Prerequisites 2

2.1 Computability theory 9
2.2 Reverse mathematics 12
2.3 Effective forcing 14

This textbook *is not* an introduction to computability theory or to reverse mathematics. The reader is assumed to have attended at least a first course in computability theory, and have a general background in mathematical logics, especially first-order logic and forcing. This chapter will recall basic facts of common knowledge, for the sake of self-containment and mostly to fix notation.

This book is a pedagogical resource to learn some specific techniques for computability-theoretic analysis for combinatorial theorems. It tries to bridge the gap between introductory textbooks in computability theory, and research articles on the field. The emphasis is put on the intellectual process of research rather than the actual theorems and end-results.

Where to learn computability theory? There are many books about computability theory. Cooper [2] is probably the most accessible resource for a first introduction to the subject. Soare [3] is a good alternative, although slightly more technical. Monin and Patey [4] provides a general overview of both computability theory and reverse mathematics.

Where to learn reverse mathematics? The field being younger, there are only a few options to learn reverse mathematics. The historical book is Simpson [5], is still a good reference, but its very formal style might be off-putting. A first reader might prefer Dzhafarov and Mummert [6] or Monin and Patey [4] as a gentle introduction. Hirschfeldt [7] monograph is also a good starting point for a reader familiar with computability theory.

2.1 Computability theory

Computability theory is essentially the study of mathematical objects or processes from a computational perspective. It has a primary focus on the structure of the degrees of computation, known as *Turing degrees*.

Definition 2.1.1. Fix a reasonable programming language. A set $X \subseteq \mathbb{N}$ is *computable*[1] if there is an algorithm which, on input $n \in \mathbb{N}$, decides whether n belongs to X or not. ◇

1: Computability theory used to be called Recursion theory. Some literature might use *recursive* for computable and *recursively enumerable* for computably enumerable.

All mainstream programming languages are mutually interpretable, thus the notion of computable set is robust. Moreover, by the Church-Turing thesis, this captures the informal notion of *effectively computable* set. One of the main features of models of computation is their relativization to *oracles*. A set X is Y*-computable* or *Turing reducible to* Y (written $X \leq_T Y$) if it is computable in a programming language enriched with the characteristic function of Y as a primitive.

We write $\Phi_0^Y, \Phi_1^Y, \Phi_2^Y, \dots$ for an effective listing of all programs[2] with oracle Y. The notation $\Phi_e^Y(x)\downarrow= v$ means that the eth program with oracle Y halts on input x and outputs v. If the program does not halt, we write $\Phi_e^Y(x)\uparrow$. Similarly, the notation $\Phi_e^Y(x)[s]\downarrow= v$ means that $\Phi_e^Y(x)\downarrow= v$ in at most s steps of computation. By convention, if $\Phi_e^Y(x)[s]\downarrow= v$, then $v, x < s$. Otherwise,

2: Depending on the context, we may furthermore assume that the programs are $\{0,1\}$-valued, or satisfy some additional decidable structural properties.

$\Phi_e^Y(x)[s]\uparrow$. We may further abstract oracle programs, and consider them as *Turing functionals* from $2^{\mathbb{N}}$ to $2^{\mathbb{N}}$, defined by $Y \mapsto \Phi_e^Y$. We then use $\Phi_0, \Phi_1, \Phi_2, \ldots$ as an effective listing of all Turing functionals.

3: We write $2^{<\mathbb{N}}$ for the set of all finite binary strings. Elements of $2^{<\mathbb{N}}$ are written with small greek letters $\sigma, \tau, \rho, \ldots$. We denote by $|\sigma|$ the *length* of the string σ and write $\sigma \preceq \tau$ if σ is a prefix of τ.

Whenever a program halts, it halts on finite time, and thus with finitely many calls to its oracle. Thus, if $\Phi_e^Y(x)\downarrow$, not only there is some $s \in \mathbb{N}$ such that $\Phi_e^Y(x)[s]\downarrow$, but furthermore there is a shortest initial segment $\sigma \prec Y$ such $\Phi_e^Z(x)[s]\downarrow = \Phi_e^Y(x)$ for every $Z \succ \sigma$. This finite binary string[3] σ is called the *use* of the computation. From a topological viewpoint, this means that Turing functionals are partial continuous functions over the Cantor space[4] $2^{\mathbb{N}}$. We extend Turing functionals to partial oracles, and write $\Phi_e^\sigma(x)\downarrow = v$ to say that the eth program with oracle σ halts on input x and outputs v in less than $|\sigma|$ steps, whose only calls to the oracle are within its domain of definition.

