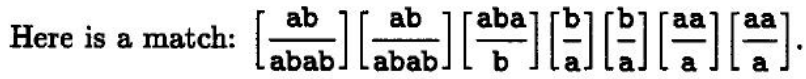
5.1 证明 EQ_{CFG} 是不可判定的。
解：只须证明 $ALL_{CFG} \leqslant_m EQ_{CFG}$ 即可。
构造 CFG G_1，使 $L(G_1)=\Sigma^*$。设计从 ALL_{CFG} 到 EQ_{CFG} 的归约函数如下：
F=“对于输入＜G＞，其中 G 是 CFG：
1）输出＜G,G_1＞。”
若＜G＞$\in ALL_{CFG}$，则<G,G_1>$\in EQ_{CFG}$ 。
若＜G＞$\notin ALL_{CFG}$，则<G, G_1>$\notin EQ_{CFG}$。
F 将 ALL_{CFG} 归约到 EQ_{CFG} 即 $ALL_{CFG} \leqslant_m EQ_{CFG}$
∵ALL_{CFG} 是不可判定的，
∴EQ_{CFG} 是不可判定的。

5.2 证明 EQ_{CFG} 是补图灵可识别的。
证明：注意到 A_{CFG}={<G,w>|G 是能派生串 w 的 CFG}是可判定的。构造如下 TM：
F=“输入<G,H>，其中 G,H 是 CFG，
1) 对于字符串 $S_1, S_2,\ldots$，重复如下步骤。
2) 检测 S_i 是否可以由 G 和 H 派生。
3) 若 G 和 H 中有一个能派生 w，而另一个不能，则接受。”
F 识别 EQ_{CFG} 的补。

5.3



5.4 如果 $A \leq_m B$ 且 B 是正则语言，这是否蕴涵着 A 也是正则语言？为什么？
解：否。例如：
对非正则语言 $A=\{0^n1^n|n\geq 0\}$和正则语言 B={0}，可以构造一个可计算函数 f 使得：

$$f(w)=\begin{cases}0, w = 0^n1^n\\ 1, w \neq 0^n1^n\end{cases}$$

于是 $w\in A \Leftrightarrow f(w)\in B$,故 $A\leq_m B$。

5.5 证明 A_{TM} 不可映射规约到 E_{TM}。
证明：反证法

假设 $A_{TM}\leq_m E_{TM}$， 则有 $\overline{A_{TM}} \leq_m \overline{E_{TM}}$ 。而 A_{TM} 的补不是图灵可识别的，从而可知 E_{TM} 的补也不是图灵可识别的。
下面构造一个识别 E_{TM} 的补的图灵机 S：
S=“输入<M>，M 是 TM,
1) 对 i=1,2,…重复下一步。

2) 对 $S_1,S_2,\ldots,S_i$ 模拟 M 运行 i 步，若有接受，则接受。”

S 可识别 E_{TM} 的补，所以 E_{TM} 的补是图灵可识别的，与上面由假设得到的 E_{TM} 的补不是图灵可识别的矛盾。所以 A_{TM} 不可映射规约到 E_{TM}。

5.6

Suppose $A \leq_m B$ and $B \leq_m C$. Then there are computable functions f and g such that $x \in A \iff f(x) \in B$ and $y \in B \iff g(y) \in C$. Consider the function composition $h(x) = g(f(x))$. We can build a TM which computes h as follows: first simulate a TM for f (such a TM exists since we assumed that f is computable) on input x and call the output y. Then simulate a TM for g on y. The output is $h(x) = g(f(x))$. Therefore, h is a computable function. Moreover, $x \in A \iff h(x) \in C$. Therefore, $A \leq_m C$ via the reduction function h.

5.7 证明：如果 A 是图灵可识别的，且 $A \leqslant_m \overline{A}$，则 A 是可判定的。

证：∵ $A \leqslant_m \overline{A} \Leftrightarrow \overline{A} \leqslant_m A$

且 A 为图灵可识别的

∴ $\overline{A}$ 也为图灵可识别的

∴由 A 和 $\overline{A}$ 都是图灵可识别的可知 A 是可判定的.

