Independence

A formula F is called independent of some theory if that theory is unable to assign a truth value to it.

Independence

A formula F is called independent of some theory if that theory is unable to assign a truth value to it.

When is an axiom independent of other axioms?
We use structures for this!

If we want to show "Axiom n" of a theory is independent of all the other axioms of the theory, all we must do is find a structure which fails to models "Axiom n" while simultaneously modeling the other axioms.

Euclidean Geometry

Informally: the axioms of “Euclidean geometry”, as Euclid wrote it, are:

1. A line segment can be formed by any two points.

Euclidean Geometry

Informally: the axioms of “Euclidean geometry”, as Euclid wrote it, are:

1. A line segment can be formed by any two points.
2. A(n infinite) line can be formed from any line segment.

Euclidean Geometry

Informally: the axioms of “Euclidean geometry”, as Euclid wrote it, are:

1. A line segment can be formed by any two points.
2. A(n infinite) line can be formed from any line segment.
3. Given any line segment, a circle can be drawn having that segment as a radius and one endpoint as its center.

Euclidean Geometry

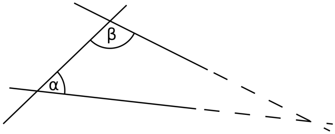
Informally: the axioms of “Euclidean geometry”, as Euclid wrote it, are:

1. A line segment can be formed by any two points.
2. A(n infinite) line can be formed from any line segment.
3. Given any line segment, a circle can be drawn having that segment as a radius and one endpoint as its center.
4. All right angles are equal.

Euclidean Geometry

Informally: the axioms of "Euclidean geometry", as Euclid wrote it, are:

1. A line segment can be formed by any two points.
2. A(n infinite) line can be formed from any line segment.
3. Given any line segment, a circle can be drawn having that segment as a radius and one endpoint as its center.
4. All right angles are equal.
5. ("Parallel postulate") If a line segment intersects two straight lines forming two interior angles on the same side that sum to less than 2 right angles, then the two lines (if extended indefinitely) meet on that side which the angles sum to less than two right angles.



Euclidean Geometry

Axiom 5 “feels like” a statement that could be proven from Axioms 1-4.

Euclidean Geometry

Axiom 5 “feels like” a statement that could be proven from Axioms 1-4.

Consequently: mathematicians starting in Euclid’s time tried to prove it from Axioms 1-4 for hundreds of years.

Euclidean Geometry

Axiom 5 “feels like” a statement that could be proven from Axioms 1-4.

Consequently: mathematicians starting in Euclid’s time tried to prove it from Axioms 1-4 for hundreds of years.

Their efforts ended in failure! Led to... projective geometry, spherical geometry, hyperbolic geometry, and other “non-Euclidean” geometries...

Finite geometries - five-point geometry

We have three axioms:

1. there are exactly five points
2. each two distinct points have exactly one line on both of them
3. each line has exactly two points

Theorem: There are exactly 10 lines.

Theorem: Each point touches exactly four lines.

Finite geometries – five-point geometry

- Create a model to show that Axiom 1 is independent of Axioms 2 and 3
- Create a model to show show Axiom 2 is independent of Axioms 1 and 3
- Create a model to show show Axiom 3 is independent of Axioms 1 and 2

Finite geometries – four-line geometry

We have three axioms:

1. there exist exactly four lines
2. any two distinct lines have exactly one point in common
3. each point lies on exactly two lines

Theorem: There are exactly six points.

Theorem: Each line contains exactly three points.

Consistency and completeness

A theory is called consistent if it does **not** derive a contradiction. A theory is called complete if every sentence (or its negation) has a proof in that theory (i.e. nothing is "undecidable").

A desirable goal: to have an axiomatic system be both complete and consistent.

Example: "Naive set theory" is **not** consistent because we were able to derive a contradiction from it (Russell's paradox).

Example: We ~~**don't know**~~ **cannot tell** whether or not "first order arithmetic" is consistent (from inside of first order arithmetic...).

