COMS11700

Undecidability

Ashley Montanaro
ashley@cs.bris.ac.uk

Department of Computer Science, University of Bristol
Bristol, UK

4 April 2014

Decidability

We are particularly interested in Turing machines which halt on all inputs. Such a machine is called a decider.

Decidability

We are particularly interested in Turing machines which halt on all inputs. Such a machine is called a decider.

- We say that M decides a language $\mathcal{L}$ if M is a decider and M recognises $\mathcal{L}$.
- $\mathcal{L}$ is said to be decidable if some Turing machine decides it.
- Otherwise, $\mathcal{L}$ is said to be undecidable.

For example, we have seen already that the language

$$\mathcal{L}_{EQ} = \{w\#w \mid w \in \{0,1\}^*\}.$$

is decidable.

Undecidable problems

- Is it the case that the Turing machine model can compute anything?
- More specifically, is every language $\mathcal{L}$ decidable?

Undecidable problems

- Is it the case that the Turing machine model can compute anything?
- More specifically, is every language $\mathcal{L}$ decidable?
- We will see that the answer is unfortunately (?) no.
- Assuming that the Church-Turing thesis is true, this implies that there are problems which we cannot solve by any mechanical means!

An undecidable problem

Let $\mathcal{L}_U$ be the following language:

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

- That is, $\mathcal{L}_U$ is the language of (descriptions of) Turing machines that do not accept when given their own descriptions as input.
- This means that on this input they either reject, or run forever.

An undecidable problem

Let $\mathcal{L}_U$ be the following language:

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

- That is, $\mathcal{L}_U$ is the language of (descriptions of) Turing machines that do not accept when given their own descriptions as input.
- This means that on this input they either reject, or run forever.

Lemma

$\mathcal{L}_U$ is undecidable.

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

Claim: $\mathcal{L}_U$ is undecidable.

Proof

The proof is by contradiction.

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

Claim: $\mathcal{L}_U$ is undecidable.

Proof

The proof is by contradiction.

- Assume there is a Turing machine N which decides $\mathcal{L}_U$.

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

Claim: $\mathcal{L}_U$ is undecidable.

Proof

The proof is by contradiction.

- Assume there is a Turing machine N which decides $\mathcal{L}_U$.
- Then, for all $y \in \{0,1\}^*$, N accepts if and only if $y \in \mathcal{L}_U$.

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

Claim: $\mathcal{L}_U$ is undecidable.

Proof

The proof is by contradiction.

- Assume there is a Turing machine N which decides $\mathcal{L}_U$.
- Then, for all $y \in \{0,1\}^*$, N accepts if and only if $y \in \mathcal{L}_U$.
- In particular, N accepts $\langle N \rangle$ if and only if $\langle N \rangle \in \mathcal{L}_U$.

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

Claim: $\mathcal{L}_U$ is undecidable.

Proof

The proof is by contradiction.

- Assume there is a Turing machine N which decides $\mathcal{L}_U$.
- Then, for all $y \in \{0,1\}^*$, N accepts if and only if $y \in \mathcal{L}_U$.
- In particular, N accepts $\langle N \rangle$ if and only if $\langle N \rangle \in \mathcal{L}_U$.
- But by the definition of $\mathcal{L}_U$, $\langle N \rangle \in \mathcal{L}_U$ if and only if N does not accept $\langle N \rangle$. Contradiction.

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

Claim: $\mathcal{L}_U$ is undecidable.

Proof

The proof is by contradiction.

- Assume there is a Turing machine N which decides $\mathcal{L}_U$.
- Then, for all $y \in \{0,1\}^*$, N accepts if and only if $y \in \mathcal{L}_U$.
- In particular, N accepts $\langle N \rangle$ if and only if $\langle N \rangle \in \mathcal{L}_U$.
- But by the definition of $\mathcal{L}_U$, $\langle N \rangle \in \mathcal{L}_U$ if and only if N does not accept $\langle N \rangle$. Contradiction.

Intuition: The barber paradox

A man from Seville is shaved by the Barber of Seville if and only if he does not shave himself. Does the barber shave himself?

Diagonalisation

Another way to view this argument is as follows.

We write down an infinite table M whose rows and columns are indexed by bit-strings $x, y \in \{0,1\}^*$. Rows represent TMs, columns represent inputs.

	0	1	00	01	...	y
0	0	1	0	0		
1	0	1	0	1		
00	1	1	1	0		
01	0	1	0	0		
⋮					⋱	
x						

- We fill in entry (x, y) of this table with 1 if the TM with description x accepts input y, and 0 otherwise.

Diagonalisation

	0	1	00	01	...	y
0	0	1	0	0		
1	0	1	0	1		
00	1	1	1	0		
01	0	1	0	0		
⋮					⋱	
x						

- Now consider the bit-string $u \in \{0,1\}^*$ whose i'th bit is equal to the negation of the i'th entry on the diagonal of M (so here u would start 1001 . . .)

Diagonalisation

	0	1	00	01	...	y
0	0	1	0	0		
1	0	1	0	1		
00	1	1	1	0		
01	0	1	0	0		
⋮					⋱	
x						

- Now consider the bit-string $u \in \{0,1\}^*$ whose i'th bit is equal to the negation of the i'th entry on the diagonal of M (so here u would start 1001 . . .)
- u differs from the first row of M in the first position, the second row in the second position, . . .

Diagonalisation

	0	1	00	01	...	y
0	0	1	0	0		
1	0	1	0	1		
00	1	1	1	0		
01	0	1	0	0		
⋮					⋱	
x						

- Now consider the bit-string $u \in \{0,1\}^*$ whose i'th bit is equal to the negation of the i'th entry on the diagonal of M (so here u would start 1001 . . .)
- u differs from the first row of M in the first position, the second row in the second position, . . .
- So u is not equal to any of the rows of M.

