COMP20121 The Implementation and Power of Computer Languages

‘Power’ Part

http://www.cs.man.ac.uk/~petera/2121/index.html .

Peter Aczel

room: CS2.52, tel: 56155

email: `petera@cs.man.ac.uk`

School of Computer Science, University of Manchester

LECTURE EIGHT

Section 3: Turing machines (ctd)

Informal description

Here is an alternative way of describing the same machine.

Informal description

Here is an alternative way of describing the same machine.

- Remember the first symbol of the input string (by going to state 1 if it is an a and to state 2 if it is a b) and erase it.

Informal description

Here is an alternative way of describing the same machine.

- Remember the first symbol of the input string (by going to state 1 if it is an a and to state 2 if it is a b) and erase it.
- Go to the right until you find a blank. (This is the end of the input string.)

Informal description

- If the symbol to the left of that blank is the one remembered, erase it. If it is a blank, stop in an accepting state (0). Otherwise stop (in a non-accepting state).

Informal description

- If the symbol to the left of that blank is the one remembered, erase it. If it is a blank, stop in an accepting state (0). Otherwise stop (in a non-accepting state).
- Go to the left until you find a blank. (This is the start of what's left of the input string.)

Informal description

- If the symbol to the left of that blank is the one remembered, erase it. If it is a blank, stop in an accepting state (0). Otherwise stop (in a non-accepting state).
- Go to the left until you find a blank. (This is the start of what's left of the input string.)
- Start over with the head pointing at the symbol to the right of that blank.

Informal description–II

Having gone through these states we know that

Informal description–II

Having gone through these states we know that

- If the first letter and the last letter of the string were not equal, the machine has stopped in a non-accepting state.

Informal description–II

Having gone through these states we know that

- If the first letter and the last letter of the string were not equal, the machine has stopped in a non-accepting state.
- If the first and the last letter were equal, they have now both been erased and the head is pointing at the new first letter (the old second letter).

Informal description–II

Having gone through these states we know that

- If the first letter and the last letter of the string were not equal, the machine has stopped in a non-accepting state.
- If the first and the last letter were equal, they have now both been erased and the head is pointing at the new first letter (the old second letter). The machine will now start over.

Informal description–II

Hence the recognized language is that

Informal description–II

Hence the recognized language is that of all

palindromes.

Accepting languages

Consider a Turing machine over the alphabet $\Sigma = \{a\}$ which accepts the set of all words of even length

Accepting languages

Consider a Turing machine over the alphabet $\Sigma = \{a\}$ which accepts the set of all words of even length, that is

$$\{a^{2i} \mid i \in \mathbb{N}\}.$$

Accepting languages

Consider a Turing machine over the alphabet $\Sigma = \{a\}$ which accepts the set of all words of even length, that is

$$\{a^{2i} \mid i \in \mathbb{N}\}.$$

We can code this with two states, for ‘odd’ and ‘even’.

Accepting languages–II

The machine moves to the right until it finds a blank, switching between these two states, where 0 is the sole accepting state.

Accepting languages–II

The machine moves to the right until it finds a blank, switching between these two states, where 0 is the sole accepting state.

δ	a	$\sqcup$
0	$(1, a, R)$	stop
1	$(0, a, R)$	stop

Accepting languages–II

Now consider instead the Turing machine given by

δ	a	$\sqcup$
0	$(1, a, R)$	stop
1	$(0, a, R)$	$(1, \sqcup, R)$

Accepting languages–II

Now consider instead the Turing machine given by

δ	a	$\sqcup$
0	$(1, a, R)$	stop
1	$(0, a, R)$	$(1, \sqcup, R)$

This machine moves to the right as well, switching between two states coding for ‘odd’ and ‘even’.

Accepting languages–II

If the machine finds a blank then

Accepting languages–II

If the machine finds a blank then

- it will stop (in an accepting state) if the current state is the one which is for ‘even’;

Accepting languages–II

If the machine finds a blank then

- it will stop (in an accepting state) if the current state is the one which is for ‘even’;
- it will keep moving to the right forever if the current state is the one for ‘odd’.

