1, 0)$, we proceed by induction on φ.

  ▷ Suppose that φ is Δ_0. Then there is not quantifiers and any atomic formula is of the form $\widehat{p_i} \in \widehat{p_j}$. We show by induction on $\mathrm{rk}(p_i)$ that $\mathcal{M}$ can decide such atomic formulas using the notations p_i and p_j.

  ∗ Suppose first that we can write $p_i = \langle \top, x_i, 0, \beta_i \rangle$. This mean that $x_i \in L_{\zeta_\Gamma}$. Then, either $p_j = \langle \top, x_j, 0, \beta_j \rangle$ and $\mathcal{M}$ can extract codes for x_i and x_j and use them to decide whether $\widehat{p_i} \in \widehat{p_j}$, as it simply means $x_i \in x_j$. Either $p_j = \langle \psi, c_1, \dots, c_m, \alpha_j, \beta_j \rangle$ and $\widehat{p_i} \in \widehat{p_j}$ if and only if:

$$L_{\zeta_\Gamma \cdot \alpha_j + \beta_j} \models \psi(x_i, \widehat{c_1}, \dots, \widehat{c_m})$$

    As $(\alpha_j, \beta_j) <_{lex} (\alpha, \beta)$, taking p_i for a notation of x_i, $\mathcal{M}$ can inductively (by to the induction hypothesis on (α, β)) decide this.

  ∗ Now, if $p_i = \langle \psi_i, b_1, \dots, b_m, \alpha_i, \beta_i \rangle$ and $p_j = \langle \psi_j, c_1, \dots, c_k, \alpha_j, \beta_j \rangle$. By definition, $\widehat{p_i} \in \widehat{p_j}$ if and only if $\widehat{p_i}$ is in $L_{\zeta \cdot \alpha_j + \beta_j}$ and satisfies ψ_j, the formula that defines $\widehat{p_j}$ over $L_\zeta \cdot \alpha_j + \beta_j$. That is, if and only if:

$$\begin{cases} \widehat{p_i} \in L_{\zeta \cdot \alpha_j + \beta_j} \\ L_{\zeta \cdot \alpha_j + \beta_j} \models \psi_j(\widehat{p_i}, \widehat{c_1}, \dots, \widehat{c_k}) \end{cases}$$

    First $\widehat{p_i} \in L_{\zeta \cdot \alpha_j + \beta_j}$ if and only if there exists a notation q_i such that $\mathrm{rk}(q_i) <_{lex} (\alpha_j, \beta_j)$ and with $\widehat{q_i} = \widehat{p_i}$. We can w.l.o.g. suppose that $\mathrm{rk}(p_i) \geqslant_{lex} (\alpha_j, \beta_j)$ as otherwise p_i satisfies the requirement. And under this assumption, given some notation q_i such that $\mathrm{rk}(q_i) <_{lex} (\alpha_j, \beta_j)$, $\widehat{q_i} = \widehat{p_i}$ if and only if for all notation r_i such that $\mathrm{rk}(r_i) < \mathrm{rk}(p_i)$ we have

$$\widehat{r_i} \in \widehat{q_i} \iff \widehat{r_i} \in \widehat{p_i} \tag{5.1}$$

    Hence using m_Σ (to enumerate codes for sets in L_Σ) and $m_{\langle\rangle}$ (to decide which reals encode a notation), $\mathcal{M}$ can enumerate the notations q_i such that $\mathrm{rk}(q_i) <_{lex} (\alpha_j, \beta_j)$ and for each of those q_i enumerate all the r_i such that $\mathrm{rk}(r_i) <_{lex} \mathrm{rk}(p_i)$ and for each of those r_i inductively decide (as $\mathrm{rk}(r_i) <_{lex} \mathrm{rk}(p_i)$) whether the equivalence (5.1) holds; which in turn enables $\mathcal{M}$ to decide whether $\widehat{q_i} = \widehat{p_i}$ for all the q_i it encounters; which finally lets it decide whether $\widehat{p_i} \in L_{\zeta \cdot \alpha_j + \beta_j}$.
    As for the second part, deciding whether $L_{\zeta \cdot \alpha_j + \beta_j} \models \psi_j(\widehat{p_i}, \widehat{c_1}, \dots, \widehat{c_k})$ can

inductively be done since $(\alpha_j, \beta_j) <_{lex} (\alpha, \beta)$.

* Last case, if $p_i = \langle \psi, b_1, \dots, b_m, \alpha_i, \beta_i \rangle$ and $p_j = \langle \top, y, 0, \beta_j \rangle$, then $\widehat{p_i} \in \widehat{p_j}$ if and only if there is a notation $q_i = \langle \top, x, 0, \beta_i' \rangle$ with $\widehat{q_i} = \widehat{p_i}$ and $x \in y$. And this is decidable with the same reasoning as in the previous case, still under the assumption that notations are decidable using $m_{\langle\rangle}$.

Hence, as $\mathcal{M}$ can decide all the atomic formulas in the Δ_0 formula φ, it can decide φ.

▷ Now, suppose that φ is Σ_n. We write $\varphi = \exists x\, \psi(x, \widehat{p_1}, \dots, \widehat{p_n})$. By the results of Proposition 5.4.27, $x \in L_{\zeta \cdot \alpha + \beta}$ if and only if x has a notation of rank strictly less than (α, β). So, $\mathcal{M}$ does the following: it uses again the fact that given the snapshot s_Σ and the machine $m_{\langle\rangle}$ it can enumerate L_Σ (and know when it has exhausted it). $\mathcal{M}$ enumerates L_{Σ_Γ} and for each notation $a \in L_\Sigma$ such that $\mathrm{rk}(a) < (\alpha, \beta)$ it tests whether $L_{\zeta_\Gamma \cdot \alpha + \beta} \models \psi(\widehat{a}, \widehat{p_1}, \dots, \widehat{p_n})$, which it can do by the inductive hypothesis of the induction on the rank of φ. And $L_{\zeta \cdot \alpha + \beta} \models \varphi(\widehat{p_1}, \dots, \widehat{p_n})$ if and only this is true for some notation a with $\mathrm{rk}(a) < (\alpha, \beta)$.

▷ When φ is Π_n, we write $\varphi = \forall x \psi$. As in the previous case, $\mathcal{M}$ enumerates the notation a in L_{Σ_Γ} and inductively decides whether $L_{\zeta_\Gamma \cdot \alpha + \beta} \models \psi(\widehat{a}, \widehat{p_1}, \dots, \widehat{p_n})$. If it fails for some a, $\mathcal{M}$ outputs false. If not (and, using s_Σ, $\mathcal{M}$ knows when it went through all notations in L_{Σ_Γ}), it outputs true. This conclude the description of $\mathcal{M}$ under the assumption that $m_{\langle\rangle}$ was given.

Now, we need to provide a description of $m_{\langle\rangle}$. As we cannot decide whether some set is a notation because it involves decidng whether an ordinal is less than ζ_Γ, we introduce the slightly more general concept of *quasi-notation*. Quasi-notations are defined inductively, as notations, with the only difference that for the case $a = \langle \varphi, b_1, \dots, b_n, \alpha, \beta \rangle$ or for the base case $a = \langle \top, x, 0, \beta' \rangle$ we only require α, β and β' to be in Σ_Γ (instead of ζ_Γ). The main interest of quasi-notations is that it will be fairly easy to decide whether a set is a quasi-notation: in our case, the conditions, as stated in the proposition, regarding the inputs that are given to $\mathcal{M}$ implies that they are all e.w. Consequently, every set that appears in a computation using them is at least a.w. In particular any ordinal appearing is in Σ_Γ. In such a context, deciding whether a set is a quasi-notation is easy enough given the inductive definition of quasi-notations. So, we can consider the machine $m^q_{\langle\rangle}$ that, given some set $a \in L_{\Sigma_\Gamma}$, decides whether a is a quasi-notation. And with it, we can finish the description of $\mathcal{M}$: it simply uses $m^q_{\langle\rangle}$ as if it was the fictitious $m_{\langle\rangle}$. Obviously, for the moment, we don't have any guarantees regarding the behavior of $\mathcal{M}$ computing with $m^q_{\langle\rangle}$ instead of $m_{\langle\rangle}$. We will work our way toward this.

First, another important fact regarding quasi-notations is the following: if a is a quasi-notation and a is e.w., then a is a notation. To see this, it is enough to observe that, by finite induction, all ordinals appearing in the inductive definition of a will be e.w. as

well and so in ζ_Γ. From there the idea is the following: if we manage to prove that the quasi-notations used in the computation of $\mathcal{M}$ (which uses $m^q_{\langle\rangle}$) are e.w., then it means that they are actually notations! And so, despite $\mathcal{M}$ working with $m^q_{\langle\rangle}$ which only decides quasi-notation, it would actually behave as if it was working with $m_{\langle\rangle}$, which decides notations.

So we consider the computation of $\mathcal{M}$ using $m^q_{\langle\rangle}$, with input s_Σ, a formula φ, two ordinals $\alpha, \beta \in \zeta_\Gamma$ with $\alpha > 0$ and notations $p_1, \dots, p_n$ of rank strictly less than (α, β), as stated above. As noted, all those inputs are e.w. Further, by well-orderdness of the lexicographical order on $\Sigma_\Gamma \times \Sigma_\Gamma$, the machine also terminates when working with quasi-notation. So, suppose that $\mathcal{M}$ answers positively, that is it thinks that $L_{\zeta\cdot\alpha+\beta} \models \varphi(\widehat{p_1}, \dots, \widehat{p_n})$ (but this may not be what it actually computed, working with quasi-notations). Suppose also that φ is Σ_n: $\varphi(\widehat{p_1}, \dots, \widehat{p_n}) = \exists x \psi(x, \widehat{p_1}, \dots, \widehat{p_n})$. As $\mathcal{M}$ answered positively, this means that it found a quasi-notation a_x of rank strictly less than (α, β) for which it thinks that "$L_{\zeta_\Gamma\cdot\alpha+\beta} \models \varphi(\widehat{a_x}, \dots, \widehat{p_n})$". As noted earlier, working with the lexicographical order, the rank of a_x being strictly less than (α, β) does not imply that it is in $\zeta_\Gamma \times \zeta_\Gamma$. But the inputs are all e.w. Consequently so is a_x, as it would be easy to ask $\mathcal{M}$ to write this a_x on its output. This, combined with the previous fact, shows that a_x is actually a notation of rank strictly less than (α, β), that is $\widehat{a_x} \in L_{\zeta\cdot\alpha+\beta}$ and $\mathcal{M}$ was correct in taking $\widehat{a_x}$ as a potential witness for φ. This gives the main ingredient to show by induction, again on φ, that $\mathcal{M}$, despite working with quasi-notations, yields the desired result.

- If φ is Δ_0: the only quasi-notations involved are the notations $p_1, \dots, p_n$. However when deciding whether $\widehat{p_i} \in \widehat{p_j}$, the machine will use quasi-notations and may be wrong. That is, it may wrongfully think that $\widehat{p_i} \in \widehat{p_j}$ because it used a quasi-notation q_i that isn't a notation. But if it is the case, as the inputs involved are e.w. we can e.w. the first quasi-notation q_i that $\mathcal{M}$ uses to decide that $\widehat{p_i} \in \widehat{p_j}$, thinking for example that "$\widehat{q_i} = \widehat{p_i}$". But this implies that q_i is a notation. And so that $\mathcal{M}$ was correct in picking this q_i. Then if it thinks that $\widehat{q_i} = \widehat{p_i}$ working with quasi-notations, this is *a fortiori* true with notations.

  And so, inductively, we could reproduce the induction scheme used to define the machine to show that $\mathcal{M}$ is correct for each of this decisions, because (a) when something holds for all quasi-notations, it *a fortiori* does for all notations and (b) when something does not hold for a quasi-notation, we can arrange so that $\mathcal{M}$ e.w. the first quasi-notation appearing in its computation for which the statement does not hold, hence showing that it is actually a notation (the important point in this previous assertion being that, by well-orderdness, at any step of the induction scheme there are only finitely many sets involved–namely the parameters that get created and carried along the inductive way–and that, by induction, all of them are e.w.)

  This shows that $\mathcal{M}$ is correct in every of its subchoices done to decide whether the

formula φ is true and consequently that it is correct when deciding whether φ is true.

- If φ is Σ_n: we write $\varphi(\widehat{p_1}, \dots, \widehat{p_n}) = \exists x \psi(x, \widehat{p_1}, \dots, \widehat{p_n})$. Suppose that $\mathcal{M}$ answers positively. That is it found a quasi-notation a_x such that it answers positively with ψ, a_x and $p_1, \dots, p_n$ as inputs. As seen, this implies that a_x is e.w., so that it is actually a notation and that $\widehat{a_x} \in L_{\zeta_\Gamma \cdot \alpha + \beta}$. By induction, this means that $L_{\zeta_\Gamma \cdot \alpha + \beta} \models \psi(\widehat{a_x}, \widehat{p_1}, \dots, \widehat{p_n})$ and so that $L_{\zeta_\Gamma \cdot \alpha + \beta} \models \varphi(\widehat{p_1}, \dots, \widehat{p_n})$. If it answered negatively, this is done as in the next case, replacing "positively" by "negatively".

- If φ is Π_n, writing $\varphi = \forall x \psi$, the machine answering positively means that for all quasi-notation a_x of rank strictly less than (α, β), it answered positively with inputs ψ, a_x and $\widehat{p_1}, \dots, \widehat{p_n}$. In particular, this means that it answers positively with those inputs for all notation a_x of rank strictly less than (α, β) (as notations are also quasi-notations). By induction, it means that for all notation a_x of rank strictly less than (α, β), $L_{\zeta_\Gamma \cdot \alpha + \beta} \models \psi(\widehat{a_x}, \widehat{p_1}, \dots, \widehat{p_n})$. And by Proposition 5.4.27, this implies that $L_{\zeta_\Gamma \cdot \alpha + \beta} \models \forall x \, \psi(x, \widehat{p_1}, \dots, \widehat{p_n})$. Again, the negative case is done like the Σ_n positive case, replacing "positively" by "negatively".

□

Proposition 5.4.29. *The ordinal ζ_Γ is a gap of reals in the constructible universe of length ζ_Γ^2. That is:*

$$(L_{\zeta_\Gamma^2} - L_{\zeta_\Gamma}) \cap \mathcal{P}(\omega) = \emptyset$$

Proof. Suppose that some new real x is defined at some stage $\gamma \in [\zeta_\Gamma, \zeta_\Gamma^2[$. That is there is a formula φ and parameters $p_1, \dots, p_n \in L_\gamma$ such that:

$$x = \{n \in \omega \mid L_\gamma \models \varphi(n, p_1, \dots, p_n)\}$$

and $x \notin L_\gamma$. As $\gamma < \zeta_\Gamma^2$ it can be written $\zeta_\Gamma \cdot \alpha + \beta$ with both α and β in L_{ζ_Γ}. By Proposition 5.4.27, all p_i have a notation in L_{ζ_Γ}, that is an e.w. notation. Further, for any $n \in \omega$, a code for $\overline{n}$ is easily computable. So we can consider the following machine: it starts by e.w. φ, α, β, $\overline{p_1} \dots, \overline{p_n}$ and s_Σ on some part of its working tape. Once they are written (that is with the usual dovetailling technique of Proposition 5.3.35), for each $n \in \omega$, it computes $\overline{n}$ and uses the parameters it eventually wrote to simulate the machine $\mathcal{M}$ described in Proposition 5.4.28. With it, it decides (and save the result on another part of its working tape) whether $L_{\zeta_\Gamma \cdot \alpha + \beta} \models \varphi(n, p_1, \dots, p_n)$ for all $n \in \omega$. Once done for all n, it can e.w. x on its output, contradicting the fact that x was a new real appearing at stage $\gamma + 1 > \zeta_\Gamma$. □

We can now finish the proof of the main theorem.

Proof of Theorem 5.4.14. We use the fact that ζ_Γ starts a big gap in the constructible universe to show that it is seen to be looping at stage $\zeta_\Gamma \cdot \omega$. It relies on Corollary 5.3.41, in the same way as the proof of Proposition 5.3.44 does.

We show first that given some a.w. ordinal α, it is possible for a machine to eventually decide (that is to e.w. 1 if it is true and 0 otherwise) whether $L_\alpha \prec L_{\Sigma_\Gamma}$. In Proposition 5.4.21 we showed that there is a real s_Σ and a machine m_Σ such that, given s_Σ and some φ and p as input, m_Σ decides whether $L_{\Sigma_\Gamma} \models \varphi(p)$. Hence, consider the following machine $\mathcal{N}$: it e.w. s_Σ, the snapshot of Proposition 5.4.21, and for all $\varphi(p)$ with $p \in L_\alpha$ and such that $L_\alpha \models \varphi(p)$, it uses this snapshot (that is with the usual dovetailing construction presented in proof of Proposition 5.3.35), to eventually decide (once s_Σ has been e.w., m_Σ, as simulated in the computation we are describing, always halt!) whether $L_{\Sigma_\Gamma} \models \varphi(p)$. Once $\mathcal{N}$ iterated through all formulas φ and parameters $p \in L_\alpha$, it eventually decides whether $L_\alpha \prec L_{\Sigma_\Gamma}$.

Now, using $\mathcal{N}$, it is possible to e.w., if there is one, the first ordinal α such that $L_\alpha \prec L_{\Sigma_\Gamma}$. Indeed, as once s_Σ has been eventually written, the simulation of m_Σ inside some machine with s_Σ as input always halts, we can desing a machine that looks for α such that $L_\alpha \prec L_{\Sigma_\Gamma}$ and with the usual dovetailling technique, that restart each time the machine that e.w. s_Σ changes its output. And there is one such α by Proposition 5.4.22. So there is $\alpha < \zeta_\Gamma$ such that L_α is an end-elementary substructure of L_{Σ_Γ}, as otherwise this computation would e.w. ζ_Γ itself. On the other side of ζ_Γ, if there was a bound on those α's, this bound would be e.w. and greater than ζ_Γ. Hence such α's are cofinal in Σ_Γ. This yields, among many others, two ordinals α and α' such that:

$$\begin{cases} \alpha < \zeta_\Gamma < \alpha' \\ L_\alpha \prec L_{\zeta_\Gamma} \prec L_{\alpha'} \end{cases}$$

As the operator Γ is suitable, it is defined by some formula φ_Γ at the different levels of the constructible universe. So, in particular, this chain of e.e.e. means, looking at the computation of $\mathcal{U}_\Gamma$, that stages α, α' and ζ_Γ all share the same snapshot. Hence, for any ν, ν' in $\{\alpha, \alpha', \zeta_\Gamma\}$ with $\nu < \nu'$ and by asymptoticity, the machine will repeat itself ω times between stages ν' and $\nu' \cdot \omega$. However, it does not yet implies that the machine is looping. More precisely, it does not escape this repetition of the segment $[\nu, \nu'[$ and is actually looping if and only if, by Corollary 5.3.41, the snapshot occurring after this repetition is the same as a snapshot occuring between ν and ν'. So we will show that the snapshot appearing at stage $\zeta_\Gamma \cdot \omega$ in the computation of $\mathcal{U}_\Gamma$, that is after some final segment of the computation below ζ_Γ is repeated ω times, was actually part of said segment of computation.

To do this, we consider first s_{ζ_Γ} the snapshot of the computation of $\mathcal{U}_\Gamma$ at stage ζ_Γ. As $L_\alpha \prec L_{\zeta_\Gamma}$, this snapshot also occurs at stage α and s_{ζ_Γ} is definable (as a real) over L_α. Hence, $s_{\zeta_\Gamma} \in L_{\zeta_\Gamma}$. We let $\alpha_0 \leqslant \alpha$ be the least stage at which this snapshot occurs.

Then, we let $s_{\zeta_\Gamma \cdot \omega}$ be the snapshot (again seen as a real) of $\mathcal{U}_\Gamma$ occurring at stage $\zeta_\Gamma \cdot \omega$. At this stage, by asymptoticity, the segment of computation between stages α_0 and ζ_Γ has been repeated ω times. Moreover, $s_{\zeta_\Gamma \cdot \omega}$ is definable over $L_{\zeta_\Gamma \cdot \omega}$ and so it is in $L_{\zeta_\Gamma \cdot \omega + 1}$. But the gap of reals starting at ζ_Γ and of length ζ_Γ^2 spans over $\zeta_\Gamma \cdot \omega + 1$. So $s_{\zeta_\Gamma \cdot \omega}$ is also in L_{ζ_Γ}. Now we consider the following sentence, writing $S(\nu)$ for the snapshot of $\mathcal{U}_\Gamma$ at some stage ν:

$$\exists \nu, \nu' \, (\nu < \nu' \wedge S(\nu) = s_{\zeta_\Gamma} \wedge S(\nu') = s_{\zeta_\Gamma \cdot \omega})$$

It is naturally true in $L_{\alpha'}$ as it witnesses stages α_0 and $\zeta_\Gamma \cdot \omega$ (As ζ_Γ, the ordinal α' is multiplicatively closed). Moreover, this sentence is actually expressible in the language of L_{ζ_Γ} as both s_{ζ_Γ} and $s_{\zeta_\Gamma \cdot \omega}$ are in L_{ζ_Γ}. As such, by elementarity, it can be reflected down to L_{ζ_Γ}. This implies that there is some stage $\nu' < \zeta_\Gamma$ such that its snapshot is $s_{\zeta_\Gamma \cdot \omega}$ and such that it appears after an occurrence of the snapshot s_{ζ_Γ}. As α_0 is the least stage whose snapshot is s_{ζ_Γ}, $\alpha_0 < \nu'$. As wanted, the repeating segment $[\alpha_0, \zeta_\Gamma[$ in the computation of $\mathcal{U}_\Gamma$ yields at stage $\zeta_\Gamma \cdot \omega$ a snapshot that is part of it, namely that appeared at stage ν'. Hence $\mathcal{U}_\Gamma$ is seen to be looping at stage $\zeta_\Gamma \cdot \omega < \Sigma_\Gamma < \mathrm{T}_\Gamma$, which is a contradiction. □

The two other structural equalities are now corollaries of the main theorem.

Corollary 5.4.30. *Let Γ be a n-symbol suitable, looping stable and simulational operator. Then $\zeta_\Gamma = \eta_\Gamma$.*

Proof. By Proposition 5.4.6, Γ emulates, w.l.o.g., the operator $\Gamma_{\sup}$. So we can apply Proposition 5.4.8 which yields $\zeta_\Gamma \leqslant \eta_\Gamma$. Now, suppose that $\zeta_\Gamma < \eta_\Gamma$. This means that there is some machine m and ordinal α such that m converges at stage α with $\zeta_\Gamma < \alpha < \eta_\Gamma$. As $\Sigma_\Gamma = \mathrm{T}_\Gamma$ by Theorem 5.4.14, $\alpha < \Sigma_\Gamma$. Then we can consider the following machine: using $\mathcal{U}_\Gamma$, it enumerates ordinals below Σ_Γ. For each ordinal ν produced by $\mathcal{U}_\Gamma$, the main machine copies it to its output and then simulates ν steps of a fresh simulation of m and, after those ν steps, goes on with the simulation of m until its output changes. If it does, the computation goes on with the simulation of $\mathcal{U}_\Gamma$ and the enumeration of a.w. ordinals. If it doesn't, it actually e.w. the ordinal ν. And the output of the simulation of m does not change if and only if $\nu \geqslant \alpha$. Hence this computation e.w. the first ordinal greater than α to appear, which contradicts the fact that $\zeta_\Gamma < \alpha$. □

Corollary 5.4.31. *Let Γ be a n-symbol suitable and simulational operator that satisfies the looping stability condition. Then $\lambda_\Gamma = \gamma_\Gamma$.*

Proof. Again by Proposition 5.4.6 and Proposition 5.4.8, $\lambda_\Gamma \leqslant \gamma_\Gamma$. Then, if a machine halts after stage λ_Γ, it must still do so before stage Σ_Γ in virtue of Theorem 5.4.14. But then, it is easy to simulate this machine along the a.w. ordinals to look for and write the a.w. ordinal stage at which this machine halts, which proves that every halting stage is actually also writable. □

Corollary 5.4.32. *Let Γ be a simulational n-symbol operator that satisfies the looping stability condition. A set a is in L_{λ_Γ} (resp. in L_{ζ_Γ} and in L_{Σ_Γ}) if and only if it is writable (resp. e.w. and a.w.) by a Γ-machine.*

Proof. The first implication is simply Proposition 5.4.12. We show the converse implication for a.w. reals, the two other cases are similar.

Suppose that the set a is a.w. That is a code for a, which we write $\hat{a}$, appears in the computation of some Γ-machine. As, by Theorem 5.4.14, $\Sigma_\Gamma = \mathrm{T}_\Gamma$, this codes appears in this computation before stage Σ_Γ and so is itself (as a real) in L_{Σ_Γ}.

Then, as with the operator Γ_{sup}, ζ_Γ is an admissible ordinal and so Σ_Γ is a limit of admissible ordinals–as otherwise the last admissible ordinal below Σ_Γ would be e.w. So there is α admissible and smaller than Σ_Γ such that $\hat{a} \in L_\alpha$. As a code for a, $\hat{a}$ describes a transitive relation E on ω such that $(\omega, E) \simeq (TC(\{a\}), \in)$. The isomorphism described by it yields an application f on ω inductively defined as:

$$f(i) = b \longleftrightarrow \forall c \in b\, \exists j E i\, (f(j) = c) \wedge \forall j E i\, \exists c \in b\, (f(j) = c)$$

As L_α is admissible, this function is definable in it by Σ-recursion and for some i_a, $f(i_a) = a$. Also, for some E-least i_0, i_0, $f(i_0) = \emptyset \in L_\alpha$. Hence, by external induction resting on the use of Σ-collection, for all i, $f(i) \in L_\alpha$. In particular, $a \in L_\alpha \subset L_{\Sigma_\Gamma}$, as wanted. □

Remark 5.4.33. *A question remains: for an operator Γ that satisfies the condition of Theorem 5.4.14 is a Γ-machine that does not halt seen to be looping at stage Σ_Γ? For such a machine, as $\Sigma_\Gamma = \mathrm{T}_\Gamma$, the snapshot s appearing at stage Σ_Γ must have appeared at an earlier stage. And the machine will be repeating up to stage $\Sigma_\Gamma \cdot \omega$ some segment $[\alpha, \Sigma_\Gamma[$ of its computation. However, it is not clear whether the snapshot appearing at stage $\Sigma_\Gamma \cdot \omega$ will be s again. It may be some snapshot t occurring in the segment of computation $[\alpha, \Sigma_\Gamma[$, as in Figure 5.8. In this case the machine is seen to be looping at stage $\Sigma_\Gamma \cdot \omega$ by Corollary 5.3.41. But it may also be some u that does not appear in this segment. In which case, by asymptoticty, the segment of computation spanning between this y and stage $\Sigma_\Gamma \cdot \omega$ will repeat itself up to stage $\Sigma_\Gamma \cdot \omega^2$. And then another snapshot appears. Either in the segment that repeated itself, either occuring before the first apparition of y. And we see how this leads us to repeat the reasoning of the proof of Proposition 5.3.43: as no new snapshot appear after stage Σ_Γ, for such an operator Γ, any Γ-machine is seen to be looping before stage $\Sigma_\Gamma \cdot \omega^\omega$. Still, it may be possible to show with Theorem 5.4.14 that, in fact, they are seen to be looping as soon as they reach stage Σ_Γ.*

5.5 A counter-example without the looping condition

By Theorem 5.4.14, for an operator satisfying the looping condition (as well as being suitable and stimulational and enhancing the lim sup machine), we have $\Sigma_\Gamma = \mathrm{T}_\Gamma$. In

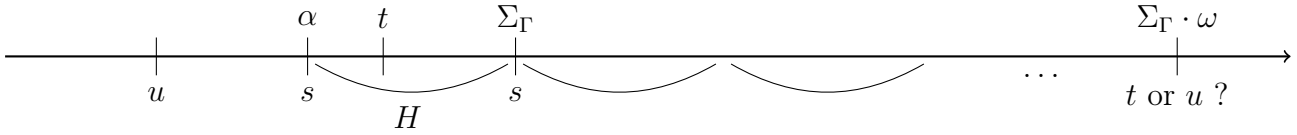


Figure 5.8: When Γ is such that $\Sigma_\Gamma = \mathrm{T}_\Gamma$, when does a Γ-machine loops?

this section, to show that the looping condition is a necessary condition, we construct a suitable and simulational operator that does not satisfy the looping condition and for which $\Sigma_\Gamma < \mathrm{T}_\Gamma$. The definition of this operator, the *tick operator*, is done in Definition 5.5.12 and is fairly straightforward. However, to prove that it indeed forms a counter-example, one convincing way to do it is to introduce another operator, the *jump operator* whose behavior is easier to study and to show that both those operators behave in a similar way.

The intuition behind this counter-example, mentioned in Remark 5.3.39, is fairly simple as well: the machines ruled by the tick operator will be repeating fpr most of their computation and they will only exit those repetitions every tick, that is every τ steps for a very big τ. After they exit their repetition, they compute a bit and "quickly" start to repeat again until the next tick. Hence with a well chosen τ, we will be able to obtain great length of computation, that is greater than τ, with only few different reals written in the computations, and so with small a.w. ordinals, that is smaller than τ.

