MATH3306

Set Theory & Mathematical Logic

Semester 2, 2022

Professor Benjamin Burton

Rohan Boas

Contents

1 Introductory notes

1.1 Gödel's incompleteness theorem

Theorem 1.1.1 (Gödel's incompleteness theorem, informal version)**.** *There are true mathematical statements that cannot be proven.*

"Proof". Take the statement "This statement has no proof."
Assume it is false. This implies that the statement has a proof. If the statement has a proof, it must be true, contradiction!
Assume it is true. This implies that the statement has no proof. Therefore, the statement cannot be proven. □

1.2 The halting problem

We would like to be able to know if an algorithm or program will halt or will loop forever. Can we write an algorithm which can tell us whether or not a given program will halt on a given input? This is known as the halting problem. The halting problem is undecidable.

Proof by contradiction. Say there does exist some program H which solves the halting problem. Let us define H. H takes as inputs a program, x, and an input for that program, y.

$$\mathrm{H}(x, y) = \begin{cases} \text{YES}, & \text{if } x \text{ halts on input } y \\ \text{NO}, & \text{if } x \text{ loops forever on input } y \end{cases}$$

Let us define a program Foo.

$$\mathrm{Foo}(x) = \begin{cases} \text{loops forever}, & \text{if } \mathrm{H}(x, x) \text{ is YES} \\ \text{halts}, & \text{if } \mathrm{H}(x, x) \text{ is NO} \end{cases}$$

TODO: finish proof, maybe rewrite with diagram □

1.3 Defining algorithms

In defining algorithms, Turing machines and recursive functions will be the primary focus. Grammars and code are also alternatives.

Definition 1.1 (Church-Turing thesis, informal version)**.** Any reasonable definitions of "algorithm" are equivalent.

2 Finite and deterministic state automata

2.1 Finite state automata

Definition 2.1 (Alphabet)**.** An alphabet A is a finite set of symbols.

Definition 2.2 (Word)**.** A word is a sequence of symbols from A.

Theorem 2.1.1. *Words can be concatenated. E.g. for words α representing "bob", and β representing "cat", $\alpha\beta\alpha$ represents "bobcatbob".*

Theorem 2.1.2. *The set of words length m is $A^m = A \times A \times \cdots \times A$.*

Theorem 2.1.3. *The empty word (i.e. of length 0) is ε.*

Theorem 2.1.4. *A^* is the set of all words over A. $A^* = \bigcup_{m \geq 0} A^m$.*

Theorem 2.1.5. *A^+ is the set of all non-empty words over A. $A^+ = \bigcup_{m \geq 1} A^m$.*

Definition 2.3 (Language)**.** A language is a subset of A^*

Definition 2.4 (Finite state automata)**.** A finite state automaton (FSA) can be defined as the 5-tuple (Q, F, A, τ, q_o) where

- Q is a finite set of states,
- $F \subseteq Q$ is the set of final/accepting states,
- A is a finite alphabet,
- $\tau \subseteq Q \times A \times Q$ is the set of transitions, and
- q_0 is the initial state.

Definition 2.5 (Computation)**.** A computation is a sequence $q_0 a_1 q_1 a_2 \ldots a_n q_n$ such that each $q_i a_{i+1} q_{i+1} \in \tau$.

Definition 2.6 (Successful)**.** A computation is successful if $q_n \in F$.

Definition 2.7 (Accepted)**.** A word $\alpha = a_1 a_2 \ldots a_n$ is accepted by the FSA if there is a successful computation $q_0 a_1 q_1 a_2 \ldots a_n q_n$.

Definition 2.8 (Recognised)**.** The language recognised by an FSA is the set of all words it accepts.

Note 2.1.1. A FSA is like a backtracking search.

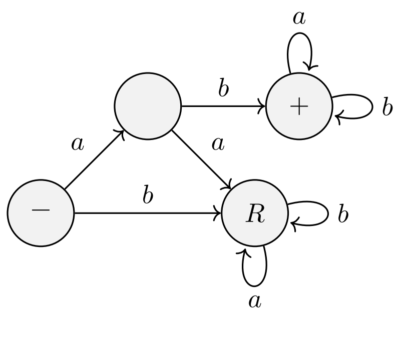

Figure 1: A simple FSA (that is also a DFA)
$-$ denotes the initial state, $+$ is an accepting state, and R is a rejection state/"black hole".

