Consistency, Optimality, and Incompleteness

Yijia Chen *
Shanghai Jiaotong University, China

Jörg Flum †
Albert-Ludwigs-Universität Freiburg, Germany

Moritz Müller ‡
Kurt Gödel Research Center, University of Vienna, Austria

Abstract

Assume that the problem P_0 is not solvable in polynomial time. Let T be a first-order theory containing a sufficiently rich part of true arithmetic. We characterize $T \cup \{Con_T\}$ as the minimal extension of T proving for some algorithm that it decides P_0 as fast as any algorithm $\mathbb{B}$ with the property that T proves that $\mathbb{B}$ decides P_0. Here, Con_T claims the consistency of T. As a byproduct, we obtain a version of Gödel's Second Incompleteness Theorem. Moreover, we characterize problems with an optimal algorithm in terms of arithmetical theories.

Keywords. Consistency, Optimal Algorithms, First-order Arithmetic, Gödel's Second Incompleteness Theorem.

1. Introduction

By Gödel's Second Incompleteness Theorem a consistent, computably enumerable and sufficiently strong first-order theory T cannot prove its own consistency Con_T. In other words, $T \cup \{Con_T\}$ is a proper extension of T.

In Bounded Arithmetic one studies the complexity of proofs in terms of the computational complexity of the concepts involved in the proofs (see e.g. [1, Introduction]). Stronger theories allow reasoning with more complicated concepts. For example, a computational problem may be solvable by an algorithm whose proof of correctness needs tools not available in the given theory; moreover, stronger theories may know of faster algorithms solving the problem. When discussing these issues with the authors, Sy-David Friedman asked whether $T \cup \{Con_T\}$ can be characterized in this context as a minimal extension of T. We could prove the following result (all terms will be defined in the paper).

Theorem 1. *Let P_0 be a decidable problem which is not decidable in polynomial time. Then there is a finite true arithmetical theory T_0 and a computable function F assigning to every computably enumerable theory T with $T \supseteq T_0$ an algorithm $F(T)$ such that (a) and (b) hold.*

(a) T_0 proves that $F(T)$ is as fast as any algorithm T-provably deciding P_0.

(b) For every theory T^* with $T^* \supseteq T$ the following are equivalent:

(i) T^* proves Con_T.

(ii) The algorithm $F(T)$ T^*-provably decides P_0.

(iii) There is an algorithm such that T^* proves that it decides P_0 and that it is as fast as any algorithm T-provably deciding P_0.

Hence, by merely knowing the extension T of T_0 we are able to compute the algorithm $F(T)$, which is, provably in T_0, as fast as any algorithm T-provably deciding P_0; however, in order to

*Email: yijia.chen@cs.sjtu.edu.cn
†Email: joerg.flum@math.uni-freiburg.de
‡Email: moritz.mueller@univie.ac.at

prove that $F(T)$ decides P_0 we need the full strength of $T \cup \{Con_T\}$. In this sense, $T \cup \{Con_T\}$ is a minimal extension of T.

It is known [8] that there are problems P_0 such that one can effectively assign to every algorithm $\mathbb{A}$ deciding P_0 a further algorithm $\mathbb{B}$ deciding P_0 such that $\mathbb{A}$ is not as fast as $\mathbb{B}$. Based on this fact, from our considerations yielding a proof of Theorem 1 we obtain a version of Gödel's Second Incompleteness Theorem.

The content of the different sections is the following. In Section 3, by a standard diagonalization technique we derive a result showing for every computably enumerable set D of algorithms the existence of an algorithm that on every input behaves as some algorithm in D and that is as fast as every algorithm in D (see Lemma 2). In Theorem 7 of Section 4 we characterize problems with an optimal algorithm in terms of arithmetical theories. Section 5 contains a proof of Theorem 1. Finally, we derive the Second Incompleteness Theorem in Section 6.

Many papers in computational complexity, older and recent ones, address the question whether hard problems have *optimal* or *almost optimal* algorithms. Although Levin [5] observed that there exists an optimal algorithm that finds a satisfying assignment for every satisfiable propositional formula, it is not known whether the class of satisfiable propositional formulas or the class of tautologies have an almost optimal algorithm.

Krajíček and Pudlák [4] showed for the latter class that an almost optimal algorithm exists if and only if "there exists a finitely axiomatized fragment T of the true arithmetic such that, for every finitely axiomatized consistent theory S, there exists a deterministic Turing machine $\mathbb{M}$ and a polynomial p such that for any given n, in time $\leq p(n)$ the machine $\mathbb{M}$ constructs a proof in T of $\mathrm{Con}_S(\underline{n})$." Here $\mathrm{Con}_S(\underline{n})$ claims that no contradiction can be derived from S by proofs of lengths at most n.

Hartmanis [2] and Hutter [3] considered 'provable' algorithms, where 'provable' refers to a computably enumerable, more or less specified true theory T. Hartmanis compares the class of problems decidable within a given time bound with the class of problems T-provably decidable within this time bound and he studies time hierarchy theorems in this context. Hutter constructs an algorithm "which is the fastest and the shortest" deciding a given problem. As Hutter says, van Emde Boas pointed out to him that it is not provable that his algorithm decides the given problem and that his proof is a "meta-proof which cannot be formalized within the considered proof system" and he adds that "a formal proof of its correctness would prove the consistency of the proof system, which is impossible by Gödel's Second Incompleteness Theorem."

Unlike these papers we do not assume in Theorem 1 that T is a true theory.

