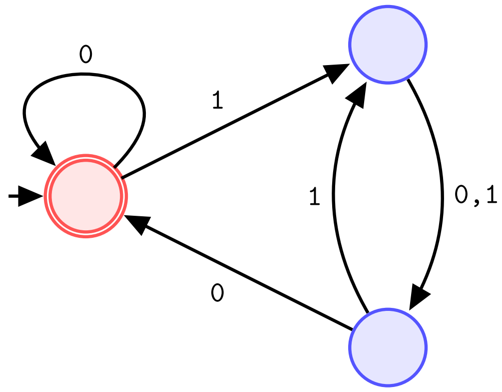
(1) Answer all parts for the following DFA M and give reasons for your answers.



1) Is $\langle M, \texttt{0100}\rangle \in A_{\text{DFA}}$?

2) Is $\langle M, \texttt{011}\rangle \in A_{\text{DFA}}$?

3) Is $\langle M\rangle \in A_{\text{DFA}}$?

4) Is $\langle M\rangle \in E_{\text{DFA}}$?

5) Is $\langle M, M\rangle \in EQ_{\text{DFA}}$?

Solution

Recall that

$$\begin{aligned}
A_{\text{DFA}} &= \{\langle B, w\rangle \mid B \text{ is a DFA that accepts the input string } w\} \\
E_{\text{DFA}} &= \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\} \\
EQ_{\text{DFA}} &= \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}
\end{aligned}$$

1) Is $\langle M, \texttt{0100}\rangle \in A_{\text{DFA}}$? Yes. 0100 is accepted by M.

2) Is $\langle M, \texttt{011}\rangle \in A_{\text{DFA}}$? No. 011 is **not** accepted by M.

3) Is $\langle M\rangle \in A_{\text{DFA}}$? No. Invalid encoding.

4) Is $\langle M\rangle \in E_{\text{DFA}}$? No. $L(M) = \{\varepsilon, \texttt{110}, \texttt{100}, \texttt{0100}, \dots\} \neq \emptyset$.

5) Is $\langle M, M\rangle \in EQ_{\text{DFA}}$? Yes. M and itself accept the same language.

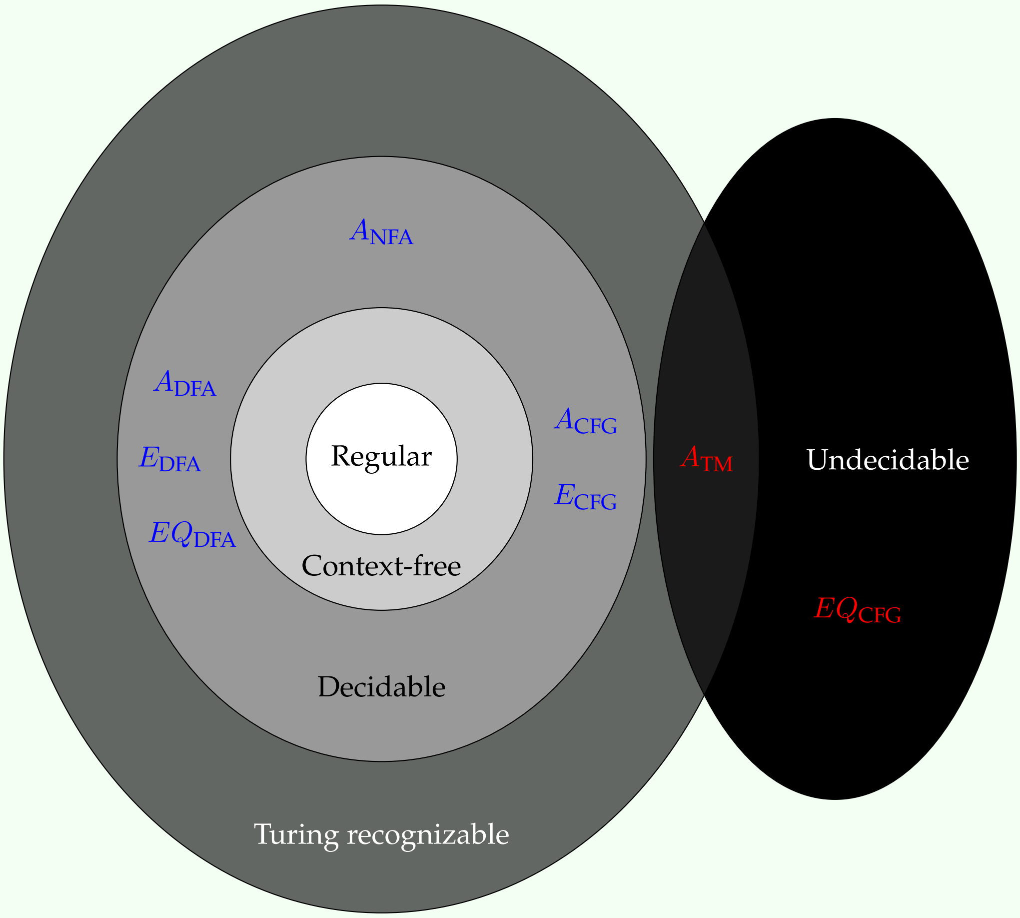
(2) Draw a Venn diagram to illustrate the relationship between the languages: *regular*, *context-free*, *decidable*, and *Turing-recognizable*; then indicate where the following problems belong to. (Refer to the lecture slides.)

