TEL AVIV UNIVERSITY אוניברסיטת תל-אביב

The Raymond and Beverly Sackler Faculty of Exact Sciences
The Blavatnik School of Computer Science

The Influence of Domain Interpretations on Computational Models

Thesis submitted for the degree of Doctor of Philosophy
by

Udi Boker

This work was carried out under the supervision of
Prof. Nachum Dershowitz

Submitted to the Senate of Tel Aviv University
October 2008

Acknowledgments

This PhD thesis would not have been written without my supervisor, Professor Nachum Dershowitz. The obvious reason is him being a full partner to the ideas, developments, and results of the thesis. But much beyond this, I would like to deeply thank him for all the guidance and support I received from him throughout the way. Starting from his very special ideas that sometimes seemed unreasonable at the beginning but down the road were found to be of great potential, through his deep insights, and ending with his uncompromising fine-tuning that included an almost uncountable number of proofreading cycles. All that in an always pleasant, calm and supporting way.
Thank you so much Nachum.

Abstract

Computational models are usually defined over specific domains. For example, Turing machines are defined over strings, and the recursive functions over the natural numbers. Nevertheless, one often uses one computational model to compute functions over another domain, in which case one is obliged to employ a representation, mapping elements of one domain into the other. For instance, Turing machines (or modern computers) are understood as computing numerical functions, by interpreting strings as numbers, via a binary or decimal representation, say.

We ask: Is the choice of the domain interpretation important? Clearly, complexity is influenced, but does the representation also affect computability? Can it be that the same model computes strictly more functions via one representation than another? We show that the answer is "yes", and further analyze the influence of domain interpretation on the extensionality of computational models (that is, on the set of functions computed by the model).

We introduce the notion of *interpretation-completeness* for computational models that are basically unaffected by the choice of domain interpretation, and prove that Turing machines and the recursive functions are interpretation-complete, while two-counter machines are incomplete.

We continue by examining issues based on model extensionality that are influenced by the domain interpretation. One such issue is the power comparison of computational models, which leads us to suggest a notion for comparing computational power of models operating over arbitrary domains.

A central motivation to understand the influence of domain interpretation on computational models is the notion of "effective computation". Church's original thesis, which later became the Church-Turing Thesis, concerned effectiveness of functions over the natural numbers. Yet, one often wants to use the thesis for referring to effectiveness over an arbitrary domain. For achieving this general notion of effectiveness, one has to understand first the influence of domain interpretations on Turing machines and to use a power comparison notion over arbitrary domains. Equipped with our result on the interpretation-completeness of Turing machines and our general comparison notion, we provide a general notion of effectiveness by interpreting the Church-Turing Thesis over arbitrary domains.

The thesis analyzes the subject via the following issues:

- Demonstrating the phenomenon – Generally, the extensionality of computational models is sensitive to the domain interpretation. This remains true even when restricting the representations to be only bijective mappings.

- Exploring the implications – Is there always an optimal representation, giving the best interpretation of a model? Are their models immune to the influence of the domain interpretation? How can we compare models over different domains? How best should one define the Church-Turing Thesis over arbitrary domains? How should one properly represent the natural numbers? These problems are addressed in the subsequent chapters of this dissertation.

- Identifying interpretation-complete models – There are computational models that are less influenced by the domain representation. We define and examine this completeness property, as well as some related extensional properties. We prove that some standard models are complete, among which are the recursive functions and Turing machines, while others are incomplete, like two-counter machines.

- Providing a comparison notion – We provide a notion for comparing the computational power of models operating over arbitrary domains. While this notion addresses the sensitivity problem to the domain representation, we raise additional concerns, this time from a more philosophical point of view. Accordingly, we suggest a conceptual framework for resolving these concerns and a corresponding mathematical notion of comparison.

- Effective computation – We provide an interpretation of the Church-Turing Thesis over arbitrary domains, based on the completeness of Turing machines and our power comparison notion. As a result, we provide a definition of an "effective string representation" of the natural numbers. In addition, we propose an axiomatization of an "effective model of computation" over an arbitrary countable domain. This axiomatization is based on Gurevich's postulates for sequential algorithms. A proof is provided showing that all models satisfying these axioms, regardless of underlying data structure, are of equivalent computational power to, or weaker than, Turing machines.

Contents

List of Figures

Chapter 1

Introduction

We explore the problem of the sensitivity of models to the domain interpretation, and propose a way to handle it. This introductory chapter parallels the structure of the thesis, as illustrated in Figure 1.1.

Sensitivity to domain interpretation. A computational model is defined over a specific domain. However, we may often use it to compute functions over a different domain. For example, using Turing machines (or modern computers) to compute functions over natural numbers requires a string representation of numbers. Another example becomes apparent when comparing the computational power of two models that operate over different domains – we are obliged to represent the domain of one model by the domain of the other. Accordingly, the elements of the original domain are interpreted as elements of the target domain (see illustration in Figure 2.1). A "representation" is usually allowed to be any mapping from one domain into another, as long as it is injective. That is, every original domain element is mapped to a unique element of the second domain. A representation may also be a relation (rather than a function), as long as two entities representing the same element behave the same in the relevant context (see, for example, [51]). The generalization of representations to relations does not eliminate the sensitivity of models to the domain interpretation.

It turns out that interpreting the domain allows for the possibility that a model be identified with one of its (strict) supermodels. The interpretation might allow one to "enlarge" the extensionality of a model, adding some "new" functions to it. A study of the sensitivity of models to interpretations via injective representations is undertaken in Section 2.1.

A reasonable response might be to restrict representations to bijections between domains. However, it turns out that there are models that can be

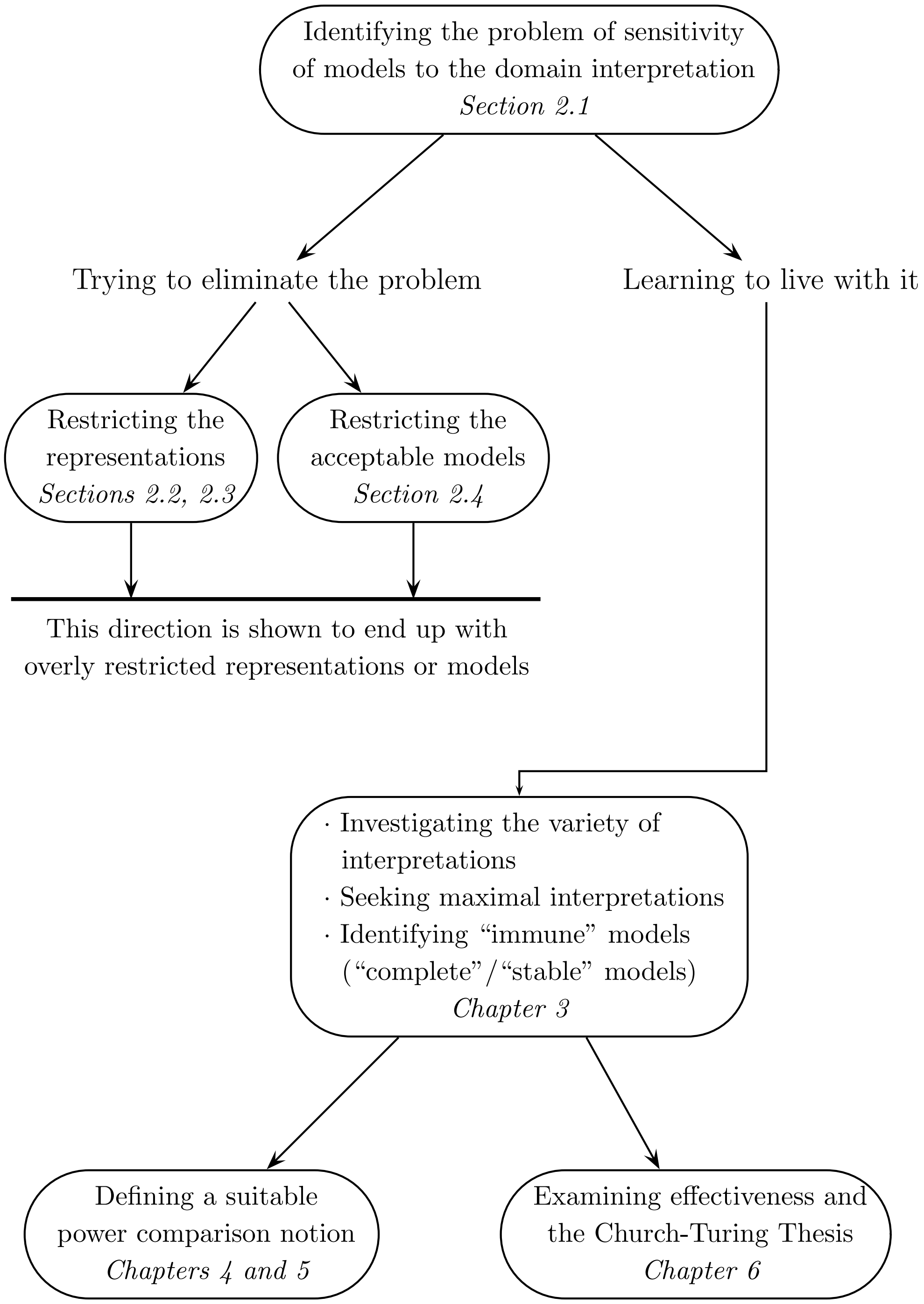

Identifying the problem of sensitivity of models to the domain interpretation *Section 2.1*
Trying to eliminate the problem
Learning to live with it
Restricting the representations *Sections 2.2, 2.3*
Restricting the acceptable models *Section 2.4*
This direction is shown to end up with overly restricted representations or models
· Investigating the variety of interpretations
· Seeking maximal interpretations
· Identifying "immune" models ("complete"/"stable" models)
Chapter 3
Defining a suitable power comparison notion *Chapters 4 and 5*
Examining effectiveness and the Church-Turing Thesis *Chapter 6*

Figure 1.1: Thesis structure

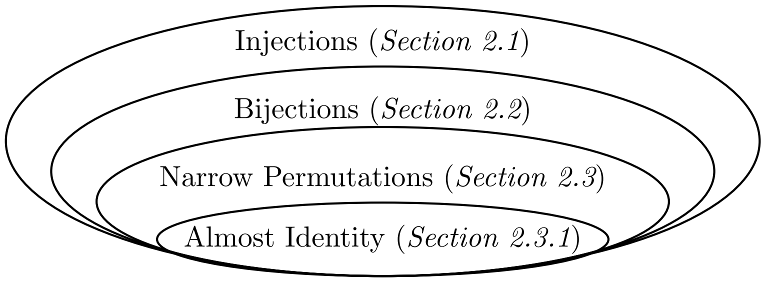

Injections (*Section 2.1*)
Bijections (*Section 2.2*)
Narrow Permutations (*Section 2.3*)
Almost Identity (*Section 2.3.1*)

Figure 1.2: Seeking representations that cannot influence the extensionality of models leaves us only with mappings that are almost the identity

identified with a supermodel even with bijective representations. That is, their extensionality is isomorphic to the extensionality of some of their strict supermodels. The case of bijective representations is explored in Section 2.2.

If bijective representations are not stringent enough, which representations are guaranteed not to influence the extensionality of computational models? It turns out that only very limited representations are, namely, "narrow permutations" (Definition 10). A sufficient and necessary criterion for these "harmless representations" is given in Section 2.3. Not only are narrow permutations a very limited family of representations, they are also not closed under composition. Thus, seeking harmless representations for comparative purposes would lead to an even more limited family of permutations that are almost the identity (Definition 13). Hence, sticking to harmless representations is not a viable option, as it almost completely evaporates the concept of interpretations. A scheme of the families of representations involved is given in Figure 1.2.

Another direction for avoiding the influence of representations could be narrowing down the definition of a computational model. Defining, for example, that the set of computed functions must be closed under functional composition. It turns out, however, that these standard computational properties are insufficient, as demonstrated in Section 2.4.

The implications. This sensitivity to the domain interpretation places a question mark on some of the main issues that concern model extensionality: How can we compare models over different domains? How should one define effective computation over arbitrary domains? How should one properly represent the natural numbers? Is there always an optimal representation? Are there models immune to the influence of the domain interpretation? These problems are briefly answered below, and more comprehensively addressed in Chapters 3, 4 and 6.

Organizing model interpretations. Generally, the various interpretations of a model may be highly varied: they may be larger or smaller than the original; for some models there are maximal interpretations while for others there are none; and there are models that are already maximal. This variety of possibilities is explored in Section 3.1.

The last property, that a model is already maximally interpreted, is the one that interests us most. We call such a model "interpretation-complete". We also define a weaker property, denoted "interpretation-stable", saying that a model is maximal with respect to bijective representations. When allowing only bijective representations, there are exactly two ways in which the interpretation may influence the model's extensionality: stable models are totally immune, in the sense that they have no better or worse interpretations via bijective representations, while unstable models have no maximum, nor minimum, interpretation via bijective representations (see the illustration in Figure 3.2). When allowing non-bijective representations, the picture is different – there might be a complete model with interpretations that are strictly contained in its original extensionality (see Figure 3.3). Interpretation-completeness and interpretation-stability, as well as some means for getting maximal interpretations, are investigated in Section 3.2.

In Section 3.3, we check for the completeness of some standard models. Turing machines and the recursive functions are shown to be complete, while two-counter machines and the untyped lambda calculus (over all lambda terms) are incomplete. As for hypercomputational models, they might be incomplete, though those preserving the closure properties of the recursive functions are ensured to be interpretation-complete.

Comparing computational power. It is common practice to compare the computational power of different models of computation. For example, the recursive functions are considered to be strictly more powerful than the primitive recursive functions, because the latter are a proper subset of the former, which includes Ackermann's function (see, for example, [45, p. 92]). Side-by-side with this "containment" method of measuring power, it is also standard to base comparisons on interpretations over different domains [17, 38, 44]. For example, one says that the (untyped) lambda calculus is as powerful – computationally speaking – as the partial recursive functions, because the lambda calculus can compute all partial recursive functions by encoding the natural numbers as Church numerals.

The problem is that unbridled use of these two distinct ways of comparing power allows one to show that some computational models are *strictly* stronger than themselves!

We define a comparison notion over arbitrary domains based on model interpretations. With this notion, model B is strictly stronger than A if B has an interpretation that contains A, whereas A cannot contain B under *any* interpretation. We provide, in Section 4.1, three variants of the comparison notion, depending on the allowed interpretations. In Sections 4.1.1 and 4.1.2 we extend the notion to non-deterministic models, and to interactive models. We continue, in Section 4.2, with some results on the relations between power comparison, isomorphism and completeness. In Section 4.3, we use the notion to compare some standard models. In Section 4.4 we redefine the comparison notion, using category-theory tools.

We follow this, in Chapter 5, with some conceptual discussions and justifications for the comparison notions.

Effective computation. Let f be some decision function (a Boolean-valued function) over an arbitrary countable domain D. What does one mean by saying that "f is computable"? One most likely means that there is a Turing machine M, such that M computes f, *using some string representation of the domain D*. But what are the allowed string representations? Obviously, allowing an arbitrary representation (any injection from D to Σ^*) is problematic – it will make any decision function "computable". For example, by permuting the domain of machine codes, the halting function can morph into the simple parity function, which returns true when the input number is even, representing a halting machine, and false otherwise. Thus, under a "strange" representation the function becomes eminently "computable" (see Section 6.1). Another approach is to allow only "natural" or "effective" representations. However, in the context of defining computability, one is obliged to resort to a vague and undefined notion of "naturalness" or of "effectiveness", thereby defeating the very purpose of characterizing computability.

Our approach to overcoming the representation problem is to ask about effectiveness of a set of functions over the domain of interest, rather than of a single function (Section 6.1). As Myhill observed [34], undecidability is a property of *classes* of problems, not of individual problems. In this sense, the halting function is undecidable in conjunction with an interpreter (universal machine) for Turing machine programs that uses the same representation. The Church-Turing Thesis, interpreted accordingly, asserts that there is no effective computational model that is more inclusive than Turing machines.

Nonetheless, there might have been a serious problem due the sensitivity of models to the domain interpretation (see Chapter 2). Fortunately, this cannot be the case with Turing machines (nor with the recursive functions), as they are interpretation-complete (Theorem 31). Hence, the Church-Turing Thesis is well-defined for arbitrary computational models.

Due to this completeness of Turing machines, we can also sensibly define what it means for a string representation of a constructible domain to be "effective" (Section 6.2). Such a representation ρ is effective when the domain's constructing functions are Turing computable via ρ (Definition 94). Hence, one may ask about the effectiveness of a single function over a constructible domain, provided that the means of construction of the domain are defined and are computable.

Axiomatizing effectiveness. Equipped with a plausible interpretation of the Church-Turing Thesis over arbitrary domains, we investigate, in Section 6.3, the general class of "effective computational models". We proffer an axiomatization of this class, based on Yuri Gurevich's postulates for a sequential algorithm [24]. The thesis is then proved, in the sense that a proof is provided that all models satisfying these axioms are of equivalent power to, or weaker than, Turing machines.

Gurevich's postulates are a natural starting point for computing over arbitrary domains. They are applicable for computations over any mathematical structure and aim to capture any sequential algorithm. Nevertheless, while the computation steps are guaranteed to be algorithmic, that is, effective, the initial states are not. In addition, the postulates refer to a single algorithm, while effectiveness should consider, as explained above, the whole computational model. We address the effectiveness of the initial state by adding a fourth axiom to the three of Gurevich. The effectiveness of an entire computational model is addressed by providing a minimal criterion for two sequential algorithms to be in the same model.

This direction of research follows Shoenfield's suggestion [41, p. 26]:

> [I]t may seem that it is impossible to give a proof of Church's Thesis. However, this is not necessarily the case.... In other words, we can write down some axioms about computable functions which most people would agree are evidently true. It might be possible to prove Church's Thesis from such axioms.

In fact, Gödel has also been reported (by Church in a letter to Kleene cited by Davis in [18]) to have thought "that it might be possible ... to state a set of axioms which would embody the generally accepted properties of [effective calculability], and to do something on that basis".

Thanks to Gurevich's Abstract State Machine Theorem, showing that sequential abstract state machines (ASMs) capture all (ordinary, sequential) algorithms (those algorithms that satisfy the three Abstract State Machine postulates), we get a third definition of an effective computational model: A model that consists of ASMs that share initial states satisfying the initial-state axiom.

The specifics of our effectiveness axiom may perhaps be arguable. Nevertheless, it demonstrates the possibility of such an axiomatization of effectiveness for *arbitrary domains*, and provides evidence for the validity of the Church-Turing Thesis, regardless of underlying data structure and internal mechanism of the particular computational model.

The relationship of the three approaches to characterizing effectiveness over arbitrary domains is summarized in Section 6.3.3 and depicted in Figure 6.1.

Related work. Usually, the handling of multiple domains in the literature is done by choosing specific representations, like Gödel numbering, Church numerals, unary representation of numbers, etc. This is also true of the usual handling of representations in the context of the Church-Turing Thesis.

A more general approach for comparing the power of different computational models is to allow any representation based on an injective mapping between their domains. This is done, for example, by Rogers [38, p. 27], Sommerhalder [44, p. 30], and Cutland [17, p. 24]. A similar approach is used for defining the effectiveness of an algebraic structure by Rice [37], Rabin [35], Fröhlich and Shepherdson [21], and Mal'tsev [29]. Our notion of comparing computational power (Definition 52) is based on this approach.

Richard Montague [32] raises the problem of representation when applying Turing's notion of computability to other domains, as well as the circularity in choosing a "computable representation". His citation is given in Section 6.1.

Stewart Shapiro [40] raises the very same problem of representation when applying computability to number-theoretic functions. He suggests a definition of an "acceptable notation" (string representation of natural numbers), based on some intuitive concepts. We discuss his notion in Sections 6.1 and 6.2.

Klaus Weihrauch [50, 51] deals heavily with the representation of arbitrary domains by numbers and strings. He defines computability with respect to a representation, and provides justifications for the effectiveness of the standard representations. We elaborate on his justifications in Section 6.2.

Our definition of an “effective representation” (Definition 94) resembles Shapiro’s notion of “acceptable notation” and goes along the lines of Weihrauch’s justifications for the effectiveness of the standard representations. We compare between these approaches in Section 6.2.

As for the axiomatization of effectiveness, several different approaches have been taken over the years. Turing [48] already formulated some principles for effective sequential deterministic symbol manipulation: finite internal states; finite symbol space; external memory that can be represented linearly; finite observability; and local action.

Robin Gandy [22], and later Sieg and Byrnes [43], define a model whose states are described by hereditarily finite sets. Effectiveness of Gandy machines is achieved by bounding the rank (depth) of states, insisting that they be unambiguously assemblable from individual “parts” of bounded size, and requiring that transitions have local causes.[1]

In [19], Dershowitz and Gurevich provide an axiomatization of Church’s Thesis based on the Abstract State Machine Thesis. They handle only numeric functions, ignoring the issue of effective computation over arbitrary domains, but allowing the use of domains richer than just the numbers.

New results. To the best of our knowledge, our work in [10, 8, 11] was the first to point out and handle the possible influence of the representation on the extensionality of computational models. In [12] we developed our interpretation of the Church Turing Thesis, based on our work on the interpretation of computational models.

Most of the results given in this thesis are published in papers of the author and his supervisor. They are roughly divided as follows:

- Chapter 2 (except for Sections 2.3.1, 2.4) – in [10];
- Chapter 3 (except for Sections 3.3.4, 3.3.5, 3.4) – in [7];
- Chapter 4 (except for Section 4.4) – in [10, 8, 11, 7];
- Chapter 5 – in [8, 11];
- Chapter 6 – in [9, 12]

Most of the results in Chapter 2 and some preliminary results of Chapters 3 and 4 were already given in the author’s M.Sc. thesis.

[1]The explicit bound on rank is removed in Sieg’s more recent work [42].

Terminology. We refer to the natural numbers, denoted $\mathbb{N}$, as including zero, and denote by $\mathbb{N}^+$ the natural numbers excluding zero. When we speak of the recursive functions, denoted $\mathbb{REC}$, we mean the partial recursive functions. Similarly, the set of Turing machines, denoted TM, includes both halting and non-halting machines. We use the term "domain" of a computational model and of a (partial) function to denote the set of elements over which it operates, not only those for which it is defined. By "image", we mean the values that a function actually takes: $\mathrm{Im}\, f := \{f(x) \mid x \in \mathrm{dom}\, f\}$.

Chapter 2

The Sensitivity of Models to the Domain Interpretation

Computational models. Our research concerns computational models. Obviously, a computational model should perform some computation; however demarcating a clear border between what is a computational model and what is not is problematic. Accordingly, for achieving maximum generality, we do not want to limit computational models to any specific mechanism; hence, we allow a model to be any object, as long as it is associated with a set of functions that it computes.

As models may have non-terminating computations, we deal with sets of partial functions. For convenience, we assume that the domain and range (co-domain) of functions are identical. For simplicity, we mainly deal with deterministic computational models. Most of our definitions and theorems can be directly extended to non-determinism, while the more involved ones are handled in Section 4.1.1.

Definition 1 (Computational Model)**.**

- *A* domain D *is any set of elements.*
- *A* computational model A *over domain D is any object associated with a set of partial functions $f : D^i \to D$, for an arity $i \in \mathbb{N}^+$. This set of functions is called the* extensionality *of the computational model, denoted* $[\![A]\!]$.
- *We write* dom A *for the domain over which model A operates.*

In what follows, we often speak of a "submodel" or a "supermodel" of a model, referring to the containment relation between their extensionalities:

Definition 2 (Submodel and Supermodel)**.**

- *A computational model A is called a* submodel *of a model B if $[\![A]\!] \subseteq [\![B]\!]$. In such a case we also refer to B as a* supermodel *of A.*
- *A computational model A is called a* strict submodel *of a model B if $[\![A]\!] \subsetneq [\![B]\!]$. In such a case we also refer to B as a* strict supermodel *of A.*

In the following sections we explore the sensitivity of models to domain interpretations, ending up with a sufficient and necessary condition for a "harmless representation" (see Figure 1.2).

2.1 Injective representations

Injective representations are the most frequently used ones. The standard decimal and binary notations of the natural numbers are injective (they are not bijective since leading zeros are ignored). Comparisons between computational models are usually done by injective encodings; for example: Church numerals and Gödel numbering are used for comparing the recursive functions and λ-calculus.

We begin by defining a "representation" to be an injective mapping.

Definition 3 (Representation)**.**

Domain. *Let D_A and D_B be two domains (arbitrary sets of atomic elements). A* representation *of D_A over D_B is an injection $\rho : D_A \to D_B$ (i.e. ρ is total and one-one).*

Function and Relation. *Representations ρ naturally extend to functions and relations f, which are sets of tuples of domain elements: $\rho(f) := \{\langle \rho(x_1), \ldots, \rho(x_n) \rangle \mid \langle x_1, \ldots, x_n \rangle \in f\}$.*

Model. *Representations also naturally extend to (the extensionalities of) computational models, which are sets of functions: $\rho([\![B]\!]) := \{\rho(f) \mid f \in [\![B]\!]\}$.*

The representation concept is illustrated in Figure 2.1 and 2.2.

An almost dual concept to representation is "interpretation" (see Figures 2.1 and 2.3):

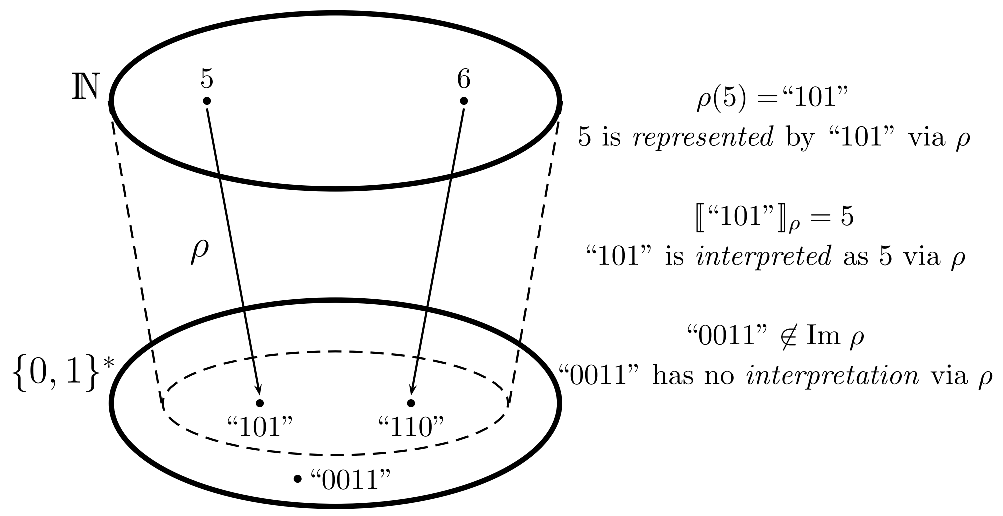

$\mathbb{N}$
5
6
ρ
$\{0,1\}^*$
"101"
"110"
"0011"
$\rho(5) =$"101"
5 is *represented* by "101" via ρ
$[\![$"101"$]\!]_\rho = 5$
"101" is *interpreted* as 5 via ρ
"0011" $\notin \mathrm{Im}\ \rho$
"0011" has no *interpretation* via ρ

Figure 2.1: Domain representation. The natural numbers are represented by binary strings via the representation ρ

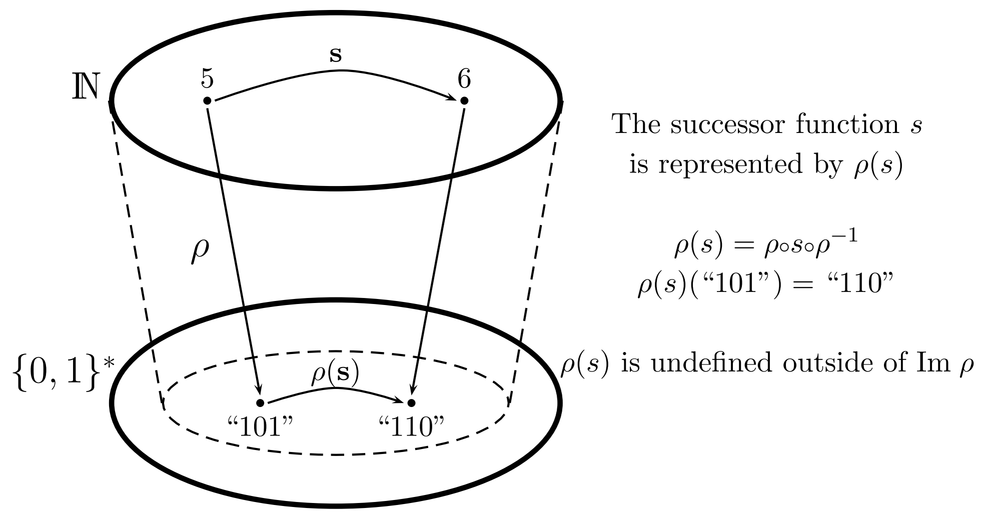

$\mathbb{N}$
5
$\mathbf{s}$
6
ρ
$\{0,1\}^*$
$\rho(\mathbf{s})$
"101"
"110"
The successor function s
is represented by $\rho(s)$
$\rho(s) = \rho \circ s \circ \rho^{-1}$
$\rho(s)($"101"$) =$ "110"
$\rho(s)$ is undefined outside of Im ρ

Figure 2.2: Function representation. The numeric functions are represented by string functions via the representation ρ

Definition 4 (Interpretation)**.** *Assume a representation $\rho : D_A \to D_B$. Then:*

1. *The* interpretation of a domain element $b \in D_B$ *via the representation ρ, denoted $[\![b]\!]_\rho$, is the element $\rho^{-1}(b)$ of D_A. If $b \notin \operatorname{Im} \rho$ then its interpretation via ρ is undefined.*

2. *The* interpretation of a function g *over D_B via the representation ρ, denoted $[\![g]\!]_\rho$, is the function $\rho^{-1}(g)$ over D_A, which is $\rho^{-1}{\circ}g{\circ}\rho$. If $g{\circ}\rho(a) \notin \operatorname{Im} \rho$ for some element $a \in D_A$ then $[\![g]\!]_\rho$ is a partial function, where $[\![g]\!]_\rho(a)$ is undefined.*

3. *The* interpretation of a computational model B *via the representation ρ, denoted $[\![B]\!]_\rho$, is the set of functions $\rho^{-1}([\![B]\!])$, which is $\{[\![g]\!]_\rho \mid g \in [\![B]\!]\}$.*

4. *When considering only total functions, the* interpretation of a total computational model B *via the representation ρ, denoted $[\![B]\!]_\rho$, is the set of functions $\{[\![g]\!]_\rho \mid g \in [\![B]\!]$ and $[\![g]\!]_\rho$ is total$\}$.*

"Interpretation via ρ" is the reverse of "representation via ρ", up to the image of the representation ($\operatorname{Im} \rho$). When the representation ρ is bijective we have that "interpretation via ρ" is exactly as "representation via ρ^{-1}."