2. Some preliminaries

First we fix some notations and introduce some basic concepts. We consider problems as subsets of Σ^*, the set of strings over the alphabet $\Sigma := \{0, 1\}$. For an algorithm $\mathbb{A}$ and a string $x \in \Sigma^*$ we let $t_{\mathbb{A}}(x)$ denote the running time of $\mathbb{A}$ on x. In case $\mathbb{A}$ does not halt on x, we set $t_{\mathbb{A}}(x) := \infty$. If $t_{\mathbb{A}}(x)$ is finite, we denote by $\mathbb{A}(x)$ the output of $\mathbb{A}$ on x.

If $\mathbb{A}$ and $\mathbb{B}$ are algorithms, then $\mathbb{A}$ *is as fast as* $\mathbb{B}$ if there is a polynomial p such that for every $x \in \Sigma^*$

$$t_{\mathbb{A}}(x) \leq p\big(t_{\mathbb{B}}(x) + |x|\big). \tag{1}$$

Note that here we do not require that $\mathbb{A}$ and $\mathbb{B}$ decide the same $P \subseteq \Sigma^*$.

An algorithm deciding a problem P is *optimal* if it is as fast as every other algorithm deciding P, that is, if it has no superpolynomial speedup on an infinite subset of Σ^*. An algorithm $\mathbb{A}$ deciding P is *almost optimal* if (1) holds for every other algorithm $\mathbb{B}$ deciding P and every $x \in P$ (hence nothing is required of the relationship between $t_{\mathbb{A}}(x)$ and $t_{\mathbb{B}}(x)$ for $x \notin P$).

We do not distinguish algorithms from their codes by strings and we do not distinguish strings from their codes by natural numbers. However, we do not fix a computation model (Turing machines, random access machines,...) for algorithms. We state the results in such a way that they hold for every standard computation model.

3. Diagonalizing over algorithms

In computability theory diagonalization techniques are used in various contexts. We will make use of the following result.

Lemma 2 (Diagonalization Lemma). *Let D be a computably enumerable and nonempty set of algorithms. Then there is an algorithm $\mathbb{A}$ such that (a) and (b) hold.*

(a) The algorithm $\mathbb{A}$ halts precisely on those inputs on which at least one algorithm in D halts, and in that case it outputs the same as some algorithm in D; more formally, for all $x \in \Sigma^*$

- *$t_{\mathbb{A}}(x) < \infty \iff t_{\mathbb{D}}(x) < \infty$ for some $\mathbb{D} \in D$;*
- *if $t_{\mathbb{A}}(x) < \infty$, then there is $\mathbb{D} \in D$ with $\mathbb{A}(x) = \mathbb{D}(x)$.*

(b) There is a $d \in \mathbb{N}$ [1] such that for all $\mathbb{D} \in D$ there is a $c_{\mathbb{D}} \in \mathbb{N}$ such that for all $x \in \Sigma^*$

$$t_{\mathbb{A}}(x) \leq c_{\mathbb{D}} \cdot \big(t_{\mathbb{D}}(x) + |x|\big)^d.$$

Moreover, there is a computable function that maps any algorithm $\mathbb{E}$ enumerating the set D of algorithms to an algorithm $\mathbb{A}$ satisfying (a) and (b).

In particular, if all algorithms in D decide $P \subseteq \Sigma^*$, then $\mathbb{A}$ is an algorithm deciding P as fast as every $\mathbb{D} \in D$.

Proof: Let the algorithm $\mathbb{E}$ enumerate the set D of algorithms, that is, $\mathbb{E}$, once having been started, eventually prints out exactly the algorithms in D. For each $i \in \mathbb{N}$ we denote by $\mathbb{E}_i$ the last algorithm printed out by $\mathbb{E}$ in i steps; in particular, $\mathbb{E}_i$ is undefined if $\mathbb{E}$ hasn't printed any algorithm in at most i steps.

Algorithm $\mathbb{A}$ is defined as follows.

```
A(x)   // x ∈ Σ*
 1. ℓ ← 0
 2. for i = 0 to ℓ
 3.       if E_i is defined then simulate the (ℓ − i)th step of E_i on x
 4.       if the simulation halts then halt and output accordingly
 5. ℓ ← ℓ + 1
 6. goto 2.
```

In line 3, if $\mathbb{E}_i$ is defined, then the algorithm $\mathbb{A}$, after carrying out the simulation of the $(\ell - i)$th step of $\mathbb{E}_i$ on x, stores the actual configuration of $\mathbb{E}_i$. Of course (the code of) $\mathbb{A}$ can be computed from (the code of) $\mathbb{E}$. It is easy to see that $\mathbb{A}$ satisfies (a). Furthermore, there are constants $c_0, d_0 \in \mathbb{N}$ such that for all $x \in \Sigma^*$ and every $\ell \in \mathbb{N}$, lines 2–4 take time at most

$$c_0 \cdot (\ell + |x|)^{d_0}. \tag{2}$$

To verify (b), let $\mathbb{D} \in D$ and $i_{\mathbb{D}}$ be the minimum $i \in \mathbb{N}$ with $\mathbb{E}_i = \mathbb{D}$. Fix an input $x \in \Sigma^*$. For

$$\ell = i_{\mathbb{D}} + t_{\mathbb{E}_{i_{\mathbb{D}}}}(x) \quad \text{and} \quad i = i_{\mathbb{D}}$$

the simulation in line 3 halts if it didn't halt before. Therefore

$$\begin{aligned} t_{\mathbb{A}}(x) &\leq O\left(\sum_{\ell=0}^{i_{\mathbb{D}}+t_{\mathbb{D}}(x)} (\ell + |x|)^{d_0}\right) \qquad \big(\text{by (2)}\big) \\ &\leq O\left((i_{\mathbb{D}} + t_{\mathbb{D}}(x) + |x|)^{d_0+1}\right) \leq c_{\mathbb{D}} \cdot \big(t_{\mathbb{D}}(x) + |x|\big)^{d_0+1} \end{aligned}$$