Consistency and completeness

The following theory, called Presburger arithmetic, is a complete and consistent theory of (additive) arithmetic with a single one-term predicate S ("successor") and a two-term predicate $+$:

1. $(\forall x)\neg(0 = Sx)$
2. $(\forall x)(\forall y)(Sx = Sy \rightarrow x = y)$
3. $(\forall x)(x + 0 = x)$
4. $(\forall x)(\forall y)(x + Sy = S(x + y))$
5. (Induction Schema) For any first predicate Px, the following is an axiom:
$$(P(0) \wedge (\forall x)(Px \rightarrow P(Sx))) \rightarrow (\forall y)(Py)$$

There is, in fact, a "decision procedure" that can be used to determine if a given formula F is true or false in Presburger arithmetic! However, Presburger arithmetic is "weak" in that it cannot even define prime numbers (or multiplication, in general). (also see "Skolem Arithmetic")

Peano Arithmetic

Note: axioms 1-7 match “first order arithmetic”; axiom 8 is “induction”

1. $(\forall x)\neg(0 = Sx)$
2. $(\forall x)(\forall y)(Sx = Sy \rightarrow x = y)$
3. $(\forall y)(y = 0 \vee (\exists x)(Sx = y))$
4. $(\forall x)(x + 0 = x)$
5. $(\forall x)(\forall y)(x + Sy = S(x + y))$
6. $(\forall x)(x \cdot 0 = 0)$
7. $(\forall x)(\forall y)(x \cdot Sy = (x \cdot y) + x)$
8. (Induction schema) For any predicate Px, the following is an axiom:

$$(P(0) \wedge (\forall x)(Px \rightarrow P(Sx))) \rightarrow (\forall y)(Py).$$

Consistency and completeness

A theory is called consistent if it does **not** derive a contradiction. A theory is called complete if every sentence (or its negation) has a proof in that theory (i.e. nothing is “undecidable”).

Is Peano arithmetic consistent? Is it complete?

Iteration – Fibonacci sequence

Most generally, the word *recursion* captures the idea of self-reference and repetition.
Example: The Fibonacci sequence is often defined “iteratively” (with a “recurrence relation”):

$$(*) \qquad F(n+1) \overset{\mathrm{def}}{=} F(n) + F(n-1); F(0) = 1, F(1) = 1.$$

The numbers $F(0) = 1$ and $F(1) = 1$ are the “initial conditions”. To find the value of $F(2)$, simply plug in $n = 1$ into $(*)$ to arrive at:

$$F(2) = F(1) + F(0) = 1 + 1 = 2.$$

To find $F(3)$ plug in $n = 2$ into $(*)$ to get

$$F(3) = F(2) + F(1) = 2 + 1 = 3.$$

etc...

$$F(4) = F(3) + F(2) = 3 + 2 = 5.$$

Recursion – Factorial

The factorial function is $n! = n(n-1)(n-2)\dots(2)(1)$. For example,

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24.$$

It can also be defined recursively (we write fac for simplicity):

$$\mathrm{fac}(n+1) = (n+1) \cdot \mathrm{fac}(n); \mathrm{fac}(1) = 1.$$

This definition shows that

$$\mathrm{fac}(2) = \mathrm{fac}(1+1) = (1+1)\mathrm{fac}(1) = 2$$

Recursion – Ackermann function

The Ackermann Ack function is defined by

$$\mathrm{Ack}(x,y)=\begin{cases} y+1 & ; x=0 \\ \mathrm{Ack}(x-1,1) & ; y=0 \\ \mathrm{Ack}(x-1,\mathrm{Ack}(x,y-1)) & ; \text{otherwise.} \end{cases}$$

Calculate...

$$\mathrm{Ack}(0,0)\overset{x=0,y=0}{=}0+1=1,$$

$$\mathrm{Ack}(0,1)\overset{x=0,y=1}{=}1+1=2,$$

$$\vdots$$

$$\mathrm{Arc}(0,y)\overset{x=0,y=y}{=}y+1,$$

$$\mathrm{Ack}(1,0)\overset{x=1,y=0}{=}\mathrm{Ack}(1-1,1)=\mathrm{Ack}(0,1)=2$$

Recursion – Ackermann function

The Ackermann Ack function is defined by

$$\mathrm{Ack}(x, y) = \begin{cases} y + 1 & ; x = 0 \\ \mathrm{Ack}(x - 1, 1) & ; y = 0 \\ \mathrm{Ack}(x - 1, \mathrm{Ack}(x, y - 1)) & ; \text{otherwise.} \end{cases}$$

Calculate...

$$\mathrm{Ack}(0, 0) \overset{x=0, y=0}{=} 0 + 1 = 1,$$

$$\mathrm{Ack}(0, 1) \overset{x=0, y=1}{=} 1 + 1 = 2,$$

$$\vdots$$

$$\mathrm{Arc}(0, y) \overset{x=0, y=y}{=} y + 1,$$

$$\mathrm{Ack}(1, 0) \overset{x=1, y=0}{=} \mathrm{Ack}(1 - 1, 1) = \mathrm{Ack}(0, 1) = 2$$

This function gets very large very fast...

$$\mathrm{Ack}(4, 3) = 2^{2^{65536}} - 3 = 2^{2^{2^{2^{2^{2}}}}} - 3$$

Primitive recursive functions

A *primitive recursive function* is a special type of recursively defined function. Their technical definition is too complicated for here, but the factorial function defined earlier is primitive recursive while the Ackermann function is *not*.

