

Logic, Computability and Incompleteness

Cardinality, Enumerability,
Diagonalization

Cardinality

- The **cardinality** of a set is a measure of its ‘size’ in terms of numbers of elements. For example, the set $\Gamma = \{0,1,2\}$ contains 3 elements, and therefore Γ has a cardinality of 3.
- Two sets have the same cardinality iff they are ‘**equinumerous**’.
- Equinumerosity is defined in terms of a 1-1 correspondence between two sets, where a 1-1 correspondence is in turn defined in terms of certain types of ‘mappings’ or **functions**.
- In brief, a function is an assignment of *values to arguments*, where the domain of the function is the set to which its arguments or ‘inputs’ belong, and the range of the function is the set to which its values or ‘outputs’ belong.

Cardinality

- A function f with the set Γ as domain and the set Δ as range, written $f: \Gamma \xrightarrow{\bullet} \Delta$, is a **bijection** (also called a 1-1 correspondence between Γ and Δ) iff
 - (1) $\forall x, z \in \Gamma$, if $x \neq z$, then $f(x) \neq f(z)$ (i.e. f is 1-1, or *injective*), and
 - (2) $\forall y \in \Delta$, $\exists x \in \Gamma$: $y = f(x)$ (i.e. f is *onto* or *surjective*).
- Two sets Δ and Γ have the **same cardinality** iff there exists a bijection $f: \Gamma \xrightarrow{\bullet} \Delta$.
- The **cardinal number** of Γ will be written as $|\Gamma|$.
- If Γ and Δ have the same cardinality, then $|\Gamma| = |\Delta|$.

«

Cardinality

- Π is a **subset** of Γ , written $\Pi \subseteq \Gamma$, iff $\forall x(x \in \Pi \rightarrow x \in \Gamma)$.
- By failure of the antecedent condition in this definition, the **empty set** \emptyset is a subset of every set,
and obviously every set Γ is a subset of itself.
- Π is a **proper subset** of Γ , written $\Pi \subset \Gamma$, iff
$$\Pi \subseteq \Gamma \text{ and } \Pi \neq \Gamma$$
- Γ has **greater cardinality** than Δ , written $|\Gamma| > |\Delta|$, iff
 - (i) there is no bijection $g : \Delta \xrightarrow{\bullet} \Gamma$ and
 - (ii) for some proper subset $\Sigma \subset \Gamma$, there is a bijection $f : \Sigma \xrightarrow{\bullet} \Delta$

Cardinality

- All sets have either finite or infinite cardinality.
A simple characteristic which distinguishes the two is that a set Γ is **infinite** iff there exists a bijection $f : \Gamma \xrightarrow{\sim} \Sigma$ *for some proper subset $\Sigma \subset \Gamma$.*
- For example, the set **N** of natural numbers ($= \{0,1,2,3,\dots\}$) is **infinite**, since the set of squares of natural numbers is a proper subset of **N**, and the function $f(x) = x^2$ is a bijection between **N** and the set of squares of natural numbers.
- The **smallest infinite** cardinality is that of the natural numbers. Any set with this cardinality is called **denumerable**, and has cardinal number \aleph_0 .

Cardinality

- A **countable** set is defined as either finite or denumerable and an **uncountable** set is **neither**.
- The **power set** of Γ , written $\mathcal{P}(\Gamma)$, is defined as the **set of all subsets** of Γ .
- Thus \emptyset and Γ form the two endpoints of the spectrum for membership in $\mathcal{P}(\Gamma)$, and all other members of the power set fall between these two extremes.
- As a simple example, let $\Gamma = \{0,1,2\}$.
- Then $\mathcal{P}(\Gamma) = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0,2\}, \{1,2\}, \{0,1,2\}\}$

Cardinality

- **Cantor's Theorem** (1891): The power set of any set always has greater cardinality than the set itself.
A brief proof of this theorem will be given in the section on **diagonalization**.
- If it is assumed that there is an infinite set, and that the power set of a set always exists,
then an **infinite hierarchy** of ever increasing cardinality is induced.
- Power set cardinality is governed by the general equation that for any set Γ :
$$|\mathcal{P}(\Gamma)| = 2^{|\Gamma|}$$
 This relation is easy to verify for finite sets.

Cardinality

- Cantor's Theorem, in combination with the fact that \mathbf{N} is denumerable, immediately yields the result that

$\mathcal{P}(\mathbf{N})$ is uncountable

this will also be proved (independently) in the section on diagonalization.

- The present section will end by mentioning a few related mathematical facts and conjectures.
- Even though the natural numbers \mathbf{N} are not **dense**, while the rational numbers \mathbf{Q} are, the set of rational numbers \mathbf{Q} is still **denumerable**.

Thus $|\mathbf{Q}| = |\mathbf{N}| = \aleph_0$

Cardinality

- However, the set \mathcal{R} of real numbers is uncountable, and furthermore it is provable that $|\mathcal{R}| = |\mathcal{P}(\mathbf{N})|$.
- The above identity relation, in combination with the exponential equation governing power set cardinality, yields the result that $|\mathcal{R}| = 2^{\aleph_0}$
- **Cantor's Continuum Hypothesis** (1878): there is no cardinal number greater than \aleph_0 and less than 2^{\aleph_0} .
- If this conjecture is true, then there is no ‘missing’ level of infinity between $|\mathbf{N}|$ and $|\mathcal{P}(\mathbf{N})|$.

Cardinality

- Establishing the truth or falsity of the **Continuum Hypothesis** (CH) is the first of Hilbert's 23 outstanding problems for 20th century mathematics.
- In 1938 Kurt Gödel proved that CH is **consistent** with the axioms of standard (ZFC) set theory
and in 1963 Paul Cohen proved that the *negation* of CH is also **consistent** with ZFC.
- Hence CH is **logically independent** of standard set theory:
it cannot be proved or disproved on the basis of these axioms.

Enumerability

- Intuitively, an enumerable set is one whose members can all be ‘arranged’ in a single list.
- Clearly every finite set is enumerable.
- For infinite sets, an acceptable list must be such that each item eventually appears as the n th entry, for some finite n .
- Thus for the set \mathbf{P} of positive integers ($= \mathbf{N} - \{0\}$),
1,2,3,4,5, ... is an acceptable list, while
1,3,5, ..., 2,4,6, ... is *not*
because in the latter case, it takes infinitely many entries to get to the first even number.

Enumerability

- In more precise terms, an enumeration of a set Γ is equivalent to an *onto* function $f: \mathbf{P} \xrightarrow{\bullet} \Gamma$. The function f must be *onto* so that every member of Γ appears at least once in the list.
- It is not required that f be 1-1, and hence an enumeration can be **redundant** (since if f is not 1-1, then at least one item $b \in \Gamma$ will occur at least twice on the list).
- In principle this is not a problem, because redundancies can be systematically eliminated by reviewing the (finitely many) entries preceding any given item on the list and **deleting it** if it has already appeared.

Enumerability

- It is also permissible to have *gaps* in the list, since in principle it is always possible to close these gaps.
- A gap in the list means that f is undefined on the respective argument $n \in P$, in which case f is a *partial function* on P .
- For example, the set E of even positive integers is very naturally enumerated by the function
- $h: P \xrightarrow{\bullet} E$ such that $h(n) = 2n$,
which defines the non-gappy list 2,4,6,8, ...
- However, E is also enumerated by the partial function j such that $j(n) = n$, if n is even, and **undefined** otherwise.
- The function j defines the *gappy* list -,2,-,4,-,6,-,8, ...

Enumerability

- The positive rational numbers can be enumerated through use of a 2-dimensional array, with (+)integer numerators comprising one axis and (+)integer denominators the other.
- A path through this array can be defined by taking all fractions whose numerators and denominators sum to 2, then 3, then 4, ..., listing the fractions in order of lowest numerator.

There will be $k - 1$ entries for each sum k . This gives the (redundant) list $1/1, 1/2, 2/1, 1/3, 2/2, 3/1, 1/4, 2/3, \dots$
- Since every positive rational will appear at least once on this list, it follows that the set of positive rationals is denumerable.

Enumerability

- The enumerability of a set is simply a result of its cardinality. Any countable set Γ is necessarily enumerable (and vice versa), because to be enumerable is just to be the range of an onto function of positive integers.
- So *if* Γ is countable, then it follows that there must be a bijection between Γ and some subset of \mathbf{P} , and this bijection is sufficient to serve as an enumeration.
- Similarly, if Γ is uncountable, then it *cannot* be enumerable, because any attempted list would have to omit (an uncountable infinity of) elements of Γ .

Enumerability

- An enumeration is effective iff the enumerated set is finite, or else there is an explicit, ‘mechanical’ procedure for determining the value $f(n) \in \Gamma$ in a finite number of steps,
for every $n \in \mathbf{P}$.
- It is important to make two immediate points about effective enumerability:
- (i) it is a claim about the *abstract existence* of a mechanical procedure, and as such carries no epistemological baggage;
a set may be effectively enumerable, even though no human being ever knows of an effective procedure for enumerating it

Enumerability

- (ii) it makes no claim about the practicability of the procedure, which means that it may not be humanly possible, due to various resource limitations, to actually compute the value $f(n)$ for even a single n .
- The only requirement for the enumeration to be effective is that it will yield the correct output value after a **finite** number of steps.
- Thus a procedure could be effective even though no computation took less than, say, 10^{50} steps.

Enumerability

- This definition of *effective* may seem overly idealized, but it is the natural limiting case mathematically, and the fact that it is in principle so liberal lends significant conceptual bite to the following negative result (to be demonstrated in due course):
- Not all enumerable sets are effectively enumerable, even under this very idealized notion of what it is to be effective.
- Thus **effective enumerability** is *not* just the result of brute cardinality.

Diagonalization

- In this section, Cantor's elegant and versatile **diagonal method** will be employed, first in a *specific* instance to show that $\mathcal{P}(\mathbf{P})$ is uncountable, and then in the *general* case to prove that the cardinality of the power set is always greater than that of the original set.
- **Proof** that $|\mathcal{P}(\mathbf{P})| > |\mathbf{P}|$:
 $\mathcal{P}(\mathbf{P})$ is by definition the set of all subsets of \mathbf{P} .
If $\mathcal{P}(\mathbf{P})$ were enumerable, then there would exist some function $f: \mathbf{P} \xrightarrow{\bullet\rightarrow} \mathcal{P}(\mathbf{P})$ which would define a list of all subsets of \mathbf{P} .
Suppose there were some such list L , and suppose that the sequence S_1, S_2, S_3, \dots is the resulting enumeration of the sets S_i of positive integers.

Diagonalization

- Let the **antidiagonal set**, with respect to the list L , written \underline{D}_L , be specified as follows.
 - (i) $\forall n \in P [n \in \underline{D}_L \text{ iff } \neg(n \in S_n)]$.
- The set \underline{D}_L is perfectly well defined given a well defined list L , and clearly $\underline{D}_L \subseteq P$ and hence $\underline{D}_L \in \mathcal{P}(P)$.
- But \underline{D}_L has been constructed in such away that it cannot appear anywhere in the given list L of subsets of P .
- For suppose that \underline{D}_L did appear somewhere in L .
Then it must be the case that $\underline{D}_L = S_k$ for some $k \in P$.

Diagonalization

But if \underline{D}_L and S_k were indeed the same set, then the extensional identity condition on sets requires that

$$\text{(ii)} \forall n \in P [n \in \underline{D}_L \text{ iff } n \in S_k].$$

- Now take the particular positive integer k which specifies the place of S_k in the list L . Formula (ii) above requires that
 $k \in \underline{D}_L$ iff $k \in S_k$, while formula (i) above requires that
 $k \in \underline{D}_L$ iff $\neg(k \in S_k)$.
- Since by hypothesis $\underline{D}_L = S_k$, this leads to the contradiction
 $\text{(iii)} \quad k \in \underline{D}_L \text{ iff } \neg(k \in \underline{D}_L).$
- And since the choice of k was arbitrary, formula (iii) establishes by *reductio ad absurdum* that the set of positive integers \underline{D}_L cannot occur anywhere on the list L .

Diagonalization

- And since an antidiagonal set can be defined for *any* purported list L , it follows that there can be no enumeration of $\mathcal{P}(\mathbf{P})$.
- Accordingly there is no bijection $f: \mathbf{P} \xrightarrow{\text{•}} \mathcal{P}(\mathbf{P})$.
- But for each $n \in \mathbf{P}$, $\{n\} \in \mathcal{P}(\mathbf{P})$. Let \mathbf{S} be the set of all such singletons $\{n\}$ for $n \in \mathbf{P}$.
- Then clearly $\mathbf{S} \subset \mathcal{P}(\mathbf{P})$ and the function $g: \mathbf{S} \xrightarrow{\text{•}} \mathbf{P}$, such that $g(\{n\}) = n$ is a bijection.
- Therefore the cardinality of $\mathcal{P}(\mathbf{P})$ is strictly greater than that of \mathbf{P} , which means that $\mathcal{P}(\mathbf{P})$ is *uncountable*. ■

Diagonalization

- **Proof of Cantor's Theorem** that the power set of *any* set always has greater cardinality than the set itself.
- Let Γ be any set (countable or otherwise), and consider any 1-1 function $f: \Gamma \rightarrow \mathcal{P}(\Gamma)$.
- Since f is 1-1, it follows that for each distinct $x \in \Gamma$, $f(x)$ is a distinct set $\Sigma \subseteq \Gamma$.
- Let the antidiagonal set Δ be defined as the set of all $x \in \Gamma$ such that $x \in \Delta \leftrightarrow \neg(x \in f(x))$.
Then $\Delta \subseteq \Gamma$, and so $\Delta \in \mathcal{P}(\Gamma)$. But $\neg\exists x \in \Gamma$ such that $f(x) = \Delta$.
For suppose there were such an x . Then, by the definition of Δ , it must be the case that $x \in \Delta \leftrightarrow \neg(x \in \Delta)$.

Diagonalization

- Hence for any set Γ and any 1-1 function $f: \Gamma \xrightarrow{\text{onto}} \mathcal{P}(\Gamma)$ it is impossible for f to be *onto*, from which it follows that there can be ***no*** bijection between Γ and $\mathcal{P}(\Gamma)$.
- And by taking the set \mathbf{S} of singletons of elements of Γ , as in the proof above, it can be established that there is a bijection $g: \mathbf{S} \xrightarrow{\text{onto}} \Gamma$, where $\mathbf{S} \subset \mathcal{P}(\Gamma)$ and $g(\{\textcolor{violet}{n}\}) = \textcolor{violet}{n}$
- Thus for *any* set Γ , the cardinality of the power set of Γ is strictly **greater than** the cardinality of Γ . ■

Logic, Computability and Incompleteness

Turing Machines and Computability

What is Computation?

- Two basic ways to look at **computation**:
 - 1) In practice, as the activity of *physical machines*, as what computers **do**, where a *computer* is an actual device that exists in space and time
 - 2) In theory, as a certain type of *mathematical ‘process’* – as something that belongs to an abstract, idealized domain, like Euclidean geometry, set theory or algebra.
- Our amazing present day *physical computing machinery* was only made possible by prior conceptual advances in the abstract mathematical theory of computation (MTC)
- But before an advanced MTC was developed (19th-20th century) first there were relatively simple **calculational devices**

Early Practice

- ‘Calculate’ comes from the Latin word for ‘stone’, and in ancient times, simple calculations were performed using piles of pebbles or *calculi* to **represent numbers**
- **Abacus**: ingenious frameset with movable beads on rods to make rapid calculations
- From the 17th century onwards there were assorted geared mechanisms for making calculations:
systems of cogs and levers that could add, multiply, etc.
- As well as wheel and disk ‘differential analysers’
- This style of approach culminated in the **Analytical Engine** of Charles Babbage in the mid 1800’s – the most powerful and innovative computing machine envisioned up to that time.

Early Theory

- The foregoing examples are purely applications-driven, and there was no overarching **theory** of computation
- The German philosopher and mathematician Gottfried Leibniz (1646-1716) entertained the idea of a **Universal Logical Language** based on an '*alphabet of human thought*', which could be used to derive all possible truths through basic combinatorial operations, a '*calculus ratiocinator*'.
- Leibniz's idea, though visionary (and highly optimistic!), remained more or less a futuristic dream.
- Then in the late 19th century, another German mathematician and philosopher, Gottlob Frege (1848-1925) developed a system of **formal logic** in an attempt to provided a rigorous **foundation for mathematics**

Frege Invents Formal Syntax

- Frege's *Begriffsschrift* (1879) introduced Universal and Existential quantification, along with the rest of the technical machinery now known as First-Order Logic.
- The logical system of the *Begriffsschrift* is the first instance of an **artificial language** constructed according to exact rules of syntax.
- This logical calculus allowed *formal proofs* to proceed as '*computations*' in accordance with a fixed set of inference rules,
as a series of **purely syntactic manipulations**
which in principle encompassed all the reasoning ordinarily used in mathematics.

Effective Procedures

- And because the system is purely **formal**, the rules can be applied without any specification of their intended **meaning**.
- This idea is **fundamental**, because central to the **mathematical theory of computation** is the intuitive notion of an effective (or ‘mechanical’) procedure, or algorithm.
which is simply a **finite set of instructions** for syntactic manipulations,
that can be followed by a human being who is capable of carrying out only very elementary operations on symbols or by potentially by a **machine** ...

Effective Procedures

- A key constraint is that the machine or the human can follow the rules without knowing what the symbols *mean*.
- The notion of an effective procedure is very general – it doesn't specify what form the instructions should take, what the manipulated symbols should look like, nor precisely what manipulations are involved.
- The underlying restriction is simply that the set of rules is finitary and can proceed **formally**,
i.e. without any additional interpretation or understanding.

Turing Machines

- So there are any number of different possible frameworks for filling in the details and making the broad intuitive idea precise.
- The British mathematician Alan Turing (1912-1954) introduced one such framework in his groundbreaking 1936 paper “**On Computable Numbers**”
- His “**automatic computing machines**”, now generally referred to as “**Turing Machines**”, supply a very intuitive and elegant rendition of the notion of an effective procedure
- But there is a variety of well known alternative frameworks, including Church’s Lambda Calculus, Gödel’s Recursive Function Theory, Lambek’s Infinite Abacus Machines, etc.

Turing Machine Basics

- In turn there are different ways of specifying TMs – we'll use the conventions adopted in B&J. The first thing we need is
- (1) a **tape**: divided into squares or cells, unbounded in both directions.

All but a finite number of cells of the tape are blank.
- (2) a **finite list of symbols** S_1, S_2, \dots, S_n . Non-blank cells are filled with exactly one symbol from the list.
- We treat ‘blank’ as the symbol S_0 .
- (3) a **read/write head**: which can scan exactly one cell of the tape at a time.

Turing Machine Basics

- (4) a **finite set of states**: which are conditional instructions saying *what to do* for a given scanned input symbol.
- What can a TM do? There are $n+4$ possible **overt** actions:
 - (i) **Nothing** (= halt)
 - (ii) **Move L** one square
 - (iii) **Move R** one square
 - (iv) **Write** S_0 in place of currently scanned symbol
 - (v) **Write** S_1 in place of currently scanned symbol
 - ⋮
 - ($n+4$) **Write** S_n in place of currently scanned symbol

Turing Machine Basics

- Plus one *covert* action: **enter next state**.
- The **present state** plus **scanned symbol** functionally determine the next *overt* and *covert* action.
- There are various ways of representing the **finite set of states** which constitute the program of instructions, including:
- Machine Table, Flow Diagram, (finite) Set of Quadruples
- We'll focus on Set of Quadruples scheme of representation, because it allows TM's to be easily enumerated.
- E.g. $q_1 S_0 R q_2$ where q_1 is the current state, S_0 the scanned symbol, R is the overt act of moving right one square and q_2 the covert 'act' of entering state q_2 .

TM Examples

- So the quadruple $q_1 S_0 R q_2$ is a **conditional instruction**:
if in state q_1 reading the symbol S_0 , **then** move R one square and enter state q_2
- If a TM is in current state q_i scanning a symbol S_k and there is no quadruple in its program of instructions beginning with the pair $q_i S_k$ then the machine can do nothing and must **halt**.
- Another example: $q_2 S_1 S_0 q_1$ where q_2 is the current state, S_1 the scanned symbol, S_0 is the overt act of printing a blank (i.e. erasing S_1) and q_1 the covert act of entering state q_1 .
- Note that the next state entered doesn't have to be new, so that $q_2 S_1 L q_2$ is a perfectly good quadruple.

TM Examples

- Note also that any (non-empty) finite set of perfectly good quadruples constitutes a perfectly good Turing Machine.
- So each of the single quadruples above *is* a particular TM.
- Here's another example of a single instruction TM:

$q_1 \ S_1 \ S_1 \ q_1$

- Any particular **computation** on a given input string can be described as a sequence of configurations, where a configuration shows the **content of the tape** at that stage in the sequence,
and it shows what **state the machine is in**
and which **square is being scanned**.

TM Examples

- A handy convention: write ‘B’ for the symbol S_0 , and write ‘1’ for the symbol S_1
- Here’s an example of a TM that, if started in state q_1 scanning the leftmost of an unbroken string of 1’s on an otherwise blank tape, will **halt** scanning a 0 if the number of 1’s in the string is even, and will **halt** scanning a 1 if the number of 1’s is odd:

$q_1 \ 1 \ B \ q_2$

$q_2 \ B \ R \ q_3$

$q_3 \ B \ 1 \ q_5$

$q_3 \ 1 \ B \ q_4$

$q_4 \ B \ R \ q_1$

TM Examples

So this machine gives a **yes/no** answer to the question
‘Is the number of 1’s in the input string odd?’

- Following are 2 sample **computations**
(= **sequence of configurations**) in parallel:

$q_1 1 B q_2; q_2 B R q_3; q_3 B 1 q_5; q_3 1 B q_4; q_4 B R q_1$

Start ...001100... ...01110...

 1 1

 00010 00110

 2 2

 00010 00110

 3 3

 00000 00010

 4 4

 00000 00010

 1 1

Halt 00000

 2

 00000

 3

 00001

 5 **Halt**

Abstract versus Physical

- TM's are mathematical abstractions, not actual machines. They don't exist in physical space-time, and they have no causal powers.
- Hence TM computations have no chronological duration or physical extension.
- Indeed, our picturesque images of tape, moving read/write head, etc. are just conceptual aids and not essential to a TM's mathematical structure.
- In order to perform *actual* computations, an abstract Turing machine must be realized or implemented by a suitable arrangement of mass/energy.

Multiple Realization

- And as Turing observed long ago, there is no privileged or unique way to do this.
- Like other abstract structures, such as chess games and isosceles triangles, Turing machines are *multiply realizable*
- What unites different types of physical implementation of the **same** abstract TM is nothing that they have in common as physical systems, but rather a structural isomorphism characterized at a higher level of description.
- Hence it's possible to implement the very same computational formalism using **modern electronic circuitry**,
a human being executing the instructions by hand with paper and pencil,

Multiple Realization

a Victorian system of **gears and levers**,

as well as more atypical arrangements of matter and energy including rolls of **toilet paper serving as tape** and **beer cans** for the symbol ‘1’.

- In any case, only particular physical realizations can perform actual computations, and any such realization must perforce be just a limited **approximation** of the idealized abstract TM.
- This is because physical devices will always have finite bounds, while abstract TM’s **do not**.
- There is no finite upper bound on input size, tape length, number of instructions defining the program, nor steps in the computation.

Syntax versus Semantics

- As in our original characterization of an **effective procedure**, the program of instructions specifying a TM is merely a recipe for syntax manipulation, since the essence of classical computation **is rule governed symbol manipulation**.
- Additionally, TM's can be interpreted as computing various **numerical functions** by (i) interpreting the input/output strings as *notation for numbers*, and (ii) supplying conventions governing *permissible configurations*.
- We will be especially interested in interpreting TM's as computing functions from positive integers to positive integers.

Some relevant conventions:

- 1) positive integers are represented in **monadic notation**.
- 2) TM's read and write only the symbols B and 1
- 3) **Permissible initial configuration**: starts in lowest numbered state reading leftmost 1 on the tape.

Input numbers are separated by single blanks.

- 4) **Permissible halting configuration**: halts reading the leftmost of an unbroken string of 1's. This gives the output value.

If it doesn't halt, or halts in a non-permissible configuration, then the computed function is undefined for that input.

- In this manner we can specify the class of **Turing Computable Functions** on the positive integers.

- For example, here's a TM that computes **addition** under the foregoing conventions:

$q_1 1 B q_1$

$q_1 B R q_2$

$q_2 1 R q_2$

$q_2 B 1 q_3$

$q_3 1 L q_3$

$q_3 B R q_4$

Machine $q_1 1 B q_1; q_1 B R q_2; q_2 1 R q_2; q_2 B 1 q_3; q_3 1 L q_3; q_3 B R q_4$
computes $2+3 = 5$

Start ...0101110...
 1
...00101110...
 1
...00101110...
 2
...00101110...
 2
...00111110...
 3
...00111110...
 3
...00111110...
 3
...00111110...
 4 **Halt**

Variations on a Theme

- We have looked at **one** common way of formulating TM's, but there are a number of possible variations, all of which are provably equivalent:
 - 1) Allow tape to be ‘infinite’ in **one** direction only. This might appear to restrict computational capacity, but it doesn’t.
 - 2) Instead of a single tape, have **multiple tapes and heads**.
 - 3) Use **two-dimensional tape**, ‘infinite’ along both dimensions.
 - 4) Allow read/write head to **move some** (fixed) **arbitrary number of squares** at each transition, not just one
still no increase in power.

Variations on a Theme

- 5) Instead of arbitrarily large finite alphabet S_1, S_2, \dots, S_n , **restrict to only two symbols**, ‘0’ and ‘1’ (as in conventions above).
- 6) Turing originally used a **quintuple formulation** rather than a set of quadruples, wherein the machine can both *write and move its head* in the same transition. This doesn’t increase power.
- 7) **Non-deterministic TM’s**: Allow distinct quadruple instructions for the same *current state/scanned symbol* pairs. This leads to multiple possible transitions from the same configuration.

Universal Turing Machines

- In some sense, a standard TM can be thought of as *dedicated* to computing a particular function. This is a limitation
- One of Turing's great insights concerns the existence of *Universal Turing Machines* (**UTM**).
- When started on a tape containing the encoding of another Turing machine, call it **T**, followed by the input to **T**, a **UTM** produces the same result as **T** would when started on that input.
- Essentially a **UTM** can simulate the behavior of **any** Turing machine (including itself).
- So **UTM** is the conceptual prototype for a **programmable computer**, where the inputted encoding of the machine to be imitated serves as the **program**.