4: We write $2^{\mathbb{N}}$ for the class of all infinite binary sequences, also known as Cantor space. It is in one-to-one correspondence with the class of sets of integers, seeing an infinite binary sequence as the characteristic function of a set of integers. We shall therefore identify the two notions and write indistinctly $X \in 2^{\mathbb{N}}$ and $X \subseteq \mathbb{N}$.

2.1.1 Turing degree

Sets of integers are not the appropriate notion to capture the notion of *computational power*. For instance, if X equals Y up to finite changes, or if we let $Y = \{2n : n \in X\}$, then X and Y are mutually computable. The Turing reduction $\leq_T$ is a pre-order on $2^{\mathbb{N}}$. It induces an equivalence relation defined by $X \equiv_T Y$ iff $X \leq_T Y$ and $Y \leq_T X$.

Definition 2.1.2. A *Turing degree* is an equivalence class over $2^{\mathbb{N}}/\equiv_T$. ◇

We write $\deg_T(X) = \{Y \in 2^{\mathbb{N}} : X \equiv_T Y\}$ for the Turing degree of X. The Turing reduction naturally extends to the Turing degrees. The Turing degrees[5] $(\mathcal{D}, \leq)$ form an upper semilattice, with join $\deg_T(X) \cup \deg_T(Y) = \deg_T(X \oplus Y)$, where $X \oplus Y = \{2n : n \in X\} \cup \{2n+1 : n \in Y\}$. The Turing degree $\mathbf{0}$ of the computable sets is the smallest element of this semilattice.

5: We write $\leq_T$ for the Turing reduction over sets, and $\leq$ for the reduction over Turing degrees. We use small boldface letters $\mathbf{a}, \mathbf{b}, \ldots$ to denote Turing degrees.

The *Turing jump* of a set X is the set $X' = \{e : \Phi_e^X(e)\downarrow\}$. The operator $X \mapsto X'$ is Turing-invariant, and therefore induces an operation $\mathbf{a} \mapsto \mathbf{a}'$ over the Turing degrees. By the undecidability of the halting set, $\mathbf{a} < \mathbf{a}'$ for every Turing degree $\mathbf{a}$. The Turing jump can be iterated as follows: $\mathbf{a}^{(0)} = \mathbf{a}$, and $\mathbf{a}^{(n+1)} = (\mathbf{a}^{(n)})'$. Any Turing degree $\mathbf{a}$ such that $\mathbf{a}' = \mathbf{0}'$ is *low*, and the degrees $\mathbf{b}$ such that $\mathbf{b}' \geq \mathbf{0}''$ are *high*.[6]

6: High degrees used to be defined as $\mathbf{b} \leq \mathbf{0}'$ and $\mathbf{b}' \geq \mathbf{0}''$. Indeed, $\mathbf{0}'$ and $\mathbf{0}''$ are respectively the lowest and the highest value that can take the jump of a degree $\mathbf{d} \leq \mathbf{0}'$, so low and high degrees where Turing degrees at these extremes.

2.1.2 Arithmetic hierarchy

Arithmetically definable sets of integers can be classified based on alternations of quantifiers.

