

Griffith University

3515ICT Theory of Computation
Decidability

(Based loosely on slides by Harald Søndergaard of
The University of Melbourne)

Decidable Problems

We are interested in algorithms for deciding problems about languages. Examples of such questions are:

- Is a given string in a given regular language?
- Is a given regular language empty?
- Is a given context-free language finite?
- Is a given context-free language a subset of another context-free language?
- Is a given context-free language deterministic?
- Is a given language Turing-decidable?

In answering such questions, we may assume the languages are presented as regular expressions, finite automata, context-free grammars, Turing machines, *etc.*

Decidable Problems (cont.)

Interesting problems regarding regular languages are generally decidable.

For example, the *acceptance problem* for DFAs is decidable: given a DFA D and a string w , does D accept w ?

Since we can encode a DFA D as a string $\langle D \rangle$, the acceptance problem is equivalent to testing for membership in the language

$$A_{DFA} = \{ \langle D, w \rangle \mid D \text{ is a DFA that accepts } w \}$$

The acceptance problem is called decidable if the corresponding language is (Turing-)decidable.

Problems about context-free languages may or may not be decidable.

Problems about Turing machines are most commonly undecidable.

DFA Acceptance Is Decidable

Theorem. $A_{DFA} = \{ \langle D, w \rangle \mid D \text{ is a DFA that accepts } w \}$ is a decidable language.

Proof outline. The crucial point is that it is possible for a Turing machine M to simulate a DFA D .

M starts with $\langle D, w \rangle$ on its tape, *e.g.*,

$$\underbrace{1\dots73}_{Q} \#\#\underbrace{ab\dots z}_{\Sigma} \#\#\underbrace{1a2\#1b3\#\dots}_{\delta} \#\#\underbrace{1}_{q_0} \#\#\underbrace{13\#29}_{F} \#\#\underbrace{ba\dots}_w \$$$

First M checks that the first five components represent a valid DFA, and if not, rejects.

Then M simulates the moves of D , keeping track of D 's state and the current position in w , by writing these details on its tape, after \$, or on a second tape.

When the last symbol in w has been read, M accepts if D is in a state in F , and rejects otherwise.

TMs as Interpreters

We did not show the details of how a Turing machine goes about simulating a DFA D . Many low-level programming steps are involved.

However, it should be clear that it *is* possible for a Turing machine to model DFA behaviour this way.

The description of D is nothing but a “program” and the a Turing machine can act as an interpreter for this language.

Later, we shall see that Turing machines themselves can be encoded as strings, and that one Turing machine can interpret other Turing machines.

This is no more strange than the fact that we can write an interpreter for Scheme, say, in Scheme.

NFA Acceptance Is Decidable

Theorem. $A_{NFA} = \{ \langle N, w \rangle \mid N \text{ is a NFA that accepts } w \}$ is a decidable language.

Proof outline. The procedure we gave for translating an NFA to an equivalent DFA was mechanical and terminating, so a halting Turing machine can do the translation.

Thus, to recognise whether a given $\langle N, w \rangle$ pair is in A_{NFA} , a Turing machine can:

1. Transform the NFA N to an equivalent DFA D ;
2. Run the TM M from the previous theorem on input $\langle D, w \rangle$.
3. If M accepts, *accept*, otherwise *reject*.

Reg. Exp. Acceptance is Decidable

Theorem.

$A_{REX} = \{ \langle E, w \rangle \mid E \text{ is a regular expression and } w \in L(E) \}$ is a decidable language.

Proof. By translating E into a DFA D and again using the Turing machine M .

Thus, in principle, it does not matter whether we describe a regular language L by a DFA, an NFA or a regular expression, the problem of deciding whether a string w is in L is decidable.

In practice, *e.g.*, in editors and compilers, we describe regular languages by regular expressions, and acceptance is performed by transforming the regular expression to an equivalent NFA, and simulating the behaviour of the NFA by maintaining the set of states it could be in after reading each successive input symbol.

DFA Emptiness Is Decidable

Theorem. $E_{DFA} = \{ \langle D \rangle \mid D \text{ is a DFA and } L(D) = \emptyset \}$ is decidable.