5.8 解：在由<M,w>构造相应骨牌簇时，添加如下一类骨牌：

若 M 中有一个左移δ(q,a)=(r,b,L)，则添加一张骨牌：

$$\left[\frac{\#qa}{\#rb}\right]。$$

并且第一张骨牌改为 $\left[\frac{\#}{\#\#q_0w}\right]$。

问题

5.9 证明 S={<M>|M 是 TM 且满足：只要它接受 w，就接受 w^R}不可判定。

证明：对任意<M,w>，其中 M 是 TM，w 是串，令 f(<M,w>)是如下 TM：

F=“输入 x，

1) 若 x≠01 或 10，则拒绝。
2) 若 x=01，则接受。

3) 若 x=10，则在 w 上运行 M。若 M 接受，则接受。”

可以看到<M,w>$\in A_{TM}\Leftrightarrow$ f(<M,w>)$\in$S。$A_{TM}\leqslant_m$S，所有 S 不可判定。

5.10 证明 S={<D,w>|双带 TM M 在输入 w 上运行时会在第二条带上写下一个非空白符}是不可判定的。

证明：对任意<M,w>，其中 M 是单带确定 TM，w 是字符串。令 f(<M,w>)=<D,w>，其中 D 是如下的双带 TM：

D=“输入 x，

1) 初始化，x 放在第一带上，第二带为空。
2) 在第一带上模拟 M 运行。
3) 若 M 接受，则在第二带上写下一个非空白符，并接受；若 M 拒绝，则拒绝。”

从 D 的构造可以看出<M,w>$\in A_{TM}\Leftrightarrow$<D,w>$\in$S，即 $A_{TM}\leqslant_m$S，所以 S 不可判定。

5.11 Use the language J from Problem 5.10. That problem showed that J is undecidable. Here we show that $J \leq_m \overline{J}$. The reduction f maps ϵ to itself and for other strings x,

$$f(x) = \begin{cases} 0s & \text{if } x = 1s \text{ for some string } s \in \Sigma^* \\ 1s & \text{if } x = 0s \text{ for some string } s \in \Sigma^* \end{cases}$$

The function f is computable, and $x \in J \iff f(x) \in \overline{J}$. Thus, $J \leq_m \overline{J}$.

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

M' = "On input x:

1. If $x = 01$ then *accept*.
2. If $x \neq 10$ then *reject*.
3. If $x = 10$ simulate M on w. If M accepts w then *accept*; if M halts and rejects w then *reject*."

If $\langle M, w\rangle \in A_{\mathsf{TM}}$ then M accepts w and $L(M') = \{01, 10\}$, so $\langle M'\rangle \in S$. Conversely, if $\langle M, w\rangle \notin A_{\mathsf{TM}}$ then $L(M') = \{01\}$, so $\langle M'\rangle \notin S$. Therefore, $\langle M, w\rangle \in A_{\mathsf{TM}} \iff \langle M'\rangle \in S$.

5.13 $USELESS_{TM}$ ={<N>| N 是 TM,并且 N 有无用状态}。

求证 $USELESS_{TM}$ 不可判定。

证明：构造 E_{TM} 到 $USELESS_{TM}$ 的规约函数:

F=“对于输入<M>,M=$(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$是 TM,

1) 构造 TM N

$N=(Q,\Sigma_1,\Gamma_1,\delta_1,q_0,q_{accept},q_{reject})$，其中，

$\Sigma_1=\Sigma\cup\{\$\}$,$\Gamma_1=\Gamma_1\cup\{\$\}$。设 $Q=\{q_0,q_1,q_2,\ldots,q_n,q_{reject},q_{accept}\}$。

对任意 $q\in Q$，$a\in\Gamma_1$：

$$\delta_1(q,a)=\begin{cases}\delta(q,a), & a\neq \$,\\ (q_{i+1},\$,L), & a=\$, q=q_i, 0\le i<n,\\ (q_{reject},\$,L), & a=\$, q=q_n.\end{cases}$$

2) 输出<N>。”

对于任意 TM M，如上构造的 TM N，除接受状态外，每个状态均非无用状态(若在输入$上运行 N，则 N 遍历 $q_0,q_1,q_2,\ldots,q_n$，最后进入 q_{reject} 并停机)。构造 N 的目的就是消除 M 中任何非接受状态为无用状态的可能。因此有：

$$<M>\in E_{TM}\Rightarrow <N>\in USELESS_{TM}$$
$$<M>\notin E_{TM}\Rightarrow <N>\notin USELESS_{TM}$$