Interpretation and extensionality share the same notation. Indeed, the interpretation of some model B via a representation $\rho : D_A \to D_B$ is its extensionality over the domain D_A, which results from the representation ρ. Note that the extensionality $[\![A]\!]$ of a model A is its interpretation $[\![A]\!]_\iota$ via the identity representation ι.

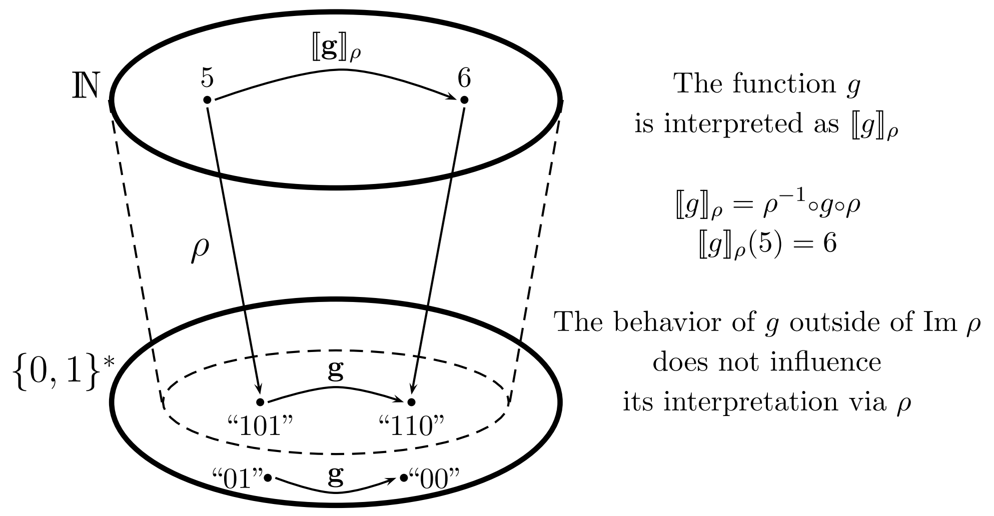
Proposition 5. *For all models A and B and representations ρ, $[\![A]\!] \subseteq [\![B]\!]$ implies that $[\![A]\!]_\rho \subseteq [\![B]\!]_\rho$.*

Sensitivity. Injective representations, however, are prone to hide some computational power. Below is a simple example of such a case.

Example 6. *The set of "even" recursive functions (R_2) can be interpreted as the set of all the recursive functions ($\mathbb{REC}$), by mapping the original natural numbers into the even numbers*

$$\begin{aligned} \rho &:= \lambda n.2n \\ R_2 &:= \left\{ \lambda n. \left\{ \begin{array}{ll} 2f(n/2) & n \text{ is even} \\ n & \text{otherwise} \end{array} \right\} \mid f \in \mathbb{REC} \right\} \end{aligned}$$

We have that $[\![R_2]\!]_\rho = \mathbb{REC} \supsetneq R_2$.



$\mathbb{N}$
5
6
$[\![\mathbf{g}]\!]_\rho$
ρ
$\{0,1\}^*$
g
"101"
"110"
"01"
"00"
The function g is interpreted as $[\![g]\!]_\rho$
$[\![g]\!]_\rho = \rho^{-1} \circ g \circ \rho$
$[\![g]\!]_\rho(5) = 6$
The behavior of g outside of Im ρ does not influence its interpretation via ρ

Figure 2.3: Function interpretation. The string functions are interpreted as numeral functions via the representation ρ

The above anomaly does not appear only with "synthetic" models, but also with some standard ones. An example of such a model is the standard two-counter machine model (see Section 3.3.4).

2.2 Bijective representations

The previous section demonstrated the sensitivity of models to injective representations. One may ask whether the restriction of representations to bijective mappings might solve the problem. We show that the answer is "no", obtaining that a model might be isomorphic to some of its strict supermodels.

Definition 7 (Isomorphism)**.** *Models A and B (or their extensionalities) are* isomorphic*, denoted $A \cong B$ (or $[\![A]\!] \cong [\![B]\!]$), if there is a bijection π such that $[\![A]\!]_\pi = [\![B]\!]$.*

In the rest of this section we provide an example of such a model. Since this phenomenon looks somewhat mysterious, we present in detail the model's mechanism and extensionality, as well as a bijective representation via which the model's interpretation strictly contains its original extensionality. In addition, we provide three bijective representations of numbers by strings via which we get three numeric interpretations of the model with strict containment between them.

The computational model is a "tailored" one, named "triangular machines" and denoted Tr. We currently don't have an example of a well-known model having the property under discussion.

Triangular machines – the mechanism.

- The machines get as input, and return as output, a finite string over the alphabet $\{a, b\}$.
- Each machine has a finite program, which must start with the command `Initialize`. This command changes all the letters of the input string to the letter "a".
- The program may have, in addition, any finite number of commands of the form `Set the` k`th letter to b`, where k is any positive natural number. This command changes the kth letter from the right to "b" if the string has at least k letters, and does nothing otherwise.
- Another command is `Add` k `a`, where k is any positive natural number. The command adds k instances of "a" to the left of the string.

A couple of examples:

Example 8. *The program*

```
Initialize
Set the second letter to b
Set the third letter to b
```

Changes the input string "ababa" into "aabba", the input "ab" into "ba", and "b" into "a".

Example 9. *The program*

```
Initialize
Add 3 a
Set the 4th letter to b
Set the 5th letter to b
```

Changes the input string "ababa" into "aaabbaaa", the input "ab" into "bbaaa", and "a" into "baaa".

Triangular machines – the extensionality. Imagine the finite strings over $\{a, b\}$ arranged in rows according to their length, and within each row according to their lexicographic order, as illustrated in Figure 2.4 (having a triangular shape).

0	ϵ							
1	a	b						
2	aa	ab	ba	bb				
3	aaa	aab	aba	abb	baa	bab	bba	bbb
4	aaaa	aaab	...					
$\vdots$		$\ddots$						
	0	**1**	**2**	**3**	**4**	**5**	**6**	...

Figure 2.4: A triangular arrangement explaining the original extensionality

0	ϵ							
1	b	a						
2	bb	aa	ab	ba				
3	bbb	aaa	aab	aba	abb	baa	bab	bba
4	bbbb	aaaa	aaab	...				
$\vdots$		$\ddots$						
	0	**1**	**2**	**3**	**4**	**5**	**6**	...

Figure 2.5: Interpretation via the representation τ

0	0							
1	1	2						
2	3	4	5	6				
3	7	8	9	10	11	12	13	14
4	15	16	...					
$\vdots$		$\ddots$						
	0	**1**	**2**	**3**	**4**	**5**	**6**	...

Figure 2.6: Interpretation via the representation π

0	0							
1	2	1						
2	6	3	4	5				
3	14	7	8	9	10	11	12	13
4	30	15	...					
$\vdots$		$\ddots$						
	0	**1**	**2**	**3**	**4**	**5**	**6**	...

Figure 2.7: Interpretation via the representation η

The extensionality of triangular machines is $[\![\mathrm{Tr}]\!] = f_{i,j}$ $(i, j \in \mathbb{N})$, where $f_{i,j}(s)$ changes the input string s into the ith string in the jth subsequent row to s's row, wrapping around for overly large i.

For example, the program of Example 9 computes the function $f_{24,3}$, while the program having only the `Initialize` command computes the function $f_{0,0}$.

Generally, we can translate every program P to an $f_{i,j}$-function in the following way:

1. Remove any duplicate commands of the form `Set the` k`th letter to b`. (Duplicated commands of the form `Add` k `a` are not redundant, though they can be changed to a single command.)

2. Set the indices i, j to zero $(i := 0, j := 0)$ by the `Initialize`. command.

3. For every command `Set the` k`th letter to b` add 2^{k-1} to the index i $(i := i + 2^{k-1})$.

4. For every command `Add` k `a` add k to the index j $(j := j + k)$.

Triangular machines – numeric interpretations. Representing the natural numbers by strings would yield a numeric interpretation of our model. Let $\pi : \mathbb{N} \to \{a, b\}^*$ be the bijective representation obtained by enumerating the strings over $\{a, b\}$ by their length and lexicographic order (See Figure 2.8). The numeric interpretation is analogous to the string extensionality, jumping between rows and columns, just that the elements are natural numbers. The triangular diagram will take an analogous form, as illustrated in Figure 2.6

The interpretation of the model via the representation π is:

$$\begin{aligned} [\![\mathrm{Tr}]\!]_\pi &= g_{i,j}(i, j \in \mathbb{N}), \text{ where} \\ g_{i,j}(n) &= 2^{row(n)+j} - 1 + i \bmod (2^{row(n)+j}), \text{ and} \\ row(n) &= \lfloor log_2(n+1) \rfloor \end{aligned}$$

One may think of a different bijective representation $\eta : \mathbb{N} \to \{a, b\}^*$, presented in Figure 2.8.

It turns out that the interpretation via η strictly contains the interpretation via π (that is, $[\![\mathrm{Tr}]\!]_\eta \supsetneq [\![\mathrm{Tr}]\!]_\pi$), adding the functions $\{\lambda n.2^{row(n)+j+1} - 2 \mid j \in \mathbb{N}\}$. The (infinitely many) additional functions can be visualized in Figure 2.6 as jumping to the last (rightmost) number in the jth subsequent row to the row of the input number.

	π	η	ξ
0	ϵ	ϵ	ϵ
1	a	b	b
2	b	a	a
3	aa	ab	bb
4	ab	ba	aa
5	ba	bb	ab
6	bb	aa	ba
7	aaa	aab	bbb
8	aab	aba	aaa
9	aba	abb	aab
10	abb	baa	aba
11	baa	bab	abb
12	bab	bba	baa
13	bba	bbb	bab
14	bbb	aaa	bba
15	aaaa	aaab	bbbb
16	aaab	aaba	aaaa
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 2.8: Various bijective representations of numbers by strings

Analogously, we can choose a different representation, having fewer functions. For example, the model's interpretation via the representation $\xi : \mathbb{N} \to \{a, b\}^*$, presented in Figure 2.8, lacks infinitely many functions, visualized in Figure 2.6 as jumping to the first (leftmost) number in the jth subsequent row to the row of the input number.

For a model that suffers from the influence of a bijective representation, as does the triangular machine model, the process above can be performed infinitely. That is, we can always choose a different representation via which we get more functions or fewer functions.

Isomorphism to a strict supermodel. Similarly to the numeric interpretations above, we can provide a bijective representation of the model's original domain, via which the model's interpretation strictly contains its original extensionality. Let $\tau : \{a, b\}^* \to \{a, b\}^*$ be the bijective representation $\tau := \eta \circ \pi^{-1}$. The triangular view obtained by this representation is illustrated in Figure 2.5.

We saw that $[\![\mathrm{Tr}]\!]_\eta \supsetneq [\![\mathrm{Tr}]\!]_\pi$. Since isomorphism preserves containment, it follows that $\pi([\![\mathrm{Tr}]\!]_\eta) \supsetneq \pi([\![\mathrm{Tr}]\!]_\pi)$, which is $\pi(\eta^{-1}([\![\mathrm{Tr}]\!])) \supsetneq \pi(\pi^{-1}([\![\mathrm{Tr}]\!]))$, which is $(\eta \circ \pi^{-1})^{-1}([\![\mathrm{Tr}]\!])) \supsetneq [\![\mathrm{Tr}]\!]$, which is $[\![\mathrm{Tr}]\!]_{\eta \circ \pi^{-1}} \supsetneq [\![\mathrm{Tr}]\!]$, getting that $[\![\mathrm{Tr}]\!]_\tau \supsetneq [\![\mathrm{Tr}]\!]$.

We have got an interpretation of the model that strictly contains its original extensionality. Note that since the representation is bijective, it follows that the model's extensionality is isomorphic to a strictly larger set of functions, or in other words, the model is isomorphic to a strict supermodel of itself.

Analogously, we may choose a representation via which the model's interpretation is strictly contained in its extensionality: $[\![\mathrm{Tr}]\!]_{\pi \circ \eta^{-1}} \subsetneq [\![\mathrm{Tr}]\!]$.

It will be shown, in Chapter 3, that this process is infinite and symmetric (Theorem 18) – once the model is sensitive to bijective representations, we can always choose a different representation via which we get more, or less, functions.

2.3 Harmless representations

Are there "harmless representations", via which all models are "protected" from having better and worse interpretations? The answer is "yes", however this family of representations is too limited for being really useful. It is exactly the family of what we call "narrow" permutations:

Definition 10 (Narrow Permutation)**.** *A permutation $\pi : D \to D$ is* narrow *if all its orbits (cycles) are bounded in length by some constant, that is, if $\exists k \in \mathbb{N}.\ \forall x \in D.\ |\{\pi^n(x) : n \in \mathbb{N}\}| \leq k$.*

Proposition 11. *A permutation $\pi : D \to D$ is narrow iff there is a positive constant $k \in \mathbb{N}^+$, such that for all $x \in D$ we have $\pi^k(x) = x$. In other words, if $\pi^k = \iota$.*

Proof. One direction is trivial. For the second, if π's orbits are bounded by k, we have $\pi^{k!}(x) = x$ for every $x \in D$. □

Theorem 12. *For every representation $\rho : D \to D$, there are models A and B such that $[\![A]\!]_\rho = [\![B]\!] \supsetneq [\![A]\!]$, if and only if ρ is not a narrow permutation.*

We generally consider representations to be injective, however for making the above theorem a more comprehensive one, we also handle in this proof the case of non-injective representations.

Proof. Suppose π is a narrow permutation with orbit size bounded by k, and assume by contradiction the existence of models A and B such that $[\![A]\!]_\pi = [\![B]\!] \supsetneq [\![A]\!]$. For every function $g \in [\![B]\!]$, there is, by assumption, some function $f_1 \in [\![A]\!]$, such that $\pi^{-1} \circ f_1 \circ \pi = g$. Since f_1 is also in $[\![B]\!]$, there is, by k-fold repetition, a function $f_k \in [\![A]\!]$, such that $f_k = \pi^{-k} \circ f_k \circ \pi^k = g$. Therefore, $[\![B]\!] = [\![A]\!]$, contradicting the assumption.

For the other direction, we must consider three cases: (i) non-surjective representations; (ii) surjective representations that are not injective; and (iii) bijections with no bound on the length of their orbits. We prove each case by constructing a computational model A over domain D that is interpreted as a strict supermodel of itself via the given representation ρ.

Case (i). Suppose ρ is non-surjective, and let $c \in D \setminus \mathrm{Im}\ \rho$. Define $[\![B]\!] = \{\lambda x.\rho^i(c) \mid i \in \mathbb{N}\}$ and $[\![A]\!] = [\![B]\!] \setminus \{\lambda x.c\}$. Since $c \notin \mathrm{Im}\ \rho$, it follows that $[\![A]\!] \subsetneq [\![B]\!]$. Since for all i we have that $\rho^{-1} \circ \lambda x.\rho^{i+1}(c) \circ \rho = \lambda x.\rho^i(c)$, it follows that $[\![A]\!]_\rho \supseteq [\![B]\!]$.

Case (ii). Suppose that ρ is surjective, but not injective, and let $c \in D$ be such that $\rho(a) = \rho(b) = c$, for some $a \neq b$ in D. Since ρ is a (single-valued) function, it follows that at least one of a and b, say a, is not in $\{\rho^i(c) \mid i \in \mathbb{N}\}$. So, let $[\![B]\!] = \{\lambda x.\rho^i(a) \mid i \in \mathbb{N}\}$ and $[\![A]\!] = [\![B]\!] \setminus \{\lambda x.a\}$. By the same argument as in case (i), we have $[\![A]\!]_\rho \supseteq [\![B]\!] \supsetneq [\![A]\!]$.

Case (iii). Suppose that ρ is an unbounded-orbit permutation. Let σ be a function that chooses a representative within each orbit: for all $x, y \in D$, $\sigma(x) = \sigma(y)$ iff $\rho^i(x) = y$ for some $i \in \mathbb{Z}$. Define $[\![B]\!] = \{\lambda x.\rho^i(\sigma(x)) \mid i \in \mathbb{N}\}$ and $[\![A]\!] = [\![B]\!] \setminus \{\lambda x.\sigma(x)\}$. Since the orbits of ρ are unbounded, it must be that $[\![A]\!] \subsetneq [\![B]\!]$. By the argument of case (i), we again have $[\![A]\!]_\rho \supseteq [\![B]\!]$. □

The family of narrow permutations is very limited and cannot be used as the only mean of interpreting models over different domains. Moreover, this family is not closed under composition. That is, there are narrow permutations π and η such that $\pi \circ \eta$ is not narrow! This situation is very problematic, since interpretations are often used in the context of order relations, for example when saying that two models have the same extensionality up to the domain interpretation. Any equivalence or order relation should be transitive, which is not the case if representations are limited to narrow permutations.

2.3.1 Purely harmless representations

Proceeding in the above direction of seeking a family of representations that would be harmless and closed under functional composition leads us to look for some strict subset of the narrow permutations. However, considering that there are many such (maximal) subsets, which is a reasonable choice?

It turns out that there is a clear distinction between two types of narrow permutations – the "problematic" and the "non-problematic" ones. For every problematic narrow permutation ρ there is a simple (problematic) narrow permutation η, such that $\rho \circ \eta$ is not narrow. On the other hand, a non-problematic narrow permutation π guarantees that for every narrow permutation ξ we have that $\xi \circ \pi$ and $\pi \circ \xi$ are narrow.

The family of non-problematic narrow permutations is the "almost identity" permutations (defined bellow), while the rest are problematic.

Definition 13 (Almost Identity). *A permutation $\pi : D \to D$ is* almost identity *if $|\{x \in D \mid \pi(x) \neq x\}| < \infty$.*

Theorem 14. *For every narrow permutation ξ and almost identity permutation π over domain D, we have that $\xi \circ \pi$ and $\pi \circ \xi$ are narrow permutations.*

Proof. Suppose that the permutation π changes k elements of a domain D, for some constant $k \in \mathbb{N}$. Define $S = \{x \in D \mid \pi \circ \xi(x) \neq \xi(x)\}$. Since π changes only k elements of D, it follows that $|S| = k$. Thus, a cycle of $\pi \circ \xi$ is a combination of at most k cycles of ξ, therefore it is bounded. By the same argument, $\xi \circ \pi$ is a narrow permutation. $\square$

Theorem 15. *For every narrow permutation ρ that is not almost identity there is a narrow permutation η with cycles of length ≤ 2, such that $\rho \circ \eta$ is not a narrow permutation.*

Proof. Since ρ is not almost identity, it follows that there is a denumerable set $S = \{C_1, C_2, \ldots\}$ of cycles of ρ of length ≥ 2. Let $m : S \to D$ be a function from the set of cycles S to the domain D, such that $m(C_i) \in C_i$, for all $C_i \in S$. Abbreviate, $m_i = m(C_i)$. Define the permutation

$$\tau(x) \quad = \quad \begin{cases} m_{i+1} & x = \rho(m_i) \text{ for some } i > 0 \\ \rho(m_{i-1}) & x = m_i \text{ for some } i > 1 \\ x & \text{otherwise} \end{cases}$$

Clearly, the cycles of τ are of length ≤ 2. Since for every $i \in \mathbb{N}$ we have that $(\tau \circ \rho)^i(m_1) \neq m_i$, it follows that $\tau \circ \rho$ is not a narrow permutation. $\Box$

The above results suggest that sticking to harmless representations is not a viable direction, as the concept of interpreting models of different domains almost completely evaporates.

2.4 Models with standard computational properties

It was shown above that restricting the family of applicable representations cannot solve the sensitivity problem. A different approach to this problem is to restrict the definition of a computational model. Nonetheless, in order to allow a variety of internal mechanisms, we seek a restriction on the model's extensionality. We consider four restrictions: (i) closure under functional composition; (ii) inclusion of the identity function; (iii) inclusion of all constant functions; and (iv) the successor function for models operating over $\mathbb{N}$.

In this section we show that the sensitivity problem remains, even when considering only models with all the above properties and allowing only bijective representations.

Closure under functional composition. Denote by $cl(\mathcal{F})$ the set $\mathcal{F}$ of functions closed under functional composition. A model closed under functional composition can still be isomorphic to a strict supermodel of itself. For example, Triangular machines (Section 2.2) are such a model. Furthermore, considering only models closed under functional composition does not change the sufficient and necessary condition for a harmless representation (Theorem 12), as all models involved in the proof are closed under functional composition.

The identity and constant functions. Adding, or removing, the identity function ι from a model has no influence on its sensitivity to any rep-

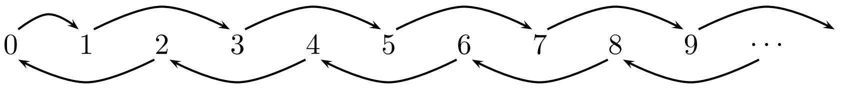


Figure 2.9: The permutation π of Example 16

resentation, as $\rho^{-1}\circ(f\circ\iota)\circ\rho = \rho^{-1}\circ(\iota\circ f)\circ\rho = \rho^{-1}\circ f\circ\rho$, for every injection ρ and function f.

Let K be the set of all constant functions over a domain D. Adding, or removing, K from a model A over D, such that $A \cap K \in \{K, \emptyset\}$, has no influence on the sensitivity of A, as $[\![A\circ K]\!]_\rho = [\![K\circ A]\!]_\rho = K$, with respect to total functions, for every injection ρ and model A.

The successor function. It turns out that a model including the successor function and closed under functional composition can still be isomorphic to a strict supermodel of itself. We demonstrate this with the following example:

Example 16. *Define the permutation π over $\mathbb{N}$,*

$$\pi(n) \quad := \quad \begin{cases} 1 & n = 0 \\ n+2 & n \text{ is odd} \\ n-2 & n \text{ is even} \end{cases}$$

The permutation π is illustrated in Figure 2.9.

Let s be the successor function over $\mathbb{N}$, and let A be a computational model with the extensionality $[\![A]\!] := \{\pi^i(s) \mid i \in \mathbb{N}\}$. Let B be the computational model obtained from A by closure under functional composition. That is, $[\![B]\!] := cl([\![A]\!])$.

We show hereby that B is isomorphic to a strict supermodel of itself, by showing that $[\![B]\!]_\pi \supsetneq [\![B]\!]$. We show that (i) $[\![B]\!]_\pi \supseteq [\![B]\!]$; and that (ii) there is a function, $s' := \pi^{-1}\circ s\circ\pi$, such that $s' \in [\![B]\!]_\pi$ while $s' \notin [\![B]\!]$.
(i) For showing that $[\![B]\!]_\pi \supseteq [\![B]\!]$ we need to show that for every function $f \in [\![B]\!]$ there is a function $f' \in [\![B]\!]$ such that $[\![f']\!]_\pi = f$. Since π is a bijection, it follows that $f' = \pi(f) = \pi\circ f\circ\pi^{-1}$. Now, a function $f \in [\![B]\!]$ is of the form $\pi^{i_1}(s)\circ\pi^{i_2}(s)\circ\ldots\circ\pi^{i_n}(s)$, and $\pi^i(s) = \pi^i\circ s\circ\pi^{-i}$. Hence, a function $f \in [\![B]\!]$ is of the form $\pi^{i_1}\circ s\circ\pi^{-i_1+i_2}\circ s\circ\pi^{-i_2+i_3}\ldots\pi^{-i_{n-1}+i_n}\circ s\circ\pi^{-i_n}$, for some $n \in \mathbb{N}$. Accordingly, $f' = \pi^{i_1+1}\circ s\circ\pi^{-i_1+i_2}\circ s\circ\pi^{-i_2+i_3}\ldots\pi^{-i_{n-1}+i_n}\circ s\circ\pi^{-(i_n+1)}$. Clearly, $f' = \pi^{i_1+1}(s)\circ\pi^{i_2+1}(s)\circ\ldots\circ\pi^{i_n+1}(s)$. Thus, $f' \in [\![B]\!]$.
(ii) Define $s' := \pi^{-1}\circ s\circ\pi = [\![s]\!]_\pi$. By definition, $s' \in [\![B]\!]_\pi$. Since $(\pi^i\circ s\circ\pi^{-i})(0)$ is odd, for all $i \in \mathbb{N}$, and $s'(0) = 4$, it follows that $s' \notin [\![A]\!]$. Since every function $f \in [\![B]\!] \setminus [\![A]\!]$ is composed of at least two functions that are composed of s and some permutations, it follows that $|\mathbb{N} \setminus \operatorname{Im} f| \geq 2 > 1 = |\mathbb{N} \setminus \operatorname{Im} s'|$. Thus, $s' \notin [\![B]\!] \setminus [\![A]\!]$, and therefore $s' \notin [\![B]\!]$.

Chapter 3

Organizing Model Interpretations

For examining the influence of the domain interpretation on a model we should compare its different interpretations. Accordingly, we are interested only in the model's interpretations over its original domain. Its interpretations over other domains are isomorphic to those over its original domain, as long as it is not a domain of a lower cardinality. For example, a model has two interpretations with strict containment between them if and only if it has such two interpretations over its original domain. A demonstration of shifting between the model's original domain and a different domain is given in Section 2.2.

We are interested in the containment relation between interpretations. That is, examining when some interpretations are better than others in the sense of strictly containing them. Accordingly, the interpretations of a model A form a partially ordered set with respect to containment (illustrated in Figure 3.1).

Viewing interpretations as a partially ordered set raises a few natural questions:

- How varied can these partially ordered sets be?
- Are there always maximal interpretations?
- Are there models already in their maximal interpretation (termed "interpretation-complete")?
- How does one choose a proper interpretation?

In what follows we will shed some light on the subject, considering the above questions and others. In Section 3.1 we answer the first two ques-

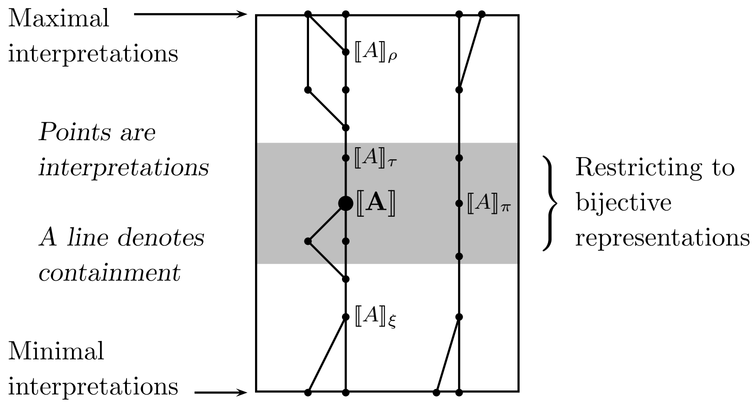


Figure 3.1: An illustration of the partially ordered set of interpretations of a model A

tions, showing how varied the set of interpretations can be. In Section 3.2 we answer the second pair of questions, dealing with the interpretation-completeness of models. Interpretation-completeness is the property that we payed the most attention to in our research, as it assures that the original model's extensionality is already its best interpretation.

3.1 The variety of interpretations

In general, the set of interpretations may be very varied:

- Some interpretations may be better than the original extensionality, while others are worse.
- There might be infinitely many interpretations each contained in the next.
- There might be models with no maximal interpretation!
- Non-bijective interpretations might sometimes add to bijective ones, while in other cases only spoil.
- There are models already having their maximal interpretation ("interpretation-complete") or at least so with respect to bijective representations ("interpretation-stable").

We prove the above assertions below, and investigate in detail the completeness and stability properties in the next section (Section 3.2).

A simple example of how different interpretations may enlarge or decrease the original extensionality is given in Example 6. Interpreting the model via the representation $\lambda n.2n$ enlarges the original extensionality, providing all the recursive functions. On the other hand, interpreting the model via the representation $\lambda n.2n{+}1$ decreases the original extensionality, leaving only the identity function.

We saw, in Section 2.2, that a model can be isomorphic to a strict supermodel of itself. In such a case, there are infinitely many interpretations enlarging the original extensionality, as well as infinitely many decreasing it, while each is contained in the next. We prove it below with the help of the following lemma. Note that this is true for bijective representations, but not necessarily for injective representations.

Lemma 17. *For all models A and B and bijections π, $[\![A]\!] \subsetneq [\![B]\!]$ implies that $[\![A]\!]_\pi \subsetneq [\![B]\!]_\pi$ and $\pi([\![A]\!]) \subsetneq \pi([\![B]\!])$.*

Proof. Since π is a bijection, it follows that $\pi^{-1} \circ f \circ \pi \neq \pi^{-1} \circ g \circ \pi$, for every $f \neq g$. Thus, $[\![B]\!]_\pi$ makes an injection between the set of functions in the original extensionality and the set of functions in the interpretation (i.e. every function of B is interpreted as a unique function via π). Therefore, $[\![B \setminus A]\!]_\pi \cap [\![A]\!]_\pi = \emptyset$. As for the second claim, when restricting to bijective representations, a model representation $\pi([\![A]\!])$ is exactly as the model interpretation $[\![A]\!]_{\pi^{-1}}$ via the inverse representation. $\square$

Theorem 18. *If A is a model and π a bijection such that $[\![A]\!] \subsetneq [\![A]\!]_\pi$, then for every $i \in \mathbb{N}$ we have that $[\![A]\!]_{\pi^i} \subsetneq [\![A]\!]_{\pi^{i+1}}$ and $[\![A]\!]_{\pi^{-i}} \supsetneq [\![A]\!]_{\pi^{-(i+1)}}$.*

Proof. By assumption, $[\![A]\!] \subsetneq [\![A]\!]_\pi$. Thus, it follows from Lemma 17 that $[\![A]\!]_\pi \subsetneq [\![[\![A]\!]_\pi]\!]_\pi$. From the definition of model interpretation, we have that $[\![[\![A]\!]_{\pi^i}]\!]_\pi = [\![A]\!]_{\pi^{i+1}}$. Hence, by induction on i we get that $[\![A]\!]_{\pi^i} \subsetneq [\![A]\!]_{\pi^{i+1}}$. As for the second part of the theorem, since π is a bijection, it follows that $[\![[\![A]\!]_\pi]\!]_{\pi^{-1}} = [\![A]\!]$. Therefore, by the same argument of the first part, we have that $[\![A]\!]_{\pi^{-i}} \supsetneq [\![A]\!]_{\pi^{-(i+1)}}$. $\square$

Corollary 19. *If A is a model for which there is no bijection π such that $[\![A]\!] \subsetneq [\![A]\!]_\pi$, then there is also no bijection η such that $[\![A]\!] \supsetneq [\![A]\!]_\eta$.*

The above theorem shows that once a model has a better interpretation via a bijective representation it cannot have a maximal interpretation via a bijective representation. Nevertheless, it might have a maximal interpretation via an injective representation. There are, however, models with no maximal interpretation at all.

