Overview of recent claims about $\mathcal{P} \neq \mathcal{NP}$

Sven Laur
swen@math.ut.ee

Helsinki University of Technology

---

†The text in orange represents author's personal opinion and thus might be slightly subjective.

Is the question $\mathcal{P} = \mathcal{NP}$ really important?

Most mathematicians seem to belive that the proof of $\mathcal{P} = \mathcal{NP}$ would have a big practical impact. However, the latter is not true:

The class of polynomial algorithms $\mathcal{P}$ is rather an artifact of complexity theory than a conceptual description of feasible algorithms.

– The class $\mathcal{P}$ is just the first "reasonable" complexity class that is closed under superposition—one can freely use sub-routines.

– Due to the limited physical resources one can never implement a Turing machine. All computing devices are finite automatons.

– Asymptotic complexity is just an approximation. For large k, the exponential working time $2^n \ll n^k$ for all feasible instances of n.

– All feasible aloforithms have working time $\mathcal{O}(n^6)$ and for many areas already $\Omega(n^2)$ is infeasible.

Could the proof of $\mathcal{P} = \mathcal{NP}$ be useful?

There are three possible levels of ignorance.

- The proof itself is non-constructive.
  - Has no practical implications, only motivates "smart" people.
- The problem $\mathcal{P} = \mathcal{NP}$ is independent from Peano Arithmetics.
  - The question becomes just a matter of taste.
- The proof is constructive, but the algorithm complexity is $\Omega(n^6)$.
  - The for sufficient $n \geq 10000$ the problems still remain intractable.
  - The non-existance of non-trivial polynomial-time algorithms with a complexity $\Omega(n^6)$ is rather an artifact of limited intellectual capabilities of mankind than a "general" law.

Could the proof of $\mathcal{P} \neq \mathcal{NP}$ be useful?

There are three possible levels of ignorance.

- The proof does not change the *status quo*.
  - The result has no practical implications, exept some lower bounds for approximations factors of $\mathcal{NP}$-hard problems become provable.
  - Still it may be difficult to find hard problem instances.
- The factorization problem is belived to be non-$\mathcal{NP}$-complete.
  - Thus $\mathcal{P} \neq \mathcal{NP}$ does not *apriori* give a complexity guarantee.
- No guarantees for practical cryptographic primitives.
  - The size and structure of problem instance is fixed.
  - Lower bounds on scheme complexity are required.

General remarks about the article

Tatsuaki Okamoto and Ryo Kashima, *Resource Bounded Unprovability of Computational Lower Bounds.*

Submitted to Cryptology ePrint archive on 9th September 2003. Last time revised on 6th January 2005.

The difference between two versions is substantial:

– Roughly twenty pages of a new material.

– Obvious flaws have been fixed, but the *essential* problems are still unaddressed.

– The mistake is implicitly hidden among assumptions.

– The readability has not been improved rather the things have gone worse: misuse and abjuce of formal notation, incorrectly stated theorems, incoherent and hard-to-follow proofs.

Historical development of the argument

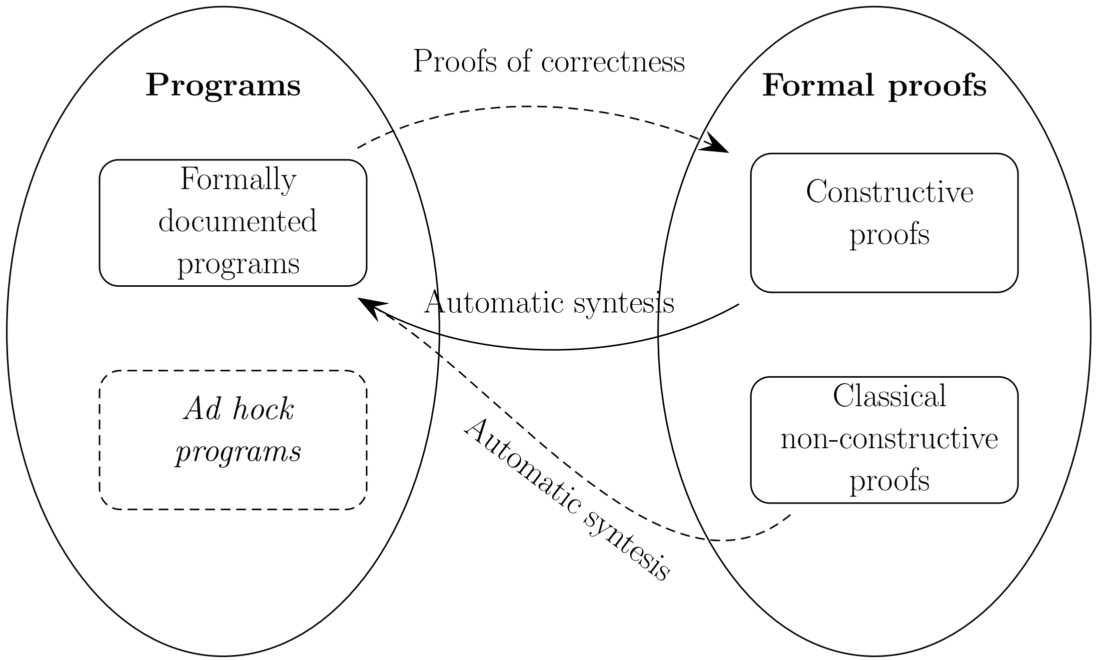
- 2003 Concept of polynomial-time provable languages:
  - First and Second Incompleteness Theorems.
  - Sketchy and flawed connection with the $\mathcal{P} = \mathcal{NP}$ problem.
- Somewhere in 2004 authors refined their arguments:
  - Concept polynomially decidable predicates in Peano Arithmetics.
  - First and Second Incompleteness Theorems.
  - Poly-time provable languages become obsolete.
- Questionable and unlinked poly-time ω-consistency assumption:
  - Non-existance of $\mathcal{P} = \mathcal{NP}$ proof under poly-time ω-consistency.

