CS 476 – 7. Decidability/Undecidability

1 Decidable languages

Languages that are always accepted or rejected by a Turing Machine (i.e. by a *decider* TM).

Definition We will use $< M >$ to denote the binary encoding of a TM. Or, alternatively, write a program for M, and let $< M >$ be its binary executable.

Examples:

Theorem 1.1 $A_{DFA} = \{< B, w > \mid B \, is \, a \, DFA \, that \, accepts \, input \, string \, w\}$ *is decidable.*

Proof We simply need to present a TM M_1 that decides A_{DFA}:
M_1 = On input $< B, w >$, 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, *accept*. If it ends in a nonaccepting state, *reject*.

$\blacksquare$

Theorem 1.2 $A_{NFA} = \{< B, w > \mid B \, is \, an \, NFA \, that \, accepts \, input \, string \, w\}$ *is decidable.*

Proof We will use M_1 from above as a subroutine.
M_2 = On input $< B, w >$, where B is an NFA and w is a string

1. Convert NFA B to an equivalent DFA C.
2. Run TM M_1 on input $< C, w >$.
3. If M_1 accepts, *accept*; otherwise, *reject*.

$\blacksquare$

Theorem 1.3 $A_{REX} = \{< R, w > \mid R \, is \, a \, regular \, expression \, that \, generates \, string \, w\}$ *is decidable.*

Proof M_3 = On input $< R, w >$, where R is a regular expression and w is a string

1. Convert regular expression R to an equivalent NFA A.
2. Run TM M_2 on input $< A, w >$.
3. If M_2 accepts, *accept*; otherwise, *reject*.

$\blacksquare$

Theorem 1.4 $E_{DFA} = \{< A > \mid A \, is \, a \, DFA \, and \, L(A) = \emptyset\}$ *is decidable.*

Proof M_4 = On input $< A >$, where A is a DFA

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

Theorem 1.5 $EQ_{DFA} = \{< A, B > \mid A \text{ and } B \text{ are } DFAs \text{ and } L(A) = L(B)\}$ *is decidable.*

Proof M_5 = On input $< A, B >$, where A, B are DFAs

1. Construct a new DFA C from A and B, where C accepts only those strings that are accepted by either A or B, but not both. Thus, if A and B recognize the same language, C will accept nothing. The language of C is:
$$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$
2. Run M_4 on input $< C >$.
3. If M_4 accepts, *accept*; otherwise, *reject*. $\blacksquare$

Theorem 1.6 $A_{CFG} = \{< G, w > \mid G \text{ is a } CFG \text{ that generates string } w\}$ *is decidable.*

Theorem 1.7 $E_{CFG} = \{< G > \mid G \text{ is a } CFG \text{ and } L(G) = \emptyset\}$ *is decidable.*

But:

Theorem 1.8 $EQ_{CFG} = \{< G, H > \mid G \text{ and } H \text{ are } CFGs \text{ and } L(G) = L(H)\}$ *is* **NOT** *decidable.*

2 Undecidability

- Problems that cannot be solved by computers (TMs).
- Practical significance: software testing.

Fact: Every TM can be encoded as a binary string according to some convention.

Example Assume:

- q_1 is the start state, q_2 is the accept state, q_3 is the reject state, and q_i, $i \geq 4$ are the rest of the states.
- $\Gamma = \{X_1, X_2, X_3, \ldots\}$ where $X_1 = 0, X_2 = 1, X_3 = \sqcup, \ldots$
- $\{L, R\} = \{D_1, D_2\}$
- A transition $\delta(q_i, X_j) = (q_k, X_l, D_m)$ is encoded as $0^i10^j10^k10^l10^m$.
- The TM is encoded as the list of its transitions seperated by 11s since transition codes do not include two consecutive 1s. We also include a prefix of 1 to be able to convert to an integer:

$$TM = 1\ Code1\ 11\ Code2\ 11\ Code3\ \ldots$$

Then the i^{th} TM is the TM, where the binary representation of the integer i is the binary encoding of TM ($< M >$). Some TMs do not make sense. For an i^{th} TM to be valid, there must be 5 blocks of 0s seperated by single 1s between any pair of 11s. For example the 3^{rd} TM does not make sense. For such TMs we will assume that the language they accept is empty.

2.1 A language that is not Turing-recognizable

TM i \ input string j	0	1	2	3	4	5	...
0	0	0	0	0	0	0	
1	0	0	0	0	0	0	
. . . .							
$< M >$	0	0	1	0	1	0	
. . . .							

The diagonal will be defined as

$$D = a_1 a_2 a_3 \dots$$

where $a_i = 1$ if $(i, i) = 0$ and $a_i = 0$ if $(i, i) = 1$.

Define:

$$L_d = \{< M > \mid < M > \notin L(M)\}$$

Assume that L_d is recognizable by some TM N; i.e. $L_d = L(N)$. Now, the question becomes: is $< N > \in L_d$?

- If yes, $< N > \notin L_d$, hence $< N > \notin L_d$.
- If no, $< N > \in L_d$, hence $< N > \in L_d$.

Therefore, no such N is possible: L_d is not recognizable.

2.2 Recognizable but not decidable language

Define:

$$A_{TM} = \{< M, w > \mid M \; is \; a \; TM, \; and \; w \in L(M)\}$$

Assume A_{TM} is recognizable by a TM U (the "universal" TM).
$\underline{U \text{ on } < M, w >}$

- Check if M is a valid TM encoding.
- If so, evaluate M on w.
- If M accepts w, *accept*.
- If M rejects w, *reject*.

Note that U is not a decider. If M loops on w, U will also loop. Then $A_{TM} = L(U)$.

Theorem 2.1 *A_{TM} is not decidable.*

Proof by contradiction.

Assume that A_{TM} is decided by some TM H.

$H(< M, w >) = \begin{cases} accept, \ if \ w \in L(M) \\ reject, \ otherwise \end{cases}$

From H, we can construct another TM D that decides L_d:

$\underline{D \text{ on } < M >\text{: Is } < M > \in L_d?}$

- Run H on $< M, < M >>$.
- Accept if H rejects.
- Reject if H accepts.

We know that no such D exists since L_d is not recognizable, i.e. not decidable. Hence, no such H exists. A_{TM} is not decidable. ∎

Corollary 2.2 *$\overline{A_{TM}}$ is not Turing-recognizable.*

Proof We know that A_{TM} is Turing-recognizable. If $\overline{A_{TM}}$ were also Turing-recognizable, A_{TM} would be decidable. But we know that A_{TM} is undecidable, so $\overline{A_{TM}}$ must not be Turing-recognizable.