2.2 Deterministic finite state automata

Definition 2.9 (Deterministic finite state automata)**.** A deterministic state automaton (DFA) is an FSA where $\forall q \in Q, \forall a \in A, \exists! q' \in Q$ s.t. $(q, a, q') \in \tau$

Theorem 2.2.1. *The definition of DFA implies there exists a function* $\delta : Q \times A \to A$*.*

Theorem 2.2.2. *FSA and DFA solve the same problems.*
A DFA is already an FSA, and an FSA can be represented by a DFA of the reachable sets of states. Though, for an FSA with n *states, the corresponding DFA has up to* 2^n *states.*

Theorem 2.2.3. *FSA/DFA can perform addition.*

Theorem 2.2.4. *FSA/DFA* cannot *perform multiplication.*

Theorem 2.2.5. *To recognise an infinite language of a DFA with a finite complement, find the complement and swap accepting and rejecting states.*

3 Turing machines

Definition 3.1 (Turing machine)**.** A Turing machine (TM) is a FSA with infinite memory in the form of a tape.
A Turing machine is a tuple $(Q, F, \sqcup, A, I, \tau, q_0)$ where

- Q is a finite set of states,
- $F \subseteq Q$ is the set of final/accepting states,
- $\sqcup$ is the "blank" symbol which is present on the tape where no new symbol has been written,
- A is a finite alphabet of all symbols that may be on the tape (including $\sqcup$),
- $I \subseteq A - \{\sqcup\}$ is a finite alphabet of all symbols from the input sequence,
- $\tau \subseteq Q \times A \times Q \times A \times \{L, R\}$ is the set of transitions, and
- q_0 is the initial state.

Definition 3.2 (Tape)**.** A tape is a tuple (a, α, β) where

- $a \in A$,
- $\alpha, \beta : \mathbb{N} \to A$, and
- $\alpha(i), \beta(i) \neq \sqcup$ for only finitely many i.

Definition 3.3 (Configuration)**.** A configuration is a snapshot of the state.
A configuration is a tuple $(q \in Q, a, \alpha, \beta)$.

Definition 3.4 (Reachable)**.** A configuration c' is reachable from c in a single move if for $c = (q, a, \alpha, \beta)$, (in the case of moving to the right) $c' = (q', a(0), \alpha', \beta')$ where

- $\alpha'(i) = \alpha(i+1) \quad \forall i \in \mathbb{N}$,
- $\beta'(i) = \begin{cases} \beta(i-1) & \forall i \geq 1 \\ a' & i = 0 \end{cases}$, and
- $(q, a, q', a', R) \in \tau$.

Definition 3.5 (Computation)**.** A computation is a finite sequence of configurations $c_0 c_1 c_2 \dots c_n$ s.t. c_i is reachable from c_{i-1} is a single move for all i.

Definition 3.6 (Terminal)**.** A configuration is terminal if no configuration is reachable from it. A Turing machine halts upon reaching such a configuration.

Definition 3.7 (Acceptinng)**.** A Turing machine accepts a word $w \in I^*$ if there exists some computation from the initial state c_w to some final state.

Definition 3.8 (Recognised)**.** The language recognised by a Turing machine is the set of all words $w \in I^*$ that the Turing machine accepts.

Definition 3.9 (Deterministic)**.** A Turing machine is deterministic if for all q and a, there exists at most one tuple $(q, a, q', a', d) \in \tau$.

Theorem 3.0.1. *The transitions of a Turing machine are partial functions of the form* $\delta : Q \times A \to Q \times A \times \{L, R\}$

4 Recursive functions

4.1 Base definitions

Definition 4.1 (Total function)**.** A total function is an ordinary function, i.e. $f : D \to C$ s.t. $f \subseteq D \times C$ where $\forall d \in D\, \exists! c \in C$ s.t. $(d, c) \in f$.