Church's Thesis and Computability

- The Church-Turing Thesis (or Church's Thesis):
Any function which is in principle computable can be computed by some Turing Machine.
- So if Church's Thesis is **true**, then the Turing computable functions are co-extensive with the class of functions which can be computed by **any** effective procedure whatever.
- Hence according to the thesis, TM's capture a maximal, stable class of phenomena, a basic '**mathematical kind**'.
- Thus, our notion of 'in principle **computability**' can be nicely expressed as '**computable by a UTM with no finite upper bound on length of program, tape or running time.**'

Church's Thesis

- Church's Thesis cannot be proved, because there is no limit on the number of different frameworks in which the intuitive notion of an effective procedure might be formally expressed.
- Future renditions of the notion cannot currently be foreseen...
- However, Church's Thesis is **refutable if false**.
- So we can gather ‘inductive evidence’ for the truth of the thesis by comparing different frameworks.
- Thus far, **every** given framework for characterizing the notion of an effective procedure has been shown to be **equivalent** to Turing computability.

Alternative Frameworks

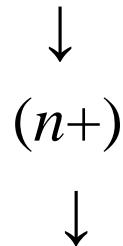
- So in support of Church's Thesis, and to explore some (seemingly) very different methods, we'll investigate two other computational frameworks.
- First we'll look (briefly) at **Abacus computable functions**, which can be shown to be **Turing computable**, i.e. $\mathbf{A} \subseteq \mathbf{T}$.
- Then we'll look at **Recursive functions**, which can be shown to be **Abacus computable**, i.e. $\mathbf{R} \subseteq \mathbf{A}$.
- Finally, it can be shown that $\mathbf{T} \subseteq \mathbf{R}$, thereby closing the chain of inclusions $\mathbf{R} \subseteq \mathbf{A} \subseteq \mathbf{T} \subseteq \mathbf{R}$, to establish that $\mathbf{R} = \mathbf{A} = \mathbf{T}$.

Abacus Machines

- In contrast to **TM**'s, Abacus Machines bear a much closer resemblance to (seemingly more powerful) high speed digital computers, because **AM**'s have an unlimited amount of **random access storage**.
- **AM**'s have an **unbounded number** of registers R_0, R_1, R_2, \dots , in each of which can be written numbers of **arbitrary size**.
- An **AM** has ‘random’ access to these registers, in the sense that it can go directly to register R_n , without inching (‘blindly’) along a tape square by square.
- Each register R_n has its own address n , which allows the machine to carry out operations such as ‘**compute the sum of the numbers in registers R_n and R_m and store it in register R_k** ’

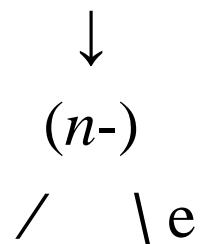
Abacus Machines

- Our pictorial representation of such a machine will depict the registers as **labelled boxes** which can contain arbitrarily large quantities of **stones**.
- The **boxes** are located in a large cave, and on the floor of the cave is an arbitrarily large supply of **stones**.
- There is a caveman executing the program, and the possible actions the caveman can perform are:
 - i) pick up *one stone* and **add** to box n



Abacus Machines

- ii) **remove** *one stone* from box n , **if** it is not empty.



- So program the caveman follows is given by a flow diagram with arrows connecting these actions on registers, where the minus nodes are branch points and can have two exit arrows, and so the next action in the sequence will depend on whether or not the box is already empty.

Abacus Machines

- In this manner, can perform basic operations such as the above ‘compute the sum of the numbers in registers R_n and R_m and store it in register R_k ’

$$[m] + [n] \rightarrow k$$

where ‘ $[m]$ ’ denotes the number of stones in box m .

- With these resources can construct **AM**’s to compute multiplication on the natural numbers, exponentiation, etc.
- See B&J chapter 6 for relevant technical details.

Abacus Computable Functions are Turing Computable

- In order to show that $\text{A} \subseteq \text{T}$, need to give a method which, applied to any **AM** flow graph yields the flow graph of a **TM** that computes the same function.
- **Basic plot:**
 - 1) give method for correlating **AM** registers with contents of **TM** tape.
 - 2) replace each $(s+)$ node with a **TM** flow graph which finds the appropriate block of 1's on the tape, adds a 1 to the R, shifts all remaining blocks to R, then returns to L-most 1 on tape.

Abacus Computable Functions are Turing Computable

- 3) replace (s -) nodes with corresponding flow graph:
finds R-most 1 in s^{th} block, erases, shifts remaining blocks,
returns to L-most 1 on tape.
- 4) connect each remaining loose arrow to a ‘mop-up’ graph
which keeps L-most 1 on tape, erases everything else on tape
except the n^{th} block (which contains output value), shifts this
block to L-most position and halts in standard configuration
(again, see B&J chapter 6 for relevant technical details).

This will then suffice to show that $\mathbf{A} \subseteq \mathbf{T}$.

- **Moral of the story:** random access storage does **not** increase in principle computing power, but only speed.

Logic, Computability and Incompleteness

Recursive Functions

Introduction

- Recursive Functions constitute a very broad class, expressed explicitly in terms mathematical equations.
- Functions in this class include members of the familiar series **addition**, **multiplication**, **exponentiation**...
- Indeed, the class is so broad it seems intuitively plausible that **all effectively computable functions are recursive**.
- We will return to recursive functions again when we look at basic number theory formalized in first order logic.
- We'll first define this class of functions, and then provide further evidence in support of the **Church-Turing Thesis** by showing that all recursive functions are Abacus computable, i.e. $\text{R} \subseteq \text{A}$

Primitive Recursive Functions

- We begin by defining the proper subclass of Primitive Recursive Functions.
- First we specify an initial stock of basic functions belonging to the class of primitive recursive functions,
and then define 2 types of operation which yield members of that class when applied to members of that class.
- There are 3 distinct categories of basic functions:
 - 1) **zero function**
 - 2) **successor function**
 - 3) **projection functions**

Basic Functions

1) zero function, for all natural numbers x ,

$$\mathbf{z}(x) = 0.$$

2) successor function, for all natural numbers x ,

$\mathbf{s}(x)$ = the natural number which is the successor of x

3) projection (or identity) functions, come in assorted arities:

$$\mathbf{id^1}_1(x) = x, \mathbf{id^2}_1(x, y) = x, \mathbf{id^2}_2(x, y) = y$$

In general $\mathbf{id}^n_i(x_1, \dots, x_i, \dots, x_n) = x_i$

- All such basic functions are **primitive recursive**.

Operations

- From the basic functions we can form new primitive recursive functions through the operations of **composition** and **primitive recursion**.
- Composition: if f is a function of m arguments and g_1, \dots, g_m are functions of n arguments, then the composition h is the function of n arguments such that

$$h^n(x_1, \dots, x_n) = f^m(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n))$$

- So if f and the g 's are primitive recursive, then so is the composition h , written $h = \text{Cn}[f, g_1, \dots, g_m]$

Examples

- Want to define 1-place p.r. function h^1 such that $h^1(x) = x + 3$.
 $h^1 = \mathbf{Cn}[\mathbf{s}, \mathbf{Cn}[\mathbf{s}, \mathbf{s}]]$ where $\mathbf{Cn}[\mathbf{s}, \mathbf{s}] = \mathbf{s}(\mathbf{s}(x))$ and so

$$\mathbf{Cn}[\mathbf{s}, \mathbf{Cn}[\mathbf{s}, \mathbf{s}]] = \mathbf{s}(\mathbf{s}(\mathbf{s}(x)))$$

- Want to define 3-place p.r. function h^3 such that

$$h^3(x_1, x_2, x_3) = x_2 + 3$$

$$h^3 = \mathbf{Cn}[h^1, \mathbf{id}^3_2] = \mathbf{Cn}[\mathbf{Cn}[\mathbf{s}, \mathbf{Cn}[\mathbf{s}, \mathbf{s}]], \mathbf{id}^3_2]$$

$$\mathbf{Cn}[h^1, \mathbf{id}^3_2](x_1, x_2, x_3) = h^1(\mathbf{id}^3_2(x_1, x_2, x_3))$$

$$= h^1(x_2) = \mathbf{s}(\mathbf{s}(\mathbf{s}(x_2))) = x_2 + 3$$

Operations

- Primitive recursion: will first specify in terms of a **schema** for defining a 2-place function $h(x,y)$ in terms of a 1-place function f and a 3-place function g .

$$h(x, 0) = f(x)$$

$$h(x, s(y)) = g(x, y, h(x, y))$$

- So, given **p.r.** functions f and g , this definition will recursively generate all values of h for a given argument x , starting with $y = 0$ and then using the previous value to define the next one.
- First yields $h(x, 0)$ then $h(x, 1)$, $h(x, 2)$, ...
- So given any pair of numbers x, y this procedure will compute the value $h(x, y)$ in $y + 1$ iterations.

Primitive Recursion

- Notation: $h = \text{Pr}[f, g]$
- Example: *informal* recursive definition of ‘+’ in terms of s

$$x + 0 = x$$

$$x + s(y) = s(x + y)$$

Need to put in *official* format

$$\text{sum}(x, 0) = f(x)$$

$$\text{sum}(x, s(y)) = g(x, y, \text{sum}(x, y))$$

So let $f = \text{id}^1_1$ and $g = \text{Cn}[s, \text{id}^3_3]$

Then $\text{sum}(x, 0) = \text{id}^1_1(x)$

$\text{sum}(x, s(y)) = \text{Cn}[s, \text{id}^3_3](x, y, \text{sum}(x, y))$

Examples

- Given our formal recursive specification of **sum** as

$$(i) \text{ sum } (x, 0) = \mathbf{id}^1_1(x)$$

$$(ii) \text{ sum } (x, s(y)) = \mathbf{Cn}[s, \mathbf{id}^3_3](x, y, \text{sum } (x, y))$$

We can see that

$$(i) \text{ sum } (x, 0) = \mathbf{id}^1_1(x) = x \quad \text{and}$$

$$(ii) \text{ sum } (x, s(y)) = \mathbf{Cn}[s, \mathbf{id}^3_3](x, y, \text{sum } (x, y))$$

$$= s(\mathbf{id}^3_3(x, y, \text{sum } (x, y)))$$

$$= s(\text{sum } (x, y))$$

so that **sum** ($x, 0$) = x and **sum** ($x, s(y)$) = $s(\text{sum } (x, y))$

Officially: **sum** = $\mathbf{Pr}[\mathbf{id}^1_1, \mathbf{Cn}[s, \mathbf{id}^3_3]]$

Sample (**informal**) Computation with **sum**

- Recursively compute the value $2+3$, *i.e.* **sum** (2, 3):

$$\text{sum } (2, 0) = \mathbf{id}^1_1(2) = 2$$

$$\text{sum } (2, s(0)) = s(\text{sum}(2, 0)) = s(2) = 3$$

$$\text{sum } (2, s(s(0))) = s(\text{sum}(2, s(0))) = s(3) = 4$$

$$\text{sum } (2, s(s(s(0)))) = s(\text{sum}(2, s(s(0)))) = s(4) = 5$$

- It's mechanical!

Examples

- Product expressed recursively in terms of **sum**.
- Informally:

$$x \cdot 0 = 0$$

$$x \cdot s(y) = x + (x \cdot y)$$

Need to put in *official* format using **p.r.** functions f and g

$$\mathbf{prod}(x, 0) = f(x)$$

$$\mathbf{prod}(x, s(y)) = g(x, y, \mathbf{prod}(x, y))$$

So let $f = z$ and $g = \mathbf{Cn}[\mathbf{sum}, \mathbf{id}^3_1, \mathbf{id}^3_3]$

Then $\mathbf{prod}(x, 0) = z(x)$

$\mathbf{prod}(x, s(y)) = \mathbf{Cn}[\mathbf{sum}, \mathbf{id}^3_1, \mathbf{id}^3_3](x, y, \mathbf{prod}(x, y))$

Product Defined Recursively in Terms of **sum**

- Given our formal recursive specification of **prod** as

$$(i) \mathbf{prod}(x, 0) = \mathbf{z}(x)$$

$$(ii) \mathbf{prod}(x, s(y)) = \mathbf{Cn}[\mathbf{sum}, \mathbf{id}^3_1, \mathbf{id}^3_3](x, y, \mathbf{prod}(x, y))$$

We can see that

$$(i) \mathbf{prod}(x, 0) = \mathbf{z}(x) = 0 \quad \text{and}$$

$$(ii) \mathbf{prod}(x, s(y)) = \mathbf{Cn}[\mathbf{sum}, \mathbf{id}^3_1, \mathbf{id}^3_3](x, y, \mathbf{prod}(x, y))$$

$$= \mathbf{sum}(\mathbf{id}^3_1(x, y, \mathbf{prod}(x, y)), \mathbf{id}^3_3(x, y, \mathbf{prod}(x, y)))$$

$$= \mathbf{sum}(x, \mathbf{prod}(x, y))$$

so that $\mathbf{prod}(x, 0) = 0$ and $\mathbf{prod}(x, s(y)) = \mathbf{sum}(x, \mathbf{prod}(x, y))$.

- Officially: $\mathbf{prod} = \mathbf{Pr}[\mathbf{z}, \mathbf{Cn}[\mathbf{sum}, \mathbf{id}^3_1, \mathbf{id}^3_3]]$

Different Arities

- The **p.r.** schema has been given in terms of defining a 2-place function, but we can generalize to cover functions of any arity.
- For example, a **3-place** function $h(x_1, x_2, y)$ can be defined in terms of a **2-place** function f and a **4-place** function g such that

$$h(x_1, x_2, 0) = f(x_1, x_2)$$

$$h(x_1, x_2, s(y)) = g(x_1, x_2, y, h(x_1, x_2, y))$$

- And a **1-place** function $h(y)$ can be defined in terms of a constant c (i.e. a **0-place** function) and a **2-place** function $g(y, x)$ such that

$$h(0) = c$$

$$h(s(y)) = g(y, h(y))$$

Different Arities

- **So in the general case:**
an n -place function $\textcolor{violet}{h}^n$ is defined in terms of
an $n-1$ place function $\textcolor{teal}{f}^{n-1}$ and
an $n+1$ place function $\textcolor{red}{g}^{n+1}$,
such that $\textcolor{teal}{f}^{n-1}$ and $\textcolor{red}{g}^{n+1}$ are both **primitive recursive** and

$$\textcolor{violet}{h}^n(x_1, \dots, x_{n-1}, 0) = \textcolor{teal}{f}^{n-1}(x_1, \dots, x_{n-1})$$

$$\textcolor{violet}{h}^n(x_1, \dots, x_{n-1}, \mathbf{s}(y)) = \textcolor{red}{g}^{n+1}(x_1, \dots, x_{n-1}, y, \textcolor{violet}{h}^n(x_1, \dots, x_{n-1}, y))$$

written: $\textcolor{violet}{h} = \mathbf{Pr}[\textcolor{teal}{f}, \textcolor{red}{g}]$

Recursive Functions

- Now we will expand to the wider class of recursive functions: retain the same set of base functions, and all functions obtainable through finite applications of composition and primitive recursion plus the new operation of minimization.
- Minimization, when applied to a function f of $n+1$ arguments, yields the n -place function $\mathbf{Mn}[f]$ such that:
$$\begin{aligned}\mathbf{Mn}[f](x_1, \dots, x_n) &= \{\text{the least } y \text{ for which } f(x_1, \dots, x_n, y) = 0\} \\ &= \{\text{undefined if } f(x_1, \dots, x_n, y) = 0 \text{ for no } y\}\end{aligned}$$

Minimization

All p.r. functions are **total**, but **Mn** can yield **partial functions**.

Mn[sum] is a **partial** function:

$$\begin{aligned}\mathbf{Mn[sum]}(x) &= \{0, \text{ if } x = 0 \\ &= \{\text{undefined otherwise}\end{aligned}$$

In effect, **Mn** allows **unbounded search** – can't necessarily tell in a finite number of steps whether or not **Mn[*f*]** is defined on a given input.

If it is, then value will be computed in a finite number of steps.

If it is not, then computation won't halt.

Hence **bounded Minimization** is a **p.r.** operation (as we'll see a bit later).

More Primitive Recursive Functions

- We need recursive functions as defined through the operation of minimization in order to characterize the entire class of **computable functions**.
- However, the proper subclass of **p.r. functions** is quite vast and we will now continue investigating its members.
- Basic strategy is to use previously defined **p.r. functions** as ingredients for constructing progressively more complex **p.r. functions**.
- We've seen **sum** defined as **iterated** successor and **prod** defined as **iterated sum**. Can in turn define **exponentiation**, **exp**, as **iterated prod**:

More Primitive Recursive Functions

- Intuitively, $\text{exp}(x, y) = x^y$
which corresponds to the informal recursive specification:

$$x^0 = 1$$

$$x^{y+1} = x \cdot x^y$$

Or more officially $\text{exp}(x, 0) = 1$

$$\text{exp}(x, \text{s}(y)) = x \cdot \text{exp}(x, y)$$

Need *fully official* format using p.r. functions f and g

$$\text{exp}(x, 0) = f(x)$$

$$\text{exp}(x, \text{s}(y)) = g(x, y, \text{exp}(x, y))$$

More Primitive Recursive Functions

let $f = \mathbf{Cn}[\mathbf{s}, \mathbf{z}]$ and $g = \mathbf{Cn}[\mathbf{prod}, \mathbf{id}^3_1, \mathbf{id}^3_3]$

Then $\mathbf{exp}(x, 0) = \mathbf{Cn}[\mathbf{s}, \mathbf{z}](x)$

$\mathbf{exp}(x, \mathbf{s}(y)) = \mathbf{Cn}[\mathbf{prod}, \mathbf{id}^3_1, \mathbf{id}^3_3](x, y, \mathbf{exp}(x, y))$

Officially: $\mathbf{exp} = \mathbf{Pr}[\mathbf{Cn}[\mathbf{s}, \mathbf{z}], \mathbf{Cn}[\mathbf{prod}, \mathbf{id}^3_1, \mathbf{id}^3_3]]$

- The **predecessor** of x , written $\mathbf{pred}(x)$, is the number immediately preceding it (except we let $\mathbf{pred}(0) = 0$).

Informally, $\mathbf{pred}(0) = 0$, $\mathbf{pred}(\mathbf{s}(y)) = y$

So $\mathbf{pred}(0) = 0$

$\mathbf{pred}(\mathbf{s}(y)) = \mathbf{id}^2_1(y, \mathbf{pred}(y))$

Officially: $\mathbf{pred} = \mathbf{Pr}[0, \mathbf{id}^2_1]$

More Primitive Recursive Functions

- The **arithmetical difference** between x and y , written $\mathbf{dif}(x,y)$ (and abbreviated as $x \dot{-} y$) is defined as $x - y$ if $x \geq y$ and 0 otherwise.

So, in abbreviated format $x \dot{-} 0 = x$,

$$x \dot{-} s(y) = \mathbf{pred}(x \dot{-} y)$$

More formally, $\mathbf{dif}(x,0) = \mathbf{id}^1_1(x)$

$$\mathbf{dif}(x, s(y)) = \mathbf{Cn}[\mathbf{pred}, \mathbf{id}^3_3](x, y, \mathbf{dif}(x, y))$$

Officially: $\mathbf{dif} = \mathbf{Pr}[\mathbf{id}^1_1, \mathbf{Cn}[\mathbf{pred}, \mathbf{id}^3_3]]$

- The 1-place function **signum** is such that $\mathbf{signum}(0) = 0$ and $\mathbf{signum}(y) = 1$ otherwise.

Expressed **informally** (as a *composition*) $\mathbf{sg}(y) = 1 \dot{-} (1 \dot{-} y)$

More Primitive Recursive Functions

- The **reverse signum** function $\underline{\text{sg}}(y) = 1 \doteq y$
- Definition by cases. Suppose f is defined in the form:

$$\begin{aligned} f(x,y) &= \{g_1(x,y) \quad \text{if } C_1 \\ &\quad \vdots \\ &= \{g_n(x,y) \quad \text{if } C_n \end{aligned}$$

where C_1, \dots, C_n are mutually exclusive, collectively exhaustive conditions on x, y and g_1, \dots, g_n are **p.r.**

- The **characteristic function** of a condition C_i on x, y is a function c_i which takes the value 1 for argument pairs (x, y) which satisfy the condition, and the value 0 for all other argument pairs.

More Primitive Recursive Functions

- If the characteristic functions c_1, \dots, c_n of the conditions C_1, \dots, C_n in the foregoing definition are **p.r.** then so is the function f ,
for it can be defined by composition out of the g s and c s as follows:

$$f(x,y) = g_1(x,y) \cdot c_1(x,y) + \dots + g_n(x,y) \cdot c_n(x,y)$$

cool....

- **Example of definition by cases:** $\mathbf{max}(x,y)$ = the larger of x,y
So $\mathbf{max}(x,y) = \begin{cases} x & \text{if } x \geq y \\ y & \text{if } x < y \end{cases}$

In this case $g_1(x,y) = \mathbf{id^2}_1$,

$$g_2(x,y) = \mathbf{id^2}_2,$$

More Primitive Recursive Functions

$$c_1(x,y) = \underline{\text{sg}}(y \dot{-} x) = 1 \text{ if } x \geq y \text{ and } 0 \text{ otherwise}$$

$$c_2(x,y) = \underline{\text{sg}}(y \dot{-} x) = 1 \text{ if } x < y \text{ and } 0 \text{ otherwise}$$

Putting these ingredients together:

$$\mathbf{max}(x,y) = \mathbf{id^2}_1(x,y) \cdot \underline{\text{sg}}(y \dot{-} x) + \mathbf{id^2}_2(x,y) \cdot \underline{\text{sg}}(y \dot{-} x)$$

- Example: $\mathbf{max}(1,2) =$

$$\mathbf{id^2}_1(1,2) \cdot \underline{\text{sg}}(2 \dot{-} 1) + \mathbf{id^2}_2(1,2) \cdot \underline{\text{sg}}(2 \dot{-} 1)$$

$$= 1 \cdot \underline{\text{sg}}(1) + 2 \cdot \underline{\text{sg}}(1)$$

$$= 1 \cdot 0 + 2 \cdot 1$$

$$= 0 + 2$$

$$= 2$$

More Primitive Recursive Functions

- General sum: $g(x_1, \dots, x_n, y) = \sum_{i=0}^y f(x_1, \dots, x_n, i)$

Recursive definition (with $f(x_1, \dots, x_n, y)$ p.r.)

$$g(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n, 0)$$

$$g(x_1, \dots, x_n, s(y)) = g(x_1, \dots, x_n, y) + f(x_1, \dots, x_n, s(y))$$

- General product: $g(x_1, \dots, x_n, y) = \prod_{i=0}^y f(x_1, \dots, x_n, i)$

Recursive definition (with $f(x_1, \dots, x_n, y)$ p.r.)

$$g(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n, 0)$$

$$g(x_1, \dots, x_n, s(y)) = g(x_1, \dots, x_n, y) \cdot f(x_1, \dots, x_n, s(y))$$

More Primitive Recursive Functions

- Logical composition of conditions:
- Negation

If $c(x,y)$ is the characteristic function for condition C
then $\underline{c}(x,y) = \text{sg}(c(x,y))$ is the characteristic function for $\neg C$

- Conjunction

The characteristic function for $C_1 \wedge \dots \wedge C_n$ is

$$c_1(x,y) \cdot \dots \cdot c_n(x,y) \quad [= 0 \text{ if any of the terms are } 0]$$

Since $\{\neg, \wedge\}$ is a **truth-functionally adequate** set of logical connectives,

the above is sufficient to express **all** truth functional combinations of conditions.

More Primitive Recursive Functions

- For example, **Disjunction**

$$C_1 \vee C_2 \equiv \neg(\neg C_1 \wedge \neg C_2)$$

So the characteristic function of the disjunction of two conditions is

$$c_d(x,y) = \underline{\text{sg}}(\underline{\text{sg}}(c_1(x,y)) \cdot \underline{\text{sg}}(c_2(x,y)))$$

- **Bounded quantification:**
- **Universal** $\forall_i (i \leq y \rightarrow c(x, i))$

Characteristic function: $u(x,y) = \prod_{i=0}^y c(x,i)$

- **Existential** $\exists_i (i \leq y \wedge c(x, i))$

Characteristic function: $e(x,y) = \text{sg}(\sum_{i=0}^y c(x,i))$

More Primitive Recursive Functions

- **Bounded minimization:** with $\underline{f}(x_1, \dots, x_n, y)$ p.r.

$\mathbf{Mn}_w [\underline{f}]$ = least y such that $0 \leq y \leq w$ and

$$\underline{f}(x_1, \dots, x_n, y) = 0$$

Definition by cases:

$$\mathbf{Mn}_w [\underline{f}](x_1, \dots, x_n)$$

$$= \{0 \text{ if } \forall y (y \leq w \rightarrow \underline{f}(x_1, \dots, x_n, y) \neq 0\}$$

$$= \{\underset{i=0}{^w \Sigma} \mathbf{sg}(\underset{k=0}{^i \Pi} \underline{f}(x_1, \dots, x_n, k)) \text{ otherwise.}\}$$

Logic, Computability and Incompleteness

The Halting Problem and
Uncomputability

Idealized Computability

- We have been exploring a highly idealized mathematical characterization of computability in the guise of abstract Turing machines
in which there is no finite upper bound on input size, tape length, number of instructions defining the program, nor steps in the computation.
- Hence abstract TM's can perform computations that cannot possibly be performed by any physical machine, past, present or future.
- In this respect TM's provide an absolute mathematical limit on what is computable, in principle (assuming C-T Thesis)

Cardinality and Uncomputability

Nonetheless, considerations of brute cardinality reveal that some relevant phenomena must remain outside these highly idealized boundaries, and that even abstract Turing machines cannot compute all functions.

This is because the set of all TMs is enumerable, and hence so is the set of all Turing computable functions,

while, e.g., the set of all functions from positive integers \mathbf{P} to positive integers is not enumerable.

$$|\{\forall \text{ functions } f: \mathbf{P} \rightarrow \mathbf{P}\}| > |\{\forall f: f \text{ is Turing computable}\}|$$

- So it follows that there must \exists functions $f: \mathbf{P} \rightarrow \mathbf{P}$ which are **not** computed by any Turing machine.

