Incompleteness Revisited

Harry Sha

Fall 2026

Review: Theory, Consequences, Theorems

The *theory of a structure* $\mathcal{M}$ is

$$\mathrm{Th}(\mathcal{M}) = \{A : A \text{ is a sentence and } \mathcal{M} \models A\}.$$

The set of *theorems* of Σ is

$$\mathrm{Thm}(\Sigma) = \{A : A \text{ is a sentence and } \Sigma \vdash A\}.$$

The set of *consequences* of Σ is

$$\mathrm{Cn}(\Sigma) = \{A : A \text{ is a sentence and } \Sigma \models A\}.$$

Soundness and completeness imply $\mathrm{Thm}(\Sigma) = \mathrm{Cn}(\Sigma)$.

Axiomatization

A *theory* T is a set of sentences in a fixed language.

A set of sentences Γ *axiomatizes* T if

$$\mathrm{Cn}(\Gamma) = \mathrm{Cn}(T).$$

T is *effectively axiomatizable* if such a Γ can be chosen Turing-recognizable: $\Gamma \in \mathbf{TR}$.

Effective axioms

If Γ is an effective axiomatization of T, then $\mathrm{Thm}(T) \in \mathbf{TR}$.

WTS $\mathrm{Thm}(T) \in TR$. Here's a recognizer:

$f(\varphi)$:
run the backwards proof search alg.
Since $\Gamma \in TR$, it has an enumerator,
which is used to retrieve axioms in Γ to put in
the antecedent.

Then, by completness/soundness, backwards search finds the proof!

Section 1

Incompleteness Revisited

Incompleteness v1 (Informal)

No formal system can simultaneously have all four of the following properties:

1. **Completeness**. Every true mathematical statement is provable.
2. **Soundness**. Every provable statement is true.
3. **Verifiable proofs**. An algorithm can decide whether a proposed proof is valid.
4. **Expressivity**. The system can describe enough of mathematics, including the operations of a Turing machine.

Incompleteness v1 (Formal)

Let Σ be a set of sentences, and $\mathcal{N}$ be the standard model (more on this later). Then, the following cannot be satisfied simultaneously.

1. **Completeness**. $\mathcal{N} \models A \implies \Sigma \vdash A$
2. **Soundness**. $\Sigma \vdash A \implies \mathcal{N} \models A$
3. **Recognizable axioms**. $\Sigma \in \mathbf{TR}$
4. **Expressivity**. For each TM M, there is a formula φ_M such that $\mathcal{N} \models \varphi_M$ if and only if M halts on the emptystring.

Proof

By contradiction, suppose that all 4 conditions hold. Let $T = \mathrm{Thm}(\Sigma)$ **Claim 1.** T is decidable **Claim 2.** T is not decidable

The two claims together form the contradiction.

Claim 1. T is decidable

on input φ:

1. run backwards proof search on φ
2. run backwards proof search on $\neg\varphi$

} in parallel.

if 1 finds a proof, accept
if 2 finds a proof, reject

Firstly, we won't find a proof for both φ, and $\neg\varphi$, else $\mathcal{N} \models \varphi$, $\mathcal{N} \models \neg\varphi$ by 2, which is impossible. If $\varphi \in T$, then $\Sigma \vdash \varphi$, so 1 eventually finds a proof (using 3). If $\varphi \notin T$, then, $\Sigma \nvdash \varphi \overset{①}{\Rightarrow} \mathcal{N} \not\models \varphi \Rightarrow \mathcal{N} \models \neg\varphi \overset{①}{\Rightarrow} \Sigma \vdash \neg\varphi$.

Claim 2. T is not decidable

The map $M \to \varphi_M$ is a reduction from HALTe.

WTS: M halts (on ε) $\iff \exists \vdash \varphi_M$.

We have M halts on $\varepsilon \iff \mathbb{N} \models \varphi_M$ (4)

$\iff \exists \vdash \varphi_M$ (1, 2).