Accepting languages–III

The machines

δ	a	$\sqcup$
0	$(1, a, R)$	stop
1	$(0, a, R)$	stop

and

δ	a	$\sqcup$
0	$(1, a, R)$	stop
1	$(0, a, R)$	$(1, \sqcup, R)$

both with sole accepting state 0 accept the same language.

Accepting languages–III

The machines

δ	a	$\sqcup$
0	$(1, a, R)$	stop
1	$(0, a, R)$	stop

and

δ	a	$\sqcup$
0	$(1, a, R)$	stop
1	$(0, a, R)$	$(1, \sqcup, R)$

both with sole accepting state 0 accept the same language.

Which one is more useful?

Recognizable *versus* decidable

Definition 16 *A language is Turing-recognizable (or recursively enumerable) if there is a Turing machine which recognizes it.*

Recognizable *versus* decidable

Definition 16 *A language is Turing-recognizable (or recursively enumerable) if there is a Turing machine which recognizes it.*

A language is Turing-decidable (or recursive) if there is a Turing machine which recognizes it which halts for all inputs.

Recognizable *versus* decidable–II

- Every Turing-decidable language is

Turing-recognizable, but not *vice versa*.

Recognizable *versus* decidable–II

- Every Turing-decidable language is Turing-recognizable, but not *vice versa*.
- In order to show that a language is Turing-decidable, we have to design a machine which halts for all inputs. Such a machine will give us a definite answer, ‘yes’ or ‘no’, for every word.

Recognizable *versus* decidable–II

- In order to show that a language is Turing-recognizable, we merely have to design a machine which halts for all inputs which it accepts.

Recognizable *versus* decidable–II

- In order to show that a language is Turing-recognizable, we merely have to design a machine which halts for all inputs which it accepts.
- With a Turing-recognizable language, we may never be certain that some given word really does not belong to the language.

C-F languages are Turing-decidable

Proposition 3.1 *Every context-free language is Turing-decidable.*

Biggest problem: Create machine which terminates for all inputs.

C-F languages are Turing-decidable

Proposition 3.1 *Every context-free language is Turing-decidable.*

The proof requires the following.

C-F languages are Turing-decidable

Proposition 3.1 *Every context-free language is Turing-decidable.*

The proof requires the following.

- Convert grammar into *Chomsky normal form (Chomsky nf)*.

C-F languages are Turing-decidable

Proposition 3.1 *Every context-free language is Turing-decidable.*

The proof requires the following.

- Convert grammar into *Chomsky normal form (Chomsky nf)*.
  - We haven't covered this, so we will only describe the properties of this normal form here.

C-F languages are Turing-decidable

Proposition 3.1 *Every context-free language is Turing-decidable.*

The proof requires the following.

- Convert grammar into *Chomsky normal form (Chomsky nf)*.
  - Then a string of length n will be generated after at most $2n-1$ applications of a production rule.

C-F languages are Turing-decidable

The proof requires the following.

- Convert grammar into *Chomsky nf*.
- There is a TM to do this.

C-F languages are Turing-decidable

The proof requires the following.

- Convert grammar into *Chomsky nf*.
- There is a TM to do this.
- Given an input string α of length n, let the machine generate all words of the language which require no more than $2n - 1$ applications of a production rule.

C-F languages are Turing-decidable

The proof requires the following.

- Convert grammar into *Chomsky nf*.
- There is a TM to do this.
- Given an input string α of length n, let the machine generate all words of the language which require no more than $2n - 1$ applications of a production rule. If α is generated along the way, accept, otherwise reject.

Turing-decidability & complementation

Since a Turing machine deciding a language has to halt for all inputs, we can also find out whether a given word does not belong to the language under consideration.

Turing-decidability & complementation

Since a Turing machine deciding a language has to halt for all inputs, we can also find out whether a given word does not belong to the language under consideration.

Theorem 3.2 *A language is Turing-decidable if and only if its complement is Turing-decidable.*

Turing-decidability & complementation

Since a Turing machine deciding a language has to halt for all inputs, we can also find out whether a given word does not belong to the language under consideration.

Theorem 3.2 *A language is Turing-decidable if and only if its complement is Turing-decidable.*

Proving this result is an exercise.

Enumerating languages

Idea: A Turing machine can do many more things than accept or reject a given input string.

