01204211 Discrete Mathematics
Lecture 12a: Undecidability (1)

Jittat Fakcharoenphol

September 18, 2025

Turing machine

มี language L
ที่ไม่มีโปรแกรม
ภาษาไพธอน P ใดๆ
ที่ (1) รับ input
แล้วจบการทำงานเสมอ
(2) $\forall w \in L$, $P(w) =$ "yes"
(3) $\forall w \notin L$, $P(w) =$ "no"

มีบางปัญหา
ที่คอมพิวเตอร์
ไม่สามารถ
แก้ได้

language.

มีโปรแกรม
ภาษาไพธอน
ที่ แก้ได้ decides

Non-context-free languages

$S \to 0S1 \mid \varepsilon$

While

$$\{0^n 1^n \mid n \geq 0\}$$

is context free, the language

$$\{\underline{0^n 1^n} 0^n \mid n \geq 0\}$$

is not.

Non-context-free languages

While

$$\{0^n 1^n \mid n \geq 0\}$$

is context free, the language

$$\{0^n 1^n 0^n \mid n \geq 0\}$$

is not.
Can we write a python program to check if a string w belongs to the language $\{0^n 1^n 0^n \mid n \geq 0\}$?

Big question

Is there a python program that “solves” any possible problem?

Big question

Is there a python program that "solves" any possible problem?
Can a computer solve any problem?

Big question

Is there a python program that "solves" any possible problem?
Can a computer solve any problem?
Is there an algorithm that solves every problem?

Big question

Is there a python program that “solves” any possible problem?
Can a computer solve any problem?
Is there an algorithm that solves every problem?
What is the limit of computation?

Answer by a counting argument

If there are “more” problems than any possible algorithms, then there should be some problem that algorithms cannot solve.
(Think of an algorithm as “a python program.)

Answer by a counting argument

If there are “more” problems than any possible algorithms, then there should be some problem that algorithms cannot solve.
(Think of an algorithm as “a python program.)
However, there are infinitely many python programs and there are infinitely many problems. It is not obvious how to much such an argument formally.

Bijections

Definition

- A function $f : A \longrightarrow B$ from domain A to range B is **one-to-one** if for any $x \neq y \in A$, $f(x) \neq f(y)$.
- A function $f : A \longrightarrow B$ from domain A to range B is **onto** if for any $x' \in B$, there exists $x \in A$ such that $f(x) = x'$.
- A function $f : A \longrightarrow B$ is a **bijection** (or bijective) if it is one-to-one and onto.

Bijection: examples

Lemma 1

For any set A, there is no bijective function $f : A \longrightarrow 2^A$.

Proof.

We prove by contradiction. Assume that there exists a bijective function f from A to 2^A. We construct a set $B \subseteq A$ such that there is no $x \in A$ such that $f(x) = B$.

Lemma 1

For any set A, there is no bijective function $f : A \longrightarrow 2^A$.

Proof.

We prove by contradiction. Assume that there exists a bijective function f from A to 2^A. We construct a set $B \subseteq A$ such that there is no $x \in A$ such that $f(x) = B$.
We define B as follows.

$$B = \{x \in A \mid x \notin f(x)\}.$$

Now suppose that there exists $x \in A$ such that $f(x) = B$. There are two cases to consider:
Case 1: $x \in B$.

Lemma 1

For any set A, there is no bijective function $f : A \longrightarrow 2^A$.

Proof.

We prove by contradiction. Assume that there exists a bijective function f from A to 2^A. We construct a set $B \subseteq A$ such that there is no $x \in A$ such that $f(x) = B$.
We define B as follows.

$$B = \{x \in A \mid x \notin f(x)\}.$$

Now suppose that there exists $x \in A$ such that $f(x) = B$. There are two cases to consider:
Case 1: $x \in B$.
Case 2: $x \notin B$.

Lemma 1

For any set A, there is no bijective function $f : A \longrightarrow 2^A$.

Proof.

