Problem Set #2 Solutions

-

Midterm Solutions

-

Problem (6):

Let M be a deterministic Turing machine: $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

We will show a construction of a deterministic Turing machine M' with only 3 states that simulates M.

Note that $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

The idea is to use a TM with 2 tapes, and use the 2nd tape to store the state M is at. So for any symbol $q \in Q$ we will have a symbol for it in Γ', that will indicate the state for the head of M' to be simulated.

We will define the transition function for M', δ', as follows. Let $Q' = \{q, q_{accept}, q_{reject}\}$ be the set of 3 states of M'. We will also assume we have a "stay" option in addition to left and right for M' - **why is that possible? ****

$\delta(q_0, a) = (q_1, b, R) \Rightarrow \delta'(q, (q_0, a)) = (q, (q_1, b), (S, R))$ – we "change" state by simply writing it down instead of the current state on the state tape and stay put for the next phase, and do the original move on the input tape.

**** why can we simulate a "stay" in M' using only 3 states**?

Because the queue tape start at the 0th position, so we can tell it to "go left". Due to the construction of TMs, when it tries to go left in the 0th position, it actually stays put. So replace "S" above with simply "L".

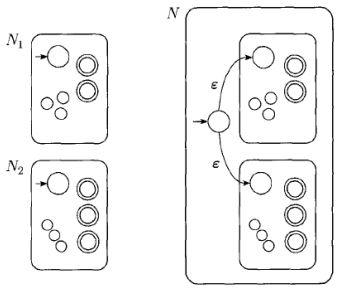
Alternative:

The simulating machine uses just one tape, but the alphabet $\Gamma' = Q \times \Gamma, \Sigma' = Q \times \Sigma$, which basically does the same – and the important thing is that we still have finite alphabets. But further look shows that this solution doesn't work because we have no way to remember what we wrote to one position on the tape as we advance forward. The use of an additional tape in the previous solution is our "memory" of what state we are carrying.

Question for Discussion:

What is the role/purpose of ϵ transitions in NFAs, and why did the author not introduce them for DFAs?

Answer:



To connect NFAs to allow showing that union, concatenation and the start operation of regular languages are regular.

For instance, if M_1 and M_2 are NFAs, we create the M that accepts $L(M_1) \cup L(M_2)$ (in the diagram to the left).

We have shown similar constructions for concatenation and star. It is simply easier to show that over showing that without ϵ transitions.

The reason we don't have ϵ in DFAs is because that would rob those machines of determinism. For instance, if at some DFA we're at state q that goes by a to state p, and we have another state r, then if we add an ϵ transition $p \xrightarrow{\epsilon} r$ then we get non-determinism since now q can move by a to both p and r.

Note: NFAs with ϵ transitions are equivalent to DFAs (without ϵ transitions).

For instance, if we have $q_1 \xrightarrow{a} q_2$ but also $q_1 \xrightarrow{\epsilon} q_3 \xrightarrow{\epsilon} q_4$, the equivalent DFA would have $\{q_1, q_3, q_4\} \xrightarrow{a} \{q_2\}$, where the sets correspond to the states of the DFA.

Decidable Problems (Textbook: 4.1)

We will look at problems as languages. For instance:

- A) Acceptance problems: does M accept w?
- E) Emptiness problems: is $L(M) = \emptyset$?
- EQ) Equality problems: Is $L(M) = L(N)$?

The first seems easier, as it concerns a certain word. Second and third seem to address all words.

Acceptance probelms

Theorem:

$A_{DFA} = \{< A, w > \mid A \text{ is a DFA that accepts } w\}$ is decidable.

Proof:

This is an example of notation for acceptance problems as languages, where A_{DFA} is the language of all pairs of some encoding of a DFA and an input word such that the word is accepted by the DFA.

The question asked: does a Turing machine (≈ algorithm) exists that decides those languages.

For A_{DFA} it's easy, our TM will simulate the DFA on the input word, and then accept if the simulation accepts, otherwise reject □

Theorem:

$A_{NFA} = \{< A, w > \mid A \text{ is an NFA that accepts } w\}$ is decidable.

Proof:

Our TM can first encode the given NFA as a DFA, and then simulate that constructed DFA on w, and accept only if the simulated DFA accepts □

Theorem:

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

Proof:

Given $< R, w >$, use a TM to convert the regular expression R into an equivalent NFA. Then apply the TM that decides whether the NFA accepts w □

Emptiness problems

Theorem:

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

Proof:

A is a DFA with a finite set of states, and we simply want to know whether there's a path from the start state to one of the accepting states. We do as follows:

- Starting at the start state, go over all states and mark all states that can be reached from it.
- When a state is marked, go back to the beginning.
- If there are no new marked states, we will get to the end of the input.

See example 3.23 in the book: $\{< G > \mid G \text{ is a connected undirected graph}\}$ - we simply apply the TM that decides this language onto the state graph of the DFA: is there a path from q_0 to an accepting state. If no, accept (because we're looking for empty languages), otherwise reject.