Theorem: Any primitive recursive function can be defined in Peano arithmetic.

Gödel numbers

The Gödel number of a formula in a language is a number assigned, uniquely, to each formula in that language. We do this by associating each symbol in a formula to a number.

Our assignment for Peano arithmetic:

$0 \leftrightarrow 1$

$\cdot \leftrightarrow 2$

$+ \leftrightarrow 3$

$= \leftrightarrow 4$

$(\leftrightarrow 5$

$) \leftrightarrow 6$

$S0 \leftrightarrow 7$

$SS0 \leftrightarrow 8$

$SSS0 \leftrightarrow 9$

$\vdots$

Gödel numbers

Let's assign a Gödel number to the following formula of Peano arithmetic:

$$S0 \cdot S0 = S0.$$

Since $S0 \leftrightarrow 7$, $\cdot \leftrightarrow 2$, $= \leftrightarrow 4$, we will encode the formula as an integer in the following way: consider the prime numbers $\{2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots\}$; use the assigned value of each symbol as the exponent of each prime, in order, for each symbol

$$S0 \cdot S0 = S0 \leftrightarrow 2^7 3^2 5^7 7^4 11^7 = 421098278139000000$$

Since we are using primes, this process can also be reversed: what formula is encoded by the number 152127360?

$$152127360 \overset{\text{factor}}{=} 2^7 3^2 5^1 7^4 11^1 \leftrightarrow S0 \cdot 0 = 0$$

Super Gödel numbers

The process described earlier can be applied to any sequence of integers.

Once we have Gödel numbers of formulas, we can talk about "super" Gödel numbers, which is the same process applied to proofs of formulas: a proof of a theorem is a list of formulas (in our deduction!). Each formula in the proof has its own Gödel number.

We say the "super Gödel" number of a proof defined by a sequence of formulas whose Gödel numbers are $\{g_1, g_2, \ldots, g_n\}$ to be the number $2^{g_1}3^{g_2}5^{g_3}7^{g_4}11^{g_5}13^{g^6}\ldots$.

“Super” Gödel numbers

From a formal proof that $S0 \cdot S0 = S0$:

Formula in proof	Gödel number
(1) $S0 \cdot S0 = (S0 \cdot 0) + S0$	$g_1 = 2^7 3^2 5^7 7^4 11^5 13^7 17^2 19^1 23^6 29^3 31^7$
(2) $S0 \cdot 0 = 0$	$g_2 = 2^7 3^2 5^1 7^4 11^1 3^1$
(3) $S0 \cdot S0 = 0 + S0$	$g_3 = 2^7 3^2 5^7 7^4 11^1 13^7$
(4) $S0 + 0 = 0 + S0$	$g_4 = 2^7 3^3 5^1 7^4 11^1 13^3 17^7$
(5) $S0 + 0 = S0$	$g_5 = 2^7 3^3 5^1 7^4 11^7$
(6) $S0 = 0 + S0$	$g_6 = 2^7 3^4 5^1 7^3 11^7$
(7) $S0 \cdot S0 = S0$	$g_7 = 2^7 3^2 5^7 7^4 11^7$

The super Gödel number of this proof of the formula $\phi = S0 \cdot S0 = S0$ is

$$\ulcorner \phi \urcorner = 2^{g_1} 3^{g_2} 5^{g_3} 7^{g_4} 11^{g_5} 13^{g_6} 17^{g_7}$$

Proof function

The Prf function $\mathrm{Prf}(m, n)$ returns "True" provided that m is the super Gödel number of a proof of the formula whose Gödel number is n. It returns "False" otherwise.

All that is required to check whether $\mathrm{Prf}(m, n)$ is true or false is to decode the number m into a proof and decode n into a formula. Observe whether or not the proof is a proof of n.

Theorem: Prf is primitive recursive.

Diagonalization and G

If ϕ is a formula, then the diagonalization of ϕ is the formula

$$\operatorname{diag}(\phi) = (\exists y)((y = \ulcorner \phi \urcorner) \wedge \phi).$$

We define $\mathrm{Gdl}(m, n) = \mathrm{Prf}(m, \operatorname{diag}(n))$; this is true whenever m is the super Gödel number of a proof of the diagonalization of the formula whose Gödel number is n.

