Computability and Complexity
COSC 4200

Decidability

Describing Turing machines.

- formal description: low-level programming, complete detail, state-transition diagram
- implementation description: in English, how the TM moves its head and uses its tapes to store data
- high-level description: description of algorithm, ignoring model

Notation

Binary strings can be used to encode graphs, grammars, automata, Turing machines; any object that can be described finitely. Our notation for encoding an object O is $\langle O \rangle$. We can encode several objects $O_1, \ldots, O_k$ as $\langle O_1, \ldots, O_k \rangle$. The encoding can be done in many ways. It does not matter which one we pick, because a Turing machine can always translate one encoding into another.

Decidable Problems for DFAs

The *acceptance problem* for DFAs:

$$A_{\mathrm{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\ \}.$$

Theorem

A_{DFA} is decidable.

Decidable Problems for DFAs

The *acceptance problem* for DFAs:

$$A_{\mathrm{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w \}.$$

Theorem

A_{DFA} is decidable.

Proof. Algorithm M:
On input $\langle B, w\rangle$, where B is a DFA and w is a string:

Decidable Problems for DFAs

The *acceptance problem* for DFAs:

$$A_{\mathrm{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\ \}.$$

Theorem

A_{DFA} is decidable.

Proof. Algorithm M:
On input $\langle B, w\rangle$, where B is a DFA and w is a string:

1. Simulate B on w.

Decidable Problems for DFAs

The *acceptance problem* for DFAs:

$$A_{\mathrm{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\ \}.$$

Theorem

A_{DFA} *is decidable.*

Proof. Algorithm M:
On input $\langle B, w\rangle$, where B is a DFA and w is a string:

1. Simulate B on w.
2. If the simulation ends in an accept state, accept.
   If it ends in a nonaccepting state, reject. □

The *acceptance problem* for NFAs:

$$A_{\mathrm{NFA}} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\ \}.$$

Theorem

A_{NFA} is decidable.

The *acceptance problem* for NFAs:

$$A_{\mathrm{NFA}} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\ \}.$$

Theorem

A_{NFA} is decidable.

Proof. Algorithm N:
On input $\langle B, w\rangle$, where B is a NFA and w is a string:

The *acceptance problem* for NFAs:

$$A_{\mathrm{NFA}} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\ \}.$$

Theorem

A_{NFA} is decidable.

Proof. Algorithm N:

On input $\langle B, w\rangle$, where B is a NFA and w is a string:

1. Convert B into an equivalent DFA C using the subset construction.

The *acceptance problem* for NFAs:

$$A_{\mathrm{NFA}} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\ \}.$$

Theorem

A_{NFA} is decidable.

Proof. Algorithm N:
On input $\langle B, w\rangle$, where B is a NFA and w is a string:

1. Convert B into an equivalent DFA C using the subset construction.
2. Run M from the previous proof to see if $\langle C, w\rangle \in A_{\mathrm{DFA}}$.

The *acceptance problem* for NFAs:

$$A_{\mathrm{NFA}} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\ \}.$$

Theorem

A_{NFA} is decidable.

Proof. Algorithm N:
On input $\langle B, w\rangle$, where B is a NFA and w is a string:

1. Convert B into an equivalent DFA C using the subset construction.
2. Run M from the previous proof to see if $\langle C, w\rangle \in A_{\mathrm{DFA}}$.
3. If M accepts, accept; otherwise, reject. $\square$

The *acceptance problem* for regular expressions:

$$A_{\mathrm{REX}} = \left\{ \langle R, w \rangle \;\middle|\; \begin{array}{l} R \text{ is a regular expression that} \\ \text{generates input string } w \end{array} \right\}.$$

Theorem

A_{REX} is decidable.

The *acceptance problem* for regular expressions:

$$A_{\mathrm{REX}} = \left\{ \langle R, w \rangle \;\middle|\; \begin{array}{l} R \text{ is a regular expression that} \\ \text{generates input string } w \end{array} \right\}.$$

Theorem

A_{REX} *is decidable.*

Proof. Algorithm P:

On input $\langle R, w \rangle$, where B is a regular expression and w is a string:

The *acceptance problem* for regular expressions:

$$A_{\mathrm{REX}} = \left\{ \langle R, w \rangle \;\middle|\; \begin{array}{l} R \text{ is a regular expression that} \\ \text{generates input string } w \end{array} \right\}.$$

Theorem

A_{REX} is decidable.