Uncomputability via Diagonalization

- Mere cardinality considerations indicate that such functions must exist. We will now define a **specific function** $u: \mathbf{P} \rightarrow \mathbf{P}$ such that u cannot be computed by any Turing machine, **on pain of contradiction**.
 u will be defined using the **diagonal method**.
- So first need to give an enumeration of all TM's and specify u relative to this list.
- Each TM can be represented as a **word**, i.e. a **finite string** of symbols, in the enumerable alphabet

$R, L, S_0, q_1, S_1, q_2, S_2, q_3, S_3, \dots$

Enumerate all Turing Machines

- Only a (very!) proper subset of words in this alphabet will represent a TM.
- There are 5 effective constraints for determining which words denote TM's:
 - i) length of word is exactly divisible by 4.
 - ii) only the symbols q_1, q_2, q_3, \dots occur in positions
 $1, 4, 5, 8, 9, \dots, 4_n, \dots, 4_{n+1}, \dots$
 - iii) only the symbols S_0, S_1, S_2, \dots occur in positions
 $2, 6, 10, \dots, 4_{n+2}, \dots$
 - iv) only the symbols R, L, S_0, S_1, S_2, \dots occur in positions
 $3, 7, 11, \dots, 4_{n+3}, \dots$

Enumerate all Turing Machines

- v) no configuration of the form q_i, S_k occurs more than once in a word.
- Will now enumerate the set of all words in this alphabet, using only the characters ‘1’ and ‘2’ for coding and then decimal notation to define the ordering on the code:

1	2	3	4	5	6	Enumeration of
R	L	S_0	q_1	S_1	q_2	alphabet.

12 122 1222 12222 122222 1222222 Code for symbol

Enumerate all Turing Machines

- In this manner, the code for the n th symbol in the alphabet will be a 1 followed by n 2's.
- A **word** is a concatenation of symbols in the alphabet, and the **code for a word** is the concatenation of the codes for its constituent symbols.
- Finally, the ordering on the **words** is given by their codes arranged in **decimal notation**.
- This yields a gappy list of **words**, where w_{12} is the first entry.
- Given the enumeration of words, it is effective (and laborious!) to determine the corresponding list of TM's

Enumerate all Turing Machines

- The first word on the list that specifies a TM (i.e. which satisfies the 5 effective constraints given above) is w_n where $n = 1222212221212222$
- Decoding n we get 4314
 - = the string formed by concatenating the 4th 3rd 1st and 4th symbols of the alphabet
 - = $q_1 S_0 R q_1$
- Adopting our standard conventions of interpretation, this TM computes the identity function $i(x) = x$.

Enumerate all Turing Machines

- The second word on the list that specifies a TM is w_m where $m = 12222122212212222$
- Decoding m we get 4324
 - = the string formed by concatenating the 4th 3rd 2nd and 4th symbols of the alphabet
 - = $q_1 \ S_0 \ L \ q_1$
- Adopting our standard conventions of interpretation, this TM also computes the identity function $i(x) = x$.

Enumerate all Turing Machines

- Can systematically eliminate gaps in the list to get an enumeration of **all Turing machines** $\text{TM}_1, \text{TM}_2, \text{TM}_3, \dots$ which in turn gives an enumeration of **all functions** of one argument f_1, f_2, f_3, \dots such that $f_n : \mathbf{P} \rightarrow \mathbf{P}$.
- Since we now have a list, it's possible to use diagonalization to define the function u on positive integers, such that

$$\begin{aligned} u(n) &= \{1, \text{ if } f_n(n) \text{ is undefined} \\ &= \{f_n(n) + 1 \text{ otherwise} \end{aligned}$$

- Now $u : \mathbf{P} \rightarrow \mathbf{P}$ is a perfectly well defined **total** function on positive integers, but it can't appear anywhere on the list as, say, f_m , since this would entail a **contradiction** as follows:

Diagonalize

Suppose u were the m^{th} function f_m on the list, for some arbitrary m . Then by the definition of u ,

$$\begin{aligned} u(m) &= \{1, \text{ if } f_m(m) \text{ is undefined} \\ &= \{f_m(m) + 1 \text{ otherwise} \end{aligned}$$

But this is impossible, since if $f_m(m)$ is **undefined** then $u(m) = 1$ and $u(m) \neq f_m(m)$,

and if $f_m(m)$ is **defined** then $u(m) = f_m(m) + 1$ and again $u(m) \neq f_m(m)$.

So u cannot appear anywhere on the complete list of Turing computable functions, and hence u is **not** Turing computable.

Diagonalize

- Although no TM computes the **total** function u , it's easy to determine its *first few* values.
- As we've already seen, f_1 on the list = $f_2 = i$ = the identity function, so that $f_1(1) = 1$ and $f_2(2) = 2$. Hence $u(1) = 2$ and $u(2) = 3$.
- So why can't we go ahead and effectively determine $u(n)$ for any positive integer n ?
- If this were possible, it would refute the Church-Turing thesis by showing that u actually is computable, even though we have already demonstrated that u is **not Turing computable**.

The Halting Problem

- The essential issue is this: for any given TM on the list, say TM_n , is there an effective procedure for determining whether or not TM_n eventually halts in standard configuration when started in standard initial configuration scanning the leftmost of an unbroken string of n 1's?
- In other words, is there an effective procedure for determining whether or not $f_n(n)$ is defined, and if so, what its value is?
- If there is such a procedure, then the function u is indeed computable.
- So an equivalent question is, given an arbitrary TM started on an arbitrary input, can we determine in a finite number of steps whether or not it will ever halt?

The Halting Problem

- In simple cases we can often tell just by inspection that a machine will never halt, e.g.
 $q_1 \ S_1 \ S_1 \ q_1$ or $q_1 \ S_1 \ S_0 \ q_1 ; q_1 \ S_0 \ L \ q_1$
- But what if confronted by, e.g. TM_{15472} on the list, whose program is specified by say, 800 distinct quadruples.
- Is there a computable **halting function** that can determine the answer?
- It can be demonstrated by direct argument that no such halting function can be computed by any **Turing machine**.
- Consider the 2-place halting function h such that:

The Halting Problem

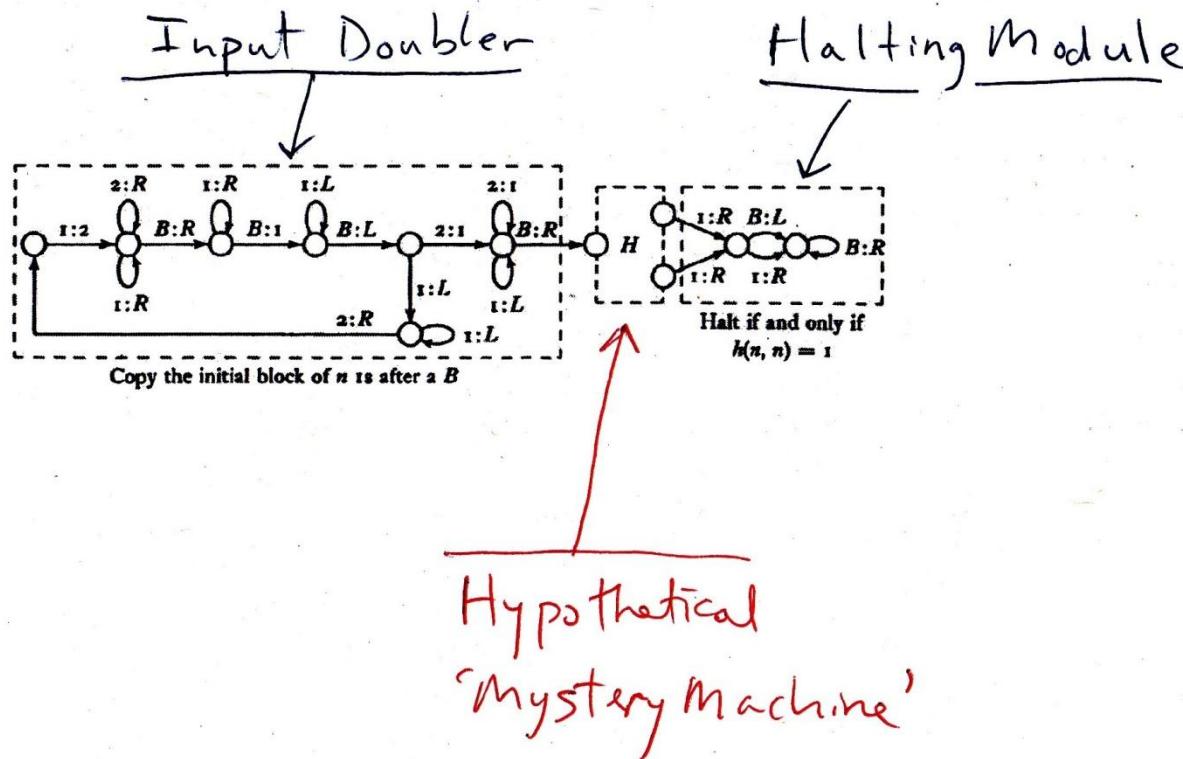
$h(m,n) = \begin{cases} 2, & \text{if machine number } m \text{ halts on input } n. \\ 1, & \text{if it never halts on input } n. \end{cases}$

- If h were computable then u would be computable too.
- Suppose that h were computed by some TM, say machine **H**.
Then we could construct the following machine **M***
such that for all positive integers n ,
M* halts on input n iff TM _{n} on the list does not.
In other words, **M*** halts on input n iff $h(n,n) = 1$.
M* is a composite of 3 machines: a ‘doubler’ which given
 $\underbrace{0111\dots1}_n 0$ as input yields $\underbrace{0111\dots1}_n \underbrace{0111\dots1}_n 0$ as output

The Halting Problem

- This is then fed into the hypothetical machine **H** as input, and lastly the output from **H** is fed into a ‘halting’ module.
- By the definition of **H**, it will output:
 - $0\underline{1}0$ if $h(n,n) = 1$, i.e. TM_n **does not halt** on input n
or
 - $0\underline{1}10$ if $h(n,n) = 2$, i.e. TM_n **does halts** on input n .
- The final ‘halting’ module of **M*** is such that it **halts** on input configuration $0\underline{1}0$ and it **runs on rightward forever** on input configuration $0\underline{1}10$.

Flow Graph of M^*



Self Destruct!

- Thus for all positive integers n , \mathbf{M}^* halts on input n if and only if the n^{th} TM on the list does not halt on n .
But, since \mathbf{M}^* is a TM, it is the k^{th} machine on the list for some positive integer k . So it is \mathbf{M}_k^* in the ordering.
By construction, \mathbf{M}_k^* must halt on input k if and only if it does not halt on input k !
- Hence no such machine \mathbf{M}^* can exist.
But the ‘doubler’ and ‘halting’ modules obviously do exist.
Conclusion: the hypothetical machine \mathbf{H} **cannot exist** and so the halting function h is **not Turing Computable**.

Back to the Beginning

- And if the Church-Turing Thesis is true, then the halting problem is **absolutely unsolvable**.
- An enumeration of a set Γ is effective iff Γ is finite, or else there is an explicit, mechanical procedure for determining the value $f(n) \in \Gamma$ in a finite number of steps, for every $n \in \mathbf{P}$.
- In the first lecture it was promised that we would encounter sets that are enumerable (because of their denumerable cardinality), but not effectively enumerable.
- Here's our first example: the set of Turing computable functions on the positive integers which are **not total** functions.

Logic, Computability and Incompleteness

First-order Logic Revisited

What is Logic?

A standard characterization:

Logic is the ‘science’ of valid arguments.

Modern ‘symbolic’ or formal logic is the *mathematical theory* of valid arguments

What is an argument?

Intuitively, an argument can be thought of as an inference, or piece of reasoning,

where certain statements are meant to support or establish a conclusion.

What is an Argument?

More precisely,

an argument is a finite set of (declarative) sentences, where one sentence is singled out as the **conclusion** and the other sentences are the *premises*.

Standard Argument Form:

Therefore, **Premise 1**
 Premise 2
 ...
 Premise *n*

Conclusion

What is Validity?

An argument is valid iff it is not possible for *all* the premises to be **true** and the conclusion **false**.

Alternatively, *if* all the premises *were true*,
then the conclusion *would have to be true* as well.

Some arguments are valid and some are not:

If it is snowing, then it is cold outside.

It is snowing.

Therefore, it is cold outside.

If the earth is round then the sky is blue.

The sky is blue.

Therefore the earth is round.

Some Examples

All politicians are human.

Some humans are wise.

Therefore, some politicians are wise.

If the sky is blue then the earth is flat.

The earth is not flat.

Therefore, the sky is not blue.

Some philosophers are politicians.

All politicians are corrupt.

Therefore, some philosophers are corrupt.

Either today is Wednesday or pigs can fly.

Pigs can't fly.

Therefore, today is Wednesday.

Formal Logic

As rational beings, we have *intuitions* about which arguments are valid and which are not.

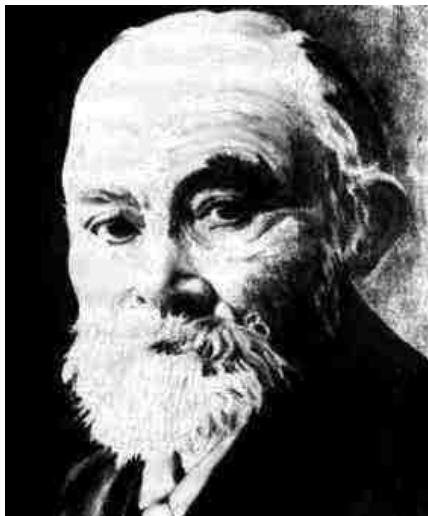
Modern logic provides a **mathematical theory** of validity whereby it is possible to **prove** that an argument is valid.

Although logic began as a branch of Philosophy and has been studied since ancient times, it underwent dramatic mathematical development in the 19th and 20th centuries.

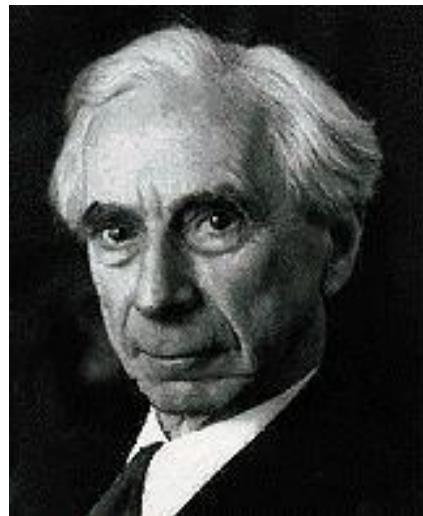
The biggest advance in logic for 2000 years was due to Gottlob Frege, who developed a system of quantifiers and variables to capture the logic of **generality** first explored by Aristotle.

Frege's system is now known as **First-order Logic**.

Some Famous Logicians



Gottlob
Frege



Bertrand
Russell



Kurt
Gödel



Alfred
Tarski

Natural vs Artificial Languages

The *medium* of logic is language.

When studying the grammar of a natural language such as English, we try to devise rules that accurately characterize a *pre-existing* phenomenon.

In contrast, artificial (or formal) languages, of the kind used in logic (and computer science), are **defined** by the grammatical rules we give.

Logic requires a precisely defined artificial **object language** in which formal arguments can be expressed and analyzed.

This step of ‘idealization’ is necessary to obtain mathematically rigorous results.

First-order Language

- **Generic First-order Syntax:**
- **Vocabulary:**
- Logical symbols: constants $\neg \vee \wedge \rightarrow \leftrightarrow \exists \forall =$
variables x, y, z, \dots
[and (,) for punctuation]
- Non-logical symbols:
 - individual constants $a, b, c \dots$
 - function symbols $f_1^n, f_2^n, f_3^n \dots$
 - sentence letters $Q, R, S \dots$
 - predicate letters $P_1^n, P_2^n, P_3^n \dots$

First-order Language

- A **language** L possesses a denumerable supply of non-logical symbols (and where each of the 4 categories above possess denumerably many elements).
- These **combine** with the logical symbols, according to the **Formation Rules**, to yield **terms** and **formulas** of the language:

Generic First-order Syntax

- **Formation Rules:**
- Definition of Terms and Formulas

Terms of L

(1) **Atomic terms**

- (i) all individual constants of L are terms of L
- (ii) all variables of L are terms of L

(2) **Compound terms**

if f^n is an n -place function symbol and t_1, \dots, t_n are terms, then $f^n(t_1, \dots, t_n)$ is a term of L (where $n > 0$)

- **Nothing else** is a term of L !

Generic First-order Syntax

Formulas of L

(1) **Atomic formulas:**

- (i) all sentence letters of L are formulas of L
- (ii) if P^n is an n -place predicate letter and t_1, \dots, t_n are terms, then $P^n t_1, \dots, t_n$ is a formula of L (where $n > 0$)

(2) **Compound formulas**

- (i) if Φ is a formula of L , then $\neg \Phi$ is a formula of L
- (ii) if Φ, Ψ are formulas of L , then $(\Phi \wedge \Psi), (\Phi \vee \Psi), (\Phi \rightarrow \Psi)$, and $(\Phi \leftrightarrow \Psi)$ are formulas of L .
- (iii) if Φ is a formula of L and v is a variable, then $\forall v \Phi$ and $\exists v \Phi$ are formulas of L .

Generic First-order Syntax

- **Nothing else** is a formula of L !
- Note that there are an infinite number of formulas, but **no** formula is infinitely long.
- Since formulas are **finite** sequences of symbols from a denumerable set,
a *language* L can have only denumerably many formulas.

Generic First-order Semantics

- **Generic First-order Semantics**
- An interpretation \mathcal{I} (or structure or model) of the language L is a way of “giving meaning” to the symbols of L .
- In particular, an interpretation \mathcal{I} of L specifies the following:
 - (i) a non-empty set D (the domain or universe of discourse).
 - (ii) for each individual constant c ,
 $\mathcal{I}(c)$ is an object $e \in D$.
 - (iii) for each n -ary function symbol f^n ,
 $\mathcal{I}(f^n)$ is an n -ary function $F^n: D^n \rightarrow D$
(where D^n is the n^{th} Cartesian product of D)

Generic First-order Semantics

(iv) for each sentence letter S

$\mathcal{J}(S)$ is a **truth value**, either **0** or **1**

(v) we treat '=' as a privileged 2-place predicate symbol,

where $\mathcal{J}(=)$ is the **set of all pairs** $\langle e, e \rangle$

such that $e \in D$.

(vi) for each n -ary predicate letter P^n [other than '=']

$\mathcal{J}(P^n)$ is a **set of ordered n -tuples**

such that $\mathcal{J}(P^n) \subseteq D^n$.

Generic First-order Semantics

The notation used in clause (iii), where $\mathcal{J}(f^n)$ is $F^n: D^n \xrightarrow{\text{green}} D$, connotes the fact that the interpretation of a function symbol f^n is an n -ary **function**

mapping n -tuples of elements of D to elements of D

while (vi) indicates that $\mathcal{J}(P^n)$ is an n -ary **relation**.

In general, n -place functions are equivalent to a (proper) subset of the set of $(n+1)$ -place relations:

- So $\mathcal{J}(f^n) \subset D^{n+1}$, with the constraint that for every n -tuple $\langle d_1, \dots, d_n \rangle \in D^n$,
there exists **exactly one** $d_{n+1} \in D$ such that
 $\langle d_1, \dots, d_n, d_{n+1} \rangle \in \mathcal{J}(f^n)$.

Example

- Consider the following very simple (fragment of an) interpretation \mathcal{J} as an illustration:
- Let the domain D of \mathcal{J} be the 2-member set { Jack, Jill }.
So $\mathcal{J}(=)$ is the set { <Jack, Jack>, <Jill, Jill> }
- Consider the first two individual constants c_1 and c_2
Let $\mathcal{J}(c_1) = \text{Jack}$ and $\mathcal{J}(c_2) = \text{Jill}$
- Consider the 1-place predicate symbol P^1
The definition requires that $\mathcal{J}(P^1) \subseteq D^1$
 $D^1 = \{\text{Jack}, \text{Jill}\}$ (which is equivalent to { <Jack>, <Jill> })
So let $\mathcal{J}(P^1) = \{\text{Jack}\}$

Example

- Consider the 2-place predicate symbol L^2
The definition requires that $\mathcal{I}(L^2) \subseteq D^2$
 $D^2 = \{\langle \text{Jack}, \text{Jack} \rangle, \langle \text{Jack}, \text{Jill} \rangle, \langle \text{Jill}, \text{Jack} \rangle, \langle \text{Jill}, \text{Jill} \rangle\}$
So let $\mathcal{I}(L^2) = \{\langle \text{Jack}, \text{Jack} \rangle, \langle \text{Jack}, \text{Jill} \rangle\}$
- Consider the 1-place function symbol f^1
The definition requires that $\mathcal{I}(f^1)$ is a 1-place function
 $F^1: D^1 \rightarrow D.$
As before, $D^1 = \{\text{Jack}, \text{Jill}\}$
So let $\mathcal{I}(f^1) = \{\langle \text{Jack}, \text{Jack} \rangle, \langle \text{Jill}, \text{Jack} \rangle\}$
- Note that the interpreted 2-place predicate L^2 cannot be used to define a corresponding 1-place function (why?).

Generic First-order Semantics

- In accord with clauses (ii)-(vi) above, the non-logical symbols of L (plus ‘=’) are given some fixed interpretation by \mathcal{I} .
- This then fixes the **truth value** of every **sentence** (i.e. **closed** formula – contains no *free variables*)
 Φ of L , **relative** to the interpretation \mathcal{I} .
- The formation rules entail that all sentences of L are either **atomic** or have one of the following **7 forms** (as determined by the main ‘connective’):
 $\neg \Phi, (\Phi \wedge \Psi), (\Phi \vee \Psi), (\Phi \rightarrow \Psi), (\Phi \leftrightarrow \Psi) \ \forall v\Phi, \exists v\Phi$

So the following **Rules of Truth** give the exhaustive procedures for computing the **truth value** (either **1** for **True** or **0** for **False**) for every sentence of L **relative** to a given interpretation \mathcal{I} :

Truth in an Interpretation

(I) If Φ is **atomic**, then it's either a sentence letter S or has the form $P^n t_1, \dots, t_n$

- (i) If Φ is a sentence letter S , then the truth value of Φ relative to \mathcal{J} , written $\mathcal{J}(\Phi)$ is simply $\mathcal{J}(S)$.
- (ii) If Φ has the form $P^n t_1, \dots, t_n$, then, because Φ is **closed**, the terms t_1, \dots, t_n must also be **closed**, and
 - $\mathcal{J}(\Phi) = 1$ iff $\langle \mathcal{J}(t_1), \dots, \mathcal{J}(t_n) \rangle \in \mathcal{J}(P^n)$ (and 0 otherwise), where, for each term t_i in the series t_1, \dots, t_n , if t_i is a constant c then $\mathcal{J}(t_i) = \mathcal{J}(c)$;
otherwise t_i is a k -place function term $f^k(t_1, \dots, t_k)$ applied to closed terms t_1, \dots, t_k and $\mathcal{J}(t_i) = \mathcal{J}(f^k)(\mathcal{J}(t_1), \dots, \mathcal{J}(t_k))$

Truth in an Interpretation

(II) For compound formulas:

$$(1) \mathcal{J}(\neg \Phi) = 1 \text{ iff } \mathcal{J}(\Phi) = 0 \text{ (and } 0 \text{ otherwise).}$$

$$(2) \mathcal{J}(\Phi \wedge \Psi) = 1 \text{ iff } \mathcal{J}(\Phi) = 1 \text{ and } \mathcal{J}(\Psi) = 1 \text{ (and } 0 \text{ otherwise).}$$

$$(3) \mathcal{J}(\Phi \vee \Psi) = 1 \text{ iff } \mathcal{J}(\Phi) = 1 \text{ or } \mathcal{J}(\Psi) = 1 \text{ (and } 0 \text{ otherwise).}$$

$$(4) \mathcal{J}(\Phi \rightarrow \Psi) = 1 \text{ iff } \mathcal{J}(\Phi) = 0 \text{ or } \mathcal{J}(\Psi) = 1 \text{ (and } 0 \text{ otherwise).}$$

$$(5) \mathcal{J}(\Phi \leftrightarrow \Psi) = 1 \text{ iff } \mathcal{J}(\Phi) = \mathcal{J}(\Psi) \text{ (and } 0 \text{ otherwise).}$$

Truth Table Format

 $\Phi \quad \neg \Phi$

1 **0**

0 **1**

 $\Phi \quad \Psi \quad (\Phi \wedge \Psi)$

1 **1** **1**

1 **0** **0**

0 **1** **0**

0 **0** **0**

 $\Phi \quad \Psi \quad (\Phi \vee \Psi)$

1 **1** **1**

1 **0** **1**

0 **1** **1**

0 **0** **0**

 $\Phi \quad \Psi \quad (\Phi \rightarrow \Psi)$

1 **1** **1**

1 **0** **0**

0 **1** **1**

0 **0** **1**

 $\Phi \quad \Psi \quad (\Phi \leftrightarrow \Psi)$

1 **1** **1**

1 **0** **0**

0 **1** **0**

0 **0** **1**

Truth in an Interpretation

(6) $\mathcal{I}(\forall v\Phi) = 1$ iff $\mathcal{I}_e^a (\Phi v/a) = 1$ for every $e \in D$,

where a is a new individual constant, \mathcal{I}_e^a is the interpretation exactly like \mathcal{I} except that $\mathcal{I}_e^a(a) = e$, and $\Phi v/a$ is the result of substituting a for every free occurrence of v in Φ (and 0 otherwise).

(7) $\mathcal{I}(\exists v\Phi) = 1$ iff $\mathcal{I}_e^a (\Phi v/a) = 1$ for some $e \in D$,

again where a is a new individual constant, \mathcal{I}_e^a is the interpretation exactly like \mathcal{I} except that $\mathcal{I}_e^a(a) = e$, and $\Phi v/a$ is the result of substituting a for every free occurrence of v in Φ (and 0 otherwise).

