CS525 Winter 2012 \ Chapter #4 Preparation

Ariel Stolerman

Problems

4.10)

Let $INF_{PDA} = \{\langle M \rangle \mid M\ is\ a\ PDA\ and\ L(M)\ is\ an\ infinite\ language\}$. Following is a proof that INF_{PDA} is decidable:

A context-free language is infinite if there exists a cycle within its derivation rules. For PDAs, we can construct a CFG corresponding to any given PDA and test it. Therefore we can construct a Turing machine N that given the input M does as follows:

- Check if the input M is a valid encoding of a PDA. If not, *reject*.
- Create G a CFG that is equivalent to M, i.e. $L(M) = L(G)$, and convert G to Chomsky normal form.
- Look for a cycle in the grammar's rules in **BFS** (in order to avoid infinite loops) such that at any iteration on the cycle the generated string is pumped (i.e. the cycles describes a derivation of the form $R \overset{*}{\rightarrow} aRb$ where $|ab| > 0$ and a, b are terminals).
- If found such cycle, *accept*. Otherwise, *reject*.

4.12)

Let $A = \{\langle R, S \rangle \mid R\ and\ S\ are\ regular\ expressions, L(R) \subseteq L(S)\}$. Following is a proof that A is decidable. We will show a Turing machine M that decides A:

- Check that R, S are proper regular expressions, otherwise *reject*.
- Construct a NFA A' from the regular expression R (such that $L(A) = L(R)$), and then a DFA A from A'.
- Construct a NFA B' from the regular expression S (such that $L(B) = L(S)$), and then a DFA B from B'.
- Construct a DFA C that recognizes $L(A) \cap \overline{L(B)}$.
- Simulate the TM from the book that decides E_{DFA} on C. If it accepts, *accept*. Otherwise, *reject*.

Note that if $L(R)$ is not fully contained within $L(S)$ then $\exists w \in L(R) \wedge w \notin L(S) \Rightarrow w \in L(R) \cap \overline{L(S)}$. Furthermore, the construction of the NFAs and DFAs can be done using a Turing machine, and the intersection of regular languages is a regular language, so we can construct a DFA for it. Thus if the intersection above is discovered to be empty, $L(A)$ must be fully contained in $L(B)$, and so $L(R)$ is fully contained in $L(S)$.

4.15)

Let $A = \{\langle R \rangle \mid R\ is\ a\ regular\ expression, \exists w \in L(R)\ s.t. w = x111y\ for\ some\ x, y \in \Sigma^*\}$. Following is a proof that A is decidable. We will construct a Turing machine M that decides A as follows:

- Check that R is a proper regular expression, otherwise *reject*.
- Construct a DFA A from the regular expression R (such that $L(A) = L(R)$).

- Construct a DFA B that recognizes the language of the regular expression $\Sigma^* \circ \{111\} \circ \Sigma^*$.
- Construct a DFA C that recognizes the language $L(A) \cap L(B)$.
- Simulate the TM from the book that decides E_{DFA} on C. If it accepts, *reject*. Otherwise, *accept*.

Clearly if $L(R)$ contains some string that contains 111, its intersection with the language of all strings that contain 111 should be non-empty. Thus simulating the Turing machine that decides whether that intersection is empty and returning the opposite answer is correct.

4.16)

Following is a proof that EQ_{DFA} is decidable by testing all DFAs on two strings up to a certain size, and that size as a function of the definitions of the two input DFAs. Let $EQ_{DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs and } L(A) = L(B)\}$. Let M be a Turing machine that decides EQ_{DFA} and defined as follows:

- Verify the input $\langle A, B\rangle$ describes 2 valid DFAs A and B with the same alphabet Σ. If not, *reject*.
- Calculate $n = |Q_A|$ and $m = |Q_B|$ the number of states in each of the DFAs.
- Enumerate all strings in Σ up to length $n \cdot m$, and for each such string w:
  - Simulate A on w
  - Simulate B on w
  - If the result of the two simulations is different, *reject*. Otherwise continue.
- If got here (after all $n \cdot m$ strings), *accept*.