True result: There are no prover that for any poly-time SAT decider $\mathcal{D}$ could produce an example, where $\mathcal{D}$ fails, in poly-time w.r.t. instance size.

Outline of the talk

- Basic concepts of formal logic
- Introduction to Peano Arithmetics
- Polynomial-time proofs for languages of decidable formulas
- Meta-level proofs and their properties
- Polynomial-time descisions for languages of canonic decidable formulas
- Why the proof of unprovability of $\mathcal{P} \neq \mathcal{NP}$ is not convincing.

Duality between programs and proofs



- Each constuctive formal proof gives a rise to a program.
- But the converse is not true—correctness proofs are hard.

Signatures and interpretation

The syntax of first order logic is determined by a signature $\sigma = \langle \mathcal{C}; \mathcal{F}; \mathcal{P} \rangle$.

- $\mathcal{C}$ contains all constant symbols such as $0, 1, \ldots$.
- $\mathcal{F}$ contains all function symbols such as $+, \cdot, \mathsf{exp}, \mathsf{rem}, \mathsf{div}$.
- $\mathcal{P}$ contains all predicate symbols such as $=, <, \leq$.
- Defining additional function or predicates is not allowed. Still one can use macro constructions to represent functions and predicates.

Interpretation $\mathcal{I}$ assigns meaning to formulas.

- A universe $\mathcal{M} \neq \emptyset$ is fixed.
- Constants, functions and predicates are instantiated.

Theories. True and provable statements

A theory $\mathcal{T}$ is determined by set of axioms $\mathcal{T}$. An interpretation $\mathcal{I}$ is consistent with $\mathcal{T}$ iff all axioms are satisfied.

Definition. *A formula ϕ follows from axioms $\mathcal{T}$ if for all consistent interpetations $\mathcal{I}$ the evaluation $\mathcal{I}(\phi)$ is true. We denote it by $\mathcal{T} \models \phi$.*

Definition. *A proof-system $\mathcal{V}$ is a set of formal rules that allows to derive only a (sub)set of true formulas.*

Definition. *A formula ϕ is provable w.r.t. $\mathcal{T}$ if ϕ is derivable with the proof-system $\mathcal{V}$. We denote it by $\mathcal{T} \vdash \phi$.*

The set of provable formulas may be considerable smaller than the set of true formulas. The opposite is impossible.

Gödel's Theorems

Theorem (Completeness Theorem). *Let a theory $\mathcal{T}$ be a finitely axiomatiable. Then the set of true formulas is recursively enumerable and every true formula is provable.*

Theorem (Incompleteness theorem). *There are true but not provable formulas in Peano Arithmetics, unless it is inconsisent.*

Corollary. *Arithmetics is not a finite axiomatiable as a theory in the first order logic.*

Theorem (Chaitin). *The fact that formula is not provable is not itselt provable in general.*

Okamoto and Kashima tried to prove that $\mathcal{P} \neq \mathcal{NP}$ statement is not provable statements by a sketching similar framework as Gödel.

Axiom scheme for Peano Arithmetics

Let ϕ be any well-formed formula in the signature $\sigma = \langle 0, 1; +, \cdot; = \rangle$.