We prove the above, with the help of the following a lemma.

Lemma 20. *Interpretations via bijective representations preserve the structure of every function. Specifically, for every function f over domain D and bijection $\pi : D' \to D$, we have that:*

1. *f is injective iff $[\![f]\!]_\pi$ is.*
2. $|D \setminus \text{Im } f| = |D' \setminus \text{Im } [\![f]\!]_\pi|$
3. *f is total iff $[\![f]\!]_\pi$ is.*

Proof.

1. Assume that f is not injective, and let $a, b \in D$ such that $f(a) = f(b)$. Define $a' = \pi^{-1}(a)$ and $b' = \pi^{-1}(b)$. We have that $[\![f]\!]_\pi(a') = \pi^{-1} \circ f \circ \pi(a') = \pi^{-1} \circ f \circ \pi(\pi^{-1}(a)) = \pi^{-1} \circ f(a) = \pi^{-1} \circ f(b) = \pi^{-1} \circ f \circ \pi(\pi^{-1}(b)) = [\![f]\!]_\pi(b')$. Hence $[\![f]\!]_\pi$ is also not injective. As for the other direction, if $[\![f]\!]_\pi$ is not injective with $[\![f]\!]_\pi(x) = [\![f]\!]_\pi(y)$ then f is not injective with $f(\pi(x)) = f(\pi(y))$.
2. If $a \not\in \text{Im } f$ then obviously $\pi^{-1}(a) \not\in \text{Im } \pi^{-1} \circ f \circ \pi$, and if $x \not\in \text{Im } \pi^{-1} \circ f \circ \pi$ then obviously $\pi(x) \not\in \text{Im } f$. Hence, $|D \setminus \text{Im } f| = |D' \setminus \text{Im } [\![f]\!]_\pi|$
3. By similar arguments.

□

Next, we show that there are models with no maximal interpretation.

Proposition 21. *There are computational models with no maximal interpretation that extends their original extensionality. That is, there is a computational model A, such that for every representation ρ for which $[\![A]\!]_\rho \supseteq [\![A]\!]$ there is a representation η such that $[\![A]\!]_\eta \supsetneq [\![A]\!]_\rho$.*

Proof. We use the model A and the bijection π of Example 16, and show that A is such a model. recall that $[\![A]\!] := \{\pi^i(s) \mid i \in \mathbb{N}\}$, where s is the successor function. By the arguments of Example 16 we have that $[\![A]\!]_\pi \supsetneq [\![A]\!]$. Therefore, by Theorem 18, it does not have a maximal interpretation via a bijective representation. We will show hereby that it does not have an interpretation via a non-bijective representation that contains its original extensionality. Hence, it will follow that it does not have a maximal interpretation extending its original extensionality.

Assume, by contradiction, a non-bijective representation $\rho : \mathbb{N} \to \mathbb{N}$, such that $[\![A]\!] \subseteq [\![A]\!]_\rho$, and let $e \in \mathbb{N} \setminus \text{Im } \rho$. We have that $[\![A]\!] \subseteq [\![A]\!]_\rho =$

$\{\rho^{-1}\circ\pi^i\circ s\circ\pi^{-i}\circ\rho \mid i \in \mathbb{N}\}$. Since $[\![A]\!]$ has infinitely many total functions, so should have $[\![A]\!]_\rho$. Thus, there are infinitely many numbers $i \in \mathbb{N}$, such that $\pi^i\circ s\circ\pi^{-i}\circ\rho(n) \neq e$ for every $n \in \mathbb{N}$. Therefore, there are infinitely many numbers $i \in \mathbb{N}$, such that $\rho(n) \neq \pi^i\circ s^{-1}\circ\pi^{-i}(e)$ for every $n \in \mathbb{N}$. From the structure of the permutation π it follows that $\pi^i\circ s^{-1}\circ\pi^{-i}(e)$ is different for every i, no matter what is e. Hence, there are infinitely many elements in $\mathbb{N} \setminus \mathrm{Im}\ \rho$.

Since there are total functions in $[\![A]\!]_\rho$, we can choose a number $k \in \mathbb{N}$ such that $\rho^{-1}\circ\pi^k\circ s\circ\pi^{-k}\circ\rho$ is a total function. By Lemma 20, we have that $\pi^k\circ s\circ\pi^{-k}$ preserves the structure of the successor function s. Specifically, it is total and injective with only one element outside of its image. Thus, for any element $a \in \mathbb{N}$ there are only finitely many elements not in $\{(\pi^k\circ s\circ\pi^{-k})^i(a) \mid i \in \mathbb{N}\}$. Now, let $a \in \mathrm{Im}\ \rho$. Since $\rho^{-1}\circ\pi^k\circ s\circ\pi^{-k}\circ\rho$ is total, it follows that $(\pi^k\circ s\circ\pi^{-k})^i(a) \in \mathrm{Im}\ \rho$ for every $i \in \mathbb{N}$. This leaves only finitely many elements outside of Im ρ, contradicting the above result that there are infinitely many elements in $\mathbb{N} \setminus \mathrm{Im}\ \rho$. Therefore, there is no injective representation via which the interpretation of A contains its original extensionality. □

From the above proof we also get that interpretations via non-bijective representations might sometimes only decrease the model's extensionality, while we saw that in other cases they can further enlarge it on top of bijective ones.

3.2 Interpretation-completeness and interpretation-stability

We saw that the extensionality of computational models is sensitive to the domain interpretation. There are, however, models that are already in their maximal interpretation (called "interpretation-complete" or in short "complete"), or at least so with respect to bijective representations (called "interpretation-stable" or in short "stable").

Definition 22 (Completeness)**.** *A model A is* interpretation-complete *if there is no representation $\rho : \mathrm{dom}\ A \to \mathrm{dom}\ A$ such that $[\![A]\!]_\rho \supsetneq [\![A]\!]$.*

The completeness property above is the one that interests us the most. It simply says that the model already has its best interpretation.

Note that when considering only total functions, the interpretation of a model is also defined to contain only total functions. In such a case a model

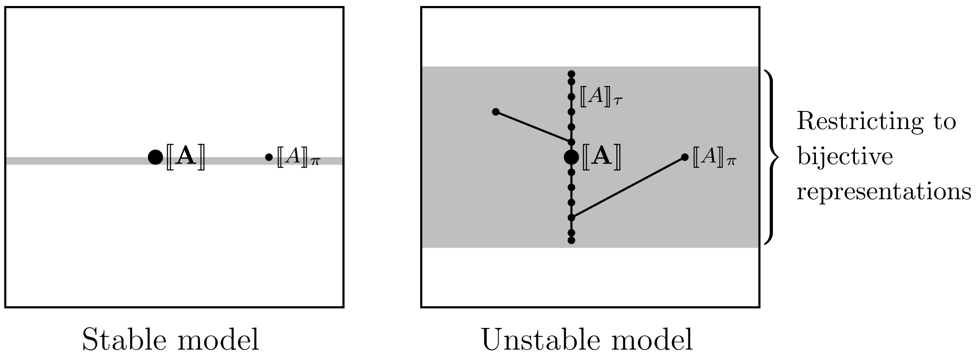


Figure 3.2: An illustration of interpretation-stable and unstable models

is considered complete even if some interpretations can extend it with partial functions.

Though we generally consider all injective representations, there are also good justifications to stick to bijective representations, as briefly seen in Chapter 2, and elaborated on in Chapters 5 and 6. Accordingly, we also define completeness with respect to bijective representations, called "interpretation-stability":

Definition 23 (Stability)**.** *A model A is* interpretation-stable *if there is no bijective representation* $\pi : \mathrm{dom}\, A \to \mathrm{dom}\, A$ *such that* $[\![A]\!]_\pi \supsetneq [\![A]\!]$.

Sticking to bijective representations, there are exactly two options for the representation influence:

- Stable model – totally immune to the influence of bijective representations. No better or worse interpretations are possible via bijective representations.
- Unstable model – there is no maximum, nor minimum, interpretation via bijective representations.

The above is illustrated in Figure 3.2, and proved in Theorem 18 and Corollary 19.

The general case, allowing non-bijective representations, is more varied:

- There are stable models that are incomplete.
- There might be a complete model with worse interpretations.
- Bijective representations preserve completeness.

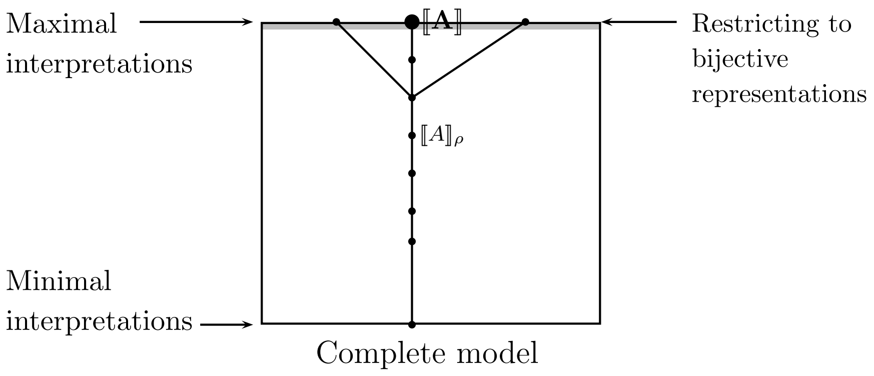


Figure 3.3: An illustration of an interpretation-complete model

The above is illustrated in Figure 3.3, and proved below.

Completeness obviously implies stability, but the opposite is not true. A simple example is the set of all constant functions except for a single constant function. A model having this extensionality is stable but incomplete (Theorems 44 and 45).

Completeness assures us that the model cannot have an interpretation better than its original extensionality. However, it might have an interpretation that decreases its original extensionality.

Proposition 24. *An interpretation-complete model might have an interpretation decreasing its extensionality. That is, there is an interpretation-complete model A and a representation ρ such that $[\![A]\!]_\rho \subsetneq [\![A]\!]$.*

Proof. Let A be a model over $\mathbb{N}$ with the following extensionality:

$$\begin{aligned} [\![A]\!] &:= \{f_k \mid k \in \mathbb{N}\} \cup \varepsilon, \text{ where} \\ & \quad\ \varepsilon \text{ is the function undefined on all elements, and} \\ f_k(n) &:= \begin{cases} k & n \text{ is even} \\ \perp & \text{otherwise} \end{cases} \end{aligned}$$

Define the representation $\rho := \lambda n.2n+1$. We have that $[\![f_k]\!]_\rho = \rho^{-1} \circ f \circ \rho = \varepsilon$, for every k. Hence, $[\![A]\!]_\rho = \{\varepsilon\} \subsetneq [\![A]\!]$. Now, assume by contradiction that A is incomplete. By assumption, we have a representation η such that $[\![A]\!]_\eta \supsetneq A$. Hence, there exists $i \in \mathbb{N}$ such that $[\![f_i]\!]_\eta = f_1$, as well as $j \in \mathbb{N}$ such that for all $k \in \mathbb{N}$, $[\![f_j]\!]_\eta \neq f_k$. Were η parity-preserving (i.e. $\eta(n)$ is even iff n is even), we have that $[\![f_j]\!]_\eta = \eta^{-1} \circ f_j \circ \eta = f_{\eta^{-1}(j)}$ if $j \in \text{Im}\ \eta$ and $[\![f_j]\!]_\eta = \varepsilon$ if $j \notin \text{Im}\ \eta$. Therefore, there is either an even number l such that $\eta(l)$ is odd, or an odd number m such that $\eta(m)$ is even. In the first case we have that $[\![f_i]\!]_\eta(l) = \bot \neq f_1(l) = 1$, and in the second case we have that $[\![f_i]\!]_\eta(m) = \eta^{-1}(i) = 1 \neq f_1(l) = \bot$. Contradiction. $\square$

Using only bijective representations, we cannot harm the extensionality of a complete model. This follows directly from Corollary 19, as a complete model is also stable.

Corollary 25. *Let A be a model and ρ a representation such that $[\![A]\!]_\rho$ is complete and $[\![A]\!]_\rho \supsetneq [\![A]\!]$. Then there is no bijective representation π such that $[\![A]\!]_\pi = [\![A]\!]_\rho$.*

Corollary 26. *Isomorphism preserves interpretation-stability and interpretation-completeness. That is, let A and B be isomorphic models, then A is interpretation-stable iff B is, and A is interpretation-complete iff B is.*

Proposition 27. *A model with a finite extensionality (implementing finitely many functions) is complete.*

Complete models have interesting properties and are generally more convenient to work with. We elaborate on some of the properties concerning power comparison and isomorphism in Chapter 4.

3.2.1 Getting maximal interpretations

A natural question is how to choose a proper representation for getting a maximal interpretation. We should consider two cases, depending on whether the relevant model is complete or not.

For an interpretation-complete model A, we can get a maximal interpretation by one of the following means:

- Its original extensionality.
- Via any bijective representation.
- If A is closed under functional composition: via a representation ρ for which there exists a total injective function $f \in [\![A]\!]$, such that $\operatorname{Im} f = \operatorname{Im} \rho$ and $f^{-1} \in [\![A]\!]$.

For an incomplete model:

- There is no general known criterion.
- A bijective representation cannot help.

- If one finds a representation via which there is a maximal interpretation, then he can get additional maximal interpretations with the above techniques for complete models.

The special case of proper representations with respect to effectiveness is considered in Section 6.2.

Most of the claims above follow directly from previous sections. We add below the required additional theorem.

Theorem 28. *Let A be a model closed under functional composition, and let $\rho : D \to \mathrm{dom}\ A$ be some representation. Then $[\![A]\!]_\rho$ is isomorphic to $[\![A]\!]$ if there is a total injective function $h \in [\![A]\!]$ such that* $\mathrm{Im}\ h = \mathrm{Im}\ \rho$ *and* $h^{-1}\restriction_{\mathrm{Im}\ h} \in [\![A]\!]\restriction_{\mathrm{Im}\ h}$.

Proof. Define the function $\pi : D \to \mathrm{dom}\ A$ by $\pi := h^{-1} \circ \rho$. Since h is a total injection with $\mathrm{Im}\ h = \mathrm{Im}\ \rho$, it follows that π is a bijection. We shall prove that $[\![A]\!]_\rho = [\![A]\!]_\pi$, by showing that for every function $f \in [\![A]\!]$ there is a function $g \in [\![A]\!]$ such that $[\![f]\!]_\rho = [\![g]\!]_\pi$ and vice versa. For every function $f \in [\![A]\!]$ define $g := h^{-1} \circ f \circ h$. By the assumptions, $g \in [\![A]\!]$. Now, $[\![g]\!]_\pi = \pi^{-1} \circ g \circ \pi = \rho^{-1} \circ h \circ g \circ h^{-1} \circ \rho = \rho^{-1} \circ h \circ h^{-1} \circ f \circ h \circ h^{-1} \circ \rho = \rho^{-1} \circ f \circ \rho = [\![f]\!]_\rho$. By the assumption that $\mathrm{Im}\ h = \mathrm{Im}\ \rho$, we get that the transition above from $[\![g]\!]_\pi$ to $[\![f]\!]_\rho$ also preserves the exact partiality of functions. The opposite direction is analogous. □

3.3 Specific models

We turn now to investigate the influence of the domain interpretation on some well known computational models, as well as on hypercomputational models.

3.3.1 The recursive functions

The recursive functions are interpretation-complete. Their completeness is of special importance due to their rôle in the notion of effectiveness. This is elaborated on in Chapter 6.

When speaking of the recursive functions we generally mean the partial recursive functions. Yet, both the patrial and the total recursive functions are complete. As for the latter, its completeness is with respect to total functions. That is, its interpretations are also restricted to contain only total functions (see Definition 4).

The completeness is of both the unary recursive functions and of the functions of any arity.

In Section 3.3.2, it will be shown that the recursive functions are isomorphic to the functions computed by Turing machines. In Section 3.3.4, it is shown that the recursive functions are isomorphic to 3-counter machines, while being a maximal interpretation of the incomplete 2-counter machine model.

We start by showing that the unary recursive functions are complete. The same proof applies to both total and partial functions.

Theorem 29. *The unary recursive functions* ($\mathbb{UREC}$) *are interpretation-complete.*

Proof. Assume $[\![\mathbb{UREC}]\!]_\rho \supseteq \mathbb{UREC}$ for some injection $\rho : \mathbb{N} \to \mathbb{N}$, and let s be the successor function. There is, by assumption, a function $s' \in \mathbb{UREC}$, such that $[\![s']\!]_\rho = s$. That is, $\rho^{-1} \circ s' \circ \rho = s$. Hence, $s' \circ \rho = \rho \circ s$. Since $\rho(0)$ is some constant and $\rho(s(n)) = s'(\rho(n))$, it follows that ρ is derived by primitive-recursion from $s' \in \mathbb{UREC}$, thus $\rho \in \mathbb{UREC}$. Since ρ is a recursive injection, it follows that ρ^{-1} is partial recursive. By the closure of the recursive functions to functional composition, we have that $\rho^{-1} \circ f \circ \rho$ is recursive for every recursive function f. Therefore, $[\![\mathbb{UREC}]\!]_\rho = \mathbb{UREC}$, showing its completeness. □

The above result is easily extended to the set of all recursive functions.

Theorem 30. *The partial recursive functions* ($\mathbb{REC}$) *and the total recursive functions are interpretation-complete.*

Proof. Assume $[\![\mathbb{REC}]\!]_\rho \supseteq \mathbb{REC}$ for some injection $\rho : \mathbb{N} \to \mathbb{N}$. Since interpretations do not change the arity of the functions, it follows that $[\![\mathbb{UREC}]\!]_\rho \supseteq \mathbb{UREC}$. Hence, by the proof of Theorem 29, we have that $\rho \in \mathbb{REC}$. Therefore, by the closure of the recursive function to inverse and to functional composition, we have that $[\![\mathbb{REC}]\!]_\rho = \mathbb{REC}$, that is $\mathbb{REC}$ is complete. By the same token, we have also that the total recursive functions are complete. □

3.3.2 Turing machines

Turing machines are interpretation-complete. As with the recursive functions, this completeness is of special importance due to the rôle of Turing machines in the notion of effectiveness. This is elaborated on in Chapter 6.

Theorem 31. *Turing machines are interpretation-complete.*

Proof. The completeness of Turing machines follows directly from its isomorphism to the recursive functions. The proof of their isomorphism (Theorem 69) is postponed to Section 4.3, as it uses some power comparison results given in Chapter 4. $\square$

When seeking a maximal interpretation for Turing machines or for the recursive functions, the criteria of Section 3.2.1 may be extended:

Theorem 32. *An interpretation* $[\![\mathrm{TM}]\!]_\rho$ *of Turing machines via some injection* ρ *is maximal if* $|\mathrm{Im}\ \rho|$ *is infinite and there is a function* $h \in [\![\mathrm{TM}]\!]$ *such that* $\mathrm{Im}\ h = \mathrm{Im}\ \rho$.

Proof. By Theorem 28, $[\![\mathrm{TM}]\!]_\rho$ is maximal once h is a total injection, and $h^{-1}\restriction_{\mathrm{Im}\ h} \in [\![\mathrm{TM}]\!]\restriction_{\mathrm{Im}\ h}$. By running h alternately on the ordered domain elements (i.e. first step of h on the string "0", then the first step on the string "1", then the second step on "0", etc..) we may produce a total injective function h' out of h, such that $\mathrm{Im}\ h' = \mathrm{Im}\ h$. By the closure of Turing machines to minimalization, we may also produce the required function h'^{-1}. $\square$

Note that the function h in the above theorem need not be total nor injective.

3.3.3 Hypercomputational models

A computational model is generally said to be "hypercomputational" if it computes more than Turing machines or more than the recursive functions (see Definition 93 in Chapter 6). Due to the completeness of Turing machines and the recursive functions, such a model may indeed be regarded as more powerful. Power comparison is treated in detail in Chapters 4 and 5, and the issue of effective computation over arbitrary domains is treated in detail in Chapter 6.

Can we conclude from the interpretation-completeness of the recursive functions that every hypercomputational model is interpretation-complete? The answer is, in general, "no". However, if the hypercomputational model preserves the basic closure properties of the recursive functions, then the answer is "yes".

The following example is an incomplete hypercomputational model.

Example 33. *Let h be the (incomputable) halting function (using some standard indexing of Turing machines). Define:*

$$\begin{array}{rcl} \rho & := & \lambda n.2n \\ h_i & := & \lambda n.\begin{cases} 2^i h(n/2^i) & 2^i \textit{ divides } n \\ 0 & \textit{otherwise} \end{cases} \end{array}$$

Let A be a computational model with the extensionality $[\![A]\!] := \mathbb{REC} \cup \{h_i \mid i \in \mathbb{N}^+\}$. That is, $[\![A]\!]$ includes all the recursive functions and all functions h_i for $i \geq 1$.

We have that $[\![A]\!]_\rho = \mathbb{REC} \cup \{h_i \mid i \in \mathbb{N}\} \supsetneq [\![A]\!]$.

Yet, the completeness proof of the recursive functions (Theorem 30) may be extended to hypercomputational models, as long as they have the relevant closure properties. This also means that they are defined over denumerable domains, as with higher cardinalities there is no meaning to primitive recursion or minimalization:

Theorem 34. *Let A be a computational model over $\mathbb{N}$ computing all the recursive function and closed under functional composition, primitive recursion and minimalization. Then A is interpretation-complete.*

Proof. Since A is closed under minimalization, it follows that it is also closed under the inverse of total injective functions. Hence, the proofs of Theorems 29 and 30 apply as is to A. $\square$

A special case of such an interpretation-complete hypercomputational model is an oracle Turing machine.

Corollary 35. *An oracle Turing machine is interpretation-complete.*

Proof. The mechanism of Turing machines ensures the closure properties required in Theorem 34. $\square$

3.3.4 Counter machines

The model of two counter machines is very interesting. It is a standard model in the literature, though it is incomplete. It is strictly contained in the recursive functions, and has a maximal interpretation of all the recursive functions. in what follows, we take a deeper look into counter machines, checking n-counter machines, for every n.

When looking at the extensionality of counter machines we refer here only to total functions. Accordingly, when checking its interpretations and completeness we also refer only to total functions.

It turns out that the model of one-counter machines is complete, two-counter machines are incomplete, and three-or more-counter machines are complete.

We start by defining the mechanism of counter machines, and then look at the extensionality and completeness of n-counter machines for every n.

The mechanism of counter machines. An n-counter machine operates by a finite program with the following commands:

`A`$_i$`+`	increment the ith counter by 1.
`A`$_i$`-`	decrement the ith counter by 1 unless already zero.
`jump #`l	unconditionally jump to line no. l.
`If A`$_i$`=0 jump #`l	if the ith counter is zero jump to line no. l.
`halt`	end the run.

The machine performs the commands line by line, unless jumping due to a *jump* command.

An n-counter machine is considered to compute a function of any arity up to $n-1$, starting with the input in the first $n-1$ counters, and providing the output, if it halts, in the last counter. A one-counter machine computes unary functions, having the input and output in its unique counter.

Example 36. *The following 2-counter machine computes the function* $\lambda x.2x+1$*. We denote the counters by* A *and* B*.*

```
1. If A=0 jump #6
2. A-
3. B+
4. B+
5. jump #1
6. B+
7. halt
```

Three-or more-counter machines. It is known that three-counter machines are Turing-complete (e.g. [31]), meaning that their extensionality is the set of recursive functions of the relevant arity. Hence, by the interpretation-completeness of the recursive functions (Theorem 30), we have that three-counter machines are interpretation-complete. Obviously n-counter machines, where $n > 3$, are also complete, as they have the same extensionality as three-counter machines (up to the relevant arity).

Two-counter machines. It was shown independently by Janis Barzdins [2], Rich Schroeppel [39], and Frances Yao (see [39]) that two-counter machines cannot compute the function $\lambda x.2^x$. On the other hand, since two-counter machines can compute all the recursive functions via an injective representation (viz. $n \mapsto 2^n$; see, for example, [31]), it follows that two-counter machines are interpretation-incomplete.

One-counter machines. We show that one-counter machines (1CM) are interpretation-complete, by the following steps:

1. A function is 1CM-computable iff it is either of the following (Theorem 38):
   - "Ultimately an adder". That is, there is a threshold t and a constant addend a, such that for every $n > t$, $f(n) = n + a$.
   - "Ultimately periodic". That is, there is a threshold t and a constant period p, such that for every $n > t$, $f(n) = f(n+p)$. (Note that the "result" of $f(n)$ and $f(n+p)$ may be a divergence.)
2. An injection ρ, such that $[\![1\text{CM}]\!]_\rho \supseteq [\![1\text{CM}]\!]$ is "ultimately a multiplier" (Lemma 39). That is, there is a threshold t and a constant multiplier m, such that for every $n > t$ and natural number c, $\rho(n + c) = \rho(n) + m \times c$.
3. For every function $f \in 1\text{CM}$ and injection ρ as above, we have that if $[\![f]\!]_\rho$ exists then it is 1CM-computable (Theorem 40).

Definition 37 (Ultimately Basic Functions)**.**

- *A function is* ultimately an adder *if there is a threshold $t \in \mathbb{N}$ and a constant addend $a \in \mathbb{Z}$, such that for every $n > t$, $f(n) = n + a$.*
- *A function is* ultimately periodic *if there is a threshold $t \in \mathbb{N}$ and a constant period $p \in \mathbb{N}^+$, such that for every $n > t$, $f(n) = f(n + p)$. (Note that since the functions may be partial, the "value" of $f(n)$ and $f(n + p)$ may be a divergence.)*
- *A function is* ultimately a multiplier *if there is a threshold $t \in \mathbb{N}$ and a constant multiplier $m \in \mathbb{N}^+$, such that for every $n > t$ and constant $k \in \mathbb{N}$, $f(n + k) = f(n) + m \times k$.*

Note: A function that is ultimately an adder is also ultimately a multiplier with the multiplier $m = 1$.

Theorem 38 ([39])**.** *A function is* 1CM*-computable iff it is ultimately an adder or ultimately periodic.*

The theorem and its proof, with minor variations, are given in [39]. See the next section (Section 3.3.4.1) for a sketch of the proof.

Notes:

- A 1CM program with a threshold t may produce any output for an input smaller than t, including divergence.
- An ultimately periodic function has finitely many output values, possibly including divergence.

Lemma 39. *An injection ρ, such that $[\![\mathrm{1CM}]\!]_\rho \supseteq [\![\mathrm{1CM}]\!]$, is ultimately a multiplier.*

Proof. Assume an injection ρ, such that $[\![\mathrm{1CM}]\!]_\rho \supseteq [\![\mathrm{1CM}]\!]$. There is, by the assumption, a function $s' \in [\![\mathrm{1CM}]\!]$, such that $[\![s']\!]_\rho = s$, where $s \in [\![\mathrm{1CM}]\!]$ is the successor function. Accordingly, $s' \circ \rho = \rho \circ s$. Since the image of $\rho \circ s$ is an infinite set, it follows that so should be the image of s'. Hence, s' is not ultimately periodic, which makes it, by Theorem 38, ultimately an adder. Let $t_1 \in \mathbb{N}$ be the threshold of s', and $a \in \mathbb{Z}$ its addend. Since ρ is injective, it follows that there is a threshold $t \in \mathbb{N}$, such that for every $n > t$, $\rho(n) > t_1$. Hence, for every $n > t$, $s' \circ \rho(n) = \rho(n) + a$. Evaluating the equation $\rho \circ s = s' \circ \rho$, we get that for every $n > t$, $\rho(n+1) = \rho(n) + a$. Now, let $\rho(t) = c$, for some constant $c \in \mathbb{N}$, then $\rho(t+1) = c + a$, and by induction $\rho(t+k) = c + a \times k$. Since ρ is injective, it also follows that the multiplier (the addend a of s') must be positive. $\square$

Theorem 40. 1CM *are interpretation-complete.*

Proof. We will show that for every injection ρ such that $[\![\mathrm{1CM}]\!]_\rho \supseteq [\![\mathrm{1CM}]\!]$, we have that for every $f \in [\![\mathrm{1CM}]\!]$, if $[\![f]\!]_\rho$ is total then $[\![f]\!]_\rho \in \mathrm{1CM}$, which makes 1CM complete. By Lemma 39, such ρ must be ultimately a multiplier (with a threshold $t_1 \in \mathbb{N}$ and a multiplier $m \in \mathbb{N}^+$), and by Theorem 38, we should consider two cases for the function f:

Case i: The function f is ultimately an adder with a threshold t_2 and an addend a. Assume that $[\![f]\!]_\rho$ is total. That is, there is a function $\tilde{f} = [\![f]\!]_\rho$, such that for every $n \in \mathbb{N}$, $\rho \circ \tilde{f}(n) = f \circ \rho(n)$. Since ρ is injective, it follows that there is a threshold $t > \max(t_1, t_2)$, such that for every $n > t$, $\rho(n) > t_2$. Hence, for every $n > t$, $f \circ \rho(n) = \rho(n) + a = \rho(t) + m \times (n - t) + a$. Since $f \circ \rho$ is injective, it follows that so is $\rho \circ \tilde{f}$. Thus, choosing t to be big enough, we have that for all $n > t$, $\rho \circ \tilde{f}(n) = \rho(t) + m \times (\tilde{f}(n) - t)$. Evaluating the equation $\rho \circ \tilde{f} = f \circ \rho$, we get that for all $n > t$, $\rho(t) + m \times (\tilde{f}(n) - t) = \rho(t) +$

$m \times (n - t) + a$. Removing $\rho(t)$ from both sides provides that for all $n > t$, $m \times (\tilde{f}(n) - t) = m \times (n - t) + a$. As the left side divides by m so should the right side, making $a = m \times a'$ for some constant $a' \in \mathbb{N}$. Hence, for all $n > t$, $\tilde{f}(n) - t = n - t + a'$. Thus for all $n > t$, $\tilde{f}(n) = n + a'$, which makes $\tilde{f}$ ultimately an adder. By Theorem 38 it follows that $\tilde{f}$ is 1CM computable.