Example

- Consider the previous interpretation \mathcal{J} , where the domain D is the 2-member set $\{\text{Jack}, \text{Jill}\}$,

$$\mathcal{J}(c_1) = \text{Jack} \text{ and } \mathcal{J}(c_2) = \text{Jill}, \quad \mathcal{J}(P^1) = \{\text{Jack}\}$$

$$\mathcal{J}(L^2) = \{\langle \text{Jack}, \text{Jack} \rangle, \langle \text{Jack}, \text{Jill} \rangle\}$$

$$\mathcal{J}(f^1) = \{\langle \text{Jack}, \text{Jack} \rangle, \langle \text{Jill}, \text{Jack} \rangle\}$$

- Now consider the sentence $P^1 c_1$.

$$\mathcal{J}(c_1) = \text{Jack}, \quad \text{and} \quad \text{Jack} \in \{\text{Jack}\}$$

$$\text{Hence } \mathcal{J}(c_1) \in \mathcal{J}(P^1), \quad \text{so } \mathcal{J}(P^1 c_1) = 1$$

- Consider the sentence $P^1 c_2$.

$$\mathcal{J}(c_2) = \text{Jill}, \quad \text{and} \quad \text{Jill} \notin \{\text{Jack}\}$$

$$\text{Hence } \mathcal{J}(c_2) \notin \mathcal{J}(P^1), \quad \text{so } \mathcal{J}(P^1 c_2) = 0$$

Example

- Consider the sentence $P^1 f^1 (c_2)$

$$\mathcal{J}(f^1(c_2)) = \mathcal{J}(f^1)(\mathcal{J}(c_2)),$$

where $(\mathcal{J}(c_2)) = \text{Jill}$ and $\mathcal{J}(f^1)(\text{Jill}) = \text{Jack}$

As before, $\text{Jack} \in \{\text{Jack}\}$, hence $\mathcal{J}(f^1(c_2)) \in \mathcal{J}(P^1)$,

$$\text{so } \mathcal{J}(P^1 f^1 (c_2)) = 1$$

- Consider the sentence $L^2 c_2 c_1$

$$\mathcal{J}(c_2) = \text{Jill}, \mathcal{J}(c_1) = \text{Jack}$$

and $\langle \text{Jill}, \text{Jack} \rangle \notin \{\langle \text{Jack}, \text{Jack} \rangle, \langle \text{Jack}, \text{Jill} \rangle\}$

Hence $\langle \mathcal{J}(c_2), \mathcal{J}(c_1) \rangle \notin \mathcal{J}(L^2)$

$$\text{so } \mathcal{J}(L^2 c_2 c_1) = 0$$

Example

- Consider the sentence $\forall x P^1 x$

$$\mathcal{J}(\forall x P^1 x) = ?$$

i) $\mathcal{J}^a_{\text{Jack}}(P^1 a) = ?$

$$\mathcal{J}^a_{\text{Jack}}(a) = \text{Jack}$$

and $\text{Jack} \in \{\text{Jack}\}$, so $\mathcal{J}^a_{\text{Jack}}(a) \in \mathcal{J}(P^1)$,

$$\text{so } \mathcal{J}^a_{\text{Jack}}(P^1 a) = 1$$

ii) $\mathcal{J}^a_{\text{Jill}}(P^1 a) = ?$

$$\mathcal{J}^a_{\text{Jill}}(a) = \text{Jill}$$

and $\text{Jill} \notin \{\text{Jack}\}$, so $\mathcal{J}^a_{\text{Jill}}(a) \notin \mathcal{J}(P^1)$,

$$\text{so } \mathcal{J}^a_{\text{Jill}}(P^1 a) = 0$$

Hence $\mathcal{J}(\forall x P^1 x) = 0$

Example

- Consider the sentence $\exists x P^1 x$

$$\mathcal{J}(\exists x P^1 x) = ?$$

i) $\mathcal{J}^a_{\text{Jack}}(P^1 a) = ?$

$$\mathcal{J}^a_{\text{Jack}}(a) = \text{Jack}$$

and $\text{Jack} \in \{\text{Jack}\}$, so $\mathcal{J}^a_{\text{Jack}}(a) \in \mathcal{J}(P^1)$,

$$\text{so } \mathcal{J}(P^1 a) = 1$$

Hence $\mathcal{J}(\exists x P^1 x) = 1$

- Consider the sentence $\forall x \exists y L^2 xy$
- Consider the sentence $\exists x \forall y L^2 xy$

Example

- Consider the sentence $\textcolor{blue}{=} (f^1(c_2), c_1)$

$$\mathcal{J}(\textcolor{blue}{=} (f^1(c_2), c_1)) = ?$$

As before, $\mathcal{J}(f^1(c_2)) = \mathcal{J}(f^1)(\mathcal{J}(c_2)),$

where $(\mathcal{J}(c_2)) = \text{Jill}$ and $\mathcal{J}(f^1)(\text{Jill}) = \text{Jack}$

And $\mathcal{J}(c_1) = \text{Jack}$

so $\langle \mathcal{J}(f^1(c_2)), \mathcal{J}(c_1) \rangle$ is $\langle \text{Jack}, \text{Jack} \rangle.$

$\mathcal{J}(=)$ is the set $\{ \langle \text{Jack}, \text{Jack} \rangle, \langle \text{Jill}, \text{Jill} \rangle \}$

and $\langle \text{Jack}, \text{Jack} \rangle \in \{ \langle \text{Jack}, \text{Jack} \rangle, \langle \text{Jill}, \text{Jill} \rangle \}$

So $\langle \mathcal{J}(f^1(c_2)), \mathcal{J}(c_1) \rangle \in \mathcal{J}(=)$

and hence $\mathcal{J}(\textcolor{blue}{=} (f^1(c_2), c_1)) = \text{1}$

Truth in an Interpretation

- The ‘Mates Quantification’ scheme in clauses (6) and (7) uses substitution to attain the same semantical results as *variable interpretation sequences*.
On the Mates approach we don’t need to assign values to variables and we only ever need to consider the truth values of **closed formulas**.
- For the interpretation of n -ary predicate letters P^n , B&J explicitly assign a **characteristic function**, say \mathcal{C}^n , of n -tuples of elements of the domain.

Thus $\mathcal{I}(P^n) = \mathcal{C}^n$ and $\mathcal{I}(P^n t_1, \dots, t_n) = \mathcal{C}^n(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n))$.

This is equivalent to assigning a set of n -tuples, as above.

Some Standard Model-Theoretic Notions

- Some standard **model-theoretic notions**:
 - (i) \mathcal{I} **satisfies** (or is a model of) Φ iff $\mathcal{I}(\Phi) = 1$
 - (ii) Φ is **satisfiable** (or consistent) iff $\mathcal{I}(\Phi) = 1$ **for some** interpretation \mathcal{I}
 - (iii) Φ is **valid** (or a logical truth) iff $\mathcal{I}(\Phi) = 1$ **for every** interpretation \mathcal{I} . In this case we write $\models \Phi$
 - (iv) **logical implication**: $\Phi \models \Psi$ iff **for every** interpretation \mathcal{I} such that $\mathcal{I}(\Phi) = 1$, it's the case that $\mathcal{I}(\Psi) = 1$
in other words, **for all** interpretations \mathcal{I} , $\mathcal{I}(\Phi) \leq \mathcal{I}(\Psi)$
Clearly $\Phi \models \Psi$ iff $\models(\Phi \rightarrow \Psi)$, so logical implication can be expressed in terms of the validity of the material conditional.

Some Standard Model-Theoretic Notions

(v) **logical equivalence**: $\Phi \equiv \Psi$ iff for all interpretations \mathcal{J} ,
 $\mathcal{J}(\Phi) = \mathcal{J}(\Psi)$.

- Clearly $\Phi \equiv \Psi$ iff $\Phi \models \Psi$ and $\Psi \models \Phi$, iff $\models (\Phi \leftrightarrow \Psi)$
- Familiar generalization of logical implication to multiple premises: $\Phi_1, \dots, \Phi_n \models \Psi$ iff for all interpretations \mathcal{J} , if \mathcal{J} satisfies each of Φ_1, \dots, Φ_n then \mathcal{J} satisfies Ψ ,
iff $\models ((\Phi_1 \wedge \dots \wedge \Phi_n) \rightarrow \Psi)$
- For a set of sentences Γ , \mathcal{J} satisfies Γ iff $\mathcal{J}(\Theta) = 1$ for every $\Theta \in \Gamma$. \mathcal{J} is then a model of Γ .

The logical consequence relation extended to sets of sentences:

Some Standard Model-Theoretic Notions

- $\Gamma \models \Psi$ iff for every interpretation \mathcal{I} , if \mathcal{I} satisfies Γ then \mathcal{I} satisfies Ψ
- It is in the above format that we will think of valid arguments, recasting the ‘standard argument form’

Premise 1
Premise 2
...
Premise *n*

Therefore, **Conclusion**

so that Γ is the set of premises and Ψ is the conclusion.

A **sentence** Φ is valid iff $\emptyset \models \Phi$

Some Standard Model-Theoretic Notions

- Essential connection between **implication** and **satisfiability**:
 $\Gamma \models \Psi$ iff the set $\Gamma \cup \{\neg \Psi\}$ is **unsatisfiable**.
If the set $\Gamma \cup \{\neg \Psi\}$ did have a model then it would be a *counterexample* to the claim $\Gamma \models \Psi$.
- **Formal Theories**:
A Formal Theory T is a set of sentences (in some formal language L) which is closed under the relation of logical consequence.
So for all sentences Φ of L , if $T \models \Phi$ then $\Phi \in T$

More on Truth Functions

Definition: A **truth function** is a function whose only inputs and outputs are the truth values **T** and **F**.

The logical connectives \neg , \wedge , \vee , \rightarrow and \leftrightarrow represent truth functions. These truth functions are given by their truth tables.

For example, the negation truth function (which corresponds to our interpretation of ' \neg ') is a one-place function which maps **T** to **F** and maps **F** to **T**.

The conjunction truth function (our interpretation of ' \wedge ') is a two-place function which maps the pair of arguments (**T**, **T**) to **T** and maps all other pairs to **F**.

How Many Truth Functions are There?

An interesting theoretical question is:

“How many distinct truth functions are there
for n arguments?”.

It is easy to see that for 1 argument, there are 2 assignments,
and then 4 different truth functions, which we call J_1, J_2, J_3, J_4 .

Thus, here is a table of the **unary truth functions** (truth
functions of one argument)

A	J_1	J_2	J_3	J_4
T	T	T	F	F
F	T	F	T	F

How Many Truth Functions are There?

Similarly, for **binary truth functions** (2 arguments), there are $2^2 = 4$ assignments and $2^4 = 16$ truth functions.

Let's list all the truth functions of 2 arguments.

How Many Truth Functions are There?

A	B	K_{13}	K_{14}	K_{15}	K_{16}	K_{17}	K_{18}	K_{19}	K_{20}
T	T	T	F	T	F	T	F	T	F
T	F	T	T	F	F	T	T	F	F
F	T	T	T	T	T	F	F	F	F
F	F	F	F	F	F	F	F	F	F

The truth function K_{13} is the same as \vee ,

The truth function K_{19} is the same as \wedge ;

K_7 is the same as \rightarrow and K_{11} is the same as \leftrightarrow .

Later, we will see that there is something quite special about the truth functions K_6 and K_{12} .

How Many Truth Functions are There?

For **3** arguments, there are $2^3 = 8$ assignments, and $2^8 = 256$ truth functions.

In general, for **n** arguments, there are $2^n = k$ assignments and 2^k truth functions.

Since there is *no upper bound* on the number of inputs **n** , it follows that there are **infinitely many** distinct truth functions.

Truth-functional Adequacy

A logical language **L** is **truth-functionally adequate**
iff

for every one of the (infinitely many) truth functions that exist,
there is a **formula** of **L** that computes the function (i.e. has the
same **truth table**).

So a **truth-functionally adequate** language is powerful enough
to express all possible truth functions!

Question: is our language **L** truth-functionally adequate?

Answer: It turns out that a proper subset of the language is
enough...

Every truth function can be defined using just \neg , \wedge and \vee .

The Adequacy of $\{\neg, \wedge, \vee\}$

The first thing to notice is that we did not need to introduce the connectives \rightarrow and \leftrightarrow as primitive, because the truth functions represented by the connectives \rightarrow and \leftrightarrow are **definable** using $\{\neg, \wedge, \vee\}$.

What do we mean by **definable**?

Definition: A **2-place** connective **C** is **definable** using the set of connectives $\{C_1, C_2, \dots\}$

just in case the formula $C(A, B)$ is **logically equivalent** to some formula $[...A...B...]$,

where the expression $[...A...B...]$ contains only connectives from the set $\{C_1, C_2, \dots\}$.

Defining Truth Functions Using \neg , \wedge and \vee

So the definitions of the truth functions represented by the connectives \rightarrow and \leftrightarrow in terms of $\{\neg, \wedge, \vee\}$ can be given using **truth-functional equivalences**.

By using truth tables you can prove that,

$$\begin{array}{lll} \text{(i)} & A \rightarrow B & \equiv \\ \text{(ii)} & A \leftrightarrow B & \equiv \end{array}$$

$$\begin{array}{lll} \neg A \vee B & & \\ (A \wedge B) \vee (\neg A \wedge \neg B) & & \end{array}$$

Thus, both \rightarrow and \leftrightarrow are **definable** using $\{\neg, \wedge, \vee\}$.

The Adequacy of $\{\neg, \wedge, \vee\}$

The fact that **every truth function** can be defined
using just \neg, \wedge and \vee

can be established by providing a general procedure such that
given *any* arbitrary **n -place** truth function \mathcal{F} ,
it is possible to construct a formula \mathbf{A} ,
using only n distinct statement letters
and the connectives \neg, \wedge and \vee ,
such that \mathbf{A} has the *same truth table* as \mathcal{F} .

Example

Example: For the sake of illustration, let $n = 3$,

Now, consider the **arbitrary** 3-place truth function \mathcal{F} specified by the truth table:

3 inputs \mathcal{F}

T T T T

T T F T

3 inputs yields

T F T F

$2^3 = 8$ assignments

T F F F

yields

F T T T

$2^8 = 256$ truth functions:

F T F F

\mathcal{F} is just one of them.

F F T F

F F F F

Example

P	Q	R	\mathcal{F}
T	T	T	T ◀
T	T	F	T
T	F	T	F
T	F	F	F
F	T	T	T
F	T	F	F
F	F	T	F
F	F	F	F

Method of construction:

Consider the *first* assignment where
 \mathcal{F} yields the output T.

Make a conjunction of the 3
corresponding **literals**

(i.e. a statement letter or its negation)
in this case $P \wedge Q \wedge R$

P	Q	R	\mathcal{F}
---	---	---	---------------

T	T	T	T
T	T	F	T
T	F	T	F
T	F	F	F
F	T	T	T
F	T	F	F
F	F	T	F
F	F	F	F

Then:

Consider the *next* assignment where
 \mathcal{F} yields the output T.

Make a conjunction of the 3
corresponding **literals**,
in this case $P \wedge Q \wedge \neg R$

P	Q	R	\mathcal{F}
T	T	T	T
T	T	F	T
T	F	T	F
T	F	F	F
F	T	T	T ◀
F	T	F	F
F	F	T	F
F	F	F	F

Then:

Consider the *last* assignment where \mathcal{F} yields the output T.

Make a conjunction of the 3 corresponding **literals**,
in this case $\neg P \wedge Q \wedge R$

Finally, let the formula \mathbf{A} be the **disjunction** of all the **conjunctions** of 3 literals obtained from the input configurations where \mathcal{F} yields the output \mathbf{T} .

In this case \mathbf{A} is the formula

$$(\mathbf{P} \wedge \mathbf{Q} \wedge \mathbf{R}) \vee (\mathbf{P} \wedge \mathbf{Q} \wedge \neg\mathbf{R}) \vee (\neg\mathbf{P} \wedge \mathbf{Q} \wedge \mathbf{R})$$

And \mathbf{A} computes *exactly the same* truth function as \mathcal{F} !

This can be confirmed by examining its truth table...

P	Q	R	\mathcal{F}	$(P \wedge Q \wedge R) \vee (P \wedge Q \wedge \neg R) \vee (\neg P \wedge Q \wedge R)$			
T	T	T	T	T	F	F	T
T	T	F	T	F	T	F	T
T	F	T	F	F	F	F	F
T	F	F	F	F	F	F	F
F	T	T	T	F	F	T	T
F	T	F	F	F	F	F	F
F	F	T	F	F	F	F	F
F	F	F	F	F	F	F	F

▲

▲

Disjunctive Normal Form Theorem

The formula **A** is in **Disjunctive Normal Form**:

an overall disjunction of conjunctions of sentence literals.

The foregoing method of construction is entirely general,
and underwrites the

Disjunctive Normal Form Theorem:

for any formula **B** in our language **L** of propositional logic,
there is a formula **A** in Disjunctive Normal Form such that

$$\mathbf{A} \equiv \mathbf{B}.$$

Defining Truth Functions using \neg and one of \wedge, \vee

Just as in the case of \rightarrow and \leftrightarrow , it turns out that we don't need to take the entire set of connectives $\{\neg, \wedge, \vee\}$ as primitive: all we need is negation and one of our remaining binary connectives.

E.g. \vee is **definable** from the set $\{\neg, \wedge\}$ as follows:

$$A \vee B \equiv \neg(\neg A \wedge \neg B)$$

And similarly \wedge is **definable** from the set $\{\neg, \vee\}$ as follows:

$$A \wedge B \equiv \neg(\neg A \vee \neg B)$$

And this shows that the sets $\{\neg, \wedge\}$ and $\{\neg, \vee\}$ are both truth-functionally adequate.

Adequate Sets of Connectives (Cont.)

While $\{\neg, \wedge, \vee\}$, $\{\neg, \wedge\}$ and $\{\neg, \vee\}$ are all **adequate set of connectives**,

some sets of connectives are **not adequate**.

For example, the set $\{\wedge, \vee\}$ is **not** adequate.

You cannot define negation \neg using $\{\wedge, \vee\}$.

But $\{\neg, \wedge\}$ and $\{\neg, \vee\}$ are **not** the *smallest* adequate sets.

We will now show that one can find a **single** 2-place connective which is adequate

(in fact there are *two* such connectives).

That is, a single connective **K** such that *any* logical connective (representing any truth function) can be defined using just **K** alone.

Two New Connectives: NAND | and NOR ↓

Let us define two new truth-functional binary connectives:

NAND, meaning ‘not (... and ...)’

NOR, meaning ‘not (... or ...)’

$$\text{NAND}(A, B) \quad \text{written } A | B \quad \equiv \quad \neg(A \wedge B)$$

$$\text{NOR}(A, B) \quad \text{written } A \downarrow B \quad \equiv \quad \neg(A \vee B)$$

NAND | and NOR ↓

Given these definitions we can quickly figure out their truth tables

$$\neg(A \wedge B) \quad \neg(A \vee B)$$

A	B	A B	A↓B
T	T	F	F
T	F	T	F
F	T	T	F
F	F	T	T

These correspond to the truth functions we called K_6 and K_{12} above.

NAND { | } is an Adequate Set!

Every truth functional connective can be defined using { | } alone
Since we already know that { \neg , \wedge } is adequate, we just need to show how to define negation \neg and conjunction \wedge from |.

It will then immediately follow that { | } alone is adequate.

The crucial trick is to define negation \neg using |.

Consider the truth table for any formula of the form $A | A$.

A	$A A$
T	F
F	T

Thus, we see that

$$(1) \quad \neg A \equiv (A | A)$$

[A similar truth table will reveal that: $\neg A \equiv (A \downarrow A)$.]

NAND { | } is an Adequate Set

Next we want to define \wedge from $|$.

Since $(A | B) \equiv \neg(A \wedge B)$ and $A \wedge B \equiv \neg\neg(A \wedge B)$
it follows that $A \wedge B \equiv \neg(A | B)$.

Thus,

$$(2) \quad A \wedge B \quad \equiv \quad \neg(A | B) \quad \equiv \quad (A | B) | (A | B)$$

I.e., using the definition of \neg , we convert $\neg(...)$ to $(...) | (...)$.

(1) and (2) mean that both \neg and \wedge can be defined using $|$ alone.

From this it follows that { | } alone is an adequate set.

NAND { | } is an Adequate Set

How do we find a formula equivalent to $A \vee B$?

$$\begin{aligned} A \vee B &\equiv \neg(\neg A \wedge \neg B) \\ &\equiv \neg((A | A) \wedge (B | B)) \\ &\equiv (A | A) | (B | B) \end{aligned}$$

So, we have

$$(3) \quad A \vee B \equiv \neg((A|A) \wedge (B|B)) \equiv (A | A) | (B | B)$$

Exactly analogous reasoning involving $\{\neg, \vee\}$ shows that $\{\downarrow\}$ is also adequate.

Historical Note

- The American logician Charles Saunders Peirce discovered the truth-functional adequacy of both NAND and NOR in 1880, but never published his findings.
- Henry Sheffer independently published results on the adequacy of NOR in 1913, whence the ‘Sheffer stroke’.
- The NAND logic gate is crucial to modern digital electronics, and plays a vital role in computer processor design.

Prenex Normal Form

- **Prenex Form:**

$Q_1 v_1, \dots, Q_n v_n \Phi$ where $Q_i = \forall$ or \exists and Φ is quantifier free

Theorem: for every formula Ψ there is some formula Θ such that Θ is in prenex form and $\Psi \equiv \Theta$.

Proof: give rules for successively moving quantifiers to the left which preserve logical equivalence:

(i) quantifier duality: $\neg Q v \Phi \equiv Q' v \neg \Phi$

where $Q = \forall$ or \exists , and $\forall' = \exists$, $\exists' = \forall$

(ii) can directly pull quantifiers out from conjunctions, disjunctions, and the consequents of conditionals (provided v does not occur free in Ψ):

Prenex Normal Form

- (1) $(Qv \Phi \wedge \Psi) \equiv Qv (\Phi \wedge \Psi)$
 $(\Psi \wedge Qv \Phi) \equiv Qv (\Psi \wedge \Phi)$
- (2) $(Qv \Phi \vee \Psi) \equiv Qv (\Phi \vee \Psi)$
 $(\Psi \vee Qv \Phi) \equiv Qv (\Psi \vee \Phi)$
- (3) $(\Psi \rightarrow Qv \Phi) \equiv Qv (\Psi \rightarrow \Phi)$
- (4) But must **reverse** quantifier in the antecedent
 $(Qv \Phi \rightarrow \Psi) \equiv Q'v (\Phi \rightarrow \Psi)$
Because $(Qv \Phi \rightarrow \Psi) \equiv (\neg Qv \Phi \vee \Psi) \equiv (Q'v \neg \Phi \vee \Psi) \equiv Q'v (\neg \Phi \vee \Psi) \equiv Q'v (\Phi \rightarrow \Psi) \blacksquare$

Examples

- $\forall y Qy \vee \neg \exists x Px$
$$\equiv \forall y Qy \vee \forall x \neg Px$$
$$\equiv \forall y (Qy \vee \forall x \neg Px)$$
$$\equiv \forall y \forall x (Qy \vee \neg Px)$$
- $\exists x Fx \rightarrow \neg \forall y Py$
$$\equiv \exists x Fx \rightarrow \exists y \neg Py$$
$$\equiv \forall x (Fx \rightarrow \exists y \neg Py)$$
$$\equiv \forall x \exists y (Fx \rightarrow \neg Py)$$
- We will use prenex normal form in our forthcoming deductive system.

Logic, Computability and Incompleteness

Binary Relations

Binary Relations

2-place or **Binary Relations** are fundamental
to human language, thought and reasoning.

(i) transitive verbs:

‘ x loves y ’, ‘ x embraces y ’, ‘ x is acquainted with y ’, etc.;

(ii) comparatives and adverbial comparisons:

‘ x is taller than y ’, ‘ x is smarter than y ’, etc.;

(iii) family relationships:

‘ x is an uncle of y ’, ‘ x is a sister of y ’, etc.;

(iv) functional relationships (in mathematics):

‘ x is the square of y ’, ‘ x is the tangent of the angle y ’,

‘ x is the (positive) square root of y ’, etc.

Properties of Binary Relations

We know **a priori** that:

Alice is not **taller than** herself;

If Alice is **taller than** Bill, then Bill is **not taller than** Alice;

If Alice is **taller than** Bill and Bill is **taller than** Carol,
then Alice is **taller than** Carol.

The relation ***taller than*** is

- **irreflexive, asymmetric and transitive.**

Reflexivity

Definition A relation R is:

- (i) **reflexive** iff $\forall x Rxx$,
- (ii) **irreflexive** iff $\forall x \neg Rxx$

Examples of **reflexive** relations:

‘ x is identical to y ’, ‘ x is the same age as y ’.

irreflexive relations:

‘ x is the sister of y ’, ‘ x is taller than y ’,
‘the number x is smaller than the number y ’.

Symmetry

Definition: A relation R is

- (i) **symmetric** iff $\forall x \forall y (Rxy \rightarrow Ryx)$;
- (ii) **asymmetric** iff $\forall x \forall y (Rxy \rightarrow \neg Ryx)$,

Examples of symmetry:

‘ x is a spouse of y ’, ‘ x is a sibling of y ’

‘ x and y are 1 metre apart’.

Examples of asymmetry: ‘ $x < y$ ’

Comparatives: ‘ x is larger than y ’, ‘ x is taller than y ’, etc.

Certain family relations are **asymmetric**: ‘ x is the mother of y ’.

Transitivity

Definition: A relation R is **transitive** iff

$$\forall x \forall y \forall z ((Rxy \wedge Ryz) \rightarrow Rxz).$$

Comparatives are **transitive**.

‘ x is taller than y ’, ‘ x is older than y ’,

‘the number x is less than the number y ’ (i.e., ‘ $x < y$ ’)

Example from the biological world:

‘ x is an ancestor of y ’.

Example from the physical world:

‘event x is earlier (in time) than event y ’

Orderings

Ordering relations.

- irreflexive,
- asymmetric
- transitive.

Technical name: (strict) **partial orderings**.

The system **N** of natural numbers is **ordered**.

$$0 < 1 < 2 < 3 < \dots$$

- the rationals **Q** (ratios of natural numbers)
- the real numbers **R** (rationals plus irrationals)

A (strict) **partial ordering** is any **transitive irreflexive relation** (asymmetry follows as a logical consequence).

Equivalence Relations

- **Definition:** A relation that is **reflexive**, **symmetric**, and **transitive** is said to be an **equivalence relation**.

Examples:

x is the same height as y ,

x is the same age as y ,

x has the same surname as y ,

x is parallel to y ,

Φ is logically equivalent to Ψ

Other Structural Properties

Connectedness: $\forall x \forall y (Rxy \vee Ryx)$

Density: $\forall x \forall y (Rxy \rightarrow \exists z (Rxz \wedge Rzy))$

Seriality: $\forall x \exists y Rxy$

Others can be expressed with the notion of **identity**:

Trichotomy: $\forall x \forall y (Rxy \vee x = y \vee Ryx)$

The Concept of Identity

Identity is a fundamental notion in human thought and reasoning. It's normally expressed using the 2-place predicate symbol ‘=’ to mean *x is identical to y*

Identity is treated as a **binary relation**, but is **logically/semantically** privileged.

The defining properties of the identity relation are considered to be purely **logical**.

In this sense, the identity predicate ‘ $x = y$ ’ is the **only predicate** that itself belongs to **logic**.