Equality Axioms

$\forall x(x = x)$

$\forall x \forall y(x = y \supset y = x)$

$\forall x \forall y \forall z((x = y \wedge y = z) \supset x = z)$

$\forall x \forall y(\phi(\ldots, x, \ldots) \supset \phi(\ldots, y, \ldots))$

Successor Axioms

$\forall x \neg(x + 1 = x)$

$\forall x \forall y(x + 1 = y + 1 \supset y = x)$

$(\phi(0) \wedge \forall x(\phi(x) \supset \phi(x + 1)) \supset \forall x \phi(x)$

Addition axioms

$\forall x(x + 0 = x)$

$\forall x \forall y(x + (y + 1) = (x + y) + 1)$

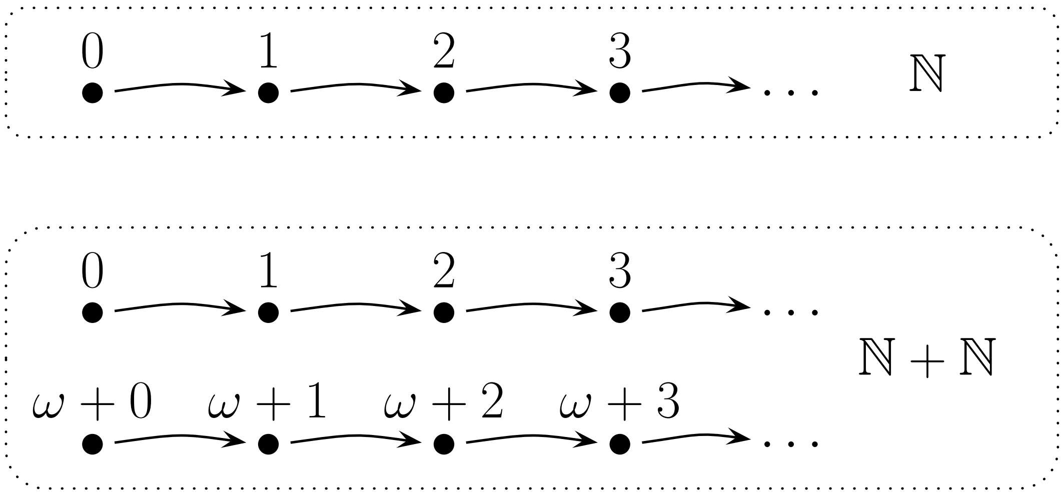
Multiplication Axioms

$\forall x(x \cdot 0 = x)$

$\forall x \forall y(x \cdot (y + 1) = x \cdot y + x)$

Why do we need induction scheme?

First order Peano Arithmetics has many models.



Induction axiom states that we do not care about non-successors of 0.

Introducing lists with variable length

Gödel originally proposed a β-function to get a grip over lists

$$\forall k\ \forall a_1,\ldots,a_k \in \mathbb{N} \quad \exists a,b \in \mathbb{N}: \quad \beta(a,b,i) = a_i, \quad i = 1,\ldots,k$$

The latter allows to write Turing machine $\mathcal{M}$ as a predicate $\rho_{\mathcal{M}}(x,y)$

$$\exists t \exists a \exists b (\underbrace{\rho_{\mathsf{init}}(\beta(a,b,0),x)}_{\text{Fix initial configuration}} \wedge \underbrace{\forall (t_1 < t)\ \rho_{\mathsf{tran}}(\beta(a,b,t_1),\beta(a,b,t_1+1))}_{\text{Force transitions of } \mathcal{M}}$$

$$\wedge \underbrace{\rho_{\mathsf{ends}}(\beta(a,b,t),y)}_{\text{Fix end configuration}})$$

The construction is computationally inefficient—Gödel just did not care.

Optimising the proof-system

The proof of $2^x = y$ has exponential in size of x if we use Gödels β-function.

It is not known wheter $2^x = y$ has an alternative representation in signature $\sigma = \langle 0, 1; +, \cdot; = \rangle$ so that the proofs have polynomial size.

Hence, we need to extend the sigature and proof-system by a adding function $\mathsf{exp}(x) = 2^x$. For convenience, we use also

$$\mathsf{len}(x) = |x| \qquad \mathsf{bit}(x, i) = x_i \qquad \boldsymbol{\beta}_{\mathsf{e}}(a, b, t) = a_i$$

where $x = x_n \cdots x_0$ and $a = a_k 2^{b(k-1)} + \cdots + a_0$

Okamoto and Kashima fail to grasp that subtlety in their article.

Formulas and proofs as numbers

Consider an efficent encoding of formulas and proofs

$$\mathfrak{F} \ni \phi \mapsto \mathsf{code}_P(\phi) \in \mathbb{N}$$
$$\mathfrak{P} \ni \pi \mapsto \mathsf{code}_P(\pi) \in \mathbb{N}$$

Then we can device a verifying Turing machine $\mathcal{V}$ such that

$$\mathcal{V}(\mathsf{code}_P(\phi), \mathsf{code}_P(\pi)) = \begin{cases} 1, & \text{if } \pi \text{ is valid proof of } \phi, \\ 0, & \text{otherwise.} \end{cases}$$

For clarity, we skip the details and use $\mathcal{V}(\phi, \pi)$ instead.