Enumerating languages

Idea: A Turing machine can do many more things than accept or reject a given input string.

For example such a machine can list (or 'enumerate') all the words of a language.

Enumerating languages

Idea: A Turing machine can do many more things than accept or reject a given input string.

For example such a machine can list (or 'enumerate') all the words of a language.

For this it just starts producing words, starting with an empty tape.

Enumerating languages

What languages do we get this way?

Enumerating languages

What languages do we get this way?

Theorem 3.3 *A language is Turing-recognizable if and only if there is a Turing machine enumerating it.*

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

closed under					

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

closed under	compl.				

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

closed under	compl.	intersect.			

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

closed under	compl.	intersect.	union		

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

closed under	compl.	intersect.	union	reversal	

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

closed under	compl.	intersect.	union	reversal	concat.

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

closed under	compl.	intersect.	union	reversal	concat.
regular	✓	✓	✓	✓	✓

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

closed under	compl.	intersect.	union	reversal	concat.
regular	✓	✓	✓	✓	✓
context-free			✓	✓	✓

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

closed under	compl.	intersect.	union	reversal	concat.
regular	✓	✓	✓	✓	✓
context-free			✓	✓	✓
Turing-dec.	✓	✓	✓	✓	✓

Properties of languages

There are many ways of creating languages from existing ones. Some classes of languages are closed under some of these.

closed under	compl.	intersect.	union	reversal	concat.
regular	✓	✓	✓	✓	✓
context-free			✓	✓	✓
Turing-dec.	✓	✓	✓	✓	✓
Turing-rec.		✓	✓	✓	✓

Chomsky's hierarchy

The different kinds of languages we have looked at in this course form a hierarchy.

Chomsky's hierarchy

The different kinds of languages we have looked at in this course form a hierarchy.

With (some of) these languages there are corresponding machines and grammars.

Chomsky's hierarchy

Languages		

Chomsky's hierarchy

Languages	Production rules for grammars are $\Gamma \longrightarrow \Delta$ such that	

Chomsky's hierarchy

Languages	Production rules for grammars are $\Gamma \longrightarrow \Delta$ such that	recognizing machines are

Chomsky's hierarchy

Languages	Production rules for grammars are $\Gamma \longrightarrow \Delta$ such that	recognizing machines are
regular	$\Gamma = R$: any one non-terminal $\Delta = xR'$ or $\Delta = x$ or $\Delta = \epsilon$ where R' non-terminal, x terminal	finite state automata

Chomsky's hierarchy

Languages	Production rules for grammars are $\Gamma \longrightarrow \Delta$ such that	recognizing machines are
regular	$\Gamma = R$: any one non-terminal $\Delta = xR'$ or $\Delta = x$ or $\Delta = \epsilon$ where R' non-terminal, x terminal	finite state automata
context-free	$\Gamma = R$ any one non-terminal Δ any string	non-deterministic pushdown automata

Chomsky's hierarchy

Languages	Production rules for grammars are $\Gamma \longrightarrow \Delta$ such that	recognizing machines are
regular	$\Gamma = R$: any one non-terminal $\Delta = xR'$ or $\Delta = x$ or $\Delta = \epsilon$ where R' non-terminal, x terminal	finite state automata
context-free	$\Gamma = R$ any one non-terminal Δ any string	non-deterministic pushdown automata
context-sensitive	Γ any string containing non-terminals Δ any string at least as long as Γ	Turing machines with bounded tape

Chomsky's hierarchy

Languages	Production rules for grammars are $\Gamma \longrightarrow \Delta$ such that	recognizing machines are
regular	$\Gamma = R$: any one non-terminal $\Delta = xR'$ or $\Delta = x$ or $\Delta = \epsilon$ where R' non-terminal, x terminal	finite state automata
context-free	$\Gamma = R$ any one non-terminal Δ any string	non-deterministic pushdown automata
context-sensitive	Γ any string containing non-terminals Δ any string at least as long as Γ	Turing machines with bounded tape
Turing-recognizable	Γ any string containing non-terminals Δ any string	Turing machines

Section 3 summary

- Turing machines are automata with random-access memory.

Section 3 summary

- Turing machines are automata with random-access memory.
- Turing machines are more powerful than PDAs.