Diagonalisation

	0	1	00	01	...	y
0	0	1	0	0		
1	0	1	0	1		
00	1	1	1	0		
01	0	1	0	0		
$\vdots$					$\ddots$	
x						

- Now consider the bit-string $u \in \{0,1\}^*$ whose i'th bit is equal to the negation of the i'th entry on the diagonal of M (so here u would start 1001 . . .)
- u differs from the first row of M in the first position, the second row in the second position, . . .
- So u is not equal to any of the rows of M.
- So there is no TM which accepts the language of strings y such that $u_y = 1$.

The halting problem

We have shown that the language

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

is undecidable. But what if we don't care about deciding this language?

The halting problem

We have shown that the language

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

is undecidable. But what if we don't care about deciding this language?

- It turns out that many decision problems which we care about in practice are also undecidable.
- A classic example is the halting problem: given a program, and an input, does it terminate on that input?

The halting problem

We have shown that the language

$$\mathcal{L}_U = \{x \in \{0,1\}^* \mid x = \langle M \rangle, \text{ where } M \text{ is a TM that does not accept } x\}$$

is undecidable. But what if we don't care about deciding this language?

- It turns out that many decision problems which we care about in practice are also undecidable.
- A classic example is the halting problem: given a program, and an input, does it terminate on that input?
- Put another way, given a Turing machine M and an input x, does M halt on input x? More formally, the language

$$\mathcal{L}_{\text{HALT}} = \{\langle M, x \rangle \mid M \text{ is a TM that halts on input } x\}$$

The halting problem

Claim

$\mathcal{L}_{\text{HALT}}$ is undecidable.

Proof

- Suppose there exists a TM M deciding $\mathcal{L}_{\text{HALT}}$.

The halting problem

Claim

$\mathcal{L}_{\text{HALT}}$ is undecidable.

Proof

- Suppose there exists a TM M deciding $\mathcal{L}_{\text{HALT}}$.
- We will show there exists a machine M' which decides $\mathcal{L}_U$.

The halting problem

Claim

$\mathcal{L}_{\text{HALT}}$ is undecidable.

Proof

- Suppose there exists a TM M deciding $\mathcal{L}_{\text{HALT}}$.
- We will show there exists a machine M' which decides $\mathcal{L}_U$.
- On input $x = \langle N \rangle$, M' simulates M to determine whether N halts on input $\langle N \rangle$.

The halting problem

Claim

$\mathcal{L}_{\text{HALT}}$ is undecidable.

Proof

- Suppose there exists a TM M deciding $\mathcal{L}_{\text{HALT}}$.
- We will show there exists a machine M' which decides $\mathcal{L}_U$.
- On input $x = \langle N \rangle$, M' simulates M to determine whether N halts on input $\langle N \rangle$.
- If M says "no", M' accepts.

The halting problem

Claim

$\mathcal{L}_{\text{HALT}}$ is undecidable.

Proof

- Suppose there exists a TM M deciding $\mathcal{L}_{\text{HALT}}$.
- We will show there exists a machine M' which decides $\mathcal{L}_U$.
- On input $x = \langle N \rangle$, M' simulates M to determine whether N halts on input $\langle N \rangle$.
- If M says "no", M' accepts.
- Otherwise, M' simulates N on input $\langle N \rangle$ and accepts if and only if N rejects (this can be done in finite time because we know that N halts).

The halting problem

Claim

$\mathcal{L}_{\text{HALT}}$ is undecidable.

Proof

- Suppose there exists a TM M deciding $\mathcal{L}_{\text{HALT}}$.
- We will show there exists a machine M' which decides $\mathcal{L}_U$.
- On input $x = \langle N \rangle$, M' simulates M to determine whether N halts on input $\langle N \rangle$.
- If M says "no", M' accepts.
- Otherwise, M' simulates N on input $\langle N \rangle$ and accepts if and only if N rejects (this can be done in finite time because we know that N halts).
- Thus M' can decide $\mathcal{L}_U$. Contradiction!

The halting problem in practice

Informally, we have shown that it is impossible to determine whether a program terminates. Is this an important problem?

- For some very simple programs, we don't know whether they halt:

```
int f(int n) {
    if (n <= 1) return 0;
    else if (even(n)) return f(n/2);
    else return f(3*n+1);
}
```

- Determining whether `f(n)` terminates for all `n` is known as the Collatz conjecture and has been an open problem for over 70 years!

- Tools exist to solve special cases of the halting problem (e.g. Microsoft Terminator project).

Reductions

This proof illustrates a key concept in proving hardness: reductions.

A reduction from $\mathcal{A}$ to $\mathcal{B}$ is a computable function $f(x)$ such that $x \in \mathcal{A}$ if and only if $f(x) \in \mathcal{B}$.

Reductions

This proof illustrates a key concept in proving hardness: reductions.

A reduction from $\mathcal{A}$ to $\mathcal{B}$ is a computable function $f(x)$ such that $x \in \mathcal{A}$ if and only if $f(x) \in \mathcal{B}$.

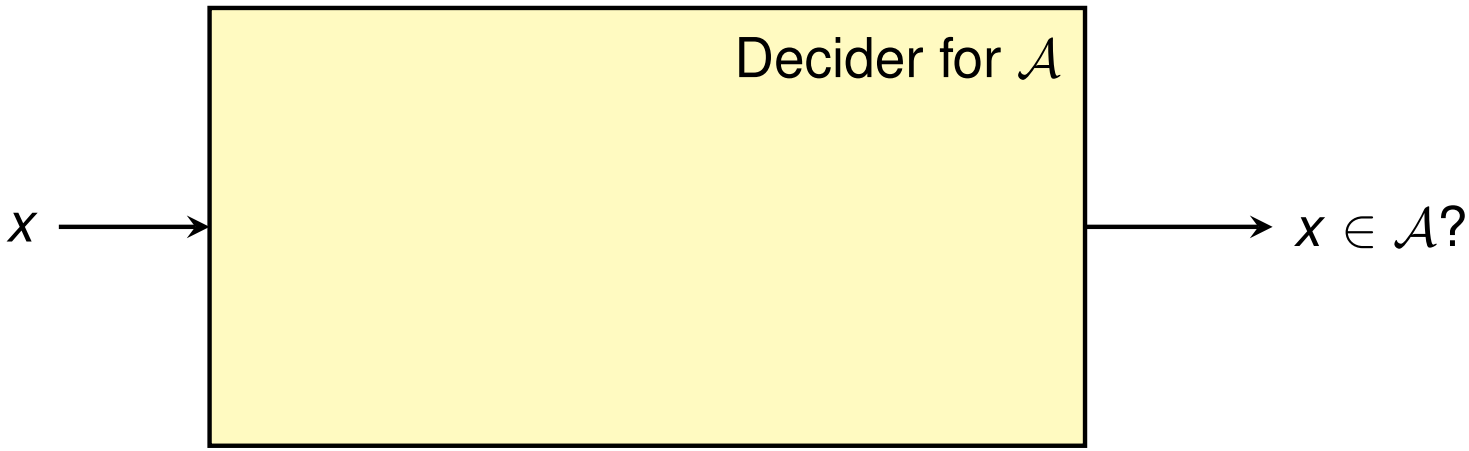
- Imagine we know that language $\mathcal{A}$ is undecidable, and we want to prove that language $\mathcal{B}$ is undecidable.
- We can achieve this by showing that, if we could decide $\mathcal{B}$, we could decide $\mathcal{A}$.
- Therefore, $\mathcal{B}$ must be undecidable!