Case ii: The function f is ultimately periodic with threshold t_2 and period p. Assume that $[\![f]\!]_\rho$ is total. That is, there is a function $\tilde{f} = [\![f]\!]_\rho$, such that for every $n \in \mathbb{N}$, $\rho\circ\tilde{f}(n) = f\circ\rho(n)$. Since ρ is injective, it follows that there is a threshold $t > \max(t_1, t_2)$, such that for every $n > t$, $\rho(n) > t_2$. Hence, for every $n > t$, $\rho(\tilde{f}(n + p)) =$ *by the assumption* $= f(\rho(n+p)) = f(\rho(n)+m\times p) =$ *since f is ultimately periodic with a period p* $= f(\rho(n)) = \rho(\tilde{f}(n))$. We've got that $\rho(\tilde{f}(n + p)) = \rho(\tilde{f}(n))$, and since ρ is injective, it follows that $\tilde{f}(n + p) = \tilde{f}(n)$. Hence, $\tilde{f}$ is ultimately periodic (with a period p), and by Theorem 38 it is 1CM-computable.

□

3.3.4.1 The extensionality of one-counter machines

In [39] Rich Schroeppel provides a deep analysis of two-counter machines, some of which is relevant also to n-counter machines. In what follows, we give a sketch proof of Theorem 38, based on Schroeppel's analysis, providing the extensionality of one-counter machines.

Strong loops. These are the loops that remain in the program when ignoring the conditional (If ...) jumps. In Example 36, there is a strong loop by lines 1,2,3,4,5.

Observations.

1. There may be a single strong loop, if at all, that is reached from the beginning of the program without conditional jumps.
2. After a conditional jump, the output of the program is determined only by the current line number, regardless of the original input to the program. This is because there is only one counter, and the conditional jump is performed when the counter value is 0. Accordingly, a conditional jump may be evaluated to its result – either a divergence or a constant output k.

The two possible functions. In what follows we show that a one-counter machine without strong loops is ultimately an adder, while having a strong loop makes it ultimately periodic.

A program without strong loops. A program has finite many instructions, so getting a big enough number it will make no conditional jumps without the aid of strong loops. Hence, a program without strong loops behaves the same for all numbers bigger than a threshold t, adding or subtracting a constant, depending on the total number of `A+` and `A-` commands in the program. This is ultimately an adder by Definition 37.

A program with strong loops. Such a program will enter its strong loop for every big enough input. It will either diverge in the loop forever, or will get outside of it by a conditional jump. As a conditional jump is evaluated regardless of the original input, we get the periodic behavior.

Finding the values of the threshold and the multiplier. It should be noted that within a strong loop we may assume that all the conditional jumps and the `A-` commands precede all the `A+` commands. This is because we may perform the following changes within the strong loop:

```
A+                =   If A=0 --> A+ ...
If A=0 --> ...        A+

A+                =  nothing (delete both lines)
A-
```

The deletion of a sequence of an `A+` followed by an `A-` might only influence the result of a conditional jump that eventually jumps to the second command, but we assume that all conditional jumps are already evaluated.

A one-counter machine program takes the following form:

```
          ⋮                        Some code without a strong loop
  ↱       A-
          If A=0 ⟶
          ⋮                        A- and If commands
  ↻       A-
strong    If A=0 ⟶
 loop     •                        The threshold point
          A+
          ⋮                        A+ commands
  ∟       A+
```

Every big enough number will make it to the threshold point. Denote by *threshold* the number of `A-` commands until the threshold point, and denote by D the number of `A+` commands in the strong loop minus the number of `A-` commands in it. If $D \geq 0$ the run will diverge. Otherwise, the output depends only on $(input - threshold) \bmod (-D)$.

The other direction of Theorem 38. It is simple to see that every function that is ultimately periodic or ultimately an adder may be implemented by a one-counter machine. The reason is that up to a constant threshold the program may implement a "full output table", giving whatever required output to any input. This is illustrated by the following program with a threshold of 2.

```
If A=0 jump #l    // input was zero, jump to the required output.
A-
If A=0 jump #m    // input was one,      -”-.
A-
If A=0 jump #n    // input was two,      -”-.
A+
A+                // original input, if was bigger than 2, is restored
...               // continue with an adder or periodic behavior
```

3.3.5 Constant functions

For getting some understanding and examples of isomorphism, interpretation-stability and interpretation-completeness, we analyze all possible sets of constant functions over some infinite domain. Constant functions are also interesting in the sense that they are somewhere in the middle between ordinary sets of elements and sets of (arbitrary) functions.

We summarize the results by the following table, in which every entry represents an isomorphism class. That is, all sets described by the same table-line are isomorphic. The table is followed by the necessary proofs. Generally, two sets of constant functions are isomorphic iff they are of the same cardinality and so are the sets of the constant functions that they are missing (Theorem 41, below).

Isomorphism Class of Sets of Constant Functions	**Properties**	**Proof**
$n(\in \mathbb{N})$ functions	Complete models	Proposition 27
An infinite set of cardinality κ, lacking an infinite set (of constant functions) of cardinality η	Unstable	Corollary 42
All functions, except for $n(\in \mathbb{N}^+)$ functions	Stable but incomplete	Theorem 44 Theorem 45
All constant functions	Complete	Proposition 43

Theorem 41. *Two sets of constant functions, A and B, are isomorphic iff they are of the same cardinality, and the sets of the constant functions that they are missing are of the same cardinality.*

Proof. Since A and B are sets of constant functions, it follows that they are isomorphic iff there is a bijection $\pi : \text{dom } A \to \text{dom } B$, such that $\pi(\text{Im } A) = \text{Im } B$. By the assumptions that A and B are of the same cardinality, and that the functions that they are missing are of the same cardinality, we can define the bijections $\pi_1 : \text{Im } A \to \text{Im } B$ and $\pi_2 : \text{dom } A \setminus \text{Im } A \to \text{dom } B \setminus \text{Im } B$. Hence, the required bijection π can be defined as the union of π_1 and π_2. □

Corollary 42. *An infinite set of constant functions, lacking infinite many constant functions, is interpretation-unstable.*

Proposition 43. *The set of all constant functions over some domain is interpretation-complete.*

Proof. An interpretation of a constant function is always a constant function. □

Theorem 44. *The set of all constant functions over some domain, except for finite-many functions, is interpretation-stable.*

Proof. The proof is by induction on the number of missing functions. For the base case, let M be the set of all constant functions over some domain D, let $c \in D$ be some element of D, and let $f_c \in M$ be the constant function whose value is c. Assume, by contradiction, that there is a bijection $\pi : D \to D$, such that $\pi(M \setminus \{f_c\}) = M$. By the assumption, $M \setminus \{f_c\}$ and M are isomorphic, therefore, by Corollary 26, $M \setminus \{f_c\}$ is stable iff M is stable. Hence, by Proposition 43, $M \setminus \{f_c\}$ is stable, which contradicts the assumption that it is isomorphic to a proper superset of itself. For the induction step, the same argument is valid, showing the isomorphism to a proper superset of itself, which is already known to be stable. □

Theorem 45. *An infinite proper subset of the set of all constant functions over some domain is interpretation-incomplete.*

Proof. Let M be the set of all constant functions over domain D, $A \subsetneq M$ an infinite proper subset of M, and B a subset of M, which is a proper superset of A ($A \subsetneq B \subseteq M$). Let $\rho : \text{Im } A \to \text{Im } B$ be an injection from the image of A to the image of B. Then, for every function $f \in [\![B]\!]$ there is a function $f' \in [\![A]\!]$, such that for every $x \in \text{Im } B$, we have that $f' \circ \rho = \rho \circ f$. Hence A is incomplete. □

3.4 Interpretation-completeness and time complexity classes

Our handling of computational models relates only to the models' extensionality. That is, we consider the set of functions that the models compute, regardless of their internal mechanism or the computation complexity. Nevertheless, one may ask about the interpretation-completeness of any set of functions, among which are the sets of functions that can be computed by a certain model in a certain complexity. For example, one may wonder whether the set of functions that are computed by Turing machines in a polynomial time complexity ($\mathcal{P}$) is interpretation-complete.

In this section we check the interpretation-completeness of three standard complexity classes – the constant, linear, and polynomial time-complexity classes with respect to Turing machines. We show that all three are interpretation-incomplete. Their incompleteness is quite straightforward, shown by choosing a representation that significantly expands the size of the original string (in the linear and polynomial cases).

This straightforward incompleteness leads us to suggest a more adequate property for complexity classes, denoted "semi-completeness", which limits the size-expansion allowed by the representations.

Functions vs. languages. When looking for interpretation-completeness we are interested in functions with images in the whole domain ($f : \Sigma^* \to \Sigma^*$), as opposed to decision (Boolean) functions. The latter are easily shown to be incomplete, by adding some Boolean result in the representation, e.g. as the least significant bit. When considering the set of functions with images in the whole domain, the simple data-hiding above does not work, since we cannot generate the additional data for the output of the function. All the same, for interpretation-stability, it might also be interesting to look at decision functions.

Constant time complexity. When considering functions of constant time complexity, the exact definition of the computational model is of significant importance (Turing machines with one tape vs. several tapes, etc..). In any case, this class of functions is easily shown to be incomplete. We shall consider here Turing machines with a single tape, operating over $\{a, b\}^*$.

A function is of constant complexity if there is a Turing machine that implements it, running no more than K steps for all inputs. Formally,

Definition 46. *A function $f : \{a, b\}^* \to \{a, b\}^*$ is of* constant time complexity *if there is a Turing machine T and a constant $K \in \mathbb{N}$, such that for every input string $s \in \{a, b\}^*$, T halts after no more than K steps with the output string $f(s)$.*

We show the incompleteness of this class by choosing a representation ρ that concatenates the result of a non-constant complexity function to the original string, as its rightmost bit. This extra function is the parity function with respect to the length of the input string.

Theorem 47. *The set of constant time complexity functions, $\mathcal{C}$, is interpretation-incomplete.*

Proof. The claim is that there is an injection $\rho : \{a, b\}^* \to \{a, b\}^*$, such that $[\![\mathcal{C}]\!]_\rho \supsetneq \mathcal{C}$.

The representation ρ would be the concatenation of the letter "a" to the string if the length of the original string is even, and concatenating the letter "b" otherwise. Denoting concatenation by $\cdot$, ρ is formally defined as follows:

$$\rho(s) \quad := \quad \begin{cases} s \cdot \text{“}a\text{”} & |s| \text{ is even} \\ s \cdot \text{“}b\text{”} & \text{otherwise} \end{cases}$$

Specifically, we show that the function ρ itself, which is not of constant time-complexity, belongs to $[\![\mathcal{C}]\!]_\rho$.

We should show three things – i) $\rho \notin \mathcal{C}$; ii) $\rho \in [\![\mathcal{C}]\!]_\rho$ and iii) for every $f \in \mathcal{C}$ there is $f' \in \mathcal{C}$, s.t. $f' \circ \rho(s) = \rho \circ f(s)$ for every $s \in \{a, b\}^*$.

i) The function ρ is not of constant time complexity as it should go over the whole input string, making it an $O(n)$ function.

ii) The function ρ is easily implemented in constant time over a string of the form $\rho(s)$: if the last bit of $\rho(s)$ is "a" then $\rho(\rho(s)) := \rho(s)\cdot$"$b$", otherwise $\rho(\rho(s)) := \rho(s)\cdot$ "a".

iii) Let f be implemented by a Turing machine T with time complexity K. We can conduct a constant-complexity machine T', such that T' will make the following steps: 1. check the last bit (the parity of the original string's length) and remove it; 2. perform the operation of T over the original string; and 3. add the appropriate last bit (the parity of the new string's length), depending on whether the operations of T added or deleted an even or an odd number of bits. For having the above parity information, the machine T' should have two copies of T, "jumping" between the copies once a bit is added to or deleted from the tape.

□

Linear time complexity. This class is also interpretation incomplete. This is shown by using a representation that significantly enlarges the original string.

Theorem 48. *The class of linear time complexity functions, $\mathcal{L}$, is interpretation-incomplete.*

Proof. The claim is that there is an injection $\rho : \{a, b\}^* \to \{a, b\}^*$, such that $[\![\mathcal{L}]\!]_\rho \supsetneq \mathcal{L}$.

The representation ρ would duplicate the original string, s, $|s|$ times (See Figure 3.4). It cannot be a straightforward duplication, but should add some separation between the duplications. See Note 49 below for a technical implementation in linear time.

We should show two things – i) there is a function g, such that $g \in [\![\mathcal{L}]\!]_\rho$ and $g \notin \mathcal{L}$; and ii) $\mathcal{L} \subseteq [\![\mathcal{L}]\!]_\rho$.

i) Let g be an $O(n^2)$ time complexity function that has an output of linear size in the size of the input. Now, over the large string, $\rho(s)$, we can apply a function $g' \in \mathcal{L}$ that activates g on the first copy of s

ρ squares the string's length.
g' is polynomial w.r.t. s,
but linear w.r.t. $\rho(s)$

$$\begin{tikzcd}
s \arrow[r, "g \notin L"] \arrow[d, "\rho"'] & t \arrow[d, "\rho"] \\
sssssssss \arrow[r, "g' \in L"'] & tttttttt
\end{tikzcd}$$

Figure 3.4: The class of linear time complexity functions is interpretation-incomplete

within $\rho(s)$. The time complexity of g' is linear relative to $\rho(s)$. For getting the required output, $\rho \circ g$, we still have to duplicate the output string, $g(s)$, $|g(s)|$ times. Since the output growth of g is limited by a constant k (that is, $|g(s)| < k|s|$), it follows that the duplication of $g(s)$ takes $|g(s)|^2 < (k|s|)^2 = k^2|s|^2$ steps, which is linear in $|\rho(s)| = |s|^2$. Hence, $g = \rho^{-1} \circ \rho \circ g \circ \rho^{-1} \circ \rho = \rho^{-1} g' \circ \rho$, getting that $g \in [\![\mathcal{L}]\!]_\rho$.

ii) Since every function of linear time complexity can only linearly enlarge the output, it follows from the arguments of the previous step that $\mathcal{L} \subseteq [\![\mathcal{L}]\!]_\rho$.

$\square$

Note 49. *The duplication, given in the above proof, cannot be a straightforward one, but should add some separation between the duplications. This may easily be done in linear time. For instance, with the $\{a, b\}$ alphabet we may repeat each character twice, and mark the separations by "ab". For example, the string abb would be represented by aabbbb*__ab__*aabbbb*__ab__*aabbbb.*

Polynomial time complexity. Somewhat surprisingly, this class is also interpretation-incomplete. We show it using the same approach as used for the linear time complexity case, just that here the representation ρ duplicates the original string $|s|^{log|s|}$ times.

Theorem 50. *The class of polynomial time complexity functions, $\mathcal{P}$, is interpretation-incomplete.*

Proof. The claim is that there is an injection $\rho : \{a, b\}^* \rightarrow \{a, b\}^*$, such that $[\![\mathcal{P}]\!]_\rho \supsetneq \mathcal{P}$.

The representation ρ would duplicate the original string, s, $|s|^{\log|s|}$ times (the technical implementation is analogous to the linear case, described in Note 49).

We should show two things – i) there is a function g, such that $g \in [\![\mathcal{P}]\!]_\rho$ and $g \notin \mathcal{P}$; and ii) $\mathcal{P} \subseteq [\![\mathcal{P}]\!]_\rho$.

i) Let g be an $O(n^{logn})$ time complexity function that has an output of polynomial size in the size of the input. Now, over the large string, $\rho(s)$, we can apply a function $g' \in \mathcal{P}$ that activates g on the first copy of s within $\rho(s)$. The time complexity of g' is polynomial relative to $\rho(s)$. For getting the required output, $\rho \circ g$, we still have to duplicate the output string, $g(s)$, $|g(s)|^{\log|g(s)|}$ times. Since the output growth of g is limited by a power of k (that is, $|g(s)| < |s|^k$) the duplication of $g(s)$ takes $|g(s)|^{\log|g(s)|} < (|s|^k)^{(\log|s|^k)} = |s|^{(k \times \log|s|^k)} = |s|^{(k \times k \times \log|s|)} = (|s|^{\log|s|})^{k^2}$, which is polynomial in $|\rho(s)| = |s|^{\log|s|}$. Hence, $g = \rho^{-1} \circ \rho \circ g \circ \rho^{-1} \circ \rho = \rho^{-1} g' \circ \rho$, getting that $g \in [\![\mathcal{L}]\!]_\rho$.

ii) Since every function of polynomial time complexity can only polynomially enlarge the output, it follows from the arguments of the previous step that $\mathcal{P} \subseteq [\![\mathcal{P}]\!]_\rho$.

□

Note: Could one use in the above proof a representation ρ that duplicates the original string, s, $2^{|s|}$ times? The answer is NO, since the simulating functions would need to exceed polynomial data growth: Let $f(s)$ be the polynomial function that given a string s duplicates it $|s|$ times. The simulating function f' should duplicate the original output $f(s)$ by the exponent of its length. That is $2^{|f(s)|} = 2^{|s| \times |s|}$ times. However, if f' is polynomial with a power of k, then the output length should be less than a power k of the input's length. That is, $|f'(\rho(s))| < |\rho(s)|^k$. But, $|f'(\rho(s))| = 2^{|s| \times |s|} > 2^{k \times |s|} = (2^{|s|})^k = |\rho(s)|^k$.

Semi-completeness. The straightforward incompleteness of the complexity classes suggests that some other completeness definitions should be considered when handling complexity classes. A possibly natural definition is to limit the data growth of the representation to be the same as that of the complexity class. For example, limiting the data growth to be polynomial when considering the class of polynomial time complexity functions. Note that this limitation applies only to the data growth and not to the computation time (furthermore, the representation need not be Turing-computable).

Definition 51 (Semi-Completeness)**.** *A class of functions with time complexity $\mathcal{T}$ is* semi-interpretation-complete *if it is interpretation-complete with respect to representations that have a data growth bounded by $\mathcal{T}$.*

The class of constant time complexity functions can be shown to be interpretation-unstable, thus it is also not semi-complete. As for the linear and polynomial time complexity classes, we currently do not have an answer.

Chapter 4

Comparing Computational Power

It is standard in the literature to compare the power of computational models. However, neglecting the possibility of interpretation-incomplete models, it is common to say that model B is strictly stronger than model A when it computes strictly more functions. This might allow one to show that some computational models are *strictly* stronger than themselves (see Chapter 2)!

We define a comparison notion over arbitrary domains based on model interpretations. With this notion, model B is strictly stronger than A if B has an interpretation that contains A, whereas A cannot contain B under *any* interpretation.

We start, in Section 4.1, by providing the mathematical definition of the comparison notion. We give a basic definition, allowing all interpretations, and two additional definitions with some restrictions on the allowed interpretations. In Sections 4.1.1 and 4.1.2 we extend the notion from "regular" deterministic models to some other types of computational models.

In Section 4.2 we provide several results with respect to power comparison, isomorphism and completeness. In Section 4.3 we use the comparison notion to compare between some standard models, as Turing machines, stack machines, etc.. We conclude the chapter, in Section 4.4, by providing a definition of the comparison notion in the terminology of category theory.

In the next chapter, Chapter 5, we provide some philosophical justifications for the comparison notion.

4.1 The comparison notions

Since we are only interested here in the extensional quality of a computational model (the set of functions or relations that it computes), not complexity-based comparison or step-by-step simulation, we use model interpretations as the basis for comparison.

We generally say that model B is at least as powerful as model A if it can compute whatever A can. When both models operate over the same domain it simply means containment: B is at least as powerful as A if $[\![B]\!] \supseteq [\![A]\!]$. However, when the models operate over different domains we ought to interpret one model over the domain of the other. Hence, the general comparison notion would say that B is at least as powerful as A if it has an interpretation that contains A.

As one textbook states [44, p. 30]:

> Computability relative to a coding is the basic concept in comparing the power of computation models.... The computational power of the model is represented by the extension of the set of all functions computable according to the model. Thus, we can compare the power of computation models using the concept 'incorporation relative to some suitable coding'.

We provide three notions of comparison, depending on the allowed representations. The basic, most permissive, comparison notion allows any injection, while the firmest notion allows only bijections. In between, we define a notion that allows injections for which the "at least as powerful" model can fix their image.

In this chapter we handle the mathematical aspects of the comparison notions, while in Chapter 5 we provide some philosophical justifications for them.

Definition 52 (Power Comparison)**.**

- *Model B is* at least as powerful *as model A, denoted $B \gtrsim A$, if it has an interpretation that contains the extensionality of A. That is, $B \gtrsim A$ iff exists an injection $\rho : \mathrm{dom}\ A \to \mathrm{dom}\ B$, such that $[\![B]\!]_\rho \supseteq [\![A]\!]$.*

- *Model B is* decently at least as powerful *as model A, denoted $B \succsim A$, if it has an interpretation that contains the extensionality of A via a representation for which it can fix its image. That is, $B \succsim A$ iff exists an injection $\rho : \mathrm{dom}\ A \to \mathrm{dom}\ B$, such that $[\![B]\!]_\rho \supseteq [\![A]\!]$ and there is a total function $g \in [\![B]\!]$ for which $\mathrm{Im}\ g = \mathrm{Im}\ \rho$ and such that for every $y \in \mathrm{Im}\ \rho$ we have that $g(y) = y$.*

- *Model B is* bijectively at least as powerful *as model A, denoted $B \gtrapprox A$, if it has a bijective interpretation that contains the extensionality of A. That is, $B \gtrapprox A$ iff exists a bijection $\pi : \text{dom } A \to \text{dom } B$, such that $[\![B]\!]_\pi \supseteq [\![A]\!]$.*

Proposition 53. *The computational power relations $\gtrsim, \succsim, \gtrapprox$ are quasi-orders.*

Transitivity of the relations follows from the fact that the composition of injections is an injection.

Obviously, the two latter comparison notions imply the former one. For the third notion to imply the second one, it is sufficient that the at least as powerful model has some surjective function, even the identity function. Additionally, when assuming a little more about the representation and the model, the second notion also implies the third one (see Theorem 28).

Note 54. *When comparing the computational power of models, it should be noted that one function cannot be "better" than another; only a model can be better than another. There is only an equivalence relation between functions, where two partial functions, f and g, over the same domain D are deemed* equal, *denoted $f = g$, if they are defined for exactly the same elements of the domain ($f(x)$ diverges iff $g(x)$ diverges for all $x \in D$) and have the same value whenever they are defined ($f(x) = g(x)$, for all $x \in D$). For example, a total function f is not better than nor equal to a partial function g that has the same values as f when it converges. This is clearly demonstrated by taking g to be a "halting function", which returns* 1 *when the input encodes a halting machine and diverges otherwise.*

Our comparison notions go along with some standard approaches, for instance in [17, p. 24], [38, p. 27] and [44, p. 30]. The latter uses the term "incorporate" for our notion of "at least as powerful":

> 1. $F(SAL(X_1))$ is *incorporated* in $F(SAL(X_2))$ relative to coding c, if for every function f in $F(SAL(X_1))$ there is a function g in $F(SAL(X_2))$ which incorporates f relative to c.
> 2. The computation models $SAL(X_1)$ and $SAL(X_2)$ are *equivalent* if there exist codings c_1 and c_2 such that
>   (a) $F(SAL(X_1))$ incorporates $F(SAL(X_2))$ via coding c_1
>   (b) $F(SAL(X_2))$ incorporates $F(SAL(X_1))$ via coding c_2

where $F(SAL(X_i))$ denotes the set of functions computed by $SAL(X_i)$ [44, p. 27].

Our decent-comparison notion follows Rice's [37, p. 298] and Rabin's definition of a computable group [35, p. 343]: "DEFINITION 3. An *indexing* of a set S is a one to one mapping $i : S \to I$ such that $i(S)$ is a recursive subset of I."

Example 55. *Turing machines are at least as powerful as the recursive functions via a unary representation of the natural numbers. See, for example, [25, p. 147]. Indeed, it is so by all three notions (see Theorem 69).*

Ben-Amram [3, p. 127] suggests a comparison notion resembling our notion, however he requires the representation to be intuitively natural.

One may wonder why we do not require the representation to be Turing-computable. A detailed answer to that is given in Chapters 5 and 6. The main points are:

- When comparing models in the scope of defining effectiveness, the requirement of a Turing-computable representation leads to a circular definition.
- One may wish to compare sub-recursive models or hypercomputational models, for which Turing-computability is not necessarily the proper constraint.

Power equivalence. The power equivalence relation between models follows directly from the power comparison notion. Models A and B are of equivalent power if $A \gtrsim B$ and $B \gtrsim A$. Below is the definition for the three comparison notions:

Definition 56 (Power Equivalence)**.**

- *Models A and B are* power equivalent*, denoted $A \sim B$, if $A \gtrsim B$ and $B \gtrsim A$.*
- *Models A and B are* decently power equivalent*, denoted $A \simeq B$, if $A \succsim B$ and $B \succsim A$.*
- *Models A and B are* bijectively power equivalent*, denoted $A \approx B$, if $A \gtrapprox B$ and $B \gtrapprox A$.*

Example 57. *The (untyped) λ-calculus is power equivalent to the recursive functions, via Church numerals, on the one hand, and via Gödelization, on* *the other. However, it is not interpretation-complete, as it cannot even compute the identity function over an arbitrary lambda-term. Hence, it is not bijectively-power equivalent to the recursive functions (otherwise contradicting the completeness of the recursive functions).*

Strictly stronger. We generally think of model B as strictly stronger than model A if it can compute more. However, because of the sensitivity of models to the domain interpretation, proper containment does not imply more computational power. That is, $[\![B]\!] \supsetneq [\![A]\!] \not\Rightarrow B \gnsim A$. This is so for all three comparison notions above. A detailed handling of the sensitivity issue is given in Chapter 2.

The proper definition of model B being strictly stronger than model A is that $B \gtrsim A$ while $A \not\gtrsim B$.

Definition 58 (Strictly Stronger)**.**

- *Model B is* stronger *than model A, denoted $B \gnsim A$, if $B \gtrsim A$ while $A \not\gtrsim B$.*
- *Model B is* decently stronger *than model A, denoted $B \succnsim A$, if $B \succsim A$ while $A \not\succsim B$.*
- *Model B is* bijectively stronger *than model A, denoted $B \gnapprox A$, if $B \gtrapprox A$ while $A \not\gtrapprox B$.*

Note that for model B to be stronger than model A there should be no injection ρ via which $A \gtrsim_\rho B$. In contradistinction to the "as powerful" case, some model B may be bijectively stronger than a model A, but not stronger than A. However, if A is interpretation-complete, we do have that $B \gnapprox A$ implies $B \gnsim A$ (Theorem 67).

Example 59. *Real recursive functions [33], operating over $\mathbb{R}^2$, are decently stronger than Turing machines. The comparison is done via an injective representation $\psi : \{0,1\}^* \to \mathbb{R}^2$, defined by $[\![x]\!]_\psi := (0, [\![x]\!]_\rho)$, where $[\![x]\!]_\rho$ is the standard binary interpretation over the natural numbers [33, p. 849]. Since the model computes the floor function ($\lambda x.\, \lfloor x \rfloor$) [33, p. 843], it follows that it also has a function which fixes the representation image. On the other hand, the real recursive functions are obviously not bijectively stronger than the recursive functions, as their domain is of a higher cardinality.*

4.1.1 Non-deterministic models

The extension of most of the definitions and theorems given so far (in this chapter and the previous ones) to non-deterministic models is quite straightforward. There are, however, some specific issues concerning the power comparison of non-deterministic models, which will be discussed below.

We begin by defining what we mean by a non-deterministic computational model and its extensionality:

Definition 60 (Non-deterministic Computational Model)**.** *A* non-deterministic computational model *A over domain D is any object associated with a set of (non-unary) relations over $D \cup \bot$. This set of relations is called the* extensionality *of the non-deterministic computational model, denoted $[\![A]\!]$.*

Note that we use the special symbol $\bot$ in the above definition for denoting a non-halting computation, while we did not need it in the definition of a deterministic model (Definition 1). The reason is that a non-deterministic computation might sometimes diverge on a domain element e and sometimes converge to some value v. In such a case, the value of the computation will be both $\bot$ and v, denoted by $\langle e, \bot\rangle, \langle e, v\rangle$ in the (multivalued) function's description. On the other hand, the divergence of a deterministic function on a domain element e, may be simply denoted by not having a tuple with e in the function's description.

As a result of using $\bot$ to denote divergence, we may assume that all functions have at least one value for every domain element.

When investigating the influence of the domain interpretation, we are concerned with the containment relation between the different interpretations of a computational model. Hence, the definitions given in this chapter and the previous ones apply to both deterministic and non-deterministic models, as in both types of models the interpretations are sets of relations.

When we compare the power of computational models, two functions are considered equal if and only if they are described by the same relation, while a model is as powerful as another if it contains all the functions of the former (see Note 54). This is also the case when we compare between non-deterministic models. There might be cases in which a different approach is required, assuming a different equality notion between functions. For example, two non-deterministic functions might be considered equal if they *may* always produce the same value. That is, a function that either diverges or converges with the value v is considered equal to a function that always converges with the value v. In such cases, the comparison notions should be adjusted to take into account the special equality notion between functions.

By this comparison approach, a non-deterministic model B may be as powerful as a deterministic model A (if it deterministically computes all the functions of A, in addition to its non-deterministic computations), while the opposite is impossible (when B actually has some non-deterministic computations).

4.1.2 Universal machines and interactive machines

We extend the comparison notions above to universal machines and interactive machines.