[1] As the proof shows the constant $d \in \mathbb{N}$ does not even depend on D but it depends on the concrete machine model one uses.

for an appropriate constant $c_{\mathbb{D}} \in \mathbb{N}$ only depending on $\mathbb{D}$. □

The preceding proof uses the idea underlying standard proofs of a result due to Levin [5]. Even more, Levin's result is also a consequence of Lemma 2:

Example 3 (Levin [5]). Let $F : \Sigma^* \to \Sigma^*$ be computable. An *inverter of* F is an algorithm $\mathbb{I}$ that given y in the image of F halts with some output $\mathbb{I}(y)$ such that $F(\mathbb{I}(y)) = y$. On inputs not in the image of F, the algorithm $\mathbb{I}$ may do whatever it wants.

Let $\mathbb{F}$ be an algorithm computing F. For an arbitrary algorithm $\mathbb{B}$ define $\mathbb{B}^*$ as follows. On input y the algorithm $\mathbb{B}^*$ simulates $\mathbb{B}$ on y; if the simulation halts, then by simulating $\mathbb{F}$ it computes $F(\mathbb{B}(y))$; if $F(\mathbb{B}(y)) = y$, then it outputs $\mathbb{B}(y)$, otherwise it does not stop. Thus if $\mathbb{B}^*$ halts on $y \in \Sigma^*$, then it outputs a preimage of y and

$$t_{\mathbb{B}^*}(y) \leq O\big(t_{\mathbb{B}}(y) + t_{\mathbb{F}}(\mathbb{B}(y)) + |y|\big). \tag{3}$$

Furthermore, if $\mathbb{B}$ is an inverter of F, then so is $\mathbb{B}^*$.

Let $D := \{\mathbb{B}^* \mid \mathbb{B} \text{ is an algorithm}\}$. Denote by $\mathbb{I}_{\mathrm{opt}}$ an algorithm having for this D the properties of the algorithm $\mathbb{A}$ in Lemma 2. By the previous remarks it is easy to see that $\mathbb{I}_{\mathrm{opt}}$ is an inverter of F. Moreover, by Lemma 2 (b) and (3), we see that for any other inverter $\mathbb{B}$ of F there exists a constant $c_{\mathbb{B}}$ such that for all y in the image of F

$$t_{\mathbb{I}_{\mathrm{opt}}}(y) \leq c_{\mathbb{B}} \cdot \big(t_{\mathbb{B}}(y) + t_{\mathbb{F}}(\mathbb{B}(y)) + |y|\big)^d.$$

In this sense $\mathbb{I}_{\mathrm{opt}}$ is an optimal inverter of F.

4. Algorithms and arithmetical theories

To talk about algorithms and strings we use *arithmetical formulas*, that is, first-order formulas in the language $L_{\mathrm{PA}} := \{+, \cdot, 0, 1, <\}$ of Peano Arithmetic. Arithmetical sentences are *true* (*false*) if they hold (do not hold) in the standard L_{PA}-model. For a natural number n let $\underline{n}$ denote the natural L_{PA}-term without variables denoting n (in the standard model).

Recall that an arithmetical formula is Δ_0 if all quantifiers are bounded and it is Σ_1 if it has the form $\exists x_1 \ldots \exists x_m \psi$ where ψ is Δ_0.

We shall use a Δ_0-formula

$$\mathit{Run}(u, x, y, z)$$

that defines (in the standard model) the set of tuples (u, x, y, z) such that u is an algorithm that on input x outputs y by the (code of a complete finite) run z; recall that we do not distinguish algorithms from their codes by strings and strings from their codes by natural numbers.

For the rest of this paper we fix a decidable $P_0 \subseteq \Sigma^*$ and an algorithm $\mathbb{A}_0$ deciding P_0.

The formula

$$\begin{aligned}\mathit{Dec}_{P_0}(u) :=& \forall x \exists y \exists z \mathit{Run}(u, x, y, z) \wedge \\ & \forall x \forall y \forall y' \forall z \forall z' \big((\mathit{Run}(\underline{\mathbb{A}_0}, x, y, z) \wedge \mathit{Run}(u, x, y', z')) \to y = y'\big)\end{aligned}$$

defines the set of algorithms deciding P_0.

Let L_{all} with $L_{\mathrm{PA}} \subset L_{\mathrm{all}}$ be a language containing countably many function and relation symbols of every arity ≥ 1 and countably many constants. A *theory* is a set T of first-order L_{all}-sentences. We write $T \vdash \varphi$ if the theory T proves the sentence φ.

Definition 4. Let T be a theory.

(a) An algorithm $\mathbb{A}$ *T-provably decides* P_0 if $T \vdash \mathit{Dec}_{P_0}(\underline{\mathbb{A}})$.