所以 $E_{TM}\leqslant_m USELESS_{TM}$ 而 E_{TM} 不可判定，因此 $USELESS_{TM}$ 不可判定。

5.14 考虑这样的问题：检查图灵机在输入 w 上，当其读写头处于带最左方格时，是否曾经试图将读写头向左移。将这个问题形式化为一个语言，并证明它是不可判定的。

解：此问题可以形式化为一个语言 S：

S={<M,w> | TM M 在输入 w 上，当其读写头处于带最左方格时，曾经试图将读写头向左移}

为证明 S 是不可判定的，可以证明 $A_{TM}\leqslant_m S$。构造一个可计算函数 $f:\Sigma^*\to\Sigma^*$，使得对每个<M,w>，其中 M 是 TM，w 是串，f(<M,w>)=<M’,w>，其中

M’=“输入 x,

1) 将工作带上内容改为$x。
2) 读写头置于 x 的第一个字符，模拟 M 运行。
3) 每当读写头移到$，保持状态，右移一格。
4) 若 M 进入接受状态，读写头左移到$,再左移一次，停机，接受；若 M 进入拒绝状态，则拒绝。”

于是$<M,w>\in A_{TM}\Leftrightarrow <M',w>\in S$。

5.15 证明 S={<M,w>|图灵机 M 在输入 w 上运行时有左移}可判定。

证明：构造如下 TM F：

F=“输入<M,w>，M 是 TM，w 是串，

1) 计算 M 的状态数，记为 p。
2) 在 w 上模拟 M，直到有左移，或停机，或已运行了|w|+p 步。
3) 若有左移，则接受；若停机但无左移，则拒绝；若无左移且不停机，则拒绝。”

需要说明的是在|w|+p 步运行中无左移也不停机的情况。由于无左移，M 运行|w|步以后进入空白区域。由于此后右移使得每次读写头所指向的都是空格，而且此后运行的 p 步至少会有一个状态出现两次，所以不停机意味着 M 进入了循环，也就不会出现左移。总之，F 判定 S。

5.16 Let $B = \{\langle M \rangle | \ M$ is a two tape TM which writes a non-blank symbol on its second tape when it is simulated on some particular input $w\}$. We show that $A_{\mathrm{TM}} \leq_{\mathrm{m}} B$ by mapping $\langle M, w \rangle$ to $\langle M' \rangle$ where M' has the following description:

M' = "On input x:

1. Simulate M on w only using the first tape.
2. If M accepts w then write `blah blah !` on the second tape."

Observe that $\langle M, w \rangle \in A_{\mathrm{TM}} \iff M' \in B$.

5.17 证明 PCP 在一元字母表上,即在字母表Σ={1}上,是可判定的.

证明：构造识别该语言的图灵机如下:

S=“对输入的骨牌序列<P>,

扫描骨牌序列。若所有的骨牌的上面的 1 的个数都大于下面的 1 的个数，或都小于下面的 1 的个数，则拒绝。否则，接受。”

S 判定这样的 PCP。

5.18 证明 PCP 在二元字母表上，即在字母表Σ={0，1}上，是不可判定的。

证明：要想证明该 PCP(记为 PCP_2)是不可判定的，只须证明 $A_{TM} \leqslant_m PCP_2$。为此需要利用定理 5.11 的证明过程和规约的传递性：

首先，把书中的 PCP 任一实例 P 映射到 PCP_2 的实例 P_2

设计从 P 到 P_2 的规约函数如下：

F=“对输入<P>，其中 P 是 PCP：

1) 造 PCP P_2，对 P 中所有骨牌中包含的字符串进行 Huffman 编码，形成一一对应的只含 0 和 1 的字符串(也可进行定长编码)。
2) 输出<P_2>。”

F 将 PCP 映射规约到 PCP_2，即 $PCP \leqslant_m PCP_2$;

其次，有定理 5.11 有 $A_{TM} \leqslant_m PCP$;

根据规约的传递性可知，$A_{TM} \leqslant_m PCP_2$

∵A_{TM} 是不可判定的，

∴PCP_2 是不可判定的。

5.19 We reduce *PCP* to $AMBIG_{\mathsf{CFG}}$, thereby proving that $AMBIG_{\mathsf{CFG}}$ is undecidable. We use the construction given in the hint in the text.