Theorem 5.5.1. *There exists a suitable and simulational operator Γ that enhances the operator Γ_{sup} and such that $\Sigma_\Gamma < \mathrm{T}_\Gamma$.*

Before defining the jump operator that will help us prove this result for the tick operator we need some preliminary results.

Definition 5.5.2. For $n \in \omega$, the n-symbol lim sup operator is the cell-by-cell operator $\Gamma^n_{\sup}$ defined by the single cell operator $\gamma^n_{\sup}$ itself defined as follows. For $h \in {}^{<On}n$,

$$\gamma^n_{\sup}(h) = k \text{ when } k \in n \text{ is the greatest integer cofinal in } h$$

with the convention that if h is not a limit word, its last letter is considered cofinal in it.

Proposition 5.5.3 (Looping condition for $\Gamma^n_{\sup}$-machine)**.** *A $\Gamma^n_{\sup}$-machine is looping if and only if there are two limit stages $\mu < \nu$ sharing the same snapshot and such that for all cell i, writing s_ν the snapshot at stage ν and $s_\nu[i]$ the value of cell i in s_ν:*

$$\max_{\nu' \in [\mu,\nu[}(C_i(\nu')) = s_\nu[i]$$

Proof. The condition $\max_{\nu' \in [\mu,\nu[}(C_i(\nu')) = s_\nu[i]$ simply comes from the fact that once the segment of computation $[\mu, \nu[$ is repeating, the lim sup of the value of some cell i after it

repeated some limit ordinal amount of time is equal to the maximum value of the cell i in this segment of computation $[\mu, \nu[$. This ensures that the snapshot s_ν also appears after the segment $[\mu, \nu[$ has been repeated a limit ordinal amount of time. □

Proposition 5.5.4. *For $n \in \omega$, the n-symbol* lim sup *operator $\Gamma^n_{\sup}$ is a suitable and simulational operator.*

Proof. This is the same proof as done for the usual 2-symbol lim sup operator. □

Proposition 5.5.5. *For $n > 1$, the n-symbol* lim sup *operator $\Gamma^n_{\sup}$ is as powerful as the usual* lim sup *operator $\Gamma_{\sup}$. That is $\Sigma_{\Gamma^n_{\sup}} = \Sigma_{\Gamma_{\sup}}$ and $\mathrm{T}_{\Gamma^n_{\sup}} = \mathrm{T}_{\Gamma_{\sup}}$*

Proof. For $n > 1$, $\Gamma^n_{\sup}$ enhances $\Gamma_{\sup}$, so we naturally have that $\Sigma_{\Gamma^n_{\sup}} \geqslant \Sigma_{\Gamma_{\sup}}$ and $\mathrm{T}_{\Gamma^n_{\sup}} \geqslant \mathrm{T}_{\Gamma_{\sup}}$. In the other direction, we show that a $\Gamma_{\sup}$-machine can simulate a $\Gamma^n_{\sup}$-machine.

For finite machines, a n-symbol machine can be easily simulated by a 2-symbol machine by representing a cell of the n-symbole machine by $n-1$ cells of the 2-symbol machines. We choose the following encoding: the n-symbol cell that reads $k \in n$ is represented by $n-1$ 2-symbol cells that read, once aggregated, 1^k0^{n-k-1}. Observe that at a limit stage, those are equivalent:

- The n-symbol operator yields k for cell i.
- k is the greatest cofinal value of i in the n-symbol cell up to this limit stage.
- 1^k0^{n-k-1} is the aggregated value of the $n-1$ 2-symbol cell simulating i with the longest 1-prefix and which is cofinal in this limit stage.
- the lim sup operator yields 1^k0^{n-k-1} for the cells simulating i.

Hence, at any limit stage, the n-symbol lim sup operator is correctly simulated by the usual 2-symbol lim sup operator and with it the simulation is faithfully carried out through all the ordinals. □

Definition 5.5.6. For Γ an asymptotic operator and H a limit segment of history, we say that H is a *looping pattern w.r.t.* Γ or a Γ-*looping pattern* when a machine with a history of the form H_0H^ω is looping and, more precisely, looping over the segment of history H.

In the previous definition, we ask Γ to be asymptotic in order to ensure that different initial segments H_0 won't lead to different behaviors after H repeated ω times. So it is equivalent to ask that one machine with this history loops or to ask that any machine with this history loops.

Example 5.5.7. Take $\Gamma^n_{\sup}$, the n-symbol lim sup operator. Then, with Proposition 5.5.3 in mind, H is a looping pattern w.r.t. $\Gamma^n_{\sup}$ if and only if for all cell i:

$$H[0][i] = \max_{\nu<|H|}(H[\nu][i])$$

Hence, if for some Γ^n_{sup}-machine the history reads at some point H_0H^ω, we know that the machine is looping and that the segment of history H will repeat indefinitely.

Definition 5.5.8 (the 3-symbol jump machine)**.** The 3-*symbol jump operator* Γ_{jump} is a 3-symbol lim sup operator with the only difference that its behavior changes after the machine has been repeating itself for some limit ordinal amount of times (or when such repeating stages are cofinal). It is called a "jump" operator as this change of behavior can be seen as a Turing jump escaping the loop.

It can be defined using the lim sup operators Γ_{102} and Γ_{210}. Those are defined using cell-by-cell lim sup operators γ_{102} and γ_{210}. Both are usual 3-symbol lim sup operators working on the alphabet 3 with the only difference being the order of priority of the letters in 3. That is for a limit cell history $h \in {}^{<On}3$,

$$\gamma_{102}(h) = \begin{cases} 1 & \text{if 1's are cofinal in } h \\ 0 & \text{if } \gamma_{102}(h) \neq 1 \text{ and 0's are cofinal in } h \\ 2 & \text{if } \gamma_{102}(h) \neq 1 \text{ and } \gamma_{102}(h) \neq 0 \end{cases}$$

and

$$\gamma_{210}(h) = \begin{cases} 2 & \text{if 2's are cofinal in } h \\ 1 & \text{if } \gamma_{210}(h) \neq 2 \text{ and 1's are cofinal in } h \\ 0 & \text{if } \gamma_{210}(h) \neq 2 \text{ and } \gamma_{210}(h) \neq 1 \end{cases}$$

From there, Γ_{jump} is defined as follow. For a limit history $H \in {}^{<On}({}^{\omega}3)$:

$$\Gamma_{jump}(H) = \begin{cases} \Gamma_{210}(H) & \text{if } H = H_0 \cdot (H_1)^\omega \text{ for a } \Gamma_{102}\text{-looping pattern } H_1 \\ \Gamma_{210}(H) & \text{if } H \text{ is a limit of terms of the form } H_0 \cdot (H_1)^\omega \\ & \text{with the } H_1\text{'s being } \Gamma_{102}\text{-looping patterns} \\ \Gamma_{102}(H) & \text{else} \end{cases}$$

That is, most of the time, a Γ_{jump}-machine behaves like a n-symbol lim sup machine with the order $1 \succ 0 \succ 2$ on its alphabet. However, when it entered a loop, that is when some looping pattern H_1 repeated ω time, it behaves like a lim sup machine with the order $2 \succ 1 \succ 0$ on its alphabet. For consistency it also does so at stages that are limit of stages ruled by the operator Γ_{210} or equivalently limit of repetition of repeating patterns. Those stages corresponds to stages whose historic H below them admits a sequence of prefixes $(H^i_0 \cdot (H^i_1)^\omega)_{i\in\omega}$ where the H^i_1 are Γ_{102}-looping patterns and such that $H = \lim_i H^i_0 \cdot (H^i_1)^\omega$. And after those stages ruled by Γ_{210}, the machine may exit the loop and behave again like a Γ_{102}-machine until it starts to repeat again.

Proposition 5.5.9. *The jump operator Γ_{jump} is a Δ_4-suitable operator. It is moreover stable, asymptotical and contraction-proof but not cell-by-cell.*

We give this proof of this first fact for the sake of completeness, but it is actually enough to be convinced that Γ is a Σ_n-suitable operator for any n.

Proof. As it is built from two lim sup operators, the jump operator is easily seen to be stable and contraction-proof. It is also asymptotical as when the history H can be written $H = H_0(H_1)^\omega$ (with H_0 possibly empty), any final segment of H can be written in the same way (possibly with a different H_0). Same goes when H is a limit of such terms. And it is not cell-by-cell as for a given cell i, the operator needs to consider the whole machine history H rather than only the cell history h_i.

As for it being Δ_4-suitable, the definition provides the skeleton that we can use to define a formula φ_{jump} such that for a machine m that computes with input y and up to limit stage ν, with H_ν being the history of the machine up to stage ν:

$$\Gamma_{jump}(H_\nu)[i] = k \longleftrightarrow L_\nu[y] \models \varphi_{jump}(i, m, y, k)$$

More precisely, we proceed by recursion as in the proof of Proposition 5.3.37. We define by recursion a Σ_1 function η such that for a machine m, an input y and an ordinal ν, $\eta(m, y, \nu) = H_\nu$, the history of m seen as an ordered sequence of snapshots of length ν. For a limit ordinal ν, $\eta(m, y, \nu)$ is simply the limit of the $\eta(m, y, \nu')$ for $\nu' < \nu$. For a successor ordinal ν, $\eta(m, y, \nu+1)$ is easily definable from $\eta(m, y, \nu)$ as the transition from a given snapshot to its successor snapshot is only determined by the code of the machine. And finally, for a limit ordinal ν, to define $\eta(m, y, \nu + 1)$, we need to define the snapshot s_ν of m that appears at limit stage ν with the help of the history below this limit stage, that is of $\eta(m, y, \nu)$. We claim that s_ν is Σ_1-definable from ν and $\eta(m, y, \nu)$, that is from the history of m below ν. Indeed, first it is easy to check whether there exists H_0 and H_1 such that $H_\nu = H_0(H_1)^\omega$. For the second case, H_ν being a limit of initial segments of the form $H_0(H_1)^\omega$ simply means that for all $\beta < \nu$, there is $\beta' \in [\beta, \nu[$, H_0 and H_1 such that the initial segment of H_ν of length β' is equal to $H_0(H_1)^\omega$. Eventually, checking whether some H_1 is a Γ_{102}-looping pattern is straightforward and done in Example 5.5.7. This makes it possible to check whether one of the two first cases of the definition of Γ_{jump} applies, in which case s_ν is computed using the formula induced by the operator Γ_{102} as in the proof of Proposition 5.3.37; else s_ν is computed using the formula induced by the operator Γ_{210}.

Eventually, in $L_\nu[y]$, we can define the formula φ_{jump}. First we write $\pi(H)$ for the Δ_0 predicate that is true when H is a Γ_{102}-looping pattern. Then, we let $\psi(i, m, y)$ be the formula defined with η and π and such that $L_\nu[y] \models \psi(i, m, y)$ if and only if H_ν, the history of m below stage ν, can be written $H_\nu = H_0 \cdot (H_1)^\omega$ with H_1 a Γ_{102}-looping pattern. This is true if and only if there is H_0 and H_1 such that for all $\nu' < \nu$, the initial segment $H_{\nu'}$ of H_ν is an initial segment of $H_0 \cdot (H_1)^\omega$. So ψ is a Σ_3 formula.

Then, as in the proof of Proposition 5.3.37, the lim sup predicate yields Σ_2 formulas φ_{102} and φ_{210} using the history function η. From those, we can define φ_{jump}.

$$\begin{aligned}\varphi_{jump}(i,m,y,k) = &\psi(i,m,y) \wedge \varphi_{210}(i,m,y,k)\\ &\vee [\forall\beta\exists\alpha > \beta\, L_\alpha \models \psi(i,m,y) \wedge \varphi_{210}(i,m,y,k)]\\ &\vee [\neg\psi(i,m,y) \wedge \neg(\forall\beta\exists\alpha > \beta\, L_\alpha \models \psi(i,m,y)) \wedge \varphi_{102}(i,m,y,k)]\end{aligned}$$

As the formula ψ is Σ_3, φ_{jump} is Δ_4 and the jump operator is a Δ_4-suitable operator. □

Proposition 5.5.10. *A Γ_{jump}-machine is looping if and only if there are two limit stages $\mu < \nu$, both ruled by the operator Γ_{210}, that is both following the repetition of a Γ_{102}-looping pattern or a limit of thereof, sharing the same snapshot and such that writing s_ν for the snapshot at stage ν and writing* $\max^{210}$ *for the* max *operator with the order $2 \succ 1 \succ 0$, we have:*

$$\max^{210}_{\nu' \in [\mu,\nu[}(C_i(\nu')) = s_\nu[i] \tag{5.2}$$

In this case, the machine is seen to be looping *at stage ν.*

Proof. Suppose that a Γ_{jump}-machine is looping and let H be the shortest repeating pattern and α be the stage at which it starts looping. Let H_0 be the history up to stage α. Then the history reads: $H_0HHH\ldots$. In particular, when H has repeated ω times, it reads H_0H^ω. As it is looping, $\Gamma_{jump}(H_0H^\omega) = H[0]$. Let μ be the stage below which the history H_0H^ω spans. We show that stage μ is ruled by the operator Γ_{210}, that is that H^ω is a Γ_{102}-looping pattern or limit of such looping patterns. Suppose that H^ω isn't. Then it is easy to see that for any limit α, H^α is neither a Γ_{102}-looping pattern nor a limit of looping patterns. Bu then it means that after this point, Γ_{jump} always acts as the operator Γ_{102}. But then H^ω satisfies Proposition 5.5.3 and H^ω is a Γ_{102}-looping pattern, which contradict our first assumption. Hence, and by definition of the jump operator, $\Gamma_{jump}(H_0H^\omega) = \Gamma_{210}(H_0H^\omega)$, i.e. the stage μ below which the historyH_0H^ω spans is ruled by the Γ_{210}-operator. Then, let ν be the stage corresponding to the history $H_0H^\omega H^\omega$. So the segment of history delimited by μ and ν is H^ω itself. With the same reasoning, stage ν is also ruled by the Γ_{210}-operator and $\Gamma_{jump}(H_0H^\omega H^\omega) = H[0] = \Gamma_{jump}(H_0H^\omega)$, which means that the snapshot at stage μ and ν match. Eventually, the fact that the machine is repeating implies that $\Gamma_{jump}((H^\omega)^\omega) = H[0]$. But also $\Gamma_{jump}((H^\omega)^\omega) = \Gamma_{210}((H^\omega)^\omega)$ and this ensures that condition (5.2) on the repeating segment (here H^ω), is satisfied as well.

Conversely, suppose that are given μ and ν satisfying the hypothesis. Let H_0 be the segment of history that spans between stages 0 and μ and H the segment that spans between stages μ and ν. As the snapshots at stages μ and ν match and Γ_{jump} is asymptotical, the history reads H again after H_0H appeared. And this goes on for at least ω repetitions of H. As ν is ruled by the operator Γ_{210}, so are every stages below which H_0H^k spans for a natural number k. Hence, as a limit of stage limit of stages ruled by Γ_{210}, the

stage below which spans H_0H^ω is also ruled by Γ_{210}. Hence $\Gamma_{jump}(H_0H^\omega) = \Gamma_{210}(H_0H^\omega)$. And $\Gamma_{210}(H_0H^\omega) = \Gamma_{210}(H_0H)$ by hypothesis (5.2). And $\Gamma_{210}(H_0H) = \Gamma_{210}(H_0) = H[0]$ by the fact that μ and ν share the same snapshot and are both ruled by Γ_{210}. Hence the stage corresponding to H_0H^ω also shares the same snapshot and is ruled by Γ_{210}. Further, carrying this reasoning on by transfinite induction, shows that the machine is looping.

□

Proposition 5.5.11. *Let m be a Γ_{jump}-machine that does not halt and let (α, β, δ) the lexicographicaly least triple of additively closed ordinals such that $\alpha < \beta < \delta$ and*

$$L_\alpha \prec_{\Sigma_4} L_\beta \prec_{\Sigma_4} L_\delta$$

Then m is seen to be looping at the latest at stage $\delta \cdot \omega$.

Proof. By the e.e.e. hypothesis, the snapshots at stage α, β and δ match. Hence, by asymptoticty, the segment of history $[\alpha, \beta[$ repeats ω times up to stage $\beta \cdot \omega$. If at stage β the operator Γ_{jump} acts as Γ_{210} then it does so at any stage $\beta \cdot k$ and also at stage $\beta \cdot \omega$, as a limit of stages ruled by Γ_{210}. Else, suppose that at stage β the operator Γ_{jump} acts as Γ_{102}. In this case, we can show that H_β, the segment of history that spans between stages α and β is a Γ_{102}-looping pattern, which contradict this supposition. This comes from the fact that by elementarity, for any cell i, any symbol appearing in i between stages α and β also appears in i cofinally in stage α, and so that the value of cell i at stages α and β is at least greater (w.r.t. $\preceq_{210}$) that the maximum of the values appearing in cell i between those stages. In definitive, $\beta \cdot \omega$ must be ruled by the operator Γ_{210}

We write $s_{\beta \cdot \omega}$ for the snapshot that appears at this stage. It may be different from s_β. However, $s_\delta = s_\beta$ and, again by asymptoticity, the segment of history $[\beta, \delta[$ starts repeating from stage δ and at least up to stage $\delta \cdot \omega$. This is depicted in Figure 5.9. With the same justification, stage $\delta \cdot \omega$ is ruled by the operator Γ_{210}.

We claim that stages $\beta \cdot \omega$ and $\delta \cdot \omega$ satisfy the looping condition of Proposition 5.5.10. First, as shown, at both those stages, operator Γ_{jump} acts like Γ_{210}. Then, writing H_δ for the history that spans between stages β and δ, observe that $H_\beta \sqsubset H_\delta$. Hence, the repetition of H_δ encloses that of H_β. This implies that for any cell i:

$$\Gamma(H_\beta^\omega)[i] \preceq_{210} \Gamma(H_\delta^\omega)[i]$$

The inequality may be strict for some cell i if and only if a symbol greater (w.r.t. $\preceq_{210}$) than $\Gamma(H_\beta \cdot \omega)[i]$ appears between stages $\beta \cdot \omega$ and δ in the cell i. We show that this is not possible. To do this, we show that:

$$\max^{210}_{\nu \in [\alpha, \beta[} (C_i(\nu)) \succeq_{210} \max^{210}_{\nu \in [\beta \cdot \omega, \delta[} (C_i(\nu))$$

Suppose that this is not the case. This means that there is some symbol $s \in 3$ appearing

in cell i between stages $\beta \cdot \omega$ and δ and greater than any symbol appearing in i between stages α and β. The second half of the affirmation translates to:

$$L_\beta \models \forall \nu > \alpha \, C_i(\nu) \prec_{210} s$$

But reflecting this affirmation up to L_δ contradicts the fact that s appeared between stages $\beta \cdot \omega$ and δ. This shows that stages $\beta \cdot \omega$ and $\delta \cdot \omega$ share the same snapshot and moreover that the condition (5.2) of Proposition 5.5.10 holds, which ends the proof.

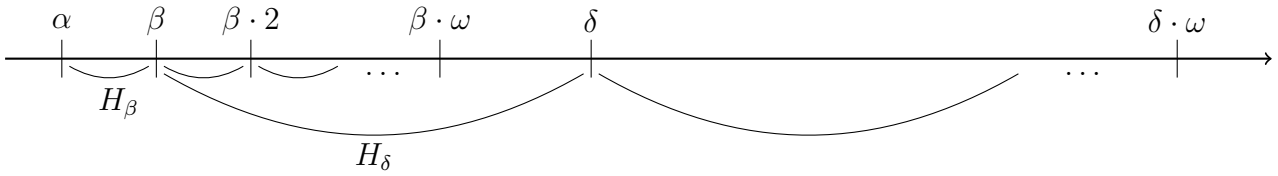


Figure 5.9: Computation of a Γ_{jump}-machine between stages α and $\delta \cdot \omega$.

□

We now define the τ-ticking operator which will be, in definitive, the operator that interests us to prove Theorem 5.5.1. However, the operator Γ_{jump} will help us define the right τ-tick operator (which depends of an ordinal τ) as well as help us study its behavior.

Definition 5.5.12 (the 3-symbol tick machine)**.** For τ a limit ordinal, the *τ-ticking operator* Γ_τ is a 3-symbol lim sup operator with the only particularity that its behavior is altered at limit stages that are multiple of τ. It is called a "ticking" operator as this change of behavior can be seen as a regular tick. Γ_τ-machines will naturally be able to harvest this tick and, at least, to know when their current stage is a multiple of τ.

Γ_τ can be defined as a cell-by-cell operator γ_τ defined using γ_{102} and γ_{210}; themselves defined as in Definition 5.5.8. For a limit history $h \in {}^{<On}3$:

$$\gamma_\tau(h) = \begin{cases} \gamma_{210}(h) & \text{if } |h| = \tau \cdot \alpha \text{ for some ordinal } \alpha \\ \gamma_{102}(h) & \text{else} \end{cases}$$

That is before stage τ and more generally at any stage that is not a multiple of τ, a Γ_τ-machine behaves like a Γ^3_{sup} machine with the order $1 \succ 0 \succ 2$ on its alphabet. And at stages $\tau \cdot \alpha$, a Γ_τ-machine behaves like a Γ^3_{sup} machine with the order $2 \succ 1 \succ 0$ on its alphabet.

Definition 5.5.13 (uniformly characterizable ordinals)**.** An ordinal α is *uniformly characterizable* with respect to the constructible hierarchy if there is a sentence ψ_α such that

for all real y:

$$\begin{cases} L_\alpha[y] \models \psi_\alpha \\ \forall \beta < \alpha \, L_\beta[y] \not\models \psi_a \end{cases}$$

It is Σ_n-*uniformly characterizable* when ψ_α is Σ_n.

Proposition 5.5.14. *There exists an additively closed Π_2-uniformly characterizable ordinal χ strictly greater than Σ_{jump} and T_{jump}, which are respectively the supremum of the ordinals a.w. by a Γ_{jump}-machine and the supremum of the ordinals a.c. by a Γ_{jump}-machine.*

Proof. As Γ_{jump} is not cell-by-cell, it is not simulational and we can't apply Proposition 5.4.8. However, we know that every a.w. real is a.w. before stage T_{jump} and that $\mathrm{T}_{jump} < \delta$ with δ defined in Proposition 5.5.11. So, for any Γ_{jump}-a.w. ordinal α, a code for α is in L_δ and α is in L_{δ^+} where δ^+ is the least admissible ordinal greater than δ. Hence, we also know that $\Sigma_{jump} \leqslant \delta^+$.

Now let χ be the least ordinal that witnesses the existence of α, β, δ and δ^+ such that the following is true in L_χ:

$$\begin{cases} L_\alpha \prec_{\Sigma_4} L_\beta \prec_{\Sigma_4} L_\delta \\ \delta^+ \geqslant \delta \\ L_{\delta^+} \models \mathrm{KP} \\ \forall \mu, \nu \, \exists \xi \, (\xi = \alpha + \beta) \end{cases}$$

This description yields a uniform (as it only quantifies over ordinals or set of the form L_α) characterization of χ strictly greater than Σ_{jump} and T_{jump}. (And note that because of Foundation, KP is not finitely axiomatizable but this is not an issue when dealing with structures that already satisfy Foundation.) □

Proposition 5.5.15. *Given a Σ_n-uniformly characterizable ordinal α, there exists a Σ_m sentence M_α with $m = \max(n, 3)$ and such that for all ordinal ν:*

$$\exists \mu > 0 \, (\nu = \alpha \cdot \mu) \longleftrightarrow L_\nu[y] \models M_\alpha$$

Proof. Let ψ_α as defined in Definition 5.5.13 and consider the following formula.

$$\begin{aligned} M_\alpha = (\psi_\alpha \wedge \forall \beta \, L_\beta \not\models \psi_\alpha) \vee \exists \alpha_0 \, [L_{\alpha_0} \models \psi_\alpha \wedge \forall \beta < \alpha_0 \; L_\beta \not\models \psi_\alpha \\ \wedge \forall \mu \, (\exists \gamma \, (\gamma = \alpha_0 \cdot \mu) \implies \forall \delta < \alpha_0 \; \exists \gamma \, (\gamma = \alpha_0 \cdot \mu + \delta))] \end{aligned}$$

Then $L_\nu \models M_\alpha$ if and only if: either $\nu = \alpha$, in which case $L_\nu \models \psi_\alpha$ and ν is the least such, either $\nu > \alpha$ and for all $\alpha \cdot \mu < \nu$, we also have $\alpha \cdot \mu + \delta < \nu$ for any $\delta < \alpha$. Which is

equivalent to saying that ν is a multiple of α. Having in mind that, given β, the predicate $L_\beta \models \psi_\alpha$ is Σ_1, it is clear that M_α is Σ_m with $m = \max(n, 3)$. □

Proposition 5.5.16. *For τ an additively closed and Σ_3-characterizable limit ordinal, the τ-tick operator Γ^τ_{tick} is a Δ_4-suitable and simulational operator. If τ is greater than Σ_{sup} this operator does not satisfy the looping stability condition.*

Proof. As it is built from two lim sup operators, the τ-tick operator is easily seen to be stable, cell-by-cell and contraction-proof for any limit ordinal τ. It is only asymptotical when τ is additively closed as otherwise, taking a final segment of an history could displace the position of the next tick, that is of the next stage multiple of τ.

Then, observe that it is easy for a machine to detect the "ticking" stages (i.e. stages multiple of τ): it can simply have a cell whose value regularly alternates between 1 and 2 and it will read 2 at a limit stage if and only if this stage is a ticking stage. From there, if $\tau > \Sigma_{\text{sup}}$, we can design a machine that waits for stage τ to exit a repetition that repeated for more than ω times, which falsifies the looping stability condition.

As for it being Δ_4-suitable, the definition again provides the skeleton that we can use to define a formula φ_{tick} such that for some machine m that compute up from input y and up to limit stage ν, with H_ν being the history of the machine up to stage ν:

$$\Gamma_{tick}(H_\nu)[i] = 0 \longleftrightarrow L_\nu[y] \models \varphi_{tick}(i, m, y)$$

We define $\eta(m, y, \nu)$ as usual. The only part that changes is to define $\eta(m, y, \nu + 1)$ when ν is limit. To define s_ν, it's necessary to decide whether ν is a multiple of τ. As τ is uniformly characterizable, by Proposition 5.5.15, there is some formula M_τ such that ν being a multiple of τ is equivalent to $L_\nu[y] \models M_\tau$. Hence, if $L_\nu \models M_\tau$, then s_ν, the last snapshot of $\eta(m, y, \nu+1)$, is defined using the formula induced by the operator γ_{102}. Else s_ν is defined using the other formula induced by γ_{210}.

From there, we can define the formula φ_{tick}. As in the proof of Proposition 5.3.37, the lim sup predicate yields the Σ_2 formulas φ_{102} and φ_{210}. In turn, this yields:

$$\begin{aligned}\varphi_{tick}(i, m, y) := \quad & M_\tau \wedge \varphi_{102}(i, m, y) \\ & \vee \neg M_\tau \wedge \varphi_{210}(i, m, y)\end{aligned}$$

By proposition 5.5.15, M_τ is Σ_3 and so φ_{tick} is Δ_4. □

Proof of Theorem 5.5.1. We claim that for a great enough τ, any τ-tick machine will behave like the jump machine having the same code with the only difference that it will be considerably "slower" as it will often be repeating for considerable amount of time (namely τ steps) before exiting the repetition and acting again like it jump counterpart. Hence, it will still produce the same a.w. ordinals. More precisely, for $\tau = \chi$, the admissible characterizable ordinal defined in Proposition 5.5.14 such that both $\tau > \Sigma_{jump}$

and $\tau > \mathrm{T}_{jump}$, we will show that we have $\mathrm{T}_\tau > \tau$ and $\Sigma_{jump} = \Sigma_\tau$. This implies that:

$$\Sigma_\tau = \Sigma_{jump} < \tau < \mathrm{T}_\tau$$

which proves the theorem. We now need to show that both $\Sigma_{jump} = \Sigma_\tau$ and $\mathrm{T}_\tau > \tau$ hold. Observe that the second equality is immediate as τ is clearly clockable by a τ-tick machine. For the first one, we make the following claim.

