

Exercise 1: Unrestricted Grammars

a) The language accepted is  $\{a^n b^n c^n \mid n \geq 1\}$ .

b) One of the many solutions:

**S**  $\rightarrow$  **aTb** |  $\epsilon$   
**T**  $\rightarrow$  **aBTaB** | **ba**  
**Ba**  $\rightarrow$  **aB**  
**Bb**  $\rightarrow$  **bb**

One possible derivation of the string `aaaabbbbbaaaabbbb`:

**S**  $\Rightarrow^*$  **aaBaBaBTaBaBaBb**  $\Rightarrow$  `aaBaBaBbaaBaBaBb`  
 $\Rightarrow^*$  `aaaaaBBBbaaaaBBBb`  $\Rightarrow^*$  `aaaaBBBbaaaabbbb`  
 $\Rightarrow^*$  `aaaabbbbaaaabbbb`

The following solution is less compact, but probably easier to understand. The non-terminals of the sentential form are converted to terminals from left to right. **C**'s represent **a**'s and **D**'s represent **b**'s:

**S**  $\rightarrow$  **aSBCD** |  $\epsilon$   
**DB**  $\rightarrow$  **BD**  
**DC**  $\rightarrow$  **CD**  
**CB**  $\rightarrow$  **BC**  
**aB**  $\rightarrow$  **ab**  
**bB**  $\rightarrow$  **bb**  
**bC**  $\rightarrow$  **ba**  
**aC**  $\rightarrow$  **aa**  
**aD**  $\rightarrow$  **ab**  
**bD**  $\rightarrow$  **bb**

Note:

The language  $L = \{a^n b^n a^n b^n \mid n \geq 0\}$  is actually context-sensitive. Any context-sensitive language has a grammar made of rules  $u \rightarrow v$ , where  $u \in (\Sigma \cup V)^* V (\Sigma \cup V)^*$  and  $v \in V^*$ , and  $|u| \leq |v|$ . You were not asked specifically to create a context-sensitive grammar.

## Exercise 2: Decidability

- a. **FINITETM =  $\{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is finite}\}$  is undecidable.**

There are different ways to solve this problem, here is one of them: we let  $F$  be a TM that decides FINITETM and construct a TM  $S$  to decide  $A_{TM}$ . Then,  $S$  works the following manner:

$S =$  "On input  $\langle M, w \rangle$ , where  $M$  is a TM and  $w$  is a string:

1. Construct the following TM  $M'$ .

$M' =$  "On input  $x$ :

1. Run  $M$  on input  $w$  and *accept* if  $M$  accepts  $w$ , otherwise *reject*."
2. Run  $F$  on input  $\langle M' \rangle$ . If  $F$  accepts, *reject*. If  $F$  rejects, *accept*.

Clearly, if  $M$  accepts  $w$ ,  $M'$  accepts its input  $x$  (whatever it is). On the other hand, if  $M$  rejects  $w$ ,  $M'$  rejects its input  $x$  (whatever it is). Therefore,  $M'$  recognizes  $\Sigma^*$  if  $M$  accepts  $w$ , and  $M'$  recognizes  $\emptyset$  if  $M$  rejects  $w$ . Since  $\Sigma^*$  is infinite and  $\emptyset$  is finite,  $F$  accepts its input if  $M$  rejects  $w$ , and rejects it if  $M$  accept  $w$ .

- b.  **$L_1 = \{\langle M \rangle \mid M \text{ is a TM and there is at least one input string for which } M \text{ halts}\}$  is undecidable.**

We let  $H$  be a TM that decides  $L_1$  and construct a TM  $S$  to decide  $A_{TM}$ . Then,  $S$  works the following manner:

$S =$  "On input  $\langle M, w \rangle$ , where  $M$  is a TM and  $w$  is a string:

1. Construct the following TM  $M'$ .

$M' =$  "On input  $x$ :

1. If  $x \neq w$ , enter a loop.
1. If  $x = w$ , run  $M$  on  $w$ . If  $M$  accepts  $w$ , *accept*. Otherwise, enter a loop.
2. Run  $H$  on input  $\langle M' \rangle$ . If  $F$  accepts, *accept*. If  $F$  rejects, *reject*.

$M'$  is built in such a way that it halts in only one case: on input  $w$ , if  $M$  accepts  $w$ . In any other case (if  $M$  rejects  $w$ , or if the input itself is not  $w$ ),  $M'$  loops. Therefore, if  $H$  accepts  $\langle M' \rangle$  (that is,  $M'$  halts on some input), it can only mean that  $M$  accepts  $w$ . On the other hand, if  $H$  rejects  $\langle M' \rangle$  (that is,  $M'$  loops for all inputs), it can only mean that  $M$  did reject  $w$ .