Proof. Algorithm P:

On input $\langle R, w \rangle$, where B is a regular expression and w is a string:

1. Convert R into an equivalent NFA A.

The *acceptance problem* for regular expressions:

$$A_{\mathrm{REX}} = \left\{ \langle R, w \rangle \;\middle|\; \begin{array}{l} R \text{ is a regular expression that} \\ \text{generates input string } w \end{array} \right\}.$$

Theorem

A_{REX} is decidable.

Proof. Algorithm P:

On input $\langle R, w \rangle$, where B is a regular expression and w is a string:

1. Convert R into an equivalent NFA A.
2. Run the TM N for A_{NFA} on input $\langle A, w \rangle$.

The *acceptance problem* for regular expressions:

$$A_{\mathrm{REX}} = \left\{ \langle R, w \rangle \;\middle|\; \begin{array}{l} R \text{ is a regular expression that} \\ \text{generates input string } w \end{array} \right\}.$$

Theorem

A_{REX} is decidable.

Proof. Algorithm P:

On input $\langle R, w \rangle$, where B is a regular expression and w is a string:

1. Convert R into an equivalent NFA A.
2. Run the TM N for A_{NFA} on input $\langle A, w \rangle$.
3. If N accepts, accept; otherwise, reject. $\square$

The *emptiness problem* for DFAs:

$$E_{\mathrm{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\ \}.$$

Theorem

E_{DFA} is decidable.

The *emptiness problem* for DFAs:

$$E_{\mathrm{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset \}.$$

Theorem

E_{DFA} is decidable.

Proof. Algorithm T:

On input $\langle A\rangle$, where A is a DFA:

The *emptiness problem* for DFAs:

$$E_{\mathrm{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\ \}.$$

Theorem

E_{DFA} is decidable.

Proof. Algorithm T:

On input $\langle A\rangle$, where A is a DFA:

1. Mark the initial state of A.

The *emptiness problem* for DFAs:

$$E_{\mathrm{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\ \}.$$

Theorem

E_{DFA} is decidable.

Proof. Algorithm T:

On input $\langle A \rangle$, where A is a DFA:

1. Mark the initial state of A.
2. Repeat until no new states get marked:
   - Mark any state that has a transition coming into it from any state that is already marked.

The *emptiness problem* for DFAs:

$$E_{\mathrm{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\ \}.$$

Theorem

E_{DFA} is decidable.

Proof. Algorithm T:

On input $\langle A \rangle$, where A is a DFA:

1. Mark the initial state of A.
2. Repeat until no new states get marked:
   - Mark any state that has a transition coming into it from any state that is already marked.
3. If no accepting state is marked, accept. Otherwise, reject. $\square$

The *equivalence problem* for DFAs:

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

Theorem

EQ_{DFA} is decidable.

The *equivalence problem* for DFAs:

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

Theorem

EQ_{DFA} is decidable.

Proof. We use the product construction to construct a DFA C with

$$L(C) = L(A) \oplus L(B) = (L(A) - L(B)) \cup (L(B) - L(A)).$$

Then

$$L(C) = \emptyset \Leftrightarrow L(A) = L(B).$$

The *equivalence problem* for DFAs:

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

Theorem

EQ_{DFA} is decidable.

Proof. We use the product construction to construct a DFA C with

$$L(C) = L(A) \oplus L(B) = (L(A) - L(B)) \cup (L(B) - L(A)).$$

Then

$$L(C) = \emptyset \Leftrightarrow L(A) = L(B).$$

Algorithm F:
On input $\langle A, B\rangle$, where A and B are DFAs:

The *equivalence problem* for DFAs:

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

Theorem

EQ_{DFA} is decidable.

Proof. We use the product construction to construct a DFA C with

$$L(C) = L(A) \oplus L(B) = (L(A) - L(B)) \cup (L(B) - L(A)).$$

Then

$$L(C) = \emptyset \Leftrightarrow L(A) = L(B).$$

Algorithm F:
On input $\langle A, B\rangle$, where A and B are DFAs:

1. Construct the DFA C as described above.

The *equivalence problem* for DFAs:

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

Theorem

EQ_{DFA} is decidable.

Proof. We use the product construction to construct a DFA C with

$$L(C) = L(A) \oplus L(B) = (L(A) - L(B)) \cup (L(B) - L(A)).$$

Then

$$L(C) = \emptyset \Leftrightarrow L(A) = L(B).$$

Algorithm F:
On input $\langle A, B\rangle$, where A and B are DFAs:

1. Construct the DFA C as described above.
2. Run algorithm T from the previous proof to see if $\langle C\rangle \in E_{\mathrm{DFA}}$.

The *equivalence problem* for DFAs:

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

Theorem

EQ_{DFA} is decidable.