Claim 5.5.16.1. *Let m be a machine code, m_{jump} the Γ_{jump}-machine with the code m and ruled by the jump operator and m_τ the Γ_τ-machine with the code m and ruled by the the τ-tick operator. We write (α_ν) for the limit stages of m_{jump} that are ruled by Γ_{210} (that is stages where some looping pattern H_1 has been repeating for ω limit times, or limit of those) and (β_ν) the limit stages of m_τ that also are ruled by Γ_{210} (that is stages multiple of τ). Then, we claim that:*

- *For any ordinal ν, the snapshots at stage α_ν in m_{jump} and β_ν in m_τ match.*
- *For any ordinal ν, we can write $H_0 \cdot H_1^\omega$ for the history of m_{jump} up to stage $\alpha_{\nu+1}$ and the history of m_τ up to stage $\beta_{\nu+1}$ reads $H_0 \cdot H_1^\tau$.*

Proof. We show this by induction. Consider stages α_0 and β_0. By definition, before those stages, respectively in machines m_{jump} and m_τ, the limit operators behaved simply as Γ_{102}. Hence, both those machines started repeating at the latest from stage Σ_{sup} onward. That is, the history of m_{jump} below α_0 reads $H_0 H_1^\omega$ where H_1 is some history of the machine below Σ_{sup} and at most the history between stages ζ_{sup} and Σ_{sup} while the history of m_τ below β_0 reads (by closedness of τ and as $\tau > \Sigma_{\mathrm{sup}}$) $H_0 H_1^\tau$. Hence, as stages α_0 and β_0 are both ruled by the operator Γ_{210} and as this operator is asymptotic and satisfies the looping stability, both stages share the same snapshot.

Then, we consider successor stages $\alpha_{\nu+1}$ and $\beta_{\nu+1}$ for some successor ordinal $\nu + 1$. By induction hypothesis, α_ν and β_ν share the same snapshot. By asymptoticity, the computation of both machines match while they are both ruled by the operator Γ_{102}, that is, respectively, until stages $\alpha_{\nu+1}$ and $\beta_{\nu+1}$. And, for the first time after stage α_ν, the machine m_{jump} produces an history of the form $H_0 H_1^\omega$ for H_1 a looping pattern at stage $\alpha_{\nu+1}$. Hence the history below stage $\alpha_{\nu+1}$ can be written $H_0 H_1^\omega$. As $\tau > \mathrm{T}_{jump}$, τ is also greater than the length of the history $H_0 \cdot H_1^\omega$ and the machine m_τ is repeating H_1 up to stage $\beta_{\nu+1}$, the history before this stage reads $H_0 \cdot H_1^\tau$. Again, by the lim sup definition of operator Γ_{210}, this implies that m_{jump} and m_τ share the same snapshot at respectively stages $\alpha_{\nu+1}$ and $\beta_{\nu+1}$.

Eventually, for some limit ν: Consider the sequence $(\alpha_{\nu'})_{\nu'<\nu}$ and the ordinal $\bigcup_{\nu'<\nu} \alpha_{\nu'}$. First, as the jump operator also acts like the Γ_{210} operator at stages that are limit of

repeating histories (i.e. limit of histories of the form $H_0 \cdot H_1^\alpha$), we have:

$$\alpha_\nu = \bigcup_{\nu' < \nu} \alpha_{\nu'}$$

On the other hand, this naturally holds for β_ν, as a limit of multiple of τ:

$$\beta_\nu = \bigcup_{\nu' < \nu} \beta_{\nu'}$$

Moreover, by induction hypothesis, we can control the history before stages α_ν and β_ν. Namely, the history before stage β_ν is the same as that before α_ν with the only difference that between some stages $\beta_{\nu'}$ and $\beta_{\nu'+1}$, it reads $H_0^{\nu'} \cdot (H_1^{\nu'})^\tau$ instead of $H_0^{\nu'} \cdot (H_1^{\nu'})^\omega$ for some specific $H_0^{\nu'}$ and $H_1^{\nu'}$. Hence, by the definition of the lim sup rule, stages α_ν and β_ν share the same snapshot and this proves the claim. □

Finally, from thisprevious claim, it is clear that the machines m_{jump} and m_τ sharing the same machine code accidentally write the same ordinals. And with it, more generally, that as wanted $\Sigma_{jump} = \Sigma_\tau$; which ends the proof. □

Chapter 6

Toward higher-order machines: a $\Sigma_3 \wedge \Pi_3$ three-symbol simulational operator

In this chapter we design a 3-symbol $\Sigma_3 \wedge \Pi_3$-suitable, simulational and looping stable operator. We will write Γ_3 for this operator and, for convenience, λ_3, ζ_3, ... for its associated constants. With what we have done so far, namely with Theorem 5.4.14, the equality $\Sigma_3 = \mathrm{T}_3$ will then be immediate. Similarly, the operator Γ_3 enhances the operator Γ_{sup} and so it is at least as powerful as this operator. But we will see how it is actually strictly more powerful.

So going further than the equality $\Sigma_3 = \mathrm{T}_3$, one of the main aim of this chapter, after the definition of Γ_3, is to prove for it a generalization of the λ-ζ-Σ theorem (i.e. Theorem 4.1.33). This, as much as is possible, corroborates the choice of this operator for the definition of a higher-order model of infinite Turing machines. This new theorem, Theorem 6.2.49 reads:

$$L_{\lambda_3} \prec_{\Sigma_1} L_{\zeta_3} \prec_{\Sigma_2} L_{\mathrm{K}_3} \prec_{\Sigma_3} L_{\Sigma_3}$$

And so, we will see how a new constant, K_3, related to a new way of writing ordinals, appears in the study of the operator Γ_3. This way of writing, called K-writing, is strictly stronger, as in more demanding, than accidentally writing and strictly weaker than eventually writing. An immediate corollary of this theorem is the fact that $\Sigma_{\mathrm{sup}} < \Sigma_3$, and another, almost immediate, is the fact that $\zeta_{\mathrm{sup}} < \zeta_3$.

Moreover, studying this new constant K_3 and the new notion of K-writing we exhibit, as in the previous chapter, some fundamental but implicit properties of the writing notions, like the notion of eventually writing, which are used extensively in the study of ITTMs. For example, only one real can be e.w. by a machine at a time. But this fact is obvious and for this reason it was never explicitely used or raised to the status of a property. However, when working with K-writing–designed such that this fact is also

true–its importance appears more clearly and with the fact that its easily not true for other writing notions (take accidentally writing for example.)

Finally, for set-theorical reasons, the proof of Theorem 6.2.49 will be a bit more involved than that of the classical λ-ζ-Σ theorem. This comes from the fact that, when speaking of eventually writing, a Σ_2 formula is a Π_1 formula in disguise: if we want to eventually decide whether $\exists x, \forall y \psi(x, y)$ is true, it is enough to eventually decide for the first witness $\widetilde{x}$ whether $\forall y \psi(\widetilde{x}, y)$, and, as Π_1 formulas are true in substructure, this can be easily eventually decided.

6.1 Preliminary results on simulational Γ-machines

The results in this section aim at better understanding the behavior of simulational Γ-machines under the light of Theorem 5.4.14, as well as giving us some tools that will be needed for the present chapter.

Definition 6.1.1 (Γ-Computable function)**.** Let Γ an operator, two sets E and F that can be encoded in ${}^{\omega}2$ with an usual encoding and $f : E \to F$ a total function. We say that f is a Γ-*computable function* if there exists a Γ-machine m_f such that : for any $b = f(a)$ and for any reals x encoding a, the computation of m_f on input x terminates and outputs some real y such that y is a code for b. For any real z that does not encode any element of E, nothing is imposed for the behavior of m_f on input z.

Definition 6.1.2 (Computable predicate)**.** Let Γ be an operator and P be a unary predicate on some set E. We say that P is a Γ-*computable predicate* if its indicator function $\chi_P : E \to 2$ is a Γ-computable function.

Definition 6.1.3 (Semicomputable predicate)**.** Let Γ be an operator and P a unary predicate on some set E such that E can be encoded in ${}^{\omega}2$. We say that P is a Γ-*semicomputable predicate* (resp. Γ-*co-semicomputable predicate*) if there exists a Γ-machine m_P such that: for any a and for any real x encoding a, m_P terminates on input x if and only if $P(a)$ (resp. $\neg P(a)$). For any real y that does not encode any element of E, nothing is imposed for the behavior of m_P on input y.

Example 6.1.4.

- Let $f : {}^{\omega}2 \to {}^{\omega}2$ be a function on the reals. Then, if there is a Γ-machine m such that on any input x, m terminates and writes $f(x)$ as its output, f is a Γ-computable function.

- Let $f : \omega_1 \to \omega_1$, $\alpha \mapsto \alpha \cdot 2$. Then f is Γ_{sup}-computable function with the usual encoding being: x encodes α when it describes some well-order $\prec$ on ω (i.e. $n \prec m \longleftrightarrow x[\langle n, m \rangle] = 1$) of order-type α. Indeed, consider the machine m such that given a code x of α, it split ω in two disjoints sets $A = (a_i)$ and $B = (b_i)$ which it

saves on its working tape. Then it writes the following well-order $\prec_2$ on its output tape: for all a_i and a_j such that $i \prec j$, it writes $a_i \prec_2 a_j$. This effectively copies $\prec$ on the output tape using A as the underlying ordered set. Then, for all b_i and b_j such that $i \prec j$, the machine writes $b_i \prec_2 b_j$. This copies $\prec$ a second time using the other part of ω. Then, for all a_i and b_j, the machines writes $a_i \prec_2 b_j$ which yields, as required, a code for a well-order of order type $\alpha \cdot 2$.

Lemma 6.1.5. *Let Γ be a simulational n-symbol operator and P a Γ-co-semicomputable predicate on the n-symbol reals (i.e. on ${}^{\omega}n$). Then, for any machine m, the first real x to appear on one of the tapes of m and such that $P(x)$ is Γ-e.w.*

Proof. Let m and x be as defined and suppose w.l.o.g. that x appeared on the output tape of m. We consider the following computation of a machine $\mathcal{M}$ that will e.w. x. Machine $\mathcal{M}$ simulates m and for all real x' that appears on the output tape of m, it writes x' on its main output and it launches a simulation of $m_P[x']$ where m_P is the machine that co-semidecides P, i.e. that terminates on input x if and only if $\neg P(x)$. If $\neg P(x')$, then $m_P[x']$ terminates and the simulation of m goes on. Else, if $P(x')$, $x' = x$ as x is the first to appear on the output tape of m and $m_P[x]$ never terminates. So as wanted $\mathcal{M}$ has e.w. x. □

Proposition 6.1.6. *Let Γ be a simulational n-symbol operator that emulates the operator Γ_{sup}. For any Γ-computable predicate P on Σ_Γ such that $P(\zeta_\Gamma)$, the set $\{\alpha \in \zeta_\Gamma \mid P(\alpha)\}$ has order-type ζ_Γ and the set $\{\alpha \in \Sigma_\Gamma \mid P(\alpha)\}$ has order-type Σ_Γ.*

Proof. We define the application $\mathrm{B} : \beta \mapsto \mathrm{OT}(\{\alpha \in \beta \mid P(\alpha)\})$, where OT denotes the order-type of a set of ordinals. Observer first that :

- B is monotonically increasing.
- for any β, $\mathrm{B}(\beta) \leqslant \beta$.
- Given some ordinal β, $\mathrm{B}(\beta)$ is Γ-computable as P is.

Then we consider $\mathrm{B}(\Sigma_\Gamma)$. Suppose toward a contradiction that $\mathrm{B}(\Sigma_\Gamma) < \Sigma_\Gamma$. We design a machine $\mathcal{M}$ that does the following: using $\mathcal{U}_\Gamma$, it iterates through the ordinals below Σ_Γ. That is it simulates $\mathcal{U}_\Gamma$ on some part of its working tape and it checks at each stage whether some ordinal is written on any tape of the machine simulated inside of $\mathcal{U}_\Gamma$. It decide whether a real encodes an ordinal as Γ emulates the operator Γ_{sup}. And for each such ordinal σ produced by $\mathcal{U}_\Gamma$, $\mathcal{M}$ checks whether $\sigma < \mathrm{B}(\Sigma_\Gamma)$ in the following fashion. It launches a fresh simulation of $\mathcal{U}_\Gamma$ and looks with it for an ordinal σ' such that $\sigma < \mathrm{B}(\sigma')$. If it finds such an ordinal σ', as σ' is a.w., we have $\sigma' < \Sigma_\Gamma$ and so $\mathrm{B}(\sigma') \leqslant \mathrm{B}(\Sigma_\Gamma)$ which yields $\sigma < \mathrm{B}(\sigma') \leqslant \mathrm{B}(\Sigma_\Gamma)$. Also, by definition of Σ_Γ, the a.w. ordinals are unbounded in Σ_Γ. Hence if and only if $\sigma < \mathrm{B}(\Sigma_\Gamma)$ do we eventually find an a.w. σ' that satisfies the inequality. Consequently, when $\sigma \geqslant \mathrm{B}(\Sigma_\Gamma)$, the machine never finds the σ' it is looking

for and it eventually writes σ. Hence, under the hypothesis that $\mathrm{B}(\Sigma_\Gamma) < \Sigma_\Gamma$, $\mathrm{B}(\Sigma_\Gamma)$ itself is eventually writable and $\mathrm{B}(\Sigma_\Gamma) < \zeta_\Gamma$.

Now, we consider $\mathrm{B}(\zeta_\Gamma)$. By monotonicity, we have $\mathrm{B}(\zeta_\Gamma) \leqslant \mathrm{B}(\Sigma_\Gamma)$ and from there $\mathrm{B}(\zeta_\Gamma) < \zeta_\Gamma$. So $\mathrm{B}(\zeta_\Gamma)$ is e.w. We show that $\mathrm{B}(\zeta_\Gamma) < \zeta_\Gamma$ is a contradiction. We do the following computation. Some machine $\mathcal{N}$ e.w. $\mathrm{B}(\zeta_\Gamma)$ and looks in parallel for σ' such that $\mathrm{B}(\sigma') = \mathrm{B}(\zeta_\Gamma)$ to e.w. it. Either $\sigma' = \zeta_\Gamma$ and we reached a contradiction as ζ_Γ is not e.w. Either $\sigma' < \zeta_\Gamma$ and the next ordinal after σ' satisfying P is ζ_Γ; as it would otherwise yield $\mathrm{B}(\sigma') < \mathrm{B}(\zeta_\Gamma)$. And so, we claim that ζ_Γ is e.w. from σ'. Indeed consider the computation that given σ' looks for, using $\mathcal{U}_\Gamma$, the least $\sigma'' > \sigma'$ such that $P(\sigma'')$. Then, as σ' is e.w., so would be ζ_Γ and we reached again the same contradiction. Hence, $\mathrm{B}(\Sigma_\Gamma) = \Sigma_\Gamma$ and $\mathrm{B}(\zeta_\Gamma) = \zeta_\Gamma$. □

For the next proposition, we need some (slight) latitude in the way we encode ordinals. Ordinals are encoded using well-orders, themselves encoded bit by bit on a real. But whether $i \prec j$ is denoted by a bit set to 1 or to 0 is simply a convention. The following definition is simply a variation of Definition 3.2.27 with respect to this convention.

Definition 6.1.7. We say that a real x *negatively encodes* or *encodes with a bitwise negative encoding* some ordinal α when x encodes a relation $\prec_\alpha$ of order type α in the following fashion:

$$i \prec_\alpha j \longleftrightarrow x[\langle i, j\rangle] = 0$$

That is, the negative encoding of x is simply the usual encoding of x in which we invert 0's and 1's.

A negative encoding is simply the bitwise negation of a positive encoding. For this reason, transforming one into the other is easily done in ω steps. However, when trying to establish precise results regarding times of computation or stages of apparition, those ω steps may make it undoable with one encoding while it is doable with the other. For this reason, Proposition 6.1.8 is stated with the negative encoding as and likely does not hold with the positive encoding. It should be clear however, that is the hypothesis was that Γ enhances Γ_{inf} rather than Γ_{sup} then the situation would be would be reversed with respect to the negative and the positive encoding. This is briefly discussed in Remark 6.1.9.

Proposition 6.1.8. *Let Γ be a n-symbol suitable, looping stable and simulational operator that also enhances the operator Γ_{sup}. Then, there exists a Γ-machine m_ζ in which a negative encoding of ζ_Γ appears for the first time at stage ζ_Γ.*

Proof. We describe Γ-machine m_ζ that will write a well-order $\prec$ of order type ζ at stage ζ.

The machine m_ζ works with a simulation of $\mathcal{U}_\Gamma$ to keep track of the machines that appear to converge, that is machines whose output has not been modified for more than

one computation step. Then, the idea is that if m and m' appeared to have converged respectively at stages α and α' with $\alpha < \alpha'$, the m_ζ will have saved the fact that m appeared to have converged before m' and it will write $m \prec m'$ in its well-order, and that will let it keep track of the order type of the machines that appear to converge.

More precisely, m_ζ begins with the empty well-order, which is, with the bitwise negative encoding, the real 1^ω. Then, for any α, between stage α and $\alpha + 1$ in the simulation of $\mathcal{U}_\Gamma$, it looks at the machines whose output did not change. Let M_α be the set of those machines. Then it goes through all $m \in M_\alpha$ ordered by the natural order on integers. If m is already in the order-type, it skips it. If m wasn't in the order-type yet, it writes a 0 in $\langle m, m\rangle$ and in all cells $\langle m', m\rangle$ for all machines m' that are already in our order-type. In particular, this means that if m and m' in M_α weren't already in the order type, we will have $m \prec m'$ if and only if $m < m'$ (where $<$ denotes the natural order on integers). Moreover, for all machines $m \notin M_\alpha$, m_ζ removes it from the order-type, simply by writing a 1 in $\langle m, m\rangle$ and all cells $\langle m', m\rangle$ and $\langle m, m'\rangle$ for any m'.

We claim that at stage ζ_Γ, this effectively writes an encoding for ζ_Γ. First, under the assumption on Γ, Corollary 5.4.30 holds and $\zeta_\Gamma = \eta_\Gamma$. Hence, a Γ-machine that converges can't stabilize after stage η_Γ. Moreover, a machine that converges up to stage ζ_Γ can't see its output modified after as by $\Sigma_\Gamma = \mathrm{T}_\Gamma$, this stage greater that ζ_Γ would be a.w. and then also e.w. So a Γ-machine converges if and only if it converges up to stage ζ_Γ. So, as Γ enhances Γ_{sup}, at stage ζ_Γ, the cell $\langle m, m'\rangle$ is set to 0 if and only if from some stage $\alpha < \zeta_\Gamma$ and up to ζ_Γ, $C_{\langle m,m'\rangle} = 0$. Which means that m and m' were converging up to ζ_Γ, that is they converge definitely and the output of m converged before that of m'. Hence, at stage ζ_Γ, the machine m_ζ has written a well-order $\prec_\theta$ which is the well order of the machine that converge definitely, ordered by the ordinal stage of convergence.

Now, we need to show that the order type of this well-order is actually, ζ_Γ, the order type of the Γ-e.w. ordinals. Observe first that, writing θ for the order type of $\prec$, θ is written in ζ_Γ steps in each of which at most ω elements are added to the well-order. This shows that $\theta \leqslant \omega \cdot \zeta_\Gamma$. By closure of ζ_Γ, $\omega \cdot \zeta_\Gamma = \zeta_\Gamma$ and $\theta \leqslant \zeta_\Gamma$. Then we need to show that $\theta \geqslant \zeta_\Gamma$. Suppose that $\theta < \zeta_\Gamma$. This means that θ is e.w. So, we consider the following computation: a machine e.w. θ and, in parallel with the usual intertwining technique, it looks for some stage $\alpha \in \Sigma_\Gamma$ (using $\mathcal{U}_\Gamma$) such that at simulation stage α of $m_\prec$, the order written in $m_\prec$ is equal to θ and moreover such that the θ machines that constitutes this well-order are not modified before Σ_Γ (this is semi-decidable using $\mathcal{U}_\Gamma$ that produce ordinals unbounded in Σ_Γ). By construction, the first α it finds that satisfies those requirements is e.w.

We claim that ζ_Γ is the least ordinal α that satisfies those conditions and so that is it would be e.w. under the assumptoin that $\theta < \zeta_\Gamma$. Indeed, first, it is clear that ζ_Γ satisfies those conditions. Then, for any stage $\alpha < \zeta_\Gamma$ at which θ is written, there are converging machine that will converge strictly between stages α and ζ_Γ. So, some of the machines that form the order-type at stage θ are not definitely converging, as otherwise it would

produce an order type strictly greater than θ at stage ζ_Γ. Hence, any such $\alpha < \zeta_\Gamma$ does not satisfies the conditions of the previously described machine and ζ_Γ is e.w.; which is a contradiction. So as wanted, $\theta = \zeta_\Gamma$ and ζ_Γ appears with the negative encoding at stage ζ_Γ in $\mathcal{M}_\prec$. □

Remark 6.1.9. *In the previous proposition, it appears that the operator Γ_{sup} is more naturally used with negative encoding while, symmetrically, the operator Γ_{inf} "corresponds" to the usual positive encoding. This comes from the fact that with the operator Γ_{sup}, convergence is symbolized by 0; that is that the constraint for some tape to have converged below limit stage α is the same,* mutatis mutandis*, as the constraint for a 0 to appear at stage α. This explains why in the previous proof we used 0's to denote pieces of information regarding the convergence of the machines, and so why we used the negative encoding.*

As, by convention, in this thesis we chose to use the Γ_{sup} operator as the classical ITTM operator, we stick with this choice. Nonetheless, as the positive encoding is more natural (the empty order is represented by 0^ω, pieces of information regarding the order are represented by 1's etc.), this provides a good argument for preferring the operator Γ_{inf} over the operator Γ_{sup}.

Proposition 6.1.10. *Let Γ be a n-symbol suitable, looping stable and simulational operator that enhances Γ_{sup}. Then the set*

$$\{\alpha \in \Sigma_\Gamma \mid \alpha \text{ is multiplicatively closed and some negative encoding of it appears in } m_\zeta \text{ for the first time at time } \alpha\}$$

has order-type Σ_Γ. This induces a Γ-computable application $\cdot^* : \Sigma_3 \to \Sigma_3$ such that β^* is the β^{th} ordinal α in this set. Hence β^* is an multiplicatively closed ordinal that appears with the negative encoding at time β^* in some Γ-machine.

Proof. The ordinal ζ_Γ is multiplicatively closed (it is easy to e.w. the product of two e.w. ordinals) and appears with the negative encoding at time ζ_Γ in m_ζ by Proposition 6.1.8. This is then an application of Proposition 6.1.6, we simply need to show that the predicate in this definition by comprehension is a Γ-computable predicate. This will also show that the application $\cdot^*$ is Γ-computable.

And this predicate is indeed Γ-computable as, first, given $\alpha \in \Sigma_\Gamma$ a machine can easily check whether it is multiplicatively closed by looking for some $\delta \leqslant \alpha$ such that $\alpha = \omega^{\omega^\delta}$. Then, for the second half of the condition, a machine can simulate m_ζ for α steps and check, at each stage $\beta < \alpha$, that a negative encoding of α does not appear on the output tape of m_ζ while also that one does appear on it at stage α. □

Definition 6.1.11 (Star-ordinal)**.** We call an ordinal part of the set described in Proposition 6.1.10 a *star-ordinal*. That is, α is a star-ordinal when there is β such that $\alpha = \beta^*$.

Proposition 6.1.12. *Whether an ordinal α is a star-ordinal can be Γ-decided in $\alpha^2 + \alpha$ steps.*

Proof. We describe a machine that decides whether some ordinal α given as input is a star-ordinal. To do this, it must check whether

1. α is multiplicatively closed
2. α appears at stage α in m_ζ and for the first time.

To check the first item, a machine can count δ^2 steps for all $\delta < \alpha$ while counting through α in parallel. Ordinal α is not multplicatively closed if and only if the machine counted through all of α strictly before having counted δ^2 steps for all $\delta < \alpha$. After those α steps, it needs ω steps to check that α was indeed counted through. Hence, the first item can be decided in $\alpha + \omega$ stages.

For the second item, as mentioned in the proof of proposition 6.1.10, it is enough to simulate m_ζ for α steps and to check that a negative encoding for α only appears in it at stage α. Observe first that if α is a star-ordinal, then any ordinal appearing earlier is strictly smaller, as otherwise with the usual truncation technique, α would be a.w. before stage α. Similarly, for reals that do not encode a well-order but who do encode a relation such that it has a well-order as an initial segment, with the same reasoning the order type of those initial segments is strictly less than α. This is important as, when deciding whether one of those reals encodes an ordinal, the usual algorithm will count through this well-ordered part before it concludes that it does not encode an ordinal.

So, if α is a star-ordinal, the machine checks that a negative encoding for α only appears in m_ζ at stage α by simulating it for α stages. It takes at most α steps for each stage below α to check that any code appearing is strictly smaller than α (observe that checking whether a real encodes an ordinal and whether this ordinal is strictly smaller than α can be done in parallel). Then, once stage α has been reached, the machine needs α more steps do observe that a code for α appeared in m_ζ. Hence this makes $\alpha^2 + \alpha$ steps.

If α is multiplicatively closed but not a star-ordinal, either α appears too early and the machine finds it before α^2 steps, or it does not appears at stage α and the machine observes this by stage $\alpha^2 + \alpha$, as required. □

6.2 The $\Sigma_3 \wedge \Pi_3$ machine

We provide a description of a $\Sigma_3 \wedge \Pi_3$ suitable, simulationnal and looping stable operator that enhances the lim sup operator that we call Γ_3. For this operator, the main aim is to prove Theorem 6.2.49 which is the generalization of the λ-ζ-Σ theorem (Theorem 4.1.33) established for the lim sup machines in [Wel00a, Theorem 2.1]. This new theorem reads

$$L_{\lambda_3} \prec_{\Sigma_1} L_{\zeta_3} \prec_{\Sigma_2} L_{\mathrm{K}_3} \prec_{\Sigma_3} L_{\Sigma_3}$$

and so, we will see how a new constant, K_3, itself linked to a new way of writing ordinals, makes its apparition when studying the operator Γ_3.

6.2.1 The $\Sigma_3 \wedge \Pi_3$ operator Γ_3

In the previous chapter, we have seen that for an operator to be suitable, it must, between other conditions, be contraction-proof. This means that for a given cell history the limit rule must yield the same value for this cell history and for the contraction of this cell history. Hence, to describe a contraction-proof operator, it is easier to describe it on histories that are already "fully contracted". We call those stutter-free, as introduced in Definition 5.3.3, and we introduce a few definitions for stutter-free histories that will be used to describe the $\Sigma_3 \wedge \Pi_3$ operator Γ_3.

Definition 6.2.1 (Stutter-free contraction)**.** As defined in the previous chapter in Definition 5.3.25, we write ctr for the application that maps an ordinal word h on some alphabet A to its contraction, the word obtained by reducing every stutter of h to a single letter until it is stutter-free.

Also, we have seen in Theorem 5.4.1, that there are only two looping stable and simulationnal 2-symbol operator, namely $\Gamma_{\sup}$ and $\Gamma_{\inf}$. So Γ_3 is a 3-symbol operator and, from now on, the cell histories we consider are ordinal words on the alphabet $3 = \{0, 1, 2\}$ while the machine histories are ordinals words on the alphabet ${}^{\omega}3$.

Definition 6.2.2 (Segments)**.** Given a stutter-free cell history h and an ordinal σ, a 1-*segment of length* σ is the data of some ordinal index ν such that $h[\nu] = 1$ and such that for all $\iota \in [\nu, \nu + \sigma[$, $h[\iota] \in \{1, 2\}$. That is, the history subword delimited by $[\nu, \nu + \sigma[$ is either 1 or are an alternation of 1 and 2's, beginning with a 1. We define 0-*segments* in the same way, replacing the 1's by 0's.

Example 6.2.3.

- the word 0202 is a 0-segment of length 4.
- the word 202 is not a 0-segment but it ends with a 0-segment of length 2.
- the word $010101\ldots$ is made from consecutive 0-segments and 1-segments of length 1.
- the word $(12)^{\omega+3}$ is a 1-segment of length $\omega + 6$.

Definition 6.2.4 (1-populated)**.** Let h be a stutter-free cell history and β_0 a fixed ordinal. We say that h is 1-*populated with respect to* β_0 if after any position ν in h, we find some position ν' that starts a 1-segment $[\nu', \nu''[$ of length $|[\nu', \nu''[| = \alpha' > \beta_0$ after less than $\alpha' + \beta_0$ letters. In other words, we ask for α' to be such that $|[\nu, \nu'[| \leqslant \alpha' + \beta_0$, where $|[\nu, \nu'[|$ is the ordinal length of the segment delimited by the positions ν and ν'. This is

depicted in Figure 6.1. Formally, with quantifiers bounded so that the variables defines valid segments, denoting the ordinal length of segments with $|\cdot|$,

$$h \text{ is 1-populated} \iff \forall \nu\, \exists \nu' \geqslant \nu, \nu'' \geqslant \nu' \,(\text{the segment } [\nu', \nu''[\text{ of } h \text{ is a 1-segment strictly longer than } \beta_0 \wedge |[\nu, \nu'[| \leqslant |[\nu', \nu''[| + \beta_0)$$

with, in the above definition α' being now $|[\nu', \nu''[|$ and with the convention that for any ν, $[\nu, \nu[$ is the empty segment, hence of length 0 and neither a 0-segment nor a 1-segment. We define 0-populated words accordingly.