We prove by contradiction. Assume that there exists a bijective function f from A to 2^A. We construct a set $B \subseteq A$ such that there is no $x \in A$ such that $f(x) = B$.
We define B as follows.

$$B = \{x \in A \mid x \notin f(x)\}.$$

Now suppose that there exists $x \in A$ such that $f(x) = B$. There are two cases to consider:
Case 1: $x \in B$.
Case 2: $x \notin B$.
In both case, we have a contradiction; therefore, our assumption is false. Thus, there is no bijection between A and 2^A. $\square$

Example: finite set

Let $A = 1, 2, 3, 4, 5, 6, 7$. Consider function $f : A \longrightarrow 2^A$ defined as

$$\begin{aligned}
f(1) &= \{\} \\
f(2) &= \{1, 2, 3\} \\
f(3) &= \{1, 2, 3, 4, 5, 6, 7\} \\
f(4) &= \{1, 3, 5, 7\} \\
f(5) &= \{2, 4, 6\} \\
f(6) &= \{7\} \\
f(7) &= \{1, 2, 3\}
\end{aligned}$$

$B =$

Example: finite set

Let $A = 1, 2, 3, 4, 5, 6, 7$. Consider function $f : A \longrightarrow 2^A$ defined as

$$\begin{aligned} f(1) &= \{\} \\ f(2) &= \{1,2,3\} \\ f(3) &= \{1,2,3,4,5,6,7\} \\ f(4) &= \{1,3,5,7\} \\ f(5) &= \{2,4,6\} \\ f(6) &= \{7\} \\ f(7) &= \{1,2,3\} \end{aligned}$$

	1	2	3	4	5	6	7
1							
2							
3							
4							
5							
6							
7							

$B =$

Example: infinite set

Let $A = \mathbb{N} = \{1, 2, 3, 4, \ldots\}$. Consider function $f : A \longrightarrow 2^A$ defined as

$$
\begin{aligned}
f(1) &= \{\} \\
f(2) &= \{1, 2, 3\} \\
f(3) &= \{1, 2, 3, 4, 5, 6, 7, \ldots\} \\
f(4) &= \{1, 3, 5, 7, \ldots\} \\
f(5) &= \{2, 4, 6, \ldots\} \\
f(6) &= \{7\} \\
f(7) &= \{1, 2, 3, 11, 12, 13, 21, 22, 23, \ldots\} \\
&\vdots \quad\quad \vdots
\end{aligned}
$$

	1	2	3	4	5	6	7	···
1								
2								
3								
4								
5								
6								
7								
⋮								

$B =$

The previous lemma informally states that there are “more” subsets than the number of elements in the set.
Let’s think about:

- A set of all python programs, and
- A set of all languages.

The previous lemma informally states that there are “more” subsets than the number of elements in the set.
Let’s think about:

- A set of all python programs, and
- A set of all languages.

Since each python program “solves” at most one language, there are not “enough” python programs to solve all possible language.

The previous lemma informally states that there are "more" subsets than the number of elements in the set.
Let's think about:

- A set of all python programs, and
- A set of all languages.

Since each python program "solves" at most one language, there are not "enough" python programs to solve all possible language.
But what exactly is a problem that cannot be "solved"?

Decision problems

- Given an integer x, is x odd?
- Given a string w, is w palindrome?
- Given a string w, is $w \in \{0^n 1^n \mid n \geq 0\}$?
- Given a map, a starting position s, a destination t, and an integer k, does there exist a path from s to t with distance at most k?
- Given a program P and input string w, when running P with w as an input, does P terminate?

Decision problems and languages

For this problem:

Given an integer x, is x odd?

we can define a corresponding language

$$L_E = \{, \ldots, -6, -4, -2, 0, 2, 4, 6, \ldots\}.$$

To solve this problem, given x, we can ask if $x \in L_E$.

Languages and programs

We will talk about languages of particular programs. For example, let $\mathbb{P}$ be the set of all python programs. In this case, $\mathbb{P}$ is a language.

Languages and programs

We will talk about languages of particular programs. For example, let $\mathbb{P}$ be the set of all python programs. In this case, $\mathbb{P}$ is a language.

$$\{P \in \mathbb{P} \mid P \text{ always terminates}\}$$

Languages and programs

We will talk about languages of particular programs. For example, let $\mathbb{P}$ be the set of all python programs. In this case, $\mathbb{P}$ is a language.

$$\{P \in \mathbb{P} \mid P \text{ always terminates}\}$$

$$\{P \in \mathbb{P} \mid P \text{ always loops}\}$$

Languages and programs

We will talk about languages of particular programs. For example, let $\mathbb{P}$ be the set of all python programs. In this case, $\mathbb{P}$ is a language.

$$\{P \in \mathbb{P} \mid P \text{ always terminates}\}$$

$$\{P \in \mathbb{P} \mid P \text{ always loops}\}$$

$$\{(P, x) \mid P \in \mathbb{P}, \text{when running } P \text{ with } x \text{ as an input, } P \text{ terminates}\}$$