Reductions

This proof illustrates a key concept in proving hardness: reductions.

A reduction from $\mathcal{A}$ to $\mathcal{B}$ is a computable function $f(x)$ such that $x \in \mathcal{A}$ if and only if $f(x) \in \mathcal{B}$.

- Imagine we know that language $\mathcal{A}$ is undecidable, and we want to prove that language $\mathcal{B}$ is undecidable.
- We can achieve this by showing that, if we could decide $\mathcal{B}$, we could decide $\mathcal{A}$.
- Therefore, $\mathcal{B}$ must be undecidable!

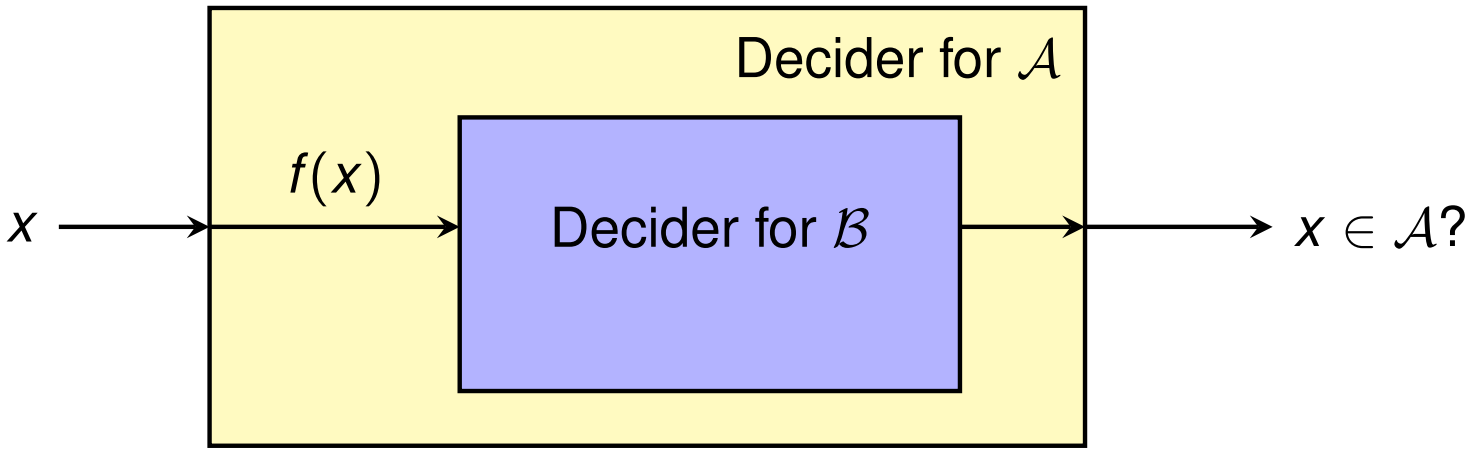


Reductions

This proof illustrates a key concept in proving hardness: reductions.

A reduction from $\mathcal{A}$ to $\mathcal{B}$ is a computable function $f(x)$ such that $x \in \mathcal{A}$ if and only if $f(x) \in \mathcal{B}$.

- Imagine we know that language $\mathcal{A}$ is undecidable, and we want to prove that language $\mathcal{B}$ is undecidable.
- We can achieve this by showing that, if we could decide $\mathcal{B}$, we could decide $\mathcal{A}$.
- Therefore, $\mathcal{B}$ must be undecidable!



Reductions: another example

Problem

Show that the following language is undecidable.

$$\mathcal{L}_1 = \{\langle M \rangle \mid M \text{ accepts } 1\}$$

Solution

We give a reduction from $\mathcal{L}_{\text{HALT}}$ to $\mathcal{L}_1$.

Reductions: another example

Problem

Show that the following language is undecidable.

$$\mathcal{L}_1 = \{\langle M \rangle \mid M \text{ accepts } 1\}$$

Solution

We give a reduction from $\mathcal{L}_{\text{HALT}}$ to $\mathcal{L}_1$.

- Given $\langle M, x \rangle$, the reduction outputs the description of a TM N.

Reductions: another example

Problem

Show that the following language is undecidable.

$$\mathcal{L}_1 = \{\langle M \rangle \mid M \text{ accepts } \texttt{1}\}$$

Solution

We give a reduction from $\mathcal{L}_{\text{HALT}}$ to $\mathcal{L}_1$.

- Given $\langle M, x \rangle$, the reduction outputs the description of a TM N.
- N begins by checking whether the input is equal to $\texttt{1}$. If not, it rejects. Otherwise, it writes x to the tape and then simulates the operation of M. If M halts, N accepts.

Reductions: another example

Problem

Show that the following language is undecidable.

$$\mathcal{L}_1 = \{\langle M \rangle \mid M \text{ accepts } \texttt{1}\}$$

Solution

We give a reduction from $\mathcal{L}_{\text{HALT}}$ to $\mathcal{L}_1$.