(b) T is *sound for P_0-decision* means that for every algorithm $\mathbb{A}$

$$\text{if } T \vdash \mathit{Dec}_{P_0}(\underline{\mathbb{A}}), \text{ then } \mathbb{A} \text{ decides } P_0.$$

(c) T is *complete for P_0-decision* means that for every algorithm $\mathbb{A}$

$$\text{if } \mathbb{A} \text{ decides } P_0, \text{ then } T \vdash Dec_{P_0}(\underline{\mathbb{A}}).$$

For a computably enumerable sound theory T that proves $Dec_{P_0}(\underline{\mathbb{A}_0})$ the set

$$D(T) := \{\mathbb{D} \mid T \vdash Dec_{P_0}(\underline{\mathbb{D}})\} \tag{4}$$

is a computably enumerable and nonempty set of algorithms deciding P_0. Thus, by Lemma 2 for $D = D(T)$ we get an algorithm $\mathbb{A}$ deciding P_0 as fast as every algorithm in $D(T)$. If in addition T is complete for P_0-decision, then $D(T)$ would be the set of all algorithms deciding P_0 and thus $\mathbb{A}$ would be an optimal algorithm for P_0. So, the problem P_0 would have an optimal algorithm if we can find a computably enumerable theory that is both sound and complete for P_0-decision. Unfortunately, there is no such theory as shown by the following proposition. We relax these properties in Definition 6 and show in Theorem 7 that the new ones are appropriate to characterize problems with optimal algorithms.

Proposition 5. *There is no computably enumerable theory that is sound and complete for P_0-decision.*

Proof: We assume that there is a computably enumerable theory T that is sound and complete for P_0-decision and derive a contradiction by showing that then the halting problem for Turing machines would be decidable.

For every Turing machine $\mathbb{M}$ we consider two algorithms. On every input $x \in \Sigma^*$ the first algorithm $\mathbb{B}_1(\mathbb{M})$ first checks whether x codes a run of $\mathbb{M}$ accepting the empty input tape and then it simulates $\mathbb{A}_0$ on x (recall $\mathbb{A}_0$ is the fixed algorithm deciding P_0). If x codes an accepting run, then $\mathbb{B}_1(\mathbb{M})$ reverses the answer $\mathbb{A}_0(x)$ of $\mathbb{A}_0$ on x, otherwise it outputs exactly $\mathbb{A}_0(x)$. Clearly $\mathbb{B}_1(\mathbb{M})$ decides P_0 if and only if $\mathbb{M}$ does not halt on the empty input tape.

The second algorithm $\mathbb{B}_2(\mathbb{M})$, on every input $x \in \Sigma^*$ first checks exhaustively whether $\mathbb{M}$ halts on the empty input tape; if eventually it finds an accepting run, then it simulates $\mathbb{A}_0$ on x and outputs accordingly. It is easy to verify that $\mathbb{B}_2(\mathbb{M})$ decides P_0 if and only if $\mathbb{M}$ halts on the empty input tape.

As T is sound for P_0-decision, it proves at most one of $Dec_{P_0}(\underline{\mathbb{B}_1(\mathbb{M})})$ and $Dec_{P_0}(\underline{\mathbb{B}_2(\mathbb{M})})$, and as it is complete for P_0-decision it proves at least one of these sentences. Hence, given $\mathbb{M}$, by enumerating the T-provable sentences we can decide whether $\mathbb{M}$ halts on the empty input tape. □

Definition 6. A theory T is *almost complete for P_0-decision* if for every algorithm $\mathbb{A}$ deciding P_0 there is an algorithm T-provably deciding P_0 that is as fast as $\mathbb{A}$.

Theorem 7. *The following are equivalent for decidable $P_0 \subseteq \Sigma^*$:*

(i) P_0 has an optimal algorithm;

(ii) There is a computably enumerable and arithmetical theory T that is sound and almost complete for P_0-decision.

Proof: (i) ⇒ (ii): We set $T := \{Dec_{P_0}(\underline{\mathbb{A}})\}$ where $\mathbb{A}$ is an optimal algorithm for P_0. Then T is a computably enumerable true arithmetical theory. Truth implies soundness and almost completeness follows from the optimality of $\mathbb{A}$.

(ii) ⇒ (i): Let T be as in (ii). Then the set $D(T)$ defined by (4) is nonempty by almost completeness of T and, by soundness, it is a computably enumerable set of algorithms deciding P_0. By Lemma 2 for $D = D(T)$ we get an algorithm $\mathbb{A}$ deciding P_0 as fast as every algorithm in $D(T)$ and hence by almost completeness as fast as any algorithm deciding P_0. Thus, $\mathbb{A}$ is an optimal algorithm for P_0. □

A result related to the implication (ii) ⇒ (i) is shown by Sadowski in [7]. He shows assuming that there does not exist an almost optimal algorithm for the set TAUT of all propositional tautologies, that for every theory T there exists a subset of TAUT in PTIME which is not T-provably in PTIME (cf. [7, Definition 7.5]).

5. Proof of Theorem 1

Recall that $P_0 \subseteq \Sigma^*$ and that $\mathbb{A}_0$ is an algorithm deciding P_0. A theory T is Σ_1-*complete* if every true arithmetical Σ_1-sentence is provable in T. The following result is a consequence of Lemma 2.

Lemma 8. *Assume that P_0 is not decidable in polynomial time. Let T be a computably enumerable Σ_1-complete theory with $T \vdash Dec_{P_0}(\underline{\mathbb{A}_0})$. Then there is an algorithm $\mathbb{A}$ such that:*

(a) The algorithm $\mathbb{A}$ is total (i.e., $t_{\mathbb{A}}(x) < \infty$ for all $x \in \Sigma^*$) and as fast as every algorithm T-provably deciding P_0;

(b) T is consistent if and only if $\mathbb{A}$ decides P_0.