Languages and programs

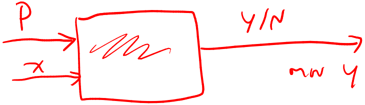
We will talk about languages of particular programs. For example, let $\mathbb{P}$ be the set of all python programs. In this case, $\mathbb{P}$ is a language.

$$\{P \in \mathbb{P} \mid P \text{ always terminates}\}$$

$$\{P \in \mathbb{P} \mid P \text{ always loops}\}$$

$$\{(P, x) \mid P \in \mathbb{P}, \text{when running } P \text{ with } x \text{ as an input, } P \text{ terminates}\}$$

$$L = \{(P, x) \mid P \in \mathbb{P}, \underbrace{P(x)} \text{ terminates}\}$$



ans Y iff P(x) terminates.

Languages and programs

We will talk about languages of particular programs. For example, let $\mathbb{P}$ be the set of all python programs. In this case, $\mathbb{P}$ is a language.

$$\{P \in \mathbb{P} \mid P \text{ always terminates}\}$$

$$\{P \in \mathbb{P} \mid P \text{ always loops}\}$$

$$\{(P, x) \mid P \in \mathbb{P}, \text{when running } P \text{ with } x \text{ as an input, } P \text{ terminates}\}$$

$$\{(P, x) \mid P \in \mathbb{P}, P(x) \text{ terminates}\}$$

$$\{(P, Q, x) \mid P, Q \in \mathbb{P}, P(x) \text{ and } Q(x) \text{ terminate with the same output.}\}$$

Programs and inputs

```
x = int(input())
if x % 2 == 0:
    print('yes')
else:
    print('no')
```

Programs and inputs

```
$ python le.py
10
yes
$ python le.py
7
no
```

```
x = int(input())
if x % 2 == 0:
    print('yes')
else:
    print('no')
```

Programs and inputs

```
x = int(input())
if x % 2 == 0:
    print('yes')
else:
    print('no')
```

```
$ python le.py
10
yes
$ python le.py
7
no
```

```
$ python le.py
fjdsklfjsdf
Traceback (most recent call last):
  File "le.py", line 1, in <module>
    x = int(input())
ValueError: invalid literal for int()
with base 10: 'fjdsklfjsdf'
```

Programs and inputs

x

./a.out < x

```
x = int(input())
if x % 2 == 0:
    print('yes')
else:
    print('no')
```

```
$ python le.py
10
yes
$ python le.py
7
no

$ python le.py
fjdsklfjsdf
Traceback (most recent call last):
  File "le.py", line 1, in <module>
    x = int(input())
ValueError: invalid literal for int()
with base 10: 'fjdsklfjsdf'

$ python le.py < le.py
```

Programs and inputs

```
x = int(input())
if x % 2 == 0:
    print('yes')
else:
    print('no')
```

```
$ python le.py
10
yes
$ python le.py
7
no
```

```
$ python le.py
fjdsklfjsdf
Traceback (most recent call last):
  File "le.py", line 1, in <module>
    x = int(input())
ValueError: invalid literal for int()
with base 10: 'fjdsklfjsdf'
```

```
$ python le.py < le.py
```

```
Traceback (most recent call last):
  File "le.py", line 1, in <module>
```

Nice programs

We can systematically modify any python program P so that

- P contains a main function that works with the input as a string.
- P never crashes. (If the original P crashes, the modified P outputs `no`.)

```
x = int(input())
if x % 2 == 0:
    print('yes')
else:
    print('no')
```

Nice programs

We can systematically modify any python program P so that

- P contains a main function that works with the input as a string.
- P never crashes. (If the original P crashes, the modified P outputs no.)

```
x = int(input())
if x % 2 == 0:
    print('yes')
else:
    print('no')
```

```
import sys
def main(w):
    try:
        x = int(w)
        if x % 2 == 0:
            print('yes')
        else:
            print('no')
    except:
        print('no')

if __name__ == '__main__':
    w = sys.stdin.read()
    main(w)
```

When running a program

When you run a program P with input x, there are three possible outcomes:

- P terminates and outputs **yes**,
- P terminates and outputs **no**, and
- P does not terminate. (It runs forever.)

When running a program

When you run a program P with input x, there are three possible outcomes:

- P terminates and outputs **yes**,
- P terminates and outputs **no**, and
- P does not terminate. (It runs forever.)

Remarks: if P crashes (even after modification), we treat it as if it terminates and outputs **no**.