Definition 2.1.3. For $n \geq 1$, a set X is Σ_n^0 if it can be written of the form

$$\{x \in \mathbb{N} : \exists y_1 \forall y_2 \ldots Q y_n \; P(x, y_1, \ldots, y_n)\}$$

where P is a computable predicate, and $Q = \forall$ if n even and $Q = \exists$ if n is odd. Π_n^0 sets are defined accordingly by starting with a universal quantifier. A set is Δ_n^0 if it is both Σ_n^0 and Π_n^0. ◇

By Post theorem, there is a correspondence between definability and computability. The Δ_1^0 sets are precisely the computable sets, and the Σ_1^0 sets are the *computably enumerable* (c.e.) ones, that is, sets of the form $W_e = \operatorname{dom} \Phi_e$

for some $e \in \mathbb{N}$. We write $W_0, W_1, \dots$ for an effective enumeration of the c.e. sets. More generally, the hierarchy can be relativized to any oracle Y by considering Y-computable predicates P. A set is $\Delta^0_n(Y)$ iff it is $Y^{(n-1)}$-computable, and $\Sigma^0_n(Y)$ if it is $Y^{(n-1)}$-c.e.[7]

A c.e. set X can be approximated by an uniformly computable sequence of increasing sets $X_0 \subseteq X_1 \subseteq X_2 \subseteq \dots$ with $X = \bigcup_s X_s$. Such a sequence is a called a *c.e. approximation* of X. Indeed, if $X = \operatorname{dom} \Phi_e$, one can let $X_s = \{x : \Phi_e(x)[s]\downarrow\}$. By Shoenfield's limit lemma, a Δ^0_2 set X can be approximated by a uniformly computable sequence of sets $X_0, X_1, X_2, \dots$ such that for every $n \in \mathbb{N}$, $\lim_s X_s(n)$ exists and equals $X(n)$. Such an approximation is called a Δ^0_2 *approximation of* X.[8]

7: There are three important families of sets:

Computable sets: Given n, it is possible to know whether it belongs to X or not, after a finite amount of time.

C.e. sets: If $n \in X$, then it will be enumerated in X after some point, but if $n \notin X$, we might never known whether it belongs to X or not.

Δ^0_2 *sets*: These are the $\emptyset'$-computable sets. Given some n, our belief of ownership to X might change finitely often over time, and then stabilize. However, we never know whether we have reached our limit or not.

8: Formally, a Δ^0_2 approximation of X is nothing but a computable function $f : \mathbb{N}^2 \to 2$ such that for every n, $\lim_s f(n,s)$ exists an equals $X(n)$.

2.1.3 Function growth

There is a duality between function growth and computational power. For example, any function dominating the halting time of programs computes the halting set. A function $f : \mathbb{N} \to \mathbb{N}$ *dominates* a function $g : \mathbb{N} \to \mathbb{N}$ if $f(x) \geq g(x)$ for every $x \in \mathbb{N}$. The *principal function* p_X of an infinite set $X = \{x_0 < x_1 < \dots\}$ is defined by $p_X(n) = x_n$.

Definition 2.1.4. A function f is *hyperimmune* if it is not dominated by any computable function. An infinite set X is *hyperimmune* it its principal function is hyperimmune.[9] ◇

9: Equivalently, an infinite set X is hyperimmune if for every c.e. array $\{F_n : n \in \mathbb{N}\}$, there is some $n \in \mathbb{N}$ such that $X \cap F_n = \emptyset$. A *c.e. array* is a c.e. sequence of finite coded non-empty sets which are pairwise disjoint.

A Turing degree $\mathbf{d}$ is *hyperimmune* if it computes (or equivalently contains) a hyperimmune function. Otherwise, $\mathbf{d}$ is *computably dominated* or *hyperimmune-free*. Every non-computable Δ^0_2 set is of hyperimmune degree, but there exists non-zero computably dominated degrees.

Definition 2.1.5. A function f is *dominating* if it eventually dominates every computable function. ◇

By Martin's domination theorem, a function is dominating iff it is of high degree. These degrees are precisely those able to uniformly list the computable sets, with repetitions.

2.1.4 DNC and PA degrees

By Kleene's recursion theorem, there is no total computable function $f : \mathbb{N} \to \mathbb{N}$ such that $\Phi_{f(e)} \neq \Phi_e$ for every $e \in \mathbb{N}$. The Turing degrees of fixpoint-free functions are those of diagonally non-computable functions.

Definition 2.1.6. A function f is *diagonally non-computable*[10] (DNC) if for every e, $f(e) \neq \Phi_e(e)$. ◇

10: A DNC function must always give a value, even if $\Phi_e(e)\uparrow$. An immediate diagonal argument shows that no such function is computable.