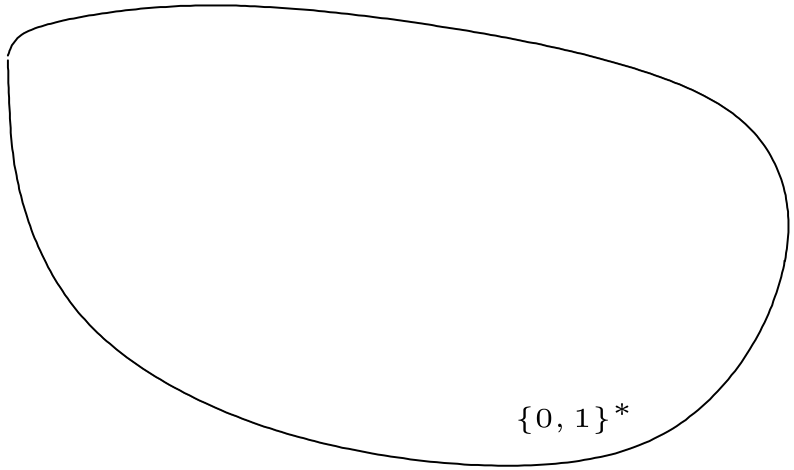
Section 3 summary

- Turing machines are automata with random-access memory.
- Turing machines are more powerful than PDAs.
- Languages recognized by a Turing machine are called Turing-recognizable, and these are the same languages as those which are enumerated by a Turing machine.

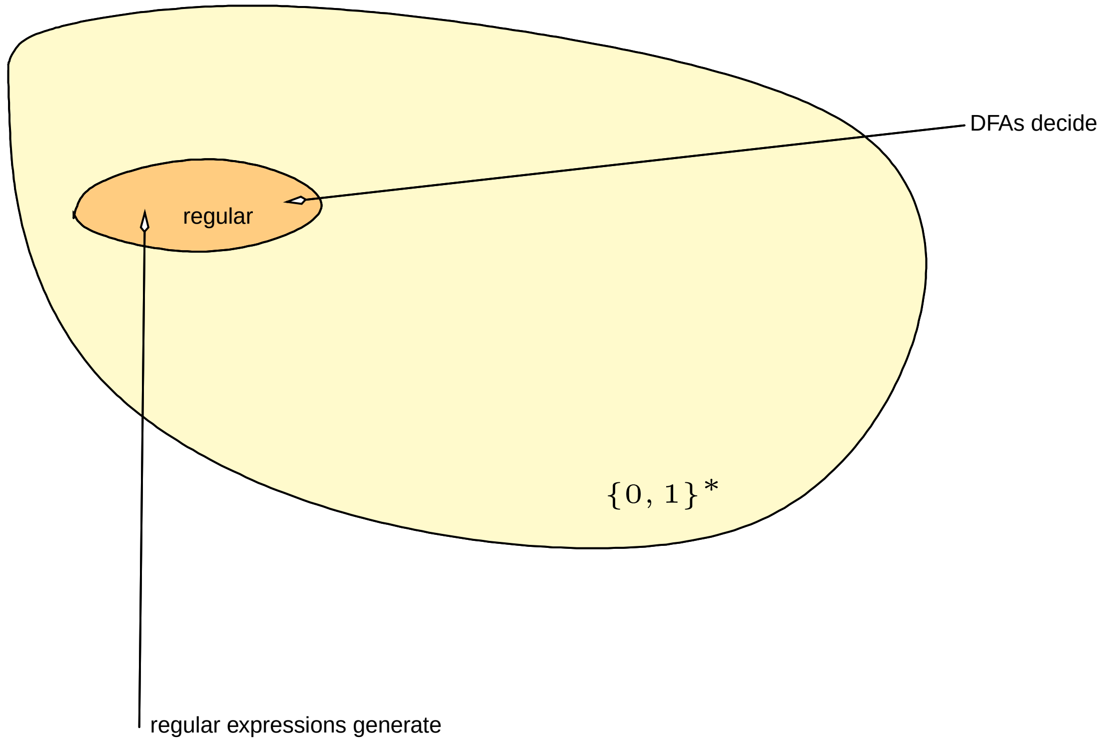
Section 3 summary

- Turing machines are automata with random-access memory.
- Turing machines are more powerful than PDAs.
- Languages recognized by a Turing machine are called Turing-recognizable, and these are the same languages as those which are enumerated by a Turing machine.
- If there is a Turing machine recognizing a language which halts for all inputs we say the language is Turing-decidable. (Rather than just having to accept words belonging to the language, the machine must effectively reject words that do not.)