Proving impossibility

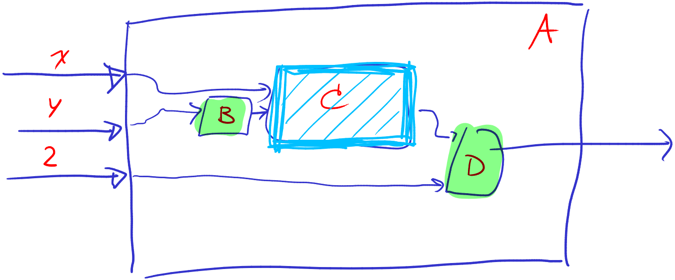
ทราบ
▶ ปัญหา R แก้ไม่ได้

Goal: ปัญหา (language) L แก้ไม่ได้

Proving impossibility

A solves language L

C solves language R



$$A(x,y,z) = D(C(x, B(y)), z)$$

- B ทำได้
- D ทำได้
- C ไม่รู้

สร้าง A

① ถ้าปัญหา R แก้ได้

⇒ ปัญหา L แก้ได้ด้วย

② L แก้ไม่ได้

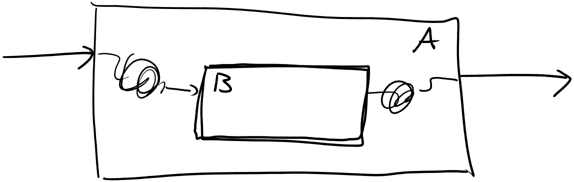
$P \Rightarrow Q$ ✓

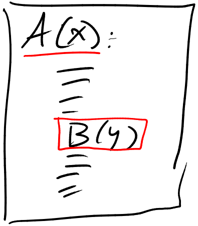
$\neg Q$ ✓

Reduction: rough idea

A แก้ปัญหา L

B แก้ปัญหา R





→ ถ้าแก้ปัญหา R ได้
⇒ แก้ปัญหา L ได้

ลดรูปจาก L ไป R

ทำ reduction จาก L ไป R

Language A

Let $\mathbb{P}$ be the set of all python programs. Let the language A be

$$\{P \in \mathbb{P} \mid \text{when running } P \text{ with } P \text{ as an input, } P \text{ terminates}\}$$

Language A

Let $\mathbb{P}$ be the set of all python programs. Let the language A be

$$\{P \in \mathbb{P} \mid \text{when running } P \text{ with } P \text{ as an input, } P \text{ terminates}\}$$

We use a function call notation $P(x)$ when refering to the execution of program P with input x.

Language A

Let $\mathbb{P}$ be the set of all python programs. Let the language A be

$$\{P \in \mathbb{P} \mid \text{when running } P \text{ with } P \text{ as an input, } P \text{ terminates}\}$$

We use a function call notation $P(x)$ when refering to the execution of program P with input x.
We restate the definition of A as

$$\{P \in \mathbb{P} \mid P(P) \text{ terminates}\}.$$

Deciders

We say that a python program P **decides** the language L if for any input string x, P when running with x as an input,

- P always terminates,
- P outputs **yes**, if $x \in L$, and
- P outputs **no**, if $x \notin L$.

Deciders: more examples

Language A

Let $\mathbb{P}$ be the set of all python programs. Let the language A be

$$\{P \in \mathbb{P} \mid \text{when running } P \text{ with } P \text{ as an input, } P \text{ terminates}\}$$

Language A

Let $\mathbb{P}$ be the set of all python programs. Let the language A be

$$\{P \in \mathbb{P} \mid \text{when running } P \text{ with } P \text{ as an input, } P \text{ terminates}\}$$

We use a function call notation $P(x)$ when refering to the execution of program P with input x.

Language A

Let $\mathbb{P}$ be the set of all python programs. Let the language A be

$$\{P \in \mathbb{P} \mid \text{when running } P \text{ with } P \text{ as an input, } P \text{ terminates}\}$$

We use a function call notation $P(x)$ when refering to the execution of program P with input x.
We restate the definition of A as

$$\{P \in \mathbb{P} \mid P(P) \text{ terminates}\}.$$

Not a decider for A

```
Input: python program P (as a string)

1. Load module P as Pmod
2. Call Pmod.main(P)
3. print('yes')                  # we reach this line,
                                 #  only if M.main(P) terminates
```

Lemma 2

There is no python program that decides A.

Lemma 2

There is no python program that decides A.

We will see the proof at the end of class.

Undecidability

If we believe that anything that a computer can do can be written as a python program,

Undecidability

If we believe that anything that a computer can do can be written as a python program, and there is no python program that decides A, when we say that

A is undecidable.