Definition 4.2 (Partial function)**.** A partial function is $f : D \to C$ s.t. $f \subseteq D \times C$ where $\forall d \in D\, \exists_{\leq 1} c \in C$ s.t. $(d, c) \in f$. I.e. undefined values are permitted.

Note 4.1.1. By convention, if not specified a function is $f : \mathbb{N}^r \to \mathbb{N}$.

4.2 Initial functions

- Zero: $z : \mathbb{N} \to \mathbb{N},\ z(n) = 0\ \forall n$
- Successor: $\sigma : \mathbb{N} \to \mathbb{N},\ \sigma(n) = n + 1\ \forall n$
- Projection: $\pi_{i,n} : \mathbb{N}^n \to \mathbb{N},\ \pi_{i,n}(k_1, \dots, k_n) = k_i$

Definition 4.3 (Composition)**.** Given $g : \mathbb{N}^r \to \mathbb{N}$, and $h_1, \dots, h_r : \mathbb{N}^n \to \mathbb{N}$, $f : \mathbb{N}^n \to \mathbb{N}$ is defined by

$$
\begin{aligned}
f(x_1, \dots, x_n) = g(& \\
& h_1(x_1, \dots, x_n), \\
& \quad \vdots \\
& h_r(x_1, \dots, x_n) \\
)&
\end{aligned}
$$

4.3 Primitive recursion

Definition 4.4 (Primitive recursion)**.** A primitive recursion on $g : \mathbb{N}^n \to \mathbb{N}$ and $h : \mathbb{N}^{n+2} \to \mathbb{N}$ is defined as $f : \mathbb{N}^{n+1} \to \mathbb{N}$ s.t.

$$
\begin{aligned}
f(x_1, \dots, x_n, 0) &= g(x_1, \dots, x_n) \\
f(x_1, \dots, x_n, y + 1) &= h(\\
& \qquad x_1, \dots, x_n, \\
& \qquad y, \\
& \qquad f(x_1, \dots, x_n, y + 1) \\
&)
\end{aligned}
$$

Note 4.3.1. Primitive recursion can only go from y to $y + 1$ and can only recurse over a signle variable.

Note 4.3.2. We can only return a single integer, however, we can store pairs etc. by combining them in a retrievable way, for example, $2^x \times 3^y$ could be chosen to store (x, y).

Theorem 4.3.1. *Primitive recursive functions are the smallest class of functions that contain the initial functions and is closed under composition and primitive recursrion.*

Theorem 4.3.2. *Addition can be represented as a primitive recusrion, $s(x,y) = x + y$, with*

$$s(x,0) = \pi_{1,1}(x)$$
$$s(x,y+1) = \sigma(s(x,y))$$

Theorem 4.3.3. *Multiplication can be represented as a primitive recusrion, $m(x,y) = x \times y$, with*

$$m(x,0) = z(x)$$
$$m(x,y+1) = s(x, m(x,y))$$

Definition 4.5 (Primitive recusrive definition of a function)**.** The primitive recusrive definition of a function $f : \mathbb{N}^n \to \mathbb{N}$ is a finite set of functions $\{f_0, \dots, f_k\}$ s.t.

$$\forall i,\ f_i = \begin{cases} \text{initial function} \\ \text{composition of functions from } \left\{f_j \mid j < i\right\} \\ \text{primitive recursion of functions from } \left\{f_j \mid j < i\right\} \end{cases}$$

Theorem 4.3.4. *A primitive recursion is equivalent to a primitive recusrive definition.*

Theorem 4.3.5. *The constant function is $c_i : \mathbb{N} \to \mathbb{N}$ where $c_i(x) = i$.*

$$c_i = \overbrace{\sigma\left(\sigma\left(\dots \sigma\left(z\right)\right)\right)}^{i \text{ times}}$$