Universal machines. Computation is often performed via universal machines, also referred to as "interpreters". That is, a single machine (or computer program) gets as input both the machine to interpret and the latter's input. One may wonder how to compare the computational power of universal machines. From our point of view, computational power is extensionality, meaning the set of computed functions. Hence, when comparing universal machines we compare the sets of functions that they interpret. Accordingly, it is exactly like comparing computational models.

Interactive machines. An interactive machine interacts with the environment. It is often considered to run forever, getting an input stream and producing an output stream. A common example for such a machine/program is the modern operating system. More details on interactive machines can be found, for example, in [5, 23].

How should one compare the power of computational models with interactive machines? Since we consider computational power to be extensionality, we should define what is the extensionality of such models. Once we agree on their extensionality (e.g. as a set of interactive functions, which are sets of streams), we can interpret them over different domains as we do with ordinary models (see Definition 4). The basic comparison notion as well as the bijective comparison notion (Definition 52) can be applied to these models, as they are based on model interpretation and set-containment. The decent-comparison notion (Definition 52) needs some adaptation to the case of interactive machines for conveying the fact that the at least as powerful model recognizes the image of the representation.

4.2 Power comparison, isomorphism, and completeness

The general approach for showing that model B is stronger than model A is tedious – we should negate the possibility that $A \gtrsim_\rho B$ for all injections ρ. The situation is much simpler with interpretation-complete models, for which proper containment does imply more power:

Theorem 61. *For an interpretation-complete model A and some model B, we have that $B \gnsim A$ iff there exists an injection ρ such that $[\![B]\!]_\rho \supsetneq [\![A]\!]$.*

Proof. If $[\![B]\!]_\rho \supsetneq [\![A]\!]$ then, by definition, $B \gtrsim A$. From the completeness of A it follows that $A \not\gtrsim B$. As for the other direction, by definition there exists an injection ρ such that $[\![B]\!]_\rho \supseteq [\![A]\!]$ for some ρ. Were $[\![B]\!]_\rho \supseteq [\![A]\!]$, it would make $A \sim B$, contradicting the assumption. $\square$

We continue below with some lemmata and theorems on the relations between the power comparison notions, isomorphism, stability and completeness.

Isomorphism and bijective power equivalence. These two notions are very similar, though not the same. However, when sticking to interpretation-stable models they do coincide. We summarize their relations in the following theorem.

Theorem 62.

1. *Isomorphism implies bijective power equivalence.*
2. *Bijective power equivalence does not imply isomorphism.*
3. *For interpretation-stable models, bijective power equivalence implies isomorphism. That is, if a model A is interpretation-stable, then for every model B, B is isomorphic to A if and only if it is bijectively power equivalent to A.*

Proof.

1. Obvious from the definitions (Definitions 7 and 56).
2. We will prove that bijective power equivalence does not imply isomorphism, by using some variants of the triangular machine model (described in Section 2.2). Looking at Figure 2.4, we have that the extensionality of triangular machines is $[\![\mathrm{Tr}]\!] = f_{i,j}$ $(i, j \in \mathbb{N})$, where

$f_{i,j}(s)$ changes the input string s into the ith string in the jth subsequent row to s's row, wrapping around for overly large i. Define two submodels, A and B, of triangular machines, having the following extensionality: $[\![A]\!] := \{f_{i,j} \mid i \geq 1 \text{ and } j \geq 0\}$ and $[\![B]\!] := [\![A]\!] \cup \{f_{0,0}\}$. That is, A lacks all the functions of $[\![\mathrm{Tr}]\!]$ that changes the input string into the first (0th) string of some row, while B adds to A only the function that changes the input string into the first string of the same row. (B still lacks all the functions that change the input string into the first string of a different row.) We will show that A and B are bijectively power equivalent, but not isomorphic. Obviously, $B \gtrapprox A$, as $[\![B]\!] \supseteq [\![A]\!]$. We also have that $A \gtrapprox B$ via the bijection τ, defined in the end of Section 2.2, since $[\![A]\!]_\tau = [\![\mathrm{Tr}]\!]$. On the other hand, it turns out that if some interpretation $[\![A]\!]_\pi$ of A via a bijection π contains $[\![B]\!]$, then it must strictly contain it (actually, it implies that $[\![A]\!]_\pi = [\![\mathrm{Tr}]\!]$). We show it in two steps:

- i: Assume that $[\![A]\!]_\pi \supseteq [\![B]\!]$ via a bijection π, such that π moves some string e from its original row. By assumption, there is a function $f_{p,q} \in [\![A]\!]$ such that $[\![f_{p,q}]\!]_\pi = f_{0,0}$. Since no two rows are of the same length, it follows that either eth row is shorter than the row of $\pi(e)$, or the other way around. In the first case, there is a string d in a row different than eth row, such that $\pi(d)$ is in the same row as $\pi(e)$. This leads to a contradiction, as $f_{0,0}(d) \neq f_{0,0}(e)$ (since they are on different rows), while $[\![f_{p,q}]\!]_\pi(d) = \pi^{-1} \circ f_{p,q} \circ \pi(d) = \pi^{-1} \circ f_{p,q} \circ \pi(e) = [\![f_{p,q}]\!]_\pi(e)$. Analogously, the second case, in which eth row is longer than the row of $\pi(e)$, leads to a contradiction.
- ii: By the previous step, if $[\![A]\!]_\pi \supseteq [\![B]\!]$ via a bijection π, then π must leave every string in its original row. Thus, there is a function $f_{p,0} \in [\![A]\!]$ such that $[\![f_{p,0}]\!]_\pi = f_{0,0}$. Since $\pi^{-1} \circ f_{p,0} \circ \pi = f_{0,0}$ and π does not move strings between rows, it follows that $\pi^{-1} \circ f_{p,0} = f_{0,0}$. Hence, π moves the 0th string of each row to the pth string of the same row. Accordingly, $\pi^{-1} \circ f_{p,i} \circ \pi = f_{0,i}$ for every $i \in \mathbb{N}$. Thus, $[\![A]\!]_\pi = [\![\mathrm{Tr}]\!]$, which strictly contains $[\![B]\!]$.

3. Let A be a stable model and assume $A \gtrapprox_\pi B \gtrapprox_\tau A$ for model B and bijections π, τ. By definition, $[\![A]\!]_\pi \supseteq [\![B]\!]$ and $[\![B]\!]_\tau \supseteq [\![A]\!]$. Were $[\![A]\!]_\pi \supsetneq [\![B]\!]$, then, by Lemma 17, $[\![[\![A]\!]_\pi]\!]_\tau \supsetneq [\![B]\!]_\tau \supseteq [\![A]\!]$, which would contradict the stability of A. Thus, $[\![B]\!] = [\![A]\!]_\pi$, which makes it A and B isomorphic.

□

Isomorphism preserves stability and completeness (Corollary 26). This is also the case, by the above theorem, with bijective power equivalence.

Corollary 63. *Bijective power equivalence preserves interpretation-stability and interpretation-completeness. That is, let A and B be bijectively power equivalent models, then A is interpretation-stable iff B is, and A is interpretation-complete iff B is.*

The formulation of the previous theorem can be strengthened for complete models:

Lemma 64. *If model A is interpretation-complete and $A \gtrsim B \gtrapprox A$ for some model B, then A and B are isomorphic.*

Proof. Suppose A is complete, and $A \gtrsim_\rho B \gtrapprox_\pi A$, for model B, injection ρ and bijection π. It follows that $[\![B]\!]_\pi = [\![A']\!] \supseteq [\![A]\!]$ for some model A'. Thus, $A \gtrsim_\rho B \gtrapprox_\pi A' \supseteq [\![A]\!]$. Therefore, from the completeness of A, it follows that $[\![A']\!] = [\![A]\!]$. Hence, $A \approx B$, and by the previous theorem, A and B are isomorphic □

Next, we provide some means to show that a model is stronger than another.

Strictly stronger. Interpretation-completeness also helps with showing that some model is strictly stronger than another.

Theorem 65. *If $A \lessapprox B$ and $[\![B]\!] \subsetneq [\![C]\!]$, for models A, B, and C, then there is a model D, such that $C \cong D$ and $[\![D]\!] \supsetneq [\![A]\!]$.*

Proof. Suppose $A \lessapprox_\pi B$ for bijection π. Then, $\pi([\![A]\!]) \subseteq [\![B]\!] \subsetneq [\![C]\!]$. Let D be a model with the extensionality $[\![D]\!] = \pi^{-1}([\![C]\!])$, from which we have that $C \cong D$. Since $\pi([\![A]\!]) \subsetneq [\![C]\!]$, it follows from Lemma 17 that $[\![A]\!] = \pi^{-1}(\pi([\![A]\!])) \subsetneq \pi^{-1}([\![C]\!]) = [\![D]\!]$. □

Theorem 66. *If model A is interpretation-complete, $A \approx B$ and $[\![B]\!] \subsetneq [\![C]\!]$, for models B and C, then $C \gnsim A$.*

Proof. If $A \approx B$ and $[\![B]\!] \subsetneq [\![C]\!]$, then $C \gtrsim B \gtrsim A$. By Corr. 63, we have that B. Thus, $B \not\gtrsim C$ and also $A \not\gtrsim C$. Hence, $C \gnsim A$. □

Theorem 67. *If model A is interpretation-complete, then $B \gnapprox A$ implies $B \gnsim A$*

Proof. Since $B \gnsim A$, it follows that $B \gtrsim A$ while $A \not\gtrsim B$. Thus, we also have that $B \gtrsim A$. Assume by contradiction that $B \not\gtrsim A$. Since, $B \gtrsim A$, it follows that $A \gtrsim B$. Hence, by Lemma 64, we have that A and B are isomorphic, contradicting the assumption that $B \gnsim A$. □

4.3 Comparison of some standard models

We turn now to compare some specific computational models.

As discussed in the beginning of this chapter, the proper containment between the recursive functions and the primitive recursive functions does not imply that the former are strictly more powerful than the latter. Nonetheless, we show hereby that, indeed, the recursive functions are strictly more powerful, even by the general comparison notion, allowing all possible interpretations of the primitive recursive functions.

Theorem 68. *The primitive recursive functions, Prim, are strictly weaker than the recursive functions.*

Proof. Clearly, $\mathbb{REC} \gtrsim_\iota Prim$. So, assume, on the contrary, that $Prim \gtrsim_\rho \mathbb{REC}$ for some injection ρ. Let $s \in \mathbb{REC}$ be the successor function. There is, by assumption, a function $s' \in Prim$ such that $s' \circ \rho = \rho \circ s$. Since $\rho(0)$ is some constant and $\rho(s(n)) = s'(\rho(n))$, we have that ρ is derived by primitive-recursion from $s' \in Prim$, thus $\rho \in Prim$. Since ρ is a recursive injection, it follows that ρ^{-1} is partial recursive. Define the recursive function $h(n) = \rho(\min_i\{\rho(i) > ack(n,n)\})$, where ack is Ackermann's function. Since $\lambda n.ack(n,n)$ grows faster than any primitive recursive function and $h(n) > ack(n,n)$, it follows that $h \notin Prim$. Since $\mathrm{Im}\ h \subseteq \mathrm{Im}\ \rho$, it follows that $t = \rho^{-1} \circ h \in \mathbb{REC}$. Thus, there is a function $t' \in Prim$, such that $t' \circ \rho = \rho \circ t = \rho \circ \rho^{-1} \circ h = h$. We have arrived at a contradiction: on the one hand, $t' \circ \rho \in Prim$, while, on the other hand, $t' \circ \rho = h \notin Prim$. □

We do not know if the primitive recursive functions ($Prim$) are interpretation-complete, Theorem 68 notwithstanding.

It is common to show that Turing machines and the recursive functions are of equivalent computational power. We show hereby that they are actually isomorphic. We base the proof on known results, given in [26], for example.

Adopting the approach of [26], TM, as a machine, operates over an infinite tape of the symbols $\{0, 1, B\}$, while TM, as a computational model (set of partial functions), operates over $\{0,1\}^*$ [26, p. 116]:

> The tape must be represented finitely in order to define the transition rules. One way is to include all nonblank symbols, so a full tape is obtained by appending infinitely many blanks to each end of a finite tape representation. ... Input and output are strings in TM-data $= \{0,1\}^*$, are found on the first tape, and consist of all symbols to the right of the scanned symbol, extending up to but not including the first blank.

Theorem 69. *Turing machines* (TM)*, over a binary alphabet, and the recursive functions* ($\mathbb{REC}$) *are isomorphic.*

Proof. Since $\mathbb{REC}$ is complete, it is sufficient, by Lemma 64, to show that $\mathbb{REC} \gtrsim \text{TM} \gtrapprox \mathbb{REC}$. Since it is well-known that $\mathbb{REC} \gtrsim \text{TM}$ via Gödelization, it remains to show that $\text{TM} \gtrapprox_\pi \mathbb{REC}$, for some bijection π. Define (as in [26, p. 131]) the bijection $\pi : \mathbb{N} \to \{0,1\}^*$, by

$$\pi(n) \quad = \quad \begin{cases} \epsilon & n = 0 \\ d \;\text{ s.t. } 1d \text{ is the shortest binary} & \\ \qquad\quad \text{representation of } \; n+1 & \text{otherwise} \end{cases}$$

For example, $\pi(i)$ is $\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$ for $i = 0, 1, 2, \ldots$.

As per [26, pp. 131–133], $\text{TM} \gtrapprox_\pi \text{RAM}$ (Random Access Machines); $[\![\text{RAM}]\!] \supseteq [\![\text{CM}]\!]$ (Counter Machines) by [26, pp. 116–118]; and $\text{CM} \gtrsim_\iota \mathbb{REC}$ by [26, pp. 207–208]. We have that $\mathbb{REC} \gtrsim \text{TM} \gtrapprox_\pi \text{RAM} \gtrapprox_\iota \text{CM} \gtrapprox_\iota \mathbb{REC}$; thus, $\text{TM} \cong \mathbb{REC}$. (The exact definitions of RAM and CM are immaterial, as they are only intermediaries in $\text{TM} \gtrapprox_\pi \text{RAM} \gtrapprox_\iota \text{CM} \gtrapprox_\iota \mathbb{REC}$.) □

From the above proof, we also have that random access machines (RAM) and counter machines with unlimited number of counters (CM) have exactly the same extensionality as the recursive functions.

This is not the case with two-counter machines, as elaborated in Section 3.3.4. They are of equivalent power to the recursive functions, however not isomorphic to them. Indeed, they are bijectively-weaker than them (otherwise contradicting the stability of the recursive functions).

By known results, two-stack machines have the same extensionality as Turing machines.

Due to the closure properties of Turing machines and the recursive functions, they are bijectively at least as powerful as some model A if and only if they are decently at least as powerful as A.

Theorem 70. *Let A be a computational model operating over a denumerable domain. Then Turing machines and the recursive functions are decently at* *least as powerful as A if and only if they are bijectively at least as powerful as A. That is,* TM $\succsim A$ *iff* TM $\gtrapprox A$ *iff* $\mathbb{REC} \succsim A$ *iff* $\mathbb{REC} \gtrapprox A$.

Proof. Follows directly from Theorem 28 and the closure properties of the recursive functions. □

4.4 Category theory and computational power

Category theory is an abstract and generic tool for analyzing mathematical areas concerning mappings between objects. Power comparison between computational models seems to be a perfect candidate for such an area; the comparison is done by matching functions of one model with functions of another, according to a mapping between the domains over which they operate. Hence, it looks natural to provide a categorial definition of computational power comparison.

Indeed, it turns out that the power comparison notion (Definition 52), as well as related properties as interpretation-completeness (Definition 22), fit very naturally into the definitions of category theory.

We define below the computational power scheme as a category, named **CompPower**. We do it in steps, each time providing the relevant category theory definitions, followed by the corresponding definitions/properties of the **CompPower** category. Our category is based on the standard **Set** category of sets [1, p. 22] and goes along the lines of the standard **Alg**(Ω) category of algebras [1, p. 22].

All category theory definitions are taken from [1]. They are marked by "Cat-Definition", and linked to their page number in [1].

Simple category. We begin with the very basic definition of a category.

Cat-Definition 1 (Category [1, p. 21])**.** *A* category *is a quadruple* $\mathbf{A} = (\mathcal{O}, hom, id, \circ)$ *consisting of*

1. *A class $\mathcal{O}$, whose members are called* objects
2. *For each pair (A, B) of objects, a set $hom(A, B)$, whose members are called* morphisms *from A to B*
3. *For each object, a morphism $id_A : A \to A$, called the* identity *on A*
4. *A* composition law *associating with each morphisms $f : A \to B$ and $g : B \to C$ a morphism $g \circ f : A \to C$, called the* composite *of f and g*

subject to the following conditions:

(a) composition is associative; i.e. $h\circ(g\circ f) = (h\circ g)\circ f$*,*

(b) the identity morphisms act as identities with respect to composition; i.e. for all morphisms $f : A \to B$ *we have that* $id_B\circ f = f$ *and* $f\circ id_A = f$*,*

(c) the sets $hom(a, b)$ *are pairwise disjoint.*

Categories will be denoted by bold letters, e.g. **Cat**. Morphisms in a category will be denoted by lowercase letters, with uppercase letters reserved for objects.

We provide now the framework of computational power comparison as the **CompPower** category.

Definition 71 (**CompPower**)**.** *The category* **CompPower** *consists of the following:*

1. *Its objects are computational models (Definition 1). That is, any entity that computes a set of functions over some domain. Each object* A *has its domain, denoted* $\mathrm{dom}\ A$*, and its extensionality, denoted* $[\![A]\!]$*, which is the set of (partial) functions that the model computes over its domain.*

2. *The morphisms* $hom(A, B)$ *are the functions* $\rho : \mathrm{dom}\ A \to \mathrm{dom}\ B$*, such that* $[\![B]\!]_\rho \supseteq [\![A]\!]$*. (See Remark 72 below, for the meaning of* $[\![B]\!]_\rho$ *for any function* ρ*.)*

3. *The identity morphism* id_A *is defined by the identity mapping over A's domain* $i : \mathrm{dom}\ A \to \mathrm{dom}\ A$*.*

4. *The composition of morphisms* $\rho : A \to B$ *and* $\eta : B \to C$ *is defined by the composition of functions* $\eta\circ\rho : \mathrm{dom}\ A \to \mathrm{dom}\ C$ *mapping between the domains.*

Note 72. *Model interpretation was defined in Chapter 2 only for injective functions between domains (Definition 4). Nevertheless, it can be analogously defined for any function between domains, as used above.*

Note 73. *We defined above that a morphism* $\rho : A \to B$ *means that* $[\![B]\!]_\rho \supseteq [\![A]\!]$*. One may define the direction of the morphism in the other way round. However, the former is more natural than the latter in category theory, as shown below when discussing "fibres".*

Note 74. *There is a common usage of the term "domain" in category theory in the context of morphisms. For a morphism $f : A \to B$, it is said that A is the domain of the morphism f and B is its codomain. This has nothing to do with our usage of the term "domain" for denoting the set of elements over which a computational model operates.*

We continue with the categorial definition of an isomorphism between objects.

Cat-Definition 2 (Isomorphism [1, pp. 28–291])**.**

- *A morphism $f : A \to B$ in a category is called an* isomorphism *provided that there exists a morphism $g : B \to A$ with $g \circ f = id_A$ and $f \circ g = id_B$.*
- *Such a morphism g is called the* inverse *of f and is denoted by f^{-1}.*
- *Such objects A and B are said to be* isomorphic*.*

Proposition 75. *The standard isomorphism relation between computational models (Definition 7) coincides with the categorial definition of isomorphism, applied to the* **CompPower** *category.*

Construct. We define now the **CompPower** category as a "construct", meaning a "concrete category" over the **Set** category of sets. This is a natural direction due to the close relations between computational power comparison and mapping between sets.

We start with defining a "functor", which is a mapping between categories. This will be required for defining one category on top of another.

Cat-Definition 3 (Functor [1, p. 29])**.** *If* $\mathbf{A}$ *and* $\mathbf{B}$ *are categories, then a* functor *F from $\mathbf{A}$ to $\mathbf{B}$ is a function that assigns to each $\mathbf{A}$-object A a $\mathbf{B}$-object $F(A)$, and to each $\mathbf{A}$-morphism $\rho : A \to A'$ a $\mathbf{B}$-morphism $F(\rho) : F(A) \to F(A')$, in such a way that*

1. *F preserves composition; i.e. $F(\rho \circ \eta) = F(\rho) \circ F(\eta)$ whenever $\rho \circ \eta$ is defined, and*
2. *F preserves identity morphisms; i.e. $F(id_A) = id_{F(A)}$ for each $\mathbf{A}$-object A.*

There are a few standard properties that functors may, or may not, have:

Cat-Definition 4 (Functor properties [1, p. 34])**.** *Let $F : \mathbf{A} \to \mathbf{B}$ be a functor.*

1. *F is called an* embedding *provided that F is injective on morphisms.*

2. *F is called* faithful *provided that all the hom-set restrictions $F : hom_{\mathbf{A}}(A, A') \to hom_{\mathbf{B}}(F(A), F(A'))$ are injective.*

3. *F is called* full *provided that all the hom-set restrictions are surjective.*

4. *F is called* amnestic *provided that an $\mathbf{A}$-isomorphism ρ is an identity whenever $F(\rho)$ is an identity.*

We can define now a "concrete category".

Cat-Definition 5 (Concrete category [1, p. 61])**.** *Let $\boldsymbol{X}$ be a category. A* concrete category *over $\boldsymbol{X}$ is a pair $(\mathbf{A}, U)$, where $\mathbf{A}$ is a category and $U : \mathbf{A} \to \mathbf{X}$ is a faithful functor. Sometimes U is called the* forgetful *(or* underlying*) functor of the concrete category and $\boldsymbol{X}$ is called the* base category *for $(\mathbf{A}, U)$.*

Note 76. *We adopt the following notational convention [1, p. 62]: Since faithful functors are injective on hom-sets, we usually assume that $hom_A(A, B)$ is a subset of $hom_X(U(A), U(B))$ for each pair (A, B) of $\mathbf{A}$-objects. This familiar convention allows one to express the property that "for $\mathbf{A}$-objects A and an $\mathbf{X}$-morphism $f : U(A) \to U(B)$ there exists a (necessarily unique) $\mathbf{A}$-morphism $A \to B$ with $U(A \to B) = f : U(A) \to U(B)$" much more succinctly, by stating "$f : U(A) \to U(B)$ is an $\mathbf{A}$-morphism (from A to B)".*

A concrete category over the category of sets has a special interest in category-theory, and the dedicated name "construct".

Cat-Definition 6 (**Set** [1, p. 22])**.** *The category* **Set** *consists of*

- *The objects are the class of all sets*
- *hom(A, b) is the set of all functions from A to B*
- *id_A is the identity function on A*
- *Composition of morphisms is the usual composition of functions.*

Cat-Definition 7 (Construct [1, p. 61])**.** *A concrete category over* **Set** *is called a* construct

The **CompPower** category may be naturally viewed as a construct:

Proposition 77. *Let U :* **CompPower** $\to$ **Set** *be the functor defined by* $F(A) := \text{dom } A$ *and* $F(\rho : A \to A') := \rho : \text{dom } A \to \text{dom } A'$*. Then* (**CompPower**$, U$) *is a construct.*

We will usually refer to the construct (**CompPower**$, U$) simply by **CompPower**, as the meaning is clear from the context.

Note that the forgetful functor U above is faithful but neither full nor an embedding.

Note 78. *The construct* **CompPower** *is not amnestic since we allow different computational models with the same extensionality. For example, having the recursive functions and the 3-counter machine models as different objects, while they both operate over the natural numbers and compute the same set of functions.*

Note 79. *There are power comparison notions in which the mapping between domains is not a function but rather a relation (see Chapter 5). In such a case* **CompPower** *cannot be viewed as a construct (a concrete category over* **Set***), but may be viewed as a concrete category over the category* **Rel** *of relations. In* **Rel** *the objects are sets and the morphisms are relations between sets. (The notation '***Rel***' is not always used to mark this category. For example in [1] it has a different usage.) Some of these comparison notions are based on a surjective function from the domain of the at least as powerful model to the domain of the other model. In such a case, one may still view* **CompPower** *as a construct but should change the definition of a morphism* $\eta : A \to B$ *to mean that* $[\![A]\!]_\eta \supseteq [\![B]\!]$*. However, this is not very standard, and does not go along with some categorial definitions as "fibres".*

Fibres and types of morphisms. A further investigation of the **CompPower** construct shows additional similarities between our case and standard definitions of category-theory.

The objects of a concrete category are divided into "fibres". A fibre is a preordered class of all objects that are mapped to the same object of the base category. In our case, a fibre gathers all computational models with the same domain and it is ordered by containment over the models' extensionality.

Cat-Definition 8 (Fibre [1, p. 63])**.** *Let* $(\mathbf{A}, U)$ *be a concrete category over* $\boldsymbol{X}$*.*

1. *The* fibre *of an* $\boldsymbol{X}$*-object* X *is the preordered class consisting of all* $\mathbf{A}$*-objects* A *with* $U(A) = X$ *ordered by:*

2. *$A \leq B$ if and only if there exists a morphism $\rho : A \to B$, such that $U(\rho) = id_X$.*

3. *$\mathbf{A}$-objects A and B are said to be equivalent, provided that $A \leq B$ and $B \leq A$.*

4. *$(\mathbf{A}, U)$ is said to be* amnestic *provided that its fibres are partially ordered classes; i.e. no two different $\mathbf{A}$-objects are equivalent.*

5. *$(\mathbf{A}, U)$ is said to be* fibre-small *provided that each of its fibres is small, i.e. a preordered set.*

Note that the definition of amnestic here coincides with that of Cat-definition 4.

CompPower is not fibre-small, as we allow any computational model. It is also not amnestic, as explained in Remark 78.

Proposition 80.

1. *The fibres of* **CompPower** *consist of computational models operating over the same domain.*

2. *We have that $A \leq B$ for computational models A and B if and only if $[\![A]\!] \subseteq [\![B]\!]$.*

3. *We have that A and B are equivalent, for computational models A and B, if and only if $[\![A]\!] = [\![B]\!]$.*

We continue with some standard classifications of morphisms.

Cat-Definition 9 (Monomorphism, Epimorphism and Bimorphism [1, pp. 109–113])**.**

- *A morphism $f : A \to B$ is said to be a* monomorphism *provided that for all pairs of morphisms $h : C \to A$ and $k : C \to A$ such that $f \circ h = f \circ k$, it follows that $h = k$.*

- *A morphism $f : A \to B$ is said to be an* epimorphism *provided that for all pairs of morphisms $h : B \to C$ and $k : B \to C$ such that $h \circ f = k \circ f$, it follows that $h = k$.*

- *A morphism $f : A \to B$ is said to be a* bimorphism *provided that it is simultaneously a monomorphism and an epimorphism.*

Proposition 81.

- *A morphism of* **CompPower** *is monomorphism if and only if its underlying mapping between domains is injective.*
- *A morphism of* **CompPower** *is an epimorphism if and only if its underlying mapping between domains is surjective.*
- *A morphism of* **CompPower** *is a bimorphism epimorphism if and only if its underlying mapping between domains is bijective.*

Power comparison and interpretation completeness. We are now in a position to define the power comparison notion (Definition 52) as well as interpretation-completeness (Definition 22) in the terminology of category theory.

Definition 82.

- *Object B is* at least as powerful *as object A if there is a monomorphism $\rho : A \to B$.*
- *Object B is* bijectively at least as powerful *as object A if there is a bimorphism $\pi : A \to B$.*

Proposition 83. *The power comparison definition (Definition 52) coincides with Definition 82 applied to the* **CompPower** *category.*

Definition 84. *An object A of a concrete category is* interpretation-complete *if for every object B such that $A \leq B$ and for which there is a monomorphism $\rho : B \to A$, it follows that A and B are equivalent.*

Additional properties. We define below some additional properties, while using some notational shortcuts, as explained in Remark 76.

Cat-Definition 10 (Balanced category [1, p. 113])**.** *A category is called* balanced *provided that each of its bimorphisms is an isomorphism.*

Proposition 85. *The category* **CompPower** *is not balanced.*

This is simply because the at least as powerful model may have additional functions.

Cat-Definition 11 (Embedding [1, p. 134]). *Let* $(\mathbf{A}, U)$ *be a concrete category over* $\mathbf{X}$.

- *An* $\mathbf{A}$*-morphism* $f : A \to B$ *is called* initial *provided that for any* $\mathbf{A}$*-object* C *an* $\mathbf{X}$*-morphism* $g : U(C) \to U(A)$ *is an* $\mathbf{A}$*-morphism whenever* $f \circ g : U(C) \to U(B)$ *is an* $\mathbf{A}$*-morphism.*
- *An initial morphism* $f : A \to B$ *that has a monomorphic underlying* $\mathbf{X}$*-morphism* $U(f) : U(A) \to U(B)$ *is called an* embedding.
- *If* $f : A \to B$ *is an embedding, then* (f, B) *is called an* extension *of* A *and* (A, f) *is called an* initial subobject *of* B.

Though the term "embedding" is used in other mathematical fields in a way that resembles our notion of power comparison, it seems not to be the case here. It turns out that even when the underlying mapping between domains is the identity, the mapping between computational models need not be an initial mapping or an embedding.

Proposition 86. *There are non-initial morphisms* $\rho : A \to B$ *in* **CompPower***, such that the underlying mapping* $U(\rho)$ *between domains is the identity.*

Proof. Let A, B and C be three computational models such that $A \subsetneq C \subsetneq B$. We have a morphism $\rho : A \to B$ on top of the identity mapping between domains, as well as a morphism $\eta : C \to B$ on top of the identity mapping between domains. However, we don't have a morphism from C to A based on the identity mapping between domains, as $A \subsetneq C$. $\square$

Related categories. There are known categories for automata simulation and for algebraic embedding [1, pp. 22–23]. The first corresponds to "mechanical" simulations, mapping between automata by mapping between their internal elements (states, transition function, etc..). The latter provides the standard embedding of model-theory, which maps between functions with the same name.

Chapter 5

Philosophical justifications for the comparison notion

We discuss, in this chapter, the subjectiveness of viewing the mathematical properties of nature, and the possibility of comparing computational models having alternate views of the world. For that purpose, we propose a conceptual framework for power comparison, by linking computational models to hypothetical physical devices. Accordingly, we deduce a mathematical notion of relative computational power, allowing for the comparison of arbitrary models over arbitrary domains (Definition 87). When considering models that have the identity function and that are closed under functional composition, this notion coincides with our decent-comparison notion (Definition 52).