Language A will be very important later on, we give it a proper name as $\textsc{SelfHalt}$.

The proof as a table

List all python programs in $\mathbb{P}$ as $P_1, P_2, P_3, \ldots$.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	$\ldots$
P_1								
P_2								
P_3								
P_4								
P_5								
P_6								
$\vdots$								
(B)								

What does B do on each input program P_i?

Another language $\textsc{Halt}$

Let

$$\textsc{Halt} = \{(P, w) \mid P \text{ is a python program such that } P(w) \text{ terminates}\}$$

We shall prove that $\textsc{Halt}$ is also undecidable (if we believe that python programs represent all possible computation).

Lemma 3

HALT *is undecidable.*

Proof.

We prove the lemma by contradiction. Assume that there is a python program P that decides HALT.

Lemma 3

HALT *is undecidable.*

Proof.

We prove the lemma by contradiction. Assume that there is a python program P that decides HALT.
We construct a program C as follows

```
Program C
Input Q
1.    Load P as module Pmod
2.    if Pmod.main(Q,Q) == 'yes':
3.        print('yes')
4.    else
5.        print('no')
```

□

Proof.

We prove the lemma by contradiction. Assume that there is a python program P that decides $\textsc{Halt}$.
We construct a program C as follows

```
Program C
Input Q
1.   if P(Q,Q) == 'yes':
2.      print('yes')
3.   else
4.      print('no')
```

Proof.

We prove the lemma by contradiction. Assume that there is a python program P that decides Halt.
We construct a program C as follows

```
Program C
Input Q
    if P(Q,Q) == 'yes':
        print('yes')
    else
        print('no')
```

Given program P, we can construct a program C that decides SelfHalt.

Proof.

We prove the lemma by contradiction. Assume that there is a python program P that decides HALT.
We construct a program C as follows

```
Program C
Input Q
1.   if P(Q,Q) == 'yes':
2.      print('yes')
3.   else
4.      print('no')
```

Given program P, we can construct a program C that decides SELFHALT. However, we know that SELFHALT is undecidable; thus, we reach a contradiction.
We conclude that there does not exist a python program P that decides HALT. $\square$

Reduction

Reduction

- We show that if HALT is decidable, then SELFHALT is also decidable.

Reduction

- We show that if HALT is decidable, then SELFHALT is also decidable.
- However, SELFHALT IS UNDECIDABLE.

Reduction

- We show that if HALT is decidable, then SELFHALT is also decidable.
- However, SELFHALT IS UNDECIDABLE.
- We conclude that HALT is also undecidable.

Reduction in picture

Let $\textsc{Accept} = \{(P, w) \mid P \in \mathbb{P}$ and $P(w)$ terminates with acceptance$\}$.

Lemma 4

$\textsc{Accept}$ *is undecidable.*

Proof.

We prove the lemma by contradiction. Assume that there is a python program Q that decides $\textsc{Accept}$.

Let $\textsc{Accept} = \{(P, w) \mid P \in \mathbb{P}$ and $P(w)$ terminates with acceptance$\}$.

Lemma 4

$\textsc{Accept}$ *is undecidable.*

Proof.

We prove the lemma by contradiction. Assume that there is a python program Q that decides $\textsc{Accept}$. We construct a program C that decides $\textsc{Halt}$ as follows

Let $\textsc{Accept} = \{(P, w) \mid P \in \mathbb{P} \text{ and } P(w) \text{ terminates with acceptance}\}$.

Lemma 4

$\textsc{Accept}$ *is undecidable.*

Proof.

We prove the lemma by contradiction. Assume that there is a python program Q that decides $\textsc{Accept}$. We construct a program C that decides $\textsc{Halt}$ as follows

```
Program C
Input P,w
1.   Replace every "print('no')" statement in P with "print('yes')"
1.   if Q(P,w) == 'yes':
2.      print('yes')
3.   else
4.      print('no')
```

Proof (cont.)

```
Program C
Input P,w
1.   Replace every "print('no')" statement in P with "print('yes')"
1.   if Q(P,w) == 'yes':
2.      print('yes')
3.   else
4.      print('no')
```

We have to make sure that our reduction is correct by considering two cases.
Case 1: when $P(w)$ halts.

Proof (cont.)

```
Program C
Input P,w
1.   Replace every "print('no')" statement in P with "print('yes')"
1.   if Q(P,w) == 'yes':
2.      print('yes')
3.   else
4.      print('no')
```

We have to make sure that our reduction is correct by considering two cases.
Case 1: when $P(w)$ halts.
Case 2: when $P(w)$ does not halt.