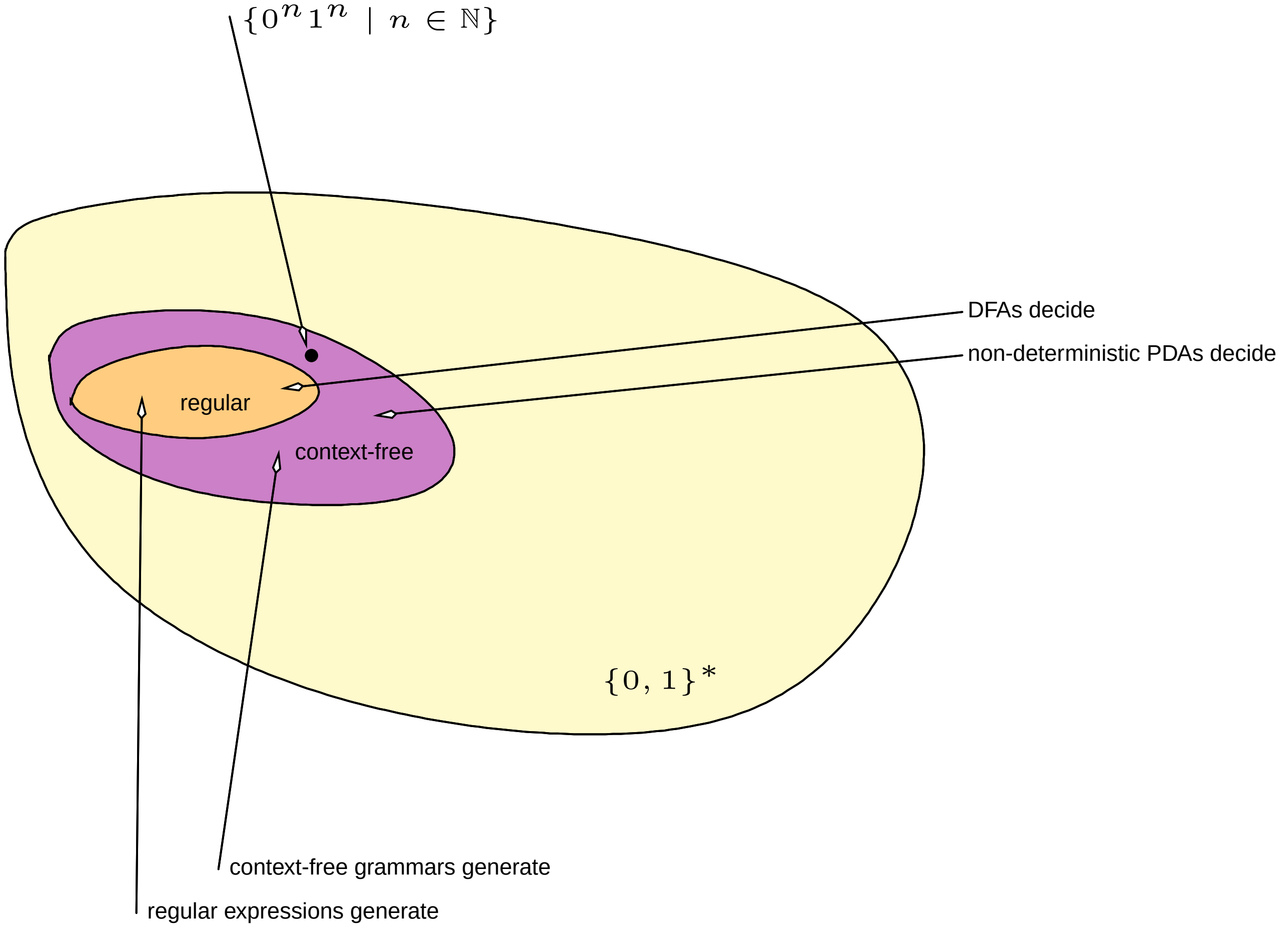
Languages over $\{0, 1\}$



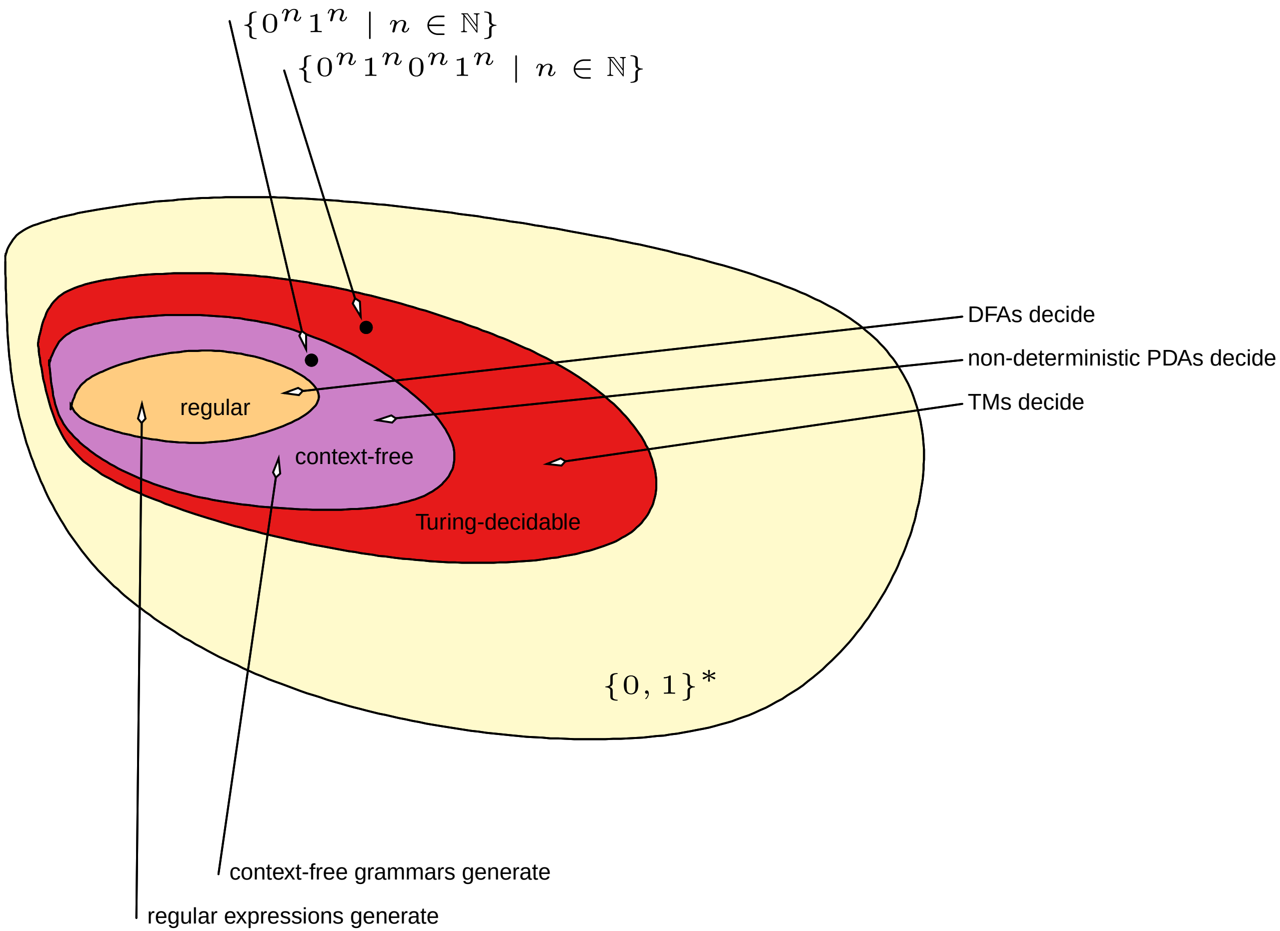