Theorem 4.3.6. *The sign function $Sg : \mathbb{N} \to \mathbb{N}$ is*

$$Sg(x) = \begin{cases} 0, & x = 0 \\ 1, & x > 0 \end{cases}$$

Theorem 4.3.7. *The predecessor function $Pred : \mathbb{N} \to \mathbb{N}$ is*

$$Pred(x) = \begin{cases} 0, & x = 0 \\ x - 1, & x > 0 \end{cases}$$

Theorem 4.3.8. *The subtraction function $\dot{-} : \mathbb{N}^2 \to \mathbb{N}$ is*

$$x \dot{-} y = \begin{cases} 0, & x < y \\ x - y, & x \geq y \end{cases}$$

Theorem 4.3.9. *The absolute value function $|x - y| : \mathbb{N}^2 \to \mathbb{N}$ is*

$$|x - y| = \begin{cases} y - x, & x < y \\ x - y, & x \geq y \end{cases}$$

$$|x - y| = (x \dot{-} y) + (y \dot{-} x) = s\left(x \dot{-} y, y \dot{-} x\right)$$

Theorem 4.3.10. *The exponentiation function $x^y : \mathbb{N}^2 \to \mathbb{N}$ is*

$$x^0 = 1$$
$$x^{y+1} = m(x^y, x)$$

N.B. tetration is also primitive recursive.

Theorem 4.3.11. *For every function $f : \mathbb{N} \to \mathbb{N}$ there is a summation function $\sum_{i=0}^{x} f(i)$.*

Theorem 4.3.12. *If f is primitive recursive then $\sum_{i=0}^{x} f(i)$ is also primitive recusrive.*

Theorem 4.3.13. *For every function $f : \mathbb{N} \to \mathbb{N}$ there is a bounded minimisation function $\min_{i \leq y} f(i)$. This function gives the smallest $f(i) \neq 0$, and gives $y+1$ if all $f(i) = 0$.*

Note 4.3.3. The following are not primitive recursive:

1. Unbounded/infinite summantion
2. Unbounded minimum

Theorem 4.3.14. *The Ackerman function is $f : \mathbb{N}^2 \to \mathbb{N}$ where*

$$f(x, y) = \begin{cases} y+1, & x = 0 \\ f(x-1, 1), & y = 0 < x \\ f(x-1, f(x, y-1)), & \textit{otherwise} \end{cases}$$

	x=0	1	2	3	4	5	···
y=0	1	2	3	4	5	6	···
1	2	3	4	5	6	7	···
2	3	5	7	9	11	13	···
3	5	13	29	61	125	253	···
4	13	65533	huge	···	···	···	···

Figure 2: A table of some values of the Ackerman function

Theorem 4.3.15. *The Ackerman function is* not *primitive recursive.*
I tried transcribing the proof but it turned into a bit of a mess, see lecture 2 of week 3 recording for the full thing or the source code where I've left my attempt commented out.

Remark 1. Primitive recusrion is not the "right" definition of "computable".

Theorem 4.3.16. *Predicates are*

Definition 4.6. The characteristic function of a predicate P is

$$\chi_P(x_1, \dots, x_n) = \begin{cases} 1 & P(x_1, \dots, x_n) \\ 0 & \neg P(x_1, \dots, x_n) \end{cases}$$

Theorem 4.3.17. *A predicate is primitive recursive when χ_P is primtive recursive.*

Theorem 4.3.18. *The following functions and predicates are primtive recursive.*

- $\neq$ *($Sg(x \dot{-} y) + Sg(y \dot{-} x)$)*
- $=$ *($1 \dot{-} \chi_{\neq}(x, y)$)*

- $\neg P$
- $p \wedge q$ *($\chi_p \cdot \chi_q$)*
- $p \vee q$ *(use De Morgan's law)*

Theorem 4.3.19. *Bounded exists ($\exists y \leq z$) and bounded forall ($\forall y \leq z$) are primtive recursive.*

Theorem 4.3.20. *$x|y$ is primitive recusrive.*

$$\chi_{x|y} = \exists_{k \leq y}(y = k \cdot x)$$

Theorem 4.3.21. *Prime tests and a function for the nth prime are primtiive recursive.*