All other predicates (like ‘**x is a brother of y**’) are treated as **non-logical**.

The Concept of Identity

- The concept of identity can be explained in terms of certain **a priori** logical principles.

(I) **The Principle of Self-Identity:**

Every object is identical to itself.

$$\forall x(x = x).$$

(II) **The Indiscernibility of Identicals:**

If entities x and y are identical, and property P is true of x , then P is true of y .

$$\forall x \forall y [(x = y \wedge (P x)) \rightarrow P(y)].$$

The Indiscernibility of Identicals was originally stated by the German mathematician and philosopher Gottfried Leibniz.

Second-order Logic

- Note that in the statement (II) above we utilize the meta-linguistic variable ‘ P ’, which is not part of our formal object language.
- Implicitly, we are making a universal quantification over all properties of individuals. This transcends the boundaries of **first-order** logic, in which the quantifiers range over individuals in the domain of discourse.
- However, the Indiscernibility of Identicals can be formalized in **second-order** logic.
- Let P be a second-order variable ranging over properties of individuals. Then (II) is rendered as the closed second-order formula: $\forall P \forall x \forall y ((x = y \wedge P(x)) \rightarrow P(y))$.

The Concept of Identity

- Given the Indiscernibility of Identicals, if there is a property P had by some object a that is not had by an object b , then it follows by *deductive logic* that $a \neq b$.

In this case P is a discerning property for \mathbf{a} and \mathbf{b} .

For example, suppose you want to *prove* that

Donald Trump is **not identical to** Boris Johnson.

The following is a logically valid argument:

Boris Johnson attended Eton College.

Donald Trump did **not** attend Eton College.

Therefore Donald Trump \neq Boris Johnson.

The Concept of Identity

As mentioned above, a relation R is an **equivalence relation** just in case it is

- reflexive $\forall x Rxx.$
- symmetric $\forall x \forall y (Rxy \rightarrow Ryx).$
- transitive $\forall x \forall y \forall z ((Rxy \wedge Ryz) \rightarrow Rxz).$

So **identity** is an **equivalence relation** (and can be defined as the “smallest” equivalence relation.)

Expressing Cardinality in FOL

- With the use of $=$ we can construct sentences which are true only in models of some specific **finite cardinality**.
For example, $\exists x \forall y (y = x)$ is an ‘axiom’ **true** only in interpretations with **1-element** domains
 $\exists x \exists y (x \neq y)$ is an axiom **true** only in interpretations with **at least 2 element** domains.
 $\exists x \exists y \forall z (z = x \vee z = y)$ is **true** only in interpretations with **at most 2 element** domains.
while $\exists x \exists y (x \neq y \wedge \forall z (z = x \vee z = y))$ is **true** only in interpretations with **exactly 2 element** domains.
Obviously these patterns generalize to **any positive integer n** .

Expressing Cardinality in FOL

- Russell showed that we can construct a sentence that is true for **no finite** n , and hence forces the domain to be **infinite**.

$\forall x \forall y \forall z ((Rxy \wedge Ryz) \rightarrow Rxz)$ expresses the property of **transitivity**,

$\forall x \forall y (Rxy \rightarrow \neg Ryx)$ expresses the property of being **asymmetric**, and

$\forall x \exists y Rxy$ expresses the property of being **serial**.

The set Δ containing these 3 sentences as elements is **satisfiable**,

but Δ is modelled by **no** interpretation with a **finite domain**.

Expressing Cardinality in FOL

- Equivalently, we can conjoin them into a single sentence to get Russell's **Axiom of Infinity**:

$$\forall x \forall y \forall z ((Rxy \wedge Ryx) \rightarrow Rxz) \wedge \forall x \forall y (Rxy \rightarrow \neg Ryx) \wedge \forall x \exists y Rxy$$

If \mathcal{J} is a model of Δ , then the extension $\mathcal{J}(R)$ of the binary predicate symbol R must be a **transitive**, **asymmetric** and **serial** relation.

And if \mathcal{J} is a model of Δ , then its domain D must be **infinite**.

Proof: Let a_1 be one of the objects in D .

By **seriality**, there is an object, a_2 say, in the domain such that

$$Ra_1a_2.$$

Expressing Cardinality in FOL

Any **asymmetric** relation is **irreflexive** (will be able to prove this in our formal system):

so $a_1 \neq a_2$.

By **seriality** again, there's an object a_3 in the domain such that Ra_2a_3 .

By **transitivity**, Ra_1a_3 .

By **irreflexivity**, we have it that $a_1 \neq a_3$ and $a_2 \neq a_3$.

[**Summary so far:** the domain must contain at least **3** distinct objects: a_1, a_2, a_3 .]

By **seriality** again, there's an object a_4 in the domain such that Ra_3a_4 .

Expressing Cardinality in FOL

By **transitivity**, Ra_1a_4 and Ra_2a_4 .

By **irreflexivity** again, $a_1 \neq a_4$, $a_2 \neq a_4$, and $a_3 \neq a_4$.

[Summary so far: the domain must contain at least **4** distinct objects: $a_1, a_2, a_3, a_4.$]

And so on....

By repeating this reasoning, there must be **infinitely many distinct objects:**

$a_1, a_2, a_3, a_4, a_5, \dots, a_n, \dots \blacksquare$

Logic, Computability and Incompleteness

The Halting Problem and
Undecidability

Decidability

We have seen that the halting function h is **not Turing Computable**

and that if the **Church-Turing Thesis** is true, then the **halting problem** is **absolutely unsolvable**.

- We will now extend these negative, limitative results to the realm of **First-Order Logic** (FOL) and show that FOL is **undecidable**.
- In general, given some set of objects of the relevant sort, a property \mathcal{P} of such objects is **decidable** just in case there is an effective, mechanical procedure such that, for any arbitrary object of the relevant sort,

Decidability

the procedure eventually classifies the object (*correctly!*) as a positive or negative instance of that property.

- More specifically,
 - (i) if a given object **has** the property \mathcal{P} , then the procedure determines in finitely many steps that **it has** the property,
 - (ii) if the object **does not have** the property \mathcal{P} , then the procedure determines in finitely many steps that **it does not have** the property.
- The property \mathcal{P} is **decidable iff** we have **both** (i) and (ii).

Decidability

For example, let Ψ be an arbitrary (finite) string of symbols from the primitive vocabulary of our formal object language L for FOL.

Now consider the property of

being a grammatically well formed formula of L .

It turns out (unsurprisingly!) that this property is **decidable**.

So, if Ψ is some arbitrary (finite) expression, we can decide in finitely many steps whether or not Ψ is a formula.

Logical Validity

- In the present context, objects of the relevant sort are **closed formulas** (i.e. sentences) of the language **L** of FOL and the property of interest is **satisfiability/logical validity**.
- We will show that there is no mechanical positive test for FOL **satisfiability**, and hence, equivalently, for **FOL invalidity**
- Essential connection between **validity** and **satisfiability**:
 $\Gamma \vDash \Psi$ iff the set $\Gamma \cup \{\neg \Psi\}$ is **unsatisfiable**.
If the set $\Gamma \cup \{\neg \Psi\}$ *did* have a model then it would be a counter-model to the claim $\Gamma \vDash \Psi$.
Thus the **satisfiability** of $\Gamma \cup \{\neg \Psi\}$ is tantamount to **invalidity** and means that $\Gamma \not\models \Psi$.

Logical Validity

- We will soon prove that FOL is **complete**, since there **is** an effective positive method for determining **validity**.
- But this is only clause (i) of the decidability problem, and as we will now show, clause (ii) fails, since there is no effective method for determining the negative cases of **invalidity**.
- Hence **validity overall** in FOL is **undecidable**.
- We'll use a reductio argument to show that there can be no effective method for determining cases of **invalidity**, by showing that if there were such a test, then the halting problem for Turing Machines would be solvable (and hence the Church-Turing Thesis would be false)

Logical Validity and the Halting Problem

- The strategy is analogous to the demonstration that Turing computable functions are Recursive.
- There, we constructed recursive functions which **numerically replicate** the sequence of configurations of any given TM computation.
- Here, we'll construct a finite set of **FOL sentences** which exactly characterize or **formalize** the sequence of configurations.
- We'll specify a method such that, given any TM and any input n , we can construct a finite set of sentences Δ and a corresponding sentence H such that

$\Delta \vDash H$ iff the TM eventually halts on input n (!)

Logical Validity and the Halting Problem

- Thus **validity** in FOL is directly correlated with **halting**, and **invalidity** with **non-halting**.
- So *if* validity in FOL were decidable *then* we could solve the halting problem and compute h , as follows:
- Given the m^{th} TM, say TM_m , and a selected input n , we construct the set of sentences Δ .
Given the specification of TM_m we also construct H .
 - Then,
 - if $\Delta \models H$ then $h(m,n) = 2$ and
 - if $\Delta \not\models H$ then $h(m,n) = 1$
 - We could thereby compute the (provably) non-Turing computable function u as well.

Formalizing TM Computations in FOL

- To begin the construction, let all the squares of the machine tape be numbered, with 0 as the starting square of the computation:

$\dots | -2 | -1 | 0 | 1 | 2 | 3 | \dots$ [this number is distinct from
the symbol occurring in the square]

- Let t be the ‘time’ variable ranging over steps in the computation
- FOL Vocabulary:
 - For each state q_i of machine \mathbf{M} , pick a 2-place predicate Q_i
 - For each symbol S_j the machine can read/write, pick a 2-place predicate S_j

Formalizing TM Computations in FOL

- Also need \mathbf{o} (a selected individual constant), $'$ (a 1-place function), $<$ (a 2-place relation), and $=$
- Intended interpretation \mathcal{J} : D = set of integers
 $tQ_i x$ is true iff at time t \mathbf{M} is in state q_i scanning square number x
 $tS_j x$ is true iff at time t the symbol S_j is in square number x .
The interpretations of \mathbf{o} , $'$, $<$ are standard.
- With the intended interpretation \mathcal{J} in mind, and the supposition that it can read/write only the symbols S_0, \dots, S_r , will now axiomatize a given machine's operation as follows.
There are only 3 types of instruction in \mathbf{M} 's specification:

Formalizing TM Computations in FOL

- $q_i S_j S_k q_m$

Corresponding axiom

$$\forall t \forall x \forall y [(tQ_i x \wedge tS_j x) \rightarrow (t'Q_m x \wedge t'S_k x \wedge (y \neq x \rightarrow (tS_0 y \rightarrow t'S_0 y \wedge \dots \wedge tS_r y \rightarrow t'S_r y)))]$$

Under the intended interpretation, this axiom says:

If machine \mathbf{M} is in state q_i at time t scanning square x on which the symbol S_j occurs, then at time $t+1$ \mathbf{M} is in state q_m scanning square x where the symbol S_k occurs, and in all squares other than x , the same symbols appear at time $t+1$ as at time t .

Formalizing TM Computations in FOL

- $q_i \text{ } S_j \text{ } R \text{ } q_m$

axiom:

$$\forall t \forall x \forall y [(tQ_i x \wedge tS_j x) \rightarrow (t'Q_m x' \wedge (tS_0 y \rightarrow t'S_0 y \wedge \dots \wedge tS_r y \rightarrow t'S_r y))]$$

- $q_i \text{ } S_j \text{ } L \text{ } q_m$

axiom:

$$\forall t \forall x \forall y [(tQ_i x' \wedge tS_j x') \rightarrow (t'Q_m x \wedge (tS_0 y \rightarrow t'S_0 y \wedge \dots \wedge tS_r y \rightarrow t'S_r y))]$$

Add all such axioms to Δ according to the set of quadruples defining M .

Formalizing TM Computations in FOL

- Now formalize the initial configuration for input n (where $t = 0$, $x = 0$ and $\text{state} = q_1$)
$$\mathbf{oQ}_1\mathbf{o} \wedge \mathbf{oS}_1\mathbf{o} \wedge \mathbf{oS}_1\mathbf{o}' \wedge \dots \wedge \mathbf{oS}_1\mathbf{o}^{(n-1)} \wedge$$
$$\forall y ((y \neq \mathbf{o} \wedge y \neq \mathbf{o}' \wedge \dots \wedge y \neq \mathbf{o}^{(n-1)}) \rightarrow \mathbf{oS}_0y))$$
- Finally need to throw in two arithmetical axioms to govern behavior of ' and <
- First axiom says that each integer is the successor of exactly one integer: $\forall z \exists x (z = x') \wedge \forall z \forall x \forall y ((z = x' \wedge z = y') \rightarrow x = y))$
- Second axiom:
$$\forall x \forall y \forall z (x < y \wedge y < z \rightarrow x < z) \wedge \forall x \forall y (x' = y \rightarrow x < y)$$
$$\wedge \forall x \forall y (x < y \rightarrow x \neq y)$$

Formalizing TM Computations in FOL

- This yields the set Δ formalizing the machine \mathbf{M} started in initial configuration on input n .
- Next need to define H . Note that a machine halts at time t iff it is in state q_i reading a symbol S_j and there is no quadruple beginning with the pair $q_i S_j$.
- There will be a **finite** number of such possibilities, and the pairs for any given machine can be determined by inspection.
- For each such pair $q_i S_j$, construct the sentence

$$\exists t \exists x (tQ_i x \wedge tS_j x)$$

And let H be the disjunction of all such existential sentences for a given machine \mathbf{M} .

$$\Delta \models H \text{ iff } \mathbf{M} \text{ halts on input } n$$

- If there are no such halting pairs, then \mathbf{M} never halts, and so let H be $\mathbf{o} \neq \mathbf{o}$.
- The construction is now complete!
- Claim: Given machine \mathbf{M} and input n , we have engineered Δ and H such that

$$\Delta \models H \text{ iff } \mathbf{M} \text{ eventually halts on input } n$$

- Verification of claim: the ‘only if’ direction is trivial: it’s clear that *if* $\Delta \models H$, *then* \mathbf{M} eventually halts on input n , since the intended interpretation \mathcal{I} is a model of Δ , and H is true in \mathcal{I} iff \mathbf{M} halts.

if M halts on n , *then* $\Delta \models H$

- The harder part of the verification is the ‘*if*’ direction, i.e. to show that *if* M halts on n , *then* $\Delta \models H$.
- Recall that a TM computation is a sequence of configurations, where a configuration includes the current state, scanned square on the tape, and the contents of all non-blank squares on the tape.
- So *a description of time s* is a characterization of the configuration at stage s in the computation.
- A *description of time s* will be a conjunction of the form:
$$\mathbf{o}^{(s)} Q_i \mathbf{o}^{(p)} \wedge \mathbf{o}^{(s)} S_{j1} \mathbf{o}^{(p1)} \wedge \dots \wedge \mathbf{o}^{(s)} S_j \mathbf{o}^{(p)} \wedge \dots \wedge \mathbf{o}^{(s)} S_{jv} \mathbf{o}^{(pv)} \wedge$$
$$\forall y [(y \neq \mathbf{o}^{(p1)} \wedge y \neq \mathbf{o}^{(p)} \wedge \dots \wedge y \neq \mathbf{o}^{(pv)}) \rightarrow \mathbf{o}^{(s)} S_0 y]$$

What does it mean?

$$\mathbf{o}^{(s)} \mathbf{Q}_i \mathbf{o}^{(p)} \wedge \mathbf{o}^{(s)} \mathbf{S}_{j1} \mathbf{o}^{(p1)} \wedge \dots \wedge \mathbf{o}^{(s)} \mathbf{S}_j \mathbf{o}^{(p)} \wedge \dots \wedge \mathbf{o}^{(s)} \mathbf{S}_{jv} \mathbf{o}^{(pv)} \wedge \\ \forall y [(y \neq \mathbf{o}^{(p1)} \wedge y \neq \mathbf{o}^{(p)} \wedge \dots \wedge y \neq \mathbf{o}^{(pv)}) \rightarrow \mathbf{o}^{(s)} \mathbf{S}_0 y]$$

1st conjunct $\mathbf{o}^{(s)} \mathbf{Q}_i \mathbf{o}^{(p)}$: current state and scanned square at time s

The next string of conjuncts

$$\mathbf{o}^{(s)} \mathbf{S}_{j1} \mathbf{o}^{(p1)} \wedge \dots \wedge \mathbf{o}^{(s)} \mathbf{S}_j \mathbf{o}^{(p)} \wedge \dots \wedge \mathbf{o}^{(s)} \mathbf{S}_{jv} \mathbf{o}^{(pv)}$$

gives the non-empty tape contents. This occurs in the range squares $p_1, \dots, p, \dots, p_v$, where the central term $\mathbf{o}^{(s)} \mathbf{S}_j \mathbf{o}^{(p)}$ indicates the symbol in the currently scanned square.

Finally $\forall y [(y \neq \mathbf{o}^{(p1)} \wedge y \neq \mathbf{o}^{(p)} \wedge \dots \wedge y \neq \mathbf{o}^{(pv)}) \rightarrow \mathbf{o}^{(s)} \mathbf{S}_0 y]$ states that the rest of the tape is blank.

if \mathbf{M} halts on n , *then* $\Delta \models H$

- Now suppose that \mathbf{M} eventually halts on input n .
Then for some s, i, p and j , it is the case that at time s the machine is in state q_i scanning square number p in which the symbol S_j appears,
and there is no quadruple in its program beginning $q_i S_j$.
- Suppose further that Δ implies a description D of time s .
Since \mathcal{J} is a model of Δ , D will be true in \mathcal{J} .
Two of the **conjuncts** of D will be $\mathbf{o}^{(s)}\mathbf{Q}_i\mathbf{o}^{(p)}$ and $\mathbf{o}^{(s)}\mathbf{S}_j\mathbf{o}^{(p)}$
and therefore D will imply $\exists t \exists x (t\mathbf{Q}_i x \wedge t\mathbf{S}_j x)$
which is one of the **disjuncts** of H .
- Therefore $\Delta \models H$

Proof by Mathematical Induction

- Hence we only need to show that for each s , if M has not halted before time s , then Δ implies *a description of time s*.
- Will prove this through Mathematical Induction.
- The basic idea of a proof by **mathematical induction** is to show that some statement or property holds for all possible cases (or values of n).

The proof consists of two steps:

The **basis step**: prove that the statement or property holds for the minimal case (or first ‘natural number’ n , so usually, $n = 0$ or $n = 1$).

Proof by Mathematical Induction

The **induction step**: prove that, *if* the statement holds for all cases up to (or at) an arbitrary given point (some natural number n), *then* the statement or property holds for all cases at the next higher point (or $n + 1$).

The hypothesis in the induction step that the statement holds for some arbitrary given point n is called the **induction hypothesis**.

To perform the **induction step**, assume the induction hypothesis is true, and then use this assumption to prove that the statement holds at the next level ($n + 1$).

Proof by Mathematical Induction

Basis step: $s = 0$.

The sentence

$$\mathbf{oQ}_1\mathbf{o} \wedge \mathbf{oS}_1\mathbf{o} \wedge \mathbf{oS}_1\mathbf{o}' \wedge \dots \wedge \mathbf{oS}_1\mathbf{o}^{(n-1)} \wedge \\ \forall y ((y \neq \mathbf{o} \wedge y \neq \mathbf{o}' \wedge \dots \wedge y \neq \mathbf{o}^{(n-1)}) \rightarrow \mathbf{oS}_0y))$$

is a description of time 0 and is contained in Δ .

Induction step: our induction hypothesis is the statement:

if \mathbf{M} has not halted before time s ,

then Δ implies *a description of time s* . Suppose this is true for s .

For **induction step**, assume the antecedent, that

\mathbf{M} has not halted before time $s + 1$.

Must then show the consequent, that

Δ implies a description of time $s + 1$.

Proof by Mathematical Induction

Since \mathcal{J} is a model of Δ , the *description of time s* is true in \mathcal{J} :

$$\mathbf{o}^{(s)} \mathbf{Q}_i \mathbf{o}^{(p)} \wedge \mathbf{o}^{(s)} \mathbf{S}_{j1} \mathbf{o}^{(p1)} \wedge \dots \wedge \mathbf{o}^{(s)} \mathbf{S}_j \mathbf{o}^{(p)} \wedge \dots \wedge \mathbf{o}^{(s)} \mathbf{S}_{jv} \mathbf{o}^{(pv)} \wedge \\ \forall y ((y \neq \mathbf{o}^{(p1)} \wedge y \neq \mathbf{o}^{(p)} \wedge \dots \wedge y \neq \mathbf{o}^{(pv)}) \rightarrow \mathbf{o}^{(s)} \mathbf{S}_0 y))$$

- So at time s \mathbf{M} is in state q_i scanning square number p in which the symbol S_j appears.
- Since \mathbf{M} does not halt at time s , its program must contain a quadruple of *exactly* one of the following 3 forms:
 - (i) $q_i S_j S_k q_m$
 - (ii) $q_i S_j R q_m$
 - (iii) $q_i S_j L q_m$

Proof by Mathematical Induction

If (i) $q_i S_j S_k q_m$ then one of the sentences in Δ is

$$\forall t \forall x \forall y [(t Q_{\textcolor{red}{i}} x \wedge t S_{\textcolor{red}{j}} x) \rightarrow (t' Q_{\textcolor{brown}{m}} x \wedge t' S_{\textcolor{brown}{k}} x \wedge (y \neq x \rightarrow (t S_0 y \rightarrow t' S_0 y \wedge \dots \wedge t S_r y \rightarrow t' S_r y)))]$$

This, together with the *description of time s*

(and the 2 axioms for ', <) imply the sentence

$$\mathbf{o}^{(\textcolor{green}{s}+1)} Q_{\textcolor{brown}{m}} \mathbf{o}^{(\textcolor{violet}{p})} \wedge \mathbf{o}^{(\textcolor{green}{s}+1)} S_{j1} \mathbf{o}^{(p1)} \wedge \dots \wedge \mathbf{o}^{(\textcolor{green}{s}+1)} S_{\textcolor{brown}{k}} \mathbf{o}^{(\textcolor{violet}{p})} \wedge \dots \wedge \mathbf{o}^{(\textcolor{green}{s}+1)} S_{jv} \mathbf{o}^{(pv)} \wedge \\ \forall y ((y \neq \mathbf{o}^{(p1)} \wedge y \neq \mathbf{o}^{(\textcolor{violet}{p})} \wedge \dots \wedge y \neq \mathbf{o}^{(pv)}) \rightarrow \mathbf{o}^{(\textcolor{green}{s}+1)} S_0 y))$$

which is a *description of time s + 1*

Proof by Mathematical Induction

If (ii) $q_i S_j R q_m$ then one of the sentences in Δ is

$$\forall t \forall x \forall y [(t Q_i x \wedge t S_j x) \rightarrow (t' Q_m x' \wedge (t S_0 y \rightarrow t' S_0 y \wedge \dots \wedge t S_r y \rightarrow t' S_r y))]$$

There is some symbol S_g such that the above together with the *description of time s* (and the 2 axioms for ', <)

imply the sentence

$$\begin{aligned} & \mathbf{o}^{(s+1)} Q_m \mathbf{o}^{(p+1)} \wedge \mathbf{o}^{(s+1)} S_{j1} \mathbf{o}^{(p1)} \wedge \dots \wedge \mathbf{o}^{(s+1)} S_j \mathbf{o}^{(p)} \wedge \mathbf{o}^{(s+1)} S_g \mathbf{o}^{(p+1)} \\ & \wedge \dots \wedge \mathbf{o}^{(s+1)} S_{jv} \mathbf{o}^{(pv)} \wedge \\ & \forall y ((y \neq \mathbf{o}^{(p1)} \wedge y \neq \mathbf{o}^{(p)} \wedge y \neq \mathbf{o}^{(p+1)} \dots \wedge y \neq \mathbf{o}^{(pv)}) \rightarrow \mathbf{o}^{(s+1)} S_0 y) \end{aligned}$$

which is a *description of time s + 1*

Proof by Mathematical Induction

If (iii) $q_i S_j L q_m$ then one of the sentences in Δ is

$$\forall t \forall x \forall y [(t Q_i x' \wedge t S_j x') \rightarrow (t' Q_m x \wedge (t' S_0 y \rightarrow t' S_0 y \wedge \dots \wedge t' S_r y \rightarrow t' S_r y))]$$

There is some symbol S_g such that the above together with the *description of time s* (and the axioms for ', <) imply the sentence

$$\begin{aligned} & \mathbf{o}^{(s+1)} Q_m \mathbf{o}^{(p-1)} \wedge \mathbf{o}^{(s+1)} S_{j1} \mathbf{o}^{(p1)} \wedge \dots \wedge \mathbf{o}^{(s+1)} S_g \mathbf{o}^{(p-1)} \wedge \mathbf{o}^{(s+1)} S_j \mathbf{o}^{(p)} \wedge \dots \wedge \\ & \mathbf{o}^{(s+1)} S_{jv} \mathbf{o}^{(pv)} \wedge \\ & \forall y ((y \neq \mathbf{o}^{(p1)} \wedge y \neq \mathbf{o}^{(p-1)} \wedge y \neq \mathbf{o}^{(p)} \dots \wedge y \neq \mathbf{o}^{(pv)}) \rightarrow \mathbf{o}^{(s+1)} S_0 y) \end{aligned}$$

which is a *description of time s + 1*.

In all 3 cases Δ implies a description of time $s + 1$. \square

Hence $\Delta \models H$ iff \mathbf{M} halts on input n .

FOL is Undecidable

And since we have just proved the non-trivial direction of the biconditional, *viz.* *if* \mathbf{M} halts on n , *then* $\Delta \models H$

contraposition yields

if $\Delta \not\models H$ *then* \mathbf{M}_m does **not** halt on input n ,
and $h(m,n) = 1$

So if FOL were decidable, then there would be an effective procedure for determining that $\Delta \not\models H$

So if FOL were decidable then the halting function would be computable (and the Church-Turing Thesis would be refuted).

Hence it follows by our original *reductio* strategy that

FOL is **undecidable**.

Logic, Computability and Incompleteness

Completeness, Compactness and
Löwenheim-Skolem

Validity in FOL

Validity: $\Phi_1, \dots, \Phi_n \models \Psi$

iff for all (FOL) interpretations \mathcal{J} , if \mathcal{J} satisfies each of Φ_1, \dots, Φ_n then \mathcal{J} satisfies (or is a model of) Ψ ,
iff $\models ((\Phi_1 \wedge \dots \wedge \Phi_n) \rightarrow \Psi)$.

This is an essentially **semantical** notion, and can be established by (informal) **semantical** proof.

The above rendition of validity is equivalent to the statement that it's impossible for all the premises to be **true** and the conclusion **false**.