- Given $\langle M, x \rangle$, the reduction outputs the description of a TM N.
- N begins by checking whether the input is equal to $\texttt{1}$. If not, it rejects. Otherwise, it writes x to the tape and then simulates the operation of M. If M halts, N accepts.
- Thus N accepts $\texttt{1}$ if and only if M halts on input x.

Reductions: another example

Problem

Show that the following language is undecidable.

$$\mathcal{L}_1 = \{\langle M \rangle \mid M \text{ accepts } \texttt{1}\}$$

Solution

We give a reduction from $\mathcal{L}_{\text{HALT}}$ to $\mathcal{L}_1$.

- Given $\langle M, x \rangle$, the reduction outputs the description of a TM N.
- N begins by checking whether the input is equal to $\texttt{1}$. If not, it rejects. Otherwise, it writes x to the tape and then simulates the operation of M. If M halts, N accepts.
- Thus N accepts $\texttt{1}$ if and only if M halts on input x.
- So, if we can decide the language $\mathcal{L}_1$, we can decide $\mathcal{L}_{\text{HALT}}$.

Reductions: another example

Problem

Show that the following language is undecidable.

$$\mathcal{L}_1 = \{\langle M \rangle \mid M \text{ accepts } \texttt{1}\}$$

Solution

We give a reduction from $\mathcal{L}_{\text{HALT}}$ to $\mathcal{L}_1$.

- Given $\langle M, x \rangle$, the reduction outputs the description of a TM N.
- N begins by checking whether the input is equal to $\texttt{1}$. If not, it rejects. Otherwise, it writes x to the tape and then simulates the operation of M. If M halts, N accepts.
- Thus N accepts $\texttt{1}$ if and only if M halts on input x.
- So, if we can decide the language $\mathcal{L}_1$, we can decide $\mathcal{L}_{\text{HALT}}$.
- So $\mathcal{L}_1$ is undecidable!

Countability

There is a connection between diagonalisation and a beautiful argument used to show that the real numbers $\mathbb{R}$ are not countable.

- A set is said to be countable if it has the same cardinality (size) as some subset of $\mathbb{N} = \{1, 2, 3, \dots\}$.
- This means that we can associate each element in the set with a unique natural number – its "position" in the set.

Countability

There is a connection between diagonalisation and a beautiful argument used to show that the real numbers $\mathbb{R}$ are not countable.

- A set is said to be countable if it has the same cardinality (size) as some subset of $\mathbb{N} = \{1, 2, 3, \dots\}$.
- This means that we can associate each element in the set with a unique natural number – its "position" in the set.
- For example, $\{\texttt{red}, \texttt{green}, \texttt{blue}\}$ is countable because we can assign

$$\texttt{red} \mapsto 1, \quad \texttt{green} \mapsto 2, \quad \texttt{blue} \mapsto 3.$$

Countability

There is a connection between diagonalisation and a beautiful argument used to show that the real numbers $\mathbb{R}$ are not countable.

- A set is said to be countable if it has the same cardinality (size) as some subset of $\mathbb{N} = \{1, 2, 3, \dots\}$.
- This means that we can associate each element in the set with a unique natural number – its "position" in the set.
- For example, $\{\texttt{red}, \texttt{green}, \texttt{blue}\}$ is countable because we can assign

$$\texttt{red} \mapsto 1, \quad \texttt{green} \mapsto 2, \quad \texttt{blue} \mapsto 3.$$

- Any finite set is countable, but the question of whether infinite sets are countable is more interesting.
- For example, consider the integers $\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$

Countability

- There are intuitively “twice as many” integers as natural numbers.
- But the integers are nevertheless a countable set!

Countability

- There are intuitively “twice as many” integers as natural numbers.
- But the integers are nevertheless a countable set!
- Write them in the order $\mathbb{Z} = \{0, 1, -1, 2, -2, 3, -3, \dots\}$.

Countability

- There are intuitively "twice as many" integers as natural numbers.
- But the integers are nevertheless a countable set!
- Write them in the order $\mathbb{Z} = \{0, 1, -1, 2, -2, 3, -3, \dots\}$.
- Then we can associate each $z \in \mathbb{Z}$ with the natural number

$$\phi(z) = \begin{cases} 2z - 1 & \text{if } z > 0 \\ -2z & \text{if } z \leq 0 \end{cases}$$

- This is a one-to-one mapping (check this!), so $\mathbb{Z}$ is countable.

Countability

- There are intuitively "twice as many" integers as natural numbers.
- But the integers are nevertheless a countable set!
- Write them in the order $\mathbb{Z} = \{0, 1, -1, 2, -2, 3, -3, \dots\}$.
- Then we can associate each $z \in \mathbb{Z}$ with the natural number

$$\phi(z) = \begin{cases} 2z - 1 & \text{if } z > 0 \\ -2z & \text{if } z \leq 0 \end{cases}$$

- This is a one-to-one mapping (check this!), so $\mathbb{Z}$ is countable.
- What about the real numbers $\mathbb{R}$?

The real numbers are not countable

We can prove that $\mathbb{R}$ is not countable using the same diagonalisation argument as before.

The real numbers are not countable

We can prove that $\mathbb{R}$ is not countable using the same diagonalisation argument as before.

- Assume that $\mathbb{R}$ is countable. Then we can associate each $x \in \mathbb{R}$ with a unique natural number $\phi(x) \in \mathbb{N}$.
- Write down a table whose n'th row is the digits of the real number corresponding to $n \in \mathbb{N}$, for example:

0	2.	7	8	1	8	...
1	13.	0	0	0	0	...
2	0.	1	1	9	3	...
3	6.	6	6	6	6	...
⋮					⋱	
n						

The real numbers are not countable

For each row i, we take the i'th digit after the decimal point and add 1 to it:

0	2.	~~7~~ 8	8	1	8	...
1	13.	0	~~0~~ 1	0	0	...
2	0.	1	1	~~9~~ 0	3	...
3	6.	6	6	6	~~6~~ 7	...
$\vdots$					$\ddots$	
n						

The real numbers are not countable

For each row i, we take the i'th digit after the decimal point and add 1 to it:

0	2.	~~7~~ 8	8	1	8	...
1	13.	0	~~0~~ 1	0	0	...
2	0.	1	1	~~9~~ 0	3	...
3	6.	6	6	6	~~6~~ 7	...
$\vdots$					$\ddots$	
n						

- Consider the real number x formed by making the i'th digit after the decimal point equal to the resulting number.