Equality problems

Theorem:

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

Proof:

A reduction to the emptiness problem: $L(A) = L(B) \Leftrightarrow L(A) \Delta L(B) = \emptyset$ - the symmetric difference of $L(A), L(B)$ is empty.

$\Leftrightarrow \big(L(A) - L(B)\big) \cup \big(L(B) - L(A)\big) **= \emptyset$.

We use the fact that regular languages are closed under union, therefore we can compute DFAs for $\overline{L(A)}, \overline{L(B)}$. They are also closed under union and intersection, therefore we can write the DFA of **, and by the previous theorem check whether the language accepted by that DFA is empty.

Theorem:

$A_{CFG} = \{< G, w > \mid G \text{ is a CFG that produces } w\}$ is decidable.

Proof:

We have proven that in Chomsky normal form, any word of length n is derived by exactly $2n - 1$ steps. Therefore we can use a TM that produces the tree of all possible derivations of $2n - 1$ steps where $|w| = n$, and check whether w is derived by any of those. Note: this way we avoid possible cycles in G.

First of course we need to transform G into G' of Chomsky normal form, and that can be done according to the proof that any G has a Chomsky normal form equivalent grammar G'.

Theorem:

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

Proof:

As before this is a reachability problem: is there a path (derivation) from the start symbol to a terminal in the graph of G's derivation rules? We mark all terminal symbols, and then check for each non-terminal symbol whether it can reach some

terminals – if so, mark it. We do that as long as we keep marking new non-terminals. When no further changes are made, we check whether the start symbol is mark. If no – we accept (again – here we check emptiness). Otherwise, we reject.

Theorem:

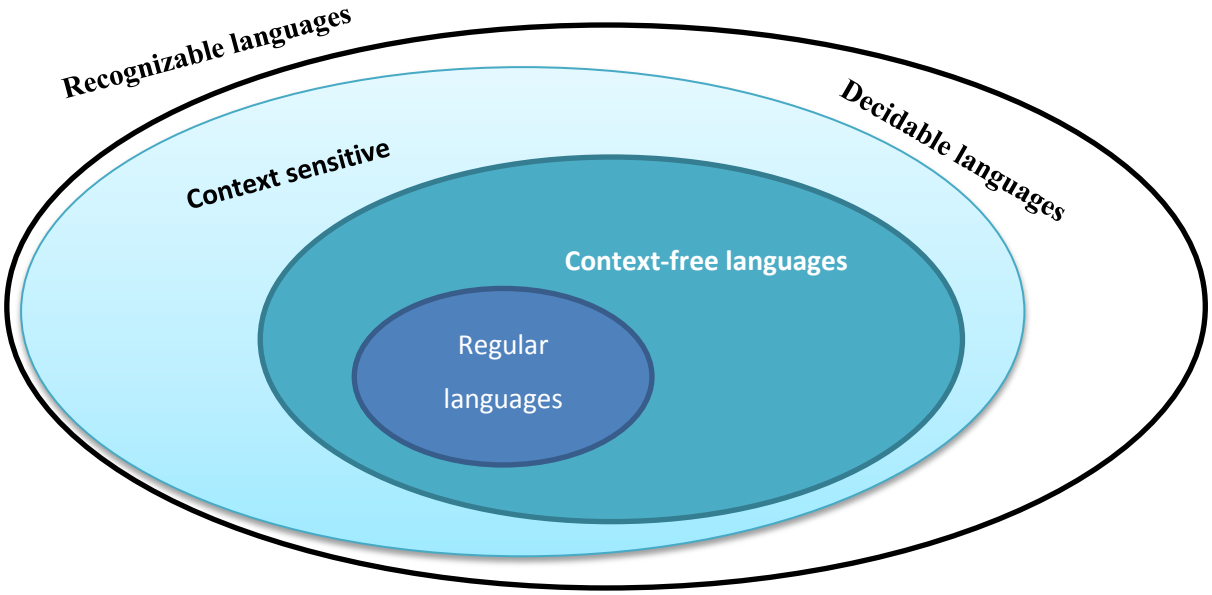
$EQ_{CFG} = \{< G, F > \mid G, F \text{ are CFGs and } L(G) = L(F)\}$ is **not decidable**.

Note:

Note: We can't use the symmetric difference as before, as we counted on the property of regular languages that is does not apply on CFLs: closure under complement (therefore $\overline{L(G)}, \overline{L(F)}$ are not necessarily CFLs).

This doesn't prove that EQ_{CFG} is not decidable, but that we will see in chapter 5.

Chomsky Hierarchy



Context-free languages: $\alpha A \beta \rightarrow \alpha u \beta$ where $\alpha, u, \beta \in (N \cup T)^*$ and $|\alpha A \beta| \leq |\alpha u \beta|$.

Decidable languages: languages that always decide for any input whether it's in the language or not.

Recognizable languages: languages that accept inputs in the language and may loop forever on inputs not in the language.