Note: The hint that appears in the first printing is incorrect. The correct version (which appears in the second and subsequent printings) is: Given an instance

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

$$\begin{aligned} S &\to T \mid B \\ T &\to t_1 T \mathtt{a}_1 \mid \cdots \mid t_k T \mathtt{a}_k \mid t_1 \mathtt{a}_1 \mid \cdots \mid t_k \mathtt{a}_k \\ B &\to b_1 B \mathtt{a}_1 \mid \cdots \mid b_k B \mathtt{a}_k \mid b_1 \mathtt{a}_1 \mid \cdots \mid b_k \mathtt{a}_k \ , \end{aligned}$$

where $\mathtt{a}_1, \ldots, \mathtt{a}_k$ are new terminal symbols. We prove that this reduction works.

We show that P has a solution iff the CFG is ambiguous. If P has a match $t_{i_1} t_{i_2} \ldots t_{i_l} = b_{i_1} b_{i_2} \ldots b_{i_l}$, the string

$$t_{i_1} t_{i_2} \ldots t_{i_l} \mathtt{a}_{i_l} \ldots \mathtt{a}_{i_2} \mathtt{a}_{i_1} = b_{i_1} b_{i_2} \ldots b_{i_l} \mathtt{a}_{i_l} \ldots \mathtt{a}_{i_2} \mathtt{a}_{i_1}$$

has two different leftmost derivations, one from T and one from B. Hence the CFG is ambiguous.

Conversely, if the CFG is ambiguous, some string w has multiple leftmost derivations. All generated strings have the form $w = w_{start} \mathtt{a}_{i_l} \ldots \mathtt{a}_{i_2} \mathtt{a}_{i_1}$ where w_{start} contains only symbols from P's alphabet. Notice that once we choose the first step in the derivation of w (either $S \to T$ or $S \to B$), the following steps in the derivation are uniquely determined by the sequence $\mathtt{a}_{i_l} \ldots \mathtt{a}_{i_2} \mathtt{a}_{i_1}$. Therefore w has at most two leftmost derivations:

i) $S \to T \to t_{i_1} T \mathtt{a}_{i_1} \to t_{i_1} t_{i_2} T \mathtt{a}_{i_2} \mathtt{a}_{i_1} \to \cdots \to t_{i_1} t_{i_2} \ldots t_{i_l} \mathtt{a}_{i_l} \ldots \mathtt{a}_{i_2} \mathtt{a}_{i_1}$, and

ii) $S \to B \to b_{i_1} T \mathtt{a}_{i_1} \to b_{i_1} b_{i_2} T \mathtt{a}_{i_2} \mathtt{a}_{i_1} \to \cdots \to b_{i_1} b_{i_2} \ldots b_{i_l} \mathtt{a}_{i_l} \ldots \mathtt{a}_{i_2} \mathtt{a}_{i_1}$.

If w is ambiguous, $t_{i_1} t_{i_2} \ldots t_{i_l} \mathtt{a}_{i_l} \ldots \mathtt{a}_{i_2} \mathtt{a}_{i_1} = b_{i_1} b_{i_2} \ldots b_{i_l} \mathtt{a}_{i_l} \ldots \mathtt{a}_{i_2} \mathtt{a}_{i_1}$, and therefore $t_{i_1} t_{i_2} \ldots t_{i_l} = b_{i_1} b_{i_2} \ldots b_{i_l}$. Thus, the initial *PCP* instance, P, has a match.

5.20 **a.** A 2DFA M with s states computing on an input x of size n has at most $s(n+2)^2$ possible configurations. Thus, on input x, M either halts in $s(n+2)^2$ steps or loops forever. To decide A_{2DFA}, it is enough to simulate M on x for $s(n+2)^2$ steps and accept if M has halted and accepted x during this finite duration.

b. Suppose a TM D decides E_{2DFA}. We construct a TM E that decides E_{TM}. E runs as follows:

E = "On input $\langle M \rangle$,

1. Construct a 2DFA M' that recognizes the language of accepting computation histories on M, namely, $\{\langle c_1 \# c_2 \# \ldots \# c_k \rangle \mid c_i$ is a configuration of M, c_1 is a start configuration, c_k is an accepting configuration, c_{i+1} legally follows c_i for each $i\}$. A 2DFA can check whether c_1 is a start configuration of M by verifying that it begins with a start state q_0 and contains legal symbols from M's input alphabet. Similarly, it can check whether c_k is an accepting configuration by verifying that it contains q_{accept} and symbols from M's tape alphabet. These two steps use only one head. To check whether c_{i+1} legally follows c_i, the 2DFA can keep its two heads on c_i and c_{i+1}, and compare them symbol by symbol. Notice that $L(M') = \emptyset$ iff $L(M) = \emptyset$.
2. Run D on $\langle M' \rangle$. If D accepts, *accept*. If D rejects, *reject*."

Since E_{TM} is undecidable, E_{2DFA} is decidable.

5.21 设 $AMBIG_{CFG}$＝{<G>|G 是歧义的 CFG}。证明 $AMBIG_{CFG}$ 是不可判定的。
证明：设 $AMBIG_{CFG}$ 是可判定的，且 R 判定 $AMBIG_{CFG}$ 构造 S 判定 PCP
S＝“对于任一 PCP 输入，如 p＝{[t_1/b_1],[t_2/b_2],...,[t_k/b_k]}

1) 利用如下规则构造一个 CFG G：
   $S \to T|B$
   $T \to t_1Ta_1|t_2Ta_2|...|t_kTa_k|t_1a_1|...|t_ka_k$
   $B \to b_1Ba_1|b_2Ba_2|...|b_kBa_k|t_1a_1|...|b_ka_k$
2) 在<G>上运行 R，如果 R 接受则接受，如果 R 拒绝则拒绝。”

其中 a_i 保证在 $T \to t_1Ta_1|t_2Ta_2|...|t_kTa_k|t_1a_1|...|t_ka_k$ 不是歧义 CFG。
这样，如果 $AMBIG_{CFG}$ 是可判定的，则有 PCP 可判定的，而 PCP 是不可判定的，导出矛盾。所以可以得到 $AMBIG_{CFG}$ 是不可判定的。

5.22 证明所有的图灵可识别问题都映射可规约到 A_{TM}。
证明：设问题 A 是图灵可识别的，且 M 是识别 A 的 TM。构造一个可计算函数 f 使得 f(w)=<M,w>， 则有

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

这说明 $A \leqslant_m A_{TM}$。

5.23 First, let P be the language $\{\langle M\rangle \mid M \text{ is a TM with 5 states}\}$. P is non-trivial, and so it satisfies the second condition of Rice's Theorem but P can be easily decided by checking the number of states of the input Turing Machine. Second, let P be the empty set. Then it does not contain any TM and so it satisfies the first condition of Rice's Theorem, but and P can be decided by a TM that always rejects.

Therefore both properties are necessary for proving P undecidable.

5.24 证明：对任意字符串 x，令 $f_1(x)=0x$。则有 $x\in A_{TM}\Leftrightarrow f_1(x)=0x\in J$。即有 $A_{TM}\leqslant_m J$。进一步有 $\overline{A_{TM}}\leq_m \overline{J}$。由 $\overline{A_{TM}}$ 图灵不可识别知 $\overline{J}$ 图灵不可识别。

对任意字符串 x，令 $f_2(x)=1x$。则有 $x\notin A_{TM}\Leftrightarrow f_2(x)=1x\in J$。即有 $\overline{A_{TM}}\leq_m J$。由 $\overline{A_{TM}}$ 图灵不可识别知 J 图灵不可识别。

5.25 给出一个不可判定语言 B 的例子，使得 $B\leqslant_m \overline{B}$。

解：可利用第 10 题的结果。令 B 为 5.24 中的 J，则 $J\leqslant_m \overline{J}$。构造归约函数如下

F=“输入 w，

1）对 w 的第一位取反，即 0 变 1，1 变 0。

2）输出 w。”

则 F 把 J 映射归约到 $\overline{J}$。而 J 又是不可判定的。

5.26 证明：

(a) 判定 A_{2DFA} 的算法如下：

L=“对于输入<M，x>，其中 M 是 2DFA，x 是串，

1) 计算 M 的状态个数 q，和 x 的长度 n。

2) 在 x 上模拟 M qn^2 步，或直至它停机；

3) 若 M 停机，则当 M 接受时接受，拒绝时拒绝。若未停机，则拒绝。”