Polynomial-time provable languages

A language of formulas $L \subseteq \mathfrak{F}$ is polynomially provable iff there exists a polynomial-time Turing machine $\mathcal{P}$ such that for any $\phi \in L$

$$\mathcal{P}(\mathsf{code}_P(\phi)) = \mathsf{code}_P(\pi) \qquad \wedge \qquad \mathcal{V}(\mathsf{code}_P(\phi), \mathsf{code}_P(\pi)) = 1.$$

The prover $\mathcal{P}$ must be polynomial w.r.t. to each input $x \in \mathbb{N}$.

- The latter is not restrction when L is polynomially decidable.
- The complexity measure $\textsc{size}(x)$ can be specified in a language specific way as long $\textsc{size}(x) = \mathcal{O}(|x|)$.
- The prover $\mathcal{P}$ may fail for some or all instances $\psi \notin L$.
- The verifier $\mathcal{V}$ must be also polynomial w.r.t. the input size.

Restriction to a single instance

An instance ϕ from a language of formulas $L \subseteq \mathfrak{F}$ is polynomially provable by a polynomial-time Turing machine $\mathcal{P}$ iff

$$\mathcal{P}(\mathsf{code}_P(\phi)) = \mathsf{code}_P(\pi) \qquad \wedge \qquad \mathcal{V}(\mathsf{code}_P(\phi), \mathsf{code}_P(\pi)) = 1.$$

The corresponding notation $\mathcal{T} \wedge \mathcal{P} \Vdash \phi$.

We can treat it as a two argument predicate $[\![\mathcal{T} \wedge \_ \Vdash \_\ ]\!]$ that maps

$$\big(\mathsf{code}_{\mathcal{U}}(\mathcal{P}), \mathsf{code}_P(\phi)\big) \mapsto \mathcal{V}(\phi, \mathcal{P}(\phi))$$

For clarity, we use $[\![\mathcal{T} \wedge \mathcal{P} \Vdash \phi]\!]$ instead of $[\![\mathcal{T} \wedge \mathsf{code}_{\mathcal{U}}(\mathcal{P}) \Vdash \mathsf{code}_P(\phi)]\!]$.

Efficient representations

Let $\rho_r(x_1, \ldots, x_k)$ be a formula that represents a relation $r \subseteq \mathbb{N}^k$. Then ρ_r is an efficient representation of r iff languages

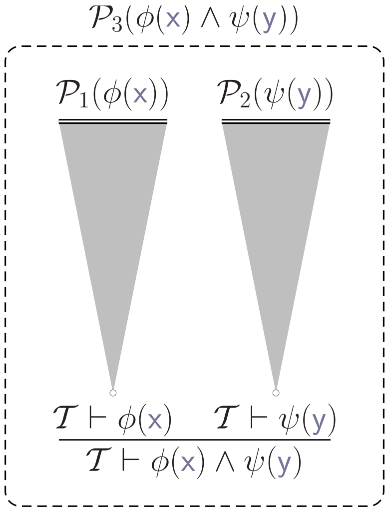
$$\begin{aligned} L_\phi &= \{\ \ \rho_r(\mathsf{x}_1, \ldots, \mathsf{x}_k) : (x_1, \ldots, x_k) \in r\} \\ L_{\neg\phi} &= \{\neg\rho_r(\mathsf{x}_1, \ldots, \mathsf{x}_k) : (x_1, \ldots, x_k) \notin r\} \end{aligned}$$

are polynomial-time provable.

Theorem. *Any poly-time computable predicate is efficiently representable.*

Proof. Extension of Gödel β-function approach with computationally efficient $\boldsymbol{\beta}_{\mathsf{e}}$ is sufficient. The fact was already noted by Cook 1971 in the $\mathcal{NP}$-completeness proof of SAT, however Okamoto and Kashima provide an unreadable proof which uses circuit evaluation instead. □

How to grow a proof tree?