The real numbers are not countable

For each row i, we take the i'th digit after the decimal point and add 1 to it:

0	2.	~~7~~ 8	8	1	8	...
1	13.	0	~~0~~ 1	0	0	...
2	0.	1	1	~~9~~ 0	3	...
3	6.	6	6	6	~~6~~ 7	...
$\vdots$					$\ddots$	
n						

- Consider the real number x formed by making the i'th digit after the decimal point equal to the resulting number.
- For each i, the i'th digit of x differs from the real number in row i.

The real numbers are not countable

For each row i, we take the i'th digit after the decimal point and add 1 to it:

0	2.	~~7~~ 8	8	1	8	...
1	13.	0	~~0~~ 1	0	0	...
2	0.	1	1	~~9~~ 0	3	...
3	6.	6	6	6	~~6~~ 7	...
$\vdots$					$\ddots$	
n						

- Consider the real number x formed by making the i'th digit after the decimal point equal to the resulting number.
- For each i, the i'th digit of x differs from the real number in row i.
- So x cannot appear in the table.

The real numbers are not countable

For each row i, we take the i'th digit after the decimal point and add 1 to it:

0	2.	~~7~~ 8	8	1	8	...
1	13.	0	~~0~~ 1	0	0	...
2	0.	1	1	~~9~~ 0	3	...
3	6.	6	6	6	~~6~~ 7	...
$\vdots$					$\ddots$	
n						

- Consider the real number x formed by making the i'th digit after the decimal point equal to the resulting number.
- For each i, the i'th digit of x differs from the real number in row i.
- So x cannot appear in the table.
- So $\mathbb{R}$ is not countable!

Decidability of proofs

The Turing machine model has another connection to the foundations of mathematics itself.

- Imagine we would like to prove logical statements about the natural numbers, like Fermat's Last Theorem:

$$\forall a, b, c, n \in \mathbb{N}\ [(a, b, c > 0 \wedge n > 2) \Rightarrow a^n + b^n \neq c^n]$$

- The Entscheidungsproblem ("decision problem") of Hilbert asked whether there was a mechanical procedure to prove such claims.

Decidability of proofs

The Turing machine model has another connection to the foundations of mathematics itself.

- Imagine we would like to prove logical statements about the natural numbers, like Fermat's Last Theorem:

$$\forall a, b, c, n \in \mathbb{N}\ [(a, b, c > 0 \wedge n > 2) \Rightarrow a^n + b^n \neq c^n]$$

- The Entscheidungsproblem ("decision problem") of Hilbert asked whether there was a mechanical procedure to prove such claims.

Claim (informal, see Sipser §6.2)

Let $\mathcal{L}_{\text{MATHS}}$ be the language of true mathematical statements about the natural numbers. Then $\mathcal{L}_{\text{MATHS}}$ is undecidable.

The basic idea: encode the operation of a Turing machine in terms of constraints on some numbers, and write down a logical statement about these numbers which is true if and only if the machine accepts its input.

Other undecidable problems

There are a vast number of interesting problems which turn out to be undecidable, some of which are apparently completely unrelated to Turing machines.

- Given a multivariate polynomial with integer coefficients, does it evaluate to 0 at some integer point? e.g.

$$f(x, y, z) = 5x^2y + 3xyz - 7xy + z^3 + 2$$

has $f(1, -3, 1) = 0$. This is known as Hilbert's tenth problem.

Pic: Wikipedia/David Hilbert

Other undecidable problems

The Post correspondence problem:

- We are given a collection S of dominos, each containing two strings from some alphabet Σ (one on the top half of the domino, one on the bottom). For example,

$$S = \left\{ \left[\frac{a}{ab}\right], \left[\frac{b}{a}\right], \left[\frac{abc}{c}\right] \right\}.$$

Other undecidable problems

The Post correspondence problem:

- We are given a collection S of dominos, each containing two strings from some alphabet Σ (one on the top half of the domino, one on the bottom). For example,

$$S = \left\{ \left[\frac{a}{ab}\right], \left[\frac{b}{a}\right], \left[\frac{abc}{c}\right] \right\}.$$

- The problem is to determine whether, by lining up dominos from S (with repetitions allowed) we can make the concatenated strings on the top of the dominos equal to the concatenated strings on the bottom.

Other undecidable problems

The Post correspondence problem:

- We are given a collection S of dominos, each containing two strings from some alphabet Σ (one on the top half of the domino, one on the bottom). For example,

$$S = \left\{ \left[\frac{a}{ab}\right], \left[\frac{b}{a}\right], \left[\frac{abc}{c}\right] \right\}.$$

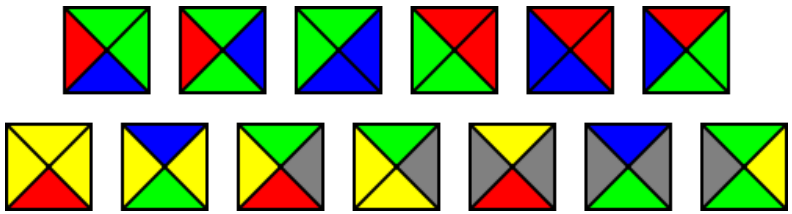
- The problem is to determine whether, by lining up dominos from S (with repetitions allowed) we can make the concatenated strings on the top of the dominos equal to the concatenated strings on the bottom.
- For example,