Moreover, there is a computable function diag *that maps any algorithm $\mathbb{E}$ enumerating some Σ_1-complete theory T with $T \vdash Dec_{P_0}(\underline{\mathbb{A}_0})$ to an algorithm $\mathbb{A}$ with (a) and (b).*

Proof: For an algorithm $\mathbb{B}$ let $\mathbb{B}\|\mathbb{A}_0$ be the algorithm that on input $x \in \Sigma^*$ runs $\mathbb{B}$ and $\mathbb{A}_0$ on x in parallel and returns the first answer obtained. Then

$$t_{\mathbb{B}\|\mathbb{A}_0} \leq O\Big(\min\{t_{\mathbb{B}}, t_{\mathbb{A}_0}\}\Big). \tag{5}$$

Claim 1. If T is consistent and $T \vdash Dec_{P_0}(\underline{\mathbb{B}})$, then $\mathbb{B}\|\mathbb{A}_0$ decides P_0.

Proof of Claim 1: By contradiction, assume that T is consistent and $T \vdash Dec_{P_0}(\underline{\mathbb{B}})$ but $\mathbb{B}\|\mathbb{A}_0$ does not decide P. Then $\mathbb{B}\|\mathbb{A}_0$ and $\mathbb{A}_0$ differ on some input $x \in \Sigma^*$. Thus $t_{\mathbb{B}}(x) \leq t_{\mathbb{A}_0}(x)$ and in particular $\mathbb{B}$ halts on x. Therefore, the following Σ_1-sentence φ is true:

$$\varphi := \exists x \exists y \exists y' \exists z \exists z' \big(Run(\underline{\mathbb{A}_0}, x, y, z) \wedge Run(\underline{\mathbb{B}}, x, y', z') \wedge \neg y = y' \big).$$

By Σ_1-completeness, $T \vdash \varphi$. However, φ logically implies $\neg Dec_{P_0}(\underline{\mathbb{B}})$ and thus T is inconsistent, a contradiction. ⊣

The set

$$D_1(T) := \{\mathbb{B}\|\mathbb{A}_0 \mid T \vdash Dec_{P_0}(\underline{\mathbb{B}})\}$$

is nonempty as $\mathbb{A}_0\|\mathbb{A}_0 \in D_1(T)$ by assumption. Let $\mathbb{A}$ be the algorithm obtained for $D = D_1(T)$ by Lemma 2. We show that statement (a) holds. By Lemma 2 (b), there is a $d \in \mathbb{N}$ such that for all $\mathbb{B}$ with $T \vdash Dec_{P_0}(\underline{\mathbb{B}})$ there is a $c_{\mathbb{B}}$ such that for all $x \in \Sigma^*$ we have $t_{\mathbb{A}}(x) \leq c_{\mathbb{B}} \cdot \big(t_{\mathbb{B}\|\mathbb{A}_0}(x) + |x|\big)^d$. Now (a) follows from (5).

For consistent T, by Claim 1 the set $D_1(T)$ only contains algorithms deciding P_0, thus $\mathbb{A}$ decides P_0 by Lemma 2.

If T is inconsistent, let $\mathbb{B}_{\text{bad}}$ be an algorithm that accepts every input in the first step. Then $\mathbb{B}_{\text{bad}}\|\mathbb{A}_0 \in D_1(T)$ by inconsistency of T. Thus, by Lemma 2 (b), the algorithm $\mathbb{A}$ runs in polynomial time and thus does not decide P_0.

As from an algorithm enumerating T we effectively get an algorithm enumerating $D_1(T)$, by Lemma 2 it should be clear that a computable function *diag* as claimed exists. □

Remark 9. As the preceding proof shows we only need the assumption that P_0 is not decidable in polynomial time in the proof of the implication from right to left in (b).

Proof of Theorem 1: Recall that Robinson introduced a finite, Σ_1-complete, and true arithmetical theory Q. Let P_0 be a decidable problem which is not decidable in polynomial time. Among others, the finite true arithmetical theory T_0 claimed to exist in Theorem 1 will extend Q and contain a formalization of Lemma 8.

We choose a Σ_1-formula $Prov(x, y)$ defining (in the standard model) the set of pairs (m, n) such that algorithm m enumerates a theory[2] that proves the sentence with Gödel number n. We let

$$Con(x) := \neg Prov\big(x, \ulcorner \underline{\neg 0 = 0} \urcorner\big)$$

[2]We may assume that every enumeration algorithm enumerates a theory by deleting those printed strings that are not sentences.

(here $\ulcorner\varphi\urcorner$ denotes the Gödel number of φ). If $\mathbb{E}$ enumerates a theory T, we write Con_T for $Con(\underline{\mathbb{E}})$.[3]

Let $f : \mathbb{N} \to \mathbb{N}$ be the function given by

$$f(m) := \ulcorner Dec_{P_0}(\underline{m})\urcorner.$$

Both, this function f and the function *diag* from Lemma 8 are computable and hence, Σ_1-definable in Q. For the sake of completeness we recall what this means, say, for f: There is an arithmetical Σ_1-formula $\varphi_f(x, y)$ such that, for all $m, k \in \mathbb{N}$,

- if $f(m) = k$, then $Q \vdash \varphi_f(\underline{m}, \underline{k})$;
- if $f(m) \neq k$, then $Q \vdash \neg\, \varphi_f(\underline{m}, \underline{k})$;
- $Q \vdash \exists^{=1} y\, \varphi_f(\underline{m}, y)$.