We are only interested in the computational aspect of computational models, that is, which problems can be solved by a model, regardless of the solution's complexity or the model's mechanisms. Hence, we are interested in models' extensionality, which is a set of (partial) functions (or multivalued functions) over the domain of its operation (See Definitions 1 and 60).

A reasonable starting point for comparing models over different domains is the belief that isomorphic models are of identical power. That is, models computing the same set of functions, up to a different naming of their domain elements, are – for all intents and purposes – deemed equipotent. Isomorphism may be a good starting point, but it is not general enough, if only because one model might operate over a domain of a larger cardinality than of the other.

We propose a general, philosophical, definition of a computational model: we understand a computational model to be a mathematical modeling and idealization of some hypothetical physical device, from a specific point of view of the world. Accordingly, we provide some possible meaning of relative computational power. In Section 5.2, we formalize this meaning into a mathematical notion.

5.1 Conceptual framework

We start with a general, philosophical, definition of a computational model, and then suggest some possible meaning of relative computational power. This intuition will be formalized in Section 5.2. It should be noted that the conceptual framework is inspired by linkage to hypothetical physical devices, providing a definition with "world entities"; however, these entities are later obviated by mathematical simplifications.

5.1.1 What is a computational model?

We can think of a computational model as a mathematical modeling and idealization of some hypothetical physical devices, from a specific point of view of the world (see Figure 5.1).

- A physical device receives a physical input and returns a physical output. For example, an electric device may take some electric voltage at two of its pins as input, and return a voltage at two other pins as output.
- A corresponding computational model takes a specific point of view of the physical world. For example, a model of a digital computer might view a voltage lower than 0.5v as the binary value 0 and of 0.5v or higher as 1. That is, the domain of the model, D, is a "view" of the physical world, W. This view is a partial surjective function $v : W \to D$.
- The device computes a function on world entities (in our example above, $\xi : \mathbb{R} \to \mathbb{R}$), while from the model's point of view it computes a function on its domain (in our example, $f : \{0, 1\} \to \{0, 1\}$).

A computational model, by itself, can be viewed as a "black box," computing a set of partial functions. The domain and range of functions are identical.

The modeling of a hypothetical device from a specific point of view of the world will be at the heart of our method of comparing different models. The world can be chosen to be any set of cardinality at least as large as the cardinality of the model's domain.

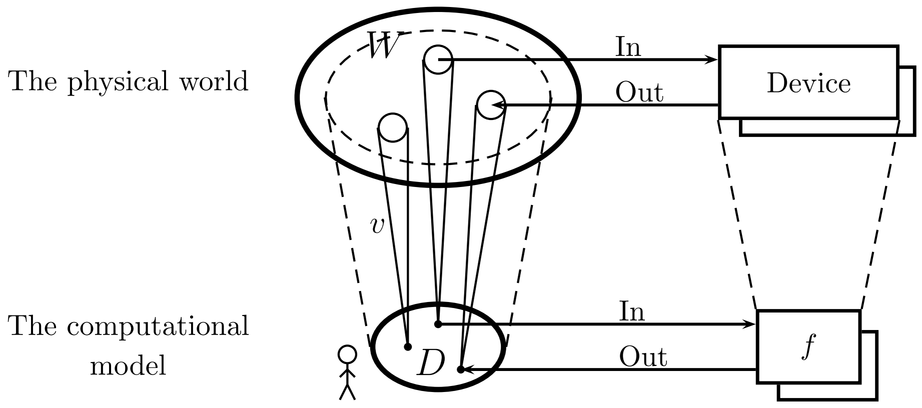


Figure 5.1: A computational model is a mathematical modeling of some hypothetical physical devices, from a specific point of view of the world

The idea that a model encapsulates a point of view of the world is shared by Minsky [30]:

> We use the term "model" in the following sense: To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A. The model relation is inherently ternary.... It is understood that B's use of a model entails the use of encodings for input and output, both for A and for A*. If A is the world, questions for A are experiments.

World size. We want to allow the world-domain W to be as big as required, as well as the resolution of its elements to be enlarged as much as required. That is, all elements $x \in W$ may be considered as sets of the same cardinality.

Different domain and range. There are models with different domain and range, like string input and Boolean output. A generalized view is to consider the "actual" model's domain to be the union of the original domain and range.

Uniform computation. It is common to have models with functions of any fixed arity ($f(x)$, $g(x,y)$, $h(x,y,z)$, $\ldots$), like the recursive functions, for instance, while we deal here only with unary functions ($f(x)$). We consider the "actual" domain (and range) of the former models to be the set of all finite *tuples* of elements of the original domain, allowing us to restrict attention to unary functions. This accords with the standard view taken for

Turing machines, the view taken for the BSS model [6, pp. 69–70], and implicitly the view taken for the recursive functions when compared to Turing machines.

Computing over structures. There are models defined over structures, that is, over sets of elements, equipped with "built-in" functions and relations. See, for example, [6, 13, 47]. We consider the underlying set to be the domain, and include the structure's functions and relations in the model.

Local nondeterminism. A model with nondeterministic functions, operating within equivalence classes, may be interpreted as a deterministic model operating over those equivalence classes. For example, we can view a digital computer as operating over intervals of real-valued voltages.

5.1.2 Comparing computational power

We generally say that model B is conceptually at least as powerful as model A, written $B \succcurlyeq A$, if it can do whatever A does. When both models have the same domain representation, it means "containment": B is at least as powerful as A if it computes all the functions that A does. The question is how should one compare models operating over different domains, as they compute formally-different functions.

We say that B is conceptually at least as powerful as A if B has the potential to do whatever A does for *every* possible user, and has the abstraction capabilities of A. See Figures 5.2 and 5.3, and Definition 87.

We formalize the above characterizations as mathematical notions in the next section.

5.2 The formal comparison notion

We formalize here the conceptual framework of the previous section.

Notation. We use here the Z-standard [14] for function arrows. Specifically, $⇸$ denotes a partial function, $\twoheadrightarrow$ is used for a total surjective function, $\rightarrowtail$ is an injection, and $⤖$ is a bijection. We use double-arrows for mappings (multi-valued functions). So $\rightrightarrows$ denotes a total surjective mapping, and $\nrightrightarrows$ a partial surjective mapping.

We consider a *computational model* A over domain D to be an abstraction of an object that computes a set of partial functions $f : D ⇸ D$. Accordingly, we get the definition given in Definition 1.

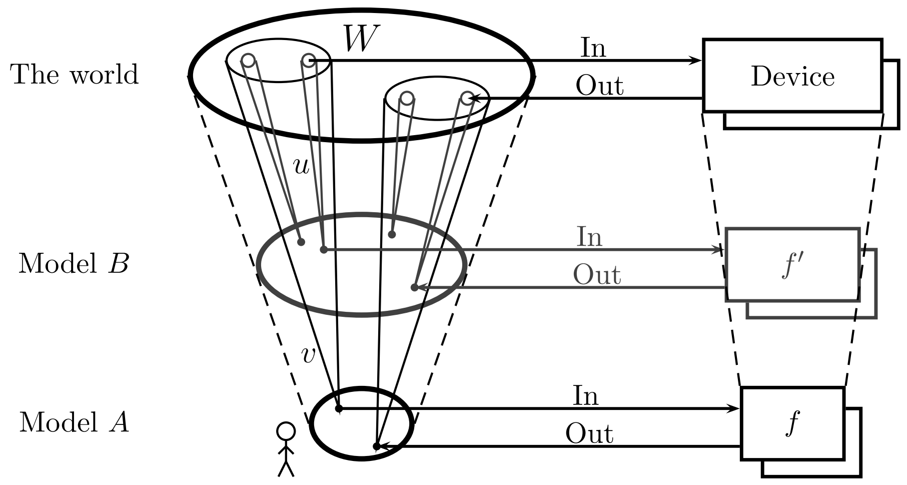


Figure 5.2: The at least as powerful model, B, has the potential to provide all the functionality of the other model, A, from any user point of view

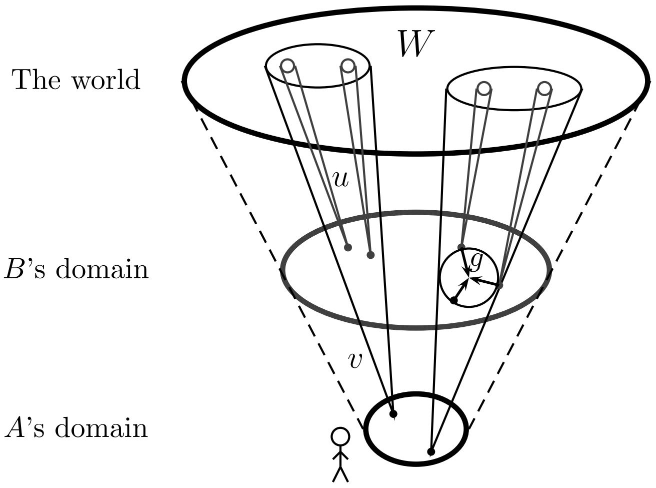


Figure 5.3: The function $g \in [\![B]\!]$ provides model B the abstraction capabilities of model A

Some clarifications regarding function notations:

- Two partial functions, f and g, over the same domain are (extensionally) *equal*, denoted $f = g$, if they are defined for exactly the same elements of the domain and have the same value whenever they are both defined.

- A function $f : D \mapsto D'$ is extended to subsets $X \subseteq D$ of the domain by $f(X) := \{f(x) \mid x \in X\}$.

- A *mapping* $\rho : D \rightrightarrows D'$ is a binary relation between D and D', that is, a subset of $D \times D'$. Its inverse ρ^{-1} is defined, as usual, as $\{\langle y, x\rangle \mid \langle x, y\rangle \in \rho\}$. Any mapping may also be viewed as a total function $\rho : D \to \mathscr{P}(D')$, from D to subsets of D', in the sense that $\rho : x \mapsto \{y \mid \langle x, y\rangle \in \rho\}$. Thus, its inverse, $\rho^{-1} : D' \rightrightarrows D$, is the function $\rho^{-1} : D' \to \mathscr{P}(D)$, from D' to the subsets of D, such that $\rho^{-1} : y \mapsto \{x \mid y \in \rho(x)\}$.

- A *total surjective mapping* $\rho : D \rightrightarrows\!\!\!\!\!\!\rightarrow D'$ is a total function, $\rho : D \to \mathscr{P}(D')$, from D to the subsets of D', such that $\bigcup_{x \in D} \rho(x) = D'$, and $\rho(x) \neq \emptyset$ for all $x \in D$.

Conceptually, we demanded, in Section 5.1, that the at least as powerful model, B, should have the potential to provide all the functionality of the other model, A, from any user point of view. How should we formalize it? We will require that the user of model A will be able to use a physical device modeled by B instead of the device modeled by A. That is, the user will get exactly the same results, from her point of view, as if she was using the device modeled by A.

An element $x \in \operatorname{dom} A$ represents some physical elements $v^{-1}(x)$, where $v : W \mapsto\!\!\!\!\rightarrow \operatorname{dom} A$ is the view of the world W by A. Model B views these physical elements as $u \circ v^{-1}(x)$ (which might be many different elements in dom B), where $u : W \mapsto\!\!\!\!\rightarrow \operatorname{dom} B$ is the view of the world by B. Analogously, an element $y \in \operatorname{dom} B$ represents some physical elements $u^{-1}(y)$, which are viewed by A as $v \circ u^{-1}(y)$. Hence, a function $f' \in [\![B]\!]$ provides the functionality of a function $f \in [\![A]\!]$, from the point of view of a user of A, if $v \circ u^{-1} \circ f' \circ u \circ v^{-1}(x) = \{f(x)\}$ for all $x \in \operatorname{dom} A$. See an illustration in Figure 5.2.

In addition, we require that B should have an "abstraction" function, which ensures that when it has a more detailed view of the world, it may also gather various points into a single one, obtaining the abstraction capabilities of A. See an illustration in Figure 5.3.

We formalize below the comparison notion, as explained above. We then prove that it is equivalent to a simplified notion, in which the "world views" are replaced with a direct mapping between model domains.

Definition 87 (Conceptual Power Comparison)**.** *Model B is* conceptually at least as powerful as *model A, denoted $B \succsim A$, if for every domain W (the world) and partial surjection $v : W \nrightarrow\!\!\!\!\twoheadrightarrow \operatorname{dom} A$ (the view of W by A), there are a partial surjection $u : W \nrightarrow\!\!\!\!\twoheadrightarrow \operatorname{dom} B$ (the view of W by B) and a function $g \in [\![B]\!]$ (the abstraction function), such that*

(a) for every function $f \in [\![A]\!]$ there is a function $f' \in [\![B]\!]$, such that $v \circ u^{-1} \circ f' \circ u \circ v^{-1}(x) = \{f(x)\}$ for all $x \in \operatorname{dom} A$,

(b) $g(z) = g(y)$ iff $v \circ u^{-1}(z) = v \circ u^{-1}(y)$ for all $y, z \in \operatorname{dom} B$, and

(c) $v \circ u^{-1} \circ g(y) = v \circ u^{-1}(y)$ for all $y \in \operatorname{dom} B$.

The first condition, (a), says that B computes every function of A, up to the mapping between the domains ($v \circ u^{-1}$). Condition (b) says that the (abstraction) function $g \in [\![B]\!]$ distinguishes between the equivalence classes generated by the mapping, while (c) says that the distinction is made by choosing a representative element within each class.
Comparison via a correlation mapping that is a partial surjective mapping is in the spirit of the "representation" of [51, p. 33].

Comparison up to equivalence classes. With the definitions above, we required the as-powerful model, B, to have all the functionality of model A, providing exactly the same values up to the mapping between domains ($v \circ u^{-1} \circ f' \circ u \circ v^{-1}(x) = \{f(x)\}$). This requirement may be loosened, by requiring B to provide the same values as A, up to the distinguishing ability of A. That is, two elements, e and r, are indistinguishable by model A if for all $f \in [\![A]\!]$ we have that $f(e) = f(r)$. Indistinguishable elements, e and $r \in \operatorname{dom} A$, may be considered equivalent with respect to A, denoted $e =_A r$. The comparison notion above may be loosened accordingly, denoted $\succsim'$, by requiring that $v \circ u^{-1} \circ f' \circ u \circ v^{-1}(x) =_A \{f(x)\}$.

Simplifications. The comparison notion may be simplified, replacing the "world entities" with a direct mapping between the models' domains.

Theorem 88. *Model B is conceptually at least as powerful as model A if and only if there are a total surjective mapping $\rho : \operatorname{dom} B \rightrightarrows\!\!\!\!\!\twoheadrightarrow \operatorname{dom} A$ (correlation mapping) and function $g \in [\![B]\!]$ (abstraction function), such that:*

(a) for every function $f \in [\![A]\!]$ there is a function $f' \in [\![B]\!]$ such that $\rho \circ f' \circ \rho^{-1}(x) = \{f(x)\}$ for all $x \in \mathrm{dom}\ A$,

(b) $g(z) = g(y)$ iff $\rho(z) = \rho(y)$ for all $y, z \in \mathrm{dom}\ B$, and

(c) $\rho \circ g(y) = \rho(y)$ for all $y \in \mathrm{dom}\ B$.

Proof. Assume that $B \succcurlyeq A$. Since Definition 87 refers to every domain W (world), it follows that if there is a view of W by A by a partial surjection $v : W \twoheadrightarrow \mathrm{dom}\ A$, then there is also a view of another domain $W' := \mathrm{Im}\ v^{-1} \cap \mathrm{Im}\ u^{-1}$ (over which both v and u are defined) by A by a total surjection $v' : W' \twoheadrightarrow \mathrm{dom}\ A$. Hence, the corresponding view of W' by B is also a total surjection $u' : W' \twoheadrightarrow \mathrm{dom}\ B$. Define a total surjective mapping $\rho : \mathrm{dom}\ B \rightrightarrows \mathrm{dom}\ A$, by $\rho := v' \circ u'^{-1}$. We have that requirements (a)–(c) above are fulfilled.

As for the other direction, assume a total surjective mapping $\rho : \mathrm{dom}\ B \rightrightarrows \mathrm{dom}\ A$ and a function g, via which requirements (a)–(c) above are fulfilled. We construct the proof in three steps:

1. *Specific world and views.* Define a domain W (a world) as a subset of $\mathrm{dom}\ A \times \mathrm{dom}\ B$, by $W := \{\langle a, b\rangle \mid a \in \rho(b)\}$. Define total surjective functions $v : W \twoheadrightarrow \mathrm{dom}\ A$ and $u : W \twoheadrightarrow \mathrm{dom}\ B$ (the views) by $v(\langle a, b\rangle) := a$ and $u(\langle a, b\rangle) := b$, for all $\langle a, b\rangle \in W$. We have specific views by A and B for which the conditions of the conceptual definition is satisfied, that is for every function $f \in [\![A]\!]$ there is $f' \in [\![B]\!]$, such that $v \circ u^{-1} \circ f' \circ u \circ v^{-1}(x) = \{f(x)\}$ for all $x \in \mathrm{dom}\ A$.

2. *Specific world and all views.* Let W, v and u be as in the previous step. Let $m : W \twoheadrightarrow \mathrm{dom}\ A$ be an arbitrary view by A of the world W. By the conceptual assumption on the world size, we may consider the domain W as a domain W', by replacing each element $y \in W$ with a set Y of cardinality $|W|$. Accordingly, the view m becomes a view $v' : W' \twoheadrightarrow \mathrm{dom}\ A$, where $|v'^{-1}(x)| = |W|$ for all $x \in \mathrm{dom}\ A$. Define a partial surjective function $\xi : W' \twoheadrightarrow W$, such that $v'(z) = v \circ \xi(z)$ for all $z \in W'$. Define the view by B of the world, as the partial surjective function $u' : W' \twoheadrightarrow \mathrm{dom}\ B$, by $u'(z) := u \circ \xi(z)$ for all $z \in W'$. We have that for every function $f \in [\![A]\!]$ there is $f' \in [\![B]\!]$, such that $v' \circ u'^{-1} \circ f' \circ u' \circ v'^{-1}(x) = \{f(x)\}$ for all $x \in \mathrm{dom}\ A$.

3. *All worlds and all views.* Let W be as in the previous step. Let $\widetilde{W}$ be an arbitrary world and $\widetilde{v} : \widetilde{W} \twoheadrightarrow \mathrm{dom}\ A$ an arbitrary view by A of the world $\widetilde{W}$. By the conceptual assumption on the world size

we can enlarge the world $|\widetilde{W}|$ as required, thus we may assume that $|\widetilde{W}| \geq |W|$. Define a total surjective function $\tau : \widetilde{W} \twoheadrightarrow W$ such that $\tau(x) = \tau(y)$ implies that $\widetilde{v}(x) = \widetilde{v}(y)$ for every $x, y \in \widetilde{W}$. Define a view $v : W \mapsto\!\!\!\twoheadrightarrow A$ by $v := \widetilde{v}\circ\tau^{-1}$. By the previous step there is a corresponding view $u : W \mapsto\!\!\!\twoheadrightarrow B$, hence we have the required view $\widetilde{u} : \widetilde{W} \mapsto\!\!\!\twoheadrightarrow B$ by $\widetilde{u} := u\circ\tau$.

□

Example 89. *Consider a modeling of a simple electric-cable by some computational model, denoted EC, providing only the identity function over the reals. Then* $\mathrm{TM} \not\succsim EC$ *and* $EC \not\succsim \mathrm{TM}$.

Example 90. *Let* $\mathbb{REC}$ *be the standard recursive functions over* $\mathbb{N}$*, and* $\mathbb{RR}$ *a duplication of the recursive functions over two different copies of the natural numbers (e.g. red and blue numbers). That is,* $\mathrm{dom}\ \mathbb{RR} = \mathbb{N} \uplus \widetilde{\mathbb{N}}$*, where* $\widetilde{\mathbb{N}}$ *is a different copy of the natural numbers, and every function of* $\mathbb{RR}$ *operates independently on each copy of the naturals. Meaning, for every* $g \in \mathbb{RR}$ *we have that* $g(x) \in \mathbb{N}$ *if* $x \in \mathbb{N}$ *and* $g(x) \in \widetilde{\mathbb{N}}$ *if* $x \in \widetilde{\mathbb{N}}$*. We have then that* $\mathbb{REC} \not\succsim \mathbb{RR}$*, while* $\mathbb{RR}$ *is not as powerful as* $\mathbb{REC}$*, since it does not have the necessary abstraction function.*

Closure under composition and inclusion of the identity function. When the at most as powerful model includes the identity function ($\lambda x.x$), the comparison notion may be simplified, requiring the correlation mapping (ρ) to be a surjective function instead of a surjective mapping. When the as-powerful model is closed under functional composition, it may be further simplified, getting the decent-comparison notion (Definition 52).

Theorem 91. *Let A be a computational model with the identity function* ($\lambda x.x \in [\![A]\!]$)*, then model B is conceptually at least as powerful as A if and only if there exist a total surjective function* $\varphi : \mathrm{dom}\ B \twoheadrightarrow \mathrm{dom}\ A$ *and a function* $g \in [\![B]\!]$*, such that:*

1. *for every function* $f \in [\![A]\!]$ *there is a function* $f' \in [\![B]\!]$ *such that* $\varphi\circ f'(y) = f\circ\varphi(y)$ *for all* $y \in \mathrm{dom}\ B$*,*
2. $\varphi\circ g(y) = \varphi(y)$ *for all* $y \in \mathrm{dom}\ B$*, and*
3. $g(z) = g(y)$ *iff* $\varphi(z) = \varphi(y)$ *for all* $y, z \in \mathrm{dom}\ B$*.*

Proof. The first direction is obvious, as a function is also a mapping. For the other direction, let A be a computational model with the identity function

($\iota := \lambda x.x \in [\![A]\!]$), and let model $B \succcurlyeq A$ via a total surjective mapping $\rho : \text{dom } B \rightrightarrows \text{dom } A$. Assume, by contradiction, that ρ is not a function. Accordingly, there are elements $e \neq t \in \text{dom } A$ and $z \in \text{dom } B$, such that $\rho^{-1}(e) = \rho^{-1}(t) = z$. Since $B \succcurlyeq A$, it follows that there is a function $\iota' \in [\![B]\!]$, such that $\rho \circ \iota' \circ \rho^{-1}(x) = \{\iota(x)\}$ for all $x \in \text{dom } A$. Therefore, $\rho \circ \iota'(z) = \{e\} = \{d\}$. Contradiction. □

When the as-powerful model is closed under functional composition, it may be further simplified, replacing the surjective function with an opposite injection ($\psi : \text{dom } A \rightarrowtail \text{dom } B$). Hence, getting the decent-comparison notion (Definition 52).

Theorem 92. *Let A be a computational model with the identity function ($\lambda x.x \in [\![A]\!]$). Then a model B, closed under functional composition, is conceptually at least as powerful as A if and only if B is decently at least as powerful as A. That is, under the above conditions, $B \succcurlyeq A$ iff $B \gtrsim A$.*

Proof. Let model A with the identity function, and model B closed under functional composition.

Assume $B \succcurlyeq A$ via a total surjective function $\varphi : \text{dom } B \twoheadrightarrow \text{dom } A$ and a function $g \in [\![B]\!]$, as in Theorem 91. Define an injection $\rho : \text{dom } A \rightarrowtail \text{dom } B$ by $\rho := g \circ \varphi^{-1}$. Since $B \succcurlyeq A$ via φ, it follows that for every function $f \in [\![A]\!]$ there is a function $f' \in [\![B]\!]$, such that $\varphi \circ f'(y) = f \circ \varphi(y)$ for all $y \in \text{dom } B$. By the closure of B to functional composition, we have that there is also a function $f'' \in [\![B]\!]$, such that $f'' = g \circ f'$. Thus, for every $x \in \text{dom } A$ we have that $f(x) = \varphi \circ f' \circ \varphi^{-1} = \varphi \circ g^{-1} \circ f' \circ g \circ \varphi^{-1} = \rho^{-1} \circ f' \circ \rho$. Hence, $B \gtrsim A$ via ρ and g.

As for the other direction, we get an analogous proof by defining the total surjective function $\varphi : \text{dom } B \twoheadrightarrow \text{dom } A$ by $\varphi := \rho^{-1} \circ g$. By the assumption that $B \gtrsim A$ via ρ, we have that for every $f \in [\![A]\!]$ there is $f' \in [\![B]\!]$, such that $f(x) = \rho^{-1} \circ f' \circ \rho$ for all $x \in \text{dom } A$. Thus, by the closure of B to function composition it also has a function $f'' = f' \circ g$. Hence, for every $y \in \text{dom } B$ we have that $\varphi \circ f''(y) = f \circ \varphi(y)$, as required. □

5.3 Ramifications of familiar notions

Various methods have been used to compare the computational power of competing models.

Extended domains. It is common to claim that a function can be replaced by any of its extensions. That is, a function $f : D \to D$ can be replaced by $f' : D' \to D'$ if $D \subseteq D'$ and $f = f' \restriction_D$. See, for example, [15, p. 654]: "Here we adopt the convention that a function on $\mathbb{N}$ is in an analog class $\mathcal{C}$ if some extension of it to $\mathbb{R}$ is, i.e. if there is some function $\widetilde{f} \in \mathcal{C}$ that matches f on inputs in $\mathbb{N}$."

By the conceptual framework, "B extends A" can be interpreted as "B having the potential to be at least as powerful as A for a user who has both domain views." For example, one can consider a user who views the world as real numbers, but can identify the natural numbers among them.

This approach is not appropriate as a general power comparison notion, since the extended model B doesn't necessarily have the abstraction capabilities of A. For example, a mathematician working with paper and pencil may consider various physical entities to "be" the symbol "a" (e.g. a, a, a, a, $\mathtt{a}$). A model that lacks the abstraction of the various "a"s, treating each of them totally differently, is not as powerful. Another example is a model that accurately doubles a real number ($\lambda x.2x$). It cannot replace the doubling function on the natural numbers for a user who doesn't have the ability to see so accurately (as might be the case with human users). Consider, also, a model with real input (as voltage or frequency) having a function h that is exactly the TM-halting function for integer inputs, but has the value 0 otherwise. So that unless we give it the exact value as input, the result is meaningless. Would one say that it is hypercomputational in any meaningful sense?

Embedding. Extending the domain is a special case of "embedding", which is power comparison based on an injection between domains, lacking the requirement for an abstraction function. This is exactly the basic power comparison definition given in Definition 52. The reasons for the inadequacy of embedding as a generic power comparison notion are analogous to that of domain-extending.

Effective encoding. A common approach for comparing models over different domains is to require some manner of effectiveness of the encoding; see [20, p. 21] and [25, p. 290], for example. Two basic methods are usually applied for effective encoding:

1. One can demand informal effectiveness: "The coding is chosen so that it is itself given by an informal algorithm in the unrestricted sense" [38, p. 27].

2. Or one can require encoding effectiveness via a specific model, say, Turing machines: "The Turing-machine characterization is especially

> convenient for this purpose. It requires only that the expressions of the wider classes be expressible as finite strings in a fixed finite alphabet of basic symbols" [38, p. 28].

By the conceptual framework, an "effective comparison" means that B is at least as powerful as A for a human user, assuming humans are capable of "effective" representations.

Effectivity is a useful notion; however, it is unsuitable as a general power comparison notion. The first, informal approach is too vague, while the second can add computational power when dealing with subrecursive models and is inappropriate when dealing with non-recursive models.

We discuss the issue of effectiveness further in the next chapter.

Chapter 6

Effective Computation

Background. In 1936, Alonzo Church and Alan Turing each formulated a claim that a particular model of computation completely captures the conceptual notion of "effective" computability. Church [16, p. 356] proposed that effective computability of numeric functions be identified with Gödel and Herbrand's general recursive functions, or equivalently, with Church and Kleene's lambda-definable functions of positive integers. Similarly, Turing [48] suggested that his computational model, namely, Turing machines, could compute anything that might be mechanically computable, but his interests extended beyond numeric functions.

Church's original thesis concerned functions over the natural numbers with their standard interpretation [16, p. 346, including fn. 3] (emphasis ours):

> The purpose of the present paper is to propose a definition of effective calculability. As will appear, this definition of effective calculability can be stated in either of two equivalent forms, (1) that a *function of positive integers* will be called effectively calculable if it is λ-definable. . . , (2) that a *function of positive integers* shall be called effectively calculable if it is recursive. . . .

Kleene, when speaking about Church's Thesis, also refers to functions over the natural numbers [27, pp. 58, 60] (emphasis ours):

> We entertain various proposition *about natural numbers* . . . This heuristic fact [all recognized effective functions turned out to be general recursive], as well as certain reflections on the nature of symbolic algorithmic processes, led Church to state the following thesis. The same thesis is implicit in Turing's description of computing machines.

> THESIS I. Every effectively calculable function (effectively decidable predicate) is general recursive.

Turing, on the other hand, explicitly extends the notion of "effective" beyond the natural numbers [49, fn. p. 166] (emphasis added):

> We shall use the expression "computable function" to mean a function calculable by a machine, and we let "effectively calculable" refer to the intuitive idea without particular identification with one of these definitions. *We do not restrict the values taken by a computable function to be natural numbers*; we may for instance have computable propositional functions.

But for Turing, even numerical calculations operate on their string representation.

Turing's model of computability was instrumental in the wide acceptance of Church's Thesis. As Trakhtenbrot explained [46]:

> This is the way the miracle occurred: the essence of a process that can be carried out by purely mechanical means was understood and incarnated in precise mathematical definitions.

Objective. Our purpose, in Section 6.1, is to formalize and analyze the Church-Turing Thesis when referring to functions over arbitrary domains.

Based on this definition, and due to the interpretation-completeness of Turing machines, we define, in Section 6.2, effective representations of constructible domains.

Equipped with a plausible interpretation of the Church-Turing Thesis over arbitrary domains, we investigate, in Section 6.3, the general class of "effective computational models".

6.1 The Church-Turing Thesis over arbitrary domains

Simply put, the Church-Turing Thesis is not well defined for arbitrary domains: the choice of domain interpretation might have a significant influence on the outcome. We explore below the importance of the domain interpretation and suggest how to overcome this problem. The main point is that regardless of how numbers or strings are represented in an effective model of computation, the functions computed are general recursive / Turing computable. Accordingly, we consider the effectiveness of a set of functions rather than of a single function (see Thesis A below).