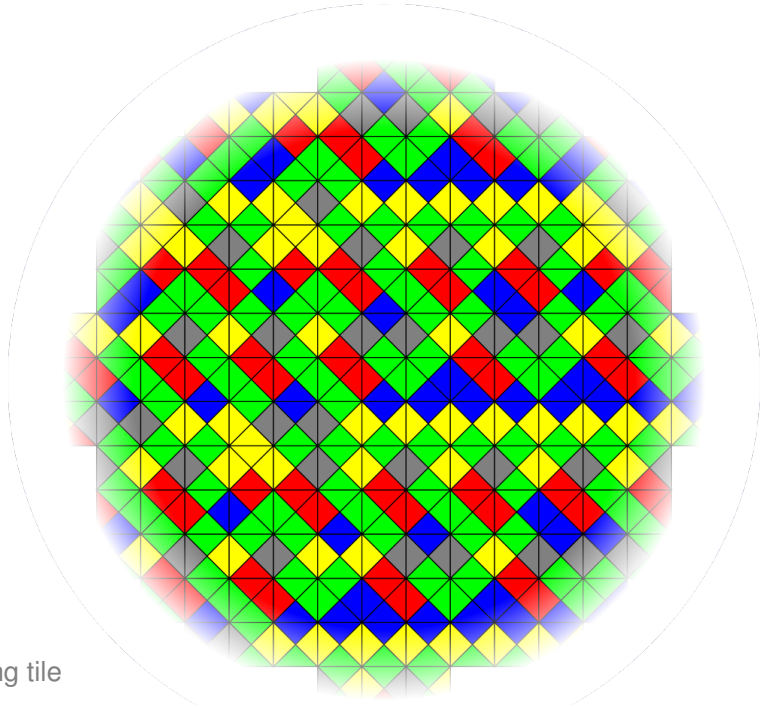
$$\left[\frac{a}{ab}\right] \left[\frac{b}{a}\right] \left[\frac{a}{ab}\right] \left[\frac{abc}{c}\right]$$

would be a valid solution.

Other undecidable problems

- A Wang tile is a unit square with coloured edges.
- Given a set S of Wang tiles, the problem is to determine whether tiles picked from S (without rotations or reflections) can be arranged edge-to-edge to tile the plane, such that adjoining edges of adjacent tiles have the same colour.
- For example, the following set S does satisfy this property:





Pics: Wikipedia/Wang tile

Decidability of problems related to automata

- Turing machines are intuitively at least as powerful as the models of automata we have studied previously.

- We can formalise this by showing that any language that can be recognised by an automaton in one of these models can be decided by a Turing machine. For example:

$$\mathcal{L}_{\text{DFA}} = \{\langle A, x\rangle \mid A \text{ is a DFA that accepts input string } x\}.$$

$$\mathcal{L}_{\text{PDA}} = \{\langle A, x\rangle \mid A \text{ is a PDA that accepts input string } x\}.$$

Decidability of problems related to automata

- Turing machines are intuitively at least as powerful as the models of automata we have studied previously.

- We can formalise this by showing that any language that can be recognised by an automaton in one of these models can be decided by a Turing machine. For example:

$$\mathcal{L}_{\text{DFA}} = \{\langle A, x\rangle \mid A \text{ is a DFA that accepts input string } x\}.$$

$$\mathcal{L}_{\text{PDA}} = \{\langle A, x\rangle \mid A \text{ is a PDA that accepts input string } x\}.$$

Claim

$\mathcal{L}_{\text{DFA}}$ and $\mathcal{L}_{\text{PDA}}$ are decidable.

Decidability of problems related to automata

Claim

$\mathcal{L}_{\text{PDA}}$ is decidable.

We prove the claim by giving an algorithm for deciding $\mathcal{L}_{\text{PDA}}$.

Proof

On input $\langle A, w \rangle$, where A is a PDA and w is a string:

1. Find a CFG G corresponding to A;

Decidability of problems related to automata

Claim

$\mathcal{L}_{\text{PDA}}$ is decidable.

We prove the claim by giving an algorithm for deciding $\mathcal{L}_{\text{PDA}}$.

Proof

On input $\langle A, w \rangle$, where A is a PDA and w is a string:

1. Find a CFG G corresponding to A;
2. Convert G into Chomsky normal form;

Decidability of problems related to automata

Claim

$\mathcal{L}_{\text{PDA}}$ is decidable.

We prove the claim by giving an algorithm for deciding $\mathcal{L}_{\text{PDA}}$.

Proof

On input $\langle A, w \rangle$, where A is a PDA and w is a string:

1. Find a CFG G corresponding to A;
2. Convert G into Chomsky normal form;
3. If $|w| \geq 1$, loop through all derivations with $2|w| - 1$ steps. Otherwise, loop through all derivations with 1 step;

Decidability of problems related to automata

Claim

$\mathcal{L}_{\text{PDA}}$ is decidable.

We prove the claim by giving an algorithm for deciding $\mathcal{L}_{\text{PDA}}$.

Proof

On input $\langle A, w \rangle$, where A is a PDA and w is a string:

1. Find a CFG G corresponding to A;
2. Convert G into Chomsky normal form;
3. If $|w| \geq 1$, loop through all derivations with $2|w| - 1$ steps. Otherwise, loop through all derivations with 1 step;
4. Accept if any of the derivations generates w; otherwise reject.

The CYK algorithm you saw in Programming and Algorithms is another, more efficient way of solving the same problem.

A hierarchy of languages

Recall that

$$\mathcal{L}_{\text{HALT}} = \{\langle M, x\rangle \mid M \text{ is a TM that halts on input } x\}$$

- We have seen that $\mathcal{L}_{\text{HALT}}$ is undecidable.
- But is $\mathcal{L}_{\text{HALT}}$ Turing-recognisable?