Lemma. *If there are polynomial-time provers* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *then there exists a polynomial-time prover* $\mathcal{P}_3$ *such that*

$$\mathrm{PA}^{\mathsf{e}} \vdash \forall x \forall y [\![\mathcal{T} \wedge \mathcal{P}_1 \Vdash \phi(\mathsf{x})]\!] \wedge [\![\mathcal{T} \wedge \mathcal{P}_2 \Vdash \psi(\mathsf{y})]\!] \supset [\![\mathcal{T} \wedge \mathcal{P}_3 \Vdash \phi(\mathsf{x}) \wedge \psi(\mathsf{x})]\!].$$

Further conclusions

Theorem. *Polynomial provability is closed under elementary proof steps.*

Lemma. *For any formula* $\phi(x_1, \ldots, x_k) \in \mathfrak{F}$ *and for any polynomial-time prover* $\mathcal{P}$*, the predicate*

$$[\![\mathcal{T} \wedge \mathcal{P} \Vdash \phi(\mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{k})]\!]$$

has an efficient representation w.r.t. input parameters $x_1, \ldots, x_k$.

Lemma. *Let* ρ_r *be a canonical efficient representation of a relation* $r \subseteq \mathbb{N}$. *Then there exists a polynomial-time prover* $\mathcal{P}$ *such that*

$$\mathrm{PA}^{\mathsf{e}} \vdash \forall x(\rho_r(x) \sim [\![\mathrm{PA}^{\mathsf{e}} \wedge \mathcal{P} \Vdash \rho_r(\mathsf{x})]\!]).$$

Proof. We must prove that correct code interpretaton is possible. $\square$

Polynomial-time Recursion Theorem

Theorem. *For any* $m \in \mathbb{N}$ *and* $c_1 \in \mathbb{N}$ *there exist a code-constant* k *and a time-bound constant* $c_2 > c_1$ *such that*

$$\mathrm{PA}^{\mathsf{e}} \vdash \forall w(\rho_{\mathsf{p\text{-}utm\text{-}p}}(\mathsf{k}, \mathsf{c}_2, \mathsf{w}) \sim \rho_{\mathsf{p\text{-}utm\text{-}p}}(\mathsf{m}, \mathsf{c}_1, \mathsf{k}, \mathsf{w})).$$

Proof. Let $k = \mathsf{code}_{\mathcal{U}}(\mathcal{K})$ where $\mathcal{K}$ executes following steps:

1. Write m to the working tape.
2. Copy its own code k to the working tape.
3. Copy the inputs w to the working tape.
4. Interptete the input (m, c_1, k, w) as universal Turing machine $\mathcal{U}_{\mathsf{p}}$.

Tatsuaki and Kashima fail to recognise the differnce in degrees $c_2 > c_1$.

Gödel sentences

Lemma. *For any polynomial-time Turing machine $\mathcal{M}$ there exist a formula $\rho_{\mathcal{M}}$ such that*

$$\mathrm{PA}^{\mathsf{e}} \vdash \forall w(\rho_{\mathcal{M}}(\mathsf{w}) \sim \neg[\![\mathrm{PA}^{\mathsf{e}} \wedge \mathcal{M} \Vdash \rho_{\mathcal{M}}(\mathsf{w})]\!]$$

For all x the formula $\rho_{\mathcal{M}}$ is called a Gödel sentence with respect to $\mathcal{M}$.

Proof. Consider a Turing machine $\mathcal{K}(w)$:

- Construct the formula $\rho_{\text{p-utm-p}}(\mathsf{k}, \mathsf{c}_1, \mathsf{w})$ for a <u>cleverly chosen</u> c_1.
- Test $\mathcal{V}(\rho_{\text{p-utm-p}}(\mathsf{k}, \mathsf{c}_1, \mathsf{w}), \mathcal{M}(\rho_{\text{p-utm-p}}(\mathsf{k}, \mathsf{c}_1, \mathsf{w}))=1$.
- Return $\neg[\![\mathrm{PA}^{\mathsf{e}} \wedge \mathcal{M} \Vdash \rho_{\text{p-utm-p}}(\mathsf{k}, \mathsf{c}_1, \mathsf{w})]\!]$.

The lemma can be proven, althought it must be done more carefully than in the article—explicit degree bounds are a big nuisance.

Incompleteness theorems

Theorem (First Incompleteness Theorem). *Let $\mathcal{M}$ be a polynomial-time Turing machine and $\rho_{\mathcal{M}}(w)$ the corresponding Gödel sentence. Then for all inputs $w \in \mathbb{N}$*

$$\mathrm{PA}^{\mathsf{e}} \wedge \mathcal{M} \nVdash \rho_{\mathcal{M}}(\mathsf{w})$$

unless PA^{e} *is inconsistent.*

Theorem (Second Incompleteness Theorem). *Let $\phi(w) \in \mathfrak{F}$ with a single free variable w and $\mathcal{M}$ a polynomial-time Turing machine. Then there exists a Turing machine $\mathcal{M}_\circ$ such that for all $w \in \mathbb{N}$*

$$\mathrm{PA}^{\mathsf{e}} \wedge \mathcal{M} \nVdash \neg[\![\mathrm{PA}^{\mathsf{e}} \wedge \mathcal{M}_\circ \Vdash \phi(\mathsf{w})]\!]$$

unless PA^{e} *is inconsistent.*

These theorems are completely useless for proving $\mathcal{P} \neq \mathcal{NP}$.