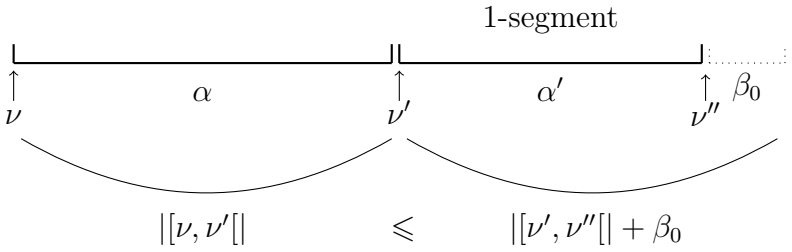


Figure 6.1: a 1-populated segment w.r.t. β_0.

Example 6.2.5.

- The stutter-free words 0 and 1 are respectively 0 and 1-populated w.r.t. $\beta_0 = 0$. The only index is $\nu = 0$ and taking $\nu' = \nu$ and $\nu'' = \nu + 1$ satisfies the predicate.
- The word $w = (01)^\omega$ is both 1-populated and 0-populated w.r.t. 0. Indeed, for any ν which is index of w, either $w[\nu] = 1$ or $w[\nu] = 0$. In the first case, take $\nu' = \nu$ and $\nu'' = \nu + 1$ and $w[\nu', \nu'']$ is the single-letter word 1 which is a 1-segment of size 1. Moreover $|[\nu, \nu'[| = 0 \leqslant 1 = |[\nu', \nu''[| + 0$. In the second case, take $\nu' = \nu + 1$ and $\nu'' = \nu + 2$. This describes a 1-segment of size 1 and we have $|[\nu, \nu'[| = 1 \leqslant 1 = |[\nu', \nu''[| + 0$
- The word $w = (012)^\omega$ is 1-populated w.r.t. 0 but not 0-populated w.r.t. any β_0. For the first affirmation: observe that this word is made from 0-segments of length 1 and 1-segments of length 2. And so for any ν index of w: if $w[\nu] = 0$, a 1-segments of length 2 starts at stage $\nu' = \nu + 1$ and it suits the definition. If $w[\nu] = 1$, $\nu' = \nu$ itself starts a 1-segment of length 2 and it also suits the definition. If $w[\nu] = 2$ then $\nu' = \nu + 2$ starts a 1-segments of length 2 and it yields $|[\nu, \nu'[| = 2 \leqslant 2 = |[\nu', \nu''[| + 0$

  As for the second affirmation, namely that this word is not 0-populated w.r.t. any β_0, observe first that as the 0-segment are all of length 1, the condition that says that the length of the 0-segment described by ν' and ν'' must be strictly greater then β_0 makes it impossible for the word to be 0-populated w.r.t. any $\beta_0 > 0$. As for

$\beta_0 = 0$, take ν such that $w[\nu] = 1$. Then, the first 0-segment starts at least at stage $\nu + 2$. Hence for any ν', ν'' describing a 0-segment $|[\nu, \nu'[| \geqslant 2$ and $|[\nu', \nu''[| = 1$, as there only are 0-segments of length 1. As $\beta_0 = 0$, the condition on the length of the segments $[\nu, \nu'[$ and $[\nu', \nu''[$ is never true for this ν and any ν' and ν'' describing a 0-segment and the 0-population predicate does not hold.

– For any limit ordinals $\beta_0 < \alpha$ such that α is additively closed, the word $(12)^{\alpha+\beta_0}(02)^{\alpha}$ is 0-populated w.r.t. β_0. Indeed, it is composed of a 1-segment of length $2\cdot(\alpha+\beta_0) = \alpha + \beta_0$ and a 0-segment of length $2 \cdot \alpha = \alpha$. Now, take any ν index in w. If ν is in the 1-segment, that is $\nu < \alpha + \beta_0$, then taking $\nu' = \alpha + \beta_0$ and $\nu'' = \alpha + \beta_0 + \alpha = \alpha \cdot 2$, $[\nu', \nu''[$ describes a 0-segment of length $\alpha > \beta_0$ which gives :

$$|[\nu, \nu'[| \leqslant \alpha + \beta_0 = |[\nu', \nu''[| + \beta_0$$

If $\nu \geqslant \alpha + \beta_0$, as in the first example, the 0-segment of length 1 that starts at stage ν or $\nu + 1$ satisfy conditions.

Remark 6.2.6. *This slightly contrived definition can be seen to capture, at first glance, one thing. If we think of segment of history in informal terms of "density" of respectively* 1*'s and* 0*'s (that is ignoring the* 2*'s which mostly help working around the contraction-proof condition), a* 1*-segment has the highest density of* 1*'s and the lowest density of* 0*'s. Similarly for some limit ordinal* α*, the segment of history* $(10)^{\alpha}$ *has balanced proportions. On the other end the segment* $(1(02)^{\omega})^{\alpha}$ *does not and has "way more"* 0*'s than* 1*'s. But still,* 1*'s are cofinals in both those histories and so both would yield a* 1 *in a machine ruled by the* lim sup *operator. On the contrary, observe that the first history, that is* $(10)^{\alpha}$*, is* 1*-populated w.r.t.* 0 *while* $(1(02)^{\omega})^{\alpha}$ *is not* 1*-populated w.r.t. any ordinal. And so, in other terms, the definition of* 1*-populated is fine enough–finer than the concept of cofinality alone–to capture this variation of "density" from a perfect equilibrium to a rather "unbalanced" history. In this sense, the rule we design is finer than the usual* lim sup *rule.*

Definition 6.2.7. The definition 6.2.4 yields a formula $\phi^1(h, \beta_0, \nu_0)$ such that for a history h, some β_0 constant and ν_0 smaller than the length of h, the stutter-free contraction of the final segment of h starting from ν_0, $h^c_{>\nu_0}$, is 1-populated w.r.t. β_0 if and only if $\phi^1(h, \beta_0, \nu_0)$ is true in L. The formula $\phi^0(h, \beta_0, \nu_0)$ is defined accordingly.

Now for the definition of the operator Γ_3. It is defined as the conjunction of a Σ_3 formula and a Π_3 formula, hence making it, as will be shown, a $\Sigma_3 \wedge \Pi_3$-suitable operator.

Definition 6.2.8 (The operator Γ_3)**.** The operator Γ_3 is the unique 3-symbol operator defined such that for a Γ_3-machine m computing from the real y as input and at limit stage λ, the cell i is set to 0 if and only if both those conditions are satisfied:

1. After some stage ν_0 and for some ordinal β_0, $h^c_{>\nu_0}$, the stutter-free contraction of the final segment of the history after stage ν_0, is 0-populated w.r.t. β_0

2. For any ν_0, $h^c_{>\nu_0}$, the stutter-free contraction of the final segment of the history after stage ν_0, is not 1-populated w.r.t. any β_0.

That is formally, with h being the history up to stage λ and with the formulas of Definition 6.2.7:

$$C_i(\lambda) = 0 \iff \exists \nu_0 < \lambda \, \exists \beta_0 \, \phi^0(h, \beta_0, \nu_0) \wedge \forall \nu_0 < \lambda \, \forall \beta_0 \, \neg \phi^1(h, \beta_0, \nu_0)$$

Else, if the 1's are cofinal in the history of the cell i, it is set to 1. Otherwise, it means that the history converges on the symbol 2 and the cell i is set to 2.

Proposition 6.2.9. *The operator Γ_3 is a $\Sigma_3 \wedge \Pi_3$-suitable operator in the sense of Definition 5.3.19. That is, there is a formula $\varphi = \psi_1 \wedge \psi_2$ where ψ_1 is Σ_3 and ψ_2 is Π_3 such that for any machine m, cell i and real y as input, $C_i^{m,y}(\lambda) = 0 \iff L_\lambda[y] \models \varphi(i, m, y)$.*

Proof. First we define $\phi^1_c(h^c, \beta_0, \nu_0)$ such that for a stutter-free contracted history h^c, some constant β_0 and ν_0 (smaller than the length of h^c), the final segment of h^c starting from ν_0 is 1-populated w.r.t. β_0 when $\phi^1(h^c, \beta_0, \nu_0)$. For this, observe that when ν' and ν'' are given, deciding whether $h^c[\nu', \nu''[$ (the factor of h^c delimited by the segment $[\nu', \nu''[$) is a 1-segment is Σ_0. That is there exists a Σ_0 predicate S^1 such that $S^1(h^c, \nu', \nu'')$ when ν' and ν'' form a 1-segment in h^c. Indeed, per our definition

$$\text{The segment } [\nu', \nu''[\text{ is a 1-segment} \longleftrightarrow h^c[\nu'] = 1 \wedge \forall \iota \in [\nu', \nu''[\, (h^c[\iota] = 1 \vee h^c[\iota] = 2)$$

This allows us to define ϕ^1_c as follows, and again ϕ^0_c is defined in an analogous way.

$$\begin{aligned} \phi^1_c(h^c, \beta_0, \nu_0) := \forall \nu < |h^c| \, (\nu > \nu_0 \implies & \exists \alpha < |h^c|, \exists \alpha' < |h^c|, \exists \nu' < |h^c|, \exists \nu'' < |h^c| \\ & \nu' \geqslant \nu \wedge \nu'' \geqslant \nu' \wedge \nu + \alpha = \nu' \\ & \wedge \nu' + \alpha' = \nu'' \wedge S^1(h, \nu', \nu'') \\ & \wedge \alpha' > \beta_0 \wedge \alpha' + \beta_0 \geqslant \alpha \end{aligned}$$

Second, with the technique used in 5.3.37, we can design by recursion a Σ_1 function η such that $\eta(m, y, \nu) = \langle h^{i,c}_\nu \rangle_{i \in \omega}$, the collection of the descriptions of the contracted history for all cell i in the machine m computing from y and up to stage ν. This means that $h^{i,c}_\nu$ is the contraction of the word h^i_ν which is in turn a word of length ν with a symbol for every position $\iota < \nu$. So the ν^{th} letter of h^i is not in h^i_ν and, consequently, neither in $h^{i,c}_\nu$. Now, if ν is a limit ordinal then $h^{i,c}_\nu$ is easily defined as the limit of the $h^{i,c}_{\nu'}$ for $\nu' < \nu$. Else, suppose that $\nu = \nu' + 1$ is a successor ordinal. We show that we can define $h^{i,c}_\nu \; from h^{i,c}_{\nu'}$. To do this, we show first how we can define $h^i[\nu']$, the $(\nu)^{\text{th}}$ letter in the history of i from $h^{i,c}_{\nu'}$. For this, we distinguish two cases.

- If ν' is a limit ordinal, $h^i[\nu']$ is set according to the limit rule. For this we use the previously defined ϕ^0_c and ϕ^1_c applied to the history $h^{i,c}_{\nu'}$ that we obtain by induction.

That is, $h^i[\nu']$ is set to 0 if and only if $\exists \nu_0 < \nu', \beta_0 < \nu' \phi_c^0(h^{i,c}_{\nu'}, \beta_0, \nu_0) \wedge \forall \nu_0 < \nu', \beta_0 < \nu' \neg \phi_c^1(h^{i,c}_{\nu'}, \beta_0, \nu_0)$. As the quantifiers are bounded by ν' the definition of $h^i[\nu']$ from ν' and $h^{i,c}_{\nu'}$ is Σ_0.

- Else, if ν' is not a limit ordinal, this means that $\nu = \nu_0 + k$ with $k \in \omega$ and $k > 1$. Then, the state of the machine at stage ν_0 is given by $\eta(m, y, \nu_0 + 1)$ which was defined by induction. Eventually, with the code fo the machine, its snapshot at stage ν is Σ_0-definable from it.

Now we consider $h^{i,c}_{\nu'}$. If $h^{i,c}_{\nu'}$ is a limit word, as it is stutter-free, its final segment is the repetition of at least two symbols and $h^{i,c}_{\nu} = h^{i,c}_{\nu'} \cdot h^i[\nu']$. Else, we let $s_{\nu'}$ be the last symbol of $h^{i,c}_{\nu'}$. If $h^i[\nu'] \neq s_{\nu'}$ then $h^{i,c}_{\nu} = h^{i,c}_{\nu'} \cdot h^i[\nu']$. Otherwise, if the symbols are the same, $h^{i,c}_{\nu} = h^{i,c}_{\nu'}$. This shows how the Σ_1 function η is recursively defined. And with it, we also define $\eta(i, m, y, \nu_0, \nu)$ as be the contraction of the history of the cell i between stages ν_0 and ν. It can easily enough be defined from $h^{i,c}_{\nu_0}$ and $h^{i,c}_{\nu}$ and we write it $h^{i,c}_{[\nu_0,\nu[}$.

Eventually, using η, we can define $\widetilde{\phi}^1$. The idea is that, while we can't define the history up to stage λ in L_λ, we are able to define it up to any $\iota < \lambda$ using η. And this is enough, as we only need to look at initials segment of the history that spans up to λ. So once ν is quantified in L_λ, we look for a segment of the history $h_{[\nu,\iota[}$ in which, when contracted, we find the desired ν' and ν''. As the position ν in h corresponds to position 0 in $h[\nu, \iota[$, it also corresponds to position 0 in $h^c[\nu, \iota[$ which makes things easier. In particular, the α of the definition of 1-population, that is the length of the segment from ν up to ν' is now simply ν' itself.

$$\begin{aligned}\widetilde{\phi}^1(i, m, y, \beta_0, \nu_0) := \forall \nu > \nu_0 \, \exists \iota > \nu \, \exists h^{i,c}_{[\nu,\iota[} \, \exists \alpha \exists \alpha' \, \exists \nu' \geqslant \nu \, \exists \nu'' \geqslant \nu' \\ (h^{i,c}_{[\nu,\iota[} = \eta(i, m, y, \nu, \iota) \\ \wedge \nu' < |h^{i,c}_{[\nu,\iota[}| \wedge \nu'' < |h^{i,c}_{[\nu,\iota[}| \\ \wedge 0 + \alpha = \nu' \wedge \nu' + \alpha' = \nu'' \wedge S^1(h^{i,c}_{[\nu,\iota[}, \nu', \nu'') \\ \wedge \alpha' > \beta_0 \wedge \alpha' + \beta_0 \geqslant \alpha)\end{aligned}$$

The counterpart $\widetilde{\phi}^0(i, m, y, \beta_0, \nu_0)$ is defined in the same way from there we obtain, eventually:

$$\varphi(i, m, y) := \exists \nu_0, \beta_0 \, \widetilde{\phi}^0(i, m, y, \beta_0, \nu_0) \wedge \forall \nu_0, \beta_0 \neg \widetilde{\phi}^1(i, m, y, \beta_0, \nu_0)$$

such that, as wanted,

$$C_i^{m,y}(\lambda) = 0 \iff L_\lambda[y] \models \varphi(i, m, y)$$

□

Remark 6.2.10. *Proposition 6.2.9 allows us to speak of* 0 *or* 1*-populated cell histories*

even when those are not stutter-free and without the risk of altering the predicate complexity. However, when speaking of the length of the histories or of some segment of history, we still need to specify whether we speak of the raw cell history itself or of the contracted cell history, as the contraction may drastically change the length of the word.

Now, as in the Σ_2 case, this defines the machine behavior at limit stages (with its head back on the first cell and set to a distinguished q_{lim} state) and this completes the definition of $\Sigma_3 \wedge \Pi_3$ machines.

Remark 6.2.11. *Here we have a rule that is slightly more complex than Σ_3, it can be seen as a difference of Σ_3 formulas (i.e. a conjunction of a Σ_3 formula with the negation of another Σ_3 formula). This won't be an issue as this complexity still falls under the main theorem that we will prove, namely that a new Σ_3-end extension appears. Indeed, for two structures M and N, it is clear that $N \prec_{\Sigma_3} M$ implies that for any $\Sigma_3 \wedge \Pi_3$ formula $\varphi(p)$ with $p \in N$, $N \models \varphi(p) \longleftrightarrow M \models \varphi(p)$.*

6.2.2 Elementary results on the Γ_3 machine.

Proposition 6.2.12. *The operator Γ_3 is a simulationnal operator.*

Proof. We need to show that it is cell-by-cell, stable, asymptotic and contraction-proof.

By construction, it is cell-by-cell and contraction-proof (as it directly works with stutter-free contracted histories). It is also clearly stable.

As for the asymptoticity of our rule, observe first that the 0 and 1-populated properties are asymptotic. This comes from the fact that, in the formula of Definition 6.2.4, once the ordinal position ν is quantified, we can only look at greater ordinal positions for ordinals ν' and ν''. Hence, when we remove an initial segment of some h that is 1 or 0-populated, if we remove some positions ν' and ν'' witnesses of some ν, we also remove ν itself and the predicate stays true. So, if for some ν_0, $\phi^0(m, i, y, \beta_0, \nu_0)$ holds then for any $\nu_0' > \nu_0$, $\phi^0(m, i, y, \beta_0, \nu_0')$ also holds by asymptoticty of ϕ_0. And the asymptoticity of $\forall \nu_0, \beta_0 \neg \phi^1(m, i, y, \beta_0, \nu_0)$ is established with the same reasoning. And those together yield the asymptoticity of the operator Γ_3. □

Proposition 6.2.13. *The operator Γ_3 enhances the operator $\Gamma_{\sup}$.*

Proof. We show that without 2's written in the history of a computation, the operator Γ_3 yields the same limit values as the classical lim sup operator. Let h be some cell history in which the letter 2 never appears. Then, h is either (in its stutter-free form), 0, 1, $(01)^\alpha$ or $(10)^\alpha$ for some limit α. The word 0 is 0-populated w.r.t. 0 and not 1-populated for any β_0 (remember that in Definition 6.2.4, the segment $[\nu', \nu'']$ must be strictly longer than β_0.) So, as wanted, for the contracted history equal to 0, the rule yields 0 at stage α. As for the word 1, it is 1-populated w.r.t. 0 and this is enough for the rule to yield a 1 at stage α. Then, as seen in Example 6.2.5, $h = (01)^\alpha$ is 1-populated w.r.t. 0. Indeed, take

any ν below α. If $h[\nu] = 1$, then $\nu' = \nu$ and $\nu'' = \nu' + 1$ are witness of ϕ^1. Else if $h[\nu] = 0$, then $h[\nu + 1] = 1$ and in particular, $\nu + 1$ starts a 1-segment of length 1, which is greater than $\beta_0 = 0$. Taking $\nu' = \nu + 1$ and $\nu'' = \nu + 2$ (which is always possible as α is a limit ordinal), we have

$$|[\nu, \nu'[| = 1 \leqslant 1 + 0 = |[\nu', \nu''[| + \beta_0$$

Same goes for $(10)^\alpha$ which is also 1-populated w.r.t. 0 as α is limit. Eventually, as wanted, both $(01)^\alpha$ and $(10)^\alpha$ yields, w.r.t. Γ_3 (or more precisely w.r.t. γ_3, the cell history version of Γ_3), 1 at limit stage α. □

Now that Γ_3-machines are defined, so are Γ_3-writable, eventually writable and accidentally writable reals. For example, a real is Γ_3-accidentally writable if it appears at some stage in the computation of some Γ_3-machine from the empty input. Then as the Γ_3 operator is a simulationnal operator, there exists a Γ_3 universal machine that we write $\mathcal{U}_3$. And as with the Γ_{sup}-machines, all Γ_3-machines will start looping before some countable stage. To show this, by Proposition 5.3.44, it is enough to show that Γ_3 is looping-stable.

Lemma 6.2.14. *Let $\alpha > 0$ and β_0 be ordinals and h be a stutter-free cell history of length λ such that λ is additively closed and $\beta_0 < \lambda$. We write h^α for the ordinal word of length $\lambda \cdot \alpha$ in which h is repeated α times. Then h is 0-populated (resp. 1-populated) w.r.t. β_0 if and only if h^α is 0-populated (resp. 1-populated) w.r.t. β_0.*

Proof. We show it in the 0-populated case. In the first direction, assume h is 0-populated w.r.t. to β_0. Then any index ν in h^α can be written $\nu = \lambda \cdot \beta + \mu$, with λ the length of h and $\mu < \lambda$. Then, for all such μ, as it is an index in h, there exists an index $\mu' > \mu$ of h starting a 0-segment big enough in h. So $\lambda \cdot \beta + \mu'$ will be in turn a witness for ν in h^α in the definition of 0-populated w.r.t. to β_0.

In the other direction, let ν be an index of h. Then $\nu < \lambda$ where λ is the length of h. We consider the same index in h^α. As h^α is by hypothesis 0-populated there are some least ν' and ν'' that satisfy the 0-populated predicate for this ν in h^α. If $\nu' < \lambda$ then $|[\nu, \nu'[| < \lambda$ as well. Hence, as β_0 is also less than λ, the least ν'' does not need to be greater than λ. And so it is also an index in h and ν' and ν'' satisfy the 0-populated predicate for this ν in h.

Now, suppose $\nu' > \lambda$ as in the example of Figure 6.2. Then, $|[\nu, \nu'[| > |[\nu, \lambda[|$ and as λ is additively closed, $\nu + |[\nu, \lambda[| = \lambda$ implies $|[\nu, \lambda[| = \lambda$. Hence, $|[\nu, \nu'[| > \lambda$. By definition of ν' and ν'' in the 0-population predicate: $\lambda < |[\nu, \nu'[| \leqslant |[\nu', \nu''[| + \beta_0$. Hence, by closedness again, $\lambda < |[\nu', \nu''[|$, as $\beta_0 < \lambda$. This means that the 0-segment $[\nu', \nu''[$ stretches at least over two different h's (while not necessarily covering both) of h^α. In particular, some part of the 0-segment $[\nu', \nu''[$ spans through the end of some subword of h; that is h finishes with a 0-segment or, in other word, some final segment of h is also a 0-segment. As a final segment of a word of additively closed length, this final segment

has the same length as the word and from there we see that h is 0-populated w.r.t. 0 and, *a fortiori*, w.r.t. β_0.

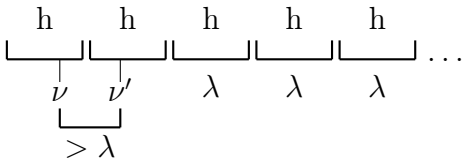


Figure 6.2:

□

Proposition 6.2.15. *The operator Γ_3 satisfies the condition of looping stability.*

Proof. Let h be a limit cell history and α be a limit ordinal. We want to show that $\Gamma(h^\omega) = \Gamma(h^\alpha)$. First, if 0 does not appear in h the equality clearly holds. The same goes if 1 does not appear in h

Hence, we can suppose that both 0 and 1 appear in h. Then, as α is a limit ordinal, $\alpha = \omega \cdot \beta$ for some β and $h^\alpha = (h^\omega)^\beta$. Further, $\mathrm{ctr}(h^\alpha) = \mathrm{ctr}(h^\omega)^\beta$ as both 0's and 1's are cofinal in h^ω. Also, as both 0 and 1 appear in h, the length of $\mathrm{ctr}(h^\omega)$ is a right multiple of ω and, as such, it is additively closed. We can apply the previous lemma which yield $\Gamma(\mathrm{ctr}(h^\omega)) = \Gamma(\mathrm{ctr}(h^\omega)^\beta)$. And the right-hand term can be rewritten $\Gamma(\mathrm{ctr}(h^\alpha))$ which yields the wanted equality as Γ is contraction-proof. □

Corollary 6.2.16. *There is some countable ordinal stage such that any Γ_3-machine either halts before this stage or is seen to be looping by this stage.*

Proof. Consequence of previous proposition with Proposition 5.3.44. This second proposition also gives a meaning for a Γ_3-machine to be "seen to be looping". □

Proposition 6.2.17. *Σ_3, the supremum of the Γ_3-a.w. is equal to T_3, the supremum of the Γ_3-a.c.*

Proof. As Γ_3 is a suitable, looping stable and simulationnal operator, this is a direct application of Theorem 5.4.14. □

Further, as in the lim sup case, we can establish a more precise looping condition, here using Lemma 6.2.14.

Proposition 6.2.18 (Looping condition)**.** *A Γ_3-machine is looping if and only if there are two limit stages $\nu < \nu'$ sharing the same snapshot, such that for all i, the length of the contraction of the segment of cell history of i between stages ν and ν' is additively closed and moreover for all i with $C_i(\nu') = 0$, ν is a witness of the existential half of $\varphi(m, i, y)$ in $L_{\nu'}$, i.e. of $\exists\nu_0\exists\beta_0\phi^0(h, \beta_0, \nu_0)$. That is such that $L_{\nu'} \models \exists\beta_0\phi^0(h, \beta_0, \nu)$.*

Remark 6.2.19. *This looping condition, apart from the technicality of the additive closeness, is analogous to the Σ_2 looping condition. As introduced in [HL00], the Σ_2 looping condition was the existence of two stages ν and ν' sharing the same snapshot and such that for all i, $C_i(\nu') = 0$ implies that all β between ν and ν', $C_i(\beta) = 0$. But this last condition means that for all i, $C_i(\nu') = 0$ implies that ν is witness in $L_{\nu'}$, of the following formula*

$$\varphi^{\Sigma_2}(m, i, y) := \exists \nu \, \forall \beta > \nu \; C_i(\beta) = 0$$

and $\varphi^{\Sigma_2}(m, i, y)$ describes exactly the Σ_2 rule.

Proof of proposition 6.2.18. Let ν and ν' be as described. As the rule is asymptotical, the machine will behave in the same way starting from ν or ν' and repeat the segment $[\nu, \nu']$. It may only escape this loop if at some limit stage, after this segment has been repeated a certain limit ordinal amout of time, the limit rule yields another snapshot. However, we can us Lemma 6.2.14 to show that the segment $[\nu, \nu']^\omega$ yields the same snapshot. Indeed, for i such that $C_i(\nu') = 0$, the condition on the witness says that $[\nu, \nu']$ is 0-populated w.r.t. some β_0 and never 1-populated. Using the lemma, the segment $[\nu, \nu']^\omega$ is also 0-populated and never 1-populated and $C_i(N) = 0$ with N being the right extremity of the segment $[\nu, \nu']^\omega$. We show in the same way that for cells i such that $C_{i'}(\nu) = 1$ we have $C_{i'}(N) = 1$. And by looping stability, the segment $[\nu, N]$ will itself yield the same snapshot after being repeated ω times and so on. This shows that the loop is never exited.

Conversely, if the machine is looping, we have some segment $[\nu, \nu']$ over which the machine repeats itself. However, for some i, writing h^c the contracted cell history of the cell i between stages ν, ν', the length of h^c may not be additively closed. Observe however that 1 is additively closed and that if $|h^c| > 2$, then repeating the segment $[\nu, \nu']$ some α times, for α limit, will multiply the quantity $|h^c|$ by α. This ensures that we can always find some stage $N > \nu'$ sharing the same snapshot as ν and such that for all i, the contracted cell history has the desired length. □

6.2.3 K-writing and some results

As said, we are able to *write*, *eventually write* and *accidentally write* reals with Γ_3-machines. We write λ_3, ζ_3 and Σ_3 the associated constants. Further, we will define a new way of writing that will be akin to the operator Γ_3 and will be strictly less demanding than eventually writing and strictly more than accidentally writing.

Definition 6.2.20 (Segments of reals)**.** Given some real x and an ordinal σ, we say that an *x-segment of length σ* appears in the computation of some machine m at stage ν when, writing H the output history of m, for all stage $\iota \in [\nu, \nu + \sigma[$, ι is an index in H and $H[\iota] = x$.

Definition 6.2.21 (Co-segments of reals)**.** Given some real x and an ordinal σ, we say that an *x-co-segment of length σ* appears in the computation of some machine m at stage ν when, writing H the output history of m, for all stage $\iota \in [\nu, \nu + \sigma[$, ι is a position in H and $H[\iota] \neq x$.

It is clear that the definition of x-segments and x-co-segments are a transposition of 1-segments and 0-segments in the context of reals where x plays the role of 1 and any $y \neq x$ plays the role of 0. And so are the following definitions of x-populated and x-co-populated segments.

Definition 6.2.22 (x-populated)**.** Given some computation, we say that the output history of the machine is x-populated after ν_0 and up to some limit ordinal λ and w.r.t. β_0 if, cofinally after ν_0, we find some x-segment of length $\alpha > \beta_0$ after less than $\alpha + \beta_0$ steps. That is, with quantifiers bounded by λ,

$$\forall \nu > \nu_0 \, \exists \nu' > \nu, \nu'' > \nu \, (\text{the segment } [\nu', \nu''[\text{ of } h \text{ is a } x\text{-segment stricly longer than } \beta_0 \\ \text{and } |[\nu, \nu'[| \leqslant |[\nu', \nu''[| + \beta_0)$$

This yields a Π_2 predicate $\phi(m, y, x, \beta_0, \nu_0)$ such that for a Γ_3-machine m, that computes from input y, and some β_0 constant, the output history is x-populated w.r.t β_0 from ν_0 and up to limit stage λ when $L_\lambda[y] \models \phi(m, y, x, \beta_0, \nu_0)$.

Definition 6.2.23 (x-co-populated)**.** Given some computation, we say that the output history of the machine is x-co-populated after ν_0 and up to some limit ordinal λ and w.r.t. β_0 if, cofinally after ν_0, we find some x-co-segment of length $\alpha > \beta_0$ after less than $\alpha + \beta_0$ steps. That is, with quantifiers bounded by λ,

$$\forall \nu > \nu_0 \, \exists \nu' > \nu, \nu'' > \nu' \, (\text{the segment } [\nu', \nu''[\text{ of } h \text{ is a } x\text{-co-segment} \\ \text{and } |[\nu, \nu'[| \leqslant |[\nu', \nu''[| + \beta_0)$$

Again, this yields a Π_2 predicate $\phi^{co}(m, y, x, \beta_0, \nu)$ such that for a Γ_3-machine m, that computes from input y, and some β_0 constant, the output history is co-x-populated w.r.t. β_0 from ν and up to limit stage λ when $L_\lambda[y] \models \phi^{co}(m, y, x, \beta_0, \nu)$.