A hierarchy of languages

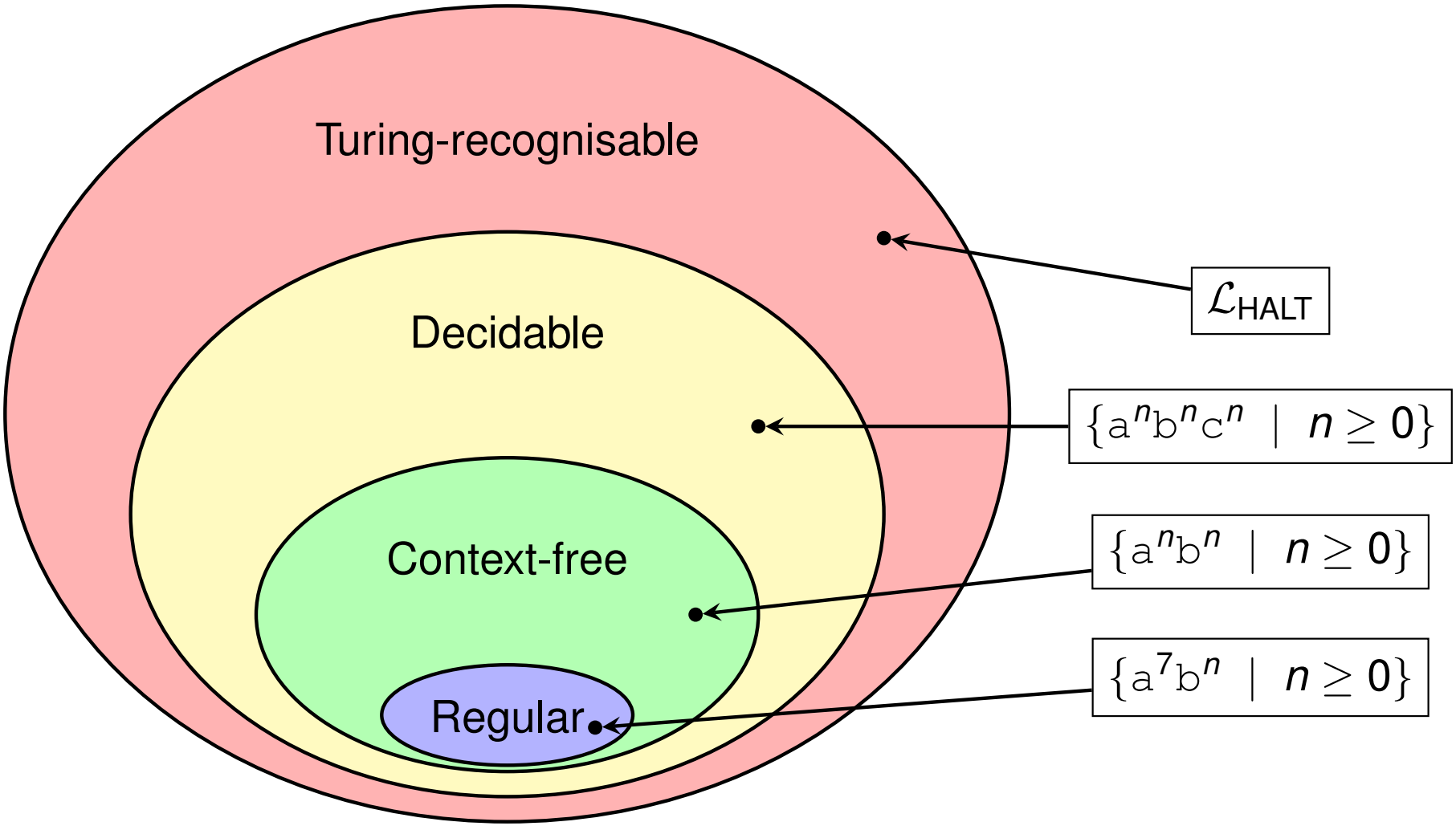
Recall that

$$\mathcal{L}_{\text{HALT}} = \{\langle M, x\rangle \mid M \text{ is a TM that halts on input } x\}$$

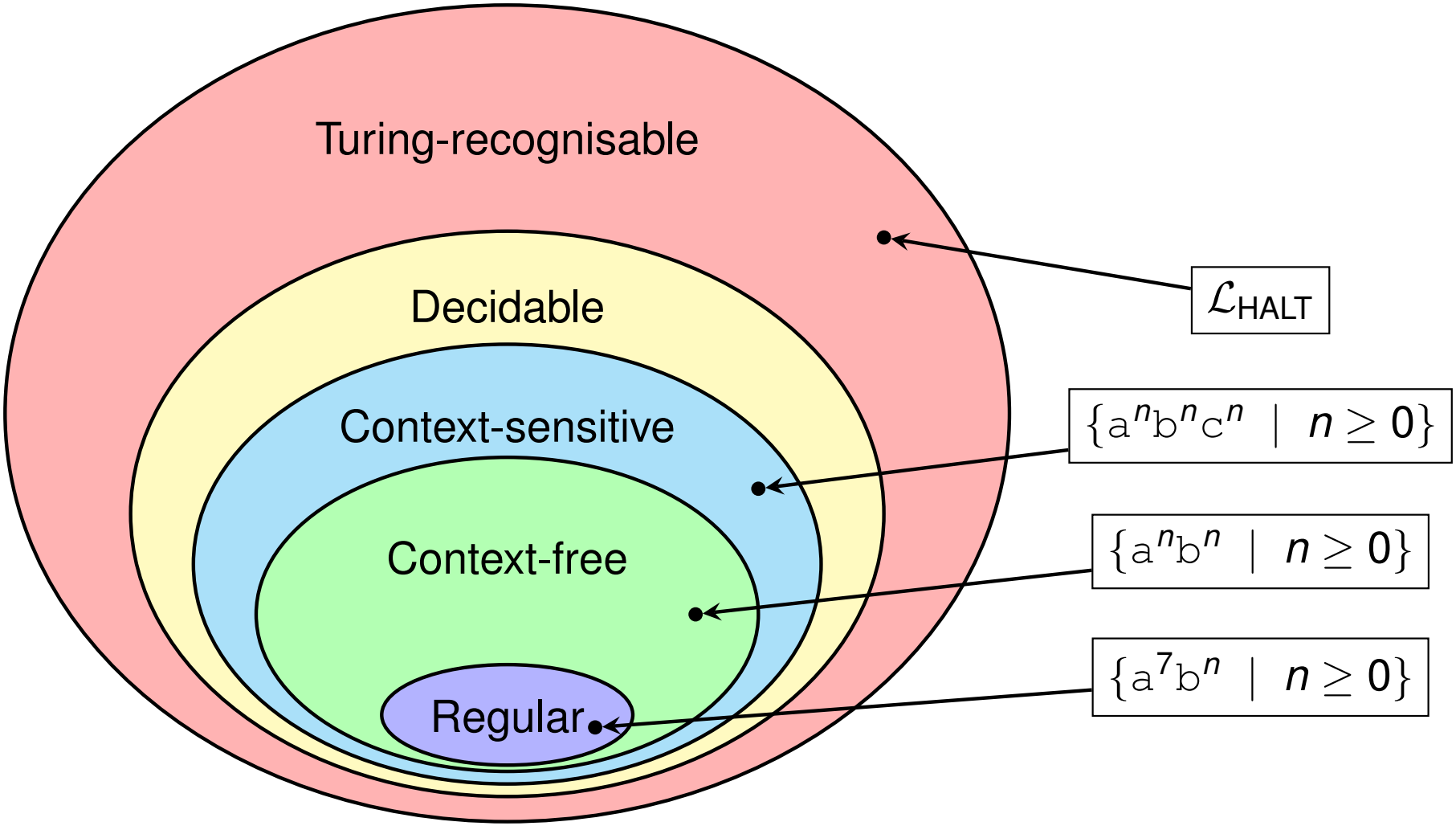
- We have seen that $\mathcal{L}_{\text{HALT}}$ is undecidable.
- But is $\mathcal{L}_{\text{HALT}}$ Turing-recognisable?
- Yes! We can use the universal TM to simulate the TM it was given as input and accept if the TM halts.
- If it does not halt, this process will run forever.