The reason we can check only the first $n \cdot m$ strings is that if the 2 DFAs do not accept the same language, there must be a string w of size $|w| \le n \cdot m$ for which $A(w) \ne B(w)$. Assume by contradiction that the first string that yields a different output of A and B is w' and $|w'| = l > n \cdot m$, then there is a sequence of states $a_0, a_1, \dots, a_l \in Q_A$ and $b_0, b_1, \dots, b_l \in Q_B$ that describe the transitions for w' in A and B respectively. Since $l > n \cdot m$, putting those sequences side by side, there must be some repetition of a pair of sequences a_i, b_i and a_j, b_j such that $a_i = a_j, b_i = b_j, i < j$. Therefore we can remove all subsequences in between leaving only a_i, b_i and by that get a smaller string w'' that A, B will act the same over exactly as over w'. We can "pump" down until receiving a string of length $\le n \cdot m$, thus contradicting the assumption – as there we have found a string w'' with length $|w''| \le n \cdot m$ such that $A(w'') \ne B(w'')$. Therefore checking all strings up to size $n \cdot m$ is sufficient.

4.18)

Let A, B be two disjoint co-Turing-recognizable languages. Since A is co-Turing-recognizable, then there exists a Turing machine M_1 that recognizes $\bar{A}$. In the same manner, there exists a Turing machine M_2 that recognizes $\bar{B}$. Since A, B are disjoint then $A \cap B = \emptyset$, therefore $\overline{A \cap B} = \bar{A} \cup \bar{B} = \Sigma^*$. We will construct the following TM M:

- Simulate M_1 and M_2 on the input string w alternatively. If M_1 accepts, *reject*. Otherwise (if M_2 accepts), *accept*.

Since the simulation of M_1 is interleaved with the simulation of M_2, and $L(M_1) \cup L(M_2) = \Sigma^*$, the simulation will always have a finite number of steps and get to an accepting / rejecting state. For all $w \in \Sigma^*$, if $w \in A$ then $w \notin B \Rightarrow w \notin \bar{A}, w \in \bar{B}$

thus M_1 will never accept, and M_2 will, so M accepts, thus $w \in A \Rightarrow w \in L(M)$. In a similar manner, if $w \in B$, then M rejects, thus $w \in B \Rightarrow w \notin L(M) \Rightarrow w \in \overline{L(M)}$. Therefore we have found a decidable language $L(M)$ (since M always halts with a decision) and $A \subseteq L(M), B \subseteq \overline{L(M)}$, as required.

4.19)

Let $S = \{\langle M\rangle \mid M \textit{ is a DFA and } w \in L(M) \Rightarrow w^R \in L(M)\}$. We will show S is decidable by constructing a Turing machine M as follows:

- Check that M is a proper DFA, otherwise *reject*.
- Construct a DFA M^R that recognizes $\{w \mid w^R \in L(M)\}$ (detailed later).
- Simulate the Turing machine from the book that decides EQ_{DFA} on the input $\langle M, M^R\rangle$. If it accepts, *accept*. Otherwise, *reject*.

We can build the DFA M^R by first constructing an NFA from M by reversing all transitions, making the previous start state the only new accepting state, and creating a new start state with ϵ transitions to all previous accepting state (that now should not accept). Then a DFA can be constructed from this NFA.

4.20)

Let $PREFIX - FREE_{REX} = \{R \mid R \textit{ is a regular expression and } L(R) \textit{ is prefix free}\}$. We prove $PREFIX - FREE$ is decidable by constructing a Turing machine M that decides it as follows:

- Check R is a proper regular expression, otherwise *reject*.
- Construct a DFA A from the input regular expression R (such that $L(R) = L(A)$).
- Construct a DFA B that recognizes $L(A) \circ \Sigma^+$ (where Σ is the alphabet of A).
- Construct a DFA C that recognizes $L(A) \cap L(B)$.
- Simulate the TM from the book that decides E_{DFA} on the input $\langle C\rangle$. if it accepts, *accept*. Otherwise, *reject*.