The self-reference: define the formula $Uy = (\forall x)\neg \mathrm{Gdl}(x, y)$. The diagonalization of this formula is the formula we call G:

$$G \overset{\mathrm{def}}{=} \operatorname{diag}(Uy) = (\exists y)(y = \ulcorner Uy \urcorner \wedge Uy).$$

What does G say?

$$G \overset{\text{def}}{=} (\exists y)(y = \ulcorner Uy \urcorner \wedge Uy)$$

1. $G = (\exists y)(y = \ulcorner Uy \urcorner \wedge Uy)$
2. $G =$ There is y such that $y =$ super Gödel number of a proof of the formula "Uy" and Uy
3. $G =$ There is y such that $y =$ super Gödel number of a proof of "$(\forall x)\neg \mathrm{Gdl}(x, y)$" and $(\forall x)\neg \mathrm{Gdl}(x, y)$
4. $G =$ There is y such that $y =$ super Gödel number of a proof of "$(\forall x)\neg \mathrm{Gdl}(x, y)$" and no (natural) number x exists such that x is the super Gödel number of a proof of $(\exists y)(y = \ulcorner Uy \urcorner \wedge Uy)$

Notice: the formula in line 1 appeared again inside of line 4...

Peano arithmetic does not prove G

Proof sketch: Suppose a proof exists for the formula G. From this we see that G is true, and moreover the proof has a super Gödel number, say, ℓ.

But if G is true, it means there is a number y, whose value is the Gödel number of the formula G, and **no** number x exists which is the super Gödel number of a proof of G.

So simultaneously the number ℓ would exist while we would also declare that no such number $x = \ell$ can exist. A contradiction! Therefore due to proof by contradiction... Peano arithmetic does not prove G!

Is G true?

Depends... from the perspective of “true” meaning “there exists a proof of it”, then no, it is not.

However, if you think about G as encoding “I am not provable”, then because we have already argued that there is no proof for G, it is in fact *true*... (“metamathematically”).

From this we see that G “must be” true while also not having a proof (to have a proof would contradict itself). This is precisely why Peano arithmetic is **not** complete.

The 2nd incompleteness theorem

Gödel's first incompleteness theorem shows that Peano arithmetic is not complete: G is true but not provable.

Natural idea: add G to the list of axioms. Now we have a "stronger theory" in which G has a proof. Do this "as much as necessary" to get a "sufficiently powerful" theory that is complete.

Gödel's 2nd incompleteness theorem tells us that will *always* fail: any theory that can "express" Peano suffers from its own G-like sentence.

Halting problem

Can you write down a general method ("algorithm") that takes the source code of a computer program as an input and returns 1 if the inputted program "halts" (or "terminates" or "stops") or returns 0 if the inputted program runs into an "infinite loop"?

Sometimes... yes:

```
while (2>1)
{
  print 1
}
```

never terminates, while

```
 print "Hello world!"
```

terminates.

Halting problem

Theorem: No algorithm exists that can decide whether a given program will halt or not.

Proof sketch:

Halting problem

Theorem: No algorithm exists that can decide whether a given program will halt or not.
Proof sketch: Suppose such an algorithm exists, that is, suppose there is a program Halt, which takes a program t as input, and has output $\mathrm{Halt}(t) = 1$ if the program t terminates and $\mathrm{Halt}(t) = 0$ if the program t does **not** terminate.

Halting problem

Theorem: No algorithm exists that can decide whether a given program will halt or not.
Proof sketch: Suppose such an algorithm exists, that is, suppose there is a program Halt, which takes a program t as input, and has output $\mathrm{Halt}(t) = 1$ if the program t terminates and $\mathrm{Halt}(t) = 0$ if the program t does **not** terminate.

Define a program as follows: the program $y(t)$ takes an input program t and asks "does t terminate or not?". If $\mathrm{Halt}(1) = 1$, then y decides to run an infinite loop. If $\mathrm{Halt}(t) = 0$, then y decides to terminate.

Halting problem

Theorem: No algorithm exists that can decide whether a given program will halt or not.
Proof sketch: Suppose such an algorithm exists, that is, suppose there is a program Halt, which takes a program t as input, and has output $\mathrm{Halt}(t) = 1$ if the program t terminates and $\mathrm{Halt}(t) = 0$ if the program t does **not** terminate.

Define a program as follows: the program $y(t)$ takes an input program t and asks "does t terminate or not?". If $\mathrm{Halt}(1) = 1$, then y decides to run an infinite loop. If $\mathrm{Halt}(t) = 0$, then y decides to terminate.

What happens if we feed the program y into itself? Does $y(y)$ terminate? If it does, then it doesn't. If it doesn't, then it does... therefore the Halt program does **not** exist!