Languages over $\{0, 1\}$



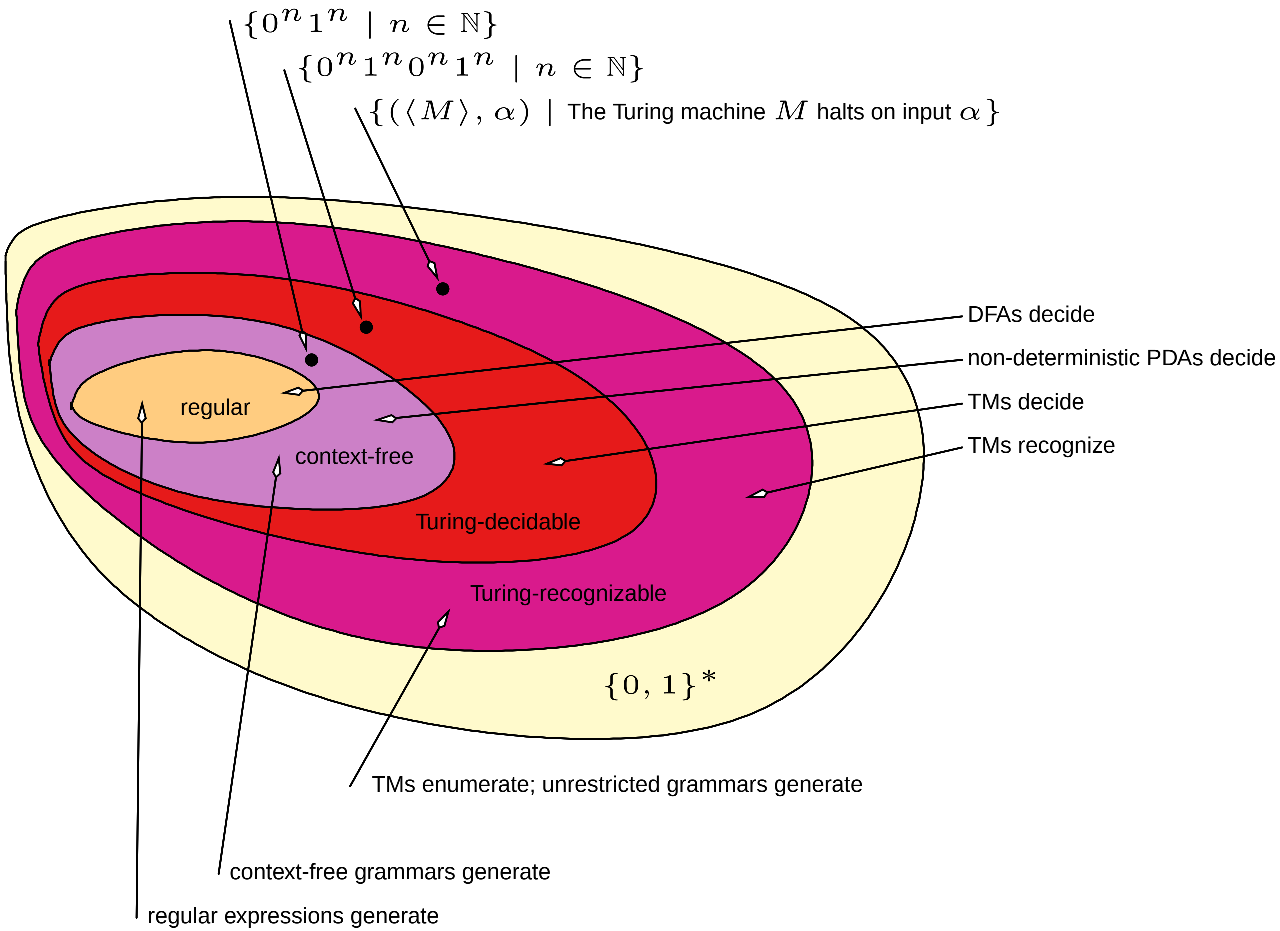
Languages over $\{0, 1\}$



Languages over $\{0, 1\}$



Languages over $\{0, 1\}$



Languages over $\{0, 1\}$