Definition 6.2.24 (K-writing)**.** We say that x is K-*written* ("kappa-written") by m from input y when there is ν_0 and β_0 such that after stage ν_0, the output history of m is x-populated w.r.t. β_0 while not being x-co-populated after any other ν_0 and β_0. That is:

$$L[y] \models \exists \nu_0, \exists \beta_0 \, \phi(m, y, x, \beta_0, \nu_0) \wedge \forall \nu_0, \beta_0 \, \neg \phi^{co}(m, y, x, \beta_0, \nu_0)$$

We may also say that x is K-written up to some limit stage λ and from input y when $L_\lambda[y]$ models the previous predicate.

This way of writing satisfies two important properties of monotonicity and mutual exclusivity. To define these properties, observe that the definition of a new way of writing reals can be seen as a function that maps histories of reals (possibily of length On) to booleans.

Definition 6.2.25 (Writing predicate)**.** A writing predicate on n-symbol machine is a class function $w : {}^{\omega}n \times {}^{<On}({}^{\omega}n) \cup {}^{On}({}^{\omega}n) \to 2$. We say that a real x was written w.r.t. w in some history $H \in {}^{<On}({}^{\omega}n) \cup {}^{On}({}^{\omega}n)$ when $w(x, H) = 1$.

As such, a writing predicate is a class function as we need to consider histories of length On. The following definition will make things a bit more handy.

Definition 6.2.26 (Suitable writing predicate)**.** For an operator Γ, a writing predicate w is suitable w.r.t. Γ when there is a formula $\varphi_w(x_1, x_2, x_3)$ such that:

- For every Γ machine m that halts on input y, writing H_y the whole history of m with input y and λ the length of H_y, we have:

$$w(x, H_y) = 1 \iff L_\lambda[y] \models \varphi_w(m, x, y)$$

- For every Γ machine m that does not halt on input y, writing H_y the whole history of m on input y, we have:

$$w(x, H_y) = 1 \iff L[y] \models \varphi_w(m, x, y)$$

Example 6.2.27. Let Γ be a n-symbol suitable operator. That is Γ is defined by some formula $\varphi_\Gamma(x_1, x_2, x_3, x_4)$. Then, it is easy to show that the e.w. writing predicate is suitable w.r.t. any such Γ. Indeed, using φ_Γ we can define two functions, $C(i, m, y, \alpha)$ equal to the value of cell i in m at stage α with input y and $Q(m, y, \alpha)$ equal to the state of m at stage α with input y. From there it is enough to consider the following formula, writing $i \in Out$ for cells i forming the output tape of a machine.

$$\begin{aligned}\varphi_{e.w.}(m, x, y) \coloneqq \forall \alpha\, Q(m, y, \alpha) \text{ is not the final state} \\ \wedge \exists \alpha \forall \beta > \alpha \forall i \in Out\, (C(i, m, y, \alpha) = C(i, m, y, \beta))\end{aligned}$$

This shows that the e.w. writing predicate is suitable w.r.t. any suitable operator. And from this, it is clear that the e.w. predicate is as much the transposition of the operator Γ_{sup} as the K-writing predicate is that of the operator Γ_3.

Proposition 6.2.28. *κ, the writing predicate associated to* K*-writing, is a suitable writing predicate w.r.t. any suitable operator.*

Proof. As in the previous example, for any suitable operator Γ, the formula of Definition 6.2.24 yields a formula to define κ w.r.t. Γ. □

Definition 6.2.29 (Looping stability)**.** We say that a writing predicate w is *looping stable* when for any real x, history H and α a limit ordinal, the following holds:

$$w(x, H^{\omega}) = w(x, H^{\alpha}) = w(x, H^{On})$$

This property is very useful as it allows us speak of reals that are written w.r.t. some writing predicate w in machines that do not halt (but that do loop), without having to consider the whole output history of length On. This will naturally come into play with looping stable and simulational operators.

Further, we can define the two following properties that we will need for the study of the K writing predicate.

Definition 6.2.30 (Monotonicity)**.** We say that a writing predicate w is *monotonic* when for all x, H and H' such that

$$\begin{cases} |H| = |H'| \\ \forall \iota < |H| \, (H[\iota] = x \implies H'[\iota] = x) \end{cases}$$

we have

$$w(x, H) = 1 \implies w(x, H') = 1$$

That is if, index-by-index, there is in H' more x's than in H, then x being w-written in H implies that it is also w-written in H'.

Definition 6.2.31 (Exclusivity)**.** We say that a writing predicate w is *exclusive* when:

$$\forall x \, [w(x, H) = 1 \implies \forall y \neq x \, (w(y, H) = 0)]$$

Proposition 6.2.32. *κ, the writing predicate associated to* K*-writing is looping stable, monotonic and exclusive.*

Proof. Since all definitions involved are analogous to their counter-part defined for limit operators, the fact that it is looping stable is established as in Proposition 6.2.15, observing that the case H^{On} does not change the proof. Monotonicity is clear and exclusivity was the aim of this construction and comes from the fact that for any $x \neq y$, ν_0 and β_0, $\neg\phi^{co}(x, \beta_0, \nu_0) \implies \neg\phi(y, \beta_0, \nu_0)$. □

Definition 6.2.33. We write K_3 for the supremum of the K-writable ordinals with a Γ_3-machine.

The following proposition shows the interest of considering the K-writable ordinals. Namely it shows that there are a.w. reals that are not K-writable and so that being K-writable is strictly more demanding.

Proposition 6.2.34. K_3, *the supremum of the* K*-writable ordinals, is* Γ_3*-accidentally writable. That is,* $\mathrm{K}_3 < \Sigma_3$.

Proof. Consider $\mathcal{U}_3$, the Γ_3 universal machine. By Corollary 6.2.16, there is a limit countable stage β such that any non-halting machine is seen to be looping at this stage. That is, in the computation of any Γ-machine m, H, a final segment of the segment of history that spans below stage β is indefinitely repeated after stage β. In particular, as the K-writing predicate is looping stable, any Γ-machine whose computation K-writes some real x (so "up to On") also K-writes x up to stage $\beta \cdot \omega$ (we can w.l.o.g. suppose that β is additively closed). We will show that at this stage $\beta \cdot \omega$ all K-writable reals appeared in some machine. From there, it will be easy to show that their sum is a.w.

We consider the following computation of a machine $\mathcal{M}$. On one virtual tape, $\mathcal{M}$ simulates $\mathcal{U}_3$. For each machine m_k in the simulation of $\mathcal{U}_3$, $\mathcal{M}$ allocates one virtual tape, t_k, on its working tape. Each time m_k is simulated one step further (that is the simulation of one step of m_k which itself takes many steps of the simulation of $\mathcal{U}_3$), $\mathcal{M}$ first copies the content of the output of m_k in t_k and then fill t_k with $2's$. The 2's are there to transform the segments of reals in the output history of any machine m_k into cell histories with 1-segments and 0-segments (possibly of size only 2).

We write $H_k(\alpha)$ the history of the output of machine m_k up to stage α and H_k for the whole history and we claim that if m_k K-writes some x_k, then at stage $\beta \cdot \omega$, x_k will be written in t_k.

First, as stated, if m_k K-writes some x_k, it will K-writes x_k up to stage $\beta \cdot \omega$. Hence, in $\beta \cdot \omega$, there is some ν_0 and β_0 such that, after ν_0, $H_k(\beta \cdot \omega)$ is x_k-populated w.r.t. β_0. So, for any cell i such that $x_k[i] = 1$, where $x_k[i]$ is the i^{th} bit of x_k, $h_{k,i}$, the history of this cell in machine m_k up to stage $\beta \cdot \omega$ will be 1-populated w.r.t. this same ν_0 and β_0 and will yield a 1 at stage $\beta \cdot \omega$. This comes from the symmetry of the definition of x-population and 1-population. Similarly, for any cell i' set to 0 in x_k, its history $h_{k,i'}$ will also be 0-populated after some point. Still in this case, this is not enough, as we need to ensure that for any ν_0, β_0, the history of the cell i', $h_{k,i'}$, is not 1-populated.

However, if at some point i' is set to 1 this means that at this point, t_k, the whole tape, has not x_k written on it, as $x_k[i'] = 0$; hence any 1-segment in $h_{k,i'}$ corresponds to a x_k-co-segment in H_k. So if $h_{k,i'}$ was 1-populated w.r.t. some constants, H_k would be x_k-co-populated w.r.t. the same constants. Observe here that what comes fundamentally into play is the exclusivity of the K-writing: if more than one real was K-written by m_k, at a limit stage, we would most likely at best be able to write a bitwise sum of those. So, this ensure that i' is set to 0 at stage $\beta \cdot \omega$ and that x_k is, as claimed, written in t_k at this stage.

Finally, we proved that at some limit stage, all K-writable reals are written on some tapes of the machine $\mathcal{M}$ we designed. Now we can consider a machine that simulates $\mathcal{M}$ and, at each limit stage, that writes on its output the sum of all ordinal appearing in

it. This machine will at some point write the sum of all K-writable ordinals (and maybe other ordinals), which shows as wanted that K_3 is accidentally writable. □

Proposition 6.2.35. *The set of the K-writable ordinals, that is of ordinals α for which there is a Γ_3-machine that* K*-writes a code for a well-order of length α, is an initial segment of $\mathcal{O}$.*

Proof. Let $\alpha < \beta$ and m_β a machine that K-writes x_β, a code for the well-order $\prec_\beta$ of order type β. There is some integer i such that $\prec_\beta|_i$, the well-order restricted to elements $\prec_\beta$-smaller than i, has order-type α. Hence, as usual, the idea is simply to truncate below i every well-order that may appear in m_β. However, observe that checking whether some real on the output encodes a well-order may take a long time; in particular, when it is indeed a well-order, it takes as much steps as its order type. And this may be an issue as, while the machine checks whether some real describes a well-order in α steps, in the meantime, whichever real x is written on the output produces a x-segment of length α. And regularly producing such parasitic segments in the output history may really well skew what the machine actually K-writes.

Still, this issue can be easily bypassed: there is not need to check whether what is written in the simulation m_β actually is an ordinal. As we only care for its behavior to be correct when x_β appear, the machine can simply always work as if it was an ordinal. And truncating a well-order can be done in ω steps by working on increasingly long initial segment of the tape on which it is written. But this almost-issue shows how we the relatively fine aspect of K-writing (when compared to e.w.) forces us to be on our guards when devising such computations.

So, with this in mind, we consider the following machine m_α: it simulates m_β and at each step simulated in m_β (this takes finitely many steps in the main machine), it does the following: it does as if the real that is written on the output of m_β was a well-order, truncates it and copies it on its own output. This can be done in ω steps. Then, it waits for ω^2 steps. After those, it goes on with the simulation of m_β.

We claim that this machine K-writes α. First, writing x_α for the real obtained when truncating x_β below i, any x_β-segment of length ν in the history of m_β is transformed into an x_α-segment of length $\omega^2 \cdot \nu$. Similarly, any x_β-co-segment of length ν is transformed, at most (x_α may be produced from some $x' \neq x_\beta$), into a x_α-co-segment of length $\omega^2 \cdot \nu$. Then, the computation also produces new segments when it takes ω steps to truncate a real and those segments do not correspond to any segment in m_β. This is why we make those ω^2 breaks: to ensure that the new segments (of length $\omega^2 \cdot \nu$) are still big when compared to those parasitic segments of length ω. So with β_0 and ν_0 such that the history of m_β after stage ν_0 was x_β-populated w.r.t. β_0, the history of m_α after stage $\omega^2 \cdot \nu_0$ will be x_α-populated w.r.t. $\omega^2 \cdot \beta_0$. Conversely, if the history of m_α is x_α-co-populated after some ν_0 and w.r.t. some β_0 then, as the x_α-segment are all of length multiple of ω^2 on the left, we can w.l.o.g. suppose that $\beta_0 = \omega^2 \cdot \beta_0'$. And so, the history of m_β would be at

some point x_β-co-populated w.r.t β'_0, which would contradict the fact that x_β is K-written by m_β. Hence, as wanted, x_α (and so the ordinal α) is K-written by m_α. □

To gap this difficulty, namely the fact that we need to keep precisely track of the length of every subcomputation when describing a machine that K-writes some real, we provide in the next section a versatile lemma to design such computations.

6.2.4 A K-writing sufficient condition

We provide a sufficient condition for some reals to be K-written. This sufficient condition will constitute the following lemma which will be extensively used to establish the most interesting results on the operator Γ_3. The idea behind this lemma is akin to that of the proof of Proposition 6.2.35: when designing a computation that K-writes some real x, to avoid the x-co-segments that appear during ancillary subcomputations (previously called "parasitic segments") to change what the computation really K-writes, it is enough to make so that the x-segments are "big" when compared to those x-co-segments. And to do so, the idea is simply to arrange so that the x-segments the machine produces are bigger and bigger. The main lemma is initially formulated so that $\widetilde{x}$ is K-writable up to stage Σ_Γ, but we show in Proposition 6.2.42 that the machine we design also K-writes $\widetilde{x}$ in On.

Remember that, in Proposition 6.1.10, we showed that there exists a computable application $\cdot^* : \Sigma_3 \to \Sigma_3$ such that β^* is the β^{th} ordinal such that β^* is multiplicatively closed and it appears negatively encoded at time β^* on the tape of m_ζ.

Lemma 6.2.36 (Main lemma)**.** *A real $\widetilde{x}$ is* K*-writable up to stage Σ_Γ if there exists two Γ_3-computable function $S : {}^\omega 2 \times \Sigma(3) \to \Sigma(3)$ and $X : \Sigma(3) \to {}^\omega 2$, that will respectively be a length-of-segments function and an injective enumeration function, and such that, writing $S_x : \alpha \mapsto S(x, \alpha)$,*

1. *For all x and all α, $S(x, \alpha) \leqslant \alpha$ and for cofinally many α's in $\Sigma(3)$, $S_{\widetilde{x}}(\alpha) = \alpha$*
2. *$S_{\widetilde{x}}$ is monotonic (and non-decreasing per previous condition).*
3. *For some least $\widetilde{\nu} < \Sigma_3$, $\widetilde{x} = X(\widetilde{\nu})$ and there is some ordinal $\mathrm{B} < \Sigma_3$ such that for all $x = X(\nu)$ with $\nu < \widetilde{\nu}$, S_x is bounded by* B*.*
4. *For any ordinal $\alpha \in \Sigma_3$ there is $\mathrm{B}_\alpha < \alpha^\omega$ such that for any real $x = X(\nu)$ with $\nu < \alpha$, the computations of $S(x, \alpha)$ and of $X(\alpha)$ take less than B_α steps.*

Sketch of the proof. We will construct a machine that uses the computable functions S and X in order to K-write $\widetilde{x}$. S will be a length-of-segments function: given some real x such that the machine wants to write a x-segment at some stage α, it will compute $S(x, \alpha)$ and this will be, in most case, the length of the x-segment to be written. As for X, it will be used as an enumerating function: it will be used to go through the reals x

for which we want to write x-segments. It naturally induces an order on reals in $\mathrm{Im}(X)$ and we write $x <_X x'$ when $x = X(\nu)$ and $x' = X(\nu')$ for ν and ν' minimal and with $\nu < \nu'$. And the machine will order the reals for which it writes segments according to this partial order $<_X$.

From there, the machine works schematically as follows: using the universal Γ_3-machine it will loop through the ordinals below Σ_3. For cofinally many (well chosen) ordinals α, it will write segments for the α first reals produced by X. The length of those segments will be determined by the function S. The aim will be to write soon enough and big enough $\widetilde{x}$-segments in order to produce an history which is $\widetilde{x}$-populated w.r.t. some β_0, as depicted in Figure 6.3. Also, as depicted in Figure 6.4, those big enough $\widetilde{x}$-segments will be used to ensure that the $\widetilde{x}$-co-segments do not appear early enough for the history to be $\widetilde{x}$-co-populated w.r.t. any constants. More precisely:

- Condition 1 allows us to ensure that there are cofinally "long enough" $\widetilde{x}$-segment which will be the basis of our $\widetilde{x}$-populated segments.
- Condition 2 (with previous condition) will ensure that the length of the $\widetilde{x}$-segment will be unbounded in Σ_Γ and always great enough. This will prevent any other x coming after $\widetilde{x}$ (w.r.t. to the partial order induced by X) to generate cofinally in Σ_Γ $\widetilde{x}$-co-populated segments. This is because, with those growing $\widetilde{x}$-segments, the β_0 in the definition of $\widetilde{x}$-co-populated would need to be greater and greater.
- Condition 3 will ensure that the segments written before $\widetilde{x}$ (before w.r.t. to the partial order induced by X) will be bounded in length. This will be the β_0 of Figure 6.3.
- And Condition *4* will ensure that the machine is not caught up too long in a sub-computation that would produce parasitic segments in the history as in the proof of Proposition 6.2.35.

So, we will obtain, for some α's, cofinally in Σ_3, the computation history described in figure 6.3, for some fixed $\beta_0 < \Sigma_3$, equal to the sum of the bounds of the S_x for all x's appearing before $\widetilde{x}$; the reason for which we need those to be universally bounded in Σ_3. This will ensure that, after some point, the initial segments of the history are cofinally $\widetilde{x}$-populated w.r.t. this β_0. This gives the positive (or existential) half of the conjunction of the K-writing predicate (that is: $\exists \nu_0, \beta_0 \phi(\widetilde{x}, \beta_0, \nu_0)$). Moreover, for any x X-greater than $\widetilde{x}$ and such that $S_x(\alpha) = \alpha$ cofinally in α (as we will see in application of the lemma that we can't rule this cases out), we will rather obtain, for α's cofinal in Σ_3, the situation of Figure 6.4. In this situation, the history will only be a $\widetilde{x}$-co-segment w.r.t. at least, $\beta_0 + S(\widetilde{x}, \alpha)$. Hence, as $S(\widetilde{x}, \alpha)$ is unbounded in Σ_3, when α goes up to Σ_3, there won't be a universal β_0^{co} w.r.t. which the initial segments of the history are $\widetilde{x}$-co-populated. This will be the basis to ensure that the negative (or universal) half of the conjunction $(\forall \nu_0, \beta_0 \neg \phi^{co}(\widetilde{x}, \beta_0, \nu_0))$ of the K-writing predicate is satisfied.

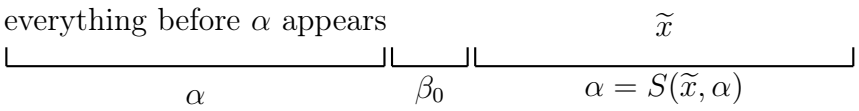


Figure 6.3: An initial segment which is $\widetilde{x}$-populated w.r.t. β_0.

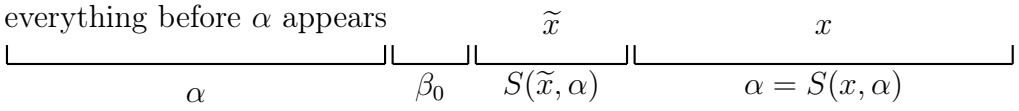


Figure 6.4: An initial segment which is $\widetilde{x}$-co-populated w.r.t. $\beta_0 + S(\widetilde{x}, \alpha)$ but not w.r.t. any β strictly smaller.

□

This subsection and the following results will be dedicated to proving this main lemma. As a reminder of what was done, we give the following special cases of previous propositions.

Proposition 6.2.37. *For any Γ_3-computable predicate P on the ordinals such that $P(\zeta(3))$, the set $\{\alpha \in \zeta(3) \mid P(\alpha)\}$ has order-type $\zeta(3)$ and the set $\{\alpha \in \Sigma(3) \mid P(\alpha)\}$ has order-type $\Sigma(3)$.*

Proof. Special case of Proposition 6.1.6. □

Proposition 6.2.38. *There exists a Γ_3-machine m_ζ that writes a negative encoding for ζ_3 for the first time at stage ζ_3.*

Proof. Special case of Proposition 6.1.8. □

And as done in Proposition 6.1.10 those two special cases yield the existence of Σ_3 star-ordinals below Σ_3 (in the sense of Definition 6.1.11, a star-ordinal α is multiplicatively closed and appears for the first time in m_ζ at stage α.) With this existence comes a computable application $\cdot^*$ that maps β to the β^{th} star-ordinal. What is very convenient is that star-ordinal are well-behaved. They are multiplicatively closed and they appear in an orderly fashion: by definition, for $\beta < \beta'$, β^* appears for the first time before $(\beta')^*$ does.

Lemma 6.2.39. *There does not exist $b_0 < \Sigma_3$ such that the length of the history intervals in which any ordinal found on the output of any Γ_3-machine is strictly smaller than b_0 are unbounded in Σ_3. That is, the other way around, for any b_0, there exists some ordinal $B_0 < \Sigma_\Gamma$ such that for all ν, there exists $\alpha \in [\nu, \nu + B_0[$ and some machine m such that $m(\alpha)$, the output of m at stage α, is an ordinal and $m(\alpha) \geqslant b_0$.*

Proof. Suppose there exists such a b_0 for which the length of the b_0-bounded-segments (that is the segments or the intervals of computation during which all ordinal accidentally

written are strictly smaller than b_0) is unbounded in Σ_3. The idea is, toward a contradiction, to design a computation that runs through greater and greater initial segments of On in order to have ordinals greater than b_0 appearing regularly enough in the computation.

As $b_0 < \Sigma_3$, b_0 is accidentally writable. So it appears for the first time at some stage α_0 in $\mathcal{U}_3$. By Proposition 6.2.17, $\alpha_0 < \Sigma_3$. Let $\alpha > \alpha_0$ be the least multiplicatively closed ordinal strictly greater than both α_0 and ω^ω. By our hypothesis, there is some earliest b_0-bounded-segment $[\nu, \nu'[$ of length greater than α. We consider the α first stages of this segment. First, for any stage $\iota \in [\nu, \nu + \alpha[$, the snapshot of $\mathcal{U}_3$ at stage ι appears in the computation of $\mathcal{U}_3$ for the first time in this segment. Indeed, suppose it appeared earlier at some stage $\iota' < \nu \leqslant \iota$. Then, the computation at stage ι acts like that at stage ι', by asymptoticity of the operator Γ_3. As $[\nu, \nu'[$ is the first b_0-bounded-segments longer than α, there would appear some ordinal greater than b_0 before stage $\nu + \alpha < \nu'$; a contradiction.

Now, we consider the following computation. We design a machine $\mathcal{M}$ that simulates in parallel two copies of $\mathcal{U}_3$, which we write $\mathcal{U}_3^1$ and $\mathcal{U}_3^2$. $\mathcal{U}_3^1$ never halts whereas $\mathcal{U}_3^2$ will be regularly halted and restarted. At any stage, if $\mathcal{U}_3^2$ is not running, $\mathcal{M}$ saves the snapshot of $\mathcal{U}_3^1$, starts $\mathcal{U}_3^2$ and runs it in parallel of $\mathcal{U}_3^1$ until the snapshot of $\mathcal{U}_3^2$ matches the one of $\mathcal{U}_3^1$ it saved. When this happens it reinitializes $\mathcal{U}_3^2$ to be started again on the next step. While this goes on, at any step, $\mathcal{M}$ writes on its output, one by one, all reals appearing in the outputs of all machines simulated by $\mathcal{U}_3^1$ and $\mathcal{U}_3^2$.

We claim that between stages ν and $\nu + \alpha$, some ordinal greater than b_0 appears in one of the simulations of $\mathcal{U}_3$ and so, just after, on the output of the machine we are designing.

However, this parallel computation and this copying of reals from one virtual tape to the output tape of $\mathcal{M}$ will slow the machine down, as with every simulation. But we have seen that it is possible to simulate ω steps of ω machines in ω steps of the simulating machine and so that at limit stages of the computation, the simulated machines are "on time". But to this, we need to add ω^2 steps needed to copy, at each of those stages, the $\omega \cdot 2$ reals appearing in $\mathcal{U}_3^1$ and $\mathcal{U}_3^2$. So, with the same reasoning, in $\mathcal{M}$, at any ordinal stage ι multiple of ω^ω, the virtual stage of the simulated machines in the simulations of $\mathcal{U}_3^1$ and $\mathcal{U}_3^2$ will coincide with ι.

Now for the claim, keeping this fact in mind. We distinguish two cases: in the computation interval $[\nu, \nu + \alpha[$ of $\mathcal{M}$, either $\mathcal{U}_3^2$ was looking for a single snapshot without finding it for all this time; either it found it. In the first case, this snapshot s of $\mathcal{U}_3^1$ that can't be found by $\mathcal{U}_3^2$ was saved by $\mathcal{M}$ at some earlier stage, that is at some stage $\nu_s < \nu$ of its computation. Also, still in this case, $\mathcal{U}_3^2$ computes for more than α steps (as $\mathcal{M}$ computes through stages $[\nu, \nu + \alpha[$ and $\alpha > \omega^\omega$ is multiplicatively closed) but less than ν_s steps (as at most, at stage ν_s of $\mathcal{U}_3^2$, s appears in it). So, the segment of computation of $\mathcal{U}_3^2$ we are considering is longer than α and entirely before stage ν. As the segment $[\nu, \nu + \alpha[$ was the first b_0-bounded-segments longer than α appearing in the universal machine, the b_0-bounded-segments in the computation of $\mathcal{U}_3^2$ are strictly bounded by α. Consequently, some real coding an ordinal greater than b_0 appeared in $\mathcal{U}_3^2$ through the computation

interval $[\nu, \nu + \alpha]$ of $\mathcal{M}$; as a result it appears also in the output of $\mathcal{M}$ in this interval, which is a contradiction.

Either, second case, $\mathcal{U}_3^2$ finds the snapshot s in its parallel computation while $\mathcal{M}$ is in the interval $[\nu, \nu + \alpha[$. Then, $\mathcal{M}$ saves a new snapshot s' appearing at stage $\nu_{s'} \in [\nu, \nu + \alpha[$. By the same argument of multiplicative closeness making up for the possible lateness of the simulated machines, this s' also appears in $\mathcal{U}_3^1$ between stages ν and $\nu + \alpha$. Moreover, as explained in the second paragraph of the proof, this snapshot can't appear before stage ν. Hence, looking for this new snapshot, $\mathcal{U}_3^2$ will compute for at least ν steps. While going through those ν steps, b_0 will appear in it as it appears at stage $\alpha_0 < \alpha \leqslant \nu + \alpha$ and so strictly before stage ν, as it does not appear in the interval $[\nu, \nu + \alpha[$. But then as α is multiplicatively closed, b_0 appears in the computation of $\mathcal{M}$ in the interval $[\nu, \nu + \alpha[$, which is again a contradiction. □

The next lemma will help the machine of the main lemma write the segments of history it needs to write in an organized way so that, for two reals y and y' different from $\widetilde{x}$ and for which S_y and S'_y are unbounded, the written y-segments and y'-segments (which are $\widetilde{x}$-co-segments) won't concatenate to form even longer and harder to control $\widetilde{x}$-co-segments.

Lemma 6.2.40. *We recall that an ordinal word of size λ on the alphabet* A *can be seen as a function $\lambda \to$ A. We show that for any countable ordinal α we can build a word W_α on α seen as an alphabet such that, for all $\beta < \beta' \leqslant \beta''$ in α and between all occurrences of β' and β'' in W_α, there is always an occurrence of β. That is, for $\iota' < \iota''$, if $W_\alpha[\iota'] = \beta'$ and $W_\alpha[\iota''] = \beta''$, then there exists $\iota \in [\iota', \iota'']$ such that $W_\alpha[\iota] = \beta$.*

Proof. We consider the function f, from α to words on α, recursively defined as:

$$\begin{aligned} f(0) &= \varepsilon \\ f(\beta + 1) &= f(\beta) \cdot \beta \cdot f(\beta) \\ f(\beta) &= \lim_{\beta' < \beta} f(\beta') \text{ when } \mathrm{Lim}(\beta) \end{aligned}$$

This means that the word $f(\beta + 1)$ is the word that starts with the word $f(\beta)$ as prefix, then has β as next letter and then finishes with $f(\beta)$. For this definition to be licit, we need to show that $\lim_{\beta' < \beta} f(\beta')$ is well-defined. To this effect, we show, by induction, that for any $\beta' < \beta$, $f(\beta') \sqsubset f(\beta)$, that is that $f(\beta')$ is a prefix of $f(\beta)$. For β' (if any) such that $\beta' + 1 = \beta$, this is immediate. Then for any $\beta' < \beta$, if β is not a limit ordinal, by induction hypothesis, $f(\beta') \sqsubset f(\beta - 1) \sqsubset f(\beta)$. If β is a limit ordinal, then, by induction hypothesis, the $f(\beta')$ for $\beta' < \beta$ are well-defined and for $\beta'' < \beta' < \beta$ we have $f(\beta'') \sqsubset f(\beta')$. Hence, $f(\beta)$ is well-defined and, by construction, for any $\beta' < \beta$, $f(\beta') \sqsubset f(\beta)$.