The idea is that if R generates a prefix-free language, then any string w generated by R cannot be a prefix of any other string generated by R. The language of all strings that have any w that is generated by R as a prefix is $L(R) \circ \Sigma^+ = \{ab \in \Sigma^+ \mid a \in L(R), b \in \Sigma^+\}$, thus if the intersection of that language with $L(R)$ is not empty, it means $L(R)$ contains some string(s) with a proper prefix from $L(R)$ itself. Therefore checking whether that intersection is empty checks the required condition of $PREFIX - FREE_{REX}$.

A similar approach for showing $PREFIX - FREE_{CFG}$ is decidable will fail since context-free languages are not closed under intersection, thus the construction of C cannot fit when applied to context-free languages.

4.22)

Let $L = \{\langle M\rangle \mid M \textit{ is a PDA that has a useless state}\}$. Following is a proof that L is decidable. We will construct a Turing machine M that decides L as follows:

- Check that M is a proper PDA, otherwise *reject*.

- For any state q in M:
  - Mark q as the only accepting state, and denote that PDA as M'.
  - Use the Turing machine that decides E_{PDA} on M'. If it accepts – *accept*.
- If got to this point, *reject*.

Clearly if marking any state q as the only accepting state, and the language recognized by that variant is empty, then there exists a useless state in the input PDA M.

4.26)

Let $C = \{\langle G, x\rangle |\ G\ is\ a\ CFG\ that\ generates\ some\ string\ w, where\ x\ is\ a\ substring\ of\ w\}$. Following is a proof that C is decidable. We will construct a Turing machine M that decides C as follows:

- Construct a DFA A that recognizes that language of the regular expression $\Sigma^* \circ \{x\} \circ \Sigma^*$ (all strings with x as their substring).
- Construct a CFG F for the context-free language $L(G) \cap L(A)$ (which is also context-free).
- Simulate the Turing machine that decides E_{CFG} on $L(F)$. If it accepts, *reject*. Otherwise, *accept*.

We know that the language that is an intersection of a CFL and a regular language is also a CFL, therefore F will be a CFG. Moreover, $L(A)$ is the language of all strings with x as their substring, and is a regular language (described above ina a regular expression). Therefore if G generates some string w with x as its substring, the intersection, $L(F)$, should be non-empty.

4.27)

Let $C_{CFG} = \{\langle G, k\rangle |\ L(G)\ contains\ exactly\ k\ strings\ where\ k \geq 0\ or\ k = \infty\}$. Following is a proof that C_{CFG} is decidable. We will construct a Turing machine M that decides C_{CFG} as follows:

- Check that G is a proper CFG and that $k \geq 0$ or $k = \infty$. If not, *reject*.
- Construct a PDA P from G, and run $INF_{PDA}(P)$ (from exercise 4.10).
- If $k = \infty$ and $INF_{PDA}(P)$ accepted, *accept*.
- If $k \neq \infty$ and $INF_{PDA}(P)$ rejected, *reject*.
- Otherwise ($k \neq \infty$ and INF_{PDA} rejected), calculate the pumping length p of $L(G)$ (can be calculated from the number of variables in the Chomsky normal form of G), initialize a counter to 0 and iterate over all strings w in of size $0,1, \dots, p-1$:
  - Simulate P on w. If it accepts, increase the counter by 1.
- At the end, if the counter $= k$, *accept*. Otherwise, *reject*.

We have proven that INF_{PDA} is decidable, and it is TM possible to construct a PDA from a given CFG. Therefore if $k = \infty$ it is easy to check if the input is in the C_{CFG}. Otherwise, we know that any string in the language is of length at most p (excluding), otherwise it could be pumped forever and the $L(G)$ would be infinite, which we already know is not true.

Therefore it is sufficient to check this finite number of strings, and count whether there are exactly k strings of them that are accepted by G (that is, its corresponding PDA P).