It might be useful to think of a DNC degree as the power, given a finite c.e. set W_e and a bound $b > \operatorname{card} W_e$, to find a value outside of W_e. A degree is DNC or high iff it contains a function which is almost-everywhere different from every total computable function.

A *binary tree* is a set $T \subseteq 2^{<\mathbb{N}}$ closed under prefix. A *path* through T is an infinite binary sequence $P \in cs$ such that every initial segment belongs to T.

We write $[T]$ for the class of all paths through T. A class $\mathcal{P} \subseteq 2^{\mathbb{N}}$ is Π^0_1 if it is for the form $[T]$ for some computable (or equivalently for some co-c.e.) tree $T \subseteq 2^{<\mathbb{N}}$. The Π^0_1 classes are the effectively closed classes in Cantor space.

11: Historically, a degree is PA if it contains a completion of Peano Arithmetic. The new definition is more useful in practice.

Definition 2.1.7. A degree $\mathbf{d}$ is *PA*[11] if for every infinite computable binary tree $T \subseteq 2^{<\mathbb{N}}$, $\mathbf{d}$ computes an infinite path. ◇

The PA degrees are precisely those which compute (or equivalently contain) a $\{0, 1\}$-valued DNC function. The class of such functions is Π^0_1, hence there exists a universal computable tree. By the low basis theorem and the computably dominated basis theorem, there are low and computably dominated PA degrees, respectively. A degree is PA or high iff it codes a uniform list of sets which contain, among others, all the computable sets.

2.2 Reverse mathematics

Reverse mathematics is a foundational program at the intersection of computability theory and proof theory, whose goal is to find optimal axioms to prove ordinary theorems.[12] The general idea consists in fixing a very weak base theory capturing *computable mathematics*, and given a theorem T, finding a set of axioms provably equivalent to T over this base theory. More recently, the term "reverse mathematics" took the broader meaning of studying mathematical theorems from the viewpoint of computability theory and proof theory.

12: By "ordinary", we mean theorem which belong to the core of mathematics, outside logics. Indeed, constructions in logics are metamathematical, and thus are often designed to escape the axiomatic strength of the standard mathematical practice.

13: There exists variants of reverse mathematics using the higher-order setting, or intuitionistic logic.

Traditional reverse mathematics[13] use the language of *second-order arithmetic*, that is, a two-sorted language with integers and sets of integers. In this language, every infinite mathematical object is represented by a set of integers. This enables to apply the framework of computability theory thanks to the correspondence between computability and definability. There are however two drawbacks: First, this restricts the scope to countable mathematics, or at least to mathematics which can be approximated through countable objects. Second, one must define an appropriate coding for every mathematical object. Thankfully, in many cases, the various natural representations of the same mathematical object are computably equivalent.

2.2.1 Base theory

The base theory RCA_0, standing for Recursive Comprehension Axiom, consists of Robinson arithmetic Q, together with the Σ^0_1-induction scheme and the Δ^0_1-comprehension scheme. More precisely, Robinson arithmetic[14] is the universal closure of the following axioms:

14: Robinson arithmetic is Peano arithmetic without the induction scheme.

(1) $x + 1 \neq 0$	(5) $x + (y + 1) = (x + y) + 1$
(2) $x = 0 \vee \exists y\ (x = y + 1)$	(6) $x \times 0 = 0$
(3) $x + 1 = y + 1 \rightarrow x = y$	(7) $x \times (y + 1) = (x \times y) + x$
(4) $x + 0 = x$	(8) $x < y \leftrightarrow \exists z\ (z \neq 0 \wedge x + z = y)$

A formula is *arithmetic* if it does not contain any second-order quantifier, but may contain second-order parameters. One can define a syntactic hierarchy of arithmetic formulas similar to the arithmetic hierarchy, by replacing the

computable predicate with a Δ^0_0 formula.[15] A Δ^0_0 *formula* contains only bounded first-order quantifiers, that is, quantifiers of the form $\forall x < y$ and $\exists x < y$.

15: Note that some computable sets (and even some primitive recursive sets) are not definable by Δ^0_0 formulas, but every c.e. set is definable by a Σ^0_1 formula, so the hierarchies coincide.