Proof. We use the product construction to construct a DFA C with

$$L(C) = L(A) \oplus L(B) = (L(A) - L(B)) \cup (L(B) - L(A)).$$

Then

$$L(C) = \emptyset \Leftrightarrow L(A) = L(B).$$

Algorithm F:
On input $\langle A, B\rangle$, where A and B are DFAs:

1. Construct the DFA C as described above.
2. Run algorithm T from the previous proof to see if $\langle C\rangle \in E_{\mathrm{DFA}}$.
3. If T accepts, accept. If T rejects, reject. $\square$

Decidable Problems for CFGs

The *acceptance problem* for CFGs:

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

Theorem

A_{CFG} is decidable.

Decidable Problems for CFGs

The *acceptance problem* for CFGs:

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

Theorem

A_{CFG} is decidable.

Proof. Algorithm S:
On input $\langle G, w\rangle$, where G is a CFG and w is a string:

Decidable Problems for CFGs

The *acceptance problem* for CFGs:

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

Theorem

A_{CFG} is decidable.

Proof. Algorithm S:
On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert G into Chomsky normal form.
   (All rules are of the form $A \to BC$ or $A \to a$.)

Decidable Problems for CFGs

The *acceptance problem* for CFGs:

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

Theorem

A_{CFG} is decidable.

Proof. Algorithm S:
On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert G into Chomsky normal form.
   (All rules are of the form $A \to BC$ or $A \to a$.)
2. List all derivations with $2n - 1$ steps, where $n = |w|$.

Decidable Problems for CFGs

The *acceptance problem* for CFGs:

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

Theorem

A_{CFG} is decidable.

Proof. Algorithm S:
On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert G into Chomsky normal form.
   (All rules are of the form $A \to BC$ or $A \to a$.)
2. List all derivations with $2n - 1$ steps, where $n = |w|$.
3. If any of these derviations generate w, accept; if not, reject. $\square$

Cocke-Kasami-Younger: $O(n^3)$-time algorithm

Theorem

Every context-free language is decidable.

Theorem

Every context-free language is decidable.

Proof. Let A be context-free and let G be a CFG for A.

Algorithm: On input w:

Theorem

Every context-free language is decidable.

Proof. Let A be context-free and let G be a CFG for A.

Algorithm: On input w:

1. Run Algorithm S (or the CKY algorithm) to see if $\langle G, w \rangle \in A_{\mathrm{CFG}}$.

Theorem

Every context-free language is decidable.

Proof. Let A be context-free and let G be a CFG for A.

Algorithm: On input w:

1. Run Algorithm S (or the CKY algorithm) to see if $\langle G, w \rangle \in A_{\mathrm{CFG}}$.
2. If S accepts, accept; otherwise, reject.

Theorem

Every context-free language is decidable.

Proof. Let A be context-free and let G be a CFG for A.

Algorithm: On input w:

1. Run Algorithm S (or the CKY algorithm) to see if $\langle G, w \rangle \in A_{\mathrm{CFG}}$.
2. If S accepts, accept; otherwise, reject.

Note that G is hardcoded into this algorithm. $\square$

The *emptiness problem* for CFGs:

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

Theorem

E_{CFG} is decidable.

The *emptiness problem* for CFGs:

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

Theorem

E_{CFG} is decidable.

Proof. Algorithm R:
On input $\langle G\rangle$, where G is a CFG:

1. Mark all terminal symbols in G.

The *emptiness problem* for CFGs:

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

Theorem

E_{CFG} is decidable.

Proof. Algorithm R:
On input $\langle G\rangle$, where G is a CFG:

1. Mark all terminal symbols in G.
2. Repeat until no new variables get marked.
    - Mark any variable A where G has a rule $A \to U_1 U_2 \cdots U_k$ and each symbol $U_1, \ldots U_k$ has already been marked.

The *emptiness problem* for CFGs:

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

Theorem

E_{CFG} is decidable.

Proof. Algorithm R:

On input $\langle G\rangle$, where G is a CFG:

1. Mark all terminal symbols in G.
2. Repeat until no new variables get marked.
   - Mark any variable A where G has a rule $A \to U_1 U_2 \cdots U_k$ and each symbol $U_1, \ldots U_k$ has already been marked.
3. If the start symbol is not marked, accept; otherwise, reject. $\square$

The *equivalence problem* for CFGs:

$$EQ_{\mathrm{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\ \}.$$

The *equivalence problem* for CFGs:

$$EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\ \}.$$

Algorithm? We will show that no algorithm exists!