Finally, we verify that $W_\alpha = f(\alpha)$ is the word on the alphabet α we wanted to build. Let $\beta < \beta' \leqslant \beta''$ in α. Any occurrence of β' is surrounded on either side by the word

$f(\beta')$ on the alphabet β'. That is, all letters in $f(\beta')$, seen as ordinals, are smaller than β'. Moreover, all ordinals smaller than β', seen as letters, appears at least once in $f(\beta')$. Hence $\beta < \beta'$ must appear between any two occurrences of β' and β''. □

Example 6.2.41. With f as previously defined : $f(4) = W_4$ yields, with concatenation being alternatively implicit or denoted with a dot, for better readability,

$$010 \cdot 2 \cdot 010 \cdot 3 \cdot 010 \cdot 2 \cdot 010$$

From those results, we can now describe the machine of the main lemma and show that it K-writes $\widetilde{x}$.

Proof of main lemma. Given $\widetilde{x}$, S and X as defined in the description of the lemma, we build a machine m that K-writes $\widetilde{x}$.

Observe first that given any encoding for an ordinal α, a machine can compute and store an encoding for the word $W_\alpha = f(\alpha)$, as defined in Lemma 6.2.40. Indeed, using the code for α, any $\beta < \alpha$ can be encoded by an integer and then W_α, which is an ordinal word on the alphabet α, can be encoded as an ordinal word on the alphabet ω. This word has length at most 2^α ($2^\alpha - 1$ when α is finite), which is α itself when α is limit and is in general less than $\alpha \cdot \omega$. Hence, it can be encoded using a code for $\alpha \cdot \omega$, one of those being easily computable from the code of α. Also, all in all and being generous, the computation of W_α takes less than $\alpha \cdot \omega^2$ steps.

Recall also that in Proposition 6.1.10 we designed an application $\beta \mapsto \beta^*$ that maps β to the β^{th} multiplicatively closed ordinal α such that α appears m_ζ (in bitwise negative encoding) at stage α. As seen in the proof of Proposition 6.1.12, given some a.w. α, a machine can check in $\alpha^2 + \alpha$ steps whether $\alpha = \beta^*$ for some β.

Then, the machine works as follows. It begins with the simulation of $\mathcal{U}_3$. This way it loops through, in an unknown order, all the ordinals below Σ_3. In this simulation, it looks for the appearance of star ordinals $\alpha = \beta^*$. And it uses those ordinals and the regularity of their first appearance (β^* appears for the first time in m_ζ at time β^*) to structure the history of the output tape of the machine. It will moreover use paddings to control this structure even more precisely. Paddings can be seen as words on reals (e.g. given three reals x, y and z, the three letter words xyz could be a padding), that the machine will write, letters by letters (that is reals by reals), on its output tape in order to add them to the output history. However by padding its output history with poorly chosen reals, the machine will produce new segments of reals in its history which may change what the computation K-writes. Indeed, padding the output history with, say, the word y^α creates a y-segment of length α and consequently an y-populated interval of length at least α. For this reason, we use the word W_α to construct our padding. We write $X_\alpha = X(W_\alpha)$ where X is applied letter by letter; that is, for an index ι, $X_\alpha[\iota] = X(W_\alpha[\iota])$. By construction of W_α, the length of the $\widetilde{x}$-co-segments in any X_α's are bounded by $2^{\widetilde{\nu}}$ where $\widetilde{\nu}$ is the least

index of $\widetilde{x}$ in X. Observe that $\widetilde{x}$ may not be a part of X_α but in this case, this mean that $\alpha < \widetilde{\nu}$ and so the claim still hold. And this is enough to control the segments of reals, and more particularly the $\widetilde{x}$-co-segments, appearing in this padding.

Further, it is clear that this padding may easily be truncated or concatenated to create greater paddings. We will use this in the first part of the algorithm. While doing some subcomputation, the algorithm will write in parallel a padding using some X_α. That is, one after the other, it will do one step of the subcomputation and then write one real of X_α on its output. However, letter by letter, it may go through all of X_α. In this case, it simply goes back to the beginning and continues from there, effectively "looping" over X_α to write the padding. Those parallel computation and padding writing are denoted by the command `do in parallel...while...` where the first instruction contains a padding writing which is done, in parallel, for as long as the second set of instruction takes.

Now, we will need to pay attention at which $\widetilde{x}$-co-segments may be created in the output while doing so. If $\alpha > \widetilde{\nu}$, then only the $\widetilde{x}$-co-segments of the extremities of the X_α may concatenate to form, at most, twice as long $\widetilde{x}$-co-segments; and so this case won't be an issue. As for the other case, we will ensure that paddings X_α for $\alpha < \widetilde{\nu}$ won't be written too many times in a row.

Moreover, this pattern will also be used for the writing of the different x-segments. That is, to write those, for each letters x of X_α, in the same order and with the same repetitions, the machine will write a x-segment, taking once more advantage of the fact that we can control the size of the $\widetilde{x}$-co-segments it thus creates.

From there, we describe the machine in its entirety in Algorithm 1.

Claim 6.2.41.1. *We claim that*

1. *For any β below Σ_3, β^* appears at least once in the loop of line **1***
2. *Computation from lines **2** to **9** takes at most than $\alpha^{\omega+1} + \alpha^\omega$ steps and strictly less than α^ω when α is a star-ordinal.*
3. *For α not a star-ordinal, the parallel computation of line **10** to **20** takes $\alpha^2 + \alpha$ steps. And so it writes a padding of length $\alpha^2 + \alpha$ steps using W_α.*
4. *For a star-ordinal $\alpha = \beta^*$, the parallel computation of line **10** to **20** takes α^ω steps. And so it writes a padding of length α^ω steps using X_α.*
5. *The computation of line 21 to 29 takes at most $\alpha^\omega + \alpha$ steps.*
6. *When some star-ordinal $\alpha = \beta^*$ comes for the first time in the computation at line **1**: after completing the parallel computation of line **20**, the whole history before it has length α^ω.*
7. *The length of the $\widetilde{x}$-co-segment appearing between lines **2** and **20** is bounded in Σ_3.*

```
1  foreach α appearing on the output of m_ζ do
2  |  foreach α′ ⩽ α do
3  |  |  do in parallel
4  |  |  |  write a padding on the output looping over X_α′
5  |  |  while
6  |  |  |  compute and store X(α′) on some part of the working tape
7  |  |  |  compute and store X_α′+1
8  |  |  end
9  |  end
10 |  do in parallel
11 |  |  write a padding on the output looping over X_α
12 |  while
13 |  |  if α is a star-ordinal then
14 |  |  |  find β such that β* = α
15 |  |  |  compute and store S(x, β) for all x appearing in X_α
16 |  |  |  wait for α^ω steps.
17 |  |  else
18 |  |  |  continue // ignore this α and continue with the loop of
   |  |  |   line 1
19 |  |  end
20 |  end
21 |  forall x in the word X_α do // that is, while reading X_α letter by
   |   letter
22 |  |  if (S(x, β) < β) then
23 |  |  |  write on the output a x-segment of length S(x, β)
24 |  |  else // (S(x, β) = β)
25 |  |  |  write on the output a x-segment of length α^ω
26 |  |  |  write on the output the end of X_α // that is writing, letter by
   |  |  |     letter, the final segment of X_α that wasn't read
   |  |  |     through in the loop of line 21
27 |  |  |  break // exit the current loop of line 21
28 |  |  end
29 |  end
30 end
```

Algorithm 1: Main loop of the machine.

Proof of claim.

1. This comes from the fact that the set $\{\beta^* < \Sigma_3\}$ has order-type Σ_3 and that for $\beta < \beta'$ we have $\beta^* < \beta'^*$; hence for all $\beta < \Sigma_3$, $\beta^* < \Sigma_3$ and β^* is accidentally writable.

2. Using α to store the information and by Condition *4*, for $\alpha' \leqslant \alpha$, the computation of $X[\alpha']$ takes strictly less than $(\alpha')^\omega$ steps. Then, using $X[\alpha']$ and $X_{\alpha'}$, the computation of $X_{\alpha'+1}$ also takes less than $(\alpha')^\omega$ steps. And when α' is limit, $X_{\alpha'}$ is the limit of the $(X_{\alpha''})_{\alpha''<\alpha'}$ and so a code for it is computed in α' steps. The sum of the $(\alpha')^\omega$ (or α' for α' limit) for $\alpha' < \alpha$ is less or equal to $\alpha^{\omega+1}$. Hence, this same sum for $\alpha' \leqslant \alpha$ yields at most $\alpha^{\omega+1} + \alpha^\omega$ steps.

   When α is a star-ordinal, for all $\alpha' < \alpha$, $(\alpha')^\omega \leqslant \alpha$ and so their sum is strictly less than α^ω. As for the last iteration of the loop, that is when $\alpha' = \alpha$, it takes by Condition *4* strictly less than α^ω steps. So the whole loop also takes strictly less than α^ω steps.

3. By Proposition 6.1.12, it takes $\alpha^2 + \alpha$ steps to detect that α is not a star-ordinal.

4. We give line by line the time complexity of the different operations in the second half of the parallel computation. The first half, writing the padding, simply goes on until the second half is done.

   line **13**: it takes $\alpha^2 + \alpha$ steps to detect that α is a star-ordinal.

   line **14**: Then, to compute β such that $\beta^* = \alpha$, the machine makes the list of the first star-ordinals (using α to arrange the information) until it reaches stage α. This takes less than $\nu^2 + \nu$ steps for each ordinal ν below α and so in the end, by multiplicative closedness of α, exactly α steps.

   line **15**: By construction of X_α, to compute $S(x, \beta)$ for all x appearing in X_α, it needs to compute S for less than α different x's of X-index strictly less than α. By Condition *4* in the main lemma, each of those computation takes less than $\mathrm{B}_\alpha < \alpha^\omega$ steps. As α^ω is multiplicatively closed, this makes for strictly less than α^ω steps.

   Hence, after those three instructions, the parallel computation went for strictly less than α^ω steps. It then waits for α^ω steps in line **16** which brings the total to α^ω .

5. As after everything has been computed before line 20, there is no unbounded overhead (the usual small and fixed overhead is negligible when compared to α^ω) for any of the operations after this line. Hence, either for all x in X_α, $S(x, \beta) < \beta$ and the whole loop takes less than α^2 steps ($\beta \leqslant \alpha$ and as α is multiplicatively closed, the length of X_α is α itself.) Either, for some x, $S(x, \beta) = \beta$ and the machine outputs a segment of length α^ω before outputing the end of X_α. This takes $\alpha^\omega + \alpha$ steps.

6. With Item 2 and Item 4 of the claim in mind, it is enough to show that when such a star-ordinal α appears at line **1**, the whole history before it has length strictly less than α^ω. To see this, observe that under the assumption that the star-ordinal α appears for the first time, it does so at stage α in m_ζ. And so the loop of line **1** has been done at most α times. Moreover any ordinal appearing before this stage in m_ζ must be strictly less than α otherwise, with the usual truncation technique, α would appear before stage α. And so, for each $\delta < \alpha$, by Items 2, 4 and 5, it takes at most $(\delta^{\omega+1} + \delta^\omega) + \delta^\omega + \delta^\omega + \delta$ steps. And so, as α is multiplicatively closed, for any such δ, $\delta^\omega \leqslant \alpha$ and it takes in the end strictly less than α^ω steps, as required.

7. The first $\widetilde{x}$-co-segments that may appear will be during the execution of lines **2** to **9**. This is why we use paddings which will, in parallel from execution of lines **5** to **29**, write different words, one after the other, on the output. Once α' is greater than $\widetilde{\nu}$ (the least ν s.t. $X(\nu) = \widetilde{x}$), every padding X'_α will contain $\widetilde{x}$. And by construction of X'_α the $\widetilde{x}$-co-segments in $X_{\alpha'}$ are bounded by $\widetilde{\nu}$. And those $X_{\alpha'}$ are written in parallel of the computation of $X[\alpha']$. That is as the machine does one step of the computation of $X[\alpha']$ and then writes one new letter of $X_{\alpha'}$ (a letter of $X_{\alpha'}$ is a word in ${}^\omega 3$) on the output, and so on. Doing so, it will regularly write $\widetilde{x}$ in the output which will ensure that the $\widetilde{x}$-co-segments stay small. It may also write X'_α many time in a row to write a padding of greater length. In this case, the length of the $\widetilde{x}$-co-segment between two X_α's will reach $\widetilde{\nu} \cdot 2$ which is still a bounded quantity. And so, once α' is greater than $\widetilde{\nu}$, the length of the $\widetilde{x}$-co-segment is controlled. Before this, that is before reaching $\alpha' = \widetilde{\nu}$, it takes, by Condition *4*, as much as $\widetilde{\nu}^\omega \cdot \widetilde{\nu}$. But, again, this is a fixed quantity less than Σ_3.

   Then, X_α will be used as a padding for the computation from lines **10** to **20**. That is, in parallel of lines **12** to **20**, the machine writes one by one the letters of X_α, eventually looping through it if it reaches its end. With the same reasoning, the length of the $\widetilde{x}$-co-segments is easily seen to be bounded.

■

Proof of the Σ_3 part of the K-writing condition We now prove that the output history of the computation below stage Σ_Γ respects the first half of the K-writing condition: $\exists \nu_0, \exists \beta_0\, \phi(\widetilde{x}, \beta_0, \nu_0)$.

We show how, as wanted, we obtain the situation depicted in Figure 6.5, for cofinally many α's in Σ_3. Indeed, by Condition 1 of the lemma, there exists cofinally many β's such that $S_{\widetilde{x}}(\beta) = \beta$. We look at what happens for such a β. By Item 6 of Claim 6.2.41.1, after $\alpha = \beta^*$ appeared for the first time at line **1**, at line **20** of the algorithm the history has length α^ω. This is depicted in the first horizontal bracket of the figure.

Then, the machine will write x-segments for all $x <_X \widetilde{x}$, and more than one for each x, by construction of X_α. For all those x it writes a x-segment of length $S(x, \beta)$. That

is, it writes x on the output and then waits for $S(x, \beta)$ steps. By Condition 3, there is B that universally bounds the S_x for $x <_X \widetilde{x}$. Hence, for β big enough, for all $x <_X \widetilde{x}$, we have : $S(x, \beta) \leqslant \mathrm{B} < \beta$. Observe that as X_α was previously written on some part of the working tape, going through X_α is done with a bounded overhead: we can arrange so that the machine finds the next x and writes it on the output tape in $\omega \cdot 2$ steps. Further, $S(x, \beta)$ has also been computed beforehand and counting through it does not incur more than a small and fixed overhead.

Hence, the time it takes to write those x-segments for $x <_X \widetilde{x}$ may be slightly more than $\mathrm{B} \cdot 2^{\widetilde{\nu}}$ where $\widetilde{\nu}$ is the least X-index of $\widetilde{x}$. The important point being that, as B and the various overheads are constant, this produces a bounded (universally bounded in Σ_3 for any β such that $S_{\widetilde{x}}(\beta) = \beta$) $\widetilde{x}$-co-segments and we write β_0 for the associated least upper bound. Moreover, as we are interested in the asymptotical behavior (again in Σ_3) of our machine, we can suppose w.l.o.g. that $\beta_0 < \alpha$. This bounded sequence made of x-segments for $x <_X \widetilde{x}$ and various bounded overheads is depicted in the second horizontal bracket.

Eventually, after going through all the $x <_X \widetilde{x}$ in X_α, the machine reaches $\widetilde{x}$. As $S_{\widetilde{x}}(\beta) = \beta$, the condition of line **22** is false and it writes on its output a $\widetilde{x}$-segment of length α^ω. And, as appears clearly in Figure 6.5, this produces an initial segment of the output history of length $\alpha^\omega \cdot 2$ which is $\widetilde{x}$-populated w.r.t. β_0. And by Condition 1, those $\widetilde{x}$-populated initial segments appear cofinally in Σ_Γ for bigger and bigger α's but with the same β_0, which is enough to satisfy the positive half of the K-writing condition in Σ_Γ.

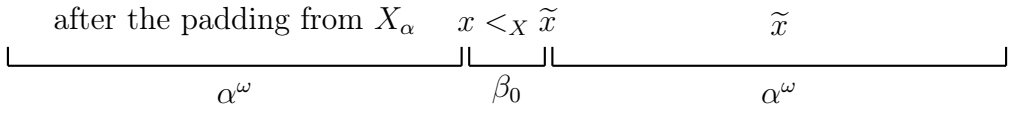


Figure 6.5: An initial segment of the output history which is $\widetilde{x}$-populated w.r.t β_0.

Proof of the Π_3 part of the K-writing condition Now we prove that the output history of the computation respects the second half of the K-writing condition: $\forall \nu_0, \beta_0 \, \neg \phi^{co}(\widetilde{x}, \beta_0, \nu_0)$. The situation, still in Figure 6.5, will give us the witnesses for the negation of $\phi^{co}(x, \beta_0, \nu_0)$. Indeed, for any $x' >_X \widetilde{x}$, there isn't any condition on $S_{x'}$ apart from the fact that $S_{x'}(\beta) \leqslant \beta$ and for some x' there may also be cofinally many β's in Σ_Γ that are fixed point for $S_{x'}$. Consequently, this may give rise to unboundedly long x'-segment which also are $\widetilde{x}$-co-segment; that is segments of the history of the output tape in which $\widetilde{x}$ does not appear. Still, we will show that despite those unbounded x'-segments, the history can't be cofinally $\widetilde{x}$-co-populated. Let ν_0 and β_0^{co} be two ordinals below Σ_Γ. We show that there exists $\nu > \nu_0$ after which we can't find ν' and ν'' describing a $\widetilde{x}$-co-segment that satisfies the $\widetilde{x}$-co-population property w.r.t. β_0^{co}.

Indeed, take some star-ordinal $\alpha = \beta^* > \nu_0$ for which arises the situation of Figure 6.5. As seen, such situations occur cofinally in Σ_3. Then set $\nu = \alpha^\omega + \beta_0$. That is ν is

the index of the beginning of the $\widetilde{x}$-segment of length α^ω, depicted by the third horizontal bracket. Consequently, any $\nu' > \nu$ starting a $\widetilde{x}$-co-segment is greater than $\nu + \alpha^\omega$; in other words for such a ν', $|[\nu, \nu'[| \geqslant \alpha^\omega$. And to satisfy the co-population property, $[\nu', \nu''[$ must in turn be a $\widetilde{x}$-co-segment whose length, to which we add the fixed β_0^{co}, is greater than α^ω. As β_0^{co} is fixed, we can w.l.o.g. suppose that it is smaller than α^ω. As α^ω is multiplicatively closed, $|[\nu', \nu''[| + \beta_0^{co} \geqslant \alpha^\omega$ is equivalent to $|[\nu', \nu''[| \geqslant \alpha^\omega$. And we claim that such a $\widetilde{x}$-co-segment, that is of length greater than α^ω, can't appear before reaching some $\alpha' = (\beta')^* > \alpha$ in the loop of line **1**. Indeed, $\widetilde{x}$-co-segment may appear from three different ways:

- They may come from intervals in the computation of $\mathcal{U}_3$ in which all α' that appear are such that α' is smaller than $\widetilde{\nu}$. However, by Lemma 6.2.39, the length of the intervals during which all ordinals appearing in $\mathcal{U}_3$ are bounded by $\widetilde{\nu}$ is bounded in Σ_3. Hence, for α big enough, we won't have α^ω iterations of the loop of line **1** without $\widetilde{x}$ appearing in some padding $X_{\alpha'}$.

- Then they may occur when the machine computes from lines **2** to **20**. But by Item 7, the $\widetilde{x}$-co-segment those lines generates are bounded in length and so, w.l.o.g., smaller than α.

- Finally, it may come from the x'-segments that the machine writes for $x' \neq \widetilde{x}$ in the loop of line **21**. However, this loop is only executed for star-ordinals. And before reaching a greater star-ordinal (that is before reaching $(\beta+1)^*$, as, by construction, the times of first apparition of the star-ordinals are ordered), all star-ordinal α' appearing are strictly smaller than $(\beta+1)^*$, and so smaller than $\alpha = \beta^*$. Hence for those, it writes x'-segments strictly smaller than α^ω. And even if α appears again in the loop of line **1**, the machine breaks out of the loop of line **21** once it reaches $\widetilde{x}$. That is, for this α, it only loops–besides $\widetilde{x}$–through the $x <_X \widetilde{x}$ and by Condition 3, for all those x's, S_x is universally bounded in Σ_3, hence w.l.o.g. strictly bounded by α^ω.

So for the specific ν we fixed (but which may have been chosen arbitrarily big in Σ_Γ), we may find witnesses ν' and ν'' for the $\widetilde{x}$-co-population only after some star-ordinal $\alpha' = (\beta')^* > \alpha$ is reached in the computation. This would give rise to the output history schematized in Figure 6.6. When α' is reached for the first time at line **1**: by multiplicative closedness of α', the tape history has length α'. As $\nu < \alpha'$, for any ν' greater than α', the history segment $[\nu, \nu']$ has length at least α'. And so, we are now looking for $\widetilde{x}$-co-segments of length α'. By Item 7, no such co-segments appears before line **20**. But because the padding of length $(\alpha')^\omega$, the co-segment now needs to be of length $(\alpha')^\omega$ By Condition 3, no such co-segment may appear when writing x-segments in the loop of line **21** for $x \leqslant \widetilde{x}$. It may appear, only after a $\widetilde{x}$-segment of length $S(\widetilde{x}, \beta')$ has been written, if there is some $x' >_X \widetilde{x}$ such that $S(x', \beta') = \beta'$. This would indeed give rise to a $\widetilde{x}$-co-segment slightly

longer than α'^{ω}. But how much longer? Not so much as, after writing this x'-segment of length $(\alpha'+1)^*$, the machine writes the end of the word X_α, letter by letter on the output. Hence $\widetilde{x}$ appears before $\widetilde{\nu}$ letters in this padding and cuts the $\widetilde{x}$-co-segment. So the $\widetilde{x}$-co-segment has at most length $(\alpha')^\omega + \omega \cdot \widetilde{\nu}$, as it takes ω steps to write each letter of X_α on the output tape.

Hence, supposing w.l.o.g. (by monotonicity and unboundedness of $S_{\widetilde{x}}$) $\omega \cdot \widetilde{\nu}$ negligible with respect to $S(\widetilde{x}, \beta')$, the initial segment this describes will be $\widetilde{x}$-co-populated segment w.r.t., at best, $S(\widetilde{x}, \beta')$. And by the convenient behavior of the star-ordinals (they appears for the first time in an orderly manner), this situation of Figure 6.5 occurs for cofinally many α's in Σ_3. And so, by Condition 2, we can choose ν such that for all star-ordinal $\alpha' = (\beta')^*$ appearing after this ν, $S(\widetilde{x}, \beta') > \beta_0^{co}$; that is some ν after which the history is never $\widetilde{x}$-co-populated w.r.t. β_0^{co}. Developing the definition of $\phi^{co}(\widetilde{x}, \beta_0^{co}, \nu_0)$, this yields: for any ν_0 and β_0^{co}, there is some stage ν at which the situation of Figure 6.5 occurs and so for any $\widetilde{x}$-co-segment occuring after it, the history from ν_0 will be $\widetilde{x}$-co-populated w.r.t. at best $S(\widetilde{x}, \beta') > \beta_0^{co}$; hence $\neg\phi^{co}(\widetilde{x}, \beta_0^{co}, \nu_0)$. And this concludes the proof of the lemma.

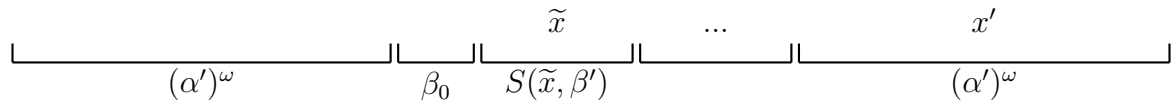


Figure 6.6: An initial segment of the output which is not $\widetilde{x}$-co-populated w.r.t to less than $S(\widetilde{x}, \beta')$.

□

Proposition 6.2.42. *The machine described in the previous lemma not only* K*-writes* $\widetilde{x}$ *up to stage* Σ_Γ *but it* K*-writes* $\widetilde{x}$ *in* L*.*

Proof. As was observed in Remark 5.4.33, it is not clear whether, for an operator Γ such that $\Sigma_\Gamma = \mathrm{T}_\Gamma$, any Γ-machine is seen to be looping at or before stage Σ_Γ. Still, by Proposition 5.5.10, any Γ-machine is seen to be looping before stage $\Sigma_\Gamma \cdot \omega^\omega$ and its history after stage Σ_Γ will be rather structured.

To see this, we take the case where the Γ_3-machine from the main lemma it is seen to be looping at stage $\Sigma_3 \cdot \omega^2$. This happens if there a segment H, spanning between two occurrences of some x at stages α and Σ_3, which produces a y at stage $\Sigma_3 \cdot \omega$ and which y then induces a repeating segments (as $\Sigma_\Gamma = \mathrm{T}_\Gamma$, this y must also have appeared before stage Σ_3) which yields another y at stage $\Sigma_3 \cdot \omega^2$. This is represented in Figure 6.7.

In this case, the looping segment is $H_y H^\omega$. And the length of H_y is bounded in Σ_Γ. Hence, it is clear that the whole history of the machine is $\widetilde{x}$-populated: take $\nu \in \Sigma_3 \cdot \omega$ greater than β, the ordinal stage at which H_y starts. Either ν, as a computation stage, is in some H. Then, as seen in the proof of the main lemma, there is some ν' and ν'' that satisfy the definition of $\widetilde{x}$-population w.r.t. this ν and the fixed β_0 of the proof. Or ν is in some H_y. Then, comes in the following H a $\widetilde{x}$-segment of length $(\beta^*)^\omega$ at the latest at

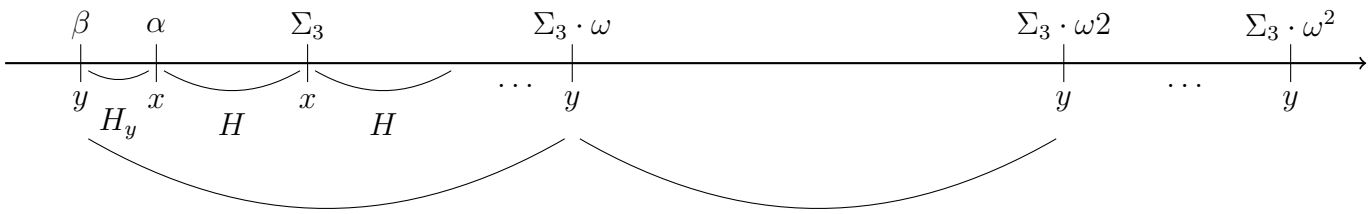


Figure 6.7: A Γ_3-machine seen to be looping at stage $\Sigma_3 \cdot \omega^2$.

stage $\nu + |H_y| + (\beta^*)^\omega + \beta_0$ and such that w.l.o.g. $(\beta^*)^\omega$ is strictly greater than $|H_y|$. So, by multiplicative closedness, there is a $\widetilde{x}$-segment of length $(\beta^*)^\omega$ appearing after $(\beta^*)^\omega + \beta_0$ steps after ν. And so it satisfies the $\widetilde{x}$-population formula for this ν w.r.t. β_0.

As for the fact that the whole history of the machine isn't $\widetilde{x}$-co-populated, it is enough to consider, as in the proof of the main lemma a stage ν in some H in which a $\widetilde{x}$-segment of length $(\beta^*)^\omega$ appeared. Indeed, the best candidate for ν' and ν'' in the definition of $\widetilde{x}$-co-population are in the same H as otherwise they would need to describe a $\widetilde{x}$-co-segment of length at least Σ_Γ. So, again, it works as in the proof of the main lemma.

Further, the same reasoning applies when the machine is not seen to be looping at stage $\Sigma_\Gamma \cdot \omega^2$ but at some stage $\Sigma_\Gamma \cdot \omega^k$ for $k \in \omega$. And so, the whole history of the machine satisfies the predicate of K-writing for $\widetilde{x}$, which simply mean that the machine K-writes $\widetilde{x}$ in L.

□

Lemma 6.2.43 (Auxiliary lemma)**.** *Let $\widetilde{x}$ be a real that satisfies the conditions of the main lemma and $\widetilde{y}$ that is $\widetilde{x}$-e.w., i.e. that is e.w. by a Γ_3-machine with $\widetilde{x}$ as output. Then y is* K*-writable as well.*

Proof. Let S and X be the computable predicates associated with x in the statement of the main lemma. We show that using them we can define S' and X' two computable predicates that also satisfy the conditions of the main lemma for the real $\langle \widetilde{x}, \widetilde{y} \rangle$.