The Σ^0_1*-induction scheme* says, for every Σ^0_1 formula $\varphi(x)$,

$$\varphi(0) \wedge \forall x(\varphi(x) \to \varphi(x+1)) \to \forall x\ \varphi(x)$$

Restricting the induction scheme to capture computable mathematics might seem strange at first sight, as this scheme seems talk only about integers. An integer is a finite object, hence is computable. However, in non-standard models, a bounded set is considered as finite from inside the model, but if the bound is non-standard, it is actually infinite from an external viewpoint, and might be non-computable. Restricting induction restricts the complexity of the finite sets in the model.

The Δ^0_1*-comprehension scheme*[16] says, for every Σ^0_1 formula $\varphi(x)$ and Π^0_1 formula $\psi(x)$,

16: Being Δ^0_1 is not a syntactic notion. One therefore uses the trick of adding $\forall x(\varphi(x) \leftrightarrow \psi(x))$ as a premise, to ensure that the predicate is Δ^0_1.

$$\forall x(\varphi(x) \leftrightarrow \psi(x)) \to \exists X \forall y(\varphi(y) \leftrightarrow y \in X)$$

By relativization of Post's theorem, $X \leq_T Y$ iff X is $\Delta^0_1(Y)$. Therefore, the Δ^0_1-comprehension scheme ensures that the second-order part is downward-closed under the Turing reduction.

2.2.2 Models of RCA_0

A *model* in second-order arithmetic is of the form

$$\mathcal{M} = (M, S, +, \times, <, 0, 1)$$

where $S \subseteq \mathcal{P}(M)$. The *first-order part* M constitutes the integers, and the *second-order part* S are the sets of integers. An ω*-model* is a model whose first-order part is the set of standard integers ω, together with the usual operations $+, \times, <$. An ω-model is therefore fully specified by its second-order part, and is often identified with it. The ω-models of RCA_0 are precisely those whose second-order part is a Turing ideal.

Definition 2.2.1. A *Turing ideal*[17] is a class $\mathcal{I} \subseteq 2^{\mathbb{N}}$ closed under the following two operations:

(1) Turing reduction: $\forall X \in \mathcal{I}\ \forall Y \leq_T X\ Y \in \mathcal{I}$;
(2) Effective join: $\forall X \in \mathcal{I}\ \forall Y \in \mathcal{I}\ X \oplus Y \in \mathcal{I}$. ◇

17: Natural classes of Turing ideals are rare in computability theory. Besides *topped* Turing ideals of the form $\{Z \in 2^{\mathbb{N}} : Z \leq_T X\}$ for a fixed set X, the most notable ideal is the *K-trivials*, used in algorithmic randomness. The low degrees do not form a Turing ideal: there exists two low degrees joining to $\mathbf{0}'$.

The class of all computable sets is the smallest Turing ideal for inclusion. Thus, RCA_0 admits a least ω-model, consisting of only computable sets. It follows that if a theorem implies the existence of a non-computable object, then it is not provable over RCA_0. In this sense, RCA_0 captures computable mathematics.

2.2.3 Big Five

The early study of reverse mathematics witnessed the emergence of four main systems of axioms, linearly ordered by logical strength, such that most of mathematics is either provable in RCA_0, or provably equivalent to one of the four systems over RCA_0. These systems, together with RCA_0, are known

as the Big Five. We shall focus on the first two systems, namely, WKL_0 and ACA_0.

- WKL_0, standing for Weak König's lemma, is RCA_0 augmented with the statement "Every infinite binary tree admits an infinite path". This system informally captures compactness arguments. It is equivalent to the Borel-Lebesgue compactness theorem and Gödel's completeness theorem, among others. Contrary to RCA_0, WKL_0 does not admit a least ω-model. The second-order parts of its ω-models are closed under PA degrees, and are called *Scott ideals*.
- ACA_0, standing for Arithmetic Comprehension Axiom, is RCA_0 with the comprehension scheme for every arithmetic formula. Many important theorems, such as the Bolzano-Weierstrass theorem, are equivalent to ACA_0. Since the halting set is Σ^0_1-definable, the second-order parts of its ω-models are closed under the Turing jump, and called *jump ideals*. ACA_0 admits a least ω-model, whose second-order part corresponds to the arithmetic sets.