Proof (cont.)

```
Program C
Input P,w
1.   Replace every "print('no')" statement in P with "print('yes')"
1.   if Q(P,w) == 'yes':
2.      print('yes')
3.   else
4.      print('no')
```

We have to make sure that our reduction is correct by considering two cases.
Case 1: when $P(w)$ halts.
Case 2: when $P(w)$ does not halt.
Since in both cases, C answers correctly, we know that given program Q deciding $\textsc{Accept}$, we can construct a program C that decides $\textsc{Halt}$. However, we know that $\textsc{Halt}$ is undecidable; thus, we reach a contradiction. We conclude that $\textsc{Accept}$ is also undecidable. $\square$

Reduction from HALT to ACCEPT in picture

How about $\textsc{Reject}$?

Let

$$\textsc{Reject} = \{(P, w) \mid P \in \mathbb{P} \text{ and } P \text{ rejects } w\}.$$

Lemma 5

There is no python program that decides $\textsc{SelfHalt}$.

Proof.

We prove by contradiction. Assume that there is a python program P that decides $\textsc{SelfHalt}$.

Lemma 5

There is no python program that decides $\textsc{SelfHalt}$.

Proof.

We prove by contradiction. Assume that there is a python program P that decides $\textsc{SelfHalt}$.

We describe a python program B that reads a string Q as an input as follows:

```
Program B
Input Q
1.     Load P as module Pmod
2.     if Pmod.main(Q) == 'yes':     # when Pmod outputs yes
3.        while True: pass           #   loop forever
4.     else:                         # when Pmod outputs no
5.        quit()                     #   halt
```

Given program Q as an input, B loops forever when

Lemma 5

There is no python program that decides SELFHALT.

Proof.

We prove by contradiction. Assume that there is a python program P that decides SELFHALT.

We describe a python program B that reads a string Q as an input as follows:

```
Program B
Input Q
1.    Load P as module Pmod
2.    if Pmod.main(Q) == 'yes':      # when Pmod outputs yes
3.       while True: pass            #   loop forever
4.    else:                          # when Pmod outputs no
5.       quit()                      #   halt
```

Given program Q as an input, B loops forever when

It terminates when

□

Proof.

We know that

- $B(Q)$ loops when $Q(Q)$ terminates, and
- $B(Q)$ terminates when $Q(Q)$ loops.

Does running B using B as an input terminate?

Proof.

We know that

- $B(Q)$ loops when $Q(Q)$ terminates, and
- $B(Q)$ terminates when $Q(Q)$ loops.

Does running B using B as an input terminate?
Let's try to plug in $Q = B$. We have

- $B(B)$ loops when

Proof.

We know that

- $B(Q)$ loops when $Q(Q)$ terminates, and
- $B(Q)$ terminates when $Q(Q)$ loops.

Does running B using B as an input terminate?
Let's try to plug in $Q = B$. We have

- $B(B)$ loops when $B(B)$ terminates,

Proof.

We know that

- $B(Q)$ loops when $Q(Q)$ terminates, and
- $B(Q)$ terminates when $Q(Q)$ loops.

Does running B using B as an input terminate?
Let's try to plug in $Q = B$. We have

- $B(B)$ loops when $B(B)$ terminates, and
- $B(B)$ terminates when

Proof.

We know that

- $B(Q)$ loops when $Q(Q)$ terminates, and
- $B(Q)$ terminates when $Q(Q)$ loops.

Does running B using B as an input terminate?
Let's try to plug in $Q = B$. We have

- $B(B)$ loops when $B(B)$ terminates, and
- $B(B)$ terminates when $B(B)$ loops.

Proof.

We know that

- $B(Q)$ loops when $Q(Q)$ terminates, and
- $B(Q)$ terminates when $Q(Q)$ loops.

Does running B using B as an input terminate?
Let's try to plug in $Q = B$. We have

- $B(B)$ loops when $B(B)$ terminates, and
- $B(B)$ terminates when $B(B)$ loops.

Since either $B(B)$ loops or terminates, and we cannot be in any of the cases, we obtain a contradiction.
Therefore, we conclude that program P does not exist. $\square$

Python as computation

Do you believe in this assumption:

Anything that a computer can do can be written as a python program.

Turing machines

Anything that a computer can do can be carried out using Turing machines.

Turing machines

Anything that a computer can do can be carried out using Turing machines.

Any possible computation can be performed by Turing machines.