Proof outline. We can design a Turing machine which takes $\langle D \rangle = (Q, \Sigma, \delta, q_0, F)$ as input and performs a reachability analysis:

1. Set $reachable = \{q_0\}$, D 's start state.
2. Set $new = \{ q \mid m \in reachable \text{ and } \delta(m, x) = q \} \setminus reachable$
3. If $new \neq \emptyset$, set $reachable = reachable \cup new$, and go to step 2.
4. If $reachable \cap F \neq \emptyset$, *accept*, otherwise *reject*.

DFA Finiteness Is Decidable

Theorem. $FINITE_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA and } L(A) \text{ is finite} \}$ is decidable.

Proof. See Sipser, Problem 4.9.

DFA Equivalence Is Decidable

Theorem. $EQ_{DFA} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \}$ is decidable.

Proof outline. We previously saw how it is possible to construct, from DFAs A and B , DFAs for $L(A) \cap L(B)$, $L(A) \cup L(B)$, and $\overline{L(A)}$.

These procedures are mechanistic and finite—a halting Turing machine M can perform them.

Hence from DFAs A and B , M can produce a DFA C to recognise

$$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

Note that $L(C) = \emptyset$ iff $L(A) = L(B)$.

Finally, M runs the Turing machine M_E to test whether the constructed DFA C is empty, and accept iff M_E accepts.

Generation by CFGs Is Decidable

Theorem. $A_{CFG} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates } w \}$ is decidable.

Proof. We can put a bound on the number of derivation steps needed to derive any $w \in L(G)$.

First transform G to an equivalent grammar G' in Chomsky Normal Form. So every rule is of form $A \rightarrow BC$ or $A \rightarrow a$.

Let $|w| = n$. To derive w we need to use rules of the first form $n - 1$ times, and rules of the second form n times. So any derivation of w has exactly $2n - 1$ steps.

A Turing machine can therefore take G and w , transform G to G' , generate all derivations of length $2n - 1$, and *accept* iff any one of these generates w .

Ugh!

Generating all possible derivations to see if a string is in a CFL is not very efficient!

A more practical parser for arbitrary CFGs is the dynamic programming algorithm described in Sipser, Theorem 7.16, which is $O(n^3)$, where $n = |w|$.

The most practical parser for arbitrary CFGs is Early's parser.

Any deterministic language can be parsed in $O(n)$ time, using Knuth's LR(k) parser. Practical (deterministic) languages can be parsed in $O(n)$ time using LALR(1) or SLR(1) parsers. *I.e.*, parsing practical CFLs is as efficient as parsing regular languages.

CFG Emptiness Is Decidable

Theorem. $E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ is decidable.

Proof. We can design a Turing machine which takes $\langle G \rangle = (V, \Sigma, R, S)$ as input and performs a “producer” analysis:

1. Set $producers = \Sigma$, all of G 's terminals.

2. Set $new =$

$$\left\{ A \mid \begin{array}{l} A \rightarrow U_1 \cdots U_n \in R, \\ \{U_1, \dots, U_n\} \subseteq producers \end{array} \right\} \setminus producers$$

3. If $new \neq \emptyset$, set $producers = producers \cup new$, and go to step 2.

4. If $S \notin producers$, *accept*, otherwise *reject*.

CFG Finiteness Is Decidable

Theorem. $FINITE_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) \text{ is finite} \}$ is decidable.

Proof. See Sipser, Problem 4.10.

Every CFL Is Decidable

Several slides back we saw that it is decidable whether a CFG G generates a string w .

The deciding Turing machine, S , thus took $\langle G, w \rangle$ as input.