- c.  $L_2 = \{\langle M, w, k \rangle \mid M \text{ accepts } w \text{ and never moves its head past the } k \text{ first positions of its tape}\}$  is decidable.

Since  $M$  isn't allowed to go past the  $k$ -th position,  $M$  is similar to an LBA (see Sipser p.178). It is a TM that has a tape of fixed size. On such a machine, the number of configurations is finite: there can be only  $kng^k$  different configurations, where  $k$  is the tape size,  $n$  is the number of states, and  $g$  is the size of the alphabet. Therefore, any computation on such a device either halts in the first  $kng^k$  steps or *loops*. The reason for this is that, after  $kng^k$  steps, at least one configuration would be repeated, and  $M$  would go on to repeat that configuration over and over again and never halt.

$L_2$  has the following decider:

$S =$  "On input  $\langle M, w, k \rangle$ , where  $M$  is a TM,  $w$  a string, and  $k$  an integer:

1. Simulate  $M$  on  $w$  during no more than  $kng^k$  steps. If  $M$  ever goes past the  $k$ -th position or if it rejects, *reject*. Otherwise, if it accepts, *accept*.
2. If the simulation is ended, but  $M$  didn't halt, *reject*."

- d.  $L_3 = \{\langle M, w, t \rangle \mid M \text{ accepts } w \text{ in at most } t \text{ steps}\}$  is decidable.

The idea is that we can interrupt a simulation of  $M$  on input  $w$  after only  $t$  steps.  $L_3$  has the following decider:

$S =$  "On input  $\langle M, w, t \rangle$ , where  $M$  is a TM,  $w$  a string, and  $t$  an integer:

1. Simulate  $M$  on input  $w$  during no more than  $t$  steps.
2. If  $M$  accepted during the simulation, *accept*. If it rejected, *reject*. If  $M$  didn't either accept or reject after  $t$  steps, *reject*."

### Exercise 3: Sipser 3.10 (write-once Turing machine)

First, we simulate a standard TM on a write-twice TM. The write-twice TM simulates a single step of the original TM by copying the entire active portion of the tape over to a fresh section of the tape to the right of the currently used section. A marker (e.g. #) that isn't part of the original tape alphabet is used to separate the original section from the copied section. The copying procedure operates character by character, altering each tape square twice: once to write the character the first time and another time to cross it off once it is duplicated to a fresh section of the tape (we have to cross it off, or mark this symbol as "copied" during the duplication procedure, otherwise it wouldn't be possible to know what has already been copied and what has not). The position of the original TM tape head is marked on the tape. When copying the cells around the marked position, the tape contents is updated according to the rules of the TM.

Simulation on a write-once TM. We operate as with a write-twice, except that every tape square of the write-twice TM is represented as two squares on the write-once TM. The first of these contains the same content as the write-twice TM, except that its symbol is never crossed off. The second of these two squares is used to mark if the symbol has already been copied. Note: the input string is not in the two-cell format when the computation starts, so the very first time the tape is copied, the copying marks (crossings) are put directly over the input symbols (instead of next to them).

### Exercise 4: Sipser 4.3 (ALL<sub>DFA</sub>)

Let  $ALL_{DFA} = \{ \langle A \rangle \mid A \text{ is a DFA that recognizes } \Sigma^* \}$ . Since we know that  $E_{DFA}$  is decidable, we can build a TM **E** that decides it. The following TM **R** decides  $ALL_{DFA}$ :

**R** = "On input  $\langle A \rangle$ , where **A** is a DFA:

1. Construct a DFA **B** that recognizes  $\overline{L(A)}$ , by swapping accept and non-accepting states.
2. Run the TM **E** on input  $\langle B \rangle$ , where **E** is the decider of  $E_{DFA}$ .
3. If **E** accepts, *accept*. If **E** rejects, *reject*."

### Exercise 5: Sipser 4.11

Let  $A = \{ \langle R, S \rangle \mid R \text{ and } S \text{ are regular expressions and } L(R) \subseteq L(S) \}$ . We observe that  $L(R) \subseteq L(S)$  if and only if  $\overline{L(S)} \cap L(R) = \emptyset$ . The following TM **X** decides A:

**X** = "On input  $\langle R, S \rangle$  where R and S are regular expressions:

1. Construct a DFA **B** such that  $L(B) = \overline{L(S)} \cap L(R)$  (we know how to convert a RE into an NFA, and an NFA into a DFA).
2. Run TM **E** on input  $\langle B \rangle$  (**E** is the decider of  $E_{DFA}$ , as in exercise 4).
3. If **E** accepts, *accept*. If **E** rejects, *reject*."