Computational model versus single function. A single function over an arbitrary domain cannot be classified as computable or not. Its computability depends on the representation of the domain.[1] For example, the (uncomputable) halting function over the natural numbers (sans the standard order) is isomorphic to the simple parity function, under a permutation of the natural numbers that maps the usual codes of halting Turing machines to strings ending in "0", and the rest of the numbers to strings ending with "1". The result is a computable standalone "halting" function.

An analysis of the classes of number-theoretic functions that are computable relative to different notations (representations) is provided by Shapiro [40, p. 15]:

> It is shown, in particular, that the class of number-theoretic functions which are computable relative to every notation is too narrow, containing only rather trivial functions, and that the class of number-theoretic functions which are computable relative to some notation is too broad containing, for example, every characteristic function.

An intuitive approach is to restrict the representation only to "natural" mappings between the domains. However, when doing so in the scope of defining "effectiveness", one must use a vague and undefined notion.

This problem was already pointed out by Richard Montague in 1960 [32, pp. 430–431]:

> Now Turing's notion of computability applies directly only to functions on and to the set of natural numbers. Even its extension to functions defined on (and with values in) another denumerable set S cannot be accomplished in a completely unobjectionable way. One would be inclined to choose a one-to-one correspondence between S and the set of natural numbers, and to call a function f on S computable if the function of natural numbers induced by f under this correspondence is computable in Turing's sense. But the notion so obtained depends on what correspondence between S and the set of natural numbers is chosen; the sets of computable functions on S correlated with two such correspondences will in general differ. The natural procedure is to restrict consideration to those correspondences which

[1]There are functions that are inherently uncomputable, via all domain representations. For example, a permutation of some countable domain, in which the lengths of the orbits are exactly the standard encodings of the non-halting Turing machines.

> are in some sense ‘effective’, and hence to characterize a computable function on S as a function f such that, for some effective correspondence between S and the set of natural numbers, the function induced by f under this correspondence is computable in Turing’s sense. But the notion of effectiveness remains to be analyzed, and would indeed seem to coincide with computability.

Stewart Shapiro suggests a definition of “acceptable notation”, based on several intuitive concepts [40, p. 18]:

> This suggests two informal criteria on notations employed by algorithms:
>
> (1) The computist should be able to *write* numbers in the notation. If he has a particular number in mind, he should (in principle) be able to write and identify tokens for the corresponding numeral.
>
> (2) The computist should be able to *read* the notation. If he is given a token for a numeral, he should (in principle) be able to determine what number it denotes.
>
> It is admitted that these conditions are, at best, vague and perhaps obscure.

Michael Rescorla argues that the circularity is inherent in the Church-Turing Thesis [36]:

> My argument turns largely upon the following constraint: a successful conceptual analysis should be *non-circular*.... I will suggest that purported conceptual analysis involving Church’s thesis generate a subtle yet ineliminable circularity: they characterize the intuitive notion of computability by invoking the intuitive notion of computability.... So that syntactic analysis can illuminate the computable number-theoretic functions, we correlate syntactic entities with non-syntactic entities like numbers. We endow the syntax with a primitive semantics. I submit that, in providing this semantics, we must deploy the intuitive notion of computability. Specifically, we must demand that the semantic correlation between syntactic entities and non-syntactic entities itself be computable. But then the proposed analysis does not illuminate computability non-circularly.

A possible solution is to allow any representation (injection between domains), while checking for the effectiveness of an entire computational model. That is, to check for the computability of a function together with the other functions that are computable by that computational model. The purpose lying behind this idea is to view the domain elements as arbitrary objects, deriving all their meaning from the model's functions. For example, it is obvious that the halting function has a meaning only if one knows the order of the elements of its domain. In that case, the successor function provides the meaning for the domain elements.

Two variants of this solution are to either allow only bijective representations, or else alow injections provided that their images are computable. These variations correspond to the variants of our comparison notion (Definition 52).

Adopting the above approach of checking for computability of an entire computational model, we interpret the Church-Turing Thesis as follows:

> **Thesis A.** *All "effective" computational models are of equivalent power to, or weaker than, Turing machines.*

By "effective", in quotes, we mean effective in its intuitive sense.

By a "computational model" we refer to any object that is associated with a set of partial functions (Definition 1). By "equivalent to, or weaker than" we refer to the comparison notions of Definition 52.

A strict supermodel of the recursive functions (or Turing machines) is a "hypercomputational" model.

Definition 93 (Hypercomputational Model)**.** *A computational model H is* hypercomputational *if it is stronger than Turing machines. That is, if $H \gnsim \mathrm{TM}$. (The corresponding variations, using bijective power comparison or decent power comparison, are $H \gnapprox \mathrm{TM}$ or $H \succnsim \mathrm{TM}$.)*

Our interpretation of the Church-Turing Thesis (Thesis A) agrees with Rice's [37, p. 298] and Rabin's definition of a computable group [35, p. 343]:

> DEFINITION 3. An *indexing* of a set S is a one to one mapping $i : S \to I$ such that $i(S)$ is a recursive subset of I. ...
> DEFINITION 4. An indexing i of a group G is *admissible* if the function m from $i(G) \times i(G)$ into $i(G)$ ... is a computable function. ...
> DEFINITION 5. A group is *computable* if it possesses at least one *admissible* indexing.

Rice and Rabin refer to groups, fields, and rings; however, the idea naturally generalizes to any algebraic structure, as done by Lambert [28, p. 594]:

> Following Rabin ... we let an (*admissible*) *indexing* for structure $\mathfrak{U}$ be a 1-1 function $\kappa : A \to \omega$ such that
> (i) $K =$ range κ is recursive;
> (ii) each $\kappa^*(F_a)$ and $\kappa^*(R_a)$ are recursive relative to K ..., where κ^* applied to an (unmixed) operation or relation in A is the operation or relation in ω naturally induced by κ....
> $\mathfrak{U}$ is *computable* iff there is an indexing for $\mathfrak{U}$.

The term "computable group" was introduced by Rice in [37] and, independently, by Rabin in his dissertation on 1956. In the same year, Fröhlich and Shepherdson [21] introduced "explicit algebraic systems", which are identical to "computable systems" in the sense of Rice and Rabin, up to isomorphism (see [35, p. 343] for a detailed comparison between these definitions).

Influence of representations. The Church-Turing Thesis, as stated above (Thesis A), matches the intuitive understanding only due to the interpretation-completeness of Turing machines (Theorem 31). Were the thesis defined in terms of two-counter machines (2CM), for example, it would make no sense: a computational model is not necessarily stronger than 2CM even if it computes strictly more functions (see Section 3.3.4). This is also the case with the untyped lambda calculus (see Section 4.3).

6.2 Effective representations

What is an effective representation? We argued above that a "natural representation" must be a vague notion when used in the context of defining effectiveness. We avoided the need of restricting the representation by checking the effectiveness of entire computational models. But what if we adopt the Church-Turing Thesis (Thesis A); can we then define what is an effective string representation?

Simply put, there is a problem here. Turing machines operate only over strings. Thus a string representation, which is an injection from some domain D to Σ^*, is not itself computable by a Turing machine. All the same, when we consider, for example, string representations of natural numbers, we can obviously say regarding some of them that they are effective. How is that possible? The point is that we look at the natural numbers as having some structure, usually assuming their standard order. A function over the natural numbers without their order is not really well-defined. As we saw, the halting function and the simple parity function are exactly the same (isomorphic) function when numbers are unordered.

Hence, even when adopting the Church-Turing Thesis, a domain without any structure cannot have an effective representation. It is just a set of arbitrary elements. However, if the domain comes with a generating mechanism (as the natural numbers come with the successor) we can consider effective representations.

Due to the interpretation-completeness of the recursive functions and Turing machines, we can define what is an effective string representation of the natural numbers (with their standard structure). A similar definition can be given for other domains, provided that they come with some finite means of generating them all, akin to successor for the naturals.

Definition 94 (Effective representation)**.** *An* effective representation *of the natural numbers by strings is an injection $\rho : \mathbb{N} \to \Sigma^*$, such that $\rho(s)$ is Turing-computable ($\rho(s) \in [\![\mathrm{TM}]\!]$), where s is the successor function over $\mathbb{N}$.*

That is, a representation of the natural numbers is effective if the successor function is Turing-computable via this representation.

Note 95. *One may also require that the image of the representation ρ is Turing computable, along the decent power comparison notion. In such a case, there would also be a corresponding bijective representation (see Theorem 28 and 32).*

We justify the above definition of an effective representation by showing that: (a) every recursive function is Turing-computable via any effective representation; (b) every non-recursive function is not Turing-computable via any effective representation; and (c) for every non-effective representation there is a recursive function that is not Turing-computable via it.

Theorem 96.

(a) Let f be a recursive function and $\rho : \mathbb{N} \to \Sigma^*$ an effective representation. Then $\rho(f) \in [\![\mathrm{TM}]\!]$.

(b) Let g be a non-recursive function and $\rho : \mathbb{N} \to \Sigma^*$ an effective representation. Then $\rho(g) \notin [\![\mathrm{TM}]\!]$.

(c) Let $\eta : \mathbb{N} \to \Sigma^*$ be a non-effective representation. Then there is a recursive function f, such that $\eta(f) \notin [\![\mathrm{TM}]\!]$.

Proof. Let $\xi : \mathbb{N} \to \Sigma^*$ be some standard bijective representation via which $\xi(\mathbb{REC}) = [\![\mathrm{TM}]\!]$ (see, for example, [26, p. 131]). The point is that, once $\rho(s) \in [\![\mathrm{TM}]\!]$, there are Turing-computable functions for switching between the ρ and the ξ representations. That is, $\rho \circ \xi^{-1}, \xi \circ \rho^{-1} \in [\![\mathrm{TM}]\!]$. It can be done by a Turing machine that enumerates in parallel over both representations until reaching the required string.
(a) Since $f \in \mathbb{REC}$, it follows that there is a function $f' \in [\![\mathrm{TM}]\!]$, such that $f = \xi^{-1}(f') = \xi^{-1} \circ f' \circ \xi$. Thus, $\rho(f) = \rho \circ f \circ \rho^{-1} = \rho \circ \xi^{-1} \circ f' \circ \xi \circ \rho^{-1}$. Hence, $\rho(f) \in [\![\mathrm{TM}]\!]$ by the closure of TM under functional composition.
(b) Assume by contradiction that $g \notin \mathbb{REC}$ but $\rho(g) \in [\![\mathrm{TM}]\!]$. Let g' be the corresponding function under the ξ representation. That is, $g' = \xi \circ \rho^{-1} \rho(g) \circ \rho \circ \xi^{-1}$. We have by the closure of TM under functional composition that $g' \in [\![\mathrm{TM}]\!]$. Since $\xi^{-1}(g') \in \mathbb{REC}$, it is left to show that $\xi^{-1}(g') = g$ for getting a contradiction: $\xi^{-1}(g') = \xi^{-1} \circ g' \circ \xi = \xi^{-1} \circ \xi \circ \rho^{-1} \rho(g) \circ \rho \circ \xi^{-1} \circ \xi = \rho^{-1} \rho(g) \circ \rho = \rho^{-1} \rho \circ g \circ \rho^{-1} \circ \rho = g$.
(c) By the definition of recursive representation, the successor is such a function. □

To see the importance of the interpretation-completeness property for the definition of an effective representation, one can check that an analogous definition cannot be provided with two-counter machines as the yardstick.

Our definition of an effective representation resembles Shapiro's notion of an "acceptable notation". He proposes three necessary "semi-formal" criteria for an acceptable notation [40, p. 19]:

(la) If the computist is given a finite collection of distinct objects, then he can (in principle) write and identify tokens for the numeral which denotes the cardinality of the collection.

(lb) The computist can *count* in the notation. He is able (in principle) to write, in order, tokens for the numerals denoting any finite initial segment of the natural numbers.

(2a) If the computist is given a token for a numeral p and a collection of distinct objects, then he can (in principle) determine whether the denotation of p is smaller than the cardinality of the collection and, if it is, produce a subcollection whose cardinality is the denotation of p.

Our notion coincides with Shapiro's second criterion (1b). It also goes along with Weihrauch's justifications for the effectiveness of the standard "numberings" (representation by natural numbers). He defines a standard numbering of a word set (the words over $\{a, b\}$, for example, are enumerated in the following order: $\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots$), and then proves three claims for justifying the effectiveness of the numbering [50, p. 80–81]:

> A numbering $\nu : \mathbb{N} \to W(\Sigma)$ is neither a word function nor a number function, hence neither of our two definitions of computability is applicable to ν. Nevertheless standard numberings $\nu : \mathbb{N} \to W(\Sigma)$ are intuitively effective. The following lemma expresses several effectivity properties of standard numberings of word sets.
>
> LEMMA (*effectivity of standard numberings of word sets*)
> Let Σ, Γ and Δ be alphabets with $\Delta = \Sigma \cup \Gamma$. Let ν_Σ (ν_Γ) be a standard numbering of $W(\Sigma)$ ($W(\Gamma)$).
>
> (1) Define $S, V : \mathbb{N} \to \mathbb{N}$ by $S(x) := x + 1$, $V(x) := x \dot{-} 1$.
> Define $S_\Sigma, V_\Sigma : W(\Sigma) \to W(\Sigma)$ by
> $S_\Sigma := \nu_\Sigma S \nu_\Sigma^{-1}$, $V_\Sigma := \nu_\Sigma V \nu_\Sigma^{-1}$.
> Then S_Σ and V_Σ are computable.
>
> (2) Let $b \in \Sigma$. Define $h^b : W(\Sigma) \to W(\Sigma)$, $S^b : W(\Sigma) \to \{1, 2\}$ and $pop : W(\Sigma) \to W(\Sigma)$ by
> $h^b(w) = wb$, $S^b(w) :=$ (1 if $w = xb$ for some $x \in W(\Sigma), 2$ otherwise), and $pop(\varepsilon) := \varepsilon$, $pop(wc) := w$.
> Define $h^b_\Sigma : \mathbb{N} \to \mathbb{N}$, $S^b_\Sigma : \mathbb{N} \to \{1, 2\}$ and $pop_\Sigma : \mathbb{N} \to \mathbb{N}$ by
> $h^b_\Sigma := \nu_\Sigma^{-1} h^b \nu_\Sigma$, $pop_\Sigma := \nu_\Sigma^{-1} pop\, \nu_\Sigma$, $S^b_\Sigma := S^b \nu_\Sigma$.
> Then h^b_Σ, pop_Σ, and S^b_Σ are computable.
>
> (3) The following functions $p : W(\Delta) \to W(\Delta)$ and $q : \mathbb{N} \to \mathbb{N}$ are computable:
> $p(w) := (\nu_\Sigma \nu_\Gamma^{-1}(w)$ if $w \in W(\Gamma), \varepsilon$ otherwise),
> $q(j) := (\nu_\Gamma^{-1} \nu_\Sigma(j)$ if $\nu_\Sigma(j) \in W(\Gamma), 0$ otherwise).

Our notion resembles Weihrauch's first and second claims (the second claim concerns the construction of strings, and plays the rôle of the successor when reversing the "numbering" for representing numbers by strings).

6.3 Axiomatization of effective computation

Section 6.1 formalized the Church-Turing Thesis over arbitrary domains. We now provide additional evidence for the thesis by validating it against a class of "effective computational models", axiomatized on top of Gurevich's postulates for a sequential algorithm [24].

Dershowitz and Gurevich [19] have done something quite similar, though handling only the effectiveness of functions over the natural numbers. They do refer to computations over other domains, however only in the process of computing what is a strictly numeric function.

Gurevich's postulates are applicable for computations over any mathematical structure (of first order) and aim to capture any sequential algorithm. This makes them a natural candidate for axiomatizing effectiveness over arbitrary domains. Yet, there are several problems:

1. The postulates concern algorithms and not computations with input and output.

2. Initial states are not limited; thus, they might not be effective.

3. The postulates consider a single algorithm and not an entire computational model.

We address the first issue, in Section 6.3.1, by adding special input and output constants, and allowing a single initial state, up to differences in input. The second issue is addressed by adding Axiom 4, which limits the initial data. The third issue is addressed, in Section 6.3.2, by requiring all functions of the same model to share the same domain representation.

A proof is provided, showing that this axiomatization yields the same definition of effectiveness as the Church-Turing Thesis does. It is based on Gurevich's Abstract State Machine Theorem [24], showing that sequential abstract state machines (ASMs) capture sequential algorithm. As a result, we get three equivalent definitions of an effective computational model over an arbitrary domain. See Figure 6.1.

We start in Section 6.3.1, with an axiomatization of "sequential procedures", along the lines of Gurevich's sequential algorithms [24]. Next, we axiomatize, in Section 6.3.2, "effective procedures" as a subclass, satifying an "effectivity axiom". We then show, in Section 6.3.3, the equivalence of Turing machines to the class of effective models.

6.3.1 Sequential procedures

Our axiomatization of a "sequential procedure" is very similar to that of Gurevich's sequential algorithm [24], with the following two main differences, allowing for the computation of a specific function, rather than expressing an abstract algorithm:

- The vocabulary includes special constants "*In*" and "*Out*".

- Initial states are identical, except for changes in *In*.

Structures The states of a procedure should be a full instantaneous description of all its relevant features. We represent them by (first order) *structures*, using the standard notion of structure from mathematical logic. For convenience, these structures will be *algebras*; that is, having purely functional vocabulary (without relations).

Definition 97 (Structures)**.**

- *A* domain D *is a (nonempty) set of elements.*
- *A* vocabulary $\mathcal{F}$ *is a collection of function names, each with a fixed finite arity.*
- *A* term *of vocabulary* $\mathcal{F}$ *is either a nullary function name (constant) in* $\mathcal{F}$ *or takes the form* $f(t_1, \ldots, t_k)$*, where* f *is a function name in* $\mathcal{F}$ *of positive arity* k *and* $t_1, \ldots, t_k$ *are terms.*
- *A* structure S *of vocabulary* $\mathcal{F}$ *is a domain* D *together with interpretations* $[\![f]\!]_S$ *over* D *of the function names* $f \in \mathcal{F}$*.*
- *A* location *of vocabulary* $\mathcal{F}$ *over a domain* D *is a pair, denoted* $f(\overline{a})$*, where* f *is a* k*-ary function name in* $\mathcal{F}$ *and* $\overline{a}$ *is a* k*-tuple of elements of* D*. (If* f *is a constant, then* $\overline{a}$ *is the empty tuple.)*
- *The* value *of a location* $f(\overline{a})$ *in a structure* S*, denoted* $[\![f(\overline{a})]\!]_S$*, is the domain element* $[\![f]\!]_S(\overline{a})$*.*
- *It is often useful to indicate a location by a (ground) term* $f(t_1, \ldots, t_k)$*, standing for* $f([\![t_1]\!]_S, \ldots, [\![t_k]\!]_S)$*.*
- *Structures* S *and* S' *with vocabulary* $\mathcal{F}$ *sharing the same domain* coincide *over a set* T *of* $\mathcal{F}$*-terms if* $[\![t]\!]_S = [\![t]\!]_{S'}$ *for all terms* $t \in T$*.*

It is convenient to think of a structure S as a memory, or data-storage, of a kind. For example, for storing an (infinite) two dimensional table of integers, we need a structure S over the domain of integers, having a single binary function name f in its vocabulary. Each entry of the table is a location. The location has two indices, i and j, for its row and column in the table, marked $f(i, j)$. The content of an entry (location) in the table is its value $[\![f(i, j)]\!]_S$.

Definition 98 (Update)**.** *An* update *of location* l *over domain* D *is a pair, denoted* $l := v$*, where* v *is an element of* D*.*

Definition 99 (Structure Mapping)**.** *Let S be structure of vocabulary $\mathcal{F}$ over domain D and $\rho : D \to D'$ an injection from D to domain D'. A* mapping *of S by ρ, denoted $\rho(S)$, is a structure S' of vocabulary $\mathcal{F}$ over D', such that $\rho(\llbracket f(\overline{a}) \rrbracket_S) = \llbracket f(\rho(\overline{a})) \rrbracket_{S'}$ for every location $f(\overline{a})$ in S.*

Structures S and S' of the same vocabulary over domains D and D', respectively, are *isomorphic*, denoted $S \cong S'$, if there is a bijection $\pi : D \leftrightarrow D'$, such that $S' = \pi(S)$.

We are now in position to provide the three axioms of a sequential procedure.

Axiom 1 (Sequential Time)**.** *The procedure can be viewed as a collection $\mathcal{S}$ of* states*, a sub-collection $\mathcal{S}_0 \subseteq \mathcal{S}$ of* initial states*, and a* transition function *$\tau : \mathcal{S} \to \mathcal{S}$ from state to state.*

Axiom 2 (Abstract State)**.**

- States*. All states are first-order structures of the same finite* vocabulary *$\mathcal{F}$.*

- Input*. There are nullary function names* In *and* Out *in $\mathcal{F}$. All initial states ($\mathcal{S}_0 \subseteq \mathcal{S}$) share a* domain *$D$, and are equal up to changes in the value of* In*. (For convenience, the initial states can be referred to, collectively, as S_0.)*

- Isomorphism Closure*. The procedure states are closed under isomorphism. That is, if there is a state $S \in \mathcal{S}$, and an isomorphism π via which S is isomorphic to an $\mathcal{F}$-structure S', then S' is also a state in $\mathcal{S}$.*

- Isomorphism Preservation*. The transition function preserves isomorphism. That is, if states S and S' are isomorphic via π, then $\tau(S)$ and $\tau(S')$ are also isomorphic via π.*

- Domain Preservation*. The transition function preserves the domain. That is, the domain of S and $\tau(S)$ is the same for every state $S \in \mathcal{S}$.*

Axiom 3 (Bounded Exploration)**.** *There exists a finite set T of "critical" terms, such that $\Delta(S, \tau(S)) = \Delta(S', \tau(S'))$ if S and S' coincide over T, for all states $S, S' \in \mathcal{S}$, where $\Delta(S, S') = \{l := v' \mid \llbracket l \rrbracket_S \neq \llbracket l \rrbracket_{S'} = v'\}$ is a set of updates turning S into S'.*

The isomorphism constraints reflects the fact that we are working at a fixed level of abstraction. See [24, p. 89]:

> A structure should be seen as a mere representation of its isomorphism type; only the isomorphism type matters. Hence the first of the two statements: distinct isomorphic structures are just different representations of the same isomorphic type, and if one of them is a state of the given algorithm A, then the other should be a state of A as well.

Domain preservation simply ensures that a specific "run" of the procedure is over a specific domain. (Should it be necessary, one could always combine many domains into one.) The bounded-exploration axiom ensures that the behavior of the procedure is effective. This reflects the informal assumption that the program of an algorithm can be given by a finite text [24, p. 90].

Definition 100 (Runs)**.**

1. *A* run *of procedure with transition function τ is a finite or infinite sequence $S_0 \rightsquigarrow_\tau S_1 \rightsquigarrow_\tau S_2 \rightsquigarrow_\tau \cdots$, where S_0 is an initial state and every $S_{i+1} = \tau(S_i)$.*

2. *A run $S_0 \rightsquigarrow_\tau S_1 \rightsquigarrow_\tau S_2 \rightsquigarrow_\tau \cdots$* terminates *if it is finite or if $S_i = S_{i+1}$ from some point on.*

3. *The* terminating state *of a terminating run $S_0 \rightsquigarrow_\tau S_1 \rightsquigarrow_\tau S_2 \rightsquigarrow_\tau \cdots$ is its last state if it is finite, or its stable state if it is infinite.*

4. *If there is a terminating run beginning with state S and terminating in state S', we write $S \rightsquigarrow_\tau^! S'$.*

Definition 101 (Procedure Extensionality)**.** *Let P be sequential procedure over domain D. The* extensionality *of P, denoted $[\![P]\!]$, is the partial function $f : D \to D$, such that $f(x) = [\![Out]\!]_{S'}$ whenever there's a run $S \rightsquigarrow_\tau^! S'$ with $[\![In]\!]_S = x$, and is undefined otherwise.*

Equality, Booleans and undefined. In contradistinction with Gurevich's ASM's, we do not have built in equality, Booleans, or undefined in the definition of procedures. That is, a procedure need not have Boolean values (True, False) or connectives ($\neg, \wedge, \vee$) pre-defined in its vocabulary; rather, they may be defined like any other function. It also should not have a special term for undefined values, though the value of the function implemented by the procedure is not defined when its run doesn't terminate. The equality notion is also not presumed in the procedure's initial state, as it comprises infinite data. Nevertheless, since every domain element has a unique construction, it follows that an effective procedure may implement the equality notion with only finite initial data. A detailed description of this implementation is given in Section 6.3.4.3.

6.3.2 Effective models

A sequential procedure may be equipped with any oracle, given as an operation of the initial state. Hence, the extensionality of such a procedure might not be effective. As a result, we are interested only in sequential procedures that use effective oracles. Since we are defining effectiveness, we get an inductive definition, allowing initial states to include functions that are the extensionality of effective procedures. The starting point must be operations that are very simple and inherently effective. These basic operations must then be finite. We begin, then, with sequential procedures, in which the initial state has finite data in addition to the domain view ("base structure"). This constraint is formalized in Axiom 4, below.

Different procedures of the same computational model have some common mechanism. The level of shared configuration between the model's procedures may vary, but they must obviously share the same domain view. Hence, we define an "effective model" to be some set of "effective procedures" that share the same "base structure".

We formalize the finiteness of the initial data by allowing the initial state to contain an "almost-constant structure".

Definition 102 (Almost-Constant Structure)**.** *A structure F is* almost constant *if all but a finite number of locations have the same value.*

Since we are heading for a characterization of effectiveness, the domain over which the procedure actually operates should have countably many elements, which have to be nameable. Hence, without loss of generality, one may assume that naming is via terms.

Definition 103 (Base Structure)**.** *A structure S of finite vocabulary $\mathcal{F}$ over a domain D is a* base structure *if every domain element is the value of a unique $\mathcal{F}$-term. That is, for every element $e \in D$ there exists a unique $\mathcal{F}$-term t such that $[\![t]\!]_S = e$.*

A base structure is isomorphic to the standard free term algebra (Herbrand universe) of its vocabulary.

Proposition 104. *Let S be a base structure over vocabulary G and domain D, then:*

- *The vocabulary G has at least one nullary function.*

- *The domain D is countable.*
- *Every domain element is the value of a unique location of S.*

Example 105. *A structure over the natural numbers with constant zero and unary function successor, interpreted as the regular successor, is a base structure.*

Example 106. *A structure over binary trees with constant nil and binary function cons, interpreted as in Lisp, is a base structure.*

For allowing the initial state to include both the domain view and the initial data, we need to define the union-operation over structures:

Definition 107 (Structure Union). *Let S' and S'' be two structures with domain D and with vocabularies $\mathcal{F}'$ and $\mathcal{F}''$, respectively. A structure S over D is the* union *of S' and S'', denoted $S = S' \uplus S''$, if its vocabulary is the disjoint union $\mathcal{F} = \mathcal{F}' \uplus \mathcal{F}''$, and if $[\![l]\!]_S = [\![l]\!]_{S'}$ for locations l in S' and $[\![l]\!]_S = [\![l]\!]_{S''}$ for locations in S''.*

We are now in position to formalize the fourth axiom, requiring the effectiveness of the initial state. It is an inductive definition, allowing any function that can be implemented by a (simpler) effective procedure.

Axiom 4 (Initial Data). *The initial state consists of:*

- *a fixed base structure BS (the domain view);*
- *a fixed almost-constant AS structure (finite initial data); and*
- *a fixed effective structure ES over the base structure BS (effective oracles);*

in addition to an input value In *over BS that varies from initial state to initial state. That is, the initial state S_0 is the union $BS \uplus AS \uplus ES \uplus \{\mathrm{In}\}$, for some base structure BS, almost-constant structure AS, and effective structure ES.*

The effective structure contains finite many functions that are the extensionality of effective procedures over the same domain view (the effective oracles). This allows the procedure to use an algorithm at any abstraction level, as long as we can assure that the underlying oracles are effective.

The freedom to add any finite initial data is obvious, but why do we limit the domain view to be isomorphic to a Herbrand universe? There are two limitations here: (a) every domain element has a name (a closed term); and

(b) the name of each element is unique. Were we to allow unnamed domain elements, then a computation could not be referred to, nor repeated, hence would not be effective. As for the uniqueness of the names, allowing a built-in equality notion with an "infinite memory" of equal pairs is obviously non-effective. Hence, the equality notion should be the result of some internal mechanism, one that can be built up from scratch.

An *effective procedure* must satisfy Axioms 1–4.

Definition 108 (Effective Model)**.** *An* effective model *is a set of effective procedures (objects satisfying Axioms 1–4) that share the same base structure.*

To sum up:

> **Thesis B.** *All "effective" computational models are effective models (per Definition 108).*

Necessity. An effective procedure should satisfy, by our definitions, Axioms 1–4. In the beginning of the chapter we argued for the necessity of the postulates from the intuitive point of view of effectiveness. Moreover, omitting any of them allows for models that compute more than Turing machines:

1. The Sequential Time Axiom is necessary if we wish to analyze computation, which is a step-by-step process. Allowing for transfinite computations, for example, would allow a model to precompute all members of a recursively-enumerable set.

2. In the context of effective computation, there is no room for infinitary functions, for example. Without closure under isomorphism there would be no value to the Bounded-Exploration Axiom, allowing the assigning of any desired value to the *Out* location.

3. By omitting the Bounded-Exploration Axiom, a procedure need not have any systematical behavior, hence may "compute" any function by simply assigning the desired value at the *Out* location. That is, for each initial state S there is a state S', such that $\tau(S) = S'$ and $[\![Out]\!]_{S'}$ is the "desired" value.

4. Omitting the Initial-Data Axiom, one may "compute" any function (e.g. a halting oracle), by simply having all its values in the initial state. Such functions could also be encoded in equalities between locations, were the initial data not (isomorphic to) a free term algebra.