1. Acceptance problems
   i. $A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$
   ii. $A_{\text{NFA}} = \{\langle B, w\rangle \mid B \text{ is an NFA that accepts input string } w\}$
   iii. $A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$
   iv. $A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$
2. Language emptiness problems
   i. $E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
   ii. $E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$
3. Language equality problems

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

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

Solution



TODO: EQ_{CFG} recognizable?

(Whatever applies to DFAs also applies to NFAs because they are equivalent.)

(3) Read about the *Post Correspondence Problem (PCP)* on Wikipedia (http://en.wikipedia.org/wiki/Post_correspondence_problem).

Is it decidable? How about when the alphabet is simply $\{\mathtt{a}\}$?

Find some (easy) examples and try to solve them by hand, e.g.

$$\left\{ \left[\frac{\mathtt{ab}}{\mathtt{abab}}\right], \left[\frac{\mathtt{b}}{\mathtt{a}}\right], \left[\frac{\mathtt{aba}}{\mathtt{b}}\right], \left[\frac{\mathtt{aa}}{\mathtt{a}}\right] \right\}$$

Try to write code to search for solutions using brute force search. You may want to first have a look at: http://code.google.com/p/post-correspondence-brute/

Solution

PCP is undecidable.

The special case where the alphabet consists of only one symbol is decidable.

The idea is as follows:

1) All tiles that have equal top and bottom strings can be ignored.

2) If all remaining tiles have top strings longer than bottom strings then the answer is False.

3) If all remaining tiles have top strings shorter than bottom strings then the answer is False.

4) If less than two dominoes are left then we have a trivial case to check.

5) Pick two dominoes $[\frac{x}{y}]$ and $[\frac{v}{w}]$ such that x is longer than y, and v is shorter than w.

Here is a sample Python script that could be used to try and find a solution to a given PCP instance. It works by trying all the possible combinations of length at most 20 tiles. Of course there may be a solution which requires longer combinations, or maybe the answer is False – code cannot rule out this case.

```
from itertools import product

tiles = [
    ("ab","abab"),
    ("b","a"),
    ("aba","b"),
    ("aa","a"),
]

def show_tile(tiles):
    ''' Pretty print the tiles combination '''
    top = bottom = middle = ''
    for tile in tiles:
        width = max( len(tile[0]),len(tile[1]) )
        top    += tile[0].center(width) + ' '
        bottom += tile[1].center(width) + ' '
        middle += ('-'*width) + ' '
    print(top)
    print(middle)
    print(bottom)
    print()

# Search all possible combinations up to length = 20
for length in range(1,21):
    for comb in product(tiles, repeat=length):
        top = bottom = ''
        for tile in comb:
            top    += tile[0]
            bottom += tile[1]
        if top == bottom:
            show_tile(comb)
```