For better readability we write arithmetical formulas using f and *diag* as function symbols.

Further, let the arithmetical formula $As\text{-}fast\text{-}as(x, y)$ define the pairs (n, m) such that algorithm n is as fast as algorithm m and let $Ptime(x)$ define the set of polynomial time algorithms. Finally, we set

$$Afap(x, y) := \forall z\big(Prov(x, f(z)) \to As\text{-}fast\text{-}as(y, z)\big).$$

Then for an algorithm $\mathbb{E}$ enumerating a theory T the statement "the algorithm $F(T)$ is as fast as any algorithm T-provably deciding P_0," that is, the statement (a) in Theorem 1 is formalized by the sentence

$$Afap\big(\underline{\mathbb{E}}, \underline{F(T)}\big). \tag{6}$$

Let $e\text{-}Rob(x)$ be a Σ_1-formula expressing that the algorithm x enumerates a theory extending $Q \cup \{Dec_{P_0}(\underline{\mathbb{A}_0})\}$.

We now define the theory T_0. It extends $Q \cup \{Dec_{P_0}(\underline{\mathbb{A}_0})\}$ by the following sentences (s1)–(s5):

(s1) $\forall x\big(e\text{-}Rob(x) \to Afap(x, diag(x))\big)$
(a formalization of Lemma 8 (a))

(s2) $\forall x\big((Con(x) \wedge e\text{-}Rob(x)) \to Dec_{P_0}(diag(x))\big)$
(a formalization of part of Lemma 8 (b))

(s3) $\forall x(Ptime(x) \to \neg Dec_{P_0}(x))$
(P_0 is not in PTIME)

(s4) $\forall x\big(\neg Con(x) \to \forall y(Sent(y) \to Prov(x, y))\big)$
(every inconsistent theory proves every sentence; here $Sent(y)$ is a Δ_0-formula defining the first-order L_{all}-sentences)

(s5) $\forall x \forall y\big((As\text{-}fast\text{-}as(x, y) \wedge Ptime(y)) \to Ptime(x)\big)$
(if algorithm x is as fast as the polynomial algorithm y, then it is polynomial too).

Let T be a computably enumerable extension of T_0 and let $\mathbb{E}$ be an algorithm enumerating T. We claim that for the algorithm

$$F(T) := diag(\mathbb{E})$$

(see Lemma 8) the statements (a) and (b) of Theorem 1 hold.

The arithmetical sentence $\underline{F(T)} = diag(\underline{\mathbb{E}})$ is Σ_1 and true, so T_0 proves it by Σ_1-completeness (as $T_0 \supseteq Q$). By the same reason, $T_0 \vdash e\text{-}Rob(\underline{\mathbb{E}})$. As T_0 contains (s1), $T_0 \vdash Afap(\underline{\mathbb{E}}, \underline{F(T)})$; that is,

[3] The notation is ambiguous, as the definition depends on the choice of $\mathbb{E}$, however not the arguments to follow.

T_0 proves that $F(T)$ is as fast as any algorithm T-provably deciding P_0. Thus (a) in Theorem 1 holds.

We turn to (b). Let T^* be a theory with $T^* \supseteq T$.

(i) $\Rightarrow$ (ii): So, we assume that $T^* \vdash \mathit{Con}_T$. We already know that T_0, and hence T^*, proves $\mathit{e\text{-}Rob}(\underline{\mathbb{E}})$. As T^* contains (s2), for $x = \underline{\mathbb{E}}$ we see that $T^* \vdash \mathit{Dec}_{P_0}(\mathit{diag}(\underline{\mathbb{E}}))$ and thus $T^* \vdash \mathit{Dec}_{P_0}(\underline{F(T)})$; that is, $F(T)$ T^*-provably decides P_0.

(ii) $\Rightarrow$ (iii): Immediate by part (a) of the theorem.

(iii) $\Rightarrow$ (i): Let $\mathbb{A}$ be an algorithm such that $T^* \vdash \mathit{Dec}_{P_0}(\underline{\mathbb{A}})$ and $T^* \vdash \mathit{Afap}(\underline{\mathbb{E}}, \underline{\mathbb{A}})$; the latter means that

$$T^* \vdash \forall z(\mathit{Prov}(\underline{\mathbb{E}}, f(z)) \to \mathit{As\text{-}fast\text{-}as}(\underline{\mathbb{A}}, z)). \tag{7}$$

Let $\mathbb{B}$ be an algorithm such that

$$T^* \vdash \mathit{Ptime}(\underline{\mathbb{B}}). \tag{8}$$

Then T^* proves the following implications:

$$\begin{aligned}
&\neg \mathit{Con}_T \to \mathit{Prov}(\underline{\mathbb{E}}, f(\underline{\mathbb{B}})) && \text{(by (s4) and as } \mathit{Sent}(f(\underline{\mathbb{B}})) \text{ is } \Sigma_1)\\
&\neg \mathit{Con}_T \to \mathit{As\text{-}fast\text{-}as}(\underline{\mathbb{A}}, \underline{\mathbb{B}}) && \text{(by (7))}\\
&\neg \mathit{Con}_T \to \mathit{Ptime}(\underline{\mathbb{A}}) && \text{(by (8) and (s5))}\\
&\neg \mathit{Con}_T \to \neg \mathit{Dec}_{P_0}(\underline{\mathbb{A}}) && \text{(by (s3)).}
\end{aligned}$$

As $T^* \vdash \mathit{Dec}_{P_0}(\underline{\mathbb{A}})$, we see that $T^* \vdash \mathit{Con}_T$. □

6. Gödel's Second Incompleteness Theorem

Let P_{exp} be the following problem:

P_{exp}	
Instance:	An algorithm $\mathbb{A}$.
Problem:	Is it true that $\mathbb{A}$ does not accept $\mathbb{A}$ in at most $2^{\lvert\mathbb{A}\rvert}$ steps?