### Exercise 6: Sipser 5.9

Suppose  $L$  is a Turing-recognizable language. We must show that there exists a Turing-computable function  $f$  such that:

$$w \in L \Leftrightarrow f(w) \in A_{TM}$$

In particular, we can define  $f(w) = \langle M, w \rangle$ , where  $M$  is a TM that recognizes  $L$ . If  $w \in L$ , then machine  $M$  accepts its input  $w$ , thus  $\langle M, w \rangle \in A_{TM}$ . On the other hand, if  $w \notin L$ , then machine  $M$  doesn't accept its input  $w$ , thus  $\langle M, w \rangle \notin A_{TM}$ .

### Exercise 7: Sipser 5.12

Let  $S = \{ \langle M \rangle \mid M \text{ is a TM that accepts } w^R \text{ whenever it accepts } w \}$ . Show that  $S$  is undecidable. There are different ways to solve this problem. Here is one solution (I use mapping reducibility, but that isn't required):

We show that  $A_{TM} \leq_m S$  by mapping  $\langle M, w \rangle$  to  $\langle M' \rangle$ , where  $M'$  is the following TM:

$M'$  = "On input  $x$ :

1. If  $x$  is neither **01** nor **10**, *reject*.
2. If  $x = \mathbf{01}$ , *accept*.
3. Otherwise, if  $x = \mathbf{10}$ , simulate  $M$  on  $w$ . If  $M$  accepts  $w$ , then *accept*, otherwise *reject*."

Clearly,  $M'$  can only accept two strings: **01** and **10**.  $M'$  unconditionally accepts **01**, and it accepts **10** only if  $M$  accepts  $w$ .

If  $\langle M, w \rangle \in A_{TM}$ , then  $L(M') = \{\mathbf{01}, \mathbf{10}\}$ , so  $\langle M' \rangle \in S$ . Conversely, if  $\langle M, w \rangle \notin A_{TM}$ , then  $L(M') = \{\mathbf{01}\}$ , so  $\langle M' \rangle \notin S$ . Therefore,  $\langle M, w \rangle \in A_{TM} \Leftrightarrow \langle M' \rangle \in S$ , and  $A_{TM} \leq_m S$ . We conclude that  $S$  is undecidable.

### Exercise 8: Sipser 5.15

Let  $LM_{TM} = \{ \langle M, w \rangle \mid M \text{ ever moves left while computing on } w \}$ .  $LM_{TM}$  is decidable. Let  $M_{LEFT}$  be the TM which on input  $\langle M, w \rangle$  determines the number of states  $n = |Q|$  of  $M$  and then simulates  $M$  on  $w$  for  $|w|+n+1$  steps. If  $M_{LEFT}$  discovers that  $M$  moves left during that simulation,  $M_{LEFT}$  accepts  $\langle M, w \rangle$ . Otherwise, it rejects  $\langle M, w \rangle$ :

$M_{LEFT}$  = "On input  $\langle M, w \rangle$ , where  $M$  is a TM and  $w$  a string:

1. Determine  $n$ , the number of states of  $M$ .
2. Simulate  $M$  on input  $w$  during at most  $|w|+n+1$  steps. If  $M$  ever goes left during that simulation, *accept*.
3. If  $M$  never went left during the simulation, *reject*."

The reason that  $M_{\text{LEFT}}$  can reject without simulating  $M$  further is as follows. Suppose  $M$  does make a left move on input  $w$ . Let  $\mathbf{p} = \mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_s$  be the shortest computation path of  $M$  on  $w$  ending in a left move. Because  $M$  has been scanning only blanks (it's been moving right) since state  $\mathbf{q}_{|w|}$ , we may remove any cycles that appear after this state and be left with a valid computation path of the machine ending in a left move. Since we said that  $\mathbf{p}$  is the *shortest* computation path, there is no such cycle after state  $\mathbf{q}_{|w|}$ . The number of states that are visited after  $\mathbf{q}_{|w|}$  can't be more than  $|Q|$  before it reaches  $\mathbf{q}_s$  (since we just said that there are no cycles). Hence,  $\mathbf{p}$  must have a length at most  $|w| + |Q| + 1$ . It is guaranteed that  $M_{\text{LEFT}}$  will detect any first left move of  $M$  on input  $w$ , thus it decides  $LM_{\text{TM}}$ .