因为 M 有 q 个状态，所以对长度为 n 的输入，M 至多有 qn^2 个不同格局(注意：带上的内容不会改变)。若模拟 qn^2 步还未停机，则必定是进入了循环，该情况下应拒绝。

(b) 构造从 A_{TM} 到 E_{2DFA} 的补的映射归约函数。对于任意给定的<M,ω>，f(<M,ω>)=<B>是如下的 2DFA 的描述：

B=“输入 x,

若 $x=\#C_1\#C_2\#\ldots\#C_m\#$ 是<M 在 ω 上的接受计算历史>，即检查 x 满足：

a) C_1 是 M 在 ω 上的起始格局。

b) 每个 C_{i+1} 都是 C_i 的合法结果。

c) C_m 是 M 的一个接受格局。

则接受。”

条件 a,c 较容易验证。验证 b 时，B 的两个读头分别在 C_i 和 C_{i+1} 的相应位置上移动，验证结果是否适当。由 B 的构造可以看到

$$<M, \omega> \in A_{TM} \Leftrightarrow f(<M, \omega>)=<B> \notin E_{2DFA}$$

此即 A_{TM} 映射可归约到 E_{2DFA} 的补。由 A_{TM} 不可判定，知 E_{2DFA} 不可判定。

5.27 证明 $EQ_{2DIM\text{-}DFA}$ 不可判定。

证明：下面证明 A_{TM} 映射可归约到 $EQ_{2DIM\text{-}DFA}$ 的补：

对于任意<M,w>，f(<M,w>)=<G,H>是如下的两个 2DIM-DFA 的描述，其中 H 满足 L(H)=∅。为构造 G，记格局序列 $C_1,C_2,\ldots,C_m$ 的编码为$<C_1,C_2,\ldots,C_m>$，它是由格局序列 $C_1,C_2,\ldots,C_m$ 组成的矩形串。其由下至上分别是 $C_1,C_2,\ldots,C_m$，一个格局一行，较短的在右边补上适当数量的空格，四周是由#围成的方框。

G= “输入串 x，

若 x=$<C_1,C_2,\ldots,C_m>$是<M 在 w 上的接受计算历史>，即 x 满足：

a) C_1 是 M 在 w 上的起始格局。
b) 每个 C_{i+1} 都是 C_i 的合法结果。
c) C_m 是 M 的一个接受格局。

则接受。”

由 G,H 的构造可以看到

$$<M,w> \in A_{TM} \Leftrightarrow f(<M,w>)=<G,H> \notin EQ_{2DIM\text{-}DFA}$$

此即 A_{TM} 映射可归约到 $EQ_{2DIM\text{-}DFA}$ 的补。由 A_{TM} 不可判定，知 $EQ_{2DIM\text{-}DFA}$ 不可判定。

5.28 Assume for the sake of contradiction that P is a decidable language satisfying the properties and let R_P be a TM that decides P. We show how to decide A_{TM} using R_P by constructing TM S. First, let $T_\emptyset$ be a TM that always rejects, so $L(T_\emptyset)=\emptyset$. You may assume that $\langle T_\emptyset \rangle \notin P$ without loss of generality because you could proceed with $\overline{P}$ instead of P if $\langle T_\emptyset \rangle \in P$. Because P is not trivial, there exists a TM T with $\langle T \rangle \in P$. Design S to decide A_{TM} using R_P's ability to distinguish between $T_\emptyset$ and T.

S = "On input $\langle M, w \rangle$:

1. Use M and w to construct the following TM M_w.
   M_w = "On input x:
   1. Simulate M on w. If it halts and rejects, *reject*. If it accepts, proceed to stage 2.
   2. Simulate T on x. If it accepts, *accept*."
2. Use TM R_P to determine whether $\langle M_w \rangle \in P$. If YES, *accept*. If NO, *reject*."

TM M_w simulates T if M accepts w. Hence $L(M_w)$ equals $L(T)$ if M accepts w and $\emptyset$ otherwise. Therefore, $\langle M_w \rangle \in P$ iff M accepts w.

5.29

(a) (20 points, *Sipser* problem 5.29) Sipser's proof shows that the two conditions stated above for p are *sufficient* to prove that A is not Turing decidable. Show that both conditions are also *necessary*.