Here are some solutions found by the above script:

$$\left[\frac{\mathtt{aa}}{\mathtt{a}}\right]\left[\frac{\mathtt{aa}}{\mathtt{a}}\right]\left[\frac{\mathtt{b}}{\mathtt{a}}\right]\left[\frac{\mathtt{ab}}{\mathtt{abab}}\right] \to \mathtt{aaaabab}$$

$$\left[\frac{\mathtt{ab}}{\mathtt{abab}}\right]\left[\frac{\mathtt{ab}}{\mathtt{abab}}\right]\left[\frac{\mathtt{aba}}{\mathtt{b}}\right]\left[\frac{\mathtt{b}}{\mathtt{a}}\right]\left[\frac{\mathtt{b}}{\mathtt{a}}\right]\left[\frac{\mathtt{aa}}{\mathtt{a}}\right]\left[\frac{\mathtt{aa}}{\mathtt{a}}\right] \to \mathtt{abababbbaaaa}$$

$$\left[\frac{\texttt{aa}}{\texttt{a}}\right]\left[\frac{\texttt{aa}}{\texttt{a}}\right]\left[\frac{\texttt{b}}{\texttt{a}}\right]\left[\frac{\texttt{ab}}{\texttt{abab}}\right]\left[\frac{\texttt{aa}}{\texttt{a}}\right]\left[\frac{\texttt{aa}}{\texttt{a}}\right]\left[\frac{\texttt{b}}{\texttt{a}}\right]\left[\frac{\texttt{ab}}{\texttt{abab}}\right] \to \texttt{aaaababaaaabab}$$

(4) Let $AMBIGCFG = \{\langle G\rangle \mid G \text{ is an ambiguous CFG}\}$.

Show that $AMBIGCFG$ is undecidable.

Hint: Use a reduction form PCP (above). Given an instance

$$\left\{\left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \dots, \left[\frac{t_k}{b_k}\right]\right\}$$

of the Post Correspondence Problem, construct a CFG G with the rules:

$$\begin{array}{rcl} S & \to & T \mid B \\ T & \to & t_1Ta_1 \mid \cdots \mid t_kTa_k \mid E \\ B & \to & b_1Ba_1 \mid \cdots \mid b_kBa_k \mid E \\ E & \to & \varepsilon \end{array}$$

where $a_1, a_2, \dots, a_k$ are new terminal symbols.

Solution

Let us *reduce* PCP to $AMBIGCFG$.

Let:

$$\left\{\left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \dots, \left[\frac{t_k}{b_k}\right]\right\}$$

be a given instance of PCP.

Let us first consider the following grammar G designed to mimic the juxtaposition of the tiles (T for the top patterns, and B for the bottom patterns):

$$\begin{array}{rcl} S & \to & T \mid B \\ T & \to & t_1Ta_1 \mid \cdots \mid t_kTa_k \mid \varepsilon \\ B & \to & b_1Ba_1 \mid \cdots \mid b_kBa_k \mid \varepsilon \end{array}$$

where $a_1, a_2, \dots, a_k$ are new terminal symbols used to keep track of which tiles were used.

When a string is generated it will be of the form $t_it_jt_k \cdots a_ka_ja_i$ (if we start with $S \to T$) or $b_ib_jb_k \cdots a_ka_ja_i$ (if we start with $S \to B$). If these two strings are equal then we know they have been generated in two different ways, and so this grammar must be ambiguous.

There is, however, a slight problem with this grammar as it allows the generation of ε which corresponds to the empty solution of PCP. To fix this we change G as follows (to ensure that at least one tile is used):

$$\begin{array}{rcl} S & \to & T \mid B \\ T & \to & t_1Ta_1 \mid \cdots \mid t_kTa_k \mid t_1a_1 \mid \cdots \mid t_ka_k \\ B & \to & b_1Ba_1 \mid \cdots \mid b_kBa_k \mid b_1a_1 \mid \cdots \mid b_ka_k \end{array}$$

Now, if $AMIGCFG$ were decidable then we would map the given PCP instance to this new grammar G then use $AMIGCFG$'s decider to decide if G is a ambiguous or not. If the answer is True then the PCP instance is also solvable, otherwise it is not. But since PCP is not decidable then such a decider cannot exist.

We conclude that $AMIGCFG$ is not decidable.

"A *quine* is a computer program which takes no input and produces a copy of its own source code as its only output. The standard terms for these programs in the computability theory and computer science literature are *self-replicating programs*, *self-reproducing programs*, and *self-copying programs*." `http://en.wikipedia.org/wiki/Quine_(computing)`

Write a quine in your preferred programming language.