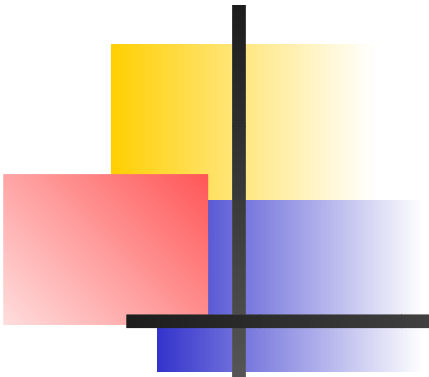
Proofs using Deciders

Proposition: Let L_1 and L_2 be decidable languages, then the concatenation $L = L_1 \circ L_2$ is also decidable.

Proof: We show decidability of L by constructing a decider for it. Let M_1 and M_2 be deciders for L_1 and L_2, respectively, then we can construct a decider M for L as follows:

M = "On input w,

1. For each way to split w into two parts, $w = w_1 w_2$, do:
2. Run M_1 on w_1.
3. Run M_2 on w_2.
4. If for any combination M_1 and M_2 accept, *accept*; otherwise, *reject*."



Decidability

Let us study some "standard" machines that are deciders.

It turns out that these standard machines help us construct more complicated proofs.

Decidability

Theorem: The language

$$A_{\text{DFA}} = \{\langle B, w\rangle | B \text{ is a DFA that accepts string } w\}$$

is decidable.

Proof: We construct a decider M_{DFA} for A_{DFA}.

M_{DFA} = "On input $\langle B, w\rangle$, where B is a DFA and w is a string:

1. Simulate B on input w.
2. If the simulation ends in an accept state of the DFA, *accept*; otherwise, *reject*."

Decidability

Theorem: The language

$$A_{\text{NFA}} = \{\langle B, w\rangle | B \text{ is an NFA that accepts string } w\}$$

is decidable.

Proof: We construct a decider M_{NFA} for A_{NFA}.

M_{NFA} = "On input $\langle B, w\rangle$, where B is an NFA and w is a string:

1. Convert NFA B into an equivalent DFA B' (this is algorithmic, so a TM can do it).
2. Run M_{DFA} on input $\langle B', w\rangle$.
3. If M_{DFA} accepts, *accept*; otherwise, *reject*."

Decidability

Theorem: The language

$$A_{\text{REX}} = \{\langle R, w\rangle | R \text{ is a regular expression that generates string } w\}$$

is decidable.

Proof: We construct a decider M_{REX} for A_{REX}.

M_{REX} = "On input $\langle R, w\rangle$, where R is a regular expression and w is a string:

1. Convert regular expression R into an equivalent DFA R' (this is algorithmic, so a TM can do it).
2. Run M_{DFA} on input $\langle R', w\rangle$.
3. If M_{DFA} accepts, *accept*; otherwise, *reject*."

Decidability

Theorem: The language

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

is decidable.

Proof: We construct a decider $M_{E_{\text{DFA}}}$ for E_{DFA}.

$M_{E_{\text{DFA}}}$ = "On input $\langle A \rangle$, where A is a DFA:

1. Mark the start state of A.
2. Repeat until no new states get marked:
3. Mark any state that has a transition coming into it from any state already marked.
4. If no accept state is marked, *accept*; otherwise, *reject*."

Decidability

Theorem: The language

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

is decidable.

Proof: In order to show this we construct a new machine C that accepts only those string that either A or B accepts but not both, i.e.

$$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B)). \quad (1)$$

Furthermore, we require that $L(C) = \emptyset$ which implies that $L(A) = L(B)$ as needed.

Regular languages are closed under complementation, union, and intersection; therefore we are able to construct machine C according to (1).

We can now construct a decider $M_{EQ_{\text{DFA}}}$ for EQ_{DFA}.

$M_{EQ_{\text{DFA}}}$ = "On input $\langle A, B\rangle$, where A B are DFAs:

1. Construct DFA C as described in (1).
2. Run $M_{E_{\text{DFA}}}$ on input $\langle C\rangle$.
3. If $M_{E_{\text{DFA}}}$ accepts, *accept*; otherwise, *reject*."

Decidability

Theorem: The language

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

is decidable.

Proof Attempt 1: We construct a decider M_{CFG} for A_{CFG}.

M_{CFG} = "On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert CFG G into an equivalent PDA G' (this is algorithmic, so a TM can do it).
2. Simulate G' on input w.
3. If the simulation ends in an accept state, *accept*; otherwise, *reject*."

$\Rightarrow$ Turns out that this does ***not*** work because a PDA is a non-deterministic machine and can have branches could go on computing forever.