We write $m_{\widetilde{y}}$ the machine that e.w. writes $\widetilde{y}$ from $\widetilde{x}$. Also, for any real x' and ordinal α, we write $m_y[x'](\alpha)$ for the output at stage α of the computation of $m_{\widetilde{y}}$ with input x'. As $m_{\widetilde{y}}$ e.w. $\widetilde{y}$ from $\widetilde{x}$, there is some ordinal stage A such that for all $\alpha >$ A, $m_y[\widetilde{x}](\alpha) = \widetilde{y}$. Finally, we can suppose that our pairing function $\langle \cdot, \cdot \rangle$ is injective and so that any real z can be written $\langle x, y \rangle$. Typically, it can be done by taking x as the even bits of z and y as the odd ones. With this, S' and X' are defined as follow.

$X' : \alpha \mapsto \langle X(\alpha), m_y[X(\alpha)](\alpha) \rangle$
$S' : \langle x, y \rangle, \alpha \mapsto \min(S(x, \alpha),$ length of the greatest y-segment before α in the computation of $m_{\widetilde{y}}[x])$

Let us show that it satisfies the condition of the lemma.

1. It is clear that for all $\langle x, y \rangle$ and α, $S'(\langle x, y \rangle, \alpha) \leqslant \alpha$. Moreover, for α greater than

A^ω (which is additively closed) and such that $S(\widetilde{x},\alpha)=\alpha$, we have $S'(\langle\widetilde{x},\widetilde{y}\rangle,\alpha)=\alpha$.

2. Monotonicity of $S'_{\langle\widetilde{x},\widetilde{y}\rangle}$ is clear too from that of the two functions whose minimum is computed.

3. As for the universal bound on the $S'_{\langle x,y\rangle}$ for $\langle x,y\rangle <_{X'} \langle\widetilde{x},\widetilde{y}\rangle$: the inequality $\langle x,y\rangle <_{X'} \langle\widetilde{x},\widetilde{y}\rangle$ implies $x <_X \widetilde{x}$, so the $S'_{\langle x,y\rangle}$ are bounded by the universal bound on the S_x for $x <_X \widetilde{x}$.

4. By hypothesis on X, computing $X(\alpha)$ takes less than $B_\alpha < \alpha^\omega$ steps. Then, from $X(\alpha)$, computing $m_y[X(\alpha)](\alpha)$ takes another α steps, and so it still yields a bound strictly below α^ω. Same goes for S'.

This shows that $\langle\widetilde{x},\widetilde{y}\rangle$ is K-writable. And by monotonicity of the K-writing predicate (see Definition 6.2.30), the machine that writes y instead of $\langle x,y\rangle$ in the paddings and in the segments K-writes $\widetilde{y}$.

□

From there, we can improve the main lemma. More precisely, we show how we can remove a constraint that can often be bothersome.

Lemma 6.2.44 (Improved main lemma)**.** *The Main Lemma 6.2.36 still holds if we don't ask for the functions $(S_x)_{x<_X\widetilde{x}}$ to be universally bounded in Σ_3 but simply one by one bounded and if we also ask the bounds B_α to be montonous. That is, it still holds if we replace Condition 3 by the following weaker Condition 3a and if we strengthen Condition 4 into the following Condition 4a.*

3a. For some least $\widetilde{\nu}<\Sigma_3$, $\widetilde{x}=X(\widetilde{\nu})$ and for all $x=X(\nu)$ with $\nu<\widetilde{\nu}$, S_x is bounded in Σ_3.

4a. For any ordinal $\alpha\in\Sigma_3$ there is $\mathrm{B}_\alpha<\alpha^\omega$ such that for any real $x=X(\nu)$ with $\nu<\alpha$, the computations of $S(x,\alpha)$ and of $X(\alpha)$ take less than B_α steps. Moreover, we ask that if $\alpha'<\alpha$ then $B_{\alpha'}\leqslant B_\alpha$.

Proof. We show that with such hypothesis, that is with Conditions 1, 2, 3a and 4a, we can still apply the main lemma, albeit with different functions and a different real. Then we show how, with this new real that is K-written and with Auxiliary Lemma 6.2.43, we can K-write $\widetilde{x}$.

So, let $\widetilde{x}$, X and S satisfying Conditions 1, 2, 3a and 4a. If those objects also satisfy Condition 3, as Condition 4a implies Condition 4, we can simply apply the main lemma. Else, it means that the functions $(S_x)_{x<_X\widetilde{x}}$ are not universally bounded. In particular, this means that there is a X-least $x_1 <_X \widetilde{x}$ for which S_{x_1} is not strictly bounded by $\widetilde{\nu}=\mathrm{rk}_X(\widetilde{x})$, the X-rank of $\widetilde{x}$. So we let α' be the least ordinal such that $S(x_1,\alpha')>\widetilde{\nu}$.

Then we consider the following segment function $S^{(1)}$: given x and α, $S^{(1)}$ first computes $S(x,\alpha)$ and then it looks for all x' of X-rank less than $S(x,\alpha)$ and returns the maximum of the $S(x',\alpha)$. That is,

$$S^{(1)}(x,\alpha) \coloneqq \max_{\mathrm{rk}_X(x')<S(x,\alpha)}(S(x',\alpha))$$

Now if the functions $(S^{(1)}_x)_{x<_X x_1}$ are universally bounded and we can apply the main lemma with x_1 instead of $\widetilde{x}$. Indeed, observe first that

- For α big enough, $S(\widetilde{x},\alpha)=\alpha$ implies $S^{(1)}(x_1,\alpha)=\alpha$.
- The monotonicity of $S^{(1)}$ comes from that of S.
- For $x <_X x_1$, S_x is strictly bounded by $\widetilde{\nu}$ and so $S^{(1)}_x$ is bounded in Σ_3 as well. We also supposed that they were universally bounded.
- For the time complexity: as $S(x,\alpha)\leqslant\alpha$, the maximum of S^α is computed in the worst case on all reals x' with $\mathrm{rk}_X(x')<\alpha$. And for those, the computation of $X(\alpha')=x'$ (for the enumeration of the x') and of $S(x',\alpha)$ takes strictly less than $\mathrm{B}'_\alpha+\mathrm{B}_\alpha$ steps, which is also less than $\mathrm{B}_\alpha\cdot 2$ by Condition 4a. And so this makes in the end for a total of strictly less than $\mathrm{B}_\alpha\cdot 2\cdot\alpha$ steps. Observe that with those bounds, $S^{(1)}$ satisfies Condition 4a as well.

Or, second possibility, the functions $(S^{(1)}_x)_{x<_X x_1}$ are not universally bounded and there is a least $x_2 <_X x_1$ for which the bound of $S^{(1)}_{x_2}$ is greater than ν_1, the X-rank of x_1, i.e. the least ordinal such that $x_1 = X(\nu_1)$. In this case, we consider the function $S^{(2)}$ that works w.r.t. $S^{(1)}$ like $S^{(1)}$ does w.r.t. S. Given x and α, $S^{(2)}$ computes $S^{(1)}(x,\alpha)$ and it returns the maximum image by $S^{(1),\alpha}$ of elements of X-rank less than $S^{(1)}(x,\alpha)$. That is,

$$S^{(2)}(x,\alpha) \coloneqq \max_{\mathrm{rk}_X(x')<S^{(1)}(x,\alpha)}(S^{(1)}(x',\alpha))$$

And we can apply the same reasoning to $S^{(2)}$. Either we can conclude, or the construction goes on. This creates a X-decreasing sequence of reals (x_i), a decreasing sequence of ordinals (ν_i), as well as a sequence of functions $(S^{(i)})_i$ such that for all $i>0$ (with $x_0=\widetilde{x}$, $\nu_0=\widetilde{\nu}$ and $S^{(0)}=S$):

- The function $S^{(i)}$ satisfies Conditions 1, 2, 3a and 4a of the main lemma (with a computation time of $(\mathrm{B}_\alpha\cdot\alpha)^n<\alpha^\omega$ for $S^{(i),\alpha}$).
- The function $S^{(i-1)}_{x_i}$ is bounded in Σ_3 but not strictly bounded by ν_{i-1}.
- The ordinal ν_i is the X-rank of x_i

As $<_X$ is a well-order, this decreasing sequence is finite and there is some finite n such that x_n and S^n satisfy, paired with X, all four conditions of the main lemma; that is Conditions 1, 2 and *4* (more precisely, the stronger condition 4a) as well as the fact that the functions $S^{(n)}_x$ for $x <_X x_n$ are universally bounded, which is Condition 3. Hence, in virtue of the main lemma this x_n is K-writable.

Now we need to show that from there, namely from the machine from the main lemma that K-writes x_n, we can in turn K-write $\widetilde{x}$. To do this, we show that for any i such that $0 < i \leqslant n$, x_{i-1} is x_i-eventually writable. To see this, remember that the function $S^{(i-1)}_{x_i}$ is bounded in Σ_3 but not strictly bounded by ν_{i-1}. So, given x_i and some ordinal α, a machine can compute $S^{(i-1)}_{x_i}(\alpha)$. In particular, it can e.w. the bound of $S^{(i-1)}_{x_i}$ (with the usual techniques, as it is Σ_2 definable without parameters other than x_i.) And so, as ν_{i-1} is less than the bound of $S^{(i-1)}_{x_i}(\alpha)$, it is in turn e.w. from x_i and then so is x_{i-1}, as wanted.

And so, having those machines working together with the usual dovetailing technique (there are finitely many such machines), $\widetilde{x}$ is e.w. from x_n. Eventually, in virtue of Auxiliary Lemma 6.2.43 (observe that x_n is K-written by applying the main lemma), $\widetilde{x}$ is K-writable. □

6.2.5 Main results

We now prove the main results regarding K_3 and K-writing.

Theorem 6.2.45. Σ_3 *is admissible.*

Proof. We show that the axiom of Σ_0-collection holds in L_{Σ_3}. Let $\varphi(x, y, z)$ be a Σ_0 predicate with three free variables and $A, p \in L_{\Sigma_3}$ such that $L_{\Sigma_3} \models \forall a \in A\, \exists b\ \varphi(a, b, p)$. Suppose moreover that the collection axiom does not hold. Then we can rewrite those assumptions in L_{Σ_3} as follows:

$$\begin{aligned} L_{\Sigma_3} \models \exists A, p\, (&\forall a \in A\, \exists b\, \varphi(a, b, p) \\ &\wedge \forall B\, \exists a \in A\, \forall b \in B \neg\varphi(a, b, p)) \end{aligned} \tag{6.1}$$

This statement is Δ_3, which gives an idea why this proof scheme would not work in the Σ_2 case where we have at most a Σ_2 elementary end-extension. We show using the Improved Main Lemma 6.2.36 that some witness (A, p) is K-writable. From there, it will be easy using the auxiliary Lemma 6.2.43 to K-write enough b's, and so to have enough $b \in L_{\mathrm{K}_3}$, to reach a contradiction.

To apply the improved lemma we need two functions, S and X that satisfy Conditions 1,2, 3a and 4a. For $<_X$ we choose the usual well-order on L. That is for any $\alpha < \Sigma_3$, $X(\alpha)$ is the α^{th} element of L_{Σ_3} w.r.t. to this well-order. In particular, if $x \in L_{\alpha+1} - L_\alpha$ and $y \in L_{\beta+1} - L_\beta$ with $\alpha < \beta$, then $x <_X y$.

For S, it is rather straightforward: let $x = \langle A, p\rangle$ encode some ordered pair (A, p). For each such x and $\alpha \in \Sigma_3$, we write

$$A_x^\alpha = \{a \in A \mid \exists b \in L_\alpha\ \varphi(a, b, p)\}$$

Then we define B_x^α to be the X-least $B \subseteq L_\alpha$ such that

$$\forall a \in A_x^\alpha\, \exists b \in B\, \varphi(a, b, p)$$

From there we define S as follows: for any x and α, if x is a valid encoding, $S(x, \alpha) = \beta$ where β is the L-rank of B_x^α, that is the least ordinal such that $B_x^\alpha \subseteq L_\beta$. Else, for invalid encodings, $S(x, \alpha) = 0$. Then we write $\widetilde{x}$ the X-least x encoding some (A, p) such that S_x is unbounded. There must be at least such a x as any witness of (6.1) will give rise to such an unbounded function. Conversely, $\widetilde{x}$ is almost a witness of (6.1). That is, writing $\widetilde{x} = \langle \widetilde{A}, \widetilde{p}\rangle$, we have :

$$L_{\Sigma_3} \models \forall B\, \exists a \in \widetilde{A}\, \forall b \in B \neg\varphi(a, b, \widetilde{p})) \tag{6.2}$$

It does not exactly mean that $(\widetilde{A}, p)$ is a witness of (6.1) but rather that there is $\widetilde{A}' \subset \widetilde{A}$ such that $(\widetilde{A}', \widetilde{p})$ is a witness. But in our case this will be enough to reach a contradiction. Also, we write $\widetilde{\alpha}$ the least α such that $\widetilde{x} \in L_{\widetilde{\alpha}}$.

We claim that this definition of S and X checks the four condition of the improved main lemma.

1. At any stage α, B_x^α is the X-least subset of L_α collecting the image of A_x^α by φ. By definition of A_x^α, $B_x^\alpha \subseteq L_\alpha$ and so $S(x, \alpha) \leqslant \alpha$. Then, as α increases, B_x^α grows only when A_x^α does. And A_x^α will grow when there is some $a' \in A$ for which there is some least $b' \in L_{\Sigma_3}$ such that $\varphi(a', b', p)$ and which b' appears latter in the constructible universe, that is $b' \in L_{\alpha'} - L_\alpha$ for some least $\alpha' > \alpha$. Then for such a least α', $A_x^{\alpha'} = A_x^\alpha \cup \{a'\}$ and $B_x^{\alpha'} = B_x^\alpha \cup \{b'\}$. Consequently, α' is the least ordinal such that $B_x^{\alpha'} \subseteq L_{\alpha'}$ and $S(x, \alpha') = \alpha'$. For $\widetilde{x}$, by (6.2), this happens cofinally often in Σ_3 so we have cofinally many α's such that $S_{\widetilde{x}}(\alpha) = \alpha$.

2. As α increases, A_x^α grows. That is, for $\alpha \geqslant \alpha'$, $A_x^\alpha \subseteq A_x^{\alpha'}$. Hence, $B_x^\alpha \subseteq B_x^{\alpha'}$ and $S_x(\alpha) \leqslant S_x(\alpha')$, which shows in particular that $S_{\widetilde{x}}$ is monotonic.

3a. By definition of $\widetilde{x}$, for any $x <_X \widetilde{x}$, S_x is bounded. They may not be universally bounded which is why we use the improved main lemma.

4a. Given α, to compute $X(\alpha)$, it is enough to compute $L_{\alpha+1}$. Applying the recursive construction of $L_{\alpha+1}$ using the operator Def can be a bit time consuming but this can be done using Gödel's functions (see [Bar75, II.5]). And so, at each stage $\beta < \alpha$ of the construction, $L_\beta \times L_\beta$ can be enumerated in $\beta + \omega$ stages and for all x, y in

$L_\beta \times L_\beta$, $\mathcal{F}_i(x, y)$ can be computed in β^2 stages. All in all it takes less than α^4 steps to compute $X(\alpha)$.

To compute $S(x, \alpha)$ for $x = X(\nu) = \langle A, p\rangle$ with $\nu < \alpha$, it is enough to compute L_α and to build A_x^α and B_x^α from it. The assumption on the X-rank of x ensures that A can be enumerated in less than α steps. Then, once L_α was computed, the definition of A_x^α and B_x^α can simply be applied to compute those. The computation of A_x^α takes α^2 steps and that of B_x^α may take up to α^3 steps. Again, this is yield a universal bound strictly less than α^ω steps.

Eventually, it is clear that those bounds are monotonic in α.

So, by the improved main lemma, $\widetilde{x} = \langle \widetilde{A}, \widetilde{p}\rangle$ is K-writable. In particular, any $a \in \widetilde{A}$ is $\widetilde{x}$-e.w., and for such a a, some $b \in L_{\Sigma_3}$ (typically the first to appear in $\mathcal{U}_3$) is itself a-e.w. And so, in in virtue of the Auxiliary Lemma 6.2.43 (it is clear that we can apply it "over" the improved main lemma as this one is, in essence, just the main lemma and the auxiliary lemma in a trench coat), this b is K-writable. Further, the least β such that $b \in L_\beta$ is less than Σ_3 (as $\Sigma_3 = \mathrm{T}_3$) and so it is b-e.w. Hence applying again the auxiliary lemma (it can also be applied "over" itself), β is K-writable and so $b \in L_{\mathrm{K}_3}$. Consequently:

$$\forall a \in \widetilde{A}\, \exists b \in L_{\mathrm{K}_3}\, \varphi(a, b, \widetilde{p})$$

which contradicts the fact that $(\widetilde{A}, \widetilde{p})$ was a witness of (6.2). And so Σ_0-collection holds in L_{Σ_3}. □

The following proposition is a transitivity proposition. It is easy to prove that if x is e.w. and if y is x-e.w., then y is e.w. as well. And since this is easy to prove, this property is often used without being explicitly named. In particular, it plays a hidden but pivotal in the original proof of the λ-ζ-Σ Theorem 4.1.33. And here too, we will need transitivity of K-writing to establish the ζ-K-Σ Theorem 6.2.49.

Proposition 6.2.46 (Transitivity of K-writing)**.** *Let x be* K*-writable and y be x-*K*-writable, that is* K*-writable with input x. Then y is* K*-writable as well.*

Proof. We write m_x for the machine that K-writes x and m_y for the machine that K-writes y with input x

We show, using the improved main lemma, that some $\widetilde{z}$ encoding x and y is K-writable. We fix X to describe the order $<_L$ on L_{Σ_3}. We write $\widetilde{z} = \langle \widetilde{x}, \widetilde{y}, \nu_0^x, \beta_0^x, \nu_0^y, \beta_0^y\rangle$, the X-least tuple such that the history of m_x, from ν_0^x onward, is $\widetilde{x}$-populated w.r.t. β_0^x and also such that the history of the computation $m_y[\widetilde{x}]$, from ν_0^y onward, is $\widetilde{y}$-populated w.r.t. β_0^y. As the whole history of m_x is not x-co-populated we have $\widetilde{x} = x$ and, with the same argument, $\widetilde{y} = y$. We show that we can apply the improved main lemma with $\widetilde{z}$ and so that it is K-writable.

To do this, we define the following computable segment function S. Given α and $z' = \langle x', y', \nu_0^{x'}, \beta_0^{x'}, \nu_0^{y'}, \beta_0^{y'} \rangle$, S looks for the greatest, if it exists, $\delta_{x'} \leqslant \alpha$ such that $\delta_{x'} > \nu_0^{x'}$ and such that x' has been K-written in m_x w.r.t. $\beta_0^{x'}$, from $\nu_0^{x'}$ and up to $\delta_{x'}$. If such a $\delta_{x'}$ does not exists, $\delta_{x'}$ is set to 0. It then looks for the greatest $\delta_{y'} \leqslant \alpha$ such that $\delta_{y'} > \nu_0^{y'}$ and for which y' has been K-written in $m_y[x']$ from $nu_0^{y'}$, up to $\delta_{y'}$ and w.r.t. $\beta_0^{y'}$. If such a $\delta_{y'}$ does, it is also set to 0. Then, S outputs $\min(\delta_{x'}, \delta_{y'})$.

We claim that X and S satisfy the conditions of the main lemma.

1. We need to show that there are cofinally many α's such that $S_{\widetilde{z}}(\alpha) = \alpha$. We consider the functions $\delta_x(\alpha)$ and $\delta_y(\alpha)$ that map any α to the δ_x and δ_y that are computed in S from $\widetilde{z}$ and this α. It is easy to see that there are cofinally many α's such that $\delta_x(\alpha) = \alpha$: by definition of x-population, for any $\nu > \nu_0^x$, there are $\nu' > \nu$ and $\nu'' > \nu'$ describing a big enough x-segment. Then looking at all ν below some N and taking α to be the l.u.b. of the least witnesses ν'' associated to each of those ν yields a fixed point for δ_x, and this fixed point α is strictly less than Σ_3 by admissibility of Σ_3. Same goes for y, there are cofinally many α's such that $\delta_y(\alpha) = \alpha$. Now, since S outputs the minimum of $\delta_x(\alpha)$ and $\delta_y(\alpha)$ the question is to know whether there are cofinally many α's that satisfy both equalities. Let α such that $\delta_x(\alpha) = \alpha$ and suppose that $\delta_y(\alpha) < \alpha$. Then there is $\alpha' > \alpha$ such that $\beta_y(\alpha') = \alpha'$. Either $\delta_x(\alpha') = \alpha'$ and we're done, either there is some $\alpha'' > \alpha'$ such that $\delta_x(\alpha'') = \alpha''$. If we don't find a shared fixed point, this yields an infinite sequence $(\alpha^n)_n$ of alternating fixed points. We write $\mathrm{A} = \bigcup_\omega \alpha_n$. Observe first that $\mathrm{A} < \Sigma_3$ as the function $n \mapsto \alpha_n$ is Σ_1-definable from x and y which are in Σ_3. Second, as A is a limit of initial segments in the history of m_x that are x-populated, A itself is x-populated (w.r.t β_0^x, encoded in $\widetilde{z}$). As for it not being x-co-populated, it is enough to suppose w.l.o.g. that A is bigger than the witness $\nu > \nu_0^x$ in the negation of the x-co-populated formula w.r.t. β_0^x. Hence, x is K-written up to this A and $\delta_x(\mathrm{A}) = \mathrm{A}$. Same goes for the computation $m_y[x]$ and this method yields cofinally many A's such that $S_{\widetilde{z}}(\mathrm{A}) = \mathrm{A}$.

2. As for monotony, this is direct: for any $\alpha < \alpha'$, we have $\delta_x(\alpha) \leqslant \delta_x(\alpha')$. Same goes for δ_y and consequently $S_{\widetilde{z}}(\alpha) \leqslant S_{\widetilde{z}}(\alpha')$.

3a. We need to show that for all $z <_X \widetilde{z}$, the function S_z is bounded in Σ_3. By definition of $\widetilde{z}$, for $z = \langle x', y', \nu_0^{x'}, \beta_0^{x'}, \nu_0^{y'}, \beta_0^{y'} \rangle <_X \widetilde{z}$, either the history of m_x is not x'-populated from $\nu_0^{x'}$ and w.r.t. $\beta_0^{x'}$, either that of $m_y[x']$ is not y'-populated from $\nu_0^{y'}$ and w.r.t. $\beta_0^{y'}$, or both. In any case, for this z', the $\beta_{x'}$ or the $\beta_{y'}$ in the definition of S are bounded and consequently so is $S_{z'}$.

4a. As seen in the proof of Theorem 6.2.45, computing $L_{\alpha+1}$ in order to compute X_α takes strictly less than α^ω steps and this yield a monotonic bound in α. As for S^α, for any z, it is enough to simulate m_x and m_y for α steps, saving their history of

computation using α and then to use it to compute δ_x and δ_y. In this end this takes less than α^2 steps.

Hence $\widetilde{z}$ is K-writable and, as required, so is $\widetilde{y}$ as it is $\widetilde{z}$-writable. □

Proposition 6.2.47. *A set a is* K*-writable if and only if $a \in L_{\mathrm{K}_3}$.*

Proof. Let a be K-writable. It is then also accidentally writable. As $\Sigma_3 = \mathrm{T}_3$, it appears before stage Σ_3 in $\mathcal{U}_3$. Hence, there is some $\alpha < \Sigma_3$ such that $a \in L_\alpha$. This α is a-K-writable (and even a-e.w.) and so, in virtue of transitivity of K-writing, α is K-writable and $a \in L_{\mathrm{K}_3}$.

Conversely, if $a \in L_{\mathrm{K}_3}$, by the same transitivity reasonning, a is K-writable. □

Proposition 6.2.48. $L_{\lambda_3} \prec_{\Sigma_1} L_{\zeta_3} \prec_{\Sigma_2} L_{\Sigma_3}$

Proof. This is done like in the Γ_{sup} setting. Suppose that $L_{\Sigma_3} \models \exists x \, \forall y \psi(x, y, p)$. Then, the first x witness of this which appears in $\mathcal{U}_3$ is e.w. And so this formula is also true in L_{ζ_3}. Conversely, if $L_{\zeta_3} \models \exists x \, \forall y \psi(x, y, p)$, we write $\widetilde{x} \in L_{\zeta_3}$ a witness of it. Suppose that this formula is not true in L_{Σ_3}. Then, there is a least $\widetilde{y} \in L_{\Sigma_3} - L_{\zeta_3}$ for which $\neg\psi(\widetilde{x}, \widetilde{y}, p)$. But then, this $\widetilde{y}$ is $\widetilde{x}$-writable and consequently e.w., which is a contradiction. The Σ_1 end-extension is shown in a similar fashion. □

Theorem 6.2.49 (ζ-K-Σ Theorem)**.** $L_{\zeta_3} \prec_{\Sigma_2} L_{\mathrm{K}_3} \prec_{\Sigma_3} L_{\Sigma_3}$

Proof. The first elementary end-extension will be a consequence of the previous proposition and of the Σ_3 elementary end-extension.

For the Σ_3 end-extension, let $p \in L_{\mathrm{K}_3}$ and $\varphi(p) = \exists x \, \forall y \, \exists z \, \psi(x, y, z, p)$ such that $L_{\Sigma_3} \models \varphi(p)$. We show that some witness x is p-K-writable which then implies that this x is K-writable by the transitivity relation established in Proposition 6.2.46. Let X be the iteration of L along its usual well-order. Let $\widetilde{x}$ be the X-smaller witness of φ in L_{Σ_3}. We show that we can apply the improved main lemma with p given as parameter to the segment function and so that $\widetilde{x}$ is p-K-writable.

We consider the following computable function S: given p, x and α, it looks for the greatest $\beta_y \leqslant \alpha$ such that:

$$\forall y \in L_{\beta_y} \exists z \in L_\alpha \psi(x, y, z, p)$$

and then, for the least β_z and such that :

$$\forall y \in L_{\beta_y} \exists z \in L_{\beta_z} \psi(x, y, z, p)$$

and $S(x, \alpha)$ equal to $\max(\beta_y, \beta_z)$.

We claim that $\widetilde{x}$, S and X satisfy the conditions of the improved main lemma.

1. First, it is clear that $S(x,\alpha) \leqslant \alpha$. Then, take some $\alpha < \Sigma_3$ such that $S_{\widetilde{x}}(\alpha) < \alpha$. This mean, with β_y as defined in $S_{\widetilde{x}}(\alpha)$, that $\beta_y < \alpha$. We consider the least α' such that

$$\forall y \in L_{\beta_y+1} \exists z \in L_{\alpha'}\ \psi(\widetilde{x}, y, z, p)$$

It exists as $\widetilde{x}$ is witness of φ and by admissibility of Σ_3, $\alpha' < \Sigma_3$. Moreover, $\alpha' > \alpha$ as β_y is strictly less than α and was the greatest such that $\forall y \in L_{\beta_y} \exists z \in L_\alpha \psi(x, y, z, p)$. And by definition of α', $\beta_z = \alpha'$ where β_z is defined as in the description of $S(\widetilde{x}, \alpha')$. And so $S(\widetilde{x}, \alpha') = \alpha'$.

2. Monotonicity of $S_{\widetilde{x}}$ is clear.

3a. Let $x <_X \widetilde{x}$. Then x is not a witness of φ, that is there is some y such that for all $z \in L_{\Sigma_3}$, $\neg\psi(x, y, z, p)$. Hence β_y is bounded and then so is β_z by admissibility of L_{Σ_3}.

4a. The bound on the computation time and their monotonicity is clear as well.

We now have $\widetilde{x} \in L_{\mathrm{K}_3}$ such that $\forall y \in L_{\mathrm{K}_3}\ \exists z \in L_{\Sigma_3}\ \varphi(\widetilde{x}, y, z)$. Then for any $y \in L_{\mathrm{K}_3}$, there is some z such that $\varphi(\widetilde{x}, y, z, p)$ which is $\langle x, y, p\rangle$-writable, namely the first z such that $\varphi(\widetilde{x}, y, z, p)$ to appear in the enumeration of $\mathcal{U}_3$. Then, z is *a fortiori* $\langle x, y, p\rangle$-K-writable. So by Proposition 6.2.46, z is K-writable which proves that $L_{\mathrm{K}_3} \models \varphi$.

Conversely, if $L_{\mathrm{K}_3} \models \varphi(p)$: writing $\widetilde{x}$ for the witness of φ in L_{K_3}, if there is some $y \in L_{\Sigma_3}$ such that for all $z \in L_{\Sigma_3}$, $\neg\psi(\widetilde{x}, y, z, p)$, then y is $\widetilde{x}$-e.w. and so, by Proposition 6.2.46, it is K-writable which is a contradiction with the fact that $L_{\mathrm{K}_3} \models \forall y \exists z \psi(\widetilde{x}, y, z, p)$. □

We finished the chapter with this corollary, itself finishing the picture of the Γ_3-ITTMs.