Definition 4.7 (Iteration)**.** For a function $f : X \to X$, the ith iterate of f is $F : X \times \mathbb{N} \to X$ where

$$\begin{aligned} F(x, i) &= \overbrace{f(f(\dots(f(x))\dots))}^{i \text{ copies of } f} \\ F(x, 0) &= x \end{aligned}$$

Theorem 4.3.22. *If f is primitive recursive, F is also primitive recursive.*

Theorem 4.3.23. *Iteration is equivalent to primitive recursion, i.e. the class of primitive recursive functions is the smallest class of functions containing the intial functions and is closed under composition and iteration.*

Definition 4.8 (Hyperoperation)**.**

$$\begin{aligned} a[1]b &= a + b \\ a[2]b &= ab \\ a[3]b &= a^b \\ a[k]b &= \underbrace{a[k-1](a[k-1](\dots a[k-1]a\dots))}_{b \text{ times}} \end{aligned}$$

Theorem 4.3.24. *If f is primitive recursive, the running time to compute f (on a TM) is at most $2[k]n$ for some fixed $k \in \mathbb{N}$.*

Definition 4.9 (Minimisation)**.** Minimisation (unbounded) is a partial function μy.

$$\mu y f(\underline{x}, y) = \begin{cases} r & f(\underline{x}, r) = 0 \text{ or } f(\underline{x}, s) \neq 0 \text{ and is defined } \forall s < r \\ \text{undefined} & \end{cases}$$

Definition 4.10 (Regular minimisation)**.** Regular minimisation is μy where $\mu y f(\underline{x}, y) = 0$ is defined for all $\underline{x}$.

Definition 4.11 (Partial recursive functions)**.** The smallest class that contains the initial functions and is closed under composition, primitive recusrion and minimisation.

Definition 4.12 (Recursive functions)**.** The smallest class that contains the initial functions and is closed under composition, primitive recusrion and regular minimisation.

Definition 4.13 (Numerical Turing machines)**.** A numerical Turing machine is a deterministic Turing machine with no final states and the alphabest is $\{0, 1\}$ where 0 is the blank symbol.

Theorem 4.3.25. *To compute $f(x_1, \ldots, x_n)$ on a numerical TM, start with an initial tape with x_1 1s, a 0, x_2 1s, a 0, and so on, with the head at the 0 before x_1. Let it run. If it halts in a configuration like $00\ldots\bar{0}11\ldots100\ldots$ then the output is the number of 1s, else undefined.*

Theorem 4.3.26. *$\forall$ numerical TMs T, $\forall n \geq 1$, T defines a partial function $\varphi_{T,n} : \mathbb{N}^n \to \mathbb{N}$.*

Theorem 4.3.27. *$f : \mathbb{N}^n \to \mathbb{N}$ is TM-computable if there is some TM which computes f.*

Theorem 4.3.28. *f is partial recusrive iff f is TM-computable.*

Definition 4.14 (Decidable)**.** Some predicate P is decidable if χ_P is recursive.

Definition 4.15 (Halting problem)**.**

$$g : \mathbb{N}^2 \to \mathbb{N} = \begin{cases} 1 & \text{if } x \text{ represents a TM and } \varphi_{T_x,1}(i) \text{ is defined} \\ 0 & \end{cases}$$

Theorem 4.3.29. *g is not partial recursive.*

Proof. Let $f : \mathbb{N} \to \mathbb{N}$

$$f(x) = \begin{cases} 1 & g(x,x) = 0 \\ \text{undefined} & g(x,x) = 1 \end{cases}$$

Claim: if g is partial recursive, then so is f. Therefore, f is TM-computable. Therefore, $f = \varphi_{T_n,1}$ for some n.
What is $f(n)$?
$g(n,n) = 0 \implies f(n) = 1 \implies g(n,n) = 1$, contradiction.
$g(n,n) = 1 \implies f(n)$ undefined $\implies g(n,n) = 0$, contradiction.
Therefore, our assumption was wrong and so g is not partial recursive. □