Language of satisfiable 3CNF formulas

Introducing propositional variables $X_i \equiv x_i = 1$ and $\neg X_i \equiv \neg(x_i = 1)$.

Language L_{3SAT} of 3CNF formulas is a subset of $\mathfrak{F}$, and we define

$$x \in r_{\mathsf{3SAT}} \qquad \Longleftrightarrow \qquad x = \mathsf{code}_P(\phi) \wedge \phi \in L_{3CNF}$$

$$\textsc{size}(x) = \begin{cases} 2n, & \text{if } \phi \in L_{\mathsf{3SAT}}, \\ 2 \cdot |x| & \text{otherwise.} \end{cases}$$

Let ρ_{3SAT} be the canonical but inefficient representation of r_{3SAT}.

Now, we have to gear our theory towards polynomial-time descisions instead of proofs.

Polynomially descidable predicates

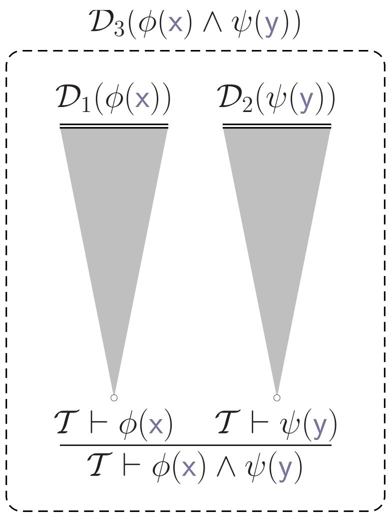
A Turing machine $\mathcal{M}$ correctly accepts, rejects or decides predicate ϕ iff

Condition	Predicate	Equivalent
$\mathrm{PA} \models \mathcal{M}(\phi) \supset \phi$	$[\![\mathrm{PA} \models \mathcal{M}(\phi) \supset \phi]\!]$	$\mathcal{M}(\phi) \supset \phi$
$\mathrm{PA} \models \neg\mathcal{M}(\phi) \supset \neg\phi$	$[\![\mathrm{PA} \models \neg\mathcal{M}(\phi) \supset \neg\phi]\!]$	$\neg\mathcal{M}(\phi) \supset \neg\phi$
$\mathrm{PA} \models \mathcal{M}(\phi) \sim \phi$	$[\![\mathrm{PA} \models \mathcal{M}(\phi) \sim \phi]\!]$	$\mathcal{M}(\phi) \sim \phi$

Consider only descidable predicates in canonical form—(efficient) predicate encoding that corresponds to a distingusher. Lets call them *simple* formulas.

Theorem. *All polynomially descidable predicates have efficient simple representation.*

How to grow a descision tree?



Lemma. *If there are polynomial-time distinguishers* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *then there exists a polynomial-time distinguisher* $\mathcal{D}_3$ *such that*

$$\begin{aligned}\mathrm{PA}^{\mathsf{e}} \vdash \forall x \forall y ([\![\mathcal{D}_1(\phi(\mathsf{x})) \sim \phi(\mathsf{x})]\!] \wedge [\![\mathcal{D}_2(\psi(\mathsf{y})) \sim \psi(\mathsf{y})]\!] \\ \supset [\![\mathcal{D}_3(\phi(\mathsf{x}) \wedge \psi(\mathsf{y})) \sim \phi(\mathsf{x}) \wedge \psi(\mathsf{x})]\!]).\end{aligned}$$

Further conclusions

Lemma. *There exists a universal prover* $\mathcal{P}_\circ$ *for a simple predicate* $\rho(\mathsf{x})$ *always outputs either a proof of* $\rho(\mathsf{x})$ *or a proof of* $\neg\rho(\mathsf{x})$.

Remark. *If the simple predicate is in an efficient representation, the working time of* $\mathcal{P}_\circ$ *is polynomial.*

Theorem. *Polynomial-time descidability is closed under elementary proof steps.*

Theorem. *If the predicate is simple, then correctness of descisions is provable in* PA^{e}. *For efficient simple predicates, the working time of the prover is polynomial.*

Gödel sentences

Lemma. *For any polynomial-time accepting-rejecting Turing machine $\mathcal{M}$ there exist an <u>efficient</u> simple predicate $\rho_{\mathcal{M}}$ such that*

$$\mathrm{PA}^{\mathsf{e}} \vdash \forall w(\ \ \rho_{\mathcal{M}}(\mathsf{w}) \sim \neg[\![\ \ \mathcal{M}(\rho_{\mathcal{M}}(\mathsf{w}))]\!])$$
$$\mathrm{PA}^{\mathsf{e}} \vdash \forall w(\neg\rho_{\mathcal{M}}(\mathsf{w}) \sim \neg[\![\neg\mathcal{M}(\rho_{\mathcal{M}}(\mathsf{w}))]\!])$$

For all x the formula $\rho_{\mathcal{M}}$ is called a Gödel sentence with respect to $\mathcal{M}$.