Corollary 6.2.50. *L_{Σ_3} is not a Σ_3-end extension of L_{ζ_3}, and so $\zeta_3 < \mathrm{K}_3$.*

Proof. The statement "m is converging" is a Σ_2 statement: there is some ordinal stage α such that for all $\beta > \alpha$, the input of m is the same at stage β and α. And so, as $\zeta_3 < \Sigma_3$,

$$L_{\Sigma_3} \models \exists \mathrm{B}\, \forall m \text{ "m is converging before stage B"}$$

and this Σ_3 formula clearly does not hold in L_{ζ_3}. As we already knew that $\zeta_3 \leqslant \mathrm{K}_3$ (eventually writing is a particular form of K-writing), the inequality is now strict. □

Chapter 7

Look-back and outlooks

The first part of my work done during this thesis, presented in Chapter 5, aimed at providing a first ground on which to develop a generalized theory of infinite Turing machines. At the cost of some restrictions–which in essence are specifications–regarding what an infinite Turing machine, and in particular its limit rule, can look like, we managed to re-establish a good part of what was done for the classical lim sup ITTM. The main result of this chapter was Theorem 5.4.14 that gives sufficient conditions for an operator Γ to satisfy the equality $\Sigma_\Gamma = \mathrm{T}_\Gamma$. The second important result was Theorem 5.5.1 which shows how the one condition of the previous theorem which we may think is disposable is actually necessary.

Then, in the second part of this work, in Chapter 6, we managed, using the theorical results of the previous chapter, to re-establish most of what was done for the lim sup ITTM for a concrete but more powerful model of infinite machines. This model is more powerful as it defined using a Σ_3 formula and as it makes full use of this "Σ_3 potential". This assertion, namely, that it makes full use of its Σ_3 definition, is supported by the last important result of the chapter, the ζ-K-Σ Theorem 6.2.49, which exhibits a Σ_3 elementary end-extension between L_{K_3} and L_{Σ_3}.

7.1 Back to simulational Γ-operators

The easiest but most important part was the definition of simulational operators. For those operators, we can define a universal machine as is ubiquitously used with classical ITTMs. An interesting aspect of this definition of simulational operators is that if someone tries to come up with a new limit rule (without making a firm attempt at breaking things), the odds are that the operator associated to this rule will be, at least, stable, cell-by-cell and asymptotic. If it is not contraction-proof, it can be easily transformed into a contraction-proof operator by adding a new symbol to its alphabet which will act as a separator. That is, we add a fresh symbol b to the alphabet of the operator and we slightly change its definition so that is behaves with a stutter-free word of the form $s_1 b s_2 b s_3 b \dots$ as it would have done with the non-necessarily stutter-free word $s_1 s_2 s_3 \dots$. This is one

of main ideas behind the definition of the 3-symbol operator Γ_3. Only the condition of looping stability may, from one attempt to design a new rule to another, not be satisfied.

And so, in the end, this makes for quite a natural and wide space of simulational operators. This naturally calls for a study of this space of operators. This can be done in terms of power of computation: for an operator Γ, how big is Σ_Γ? How often is an ordinal α equal to some Σ_Γ? How are the Σ_Γ distributed in ω_1? And to be more precise, we can also reformulate Post's problem: can we find a degree Σ_Γ between 0 and $\Sigma_2 = \Sigma_{\Gamma_{\text{sup}}}$? Or between any two Σ_Γ and $\Sigma_{\Gamma'}$? etc.

It can also be done in terms of logical complexity: what are the Σ_n-suitable and simulational operators? Can we even design a Σ_n-suitable and simulational operator for any natural number n? Do they always lead to more powerful machines? Under which conditions does it yield a higher-order elementary end-extension?

Then, we can wonder how those operators relate to the more general concept of arithmetic operators (i.e. arithmetic functions on $\mathcal{P}(\omega)$) along with usual constraints (inductive, quasi-inductive, etc.) as used in [Wel05]. In the same way, the idea of studying the evolution of some subset of ω (here, represented by the tape) is also one of the fundamental motivation of cellular automatons. From this point of view, it would likely be fruitful to compare the constraint on simulational operator to those established for different models of cellular automatons (see for example [CL17] which develops shift-invariant filters on ω to define global cellular automatons).

Finally, it can be done in terms of classification of ordinally index words. Again, this was the intuition behind the Γ_3 operator, as mentionned in Remark 6.2.6. The idea is to see a limit rule as a infinite word classifier. It classifies the words (say, on 2 symbols) based on their density of respectively 0's and 1's. A limit rule yields, at a limit stage, 1 if it decides that the word formed by the history of the cell has "more" 1's than 0's. Obviously this notion of density is highly vague–but that is what makes it interesting. Given a limit ordinal α and a word $w \in {}^\alpha 2$, does it have asymptotically "more" 1's than 0's? If after some stage $\beta < \alpha$, only 0's appear, surely this word is more densely packed with 0's. Else, we can decide that it is a bit too unclear and that in this case, by convention, the density of 1 is higher. And this way of seeing things simply yields the usual lim sup limit rule.

But we may be left a bit disappointed with the choice of this convention. For example, it implies that a word w' in which segments of 1's are cofinals but bounded in length while the 0-segment are unboundedly long (naturally, here we mean unbounded in the length of the word), is more densely populated with 1's than with 0's. So we can change the rule to better reflect this: we now classify a bit more precisely by specifying that with bounded 1-segments and unbounded 0-segments, a word is now 0-dense. With this new operator, the word $\Pi_{k\in\omega}10^k$ would then yield a 0 instead of a 1. Observe that now, this operator that looks whether 1-segments are bounded and whether 0-segments are unbounded is a Δ_3-suitable operator (more precisely $\Sigma_2 \wedge \Pi_2$ but not less!). But still, according to this new operator, for any k, the word $(10^k)^\omega$ would be–against our intuition–more inhabited

with 1's than with 0's.

And this idea, to some extent, points toward a shift, away from operators and rules and closer to the space of ordinal words. Can we define a partitioning of this space that matches our intuition? That is, what are the functions φ which induce a tri-partioning $X_1 \sqcup X_0 \sqcup X_?$ of this space such that for all word w, writing $\overline{w}$ its bitwise negation,

$$\begin{aligned} w \in X_1 &\longleftrightarrow \varphi(w) \wedge \neg\varphi(\overline{w}) \\ w \in X_0 &\longleftrightarrow \varphi(\overline{w}) \wedge \neg\varphi(w) \\ w \in X_? &\longleftrightarrow [\varphi(w) \wedge \varphi(\overline{w})] \vee [\neg\varphi(w) \wedge \neg\varphi(\overline{w})] \end{aligned}$$

and this with the "smallest" $X_?$ possible? Surely topology has some ideas about this one. And as we are interested in asymptotic behavior, such functions φ would already yield stable, cell-by-cell and asymptotical operators. Also, as already seen, it is easy to add a symbol in order to transform it in a contraction-proof operator, and already the operator would be simulational. Finally, as we are interested in the density of 1's, we could wager (to some extent, as it is asymptotic density) that from the point of view of such a φ, w and w^ω have similar density. And so that $\varphi(w) \implies \varphi(w^\omega)$, which would mean that the operator induced by φ is moreover looping stable. With this, we see how the constraints on the operators devised in the Chapter 5 come even more naturally in this reformulation.

We finish this section with an open question, asked and left open in Remark 5.4.33. For a simulational and looping stable operator Γ, we showed that $\Sigma_\Gamma = \mathrm{T}_\Gamma$. Then we are tempted to say that any Γ-machine is seen to be looping at the latest at stage Σ_Γ. Still, if we apply the usual techniques, relying on asymptoticity and looping stability, as well as the fact that no new reals appear after stage T_Γ, we only seem to be able to say that it is seen to be looping before stage $\Sigma_\Gamma \cdot \omega$. Now, it is a bit infuriating, after all the work done to establish the equality $\Sigma_\Gamma = \mathrm{T}_\Gamma$, to obtain such a lukewarm result. If we believe in mathematical orderliness, we may think that we can bring this bound back to Σ_Γ–but there still misses a proof.

7.2 Toward Σ_n machines

In the second part of this work, we introduced the Σ_3 operator Γ_3. It can be seen, under many criteria, as an adequate generalization of the usual Σ_2 operator. Among those criteria being the fact that most of the results on the usual ITTMs have been ported to Γ_3-machines.

We take a look back to see what are the main traits of the operator Γ_3 and how those were crucial to establish the main results. We will see that many of those traits are shared with the operator Γ_{sup}. Only, they appear more clearly and so does their importance, in the more complex setting of 3-symbol Σ_3-operators.

Starting with the end, as often, the aim was to design an operator that yields a Σ_3-elementary end-extension. Having the proof of the λ-ζ-Σ Theorem in mind, we look in the eyes at a Σ_3 formula:

$$\varphi = \exists x \, \forall y \, \exists z \, \psi(x, y, z)$$

On the contrary to the what was happening in the Σ_2 case, we can't eventually decide whether some x is a witness of φ and so we can't e.w. some least witness of φ. This comes from the fact that for a well-behaved operator Γ, we can't decide whether $L_{\Sigma_\Gamma} \models \exists z \psi(x, y, z)$. So for any well-behaved operator Γ we can't hope to establish this Σ_3 extension between L_{ζ_Γ} and L_{Σ_Γ}. Hence there needs to be some constant between ζ_Γ and Σ_Γ bridging this gap. But, in turn, this means that there needs to be a new way of writing reals. What may it be like?

To answer this question, observe however that a machine can decide, given some $\alpha \in \Sigma_\Gamma$, whether $L_\alpha \models \exists z \psi(x, y, z)$. And for such an α it can look for the least y (supposing there is any) for which $L_\alpha \not\models \exists z \psi(x, y, z)$. If x is a witness of φ, as α gets bigger, the rank of this least y will get bigger as well (supposing that it always exists). And so, for this x it yields a sequence of ordinals (the ranks of those least y's) unbounded in Σ_3. With this, it is very convenient to design a way of writing thought around x-segments: we are be easily able to have some machine write unboundedly big x-segments for the witnesses x but not for other reals that are not witness of φ. We will come back to this.

In parallel we are faced with another difficulty. Suppose we have introduced a new way of writing real which then produces some constant μ, the supremum of the "μ-writable" ordinals. The difficulty is now to show that $\mu < \Sigma_\Gamma$. If we design a too lenient way of writing, it may happen that any a.w. real is also μ-writable. Indeed, if we stick with our first approximation: "x is μ-written in the computation of m if the x-segments of its history are unboundedly long (in Σ_Γ)", it is easy enough to devise a computation that μ-writes all a.w. ordinals. This hints at the too obvious fact that, on the contrary, it isn't possible for a computation to e.w. more than one real. Again putting a pin on this issue, we look at the proof of the fact that $\zeta < \Sigma$ for the classical ITTMs.

The idea of this proof for the classical ITTMs is simple enough: to prove $\zeta < \Sigma$, it suffices to exhibit an a.w. but not e.w. ordinal. By definition, ζ is not e.w., so it would be enough to a.w. it. Moreover, at some point all e.w. reals have been eventually written in some machine and so, simulating the universal machine, at some point, we can write their sum. Now, what is hidden between the very convenient fact that once a real is e.w. it stays on the output tape, is the fact that when some real is e.w. below a limit stage α, in virtue of the perfect symmetry between the lim sup rule and the notion of eventually writing, it appears at this stage α. All e.w. ordinals are e.w. below ζ and so they all appear at stage ζ, at which stage their sum can be computed. So, if we take this path,

we need our new writing notion to also be the symmetry of the rule associated with Γ. Moreover, this proof works because only one real can be e.w. at a time. In our case, what would appear at stage μ for a machine which μ-writes many real at once? This is why we need the notion of "exclusivity" that was introduced in Definition 6.2.31, in the context of formalization of what a "writing predicate" is. Looking back at Proposition 6.2.34 which states that $\mathrm{K}_3 < \Sigma_3$, and now used to those kind of reasoning, it is an immediate consequence of the fact that κ, the writing predicate associated to K-writing, is an exclusive writing predicate and of its symmetry with the $\Sigma_3 \wedge \Pi_3$ rule.

This solves the issue that was previously pinpointed and what is left is now the effective design of the writing predicate μ. But if we want μ and Γ to be the symmetry of one another, this design inherits from the constraint on Γ (simulational, looping stable etc.). To those, we also add the monotonicity (see Definition 6.2.30): if m μ-writes x and we replace some y in its history by x, we surely want the history to still be an history in which x is μ-written. With all of this, I believe there is a good chance at managing to design a simpler Σ_3 operator for which we can establish the same kind of end-extension, but in a more straight-forward way. Similarly, this gives most of the tools to design higher-order Σ_n operators.

As a last word, let us see how the idea of using x-segment to K-write a witness of a Σ_3 formula may be generalized to higher order. We consider the following Σ_4 formula.

$$\varphi = \exists x \forall y \exists z \forall t \, \psi(x, y, z, t)$$

Here, we could do the same thing: for each x and α we look at the least y such that $\exists z \in L_\alpha \forall t \in L_\alpha \psi(x, y, z, y)$ does not hold. Clearly, when x is witness, the rank of those y will be unbounded in Σ_Γ. However, there may also be some x's which are not witnesses and for which the rank of the y also are unbounded. Indeed, for such a x which is not witness of φ, there may be some y such that

$$\begin{cases} \neg \exists z \, \forall t \, \psi(x, y, z, t) \\ \forall \alpha \, \exists z \, \forall t \in L_\alpha \, \psi(x, y, z, t) \end{cases}$$

But now, this mean that for this x which is not witness and this y, the rank of the z witness of $\exists z \in L_\alpha \forall t \in L_\alpha \, \psi(x, y, z, y)$ are themselves unbounded in Σ_Γ.

To parry this, we could imagine the notion of *quasi-segments*: a x-quasi-segment in a word w is the data of two ordinals α and β such that between those two stages the x-co-segments are "small". In other words, a x-quasi-segment is a x-segment in which we allow for some holes, as long as they stay "small", or maybe bounded. With such a notion, for this x which is not witness, when the machine looks for a z such that $\exists z \in L_\alpha \forall t \in L_\alpha \, \psi(x, y, z, y)$, it will make holes in the x-segment of length $\mathrm{rk}(z)$. And so, as there isn't a fixed z such that $\forall t \, \psi(x, y, z, t)$, those holes will get bigger and bigger

and, asymptotically, this won't form unboundedly long x-quasi-segments in the history of the machine. And this definition naturally generalizes: a x-quasi-quasi-segment in a word w is the data of two ordinals α and β such that between those two stages the x-co-quasi-segments are "small" and so on...

Bibliography

[Bar75] Jon Barwise. *Admissible sets and structures*. Vol. 7. Perspectives in Logic. Springer-Verlag, 1975.

[BDL23] Kenza Benjelloun, Bruno Durand, and Grégory Lafitte. "Writability power of ITTMs: ordinals and constructible sets". Preprint. 2023.

[Car19] Merlin Carl. *Ordinal Computability: An Introduction to Infinitary Machines*. Berlin, Boston: De Gruyter, 2019.

[CL17] Julien Cervelle and Grégory Lafitte. "On shift-invariant maximal filters and hormonal cellular automata". In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2017, pp. 1–10.

[CRS20] Merlin Carl, Benjamin Rin, and Philipp Schlicht. "Reachability for infinite time Turing machines with long tapes". In: *Logical Methods in Computer Science* 16 (2020).

[Dav58] Martin Davis. *Computability & Unsolvability*. New York: Dover Publications, 1958.

[Dev84] Keith J. Devlin. *Constructibility*. Vol. 6. Perspectives in Logic. Springer-Verlag, 1984.

[DL19] Bruno Durand and Grégory Lafitte. "An algorithmic approach to characterizations of admissibles". In: *Computing with Foresight and Industry*. Ed. by Florin Manea et al. Cham: Springer International Publishing, 2019, pp. 181–192.

[FS13] Tim Fischbach and Benjamin Seyfferth. "On λ-Definable Functions on Ordinals". In: *The Nature of Computation. Logic, Algorithms, Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 135–146.

[FW07] Sy-David Friedman and Philip D. Welch. "Two observations regarding infinite time Turing machines". In: *Bonn International Workshop on Ordinal Computability*. 2007, p. 44.

[FW11] Sy-David Friedman and Philip D. Welch. "Hypermachines". In: *The Journal of Symbolic Logic* 76.2 (2011), pp. 620–636.

[HL00] Joel David Hamkins and Andy Lewis. "Infinite time Turing machines". In: *The Journal of Symbolic Logic* 65.2 (2000), pp. 567–604.

[Hod80] Harold T Hodes. "Jumping through the transfinite: the master code hierarchy of Turing degrees1". In: *The Journal of Symbolic Logic* 45.2 (1980), pp. 204–220.

[Hod93] Wilfrid Hodges. *Model theory*. Encyclopedia of Mathematics and its Applications. Cambridge university press, 1993.

[Jec03] Thomas Jech. *Set theory: The third millennium edition, revised and expanded*. Springer Monographs in Mathematics. Springer, 2003.

[Jen72] R Björn Jensen. "The fine structure of the constructible hierarchy". In: *Annals of mathematical logic* 4.3 (1972), pp. 229–308.

[Kec75] Alexander S. Kechris. "The Theory of Countable Analytical Sets". In: *Transactions of the American Mathematical Society* 202 (1975), pp. 259–297. ISSN: 00029947.

[KM08] Peter Koepke and Russell Miller. "An enhanced theory of infinite time register machines". In: *Conference on Computability in Europe*. Springer. 2008, pp. 306–315.

[Koe05] Peter Koepke. "Turing computations on ordinals". In: *Bulletin of Symbolic Logic* 11.3 (2005), pp. 377–397.

[Koe09] Peter Koepke. "Ordinal computability". In: *Mathematical Theory and Computational Practice*. Springer. 2009, pp. 280–289.

[KS06] Peter Koepke and Ryan Siders. "Computing the recursive truth predicate on ordinal register machines". In: *Logical Approaches to Computational Barriers, Computer Science Report Series* 7 (2006), pp. 160–169.

[KS08] Peter Koepke and Ryan Siders. "Minimality considerations for ordinal computers modeling constructibility". In: *Theoretical computer science* 394.3 (2008), pp. 197–207.

[KS09] Peter Koepke and Benjamin Seyfferth. "Ordinal machines and admissible recursion theory". In: *Annals of Pure and Applied Logic* 160.3 (2009), pp. 310–318.

[Kun80] Kenneth Kunen. *Set Theory: An Introduction to Independence Proofs*. North-Holland, 1980.

[LP71] Stephen Leeds and Hilary Putnam. "An intrinsic characterization of the hierarchy of constructible sets of integers". In: *Studies in Logic and the Foundations of Mathematics*. Vol. 61. Elsevier, 1971, pp. 311–350.

[Mad17] David A Madore. "A Zoo of Ordinals". In: *Unpublished manuscript* (2017).

[Neu23] John von Neumann. "Zur Einführung der transfiniten Zahlen". In: *Acta Scientiarum Mathematicarum (Szeged)* 1.4 (1923), pp. 199–208.

[Rog87] Hartley Rogers Jr. *Theory of recursive functions and effective computability*. MIT press, 1987.

[Sac90] Gerald E Sacks. *Higher recursion theory*. Vol. 2. Perspectives in Logic. Springer-Verlag, 1990.

[Sim09] Stephen George Simpson. *Subsystems of second order arithmetic*. Vol. 1. Perspectives in Logic. Cambridge University Press, 2009.

[SS12] Philipp Schlicht and Benjamin Seyfferth. "Tree representations via ordinal machines". In: *Computability* 1.1 (2012), pp. 45–57.

[Tur+36] Alan Mathison Turing et al. "On computable numbers, with an application to the Entscheidungsproblem". In: *J. of Math* 58.345-363 (1936), p. 5.

[Wel00a] Philip D. Welch. "Eventually infinite time Turing machine degrees: Infinite time decidable reals". In: *The Journal of Symbolic Logic* 65.3 (2000), pp. 1193–1203.

[Wel00b] Philip D. Welch. "The length of infinite time Turing machine computations". In: *Bulletin of the London Mathematical Society* 32.2 (2000), pp. 129–136.

[Wel05] Philip D. Welch. "The transfinite action of 1 tape Turing machines". In: *Conference on Computability in Europe*. Springer. 2005, pp. 532–539.

[Wel09] Philip D. Welch. "Characteristics of discrete transfinite time Turing machine models: Halting times, stabilization times, and Normal Form theorems". In: *Theoretical Computer Science* 410.4 (2009), pp. 426–442.

Appendix A

Résumé en français

Ce travail de thèse porte sur les machines de Turing infinies (ITTM) telles que développées par Hamkins et Lewis au début des années 2000. Ces machines permettent d'effectuer des calculs en temps ordinaux. Ainsi, une machine pourra calculer pendant $\omega^2 + 5$ étapes avant de s'arrêter. Pour autant, ces machines ont des limites, par exemple dans le nombre ordinal d'étapes de calcul qu'elles peuvent effectuer avant de s'arrêter. Ainsi, ce travail s'intéresse en particulier aux questions de généralisation de ce modèle de calcul ordinal. Un aspect notable de ces machines infinies est que, comparée au modèle classique de Turing, celles-ci sont presque aussi simples. Une machine de Turing infinie a la même structure qu'une machine de Turing classique à trois rubans. Elle fait des calculs en temps ordinaux et, comme dans le modèle classique, à n'importe quelle étape l'instantané de la machine (c'est-à-dire le contenu de ses rubans, son état ainsi que la position de sa tête de lecture) permet de calculer l'instantané à l'étape suivante. Le seule différence est aux étapes ordinales limites. En effet, les ordinaux limites sont par définition les ordinaux qui ne sont successeurs de personne. Il faut donc une règle spéciale pour le passage à ces ordinaux limite. La règle est suffisamment simple pour être décrite en deux lignes : à une étape ordinale limite, la tête de lecture est remise au début, la machine est dans un certain état spécial et la valeur de chaque cellule est la limite supérieure de l'historique de ses valeurs précédentes. Si le choix pour la tête de lecture et pour l'état limite peuvent sembler logique, le choix de la valeur limite pour les cellules peut apparaître arbitraire. Pourquoi pas une limite inférieure ? Ou encore quelque chose de plus compliqué ? Finalement, la justification de cette règle limite est une corroboration : avec cette règle, Hamkins et Lewis ont montré que le modèle de machines qu'ils ont développé est robuste, puissant et se comporte bien.

L'objectif est donc de proposer des règles limites différentes de la règle limsup, et qui produiraient des généralisations plus puissantes de cette machine infinie.

Le point de départ de ce travail est le concept de machine universelle. En effet, la plupart des preuves portant sur les ITTMs utilisent une « machine universelle », c'est-à-dire une machine qui simule en parallèle toutes les autres ITTMs. Une une telle machine

est en fait simple à définir, mais cette simplicité est fortuite : il pourrait y avoir de nombreuses difficultés qui sont évitées grâce à plusieurs propriétés implicites des ITTMs. Nous avons ainsi mis en lumière un ensemble de quatre de ces propriétés satisfaites par la règle de limsup. Celles-ci permettent ensuite de définir un concept plus général de machine simulationnelle. Ce sont des modèles de machines pour lesquelles la règle limite satisfait ces propriétés. Et pour tous les modèles de machine simulationnelle possible, en utilisant ces quatre propriétés, on prouve qu'il existe une machine universelle. Ensuite, en utilisant ce résultat d'existence, le premier résultat de ce travail est un théorème qui établit, pour n'importe quel modèle de machine simulationnelle qui satisfait à deux autres contraintes, une égalité entre d'une part le supremum des temps de calcul de ces machines et d'autre part le supremum des ordinaux que ces machines peuvent écrire.

Le second résultat de ce travail se base sur ce premier résultat. Un corollaire immédiat de la première partie est le suivant : il n'existe que deux modèles de machine simulationnelle (et non pathologiques), à savoir les ITTMs avec la règle limsup et leur symétrique avec la règle liminf. Ainsi, pour produire des machines infinies d'ordre supérieur, il faut construire des machines à n symboles. C'est le second résultat : nous avons construit un modèle de machine simulationnelle à 3 symboles, strictement plus puissant que celui des ITTMs et pour lequel nous parvenons à établir les principaux résultats mettant en relation les ITTMs avec la théorie des ensembles.

ECOLE DOCTORALE

Titre : Vers des machines de Turing en temps infini d'ordre supérieur et à plusieurs symboles

Mots clés : Calculabilité, théorie des ensembles, machine de Turing, ordinaux

Résumé : Ce travail traite des machines de Turing infinies (ITTM) telles que développées par Hamkins et Lewis au début des années 2000. Plus particulièrement, il s'intéresse à leur généralisation. Un aspect notable de ces machines infinies est que, en comparaison du modèle classique de Turing, celles-ci sont presque aussi simples. Une ITTM a la même structure qu'une machine de Turing à trois rubans. Elle fait des calculs en temps ordinaux et à n'importe quelle étape, l'instantané de la machine permet de calculer, comme dans le modèle classique, l'instantané à l'étape suivante. Le seule différence est aux étapes limites : la tête de lecture est remise au début, la machine est dans un certain état spécial et la valeur de chaque cellule est la limite supérieure de l'historique de ses valeurs précédentes. Si le choix pour la tête de lecture et pour l'état limite sont d'une façon logique, le choix de la valeur limite pour les cellules peut apparaître arbitraire. Pourquoi pas une liminf ? Ou encore quelque chose de plus compliqué ? Finalement, la justification de cette règle limite est une corroboration : avec cette règle, Hamkins et Lewis ont montré que le modèle de machines qu'ils ont développé est robuste, puissant et se comporte bien.
L'objectif est de proposer des règles limites différentes de la règle limsup produisant donc des généralisations de ce modèle de machines. La plupart de preuves portant sur les ITTMs utilise une ≪ machine universelle ≫, c'est-à-dire une machine qui simule en parallèle toutes les autres ITTMs. Une une telle machine est en fait simple à définir ; mais cette simplicité est fortuite : il pourrait y avoir de nombreuses difficultés qui sont évitées grâce à plusieurs propriétés implicites des ITTMs. Nous avons ainsi mis en lumière un ensemble de quatre propriétés, satisfaites par la règle de limsup. Elles nous permettent de définir un concept plus général de machine simulationnelle : des modèles de machines dont la règle limite satisfait ces propriétés et pour lesquelles on prouve qu'il existe une machine universelle. Le premier résultat de ce travail est un théorème qui établit, pour ces modèles de machines auxquelles deux contraintes sont rajoutées, une égalité entre les temps de calcul et les ordinaux qui peuvent être écrits.
Le second résultat principal se base sur ce premier résultat. Un corollaire immédiat de la première partie est le suivant : il n'existe que deux modèles de machine simulationnelle (et non pathologiques), à savoir les ITTM avec la règle limsup et leur symétrique avec la règle liminf. Ainsi, pour produire des machines infinies d'ordre supérieur, il faut construire des machines à n symboles. C'est le second résultat : nous avons construit un modèle de machine simulationnelle à 3 symboles, strictement plus puissant que celui des ITTMs et pour lequel nous parvenons à établir les principaux résultats mettant en relation les ITTMs avec la théorie des ensembles.

Title : Toward higher-order and many-symbol infinite time Turing machines

Keywords : Computability, set theory, Turing machine, ordinals

Abstract : This thesis studies infinite time Turing machines (ITTM) as developed by Hamkins and Lewis at the beginning of the years 2000. In particular, it aims at providing new generalizations of this model of infinite computation, or the tools and the results to develop those. A notable aspect of this model of infinite computation is that it is simple enough when compared to the usual finite model of Turing machines: an ITTM has the same structure as a three tapes Turing machine, it computes through the ordinals and at any successor stage, the next snapshot of the machine is a function of its machine code and the actual snapshot, as done in the classical setting. The only difference being that, at limit, tape heads are back on their first cells, the state is set to some distinguished limit state and the value of any cell is set to the limit superior of its previous values. While the choices for the heads and the states at limit stages may appear somewhat canonical, the principal justification for the rule of the limsup is actually a corroboration: with this rule, Hamkins and Lewis showed how this produces a robust, powerful and well-behaved model of infinite computation. So this work was focused on devising limit rules that would yield more powerful but equally well-behaved models of generalized infinite Turing machines. Most of the proofs done on ITTMs use a universal machine: an ITTM which simulates in parallel all other ITTMs. It happens to be straightforward to define such an universal ITTM. But its definition is only fortuitously straightforward. This construction rests on strong but implicit properties of the limsup rule. Hence, we exhibit a set of four properties satisfied by the limsup rule that allow us to define the more general concept of simulational machine: a model of infinite machines whose machines compute with a limit rule that satisfy this set of four properties, for which we prove that there exists a universal machine. The first main result is that the machines in this class of infinite machines satisfy (with two other constraints) an important equality satisfied by the usual ITTM, relating the time of computations and the ordinals that are writable.
The second main result builds on the previous result. An immediate corollary is the following: there exists only two 2-symbol simulational and "well-behaved" model of ITTM; namely the limsup ITTM and the liminf ITTM. So, to produce higher-order machines, we need to consider n-symbols machine. And this is the second result: we construct a 3-symbol ITTM, strictly more powerful that the previous one and for which we establish the same set-theoretic results that were established for it.

Institut Polytechnique de Paris
91120 Palaiseau, France