NOTE: in practice we actually do use PDAs to decide membership, since infinite computations are not a problem since we always "guess" just the appropriate transitions using dynamic programming techniques and when we cannot guess the appropriate transition that usually signals an error.

Decidability

Theorem: The language

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

is decidable.

Proof Attempt 2: We construct a decider M_{CFG} for A_{CFG}.

M_{CFG} = "On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert CFG G into an equivalent Chomsky normal form G' (this is algorithmic, so a TM can do it).
2. List all derivations with $2n - 1$ steps, where n is the length of w, except if $n = 0$, then list all derivations with 1 step.
3. If any of these derivations generate w, *accept*; otherwise, *reject*."

NOTE: now the computation is bounded by $2n - 1 \rightarrow$ decider.

Decidability

Theorem: The language

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

is decidable.

Proof: We construct a decider $M_{E_{\text{CFG}}}$ for E_{CFG}.

$M_{E_{\text{CFG}}}$ = "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:
3. Mark any variable A where G has a rule $A \rightarrow U_1 U_2 \ldots U_k$ and each symbol $U_1 U_2 \ldots U_k$ has already been marked.
4. If the start symbol is not marked, *accept*; otherwise, *reject*."□

Example:

$$\begin{aligned} S &\rightarrow ASA \\ S &\rightarrow aB \\ A &\rightarrow B \\ A &\rightarrow S \\ B &\rightarrow b \\ B &\rightarrow \epsilon \end{aligned}$$

Decidability

Theorem: Every context-free language is decidable.

Proof: Let A be a CFL, then we know that there must be a CFG G such that $L(G) = A$. Now we can construct a decider M_A for A as follows:

M_A = "On input w, where w is some string:

1. Run M_{CFG} on $\langle G, w \rangle$.
2. If M_{CFG} accepts, *accept*; otherwise, *reject*."

Example

Example: Let

$A = \{\langle M \rangle | M$ is a DFA which does not accept any string containing an odd number of 1's$\}$

Show that A is decidable.

Proof: We need to construct a decider M for A.

M = "On input $\langle M \rangle$, where M is a DFA:

1. Construct a DFA O that accepts every string containing an odd number of 1's.
2. Construct a DFA B such that $L(B) = L(M) \cap L(O)$.
3. Run $M_{E_{\text{DFA}}}$ on input $\langle B \rangle$.
4. If $M_{E_{\text{DFA}}}$ accepts, *accept*; otherwise, *reject*."

The Halting Problem

Theorem: The language

$$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w.\}$$

is undecidable.

Proof: By contradiction. Assume that there exists a decider H for language A_{TM}. Then,

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

We construct a new TM D such that

D = "On input $\langle Q\rangle$ where Q is a TM:

1. Run H on input $\langle Q, \langle Q\rangle\rangle$.
2. Output the opposite of what H outputs; that is, if H accepts, *reject* and if H rejects, *accept*."

Now,

$$D(\langle M\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M\rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M\rangle \end{cases}$$

But...

The Halting Problem

Now consider,

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

This is a contradiction. Therefore, neither H nor D can exist and A_{TM} is undecidable.□

A_{TM} is Turing-Recognizable

Theorem: The language

$$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w.\}$$

is Turing-Recognizable.

Proof: We construct the TM U that recognizes A_{TM},

U = "On input $\langle M, w\rangle$ where M is a TM and w is a string:

1. Simulate M on string w.
2. If M ever enters its accept state, *accept*; if M ever enters its reject state, *reject*."

Non-Turing-Recognizable Languages

The undecidability of A_{TM} has profound ramifications $\Rightarrow$ the existence of languages that are non-algorithmic, that is, languages that are not Turing-recognizable.

One such language is the complement of A_{TM}.

Non-Turing-Recognizable Languages

Theorem: The language $\overline{A_{TM}}$ is not Turing-recognizable.

Proof: By contradiction. Observe that A_{TM} is undecidable. In addition, observe that A_{TM} is Turing-recognizable. Now, assume that $\overline{A_{TM}}$ is also Turing-recognizable. Observing that a string w is either an element of A_{TM} or an element of $\overline{A_{TM}}$ we can construct the following decider for A_{TM}. Let M_1 and M_2 be recognizers for A_{TM} and $\overline{A_{TM}}$, respectively:

M = "On input w, where w is a string:

1. Run M_1 and M_2 in parallel on w.
2. If M_1 accepts, *accept*; if M_2 accepts, *reject*."

Note that this machine is a decider because it will halt on every input w. Also note, that this decider contradicts our theorem that A_{TM} is undecidable. Therefore, our assumption that $\overline{A_{TM}}$ is Turing-recognizable must be wrong.

This shows that $\overline{A_{TM}}$ is not Turing-recognizable.□