Theorem 10 ([8]). *There is a polynomial time computable function g that maps any algorithm $\mathbb{A}$ deciding P_{exp} to an algorithm $g(\mathbb{A})$ deciding P_{exp} such that $\mathbb{A}$ is not as fast as $g(\mathbb{A})$.*

Proof: We fix a polynomial time computable function which assigns to every algorithm $\mathbb{A}$ and $n \geq 1$ an algorithm $\mathbb{A}_n$ where $\mathbb{A}_n$ is "the same as $\mathbb{A}$ but padded with n useless instructions." The properties of $\mathbb{A}_n$ we need are

$$\lvert\mathbb{A}_n\rvert \geq n,\ t_{\mathbb{A}_n} = t_{\mathbb{A}},\ \text{and}\ \ \mathbb{A}_n \text{ and } \mathbb{A} \text{ accept the same language.} \tag{9}$$

Note that any algorithm $\mathbb{A}$ deciding P_{exp} does not reject $\mathbb{A}$. Hence, for such an $\mathbb{A}$ we have $\mathbb{A} \in P_{\text{exp}}$ and $t_{\mathbb{A}}(\mathbb{A}) > 2^{\lvert\mathbb{A}\rvert}$. Moreover, by (9), we have

$$t_{\mathbb{A}}(\mathbb{A}_n) = t_{\mathbb{A}_n}(\mathbb{A}_n) > 2^{\lvert\mathbb{A}_n\rvert} \geq 2^n \tag{10}$$

(the strict inequality holding as $\mathbb{A}_n$ decides P_{exp}, too).

The function g computes for any algorithm $\mathbb{A}$ the following algorithm $\mathbb{B} := g(\mathbb{A})$: On input x the algorithm $\mathbb{B}$ first checks whether $x \in \{\mathbb{A}_1, \mathbb{A}_2, \ldots\}$ (this can be done in time polynomial in $\lvert x\rvert$); if so, $\mathbb{B}$ immediately accepts, otherwise it simulates $\mathbb{A}$ on x and answers accordingly. Clearly, if $\mathbb{A}$ decides P_{exp}, then $\mathbb{B}$ decides P_{exp} and superpolynomially speeds up $\mathbb{A}$ on $\{\mathbb{A}_1, \mathbb{A}_2, \ldots\}$ by (10). □

Using this result and results of preceding sections we derive the following version of Gödel's Second Incompleteness Theorem:

Theorem 11. *There is a finite true arithmetical theory T_1 such that for every computably enumerable theory $T \supseteq T_1$,*

$$\textit{if } T \textit{ is consistent, then } T \textit{ does not prove } \mathit{Con}_T.$$

Proof: We take as P_0 the problem $P_{\exp}$ of the preceding theorem and let g be the function defined there. We know that P_0 is not decidable in polynomial time. Furthermore, as in the previous sections, we fix an algorithm $\mathbb{A}_0$ deciding P_0. Let T_0 be the true arithmetical, finite, and Σ_1-complete theory defined in the previous section satisfying Theorem 1.

Being computable, g is Σ_1-definable; for simplicity of notation we use g like a function symbol in arithmetical formulas. This is to be understood as explained in the previous proof. The theory T_1 is obtained from T_0 by adding the sentence

(s6) $\forall x(\mathit{Dec}_{P_0}(x) \to \mathit{Dec}_{P_0}(g(x)))$.

Let $T \supseteq T_1$ be a theory enumerated by the algorithm $\mathbb{E}$. Assume that T is consistent. Then, by Lemma 8 (b),

$$\mathit{diag}(\mathbb{E}) \text{ decides } P_0$$

and thus, by Theorem 10,

$$\mathit{diag}(\mathbb{E}) \text{ is not as fast as } g(\mathit{diag}(\mathbb{E})). \tag{11}$$

Observe that $T \vdash \textit{e-Rob}(\underline{\mathbb{E}})$ being a true Σ_1-sentence. By contradiction, suppose that $T \vdash \mathit{Con}_T$, that is, $T \vdash \mathit{Con}(\underline{\mathbb{E}})$. Then, $T \vdash \mathit{Dec}_{P_0}(\mathit{diag}(\underline{\mathbb{E}}))$ by (s2) and hence, $T \vdash \mathit{Dec}_{P_0}(g(\mathit{diag}(\underline{\mathbb{E}}))$ by (s6). Setting $\mathbb{B} := g(\mathit{diag}(\mathbb{E}))$ the sentence $\underline{\mathbb{B}} = g(\mathit{diag}(\underline{\mathbb{E}}))$ is a true Σ_1-sentence; so T proves it. Then, $T \vdash \mathit{Dec}_{P_0}(\underline{\mathbb{B}})$. This means that $\mathbb{B} = g(\mathit{diag}(\mathbb{E}))$ T-provably decides P_0. By Lemma 8 (a), $\mathit{diag}(\mathbb{E})$ is as fast as $g(\mathit{diag}(\mathbb{E}))$ contradicting (11). □

Let T_1 be the theory just defined. We show that for every true and computably enumerable arithmetical theory $T \supseteq T_1$, the extension $T \cup \{\mathit{Con}_T\}$ knows of strictly faster algorithms deciding $P_{\exp}$ than T:

Corollary 12. *Let T_1 be the theory defined in the previous proof. Then for every true and computably enumerable arithmetical theory $T \supseteq T_1$ there is an algorithm $\mathbb{A}$ such that:*

(a) The algorithm $\mathbb{A}$ $T \cup \{\mathit{Con}_T\}$-provably decides $P_{\exp}$ and is as fast as every algorithm that T-provably decides $P_{\exp}$;

(b) No algorithm that T-provably decides $P_{\exp}$ is as fast as $\mathbb{A}$.