6.3.3 Effective equals computable

In the sense of our above definition of effectiveness (Definition 108) we have that:

Theorem 109. *Turing machines are an effective model.*

Furthermore,

Theorem 110. *Turing machines are at least as powerful as any effective model. This is so by all three comparison notions. That is,* $\mathrm{TM} \gtrsim E$ *and* $\mathrm{TM} \succsim E$ *and* $\mathrm{TM} \gtrapprox E$ *for every model* E *satisfying the effectiveness axioms.*

The proofs of Theorems 109 and 110 are somewhat lengthy, so are relegated to Section 6.3.4. They make usage of Abstract State Machines, which operate over arbitrary domains, and are based on Gurevich's Abstract State Machine Theorem [24], showing that sequential abstract state machines (ASMs) capture sequential algorithms, defined axiomatically.

Definition 111 (Effective State Model)**.** *An ASM model satisfying the initial data restrictions is called an* Effective State Model *(or* ESM*).*

This suggests the following variant thesis:

> **Thesis C.** *Every "effective" computational model is behaviorally equivalent to an ESM.*

Adopting the decent-power comparison notion (Definition 52), we get a closer relationship between the three definitions of effectiveness (Theses A–C): When considering only the extensionality of computational models (that is, the set of functions that they compute) we have that the three effectiveness criteria (Theses A–C) are equivalent.

Definition 112 (Extensionally Effective)**.** *A model* A *is* extensionally effective *if the set of functions that it computes may be represented by Turing-computable functions. That is, if* $A \precsim \mathrm{TM}$*.*

Theorem 113. *A model* A *is extensionally effective if and only if there exists an effective model* B*, such that* $[\![A]\!] \subseteq [\![B]\!]$*.*

Proof. If $[\![A]\!] \subseteq [\![B]\!]$ for some effective model B, then obviously $A \precsim B$, and by Theorem 110 we also have that $B \precsim \mathrm{TM}$. Hence, $A \precsim \mathrm{TM}$.

As for the other direction, if A operates over a finite domain then its extensionality is obviously a subset of the extensionality of an effective model.

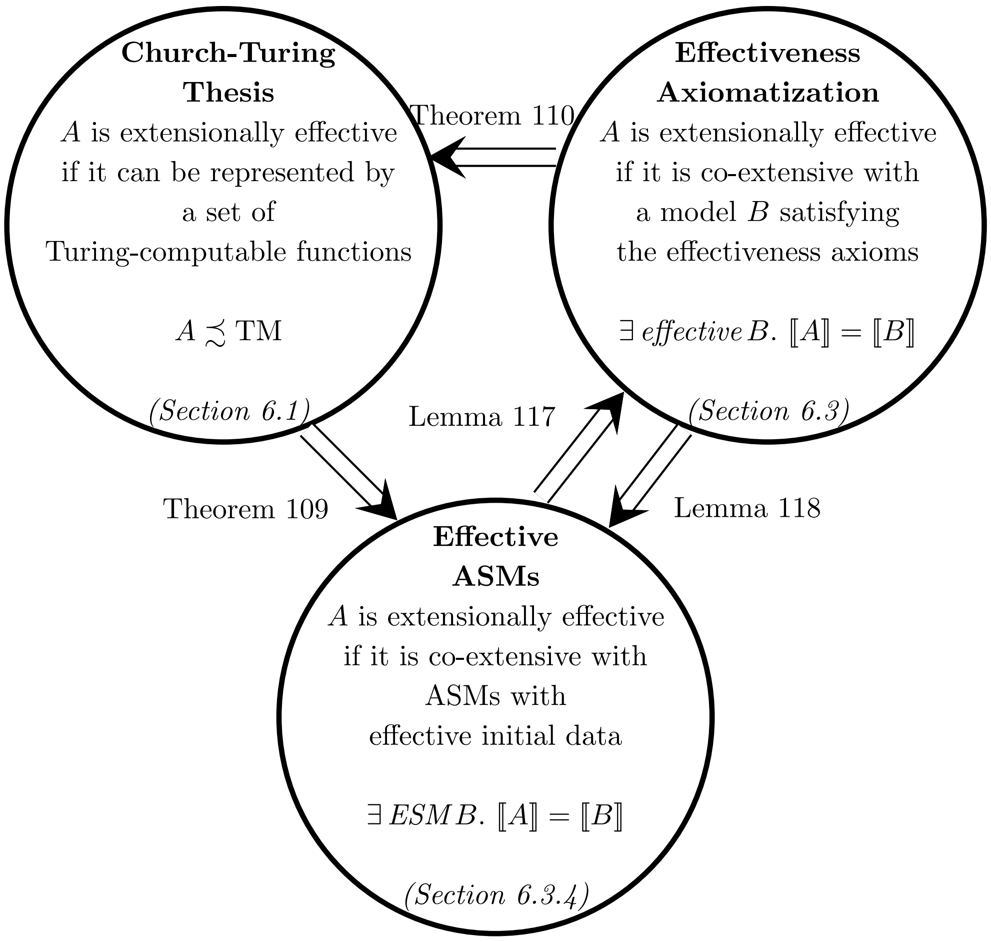

Church-Turing Thesis
A is extensionally effective if it can be represented by a set of Turing-computable functions
$A \precsim \text{TM}$
(Section 6.1)
Effectiveness Axiomatization
A is extensionally effective if it is co-extensive with a model B satisfying the effectiveness axioms
$\exists\, \textit{effective}\, B.\ [\![A]\!] = [\![B]\!]$
(Section 6.3)
Effective ASMs
A is extensionally effective if it is co-extensive with ASMs with effective initial data
$\exists\, \textit{ESM}\, B.\ [\![A]\!] = [\![B]\!]$
(Section 6.3.4)
Theorem 110
Theorem 109
Lemma 117
Lemma 118

Figure 6.1: Equivalent characterizations of *extensional effectiveness* of a computational model over an arbitrary domain

Otherwise, since $A \precsim \text{TM}$, it follows from Theorem 70 that there is a bijection π such that $A \lessapprox_\pi \text{TM}$. Hence, $[\![A]\!] \subseteq [\![\text{TM}]\!]_\pi$. Since effective models are closed under isomorphism, it follows that $[\![\text{TM}]\!]_\pi$ is an effective model, as required. □

Thanks to Gurevich's Abstract State Machines Theorem [24], we have the analogous theorem with respect to ASMs:

Theorem 114. *A model A is extensionally effective if and only if there is an ESM B, such that $[\![A]\!] = [\![B]\!]$.*

The resulting relationship between the different characterizations of effectiveness is depicted in Figure 6.1.

6.3.4 Proofs of two theorems

We provide here proofs of Theorems 109 and 110. First, we require some additional definitions and lemmata.

6.3.4.1 Programmable machines

In Section 6.3.1, we axiomatized sequential procedures. To link these procedures with Turing machines, we define some mediators, named "programmable procedures," along the lines of Gurevich's Abstract State Machines (ASMs) [24]. We then show that sequential procedures and programmable procedures are equivalent (Lemma 117).

A "programmable procedure" is like a sequential procedure, with the main difference that its transition function should be given by a finite "flat program" rather than satisfy some constraints.

Definition 115 (Flat Program)**.** *A* flat program *P of vocabulary $\mathcal{F}$ has the following syntax:*

`if` $x_{11} \doteq y_{11}$ `and` $x_{12} \doteq y_{12}$ `and` $\ldots$ $x_{1k_1} \doteq y_{1k_1}$ `then` $l_1 := v_1$

`if` $x_{21} \doteq y_{21}$ `and` $x_{22} \doteq y_{22}$ `and` $\ldots$ $x_{2k_2} \doteq y_{2k_2}$ `then` $l_2 := v_2$

$\vdots$

`if` $x_{n1} \doteq y_{n1}$ `and` $x_{n2} \doteq y_{n2}$ `and` $\ldots$ $x_{nk_n} \doteq y_{nk_n}$ `then` $l_n := v_n$
where each $\doteq$ is either '$=$' or '$\neq$', $n, k_1, \ldots, k_n \in \mathbb{N}$, and all the x_{ij}, y_{ij}, l_i, and v_i are $\mathcal{F}$-terms.

Each line of the program is called a rule*. The part of a rule between the* `if` *and the* `then` *is the* condition*, l_i is its location, and v_i is its value.*

The activation *of a flat program P on an $\mathcal{F}$-structure S, denoted $P(S)$, is a set of updates $\{l := v \mid$ there is a rule in P, whose condition holds (under the standard interpretation), with location l and value $v\}$, or the empty set if the above set includes two values for the same location.*

Coding Style. To make flat programs more readable, let

```
% comment
if cond-1
   stat-1
   stat-2
else
   stat-3
```

stand for

```
if cond-1 then stat-1
if cond-1 then stat-2
if not cond-1 then stat-3
```

and, similarly, for other such abbreviations.

Definition 116 (Programmable Procedure)**.** *A* programmable procedure *is composed of: $\mathcal{F}, \mathit{In}, \mathit{Out}, D, \mathcal{S}, \mathcal{S}_0$, and P, where all but the last component is as in a sequential procedure (see Section 6.3.1), and P is a flat program of $\mathcal{F}$.*

The run *of a programmable procedure and its* extensionality *are defined as for sequential procedures (Definitions 100 and 101), where the transition function τ is given by $\tau(S) = S' \in \mathcal{S}$ such that $\Delta(S, S') = P(S)$.*

6.3.4.2 Sequential equals programmable

We show that every programmable procedure is sequential (satisfying the three axioms), and every sequential procedure is programmable. This result is derived directly from the main lemma of [24].

Lemma 117. *Every programmable procedure is sequential. That is, let A be a programmable procedure with states $\mathcal{S}$ and a flat program P, then there exists a sequential procedure B with the same elements of A, except for having a transition function τ instead of the program P, such that $\Delta(S, \tau(S)) = P(S)$ for every $S \in \mathcal{S}$.*

Proof. Let $A = \langle \mathcal{F}, \mathit{In}, \mathit{Out}, D, \mathcal{S}, \mathcal{S}_0, P\rangle$ be an arbitrary programmable procedure. Define the finite set of critical $\mathcal{F}$-terms T to include all terms and subterms of P. Define a transition function $\tau : \mathcal{S} \to \mathcal{S}$ by $\tau(S) = S'$ such that $\Delta(S, S') = P(S)$. To show that $B = \langle \mathcal{F}, \mathit{In}, \mathit{Out}, D, \mathcal{S}, \mathcal{S}_0, \tau\rangle$ is a sequential procedure such that $\Delta(S, \tau(S)) = P(S)$ for every $S \in \mathcal{S}$ it remains to show that B satisfies the constraints defined for τ in a sequential procedure. Since the flat program P includes only terms in T (and does not refer directly to domain elements), it obviously follows that τ satisfies the isomorphism constraint. Since T includes all the terms of P, as well as the subterms of the location-terms of P, it obviously follows that states that coincide over T have the same set of updates by τ. Thus, τ satisfies the bounded-exploration constraint. $\square$

Lemma 118. *Every sequential procedure is programmable. That is, let B be a sequential procedure with states $\mathcal{S}$ and a transition function τ, then there exists a programmable procedure A with the same elements of B, except for* *having a flat program P instead of τ, such that $\Delta(S, \tau(S)) = P(S)$ for every $S \in \mathcal{S}$.*

This follows directly from Gurevich's proof that for every sequential algorithm there exists an equivalent sequential abstract state machine [24, Lemma 6.11].

6.3.4.3 Effective equals computable

We prove now that Turing machines are of equivalent computational power to all effective models.

Turing Machines are Effective. First, we show that the class of effective procedures is at least as powerful as Turing machines, as the latter is an effective model.

Proof (of Theorem 109). We consider Turing machines with two-way infinite tapes. By way of example, let the tape alphabet be $\{0, 1\}$.

A Turing machine state (instantaneous description) contains three things: *Left*, a finite string containing the tape section left of the reading head; *Right*, a finite string with the tape section to the right to the read head; and q, the internal state of the machine. The read head points to the first character of *Right.*

Turing machines can be viewed as an effective model with the following components:

Domain: The domain consists of all finite strings over $0, 1$. That is the domain $D = \{0, 1\}^*$.

Base structure: Constructors for the finite strings: the constant symbol @ for the empty string (ε), and unary function symbols *Con*_0 and *Con*_1, for concatenating 0 and 1, respectively, to the right of the string.

Almost-constant structure:

- Input and Output (nullary functions): *In*, *Out*. The value of *In* at the initial state is the content of the tape, as a string over $\{0, 1\}^*$.
- Constants for the alphabet characters and TM-states (nullary): 0, 1, q_0, q_1, …, q_k. Their actual values are of no significance, as long as they are all different.

- Variables to keep the current status of the Turing machine (nullary): *Left*, *Right*, and q. Their initial values are: $\mathit{Left} = \varepsilon$, $\mathit{Right} = \varepsilon$, and $q = q\_0$.

Effective structure:

- Functions to examine the tape (unary functions): *Head* and *Tail*. Their initial values are as in the standard implementation of *Head* and *Tail*. Their effective implementation is given below, after the description of the Turing machine model.
- The Boolean equality notion =. Note that the standard equality notion contains infinite data, thus cannot be contained in the almost-constant structure, nor in the base structure. Nevertheless, since every domain element has a unique construction, the equality notion can be effectively implemented with only finite initial data. This implementation is explained after the implementation of *Head* and *Tail*.

Transition function: By Lemma 117, every programmable procedure is a sequential procedure. Thus, a programmable procedure that satisfies the initial-data postulate is an effective procedure. Every Turing machine is an effective procedure with a flat program looking like this (% is used for explanatory comments):

```
if q = q_0  % TM's state q_0
   if Head(Right) = 0
      % write 1, move right, switch to q_3
      Left := Con_1(Left)
      Right := Tail(Right)
      q := q_3
   if Head(Right) = 1
      % write 0, move left, switch to q_1
      Left := Tail(Left)
      Right := Con_0(Right)
      q := q_1
   if Right = @
      % write 0, move left, switch to q_2
      Left := Tail(Left)
      Right := Con_0(Right)
      q := q_2
if q = q_1   % TM's state q_1
        ...
if q = q_k   % the halting state
   Out := Right
```

In the above description of Turing machines as an effective model we've used the functions *Head* and *Tail*. We show now their effectiveness.

The implementation sequentially enumerates all strings, assigning their *Head* and *Tail* values, until encountering the input string. Note that it uses the equality notion, which is shown to be effective afterwards.

It uses the same base structure and almost-constant structure described above, with the addition of the following nullary functions (Name = initial value): `New` $= \varepsilon$, `Backward` $= 0$, `Forward` $= 1$, `AddDigit` $= 0$, and `Direction` $= \varepsilon$.

```
% Sequentially constructing the Left variable
% until it equals to the input In, for filling
% the values of Head and Tail.
% The enumeration is: empty string, 0, 1, 00, 01, ...
if Left = In % Finished
   Right := Left
   Left := @
else % Keep enumerating
   if Direction = New % default val
      if Left = @ % @ -> 0
         Left := Con_0(Left)
         Head(Con_0(Left)) := 0
         Tail(Con_0(Left)) := Left
      if Head(Left) = 0 % e.g. 110 -> 111
         Left := Con_1(Tail(Left))
         Head(Con_1(Tail(Left)) := 1
         Tail(Con_1(Tail(Left)) := Tail(Left)
      if Head(Left) = 1 % 01->10; 11->000
         Direction := Backward
         Left := Tail(Left)
         Right := Con_0(Right)
   if Direction = Backward
      if Left = @ % add rightmost digit
         Direction := Forward
         AddDigit := True
      if Head(Left) = 0 % change to 1
         Left := Con_1(Tail(Left))
         Direction := Forward
      if Head(Left) = 1 % keep backwards
         Left := Tail(Left)
         Right := Con_0(Right)
   if Direction = Forward % Gather right 0s
      if Right = @ % finished gathering
         Direction := New
```

```
     if AddDigit = 1
        Left := Con_0(Left)
        Head(Con_0(Left)) := 0
        Tail(Con_0(Left)) := Left
        AddDigit = 0
  else
        Left := Con_0(Left)
        Right := Tail(Right)
        Head(Con_0(Left)) := 0
        Tail(Con_0(Left)) := Left
```

The equality notion. The standard equality notion has infinite data, thus cannot be given in the initial state. However, since the domain elements are uniquely constructed, it follows that it can be effectively implemented using only finite initial data. The implementation scheme is quite similar to the above implementation of *Head* and *Tail*. Initially, the value of the equality function is $\perp$ at all locations. The implementation sequentially enumerates the strings, assigning *True* as the value of equality of each string with itself and *False* for comparisons with all preceeding strings. This continues until the process gives one of the defined values to the equality operation applied to the inputs. $\square$

Effective procedures are computable. Next, we show that all effective models are equal to or weaker than Turing machines by mapping every effective model to a **while**-like computer program (CP). The computer program may be of any programming language known to be of equivalent power to Turing machines, as long as it operates over the natural numbers and includes the syntax and semantics of flat programs.

Lemma 119. *Every infinite base structure S of vocabulary $\mathcal{F}$ over a domain D is isomorphic to a computable structure S' of the same vocabulary over $\mathbb{N}$. That is, there is a bijection $\pi : D \leftrightarrow \mathbb{N}$ such that for every location $f(\overline{a})$ of S we have that $[\![f(\overline{a})]\!]_S = \pi^{-1}([\![f(\pi(\overline{a}))]\!]_{S'})$.*

Proof. Let S be a base structure of vocabulary $\mathcal{F}$ over a domain D. Let $\mathcal{T}$ be the domain of all $\mathcal{F}$-terms, and $\tilde{S}$ the standard free term algebra (structure) of $\mathcal{F}$. Since all structure functions are total, it follows that every $\mathcal{F}$-term has a value in D, and by Proposition 104, every element $e \in D$ is the value of a unique $\mathcal{F}$-term. Therefore, there is bijection $\varphi : D \leftrightarrow \mathcal{T}$, such that $\varphi^{-1}(t) = [\![t]\!]_S$ for every $t \in \mathcal{T}$. Hence, S and $\tilde{S}$ are isomorphic via φ. Since $\mathcal{F}$ is finite, it follows that its set of terms $\mathcal{T}$ is recursive. Define a computable enumeration $\eta : \mathcal{T} \leftrightarrow \mathbb{N}$. Define a structure S' of vocabulary $\mathcal{F}$ over $\mathbb{N}$ by the following computable recursion: $[\![f(n_1, \ldots, n_k)]\!]_{S'} = \eta(f(\eta^{-1}(n_1), \ldots, \eta^{-1}(n_k)))$. That is, for computing the value of a function f on a tuple $\overline{n}$ the program should recursively find the terms of $\overline{n}$, and then compute the enumeration of the combined term. By the construction of S' we have that S' and $\tilde{S}$ are isomorphic via η. Hence, S' and S are isomorphic via $\varphi \circ \eta$. $\square$

Lemma 120. *Computer programs* (CP) *are bijectively at least as powerful as any effective model.*

Proof. We show that for every effective model E over domain D there is a bijection $\pi : D \to \mathbb{N}$ such that CP $\gtrapprox_\pi E$.

When the effective model E has a finite base structure, then the computability is obvious due to the finite number of possible procedures. We consider then the infinite case; let E be an effective model over a domain D with an infinite base structure BS. By Lemma 119 there is a bijection $\pi : D \leftrightarrow \mathbb{N}$, such that the structure $BS' := \pi(BS)$ is computable. Let P_{BS} be a computer program implementing BS'. For each effective procedure $e \in E$, let AS_e be its almost-constant structure. Since AS_e is almost constant, it follows that $AS'_e := \pi(AS_e)$ is computable; let P_{AS_e} be a computer program implementing AS'_e. Analogously, we have by induction a computer program P_{ES_e} implementing the effective structure of e. By Lemma 118, the transition function of every effective procedure $e \in E$ can be defined by a flat program P_e. For every effective procedure $e \in E$, define a computer program $P'_e = P_{BS} \cup P_e \cup P_{AS_e} \cup P_{ES_e}$. Since $BS' = \pi(BS)$, $AS'_e = \pi(AS_e)$ and $ES'_e = \pi(ES_e)$, it follows that $[\![P'_e]\!] = \pi([\![e]\!])$. Therefore, there is a bijection $\pi : D \leftrightarrow \mathbb{N}$, such that for every effective procedure $e \in E$ there is a computer program $P'_e \in$ CP such that $[\![P'_e]\!] = \pi([\![e]\!])$. Hence, CP $\gtrapprox E$. $\square$

We are now in position to prove that Turing machines are at least as powerful as any effective model.

Proof (of Theorem 110). By Lemma 120, computer programs (CP) are bijectively at least as powerful as any effective model, while Turing machines (TM) are bijectively power equivalent to computer programs. (There are standard bijections between Σ^* and $\mathbb{N}$.) $\square$

Chapter 7

Discussion

Key points. In this thesis, we have explored various aspects of the influence of domain interpretations on the extensionality of computational models, and suggested how we believe they should be handled. Some of the key points are:

- The extensionality of computational models is a very interesting set (of functions) – it varies from "fluid" sets for which containment does not mean more power, as with ordinary sets, to explicit, complete, sets for which nothing can be added without additional power.
- Get to know your model: Is it interpretation-stable or interpretation-complete?
- Compare computational power properly.
- Turing machines and the recursive functions are shown to be robust, this time from the representation/interpretation point of view.
- Effectiveness does not apply to a single function; it is a set of functions, or of a function together with its domain constructors, that may be deemed effective.

Domain Representation. One might think that our study is a part of the "domain representation" research area. This is not the case.

Indeed, the basic concept of "representation" is the same: one set of elements is represented by a subset of another set. Nevertheless, the issues studied are very different. The area of domain representation concerns mapping of some (possibly topological/metric) sets into *domains* (partially ordered sets with some properties), and studies the (topological/metric) properties preserved via the mappings.

For example, in [4], a "topological space" is represented by a "domain" via a "quotient map". A *topological space* is a set together with a collection of its open subsets. A *domain* is a partially ordered set satisfying some properties as a least element, a least upper bound to every two elements, etc.... A domain may be equipped with some topology, normally a Scott topology. A *quotient map* is a continuous function between topological spaces that obeys some rules of open-set preservation.

Further research. A central issue, yet to be understood, is the relation between the internal mechanism of a computational model and its extensional properties of interpretation-completeness and interpretation-stability. With the recursive functions, this relation is apparent, shown in the proof of their interpretation-completeness. However, we do not know how to relate, in general, the completeness or stability of a computational model to its internal mechanism. Another related question is whether there is some standard, well known, computational model that is not interpretation-stable. Specifically, we do not know yet whether the primitive recursive are interpretation-stable.

Bibliography

[1] J. Adámek, H. Herrlicha, and G. E. Strecker. *Abstract and Concrete Categories The Joy of Cats*. 2005. Available at: `http://www.math.uni-bremen.de/~dmb/acc.pdf`.

[2] J. Barzdins. About one class of Turing machines (Minsky machines). *Algebra and Logic (seminar)*, 1(6):42–51, 1962. In Russian.

[3] A. Ben-Amram. *Algorithms and Programs. An Introduction to the Theory of Programming Languages, Computability and Complexity.* The Academic College of Tel-Aviv Yaffo, 2002. In Hebrew.

[4] J. Blanck. Domain representations of topological spaces. *Theoretical Computer Science*, 247(1-2):229–255, 2000.

[5] A. Blass and Y. Gurevich. Ordinary interactive small-step algorithms, I. *ACM Trans. Comput. Log.*, 7(2):363–419, 2006.

[6] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer-Verlag, New York, 1998.

[7] U. Boker and N. Dershowitz. The influence of domain interpretations on computational models. *Applied Mathematics and Computation.* A special issue on Physics and Computation, to appear in 2009.

[8] U. Boker and N. Dershowitz. How to compare the power of computational models. In *Computability in Europe 2005: New Computational Paradigms (Amsterdam), S. Barry Cooper, Benedikt Löwe, Leen Torenvliet, eds.*, volume 3526 of *Lecture Notes in Computer Science*, pages 54–64, Berlin, Germany, 2005. Springer-Verlag.

[9] U. Boker and N. Dershowitz. Abstract effective models. In *New Developments in Computational Models: Proceedings of the First International Workshop on Developments in Computational Models (DCM 2005), Lisbon, Portugal (July 2005), M. Fernández and I. Mackie, eds.*,

volume 135 of *Electronic Notes in Theoretical Computer Science*, pages 15–23, 2006.

[10] U. Boker and N. Dershowitz. Comparing computational power. *Logic Journal of the IGPL*, 14(5):633–648, 2006.

[11] U. Boker and N. Dershowitz. A hypercomputational alien. *Applied Mathematics and Computation*, 178(1):44–57, 2006.

[12] U. Boker and N. Dershowitz. The Church-Turing thesis over arbitrary domains. In A. Avron, N. Dershowitz, and A. Rabinovich, editors, *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 199–229. Springer, 2008.

[13] O. Bournez, F. Cucker, P. J. de Naurois, and J.-Y. Marion. Computability over an arbitrary structure. sequential and parallel polynomial time. *FoSSaCS*, pages 185–199, 2003.

[14] J. P. Bowen. Glossary of Z notation. *Information and Software Technology*, 37(5–6):333–334, May–June 1995. Available at: `http://staff.washington.edu/~jon/z/glossary.html`.

[15] M. L. Campagnolo, C. Moore, and J. F. Costa. Iteration, inequalities, and differentiability in analog computers. *Journal of Complexity*, 16:642–660, 2000.

[16] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[17] N. Cutland. *Computability: An Introduction to Recursive Function Theory.* Cambridge University Press, Cambridge, 1980.

[18] M. Davis. Why Gödel didn't have Church's Thesis. *Information and Control*, 54(1/2):3–24, 1982.

[19] N. Dershowitz and Y. Gurevich. A natural axiomatization of computability and proof of Church's Thesis. *Bulletin of Symbolic Logic*, 14(3):299–350, 2008.

[20] E. Engeler. *Formal Languages: Automata and Structures.* Lectures in Advanced Mathematics. Markham Publishing Company, Chicago, IL, 1968.

[21] A. Fröhlich and J. C. Shepherdson. Effective procedures in field theory. *Philosophical Transactions of the Royal Society of London*, 248:407–432, 1956.

[22] R. Gandy. Church's thesis and principles for mechanisms. In *The Kleene Symposium, J. Barwise, D. Kaplan, H. J. Keisler, P. Suppes, A. S. Troelstra, eds.*, volume 101 of *Studies in Logic and The Foundations of Mathematics*, pages 123–148. North-Holland, 1980.

[23] D. Q. Goldin, S. A. Smolka, and P. Wegner, editors. *Interactive computation: the new paradigm.* Springer, Berlin ; New York, 2006.

[24] Y. Gurevich. Sequential Abstract State Machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1:77–111, 2000.

[25] F. Hennie. *Introduction to Computability.* Addison-Wesley, Reading, MA, 1977.

[26] N. D. Jones. *Computability and Complexity from a Programming Perspective.* The MIT Press, Cambridge, Massachusetts, 1997.

[27] S. C. Kleene. Recursive predicates and quantifiers. *Transactions of the American Mathematical Society*, 53(1):41–73, 1943.

[28] W. M. Lambert, Jr. A notion of effectiveness in arbitrary structures. *The Journal of Symbolic Logic*, 33(4):577–602, 1968.

[29] A. Mal'tsev. Constructive algebras I. *Russian Mathematical Surveys*, 16:77–129, 1961.

[30] M. L. Minsky. Matter, mind and models. *Proc. International Federation of Information Processing Congress*, 1:45–49, 1965. Available at `http://web.media.mit.edu/~minsky/papers/MatterMindModels.html`.

[31] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, N.J., 1967.

[32] R. Montague. Towards a general theory of computability. *Synthese*, 12(4):429–438, 1960.

[33] J. Mycka and J. F. Costa. Real recursive functions and their hierarchy. *Journal of Complexity*, 20(6):835–857, 2004.

[34] J. Myhill. Some philosophical implications of mathematical logic. Three classes of ideas. *The Review of Metaphysics*, 6(2):165–198, 1952.

[35] M. O. Rabin. Computable algebra, general theory and theory of computable fields. *Transactions of the American Mathematical Society*, 95(2):341–360, 1960.

[36] M. Rescorla. Church's thesis and the conceptual analysis of computability. *Notre Dame Journal of Formal Logic*, 48(2):253–280, 2007.

[37] H. G. Rice. Recursive and recursively enumerable orders. *Transactions of the American Mathematical Society*, 83(2):277–300, 1956.

[38] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1966.

[39] R. Schroeppel. A two counter machine cannot calculate 2^{N}. Technical report, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1972. available at: `ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-257.pdf`.

[40] S. Shapiro. Acceptable notation. *Notre Dame Journal of Formal Logic*, 23(1):14–20, 1982.

[41] J. R. Shoenfield. *Recursion Theory*, volume 1 of *Lecture Notes In Logic*. Springer-Verlag, Heidelberg, New York, 1991.

[42] W. Sieg. Church without dogma: Axioms for computability. In *New Computational Paradigms: Changing Conceptions of What is Computable, S. Barry Cooper, Benedikt Löwe and Andrea Sorbi, eds.*, pages 139–152. Springer New York, 2007.

[43] W. Sieg and J. Byrnes. An abstract model for parallel computations: Gandy's thesis. *The Monist*, 82(1):150–164, 1999.

[44] R. Sommerhalder and S. C. van Westrhenen. *The Theory of Computability: Programs, Machines, Effectiveness and Feasibility.* Addison-Wesley, Workingham, England, 1988.

[45] G. J. Tourlakis. *Computability.* Reston Publishing Company, Reston, VA, 1984.

[46] B. A. Trakhtenbrot. Comparing the Church and Turing approaches: Two prophetical messages. In *The Universal Turing Machine: A half-century survey, R. Herken, ed.*, pages 603–630, Oxford, 1988. Oxford University Press.

[47] J. V. Tucker and J. I. Zucker. Abstract versus concrete computation on metric partial algebras. *ACM Transactions on Computational Logic*, 5(4):611–668, 2004.

[48] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936–37. Corrections in vol. 43 (1937), pp. 544-546. Reprinted in M. Davis (ed.), *The Undecidable*, Raven Press, Hewlett, NY, 1965. Available at: `http://www.abelard.org/turpap2/tp2-ie.asp`.

[49] A. M. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, 45:161–228, 1939.

[50] K. Weihrauch. *Computability*, volume 9 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1987.

[51] K. Weihrauch. *Computable Analysis — An introduction.* Springer-Verlag, Berlin, 2000.