1. [10 points] (after Sipser 2.11) Convert the CFG of Exercise 2.1 to an equivalent PDA, using the procedure given in Theorem 2.20, or equivalently, the Produce-Match method described in class. The CFG is given below and E is the start symbol.

\[
\begin{align*}
E & \rightarrow E + T \mid T \\
T & \rightarrow T * F \mid F \\
F & \rightarrow (E) \mid a
\end{align*}
\]

Present the transitions for your PDA in a tabular form with five columns:

\[
\text{current state, symbol read, stack popped, next state, stack pushed}
\]

Where any of \{symbol read, stack popped, stack pushed\} can be \(\varepsilon\), the empty string. In the case of the stack popped and pushed, make the top of the stack be the leftmost letter. (Using a spreadsheet or table is recommended.)

2. [10 points] Using your PDA in problem 1, trace the behavior on the input string \(a^*(a+a)\).

3. [15 points] (Sipser 4.30) Let A be a Turing-recognizable language consisting of descriptions of Turing machines, \(\{\langle M_1 \rangle, \langle M_2 \rangle, \ldots\}\), where every \(M_i\) is a decider. Prove that some decidable language \(D\) is not decided by any decider \(M_i\) such that \(\langle M_i \rangle \in A\). (Hint: You may find it helpful to consider an enumerator for \(A\).) Further hint: Identify the set of all strings over the alphabet of \(A\) with the natural numbers, enabling the use of a string as an index into \(A\).

4. [10 points] Determine whether or not language \(WNB = \{\langle M \rangle \mid M \text{ writes a non-blank symbol when started on a blank tape}\}\) is decidable. Assume the tape alphabet is \{blank, 0, 1\}. Prove your answer.

5. [10 points] Let \(\text{INFINITE}_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is an infinite language}\}\). Is decidable, recognizable, corecognizable, or neither? Justify your answers.

6. [25 points] Present a proof by example of Church’s Theorem:

There is no algorithm for determining whether or not an arbitrary first-order predicate calculus formula is valid.

We know from class and the textbook that there is no algorithm that will determine whether an arbitrary Turing machine will reach a halting state when started on an arbitrary tape. So one way to prove Church’s theorem is to show that
any Turing machine and its tape can be translated into a set of clauses, such that
the set of clauses is unsatisfiable iff the machine halts on the tape. (One could
even construct a program that does this translation.)

An acceptable proof by example in the present case is to show how to translate
any TM into a set of clauses with the desired property. For full generality, the
example TM should use more tape than required for the initial input. Good
elements of machines that use extra tape are found in the Busy Beaver family of
machines BB(n), where n ≥ 1 is the number of states in the machine’s control (not
counting the Halting state), and the tape alphabet is limited to \{a, b\}, with b
representing blank. The definition of the best BB(n) is that it is the n-state
machine producing the most a’s when started on an all blank tape. [Normally the
Busy Beaver problem uses 0 and 1 for tape symbols (with 0 for blank) but I
encode it this way to simplify the next problem.]

Transitions for BB(4) are shown below. A is the initial state. The special state H
indicates halting and does not count as one of the four states.

BB(4):

<table>
<thead>
<tr>
<th>Transition</th>
<th>Clausal Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, b → B, a, R</td>
<td>B, b → A, a, L</td>
</tr>
<tr>
<td>A, a → B, a, L</td>
<td>B, a → C, b, L</td>
</tr>
<tr>
<td>C, b → H, a, R</td>
<td>C, a → D, a, L</td>
</tr>
<tr>
<td>D, b → D, a, R</td>
<td>D, a → A, b, R</td>
</tr>
</tbody>
</table>

The problem is to translate such a set of TM transitions into clauses, and add other
general clauses, such that the set of clauses is unsatisfiable iff the TM halts on an
all blank tape. The clauses will thus need to be able to simulate the TM’s actions
as resolution steps.

The key to constructing the clauses is to use two logical terms to represent the
tape: One term represents the tape to the left of the TM head in reverse order, and
a second term represents the tape beneath and to the right of the TM head. For
example, if the two tape segments are:

```
| a b b a a |
| la l b a a a |
```

with the vertical bars showing the position of the TM head, then the logical terms
would be a(a(b(b(a(d))))) and a(b(a(a(a(a(d))))) where d is a dummy constant symbol
representing the boundary of the tape in use.

Your translation should be able to use one clause per each right-moving transition
and two clauses per each left-moving transition. You will need to have special
classes that add a new blank cell to either end of the tape.

A similar example of this two-term representation is the pegs puzzle in the slides
on Resolution Theorem Proving (171-179).
7. [20 points] Submit a demonstration of the translation in the previous problem in
the clausal form required by the Prover9 theorem prover. Prover9 should be able
to show that your clauses are unsatisfiable.

Also submit a Prover9 run on the following incorrect version of BB(4). Prover9
should not be able to show these clauses are unsatisfiable, as the machine does not
halt.

Incorrect BB(4):

<table>
<thead>
<tr>
<th>Clause 1</th>
<th>Clause 2</th>
<th>Clause 3</th>
<th>Clause 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A, b \rightarrow B, a, R$</td>
<td>$B, b \rightarrow A, a, L$</td>
<td>$C, b \rightarrow H, a, R$</td>
<td>$D, b \rightarrow D, a, R$</td>
</tr>
<tr>
<td>$A, a \rightarrow B, a, L$</td>
<td>$B, a \rightarrow C, b, L$</td>
<td>$C, a \rightarrow D, a, L$</td>
<td>$D, a \rightarrow D, b, R$</td>
</tr>
</tbody>
</table>