Proof. Consider a Turing machine $\mathcal{K}$:

- Loads its own code k.
- Constructs the formula $\rho_{\mathsf{utm\text{-}p}}(\mathsf{k}, \mathsf{w})$.
- Outputs $\neg\mathcal{M}(\rho_{\mathsf{p\text{-}utm\text{-}p}}(\mathsf{k}, \mathsf{w}))$.

Incompleteness theorems

Theorem (First Incompleteness Theorem). *A polynomial-time Turing machine $\mathcal{M}$ cannot correctly descide any instance $\rho_{\mathcal{M}}(w)$ of the corresponding Gödel sentence.*

$$\mathrm{PA}^{\mathsf{e}} \vdash \neg[\![\ \ \mathcal{M}(\rho_{\mathcal{M}}(\mathsf{w})) \supset \ \ \rho_{\mathcal{M}}(\mathsf{w})]\!]$$

$$\mathrm{PA}^{\mathsf{e}} \vdash \neg[\![\neg\mathcal{M}(\rho_{\mathcal{M}}(\mathsf{w})) \supset \neg\rho_{\mathcal{M}}(\mathsf{w})]\!]$$

unless PA^{e} *is inconsistent.*

Theorem (Second Incompleteness Theorem). *Let $\phi(w) \in \mathfrak{F}$ be a simple predicate. Then for any polynomial-time Turing machine $\mathcal{M}$, we can construct a polynomial-time Turing machine $\mathcal{M}_\circ$ such that for all $w \in \mathbb{N}$*

$$\mathrm{PA}^{\mathsf{e}} \wedge \mathcal{M} \not\vdash \neg[\![\mathcal{M}_\circ(\phi(\mathsf{w})) \sim \phi(\mathsf{w})]\!]$$

unless PA^{e} *is inconsistent.*

Towards the proof

Lemma. *Let $\rho_{\mathcal{M}_\circ}$ be a Gödel centense w.r.t. polynomial-time Turing machine $\mathcal{M}_\circ$. Then there exists a polynomial-time Turing machine $\mathcal{M}_\star$ such that*

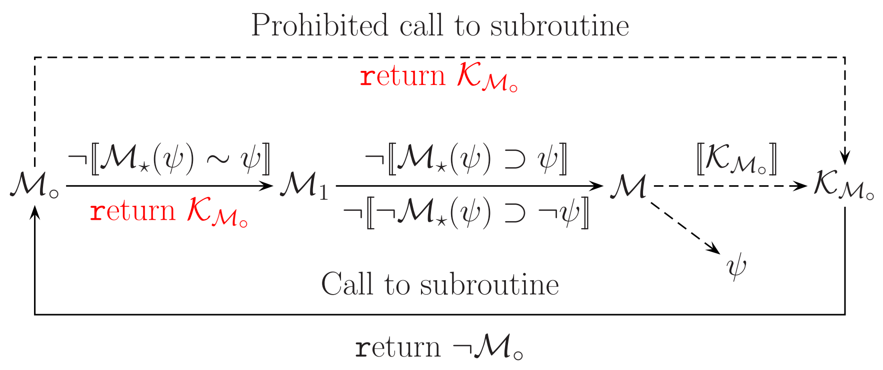
$$\mathrm{PA}^{\mathsf{e}} \vdash \forall w(\neg[\![\ \mathcal{M}_\star(\psi(\mathsf{w})) \supset \ \psi(\mathsf{w})]\!] \supset \ \rho_{\mathcal{M}_\circ}(\mathsf{w}))$$

$$\mathrm{PA}^{\mathsf{e}} \vdash \forall w(\neg[\![\neg\mathcal{M}_\star(\psi(\mathsf{w})) \supset \neg\psi(\mathsf{w})]\!] \supset \neg\rho_{\mathcal{M}_\circ}(\mathsf{w}))$$

Proof.

- $\mathcal{M}_\star$ computes and outputs predicate $\rho_{\mathcal{M}_\circ}(\mathsf{w})$.
- By the construction Gödel sentences are efficiently computable, therefore $\mathcal{M}_\star$ runs in polynomial-time.
- The claims are obvious and can be formally proved.