2.2.4 Computable reductions

More recently, the reverse mathematical framework was enriched with new reductions belonging to the computability-theoretic realm. A *problem*[18] is a relation $\mathsf{P} \subseteq 2^{\mathbb{N}} \times 2^{\mathbb{N}}$. An *instance* of P is an element of $\operatorname{dom} \mathsf{P} = \{X \in 2^{\mathbb{N}} : \exists Y\ (X, Y) \in \mathsf{P}\}$. Given an instance X of P, we denote by $\mathsf{P}(X) = \{Y : (X, Y) \in \mathsf{P}\}$ the class of *solutions* to X.

18: For instance, König's lemma is the problem whose instances are infinite, finitely branching trees, and a solution to a tree is an infinite path.

Definition 2.2.2. A problem P is *computably reducible* to Q (denoted $\mathsf{P} \leq_c \mathsf{Q}$) if for any instance X of P, there exists an instance $\tilde{X}$ of Q computable in X, such that for any Q-solution $\tilde{Y}$ to $\tilde{X}$, $X \oplus \tilde{Y}$ computes a P-solution to X.[19] ◇

19: One can see a computable reduction as the construction of a P-solver using a Q-solver, with only computable manipulations. Note that the original instance X of P can be used in the computation of the solution.

When the problems P and Q can be formulated as a second-order sentences, a reduction $\mathsf{P} \leq_c \mathsf{Q}$ can be seen as an implication $\mathsf{Q} \to \mathsf{P}$ over ω-models, in which only one application of Q is allowed.

2.3 Effective forcing

The framework of forcing was originally introduced by Paul Cohen to prove independence results in set theory. It is a central tool in computability theory to build sets of integers with specific computational properties, and can be seen as an elaboration of the finite extension method. The simplicity of its use in computability theory makes the setting ideal for a gentle introduction to forcing.

Definition 2.3.1. A *notion of forcing* is a partial order $(\mathbb{P}, \leq)$ together with an interpretation function $[\cdot] : \mathbb{P} \to \mathcal{P}(2^{\mathbb{N}})$ such that if $p \leq q$, then $[p] \subseteq [q]$.◇

Elements of $\mathbb{P}$ are called *conditions*. If $p \leq q$, then p is an *extension*[20] of q. Informally, a condition p is a partial approximation of the constructed object G, and $[p]$ is the class of all "candidate" objects. If $q \leq p$, then the approximation q is "more precise" than p, hence has less candidates.

20: The term "extension" suggests that p carries more information than q, thus the decreasing order might be confusing. It might be helpful to think of p and q in terms of interpretation. Then the decreasing order represents the decreasing in candidates.

Example 2.3.2. The following are notions of forcing

- Cohen forcing: $2^{<\mathbb{N}}$ with $\tau \leq \sigma$ if σ is a prefix of τ. The interpretation of σ is $[\sigma] = \{X \in 2^{\mathbb{N}} : \sigma \prec X\}$.
- Jockusch-Soare forcing: $\mathbb{P}$ is the partial order of computable infinite binary trees, ordered by inclusion. The interpretation of T is the class of its paths $[T]$.

2.3.1 Filter and genericity

Infinite objects are usually constructed by successive refinement of approximations. In the forcing setting, this would correspond to the construction of an infinite, decreasing sequence of conditions.

Definition 2.3.3. A *filter* on $(\mathbb{P}, \leq)$ is a non-empty class $\mathcal{F} \subseteq \mathbb{P}$ satisfying:

1. upward-closure: $\forall p \in \mathcal{F} \forall q \in \mathbb{P}\ (p \leq q \to q \in \mathcal{F})$
2. compatibility: $\forall p, q \in \mathcal{F} \exists r \in \mathcal{F}\ (r \leq p, q)$. ◇

21: The distinction between the two notions is not relevant in computability theory, and one might think of a filter as an infinite decreasing sequence of conditions.