And this impossibility can be proved using a *reductio* strategy

Example

- For example, here's a semantic proof that the argument
$$\exists x \forall y Rxy \text{ therefore } \forall x \exists y Ryx \quad \text{is valid.}$$
- Suppose the argument were not valid.

Then there must exist an interpretation \mathfrak{I} such that $\mathfrak{I}(\exists x \forall y Rxy) = \text{true}$, and $\mathfrak{I}(\forall x \exists y Ryx) = \text{false}$.

According to the definition of truth for the quantifiers, $\mathfrak{I}(R)$ must then be such that, given the domain D of \mathfrak{I} , there is an element $e \in D$ such that for all elements $e' \in D$, the pair $\langle e, e' \rangle \in \mathfrak{I}(R)$. But then, for all elements e' of D , there is an element of D , namely this same e , such that $\langle e, e' \rangle \in \mathfrak{I}(R)$.

Example

Thus according to the definition of truth for the quantifiers, it must be the case that $\mathfrak{I}(\forall x \exists y Ryx) = \text{true}$, **contrary** to hypothesis.

Hence it is impossible for there to be such a counter-model \mathfrak{I} , and so the argument is **valid** ■

- This is a perfectly rigorous and legitimate proof, but this method becomes progressively more unwieldy as arguments become more complex.
- Hence the need for a **mechanical**, **syntactical** method of proof that captures the underlying **semantical** facts.

Formal, Syntactical Proof

- There are a number of alternative Formal Proof techniques, and all of them are co-extensive in terms of capturing **exactly the same** set of underlying semantical facts.
- But whatever particular proof method is chosen for FOL, we want it to have the following 2 essential characteristics...

Soundness and Completeness

Soundness and **Completeness** are two basic metalogical properties of logical systems, intimately relating the semantical notion of **validity** and the syntactical notion of **provability**.

Validity: $\Phi_1, \dots, \Phi_n \models \Psi$ iff for all interpretations \mathcal{I} , if \mathcal{I} satisfies each of Φ_1, \dots, Φ_n then \mathcal{I} satisfies Ψ ,
iff $\models ((\Phi_1 \wedge \dots \wedge \Phi_n) \rightarrow \Psi)$

Provability: $\Phi_1, \dots, \Phi_n \vdash \Psi$ iff there is a proof or formal, rule governed derivation of Ψ from sentences Φ_1, \dots, Φ_n
iff $\vdash ((\Phi_1 \wedge \dots \wedge \Phi_n) \rightarrow \Psi)$

Soundness and Completeness

Soundness is a correctness property of formal proof systems and establishes that **only** valid arguments (or sentences) are provable:

$$\text{if } \Phi_1, \dots, \Phi_n \vdash \Psi \text{ then } \Phi_1, \dots, \Phi_n \vDash \Psi$$

Completeness is an adequacy property of formal proof systems and establishes that **all** valid arguments (or sentences) are provable:

$$\text{if } \Phi_1, \dots, \Phi_n \vDash \Psi \text{ then } \Phi_1, \dots, \Phi_n \vdash \Psi$$

If a system of Logic is both sound and complete, then the model-theoretic and proof-theoretic notions coalesce, and

$$\Phi_1, \dots, \Phi_n \vdash \Psi \text{ if and only if } \Phi_1, \dots, \Phi_n \vDash \Psi$$

In which case we can treat them as more or less interchangeable.

A Formal Proof System

- We will now examine a method of formal proof which will constitute a mechanical positive test for FOL validity, where this method is provably both **sound** and **complete**.
- The mechanical test for **validity** is designed as a positive test for **unsatisfiability** of sets of sentences Δ .
- The test will take the form of a systematic search for a **refutation** of Δ , such that *if* there is a **refutation** of Δ , then Δ is **unstatisfiable**.

This will correspond to the soundness of the method.

- Conversely, *if* Δ is **unstatisfiable** then there is a **refutation** of Δ .
This will correspond to the completeness of the method.

A Formal Proof System

- Let Δ be a set of sentences in **prenex normal form**, from which all vacuous quantifiers have been removed.
- A **refutation** of Δ is a derivation \mathcal{D} from Δ in which some **finite set** of **quantifier-free sentences** in the derivation is **unsatisfiable**.
- In turn, a derivation \mathcal{D} from Δ is a list of sentences (finite or denumerable), in which every entry is either a member of Δ or is obtainable from a previous entry in the list by one of the following two inference rules:

A Formal Proof System

UI

⋮

(m) $\forall v\Phi$

⋮

(n) $\Phi v/t$ (m) annotation

where t may be any (closed) term

EI

⋮

(m) $\exists v\Phi$

⋮

(n) $\Phi v/t$ (m) annotation

where t is a name which doesn't occur in Δ
or in any other line earlier than n .

Example

- Claim: $\forall x L^2(x, f^1(x)) \vdash \forall x \exists y L^2(x, y)$
Counterexample set = { $\forall x L^2(x, f^1(x))$, $\neg \forall x \exists y L^2(x, y)$ }
 $\Delta = \{ \forall x L^2x, f^1(x), \exists x \forall y \neg L^2(x, y) \}$

derivation \mathcal{D} from Δ :

1. $\exists x \forall y \neg L^2x, y$ Δ
2. $\forall y \neg L^2a, y$ 1. EI a
3. $\forall x L^2(x, f^1(x))$ Δ
4. $L^2(a, f^1(a))$ 3. UI a
5. $\neg L^2(a, f^1(a))$ 2. UI $f^1(a)$

4. and 5. constitute a **finite set** of **quantifier-free sentences** that is **unsatisfiable**. Hence \mathcal{D} is a refutation of Δ .

A Formal Proof System

The basic idea is that if \mathcal{D} is a refutation of Δ ,
then Δ has no model.

And if Δ is the counterexample set for some argument

$\Phi_1, \dots, \Phi_n \text{ therefore } \Psi$,

then the argument is established as **valid**.

In which case \mathcal{D} is a formal proof of **validity**, and hence

$\Phi_1, \dots, \Phi_n \vdash \Psi$

- We will first establish the **correctness** of the formal method
and then its **adequacy**....

Soundness

- **Soundness Theorem:** if there is a refutation of Δ , then Δ is **unsatisfiable** (where Δ is a set of sentences in prenex normal form, from which all vacuous quantifiers have been removed).
- **Strong Soundness Theorem:** if \mathcal{J} is a model of Δ and \mathcal{D} is a **derivation** from Δ , then the set of all sentences in \mathcal{D} has a model \mathcal{L} , where \mathcal{L} *differs* from \mathcal{J} (at most) in what it assigns to names and function symbols which occur in sentences of \mathcal{D} but not in Δ .
- **Strong Soundness Theorem** implies the (normal) **Soundness Theorem**, since if Δ **were satisfiable** it would have a model \mathcal{J} , and therefore so would all sentences in \mathcal{D} , in which case there could be **no refutation**.

Proof of Strong Soundness Theorem

- **Proof of Strong Soundness Theorem:** by inductive construction of a model \mathcal{L} of \mathcal{D} on the basis of \mathcal{J} .

Let $\Delta_0 = \Delta$ $\mathcal{J}_0 = \mathcal{J}$

$\Delta_n = \Delta \cup \{S_1, \dots, S_n\}$, where S_1, \dots, S_n are the first n sentences in \mathcal{D} .

Define a model \mathcal{J}_{n+1} of Δ_{n+1} , where induction step is based on the annotation A_{n+1} used in the derivation to get S_{n+1} .

Four cases:

i) A_{n+1} is ‘ Δ ’, in which case $\mathcal{J}_{n+1} = \mathcal{J}_n$.

ii) A_{n+1} is ‘UI’ and the instantial term t_{n+1} contains only names and function terms already occurring in Δ_n . Then $\mathcal{J}_{n+1} = \mathcal{J}_n$.

Proof of Strong Soundness Theorem

iii) A_{n+1} is ‘UI’ and the instantial term t_{n+1} contains names or function terms *not* occurring in Δ_n .

Take some element d in D (the domain of \mathcal{J}), and in every case let \mathcal{J}_{n+1} assign d as the denotation of new names, and all new functions are interpreted as constant functions with d as value.

\mathcal{J}_{n+1} is a model of Δ_n by continuity, and a model of Δ_{n+1} since UI is truth preserving.

iv) A_{n+1} is ‘EI’, in which case the instantial term t_{n+1} is new. Since the premise of this rule is in Δ_n , the premise must be true in \mathcal{J}_n . So there must be at least one element $e \in D$ such that \mathcal{J}_e^{tn+1} is a model of S_{n+1} and also of Δ_n .

So let $\mathcal{J}_{n+1} = \mathcal{J}_e^{tn+1}$ for some such e .

Proof of Strong Soundness Theorem

- Now define the model \mathcal{L} to be just like \mathcal{J} , except that for each function symbol or name appearing in \mathcal{D} but not in Δ ,
 \mathcal{L} assigns whatever \mathcal{J}_n assigns it,
where S_n is the first entry in \mathcal{D} in which the new term occurs.
- Hence if Δ had a model \mathcal{J} , then all the sentences in \mathcal{D} would have a model \mathcal{L} , in which case \mathcal{D} could not contain a refutation
(Strong Soundness Theorem) ■
- Therefore if some derivation \mathcal{D} from Δ is a refutation, then Δ has no model \mathcal{J} and is unsatisfiable (Soundness Theorem) ■

Completeness Theorem

- **Completeness** of FOL will be the culmination of our **positive** metatheoretical results. First proved by Kurt Gödel in 1930.
- **Completeness Theorem:** if a set of sentences Δ is **unsatisfiable**, then it has a **refutation**.

Completeness Proof: Canonical Derivation

- Proof: first need to define a canonical derivation from Δ , such that, if Δ is unsatisfiable, then any canonical derivation from Δ will be a refutation.
- Definition: \mathcal{D} is a canonical derivation from Δ iff it satisfies the following 5 conditions:
 - i) every sentence $\Phi \in \Delta$ occurs in \mathcal{D} .
 - ii) if $\exists v\Phi \in \mathcal{D}$, then for some term t , $\Phi v/t \in \mathcal{D}$.
 - iii) if $\forall v\Phi \in \mathcal{D}$, then for some term t , $\Phi v/t \in \mathcal{D}$.
 - iv) if $\forall v\Phi \in \mathcal{D}$, then for every term t that can be constructed from names and function symbols occurring in \mathcal{D} , $\Phi v/t \in \mathcal{D}$.
 - v) all function symbols occurring in \mathcal{D} appear in Δ .

Completeness Proof: Canonical Derivation

Program for constructing a **canonical derivation** from Δ :

Let S_1, S_2, S_3, \dots be an enumeration of the sentences in Δ .

Stage 1(a): enter S_1 as the first line in \mathcal{D} .

1(b): add as many entries to \mathcal{D} as possible using **EI** with **restrictions** (to be stated momentarily)

1(c): add as many entries to \mathcal{D} as possible using **UI**

- restrictions: no sentence is the premise of more than one application of **EI**, no sentence occurs twice, and each instantial term has fewer than N (= **current stage number**) occurrences of function symbols, and is formed from names and function symbols already in \mathcal{D} .

Stage 2(a): enter S_2 as the n^{th} line in \mathcal{D} and repeat...

Example

$$\Delta = \{\mathbf{S}_1, \mathbf{S}_2\} = \{\forall x L^2x, f^1(x), \exists x \forall y \neg L^2x, y\}$$

1.	$\forall x L^2x, f^1(x)$	Δ	(1a)	
2.	$L^2a, f^1(a)$	1. UI	(1c)	a
3.	$\exists x \forall y \neg L^2x, y$	Δ	(2a)	
4.	$\forall y \neg L^2b, y$	3. EI	(2b)	b
5.	$L^2b, f^1(b)$	1. UI	(2c)	b
6.	$L^2f^1(a), f^1(f^1(a))$	1. UI	(2c)	$f^1(a)$
7.	$L^2f^1(b), f^1(f^1(b))$	1. UI	(2c)	$f^1(b)$
8.	$\neg L^2b, a$	4. UI	(2c)	a
9.	$\neg L^2b, b$	4. UI	(2c)	b
10.	$\neg L^2b, f^1(a)$	4. UI	(2c)	$f^1(a)$
11.	$\neg L^2b, f^1(b)$	4. UI	(2c)	$f^1(b)$

11. and 5. **unsatisfiable**. Hence \mathcal{D} is a refutation of Δ

Completeness Proof: Lemma II and Matching

- Definition of Matching: suppose Γ is a set of quantifier-free sentences.
An interpretation \mathcal{I} matches Γ iff \mathcal{I} is a model of Γ (written $\mathcal{I} \models \Gamma$), and there are no elements in the domain of \mathcal{I} not named by terms in Γ .
- **Lemma II.** Suppose \mathcal{D} is a canonical derivation from Δ , Γ is the set of all quantifier-free sentences in \mathcal{D} , and \mathcal{I} matches Γ . Then \mathcal{I} is a model of \mathcal{D} and hence of Δ .
- **Proof** by reductio (see B&J p. 134-5).

Completeness Proof: an **OK** set of sentences

Now all that remains to be proved is that if every finite subset of Γ is satisfiable, then some interpretation \mathcal{I} matches Γ .

- This will show that if Δ is unsatisfiable, then a canonical derivation \mathcal{D} is a refutation, since it must possess a finite subset of quantifier-free sentences which is unsatisfiable.
- So this is really the contraposition of completeness – if \mathcal{D} is not a refutation then Δ is satisfiable.
- Will prove this by constructing an \mathcal{I} which must be a model of Δ if every finite subset of Γ is satisfiable.
- To do this, first need to introduce the concept of an **OK** set of sentences:
 - a set of sentences Σ is **OK** iff
 - every finite subset of Σ is satisfiable.

Completeness Proof: Lemma III

- So to prove **completeness**, only need to prove
- **Lemma III**: if Γ is an enumerable, **OK** set of quantifier-free sentences, then there is an \mathcal{I} which **matches** Γ .
- **Proof**: must define such an \mathcal{I} using the given information:
General procedure:

First enumerate all atomic sentences A_1, A_2, \dots which are

- (i) **sentence letters** (propositional) occurring in Γ , **or**
- (ii) formed by filling in the argument places of the ‘=’ sign, using **terms** appearing in Γ , **or**
- (iii) formed by filling in the argument places of **predicate letters** occurring in Γ , using **terms** appearing in Γ .

Completeness Proof: Construct an \mathcal{J} which matches Γ

Now define the sequence $\Gamma_1, \Gamma_2, \dots$ and verify that all members in the sequence are **OK**:

Let $\Gamma_1 = \Gamma$ **OK** by hypothesis.

Now suppose Γ_n has been defined and is **OK**.

Then at least one of the sets $\Gamma_n \cup \{\text{A}_{\textcolor{violet}{n}}\}$ or $\Gamma_n \cup \{\neg\text{A}_{\textcolor{violet}{n}}\}$ is **OK**.

Define Γ_{n+1} as the **OK** one if just one is,
and $\Gamma_n \cup \{\text{A}_{\textcolor{violet}{n}}\}$ if both are **OK**.

Let \mathbf{B}_i be whichever of $\text{A}_{\textcolor{violet}{i}}$, $\neg\text{A}_{\textcolor{violet}{i}}$ is in the expansion Γ_{i+1} .

Completeness Proof: Construct an \mathcal{J} which matches Γ

Now, if r, s are terms in Γ , exactly one of $r = s$, $\neg(r = s)$ is in the sequence of \mathbf{B} 's.

Definition: $r \sim s$ iff $r = s$ is one of the \mathbf{B} 's.

\sim is an equivalence relation on the set of terms in Γ .

Now to define \mathcal{J} which matches Γ : want \mathcal{J} to assign each term t its own equivalence class $[t]$ as denotation (!)

and want $\mathcal{J}(\mathbf{B}_i) = 1$ for each i . So...

A) let the domain D of \mathcal{J} be the set of all equivalence classes of terms in Γ .

B) let $\mathcal{J}(t) = [t]$ for each individual constant t

Completeness Proof: Construct an \mathcal{J} which matches Γ

C) for each n -place **function symbol** f^n , let $\mathcal{J}(f^n)$ be the **function** g^n such that for all $[t_1], \dots, [t_n]$, in D ,

$g^n([t_1], \dots, [t_n]) = [f^n(s_1, \dots, s_n)]$ if there are terms s_1, \dots, s_n in $[t_1], \dots, [t_n]$ such that $f^n(s_1, \dots, s_n)$ is a term appearing in Γ .

Otherwise $g^n([t_1], \dots, [t_n]) = [t]$, for any term t in Γ .

D) a **sentence letter** is true in \mathcal{J} iff it is one of the **B's**.

E) for each n -place **predicate letter** P^n occurring in Γ

$<[t_1], \dots, [t_n]> \in \mathcal{J}(P^n)$ iff $P^n(t_1, \dots, t_n)$ is one of the **B's**.

The definition of \mathcal{J} is now finished. It follows by induction (using **B**, **C**) that each **complex** term t occurring in Γ denotes its own equivalence class $[t]$. (B&J p.139).

Completeness Proof: Construct an \mathcal{J} which matches Γ

Also, it follows that each of the \mathbf{B} 's is true in \mathcal{J} , since by cases ((i)-(iii) above), \mathbf{A}_i is true in \mathcal{J} iff $\mathbf{A}_i = \mathbf{B}_i$ (B&J p.140)

To see that \mathcal{J} matches Γ , first, it's clear that every object in D is named by a term of Γ .

So just need to show that \mathcal{J} is a model of Γ :

Suppose sentence $\mathbf{S} \in \Gamma$.

\mathbf{S} is a truth-functional combination of some finite set

$\{\mathbf{A}_1, \dots, \mathbf{A}_k\}$ of the \mathbf{A} 's.

In any interpretation \mathcal{J} in which all of $\mathbf{B}_1, \dots, \mathbf{B}_k$ are true, each $\mathbf{A}_1, \dots, \mathbf{A}_k$ has the same truth value as in \mathcal{J} , and hence \mathbf{S} has the same value as in \mathcal{J} .

Finally, need to show that this value = 1

Completeness Proof: Construct an \mathcal{J} which matches Γ

All of $\mathbf{B}_1, \dots, \mathbf{B}_k$ are in Γ_{k+1} , as is \mathbf{S} , which is in Γ_1 .

Thus $\{\mathbf{B}_1, \dots, \mathbf{B}_k, \mathbf{S}\} \subseteq \Gamma_{k+1}$.

Since Γ_{k+1} is **OK**, this finite subset must be **satisfiable**, and hence all its members **true** in **some** interpretation \mathcal{J} .

And since all of $\mathbf{B}_1, \dots, \mathbf{B}_k$ and \mathbf{S} are true in \mathcal{J} , **S is true in \mathcal{J}** .

- Thus \mathcal{J} matches Γ , and by **Lemma II** $\mathcal{J} \vDash \Delta$. So,
- **Completeness Theorem:** if Γ is the set of quantifier-free sentences in a canonical derivation \mathcal{D} from Δ , then if \mathcal{D} is **not** a refutation (i.e. Γ is **OK**), then \mathcal{J} is a model of Δ ■

Completeness of the Formalism

- We have now demonstrated that our method of formal, syntactic proof is complete.
- This shows that if some formula Ψ follows as a **logical consequence** of a set of formulas Γ , then our proof method is strong enough to yield a **formal demonstration** of this fact.
- In particular, if Γ is a set of **axioms** for some formal theory T , then our deductive apparatus is strong enough to yield a **proof** of every sentence in the language which follows as a **logical consequence** of these **axioms**.
- Some immediate consequences of the **Soundness** and **Completeness** proofs:

Compactness Theorem

- **Compactness Theorem:** A set of sentences Σ is unsatisfiable iff some finite subset $\Sigma_0 \subseteq \Sigma$ is unsatisfiable.
- **Proof:** if Σ is unsatisfiable, then by Completeness a canonical derivation \mathcal{D} from Σ is a refutation.
- Let $\{A_1, \dots, A_m\}$ be the finite set of quantifier-free sentences in \mathcal{D} that is unsatisfiable.
- Let j be the number of the line in \mathcal{D} at which A_m occurs, and truncate \mathcal{D} at line j to obtain \mathcal{D}_0 , which is finite.
Let $\{S_1, \dots, S_n\}$ be the members of Σ occurring in \mathcal{D}_0 , and let $\Sigma_0 = \{S_1, \dots, S_n\}$.
 Σ_0 is unsatisfiable (by Soundness) ■

Compactness Theorem

- Direct result: finite entailment:
if $\Delta \vDash \Psi$, then a finite subset $\Delta_0 \subseteq \Delta$ is such that
$$\Delta_0 \vDash \Psi$$

Löwenheim-Skolem Theorem

- **Löwenheim-Skolem Theorem:** If a set of sentences Δ has a model, then it has a model with an **enumerable domain**.
- **Proof:** if Δ is **satisfiable**, then the **canonical derivation** \mathcal{D} from Δ is not a **refutation** (by **Soundness**). Hence every finite subset of Γ (the set of quantifier-free sentences in \mathcal{D}) is **satisfiable** (by **Compactness**).

By **Lemma III** there is a model \mathcal{J} that **matches** Γ , and by **Lemma II** \mathcal{J} is a model of Δ .

Since \mathcal{J} **matches** Γ , every object in the domain of \mathcal{J} is named by some term in Γ . Since Γ is **enumerable**, there are only **enumerably** many such terms.

Therefore \mathcal{J} has an **enumerable** domain ■

Cardinality

- The Löwenheim-Skolem Theorem reveals a fundamental fact about the expressive power of sentences in FOL with respect to the cardinality of their models.
- This version is the ‘downward’ Löwenheim-Skolem Theorem and shows that you can’t force there to be only models with a domain of cardinality greater than \aleph_0 .
- Hence any (consistent) set of sentences (e.g. a formal theory of numbers) will be satisfied by an interpretation with a countable domain.

Cardinality

- We've seen the ‘downward’ Löwenheim-Skolem Theorem, showing that you can't force there to be only models with a domain of cardinality greater than \aleph_0 .
- There is also an ‘upward’ version of the theorem: if a set Δ has an infinite model, then it has a model with uncountably many elements.
- This shows we also can't force only less than uncountable models.
- In other words, FOL cannot distinguish between different levels of infinity.

What is First-Order Logic?

- It turns out that compactness plus upward and downward Löwenheim-Skolem (L-S) metalogically capture FOL...
- **Lindströms Theorem** (1969):

Let \mathcal{L} be any ‘extension’ of FOL with the two properties:

i) downward L-S

and

ii) either upward L-S or compactness.

Then \mathcal{L} is no ‘stronger’ than FOL, in the sense that every sentence of \mathcal{L} has exactly the same models as some sentence of FOL.

Logic, Computability and Incompleteness

Formal Arithmetic and the Diagonal
Lemma

Hilbert's Program

- Beginning at the turn of the 20th century, Hilbert proposed a strategy for the foundation of classical mathematics that eventually developed into the so-called ‘Formalist Program’.
- This program was in response to the foundational crisis prompted by the newly discovered inconsistency of ‘naïve’ set theory, in the form of **Russell’s paradox**, which also infected Frege’s ‘Logicist’ foundational system.
- Russell’s paradox is famously formulated in terms of **the set of all sets that are not members of themselves**.
- It leads directly to a contradiction in naïve set theory, because this theory assumes the unrestricted Comprehension Axiom:

Hilbert's Program

- **Comprehension Axiom:** for any formula $\varphi(x)$ containing x as a free variable, there exists the **set** $\{x: \varphi(x)\}$ whose **members** are exactly those objects that satisfy $\varphi(x)$.
- Thus, if the **formula** $\varphi(x)$ stands for “ x is prime”, then $\{x: \varphi(x)\}$ will be the **set** of prime numbers.
- If $\varphi(x)$ stands for “ $\neg(x = x)$ ”, then $\{x: \varphi(x)\}$ will be the null **set**.
- But if we let $\varphi(x)$ stand for $x \in x$ and let $S = \{x: \neg\varphi(x)\}$, then **S** is the **set** whose members are exactly those objects that are not members of themselves.
- Is **S** a member of itself?
- Can easily deduce $(S \in S) \leftrightarrow \neg(S \in S)$

Hilbert's Program

- Russell's paradox corresponds to the fact that the FOL formula $\exists x \forall y (Rxy \leftrightarrow \neg Ryy)$ is unsatisfiable – there can be no such x . If we let Rxy mean ' y is an element of x ', then in standard set theoretical notation this is the same as $\exists x \forall y (y \in x \leftrightarrow y \notin y)$
If we assume the (intuitively plausible) Comprehension Axiom then we can prove that there is such an x , and hence our theory will be able to prove a contradiction...
Hilbert sought to avoid such disasters by advocating an idealized foundational program in which all of mathematics is deducible in an axiomatizable formal theory where the axioms themselves are (independent and) provably consistent.

Representability in a Theory

- As we saw when revisiting FOL, a **Formal Theory T** is a set of sentences (in some formal language L) which is **closed under the relation of logical consequence**.