So the set of decidable languages is strictly smaller than the set of Turing-recognisable languages.

A hierarchy of languages



There is another level in this hierarchy which we haven't yet seen:



Context-sensitive languages

A context-sensitive language is generated by a context-sensitive grammar.

- This is a grammar with rules of the form

$$\alpha \to \beta,$$

where α and β are strings (of terminals and non-terminals) such that $|\alpha| \leq |\beta|$. If the start variable S does not appear on the right-hand side of any rules, we also allow the rule $S \to \varepsilon$.

Context-sensitive languages

A context-sensitive language is generated by a context-sensitive grammar.

- This is a grammar with rules of the form

$$\alpha \to \beta,$$

  where α and β are strings (of terminals and non-terminals) such that $|\alpha| \leq |\beta|$. If the start variable S does not appear on the right-hand side of any rules, we also allow the rule $S \to \varepsilon$.

- The name "context-sensitive" comes from the fact that such grammars have a normal form where all rules are of the form $\alpha A \beta \to \alpha \gamma \beta$; so the rule applied to A can depend on the surrounding symbols.

Context-sensitive languages

A context-sensitive language is generated by a context-sensitive grammar.

- This is a grammar with rules of the form

$$\alpha \to \beta,$$

  where α and β are strings (of terminals and non-terminals) such that $|\alpha| \leq |\beta|$. If the start variable S does not appear on the right-hand side of any rules, we also allow the rule $S \to \varepsilon$.

- The name "context-sensitive" comes from the fact that such grammars have a normal form where all rules are of the form $\alpha A \beta \to \alpha \gamma \beta$; so the rule applied to A can depend on the surrounding symbols.

For example, $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 1\}$ is generated by the grammar

$$\begin{aligned} S &\to \mathtt{a}ST\mathtt{c} \mid \mathtt{abc} \\ \mathtt{c}T &\to T\mathtt{c} \\ \mathtt{b}T &\to \mathtt{bb} \end{aligned}$$

Context-sensitive languages

Just as with the other classes of languages we have met, context-sensitive languages are recognised by a corresponding class of automata: linear bounded automata.

Context-sensitive languages

Just as with the other classes of languages we have met, context-sensitive languages are recognised by a corresponding class of automata: linear bounded automata.

- A linear bounded automaton is a nondeterministic Turing machine which is not allowed to move off the tape to the right of the input.
- That is, there is a barrier to the right of the last input symbol which the machine cannot move beyond. Other than this, the machine behaves exactly like a normal NDTM.

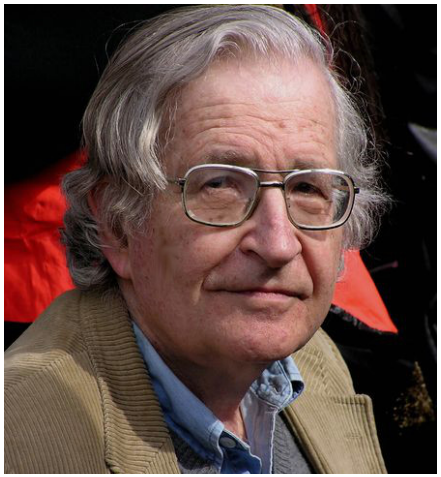
It can be shown using diagonalisation that there exist decidable languages which are not context-sensitive.

The Chomsky hierarchy

We can summarise (some of) these types of languages as follows:

Languages	Automaton	Production rules
Regular	DFA / NFA	$A \to \mathtt{a}$ or $A \to \mathtt{a}B$
Context-free	PDA	$A \to \alpha$
Context-sensitive	Linear bounded automaton	$\alpha A \beta \to \alpha \gamma \beta$
Turing-recognisable	Turing machine	$\alpha \to \beta$

Here α, β, γ are arbitrary strings of terminals and non-terminals.
This classification is called the Chomsky hierarchy.



Pic: Wikipedia/Noam Chomsky

Summary and further reading

To summarise:

- There are languages which cannot be decided by any Turing machine.
- A specific and important example of such a language is the halting problem.
- An important way of proving that a language is undecidable is the use of reductions.

Summary and further reading

To summarise:

- There are languages which cannot be decided by any Turing machine.
- A specific and important example of such a language is the halting problem.
- An important way of proving that a language is undecidable is the use of reductions.

Some additional references:

- The barber paradox can be formalised as Russell's paradox.
- Further reading: Sipser §4.1, §4.2
- Also *Computation: Finite and Infinite Machines* by Minsky
- A nice selection of undecidable problems: `http://math.mit.edu/~poonen/papers/sampler.pdf`