TODO move stuff around.
How to encode TM in $\mathbb{N}$? Gödel numbering:
Alphabet symbols are $s_0, s_1, \ldots$ where s_0 is blank. States are $q_0, q_1, \ldots$ where q_0 is initial.
Transition function $\subseteq Q \times A \times Q \times A \times \{L, R\}$.
Assign numbers to everything: $x : LRs_0q_0s_1q_1 \ldots$ $\lceil x \rceil : 234567\ldots$
Arrows $\tau \in$ transition functions, $t = (a, b, c, d, e)$ $\lceil t \rceil = 2^{\lceil a \rceil}3^{\lceil b \rceil}5^{\lceil c \rceil}\ldots$
Only thing we need is the arrows.
TM by transition functions $\{t_0, t_1, \ldots\}$ is $\lceil t \rceil = 2^{\lceil t_0 \rceil}3^{\lceil t_1 \rceil}5^{\lceil t_2 \rceil}\ldots$
So we can feed a TM as an input where we need an $\mathbb{N}$.
If $x = \lceil TM \rceil$, the TM is T_x.

Theorem 4.3.30. *The number of TMs is countable.*

Theorem 4.3.31. *$\varphi : \mathbb{N}^2 \to \mathbb{N}$*

$$\varphi(x,y) = \begin{cases} \varphi_{T_x,1}(y) & x \text{ represents a TM} \\ \text{undefined} & \end{cases}$$

φ simulates a TM. φ is partial recursive. Therefore, φ is TM-computable. "A universial TM"

5 Formal systems

Definition 5.1 (Formal system)**.** A formal system is a set of grammars with a set of axioms and inferences.

Definition 5.2 (Robinson's theory Q)**.**

Definition 5.3 (Peano arithmetic)**.**

Definition 5.4 (Consistency)**.** T is consistent if $\nexists$ statement Q s.t. $T \vdash Q$ and $T \vdash \neg Q$.

Definition 5.5 (ω-consistency)**.** T is ω-consistent if $\nexists$ formula $\varphi(x)$ s.t.

- $T \vdash \varphi(\bar{n})$ for every numeral $\bar{n}$, and
- $T \vdash \neg\forall x\, \varphi(x)$.

I.e. you cannot prove every individual case of a formula and also prove there exists a counterexample.

Definition 5.6 (Numeral)**.** A numeral is a logical symbol $SS \cdots S0$ for some number of 'S's. Define $\bar{n}$ to be the numeral with n 'S's.

Definition 5.7 (Recusrively enumerable)**.** A set $A \subseteq \mathbb{N}$ is recusively enumerable if A is a range of some recursive function $f : \mathbb{N} \to \mathbb{N}$ or $A = \varnothing$.
I.e. there is a turing machine which computes the members of the set A after a finite number of steps.

Definition 5.8 (Theorem)**.** A theorem Q in T is a statement in T s.t. $T \vdash Q$.

Definition 5.9 (Efficiently axiomatised)**.** A formal system T is efficiently axiomatised if the set of all theorems in T is recusrively enumerable.

Is formal arithmetic eifficently axiomatised?

- Robinson's theory Q $\to$Yes, test $0, 1, \ldots$ as the Gödel number of a proof (easy since finitely many axioms).
- Peano arithmetic $\to$Yes, the set of induction axioms is recusively enumerable so we can still check.

Definition 5.10 (Representability)**.** Let R be a k-ary relation (i.e. $R \subseteq \mathbb{N}^k$). R is represented by the formula $\varphi(v_1, \ldots, v_k)$ if

- $T \vdash \varphi(\bar{n_1}, \ldots, \bar{n_k})$ when $R(n_1, \ldots, n_k)$ is true
- $T \vdash \neg\varphi(\bar{n_1}, \ldots, \bar{n_k})$ when $R(n_1, \ldots, n_k)$ is false

Theorem 5.0.1. *If R is recusrive, R is represented in the FA T.*
Examples: $x|y$, p is prime, m is the GN of a formula with one free variable, m, n where m is the GN of a proof of the sentence n.

Theorem 5.0.2 (Gödel's first incompleteness theorem)**.** *Let T be a formal system that*

- *is efficiently axiomatised,*
- *is ω-consistent, and*
- *contains Robinson's theory Q.*

Then T is incomplete, i.e. there are true statements without a proof (nor proof of the negation).