So for all sentences Φ of L , if $T \vdash \Phi$ then $\Phi \in T$ in which case Φ is a **theorem** of T , written $\vdash_T \Phi$

Representability in a Theory:

an n -place function of natural numbers f^n is representable in a theory T **iff** there is a formula $A(x_1, \dots, x_n, x_{n+1})$ in the language of T such that for any natural numbers p_1, \dots, p_n, j

if $f^n(p_1, \dots, p_n) = j$ **then** $\vdash_T \forall x(A(p_1, \dots, p_n, x) \leftrightarrow x = j)$

where p is the **numeral** for p , i.e. **0** followed by p applications of the successor function '

Robinson Arithmetic

- In this case $A(x_1, \dots, x_n, x_{n+1})$ represents f^n in \mathbf{T} .
Thus if $A(x_1, \dots, x_n, x_{n+1})$ represents f^n in \mathbf{T}
and $f^n(p_1, \dots, p_n) = j$ then both
 $\vdash_{\mathbf{T}} A(\mathbf{p}_1, \dots, \mathbf{p}_n, \mathbf{j})$ and $\vdash_{\mathbf{T}} \forall x(A(\mathbf{p}_1, \dots, \mathbf{p}_n, x) \rightarrow x = \mathbf{j})$
- The formal theory of particular interest to us will be Q ,
the theory of **Robinson Arithmetic**
- The language L of Q is **FOL**
with $\mathbf{o}, ', +, \cdot$ as distinguished vocabulary items.
- Q is the set of all sentences in L which are **logically entailed**
by the following 7 **axioms**:

Robinson Arithmetic

Q1: $\forall x \forall y (x' = y' \rightarrow x = y)$

Q2: $\forall x \mathbf{0} \neq x'$

Q3: $\forall x (x \neq \mathbf{0} \rightarrow \exists y x = y')$

Q4: $\forall x (x + \mathbf{0} = x)$

Q5: $\forall x \forall y (x + y') = (x + y)'$

Q6: $\forall x (x \cdot \mathbf{0} = \mathbf{0})$

Q7: $\forall x \forall y (x \cdot y') = (x \cdot y) + x$

Each axiom is a **single sentence**, so Q is **finitely axiomatizable**

Representability in Robinson Arithmetic

- Robinson Arithmetic Q differs from the stronger theory of Peano Arithmetic PA , in that it *lacks* the **schema** of Mathematical Induction:
$$[\Phi(\mathbf{0}) \wedge (\forall x (\Phi(x) \rightarrow \Phi(x'))] \rightarrow \forall x \Phi(x)$$
where $\Phi(v)$ is any formula in the language L with the variable v free.
- The **schema** of Mathematical Induction introduces infinitely many axioms as instances of the schema.
- Very important property of Q :
All recursive functions are representable in Q

Representability in Robinson Arithmetic

- Thus for **every** function f^n of natural numbers obtainable from the set of **Base functions**:
 - 1) **zero** function
 - 2) **successor** function
 - 3) **projection** functions

through finite applications of **Composition**, **Primitive recursion** and **Minimization**,

there is a formula $A(x_1, \dots, x_n, x_{n+1})$ in the language **L**
such that **if** $f^n(p_1, \dots, p_n) = j$ **then both**

$$\vdash_Q A(\mathbf{p}_1, \dots, \mathbf{p}_n, \mathbf{j}) \text{ and } \vdash_Q \forall x(A(\mathbf{p}_1, \dots, \mathbf{p}_n, x) \rightarrow x = \mathbf{j})$$

Arithmetization of Syntax

- We'll now look at Gödel numbering, which is the first ingredient needed to achieve formal ‘self-reference’ in arithmetic.
- Gödel numbering is a scheme for assigning natural numbers to *expressions* in a **formal object language**.
- Necessary characteristics of a Gödel numbering scheme:
 - 1) different *expressions* get different *numbers*
 - 2) given any *expression* can effectively calculate its Gödel number
 - 3) given any *number* can effectively determine
 - (i) whether it's the Gödel number of an *expression*, and

Arithmetization of Syntax

(ii) if so, can effectively recover the expression from the number.

- Particular Gödel numbering scheme used in B&J:
- Numbers 1-7 are used to distinguish basic categories of symbols:
1, 2 for punctuation symbols, 3 for truth-functional connectives, 4 for quantifiers, 5 for variables, 6 for function symbols and 7 for predicate symbols.
- Numbers 8, 9 used to make internal distinctions
(e.g. for non-zero superscripts and subscripts).

Scheme is given in charts on p. 171:

Arithmetization of Syntax

		()	&	\exists	x_0	f_0^0	f_0^1	f_0^2	...	A_0^0	A_0^1	A_0^2	...
		,	v	\forall	x_1	f_1^0	f_1^1	f_1^2	...	A_1^0	A_1^1	A_1^2	...
		-			x_2	f_2^0	f_2^1	f_2^2	...	A_2^0	A_2^1	A_2^2	...
		\leftrightarrow		
		\rightarrow		
1	2	3	4					
29	39	399	3999	39999	5	6	68	688	...	7	78	788	...
					59	69	689	6889	...	79	789	7889	...
					599	699	6899	68899	...	799	7899	78899	...
				
				

Arithmetization of Syntax

- Each basic vocabulary **symbol** is thereby given a **unique number**. To do this, the actual coding must deal directly with official **object language symbols** systematically arranged.
- However, in practice it's *inconvenient* to work purely with object language symbols, so we will adopt some conventions:

<u>official symbol</u>	<u>'informal' name</u>
------------------------	------------------------

x_0, x_1, x_2, \dots	x, y, z, \dots
------------------------	------------------

f_0^0	\mathbf{o}
---------	--------------

f_0^1	$'$
---------	-----

f_0^2	$+$
---------	-----

f_1^2	\cdot
---------	---------

A_0^2	$=$
---------	-----

Arithmetization of Syntax

- Concatenation of basic vocabulary symbols to form complex expressions is reflected by concatenation of the numbers of the symbols involved and then read in decimal notation.

The Gödel number of symbol ‘A’ is written $gn[A]$.

So if $gn[A] = i$ and $gn[B] = j$

then $gn[AB] =$ the number denoted by ‘ ij ’ in decimal notation.

example:

Arithmetization of Syntax

Given the foregoing naming conventions:

$$\mathbf{gn}[\forall] = 49, \quad \mathbf{gn}[x] = 5, \quad \mathbf{gn}[y] = 59$$

$$\mathbf{gn}[\mathbf{o}] = 6, \quad \mathbf{gn}['] = 68, \quad \mathbf{gn}[+] = 688, \quad \mathbf{gn}[\cdot] = 6889$$

$$\mathbf{gn}[=] = 788$$

And given the method for determining the **Gödel number** of a concatenation of symbols

$\mathbf{gn}[\forall x(x = x)]$ = the concatenation of the **numbers** of the
seven constituent symbols

$$\forall \ x \ (\ x \quad = \quad x \)$$

$$/ \ | \ | \ | \ | \ | \ \backslash$$

$$49 \ 5 \ 1 \ 5 \ 788 \ 5 \ 2 \quad = \quad 4951578852$$

Arithmetization of Syntax

- In this manner, the language L of the theory Q that is *intended* to be about the natural numbers can instead be interpreted as being about **its own syntax** (!)
So there will be (unintended) interpretations \mathcal{J} in which L can be seen as ‘*making assertions about itself*’
- Furthermore the sentences of L which are **theorems** of Q must be true **in every** model of Q .
- In particular, **all recursive operations** on **expressions** and **sequences of expressions** can be **represented** in Q ,
which means that the corresponding sentences are **provable** in the system.

Arithmetization of Syntax

- So *via* its theorems, the theory Q can be interpreted as proving things about itself, by associating expression in L with **Gödel numbers** and then proving assertions about these numbers.
And these sentences must be **true** in *every model* of Q .
- This possibility is realized by Gödel's ingenious version of **diagonalization**, which is the technical heart of the **limitative metatheoretical** results to follow....

Diagonalization

- Convention: if $gn[A] = n$, let $\ulcorner A \urcorner = n$
i.e. $0''\dots'$ with n applications of the successor function.
- So n is the **numeral** for the **Gödel number** of A .
Hence $\ulcorner A \urcorner$ is the **Gödel numeral** of A ,
in which case n can be construed as a **name** in the object language L denoting the object language expression ‘ A ’.
- This is the second step in achieving formal ‘self-reference’.
- Now let the **diagonalization** of A be defined as the sentence

$$\exists x (x = \ulcorner A \urcorner \wedge A)$$

Diagonalization

- If A has just the variable x free, written $A(x)$, then the diagonalization of A is logically equivalent to $A(\ulcorner A \urcorner)$

$$\exists x (x = \ulcorner A \urcorner \wedge A) \equiv A(\ulcorner A \urcorner)$$

- **Lemma:** there is a recursive function **diag** such that **diag**(n) = the Gödel number of the diagonalization of the expression with Gödel number n .
- **Proof:** by construction.
 - 1) Let $\text{lh}(n) = \mu m (0 < m \wedge n < 10^m)$ [read ‘the least m such that (...)’]
So $\text{lh}(n) =$ the **number of digits** in the decimal notation for the number n .

Diagonalization

2) Let $m * n = m \cdot 10^{\text{lh}(n)} + n$

$m * n$ is the number denoted by the arabic numeral formed by concatenating the arabic numeral for m with the numeral for n

3) Define the function **num**(x) such that

$$\mathbf{num}(0) = 6 \quad \mathbf{num}(n+1) = \mathbf{num}(n) * 68$$

So **num**(n) = the Gödel number of the numeral **n** (!)

4) The diagonalization of formula A was defined as

$$\exists x (x = \ulcorner A \urcorner \wedge A).$$

And if **gn**[A] = n , then $\ulcorner A \urcorner = \mathbf{n}$.

Hence the diagonalization of A is the formula $\exists x(x = \mathbf{n} \wedge A)$.

So let **diag**(n) = $4515788 * (\mathbf{num}(n) * (3 * (n * 2)))$

Diagonalization

$$\text{diag}(n) = \underbrace{4515788 * (\underbrace{\text{num}(n)} * (3 * (n * 2))))}_{\exists x (x = n \wedge A)}$$

The Diagonal Lemma

- Hence **diag**(n) is the Gödel number of the diagonalization of the expression with Gödel number n , and **diag** is recursive by construction \square
- Since all recursive functions are representable in Q , **diag** is representable in Q .
- **Diagonal Lemma:** Let \mathbf{T} be a theory in which **diag** is representable. Then for any formula $B(y)$ in the language of \mathbf{T} with just the variable y free, there is a sentence G such that
$$\vdash_{\mathbf{T}} G \leftrightarrow B(\ulcorner G \urcorner)$$
- **Proof:** exhibit a procedure for constructing such a G for any given $B(y)$.

The Diagonal Lemma

Let the formula $A_d(x, y)$ represent **diag** in **T**.

Then for any numbers n, k ,

if **diag**(n) = k then $\vdash_T \forall y(A_d(n, y) \leftrightarrow y = k)$

Let F be defined as the formula $\exists y(A_d(x, y) \wedge B(y))$.

F contains just the variable x free.

Now let G be defined as the diagonalization of F (!)

i.e. G is the sentence $\exists x(x = \ulcorner F \urcorner \wedge \exists y(A_d(x, y) \wedge B(y)))$.

Suppose $gn[F] = n$, so $\ulcorner F \urcorner = n$

As noted above, G is logically equivalent to the result of instantiating the variable x with $\ulcorner F \urcorner$ which is n :

The Diagonal Lemma

To repeat, G is the sentence $\exists x(x = \ulcorner F \urcorner \wedge \exists y(A_d(x, y) \wedge B(y)))$

and $G \equiv \exists y(A_d(n, y) \wedge B(y))$

so (i) $\vdash_T G \leftrightarrow \exists y(A_d(n, y) \wedge B(y))$

Next, suppose $\text{diag}(n) = k$.

Then, since $gn[F] = n$ and G is the diagonalization of F ,

$gn[G] = k$ and $\ulcorner G \urcorner = k$

Now, since $A_d(x, y)$ represents diag in T and $\text{diag}(n) = k$

we get (ii) $\vdash_T \forall y(A_d(n, y) \leftrightarrow y = k)$

Taking (i) and substituting provable equivalents from (ii)

we get (iii) $\vdash_T G \leftrightarrow \exists y(y = k \wedge B(y))$

The Diagonal Lemma

Taking (iii) $\vdash_T G \leftrightarrow \exists y (\textcolor{teal}{y} = \mathbf{k} \wedge B(y))$

and applying the same strategy used to get (i) yields

$$\vdash_T G \leftrightarrow B(\mathbf{k})$$

And since $\ulcorner G \urcorner = \mathbf{k}$

it's now immediate that

$$\vdash_T G \leftrightarrow B(\ulcorner G \urcorner) \blacksquare$$

Conceptual Overview

- It's fitting at this point to step back for a moment and reflect on the evolution of our theoretical perspective.
- Leibniz (1646-1716) speculated about the development of a precise artificial language, a '*calculus ratiocinator*', in which all of human thought could be reduced to calculation.
- Frege's *Begriffsschrift* (1879) is the first actual instance of an **artificial language** constructed according to exact rules of syntax,
and the *Begriffsschrift*'s system of first-order logic was powerful enough to formalize all the reasoning ordinarily used in mathematics.
- So (in principle at least) all of mathematics could be carried out **inside** this formal system.

Conceptual Overview

- Whitehead and Russell then succeeded in developing all of classical mathematics *within* the artificial logical system of *Principia Mathematica* (PM, 1910).
- Beginning around the turn of the century, Hilbert proposed a perspective in which we abstract away from proofs *inside* such formal systems
and instead look at them from the **outside**
from a **metamathematical** perspective in which we prove higher-level metalogical results *about* these object level systems.
Thus in 1930 Gödel proved the **completeness** of first-order logic,
and in 1936 Church established its **undecidability**.

Conceptual Overview

- In his 1931 work on the **incompleteness** of PM, Gödel took the level of abstraction a step further by embedding these metamathematical concepts **inside** the object level system itself to attain yet new results.
- By using a scheme for numerically coding the syntax of a formal theory of arithmetic, it was possible to interpret sentences **in** the formal object language as making assertions about various properties of sentences **in** the formal object language .
- In particular, as we'll soon see, a property of special interest is ‘sentence Φ is **provable in** formal arithmetic’.

Conceptual Overview

- Thus, given a numerical coding scheme, Gödel was able to construct a sentence Ψ in formal arithmetic, which can be interpreted as asserting that ‘sentence Φ is provable in formal arithmetic’.
- And using the diagonal lemma, Gödel was able to construct a case where
 Ψ is identical to Φ !
- Thus, *via* this employment of Cantor’s diagonal method, matters can be arranged such that the object language sentence asserted to be provable and the object language sentence making the assertion are one and the same...

Logic, Computability and Incompleteness

Undecidability, Indefinability and
Gödel's First Theorem

Definability and Decidability

- Some important technical concepts and terminology:

Definability: a set of **natural numbers** Θ is

definable in a theory \mathbf{T} iff

there is a *formula* $B(x)$ in the language of \mathbf{T} such that for any number k ,

if $k \in \Theta$ then $\vdash_{\mathbf{T}} B(k)$, and

if $k \notin \Theta$ then $\vdash_{\mathbf{T}} \neg B(k)$

in which case the formula $B(x)$ **defines** the set Θ in \mathbf{T} .

Decidability: a set of **expressions** is decidable if the set of

Gödel numbers of its members is a **recursive set**, i.e.

if the characteristic function of the set is **recursive**.

Definability and Decidability

Thus, if Θ^{Gn} is the set of Gödel numbers of **expressions** in Θ
and if $f_{\Theta^{\text{Gn}}}$ is the characteristic function of Θ^{Gn}
then $f_{\Theta^{\text{Gn}}}(n) = 1$ iff $n \in \Theta^{\text{Gn}}$ (and = 0 otherwise)
and if $f_{\Theta^{\text{Gn}}}$ is **recursive** then the set of expressions Θ is decidable
So a **theory T** is **decidable** iff

the set of Gödel numbers of its theorems is a **recursive set**.

- Connection between the two notions:
 - if a set of expressions Θ is **decidable** then the respective characteristic function is **recursive**
and hence is representable in Q

Definability and Decidability

Which in turn means that the set of Gödel numbers of expressions in Θ is **definable** in Q .

This is because if the characteristic function

$f_{\Theta^{\text{Gn}}}$ of Θ^{Gn} is **recursive**

and the formula $A_{f_{\Theta^{\text{Gn}}}}(x,y)$ represents $f_{\Theta^{\text{Gn}}}$ in Q ,

then $A_{f_{\Theta^{\text{Gn}}}}(x,1)$ **defines** Θ^{Gn} in Q (!)

So if a theory **T** is **decidable** then

the set of Gödel numbers of its theorems is **definable** in Q .

Definability and Decidability

- **Lemma:** if \mathbf{T} is a consistent extension of \mathbf{Q} , then the set of Gödel numbers of theorems of \mathbf{T} is **not definable** in \mathbf{T} .
- **proof:** by *reductio*, using basic template furnished by the diagonal lemma.

Let $C(y)$ define the set of Gödel numbers of theorems of \mathbf{T} .

The function **diag** is representable in \mathbf{T} and $\neg C(y)$ is a formula with only the variable y free.

So by the **diagonal lemma** there is a sentence G such that

$$(*) \quad \vdash_{\mathbf{T}} G \leftrightarrow \neg C(\ulcorner G \urcorner).$$

Suppose $gn[G] = k$, so $\ulcorner G \urcorner = k$. Then

$$(i) \quad \vdash_{\mathbf{T}} G \leftrightarrow \neg C(k).$$

Definability and Decidability

It follows by (sub) **reductio** that $\vdash_{\mathbf{T}} G$, for if **not** $\vdash_{\mathbf{T}} G$, then, since $C(y)$ defines the set of **theorems** of \mathbf{T} , we get $\vdash_{\mathbf{T}} \neg C(k)$ and hence $\vdash_{\mathbf{T}} G$ by (i) [going R to L]. So $\vdash_{\mathbf{T}} G$. Thus $k \in \Theta$ and $\vdash_{\mathbf{T}} C(k)$.

By (i) we get $\vdash_{\mathbf{T}} G \rightarrow \neg C(k)$ contraposition yields $\vdash_{\mathbf{T}} \neg \neg C(k) \rightarrow \neg G$, which yields $\vdash_{\mathbf{T}} C(k) \rightarrow \neg G$, and finally by modus ponens $\vdash_{\mathbf{T}} \neg G$. So both $\vdash_{\mathbf{T}} G$ and $\vdash_{\mathbf{T}} \neg G$, rendering \mathbf{T} inconsistent, contrary to initial hypothesis.

Conclusion: *there can be no such $C(y)$* ■

Undecidability of FOL (from a different angle)

- Bigger conclusion: no **consistent** extension of Q is **decidable**.
Why?
- Because if the theory \mathbf{T} were **decidable**, then its set of theorems would be **definable** in Q and hence in \mathbf{T}
- **Church's Theorem**: FOL is **undecidable**.
- **proof**: we have just established that Q is **undecidable**, since it is a consistent extension of itself.

Let Φ be the single sentence formed by conjoining all of the 7 **axioms** of Q .

Then a sentence S is a **theorem** of Q iff the conditional $\Phi \rightarrow S$ is a **theorem** of FOL.

Undecidability of FOL (from a different angle)

In other words

$$\vdash_Q S \text{ iff } \vdash_{\text{FOL}} (\Phi \rightarrow S)$$

Hence (intuitively) if FOL were decidable *then* so would Q be.

To carry out this *reductio proof* more formally,

let $gn[\Phi] = q$ and let the function f be defined such that

$$f(n) = 1 * (q * (39999 * (n * 2)))$$

f is recursive (by construction)

and if n is the Gödel number of the sentence S ,

then $f(n)$ is the Gödel number of the sentence $(\Phi \rightarrow S)$

Undecidability of FOL (from a different angle)

- Let Θ be the set of Gödel numbers of **theorems** of **FOL**.
If Θ is recursive then so is $\{n: f(n) \in \Theta\}$.
But $\{n: f(n) \in \Theta\}$ is the set of Gödel numbers of **theorems** of **Q** ,
which has just been shown **not** to be **decidable**.
- Thus Θ is **not** recursive and **FOL** is **not** decidable ■

Indefinability of Arithmetical Truth [optional start]

- **Tarski's Theorem**: the set of Gödel numbers of **true** sentences of arithmetic is **not definable** in arithmetic.
- **proof**: suppose some formula $C(y)$ **defined** the set of truths.
Then for all sentences S in the language of arithmetic:
 - (i) if S then $\vdash_Q C(\ulcorner S \urcorner)$ and
 - (ii) if $\neg S$ then $\vdash_Q \neg C(\ulcorner S \urcorner)$

By the **diagonal lemma** there is a sentence G such that

$$(*)' \quad \vdash_Q G \leftrightarrow \neg C(\ulcorner G \urcorner).$$

G is either true or false, and since $C(y)$ **defines** the set of Gödel numbers of **true** sentences, exactly one of

$$\vdash_Q C(\ulcorner G \urcorner) \quad \text{or} \quad \vdash_Q \neg C(\ulcorner G \urcorner) \text{ must obtain.}$$

Indefinability of Arithmetical Truth [optional]

Suppose $\vdash_Q \neg C(\ulcorner G \urcorner)$. Then $\vdash_Q G$ by (*')
and $\vdash_Q C(\ulcorner G \urcorner)$ by (i), and Q is inconsistent.

Suppose $\vdash_Q C(\ulcorner G \urcorner)$. Then

$\vdash_Q G \rightarrow \neg C(\ulcorner G \urcorner)$ by (*'), contraposition yields

$\vdash_Q \neg \neg C(\ulcorner G \urcorner) \rightarrow \neg G$, then $\vdash_Q C(\ulcorner G \urcorner) \rightarrow \neg G$

MP yields $\vdash_Q \neg G$, and finally $\vdash_Q \neg C(\ulcorner G \urcorner)$ by (ii),
and Q is inconsistent.

So if Q is consistent then there is no such $C(y)$

and the set of Gödel numbers of **true** sentences
of arithmetic is **not definable** in arithmetic ■

Indefinability of ‘True-in- L ’ in L [optional]

- More general version of **Tarski’s Theorem**: suppose $\mathbf{Tr}(x)$ is a formula in a language L attaching to names of formulas of L , and $\mathbf{Tr}(x)$ is intended to be a **truth predicate** for L , in which case it must satisfy the Tarski biconditional schema:

for all sentences S of L ,

$$\vdash_L \mathbf{Tr}(\ulcorner S \urcorner) \leftrightarrow S$$

The Tarski biconditional schema is famously illustrated by the example:

The sentence ‘Snow is white’ is true iff snow is white.

Suppose further that the **diagonal function** is representable in L .

Indefinability of ‘True-in- L ’ in L [optional finish]

Since $\mathbf{Tr}(x)$ is a formula of L , so is $\neg \mathbf{Tr}(x)$

and by the **diagonal lemma** there is a sentence G of L such that

$$(*)' \quad \vdash_L G \leftrightarrow \neg \mathbf{Tr}(\ulcorner G \urcorner)$$

G is the notorious ‘liar’ sentence that ‘asserts its own falsity’

Since G is a sentnce of L , the Tarski biconditional schema must apply to G , yielding

$$\vdash_L \mathbf{Tr}(\ulcorner G \urcorner) \leftrightarrow G \quad \text{which, in combination with}$$

$$(*)' \quad \vdash_L G \leftrightarrow \neg \mathbf{Tr}(\ulcorner G \urcorner) \quad \text{yields the contradiction}$$

$$\vdash_L \mathbf{Tr}(\ulcorner G \urcorner) \leftrightarrow \neg \mathbf{Tr}(\ulcorner G \urcorner)$$

Conclusion: if L is consistent then it cannot contain its own truth predicate ■

Gödel's First Incompleteness Theorem

- A formal theory \mathbf{T} is (negation) **complete** iff for **all** sentences S in the language of \mathbf{T} , either $\vdash_{\mathbf{T}} S$ or $\vdash_{\mathbf{T}} \neg S$.
- So a formal theory \mathbf{T} is **incomplete** iff it is **not** the case that for **all** sentences S in the language of \mathbf{T} , either $\vdash_{\mathbf{T}} S$ or $\vdash_{\mathbf{T}} \neg S$.
- **Gödel's First Incompleteness Theorem** (1931):
If formal arithmetic is **consistent**, **then** it is **incomplete**.

Gödel's First Incompleteness Theorem

- **proof:** will construct a Gödel sentence S that ‘asserts its own **unprovability**’, and demonstrate that neither S nor $\neg S$ is **provable if** the formal theory of arithmetic is consistent.

To do this, will first need to scrutinize (and then ‘arithmetize’) the structure of formal proofs.

For present purposes we’ll think of axiomatic (‘Hilbert style’) formal proofs.

Basic ingredients required for an axiomatic system **AX**:
a set of **axioms** and a set of **inference rules**.

Formal Axiomatic Proofs

Then a **proof** of some conclusion C

from premises B_1, \dots, B_n

is a **finite sequence of formulas**,

F_1, F_2, \dots, F_k

where F_k is the conclusion C ,

and where each F_1, F_2, \dots , in the sequence is either one of the premises B_i , or is one of the axioms, or is obtained from some earlier F_i 's in the sequence by using a **rule of inference**.

If there is such a **proof** sequence, then we write

$B_1, \dots, B_n \vdash_{\text{AX}} C$

Formal Axiomatic Proofs

For convenience, a proof sequence can also be written vertically, as follows:

1. F_1
2. F_2
- ⋮
- $k.$ F_k (i.e., C)

- We will be concerned with axiomatic proofs of theorems of \mathcal{Q} (and axiomatic extensions),
where **FOL**= can be formalized in terms of finite collection of **axiom schemas** and the *single inference rule* of modus ponens (**MP**).

Formal Axiomatic Proofs

Here is such an axiomatic proof system for Propositional Logic using just the connectives \neg and \rightarrow

Logical Axioms Schemas

- (I) $A \rightarrow (B \rightarrow A)$
- (II) $((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
- (III) $(\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$

Any *instance* of an Axiom Schema is a **logical axiom**

Rule of Inference: **Modus Ponens**

If you have formulas A and $A \rightarrow B$ at some point in the proof sequence (in either order), then you can add B at a later point in the proof sequence.

[This axiomatic system for propositional logic is **complete**]

Formal Axiomatic Proofs

Here is an axiomatic proof for

$$\vdash P \rightarrow P \text{ (so no premises involved)}$$

1. $(P \rightarrow ((P \rightarrow P) \rightarrow P)) \rightarrow ((P \rightarrow (P \rightarrow P)) \rightarrow (P \rightarrow P))$
instance of (II)
2. $P \rightarrow ((P \rightarrow P) \rightarrow P)$ instance of (I)
3. $(P \rightarrow (P \rightarrow P)) \rightarrow (P \rightarrow P)$ MP 1, 2
4. $P \rightarrow (P \rightarrow P)$ instance of (I)
5. $P \rightarrow P$ MP 3, 4

Formal Axiomatic Proofs

- To extend this Propositional fragment to a formalization of full **FOL**= we need the following **Logical Axiom Schemas**:

(IV) $\forall x(\Phi \rightarrow \Psi) \rightarrow (\forall x\Phi \rightarrow \forall x\Psi)$

(V) $\Phi \rightarrow \forall x\Phi$ if x not free in Φ

(VI) $\forall x\Phi \rightarrow \Phi(x/t)$ where t replaces all free occurrences of x in Φ

(VII) $t = t$ for any term t

(VIII) $s_1 = t_1 \rightarrow (s_2 = t_2 \rightarrow (\dots \rightarrow s_n = t_n \rightarrow (f(s_1, \dots, s_n) = f(t_1, \dots, t_n))))$

for any $n \geq 1$, and $2n$ terms $s_1, \dots, s_n, t_1, \dots, t_n$ and n -place function symbol f

(IX) $s_1 = t_1 \rightarrow (s_2 = t_2 \rightarrow (\dots \rightarrow s_n = t_n \rightarrow (P(s_1, \dots, s_n) \rightarrow P(t_1, \dots, t_n))))$

for any $n \geq 1$, and $2n$ terms $s_1, \dots, s_n, t_1, \dots, t_n$ and n -place predicate symbol P

As before, any *instance* of an Axiom Schema is a **logical axiom**

Formal Axiomatic Proofs

We then add the 7 **non-logical** axioms of \mathcal{Q} to the **set of axioms**.