Since HALTe is not decidable, neither is T ∎

Section 2

Arithmetic

Incompleteness v1 (Formal, stronger)

Instead of assuming the expressivity property, we'll show that expressing arithmetic suffices.

Let Σ be a set of sentences in the language of arithmetic, and $\mathcal{N}$ be the standard model. Then, the following cannot be satisfied simultaneously.

1. **Completeness**. For all A, $\Sigma \vdash A$ or $\Sigma \vdash \neg A$
2. **Soundness**. $\mathcal{N} \models \Sigma$
3. **Recognizable axioms**. $\Sigma \in \mathbf{TR}$

The goal of the rest of this section is to construct φ_M as a sentence in this language. Satisfying the property we needed previously: $\mathcal{N} \models \varphi_M$ if and only if M halts on the empty string.

Note that the completeness and soundness conditions have also changed slightly but are implied by the conditions from the previous statement.

Language of Arithmetic

$$\mathcal{L}_E = \{0, S, <, +, \cdot, E\}.$$

- Constant: 0.
- Unary function: S (successor).
- Binary functions: $+$, $\cdot$, and E (exponentiation).
- Binary relation: $<$.

We may be loose and write x^y for $E(x, y)$, and $x + 1$ for $S(x)$ etc...

Question: What are the terms in this language?

$0, \quad S(0), \quad S(S(0)), \quad x + S(0)$

$E(0, S(0)) + S(y) \quad \ldots$ etc.

Numerals

$$\overline{0} = 0, \quad \overline{1} = S(0), \quad \overline{2} = S(S(0))$$

$$\ldots \quad \overline{n} = \underbrace{S \ldots S}_{n}(0)$$

for any $n \in \mathbb{N}$, the numeral, $\overline{n}$, is the term corresponding to n. Note, numerals are terms.

Standard Model

The *standard model of arithmetic* $\mathcal{N}$ has domain $\mathbb{N} = \{0, 1, 2, \ldots\}$, with all functions and relations taking their standard meanings. E.g. $+^{\mathcal{N}}$ is actual addition in $\mathbb{N}$.

We take $m^0 = 1$, including $0^0 = 1$.

Section 3

Arithmetization

Goal

Construct φ_M is a sentence in the language of arithmetic

φ_M loosely

φ_M : M halts on ε.

↓

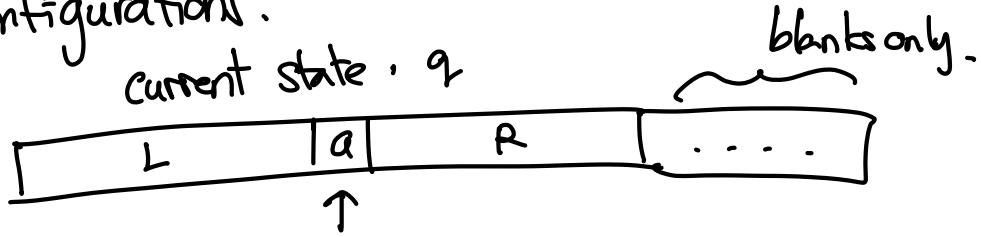
$\exists t$, after t steps M halts on ε.

↓

$\exists t \exists (C_0, \ldots, C_t)$. s.t. C_0 is a starting-config ∧
C_t is a halting config ∧
$\forall i < t$, C_{i+1} follows from C_i

each of the C_i codes a config

encoding configurations.



encode as a tuple: (q, L, a, R).