Construction of the magic $\mathcal{M}_\circ$



- $\mathcal{M}_\circ$ passes descision of $\neg[\![\mathcal{M}_\star(\psi(\mathsf{w})) \sim \psi(\mathsf{w})]\!]$ to $\mathcal{M}_1$. Here $\mathcal{M}_\star(\psi(\mathsf{w})) = \mathcal{K}_{\mathcal{M}_\circ}(w)$.
- $\mathcal{M}_1$ passes it further to $\mathcal{M}$ that has to "execute" $\mathcal{K}_{\mathcal{M}_\circ}(w)$ and compute $\psi(\mathsf{w})$. If $\mathcal{M}$ gets a provably correct result, it reveals $\rho_{\mathcal{M}_\circ}(\mathsf{w})$.
- Thus $\mathcal{M}_\circ$ has executed the prohibited call.

The actual proof is more involved—one has to reach zen-state to grasp all details and verify the 'construction, but it is doable!

Implication of Second Incompleteness Theorem

There is no polynomial-time prover $\mathcal{P}$ that could prove for all polynomial-time Turing machines $\mathcal{D}$ that they make incorrect descisions.

- Exact polynomial complexity may depend on the Turing machine $\mathcal{D}$.
- Result indicates that for a constructive proof of $\mathcal{P} \neq \mathcal{NP}$ we have to use at least super-polynomial prover $\mathcal{P}$ to generate counter examples for a concrete candidate distinguisher $\mathcal{D}$.
- The result does not indicate that there is no provably totally recursive counter example generator for L_{3SAT} distinquishers.
- The bound is quite natural, as for generating counter examples the prover $\mathcal{P}$ has to "evaluate" 3SAT formulas.

Computational content of $\mathcal{P} \neq \mathcal{NP}$ proof

The proof of $\mathcal{P} \neq \mathcal{NP}$ is equivalent to

$$\mathrm{PA}^{\mathrm{e}} \vdash \forall \mathcal{M}\ \forall n\ \exists w \geq n\ \neg[\![\mathcal{M}(\rho_{\mathsf{3SAT}}(\mathsf{w})) \sim \rho_{\mathsf{3SAT}}(\mathsf{w})]\!]$$

The latter <u>does not apriori</u> mean that given $\mathcal{M}$ and n the counter example w can be computed in polynomial-time in $|n|$.

If the proof is non-constructive then there might be no hints how to compute w at all.

Hence even if we have a proof $\mathrm{PA}^{\mathrm{e}} \vdash \exists w \geq n\ \neg[\![\mathcal{M}(\rho_{\mathsf{3SAT}}(\mathsf{w})) \sim \rho_{\mathsf{3SAT}}(\mathsf{w})]\!]$ we might be unable to pin-point w. Still it is trivial to prove it in polynomial-time w.r.t. formula length, if we have finite proof of

$$\mathrm{PA}^{\mathrm{e}} \vdash \forall \mathcal{M}\ \forall n\ \exists w \geq n\ \neg[\![\mathcal{M}(\rho_{\mathsf{3SAT}}(\mathsf{w})) \sim \rho_{\mathsf{3SAT}}(\mathsf{w})]\!].$$

Unjustified and questionable assumption

Definition. *The theory $\mathcal{T}$ is an polynomially ω-consistent w.r.t. two argument descidable predicate $\psi(\mathcal{M}, w)$ iff the following holds:*

- *Let $\mathcal{P}$ be a polynomial-time prover such that for any $\mathcal{M}$ there exists an infinite set $\{m_1, m_2 \ldots\} \subseteq \mathbb{N}$ so that*

$$\mathcal{T} \wedge \mathcal{P} \Vdash \exists w \geq m_i \ \psi(\mathcal{M}, w)$$

- *Then there must exist another polynomial-time prover $\mathcal{P}_\circ$ such that for any $\mathcal{M}$ there exist a constant c an infinite set $\{n_1, n_2 \ldots\} \subseteq \mathbb{N}$ so that*

$$\mathrm{PA}^{\mathsf{e}} \wedge \mathcal{P} \Vdash \exists w(\mathsf{n_i} \leq w < \mathsf{m_i} + |\mathsf{n_i}|^c) \ \psi(\mathcal{M}, w)$$

Computational content of polynomial ω-consistency

It is rather hard or even impossible to link polynomial ω-consistency with any other logic concept. Thus, we provide *ad hoc* interpretation.

Intuitively, polynomial ω-consistency explicity states that any proof:

- Is constructive or has an extractable explicit computational content.
- The corresponding algorithm has a polynomial complexity.

Under these circumstances unprovability of $\mathcal{P} \neq \mathcal{NP}$ is evident.

Since Peano Arithmetics is not proven to be polynomial ω-consistent, there is essentially no progress.

It would be trully surprising if Peano Arithmetics is polynomially ω-consistent.