- Thus the formal theory \mathcal{Q} is the deductive closure in the language \mathcal{L} of the axiomatic system that we have just defined.
- Hence if sentence S of \mathcal{L} is a theorem of \mathcal{Q}
(written $\vdash_{\mathcal{Q}} S$ and implying that $S \in \mathcal{Q}$),
then there is a finite sequence of formulas,

$$F_1, F_2, \dots, F_k$$

where F_k is the sentence S , and each F_1, F_2, \dots , in the sequence is either an axiom of **FOL-**,

or an axiom of \mathcal{Q} , or is obtained from two previous formulas in the sequence F_n and F_m using **MP**.

Arithmetizing Proofs

- The set of (Gödel numbers of) axioms of **FOL=**
plus axioms of ***Q*** is **recursive** and hence is **definable** in ***Q***
Furthermore, it's a **recursive** matter to determine whether a
given sentence follows from 2 other sentences via **MP**:

Arithmetizing Proofs

E.g., consider the sequence $\dots, (A \rightarrow B), \dots, A, \dots, B$

$$gn[(] = 1, \quad gn[)] = 2, \quad gn[\rightarrow] = 39999$$

Suppose $gn[A] = n$ and $gn[B] = k$,

$$\text{then } gn[(A \rightarrow B)] = 1n39999k2$$

- So if sentence with Gödel number k follows in a proof sequence by MP,
then there had to be two previous entries in the sequence with Gödel numbers n and $1n39999k2$.
- Since there are only finitely many previous entries, in principle it's an effective matter to determine whether or not this is the case.

Arithmetizing Proofs

- Let the Gödel number of a proof be the Gödel number of the total expression consisting of the sentences of the proof sequence separated by commas, where $gn[,] = 29$.

As an example, consider a proof sequence of the form

$A, (A \rightarrow B), B$ [suppose A and $(A \rightarrow B)$ are axioms]

Since B is the last formula in the sequence, this is a proof of B .

Again, suppose $gn[A] = n$ and $gn[B] = k$

Then the Gödel number of the proof of B is $n291n39999k229k$

Let this number $n291n39999k229k = j$

Then j is the Gödel number of a proof

of the sentence with Gödel number k .

2-Place Proof Relation

- The technical relation **proof** is specified such that:
proof = { $\langle j, k \rangle$: j is the Gödel number of a proof of
the sentence with Gödel number k }.
proof is a recursive relation and hence is **definable** in \mathcal{Q} .
- Let the formula $Pr(x,y)$ define the relation **proof** in \mathcal{Q} .
So if $\langle j, k \rangle \in \text{proof}$ then $\vdash_{\mathcal{Q}} Pr(j, k)$ and
if $\langle j, k \rangle \notin \text{proof}$ then $\vdash_{\mathcal{Q}} \neg Pr(j, k)$

1-Place Proof Predicate

- Now take the formula $\textcolor{blue}{Pr}(x,y)$ and bind the free variable x with an existential quantifier to get $\exists \textcolor{brown}{x} \textcolor{blue}{Pr}(\textcolor{brown}{x},y)$.
- This formula has only the variable y free, and we will abbreviate

$$\exists \textcolor{brown}{x} \textcolor{blue}{Pr}(\textcolor{brown}{x},y) \text{ as } \textcolor{green}{Prov}(y).$$

Thus $\textcolor{green}{Prov}(y)$ ‘asserts’, **in the theory Q** ,

that there is a proof in Q of the sentence with Gödel number y ,
and hence that this sentence is a **theorem** of Q (!)

$\textcolor{green}{Prov}(y)$ has 3 essential features that will be used to characterize
the general notion of a **provability predicate**:

1-Place Proof Predicate

For all sentences A, B in the language of \mathcal{Q}

- (i) if $\vdash A$, then $\vdash \text{Prov}(\ulcorner A \urcorner)$

This property follows directly from the fact that $\text{Pr}(x,y)$ defines the relation **proof** in \mathcal{Q} .

For suppose $\vdash_{\mathcal{Q}} A$. Then there is a proof in \mathcal{Q} of A .

Let the Gödel number of the proof be m .

Then $\langle m, \text{gn}[A] \rangle \in \text{proof}$ and so $\vdash_{\mathcal{Q}} \text{Pr}(m, \ulcorner A \urcorner)$

hence $\vdash_{\mathcal{Q}} \exists x \text{Pr}(x, \ulcorner A \urcorner)$,

i.e. $\vdash \text{Prov}(\ulcorner A \urcorner)$

1-Place Proof Predicate

(ii) $\vdash \text{Prov}(\Gamma A \rightarrow B \sqcap) \rightarrow (\text{Prov}(\Gamma A \sqcap) \rightarrow \text{Prov}(\Gamma B \sqcap))$

This property follows directly from the fact that
a proof of B can be obtained from a proof of $A \rightarrow B$
and a proof of A by writing one after the other.

In turn, this can be formalized in \mathcal{Q} .

(iii) $\vdash \text{Prov}(\Gamma A \sqcap) \rightarrow \text{Prov}(\Gamma \text{Prov}(\Gamma A \sqcap) \neg)$

This property is the **formalization** of (i) in the object language

Thus $\vdash A \text{ only if } \vdash \text{Prov}(\Gamma A \sqcap)$

/ | | | \

$\text{Prov}(\Gamma A \sqcap) \rightarrow \text{Prov}(\Gamma \text{Prov}(\Gamma A \sqcap) \neg)$

1-Place Proof Predicate

Prov (y) has the additional characteristic of ‘correctness’:

(iv) if $\vdash \text{Prov} (\ulcorner A \urcorner)$, then $\vdash A$

This property also follows directly from the fact that ***Pr*** (x,y)
defines the relation **proof** in ***Q***.

if $\vdash \text{Prov} (\ulcorner A \urcorner)$, i.e. $\vdash \exists x \text{Pr} (x, \ulcorner A \urcorner)$,

then this is a true statement in arithmetic, so there is some number m such that m is the Gödel number of a proof of A .

thus $\vdash A$

Proof of Gödel's First Incompleteness Theorem

- The **diagonal lemma**, (i) and (iv) are sufficient to now prove **Gödel's First Incompleteness Theorem**:
if formal arithmetic is **consistent**, then it is **incomplete**.
- **proof:** Since $\neg \text{Prov}(y)$ is a formula with only the variable y free, it follows by the **diagonal lemma** that there is a sentence S such that

$$(*'') \quad \vdash S \leftrightarrow \neg \text{Prov}(\ulcorner S \urcorner)$$

Assume $\vdash S$.

Then $\vdash \neg \text{Prov}(\ulcorner S \urcorner)$ by $(*'')$

and $\vdash \text{Prov}(\ulcorner S \urcorner)$ by reductio hypothesis and (i)
and arithmetic is **inconsistent**.

Proof of Gödel's First Incompleteness Theorem

Assume $\vdash \neg S$.

By (*'') alone we get $\vdash \neg \text{Prov}(\ulcorner S \urcorner) \rightarrow S$

and contraposition and double neg. elim. on this yield

$\vdash \neg S \rightarrow \text{Prov}(\ulcorner S \urcorner)$.

With the reductio hypothesis $\vdash \neg S$, MP, and distribution of \vdash over the conditional we get $\vdash \text{Prov}(\ulcorner S \urcorner)$,
and (iv) yields $\vdash S$.

So $\vdash S$ and $\vdash \neg S$ and arithmetic is **inconsistent**.

Thus *if* formal arithmetic is consistent, *then*

neither $\vdash S$ *nor* $\vdash \neg S$, and formal arithmetic is **incomplete** ■

Consequences for Hilbert's Program

- A fundamental wedge has thereby been driven between **truth** in the **intended model** and **provability** in a **formal system**.
- So Gödel's First Incompleteness Theorem is generally taken to **refute** one of the basic tenets of Hilbert's Program by establishing that **not** all of the **true** statements in elementary arithmetic can be **proved** in an axiomatizable formal theory.
- All statements in the language of arithmetic are either **true** or **false** in the *intended model*, but neither S nor $\neg S$ is **provable**.
- Hence even elementary arithmetic cannot be reduced to “... an inventory of provable formulas”

Consequences for Hilbert's Program

- So it would appear that the method of axiomatization and finitary proof is inherently too weak to capture all mathematical truths.
- And this type of ‘foundation’ thereby seems to be rendered inadequate, *in principle*.
- However, the areas in which such ‘undecidable’ sentences (i.e. where neither A nor $\neg A$ is provable) actually arise appears to be quite narrow and specialized.
- No arithmetical conjecture or problem that has occurred to mathematicians in contexts, outside of logic and the foundations of mathematics,
has ever been proved to be **undecidable**.

Consequences for Hilbert's Program

- But there has been work to find statements closer to ‘ordinary mathematics’ than Gödel sentences that *are undecidable* in PA.
e.g. **The Paris-Harrington Theorem** (1976)

Consequences for Philosophy of Mind?

- Since $S \leftrightarrow \neg \text{Prov}(\Gamma S \upharpoonright)$, the ‘Gödel sentence’ S can be interpreted as ‘asserting its own unprovability’, and if arithmetic is consistent then S is unprovable, hence **true**.
- The human mind seems able to intuitively grasp the **truth** of the Gödel sentence, even though the sentence does not follow as a consequence of the finitary deductive system.
- Does this show that the human mind cannot be reduced to a finitary deductive system?
- And given the relationship between computation and finitary deductive systems, does this show that the **Computational Theory of Mind** is **false??**

Consequences for Philosophy of Mind?

- Both Lucas, and more recently Penrose, have put forward arguments to this effect.
- But are such arguments convincing?
- Gödel's theorem establishes that if the formal theory of arithmetic is consistent, then S is not provable.
- Thus in order for the human mind to ‘know’ that S is **not provable** and hence **true**,
the human mind must first ‘know’ that arithmetic is consistent.
- **Q** is quite a simple theory,
so suppose we grant the claim that we *know* that
 Q is consistent and that S is **unprovable** and thus **true**.

Consequences for Philosophy of Mind?

- Indeed, *whenever* we know a formal theory T (in which **diag** is representable) to be consistent, we also know the truth of a corresponding sentence S that is not provable in T .
- Does this show that the human mind has a special power and can thereby outperform any given formal system?
- **No –**
- While it *does* follow that **we know** the conditional
- ‘**If** the formal system T is consistent **then** the corresponding sentence S is true’
this is not equivalent to the claim
- ‘**If** the formal system T is consistent **then we know** that the corresponding sentence S is true’

Consequences for Philosophy of Mind?

- Thus consider cases where the formal system is so complex that we have **no idea** whether or not it is **consistent**.
- Then we also have **no idea** whether or not the corresponding sentence ***S*** is **true!**
- So what strictly follows from the claim that ***we know*** the simple system ***Q*** is consistent, and hence that the original **Gödel sentence** *is true?*
- Only that, *if* some version of CTM is the case, *then* the formal system on which the human mind operates isn't ***Q*!**
- But the possibility still remains open that it could be some other, much more sophisticated and elaborate formal system (of which we are unaware).

Inexhaustability

- While these standard attempts to derive anti-CTM consequences from Gödel's Theorem are not successful, Gödel himself put forward a potential argument based on the seeming ‘inexhaustability’ of human mathematical knowledge: it may not be possible to specify **any one** formal system which completely **exhausts** all of our mathematical knowledge.
- If this is true, then perhaps human mathematical knowledge cannot ultimately be captured in terms of computations and finitary deductive mechanisms?

Proving Gödel's First Theorem the Fast Way

- We have stated and proved Gödel's First Incompleteness Theorem in the ‘classic’ manner, by constructing a sentence S that ‘asserts its own unprovability’.
- However, since the time of Gödel’s original Theorem, more streamlined methods have been devised for establishing a theoretically equivalent result:
- A formal theory T is **axiomatizable** iff there is a **decidable subset** of T whose logical consequences are the **theorems** of T .
- So any **decidable** theory **is axiomatizable** (why?), but not every **axiomatizable** theory is **decidable** (why?).

Proving Gödel's First Theorem the Fast Way

- As above, a formal theory \mathbf{T} is **complete** iff for **all** sentences S in the language of \mathbf{T} , either $\vdash_{\mathbf{T}} S$ or $\vdash_{\mathbf{T}} \neg S$.
- Furthermore,
 - (i) any **axiomatizable, complete** theory is **decidable** (Theorem 5, B&J p. 177).
 - (ii) **no consistent** extension of Q is **decidable**.
- It follows as an immediate consequence of (i) and (ii) that:
Gödel's First Incompleteness Theorem (reformulated)
There is **no consistent, complete**, and **axiomatizable** extension of Q ■

Logic, Computability and Incompleteness

Gödel's Second Theorem, Löb's
Theorem and the Logic of Provability

Hilbert's Program (again)

- As we saw before, Hilbert's **Formalist Program** for the foundation of mathematics advocated an approach in which all of mathematics is deducible in an **axiomatizable formal theory** where the axioms themselves are **provably consistent**.
- In particular, Hilbert sought an ‘internal’ and finitary consistency proof for the axioms of elementary number theory.
- **Gödel’s First Theorem** can be seen as refuting Hilbert’s goal of reducing arithmetic to an ‘inventory of provable formulas’, while **Gödel’s Second Theorem** can be seen as undermining Hilbert’s quest for an internal consistency proof for the axioms of arithmetic.

The Unprovability of Consistency

Gödel's First Incompleteness Theorem (roughly): *if* formal arithmetic is consistent, *then* neither S nor $\neg S$ is provable, where S is constructed such that $\vdash S \leftrightarrow \neg \text{Prov}(\ulcorner S \urcorner)$.

So if arithmetic is consistent then S is unprovable, hence **true**.
Is formal arithmetic consistent?

Gödel's Second Incompleteness Theorem (roughly):

if formal arithmetic is consistent,
then it cannot prove its own consistency.

A **basic fact** of Classical Logic: if formal arithmetic is a consistent theory, then at least one sentence is unprovable.

The Unprovability of Consistency

So let the **consistency of arithmetic** be expressed
in arithmetic (!) by the sentence:

$$\neg \text{Prov} (\ulcorner 0 = 0' \urcorner) \quad (\underline{\text{con}})$$

Gödel's Second Incompleteness Theorem:

if $\vdash \underline{\text{con}}$ *then* arithmetic is **inconsistent**.

As above, Gödel's First Theorem states that

(#) *if* arithmetic is consistent, *then* it is not provable that S

Since **con** is a sentence of the object language that expresses the consistency of arithmetic, then a **formalization** of the **First Theorem**, in terms of (#), would yield:

$$\underline{\text{con}} \rightarrow \neg \text{Prov} (\ulcorner S \urcorner) \quad (!)$$

The Unprovability of Consistency

or equivalently $\underline{\text{con}} \rightarrow S$

And *if* this formalization of the First Theorem were provable in formal arithmetic, *then* it follows (on the assumption of consistency) that **not** $\vdash \underline{\text{con}}$.

Why? Because $\vdash \underline{\text{con}} \rightarrow S$ entails that

if $\vdash \underline{\text{con}}$ *then* $\vdash S$

And if arithmetic is consistent, then (by 1st Theorem)

not $\vdash S$, and contraposition on the above yields **not** $\vdash \underline{\text{con}}$.

So *if* the conditional $\underline{\text{con}} \rightarrow S$ is a theorem of arithmetic,
then if arithmetic is consistent,

then the consistency sentence $\underline{\text{con}}$ cannot be provable in arithmetic.

Gödel's Second Incompleteness Theorem

Thus to prove Gödel's Second Incompleteness Theorem need to show that $\vdash \text{con} \rightarrow S$, and the underivability of the consequent S will yield the underivability of the antecedent con.

proof: will require the **diagonal lemma** and characteristics (i) – (iii) of a **provability predicate**.

As before, the **diagonal lemma** gives

$\vdash S \leftrightarrow \neg \text{Prov}(\ulcorner S \urcorner)$, which yields $\vdash S \rightarrow \neg \text{Prov}(\ulcorner S \urcorner)$
and then $\vdash \neg \neg \text{Prov}(\ulcorner S \urcorner) \rightarrow \neg S$ and finally

$$(0) \quad \vdash \text{Prov}(\ulcorner S \urcorner) \rightarrow \neg S$$

Recall (i) if $\vdash A$, then $\vdash \text{Prov}(\ulcorner A \urcorner)$, which applied to (0)
gives $\vdash \text{Prov}(\ulcorner \text{Prov}(\ulcorner S \urcorner) \rightarrow \neg S \urcorner)$

Gödel's Second Incompleteness Theorem

Recall (ii) $\vdash \text{Prov}(\Gamma A \rightarrow B \top) \rightarrow (\text{Prov}(\Gamma A \top) \rightarrow \text{Prov}(\Gamma B \top))$

Which gives $\vdash \text{Prov}(\Gamma \underline{\text{Prov}(\Gamma S \top)} \rightarrow \underline{\neg S} \top) \rightarrow$
 $(\text{Prov}(\Gamma \text{Prov}(\Gamma S \top) \top) \rightarrow \text{Prov}(\Gamma \neg S \top))$

Now modus ponens gives

(1) $\vdash \text{Prov}(\Gamma \text{Prov}(\Gamma S \top) \top) \rightarrow \text{Prov}(\Gamma \neg S \top)$

Similarly, take $\vdash (S \wedge \neg S) \rightarrow \mathbf{o} = \mathbf{o}'$ and apply (i) to get

$\vdash \text{Prov}(\Gamma \underline{(S \wedge \neg S)} \rightarrow \underline{\mathbf{o} = \mathbf{o}'} \top)$ then apply (ii) to get

$\vdash \text{Prov}(\Gamma \underline{(S \wedge \neg S)} \rightarrow \underline{\mathbf{o} = \mathbf{o}'} \top) \rightarrow$

$(\text{Prov}(\Gamma (S \wedge \neg S) \top) \rightarrow \text{Prov}(\Gamma \mathbf{o} = \mathbf{o}' \top))$ and MP to get

(\\$) $\vdash \text{Prov}(\Gamma (S \wedge \neg S) \top) \rightarrow \text{Prov}(\Gamma \mathbf{o} = \mathbf{o}' \top)$

Gödel's Second Incompleteness Theorem

Note that $\vdash \text{Prov}(\Gamma(S \wedge \neg S) \top) \leftrightarrow (\text{Prov}(\Gamma S \top) \wedge \text{Prov}(\Gamma \neg S \top))$

Substitution of provable equivalents in (\$) yields

$$(2) \quad \vdash (\text{Prov}(\Gamma S \top) \wedge \text{Prov}(\Gamma \neg S \top)) \rightarrow \text{Prov}(\Gamma \mathbf{0} = \mathbf{0}' \top)$$

Recall (iii) $\vdash \text{Prov}(\Gamma A \top) \rightarrow \text{Prov}(\Gamma \text{Prov}(\Gamma A \top) \top)$

which gives: (3) $\vdash \text{Prov}(\Gamma S \top) \rightarrow \underline{\text{Prov}(\Gamma \text{Prov}(\Gamma S \top) \top)}$

Recall (1) $\vdash \underline{\text{Prov}(\Gamma \text{Prov}(\Gamma S \top) \top)} \rightarrow \text{Prov}(\Gamma \neg S \top)$

which in combination with (3) yields

$$(4) \quad \vdash \text{Prov}(\Gamma S \top) \rightarrow (\text{Prov}(\Gamma S \top) \wedge \text{Prov}(\Gamma \neg S \top))$$

With (4) and (2) we get $\vdash \text{Prov}(\Gamma S \top) \rightarrow \text{Prov}(\Gamma \mathbf{0} = \mathbf{0}' \top)$

And by contraposition

$$(5) \quad \vdash \neg \text{Prov}(\Gamma \mathbf{0} = \mathbf{0}' \top) \rightarrow \neg \text{Prov}(\Gamma S \top).$$

Gödel's Second Incompleteness Theorem

Since con has been defined as the sentence $\neg Prov(\ulcorner \mathbf{o} = \mathbf{o}' \urcorner)$
and $\vdash S \leftrightarrow \neg Prov(\ulcorner S \urcorner)$,

Rewriting (5) $\vdash \neg Prov(\ulcorner \mathbf{o} = \mathbf{o}' \urcorner) \rightarrow \neg Prov(\ulcorner S \urcorner)$ as
 $\vdash \underline{\text{con}} \rightarrow S$ yields the desired result
and if arithmetic is consistent then not $\vdash \underline{\text{con}}$ ■

This is another incompleteness result, because if arithmetic is consistent then con is **true**,
so if arithmetic is consistent then con is yet another **unprovable truth**, and the wedge between **truth** and **provability** that started with the First Theorem is driven even deeper.

Löb's Theorem

We've just seen a 'direct' proof of Gödel's Second Incompleteness Theorem.

However, it is also possible to prove this theorem as a corollary of the closely related but more general Löb's Theorem, which is motivated as follows.

Another way to think of provable consistency is in terms of the characteristic

for all sentences A ,

$$(\text{v}) \quad \vdash \text{\color{green}Prov} (\ulcorner A \urcorner) \rightarrow A$$

Which 'asserts that' if a sentence is provable, then it is true, so that it's **provable** in the formal theory that only **truths** are **provable**.

Löb's Theorem

Löb's Theorem: if $\mathbf{B}(y)$ is a provability predicate for some theory \mathbf{T} that extends \mathbf{Q} , then for any sentence A in the language of \mathbf{T}

if $\vdash_{\mathbf{T}} \mathbf{B}(\ulcorner A \urcorner) \rightarrow A$ *then* $\vdash_{\mathbf{T}} A$

proof: assume (1) $\vdash_{\mathbf{T}} \mathbf{B}(\ulcorner A \urcorner) \rightarrow A$.

Let $\mathbf{D}(y)$ be the formula $\mathbf{B}(y) \rightarrow A$.

The diagonal lemma guarantees a sentence C such that

$\vdash_{\mathbf{T}} C \leftrightarrow \mathbf{D}(\ulcorner C \urcorner)$, i.e.

(2) $\vdash_{\mathbf{T}} C \leftrightarrow (\mathbf{B}(\ulcorner C \urcorner) \rightarrow A)$

(1) and (2) in combination with (i) – (iii) yield $\vdash_{\mathbf{T}} A$ ■

(see B&J p. 187 for the detailed steps).

Gödel's Second Incompleteness Theorem Again

Reformulation of Gödel's Second Incompleteness Theorem:
if $\mathbf{B}(y)$ is a provability predicate for some consistent theory \mathbf{T}
that extends \mathbf{Q} , then \mathbf{T} cannot prove its own consistency

i.e. $\text{not } \vdash_{\mathbf{T}} \neg \mathbf{B}(\ulcorner \mathbf{o} = \mathbf{o}' \urcorner)$

proof: suppose $\vdash_{\mathbf{T}} \neg \mathbf{B}(\ulcorner \mathbf{o} = \mathbf{o}' \urcorner)$.

Then $\vdash_{\mathbf{T}} \mathbf{B}(\ulcorner \mathbf{o} = \mathbf{o}' \urcorner) \rightarrow \mathbf{o} = \mathbf{o}'$ (by prop. logic)

and by Löb's Theorem $\vdash_{\mathbf{T}} \mathbf{o} = \mathbf{o}'$

But $\vdash_{\mathbf{Q}} \neg \mathbf{o} = \mathbf{o}'$, and thus \mathbf{T} is inconsistent

So if \mathbf{T} is consistent, then it can't prove its own consistency ■

The Henkin Sentence

Historically, Löb's Theorem was used to answer a question posed by Henkin with regard to the the ‘Henkin Sentence’ H . Unlike the Gödel sentence S , H ‘asserts its own **provability**’.

Given the proof predicate $\textcolor{green}{Prov}(y)$

The diagonal lemma guarantees an H such that

$$\vdash H \leftrightarrow \textcolor{green}{Prov}(\ulcorner H \urcorner)$$

Is H provable (and hence **true**)?

It follows directly from Löb's Theorem that $\vdash H$ ■

Modal Logic of Provability

The defining characteristics of a provability predicate have very clear analogues in modal logic:

(i) *if* $\vdash A$, *then* $\vdash \text{Prov}(\Box A)$

corresponds to the modal inference rule of *Necessitation*

if $\vdash A$, *then* $\vdash \Box A$

(ii) $\vdash \text{Prov}(\Box A \rightarrow \Box B) \rightarrow (\text{Prov}(\Box A) \rightarrow \text{Prov}(\Box B))$

corresponds to the K axiom schema

$\Box(A \rightarrow C) \rightarrow (\Box A \rightarrow \Box C)$

Modal Logic of Provability

(iii) $\vdash \text{Prov}(\Box A^\perp) \rightarrow \text{Prov}(\Box \text{Prov}(\Box A^\perp)^\perp)$

corresponds to the **S4** axiom schema

$$\Box A \rightarrow \Box \Box A$$

(v) $\vdash \text{Prov}(\Box A^\perp) \rightarrow A$

corresponds to the **T** axiom schema

$$\Box A \rightarrow A$$

The modal theory **S4** is the closure of all the **K**, **S4** and **T** axioms under **logical consequence** and the rule of *Necessitation*.

Modal Logic of Provability

Hence **S4** is too strong to represent the logic of arithmetical proof, as shown by the Gödel-Löb results.

Instead, need to replace **T** axiom schema with the formalized and then modalized version of Löb's Theorem.

Recall **Löb's Theorem**:

if $\vdash \text{Prov}(\Box A \Box) \rightarrow A$ *then* $\vdash A$

Löb's Theorem **formalized in arithmetic**:

$\vdash \text{Prov}(\Box A \Box) \rightarrow A$ *only if* $\vdash A$

/ / | / / \

$\vdash \text{Prov}(\Box \text{Prov}(\Box A \Box) \rightarrow A \Box) \rightarrow \text{Prov}(\Box A \Box)$

Modal Logic of Provability

The **modal** version of the formalization

$$\vdash \textcolor{green}{Prov}(\Box \textcolor{green}{Prov}(\Box A^\top) \rightarrow A^\top) \rightarrow \textcolor{green}{Prov}(\Box A^\top)$$

yields the **G** axiom schema:

$$\Box(\Box A \rightarrow A) \rightarrow \Box A$$

- The **modal theory G** is the closure of all the **K**, **S4** and **G** axioms under logical consequence and the rule of *Necessitation*.
- Hence the **modal theory G** represents the **logic of provability in formal arithmetic**.