Filters are a generalization of decreasing sequences of conditions[21], in that every sequence $p_0 \geq p_1 \geq \dots$ induces a filter $\mathcal{F} = \{q \in \mathbb{P} : \exists n\ p_n \leq q\}$. When the filter is appropriately chosen, there is a unique element $G_{\mathcal{F}} \in \bigcap_{p \in \mathcal{F}} [p]$, which is the object constructed by the filter.

Definition 2.3.4. A class $\mathcal{D} \subseteq \mathbb{P}$ is *dense* if for every $p \in \mathbb{P}$, there is some $q \leq p$ in $\mathcal{D}$. ◇

Intuitively, a class is dense if, when defining an infinite decreasing sequence of conditions, it is never too late to intersect $\mathcal{D}$. Indeed, at any point p_n of the construction, there exists an extension $p_{n+1} \leq p_n$ in $\mathcal{D}$.

Definition 2.3.5. A filter $\mathcal{F}$ is *generic* for a family of classes $\{\mathcal{D}_i\}_{i \in I}$ if $\mathcal{F} \cap \mathcal{D}_i \neq \emptyset$ for every $i \in I$. ◇

22: The concept of "sufficient genericity" alone does not exist, and always depends on a property $\varphi(G)$. We shall however sometimes say "Let $\mathcal{F}$ be a sufficiently generic filter" to mean that its level of genericity will be determined by the future properties we want $G_{\mathcal{F}}$ to satisfy.

One can easily see by a greedy construction of an infinite decreasing sequence of conditions that every countable family of dense classes admits a generic filter. Given a notion of forcing $(\mathbb{P}, \leq)$ and a property $\varphi(G)$, the statement "Every sufficiently generic[22] set satisfies $\varphi(G)$" means that there exists a countable sequence of dense classes $\{D_n\}_{n \in \mathbb{N}}$ such that, for every $\{D_n\}_{n \in \mathbb{N}}$-generic filter $\mathcal{F}$, $\varphi(G_{\mathcal{F}})$ holds.

All the notions of forcing we shall consider satisfy the following property:

(†) For every $n \in \mathbb{N}$, the following class is dense:

$$\mathcal{D}_n = \{p \in \mathbb{P} : \exists \sigma \in 2^n\ [p] \subseteq [\sigma]\}$$

In particular, for every $\{D_n\}_{n \in \mathbb{N}}$-generic filter $\mathcal{F}$, the intersection $\bigcap_{p \in \mathcal{F}} [p]$ will be a singleton.

2.3.2 Forcing relation

The core feature of forcing is the ability, given only an approximation $p \in \mathbb{P}$ of the object under construction, to already determine some properties the set will satisfy, no matter the remainder of the construction. Surprisingly, a very large class of properties can be determined in advance by approximations.

Definition 2.3.6. A condition $p \in \mathbb{P}$ *forces*[23] a property $\varphi(G)$ if for every sufficiently generic filter $\mathcal{F}$ containing p, $\varphi(G_{\mathcal{F}})$ holds. ◇

23: The naive approach would be to say that a condition p forces a property $\varphi(G)$ if it holds for every $G \in [p]$. This relation is too strong and does not enjoy the desirable properties of a forcing relation.

The above definition shall be referred to as a *semantic* definition. From a definitional viewpoint, the semantic definition is very complicated, as it requires to quantify over filters, which are higher-order objects. Thankfully, there exists an inductive syntactic definition of the forcing relation with much better definitional features.

In our setting, we shall be interested only in arithmetic properties.

Proposition 2.3.7. Let $(\mathbb{P}, \leq)$ be a notion of forcing satisfying (†) and $\varphi(G)$ be an arithmetic formula.

1. If p forces $\varphi(G)$ and $q \leq p$, then q forces $\varphi(G)$.
2. The class $\{p \in \mathbb{P} : p \text{ forces } \varphi(G) \text{ or } p \text{ forces } \neg\varphi(G)\}$ is dense. ★

This last property is essential, as it says that every arithmetic property can be decided by some condition. In particular, for every sufficiently generic filter $\mathcal{F}$, and every arithmetic formula $\varphi(G)$, then $\varphi(G_{\mathcal{F}})$ holds iff there is a condition $p \in \mathcal{F}$ forcing $\varphi(G)$.