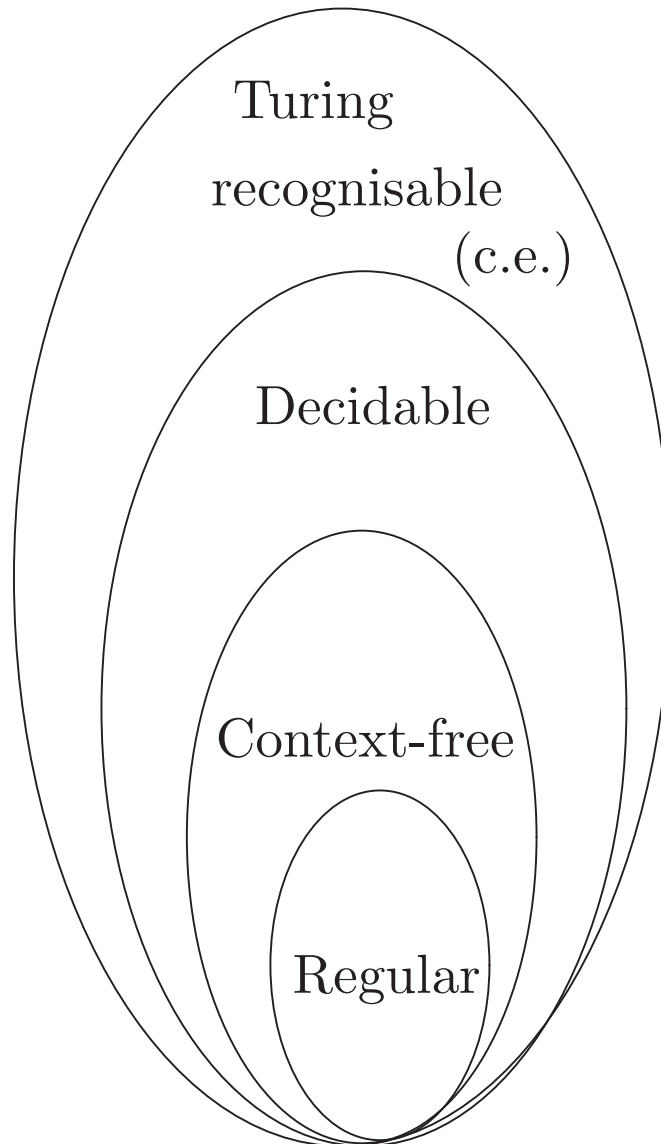
Now we are claiming that any *particular* CFL L_0 is decidable:

Theorem. Every context-free language L_0 is decidable.

Proof. This amounts to saying that we can *specialise* (or partially evaluate) S .

If L_0 has a grammar G_0 , the decider for L_0 simply takes input w and runs S on $\langle G_0, w \rangle$.

Every CFL Is Decidable (cont.)



Are there c.e. languages that are not decidable?
Are there any languages that are not c.e.?

Undecidable CFL Problems

Context-free languages are *much* more complex than regular languages. The following problems (which are decidable for regular languages) are undecidable for context-free languages.

- Equality: The language $EQ_{CFG} = \{ \langle G, H \rangle \mid G \text{ and } H \text{ are CFLs and } L(G) = L(H) \}$ is undecidable.
- Ambiguity: The language $AMB_{CFG} = \{ \langle G \rangle \mid G \text{ is an ambiguous CFG} \}$ is undecidable.

(Have pity on the poor TOC lecturer trying to decide whether his students' grammars generate the correct language or are unambiguous.)

More Decidable Problems

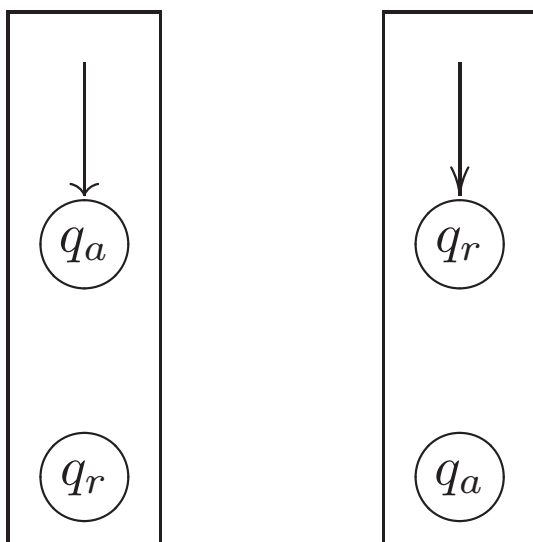
- Is natural number $n \geq 2$ a prime?
- Is every even natural number $n > 2$ the sum of two primes? (Goldbach)
- Is there a path from node x to node y in directed graph G ?
- Is graph G_1 isomorphic to a subgraph of graph G_2 ?
- Is propositional formula P is a tautology?
- Are terms s and t are unifiable?
- Many, many more.

More Decidable Problems (cont.)

As far as decidability is concerned, a problem is only interesting if it is parameterised, *i.e.*, if it has infinitely many instances.

Below is the Turing machine that decides:

“Is every even natural number $n > 2$ the sum of two primes?”



Currently we don't know which of these two it is, but in any case it is a simple Turing machine!