Proof: Let T be as stated. By Theorem 1 for $P_0 := P_{\exp}$ and $T^* := T \cup \{\mathit{Con}_T\}$, we get that the algorithm $\mathbb{A} := F(T)$ $T \cup \{\mathit{Con}_T\}$-provably decides P_0. Furthermore,

$$\mathbb{A} \text{ is as fast as any algorithm that } T\text{-provably decides } P_0. \tag{12}$$

This shows (a). For (b) let $\mathbb{B}$ be an arbitrary algorithm that T-provably decides P_0. Then, by (s6),

$$\text{the algorithm } g(\mathbb{B}) \text{ } T\text{-provably decides } P_0. \tag{13}$$

As T is a true arithmetical theory, the algorithms $\mathbb{B}$ (and $g(\mathbb{B})$) decide P_0. Hence, by Theorem 10,

$$\mathbb{B} \text{ is not as fast as } g(\mathbb{B}). \tag{14}$$

From (12)–(14) we conclude that $\mathbb{B}$ is not as fast as $\mathbb{A}$. □

One can get rid of the assumption that T must be a *true* arithmetical theory in the previous result by adding to T_1 a further true arithmetical sentence:

Corollary 13. *There is a finite true arithmetical theory T_2 such that for every consistent, computably enumerable theory $T \supseteq T_2$ there is an algorithm $\mathbb{A}$ such that:*

(a) The algorithm $\mathbb{A}$ *$T \cup \{Con_T\}$-provably decides* $P_{\exp}$ *and is as fast as every algorithm that T-provably decides* $P_{\exp}$*;*

(b) No algorithm that T-provably decides $P_{\exp}$ *is as fast as* $\mathbb{A}$*.*

Proof: Again, we take as P_0 the problem $P_{\exp}$ and let $\mathbb{A}_0$ be an algorithm deciding it. Let h be the function that maps an algorithm $\mathbb{B}$ to $\mathbb{B}\|\mathbb{A}_0$ (as in the proof of Lemma 8 the algorithm $\mathbb{B}\|\mathbb{A}_0$ on input $x \in \Sigma^*$ runs $\mathbb{B}$ and $\mathbb{A}_0$ in parallel and returns the first answer obtained).

The theory T_2 is obtained from T_1 by adding the true arithmetical sentence

(s7) $\forall x(Dec_{P_0}(x) \to Dec_{P_0}(h(x)))$.

Let T be as stated and again let $\mathbb{A} := F(T)$. As in the previous proof, we see that statement (a) holds true.

For (b), let $\mathbb{B}$ be an algorithm with $T \vdash Dec_{P_0}(\underline{\mathbb{B}})$. Using first (s7) and then (s6) we get

$$\text{the algorithm } g(h(\mathbb{B})) \text{ } T\text{-provably decides } P_0.$$

By (a), therefore it suffices to show that $\mathbb{B}$ is not as fast as $g(h(\mathbb{B}))$. As by definition of h, the algorithm $h(\mathbb{B})$ is as fast as $\mathbb{B}$ (see (5)), it already suffices to show that $h(\mathbb{B})$ is not as fast as $g(h(\mathbb{B}))$. By Claim 1 in the proof of Lemma 8, we know that $h(\mathbb{B})$ decides P_0. Then Theorem 10 indeed proves that $h(\mathbb{B})$ is not as fast as $g(h(\mathbb{B}))$. □

Acknowledgments. The authors thank the John Templeton Foundation for its support under Grant #13152, *The Myriad Aspects of Infinity.* Yijia Chen is affiliated with BASICS and MOE-MS Key Laboratory for Intelligent Computing and Intelligent Systems which is supported by National Nature Science Foundation of China (61033002).

References

[1] S. A. Cook and P. Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, 2010.

[2] J. Hartmanis. Relations between diagonalization, proof systems, and complexity gaps. *Theoretical Computer Science*, 8:239–253, 1979.

[3] M. Hutter. The fastest and shortest algorithm for all well-defined problems. *International Journal of Foundations of Computer Science*, 13:431–443, 2002.

[4] J. Krajíček and P. Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic*, 54:1063–1079, 1989.

[5] L. Levin. Universal search problems (in Russian). *Problemy Peredachi Informatsii*, 9:115-116, 1973.

[6] J. Messner. On optimal algorithms and optimal proof systems. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS'99)*, Lecture Notes in Computer Science 1563, 361–372, 1999.

[7] Z. Sadowski. On an optimal propositional proof system and the structure of easy subsets. *Theoretical Computer Science*, 288:181–193, 2002.

[8] L. Stockmeyer *The complexity of decision problems in automata theory and logic.* Ph.D. thesis, MIT, 1974.