Proof #1 (via Halting problem). Assume T is complete. Aim to solve the halting problem: define

$$Halt(m,x) = \begin{cases} 1 & \text{if } TM_{m,1} \text{ halts on input } x \\ 0 & \text{otherwise} \end{cases}$$

Simelataneously,

- simulate $TM_{m,1}$ on input x with a universal TM, and
- enumarate all theorems in T until we see "$TM_{M,1}(x)$ does not halt." i.e. (and let φ be) $\forall s$, s is not a sequence of steps for TM_m that starts with input x and ends in a halting state.

If the first halts, output 1. If the latter produces a theorem, output 0. This algorithm must fail for some m, x, else we can solve the halting problem. Therefore, m does not halt on x, and φ is not a theorem of T. This means $T \nvdash \varphi$. If $T \vdash \neg\varphi$, $T \vdash \exists s$ s.t. s is a sequence which does halt. But, give me a sequence s, I can simulate TM_m on x and write out its behaviour, which shows $T \vdash s$ is not a halting sequence. By ω-consistency, $T \nvdash \neg\varphi$.
Therefore, T is incomplete. □

Fact: we can use consistency rather than ω-consistency to prove this.

Proof #2 (via undecidable statement). Using FA as our formal system. Define $Pf(x,y,z) \subseteq \mathbb{N}^3$ to be y is the GN of a formula φ with one free variable and x as a proof of $\varphi(\bar{z})$. Recusive $\therefore$ is represented by formula $\underline{Pf}(x,y,z)$. Build new formula $\psi(y) : \nexists x Pf(x,y,y)$, (i.e. FA cannot prove $\psi_y(\bar{y})$). Let g be the GN of ψ. $\psi(g)$ enocdes "FA cannot prove $\psi(g)$".

- If FA $\vdash \psi(\bar{g})$, Let m be the GN of the proof. Therefore, $Pf(m,g,g)$ is true. Therefore, FA $\vdash \underline{Pf}(\bar{m},\bar{g},\bar{g})$. Therefore, FA $\vdash \exists x \underline{Pf}(x,\bar{g},\bar{g})$. Therefore, FA $\vdash \neg\psi(\bar{g})$. Therefore, FA is inconsistent. Contradiction (elided: show ω-consistency implies consistency).
- If FA $\vdash \neg\psi(\bar{g})$, Then FA $\nvdash \psi(\bar{(g)})$. Therefore, $\forall m, Pf(m,g,g)$ is false. Therefore, $\forall m, FA \vdash \neg\underline{Pf}(\bar{m},\bar{g},\bar{g})$. Therefore, FA $\nvdash \exists m \underline{Pf}(m,\bar{g},\bar{g})$. Therefore, FA $\nvdash \neg\psi(\bar{g})$. Contradiction.

Therefore, $\psi(\bar{g})$ is undecidable. □

This proof can be extended to any extension Σ of FA where "n is the GN of an axiom in Σ" is recursive.

Theorem 5.0.3 (Gödel's second incompleteness theorem)**.** *If FA is consistent, then $FA \nvdash con$ where con encodes "$FA \nvdash \bar{0} = \bar{1}$".*

Proof. Define $Prf(x,y) \subseteq \mathbb{N}^2$ as a relation that "x is a proof of y" where x is a GN of a proof and y is a GN of a sentence. $Prf(x,y)$ is represented by the formula $\underline{Prf}(x,y)$. Let n be the GN of the sentence "$\bar{0} = \bar{1}$". Let con (consistency) be $\forall x, \neg\underline{Prf}(x,\bar{n})$. Then (Gödel): FA is consistent $\implies$ FA $\forall con.\ con \implies \psi(\bar{g})$ (from first theorem proof). Therefore, FA $\vdash con \implies \psi(\bar{g})$. By the first incompleteness theorem, FA $\nvdash \psi(\bar{g})$. If FA $\vdash con$, FA $\vdash \psi(\bar{g})$. Contradiction. Therefore, FA $\nvdash con$. □