- q: current state
- L: string before
- a: symbol @ current tape head pos.
- R: string after,

problem: tuples aren't numbers!,
neither are the strings, L, R!

we'll deal w/ this in a bit, first let's

One step of a TM

e.g. Current config: (q, L, a, R), and

$\delta(q, a) = (q', a', \text{RIGHT})$

The next config should be : $(q', L{+}a', R[0], R[1:])$

We can then define a formula : $\text{Step}(C_i, C_{i+1})$ that says C_{i+1} is the config. that follows from C_i. if $C_i = (q, L, a, R)$ $C_{i+1} = (q', L', a', R')$.

$$\bigvee_{\delta(q'', a'') = (q''', a''', \text{RIGHT})} q = q'' \wedge q' = q''' \wedge L' = L{+}a''' \wedge a = a'' \wedge a' = R[0] \wedge R' = R[1:] \quad \bigvee \text{ the left move case ...}$$

What we need

Essentially we need to show how to represent sequences and operations on those sequences using just numbers!

Sequences over a fixed alphabet

Suppose the alphabet has k elements, wLOG, let's say $\{0, \ldots, k-1\}$.

represent a sequence as follows.

$$t_{n-1}\, t_{n-2} \ldots t_0$$

$\downarrow$ prepend k,

$$k\, t_{n-1}\, t_{n-2} \ldots t_0$$

$\downarrow$ interpret base $b = k+1$

$$k\, b^n + \sum_{i=0}^{n-1} t_i\, b^i$$

this is a single number!

note: adding k, keeps the encoding injective, else you have problems w/ leading 0s.

Sequences over an arbitrary numbers

The previous is enough to store the tape contents since those are sequences over a fixed alphabet. But a particular string can be arbitrarily large, so we might have problems encoding a configuration, (q, L, a, R), (what base should I use?)

Start by introducing a way to pair numbers: $\text{pair}(x, y) = 2^x(2y+1)$, we can check if $z = \text{pair}(x, y)$ w/ the following formula:

$$\text{Pair}(x, y, z) : z = 2^x(2y+1).$$

So now we could do $\langle q, L, a, R \rangle$ as

$$\text{pair}(q, (\text{pair}(L, \text{pair}(a, R))))$$

if we then wanted to do arbitrary length sequences of arbitrary sized numbers we can combine the two approaches:

For a sequence $t_{n-1}, \ldots, t_1$,

find a large enough base, b, and then encode the sequence w/ the base as a pair!

Suppose b is some base that works for the sequence $t_k, \ldots, t_1$. Encode the sequence in base b, to get an encoding, u, and then represent that as a single number using $\text{pair}(b, u)$.

We can now represent sequences of configurations!

Selected operations

Suppose b is a base and u is a number the sequence it represents is $u_n u_{n-1} \cdots u_0$: $u = \sum_{i=0}^{n} u_i b^i$

Shorthand, I really needed to use another $\exists$, w/ the previous formula.

$\text{Len}(b,u,\ell)$: $b^{\ell-1} \leq u < b^\ell$ "$\text{len}(u)$ is ℓ".

$\exists \text{Elem}(b,u,i,c)$. "$u_i = c$" example, base 10.

$b=10$, $u=4852$, $i=1$ (want $c=5$).

$u = 48 \cdot 10^2 + 5 \cdot 10 + 2$

w c y

$\exists w,y.\ y < b^i \wedge w < b^{\text{len}(u)-i-1} \wedge c < b \wedge u = w \cdot b^{\text{len}(u)-i-1} + cb^i + y$

I may be missing a few checks, but you get the point!

Takeaway: Can encode sequence & simulate sequence ops. using arithmetic!

For a given TM, M, can define the formulas:

$Hist_M(x, h, t)$: "h encodes the sequence of t configurations when M is run on x"

note, x here is the number encoding the input sequence! (not the sequence itself).

we also get

- $AccHist_M(x,h,t)$
- $RejHist_M(x,h,t)$.
- $HaltHist_M(x,h,t) = AccHist_M(x,h,t) \vee RejHist_M(x,h,t)$
- $Halts_M(x) = \exists t,h \; HaltHist_M(x,h,t)$
- $Accepts_M(x) = \exists t,h \; AccHist_M(x,h,t)$ ... etc!

In the standard model, all arithmetic operations are interpreted as we intended, so $\mathbb{N} \models Halts_M(x) \iff M$ halts on x.