Solution: If p is trivial, then either all TM descriptions are in the language (if p includes all TMs) or the language is empty. In the former case, we just build a TM that makes sure that M is a valid TM description and accepts. In the lattter case, we build a TM that rejects all strings (e.g. see the solution to question 2a).
If p is a property of the machine rather than the language, it *may* be decidable. For example the question of whether or not a TM has 42 states (see question 1b) is decidable.

5.30 (a) $INFINITE_{\mathsf{TM}}$ is a language of TM descriptions. It satisfies the two conditions of Rice's theorem. First, it is nontrivial because some TMs have infinite languages and others do not. Second, it depends only on the language. If two TMs recognize the same language, either both have descriptions in $INFINITE_{\mathsf{TM}}$ or neither do. Consequently, Rice's theorem implies that $INFINITE_{\mathsf{TM}}$ is undecidable.

5. 31

Solution: We build our solution in stages. Later stages use Turing machines that we construct in earlier stages. The first step is to construct a Turing machine M_{query} that takes an input x and accepts if iterating f starting from x eventually yields 1, and loops forever otherwise.

M_{query}. On input $\langle x \rangle$:

1. If $x = 1$, then accept.
2. If x is odd, update $x \leftarrow 3x + 1$. If x is even, update $x \leftarrow x/2$.
3. Go to step 1.

Our next step is to construct a Turing machine M_{loop} which iterates over all positive integers, looking for a counter-example to the $3x + 1$ conjecture. That is, M_{loop} searches for some x such that iterating f starting from x never reaches 1 and accepts if it finds such an x. To do this, it seems tempting to have M_{loop} simulate M_{query} first on $x = 1$, next on $x = 2$, and so forth. Whenever iterating f on x yields 1, the simulation of M_{query} will eventually end and M_{loop} would then proceed to the next number $x + 1$. But what if M_{loop} actually finds a counter-example x? In this case, the simulation of M_{query} on x will never terminate, and M_{loop} will be in the unfortunate situation that it has found what it is looking for, but it doesn't know it has found it!

To get around this, we use H. Instead of simulating M_{query} on x, we have M_{loop} check whether or not M_{query} would accept x by passing $\langle M_{\text{query}}, x \rangle$ to H.

M_{loop}. On input $\langle w \rangle$:

1. Ignore the input w.
2. For each natural number $y = 1, 2, \ldots$:
    3. Run H on $\langle M_{\text{query}}, y \rangle$.
    4. If H rejects, then accept. Otherwise, continue the loop.

Finally, in order to solve the $3x+1$ problem, we need to know whether or not M_{loop} finds a counter-example. Again, we might be tempted to simulate M_{loop} and see if it ever finds a counter-example and accepts. The problem is that there may not be any counter-example, in which case M_{loop} will loop forever and our simulation will not terminate. The trick is to use H again to see if M_{loop} finds a counter example.

M_{3x+1}. On input $\langle w \rangle$:

1. Ignore the input w.
2. Run H on $\langle M_{\text{loop}}, \varepsilon \rangle$.
3. If H accepts, then print "There is a counter-example to the $3x+1$ conjecture." Otherwise, print "The $3x+1$ conjecture holds."

5. 33

(35 points, Sipser problem 5.34) Let P be a PDA and $WW = \{ww \mid w \in \{0,1\}^*\}$. Use computational histories to show that the question of whether P accepts some string in WW is undecidable.
Solution (sketch): Given a string $M\#x$ where M describes a TM and x describes an input to M, we'll build a PDA that accepts a string of the form $h\$h\$$ iff $x \in L(M)$. As we described in the Nov. 10 notes, we'll write the history as $C_0 C_1^{\mathcal{R}} C_2 C_3^{\mathcal{R}} \ldots C_n$, where C_0 is the initial configuration, C_n is the final configuration (reversed if n is odd, even-indexed configurations are written in normal order, and odd-indexed configurations are reversed. Our PDA will push C_0 onto its stack while confirming that it is the initial configuration for M running with input w. For each odd indexed configuration, the PDA pops the previous configuration off of its stack and verifies that the new configuration is the successor of the previous one. When the PDA reads the first \$, it skips configuration C_0 and then pushes C_1 onto the stack. This time, it reads the configurations confirming that each even-indexed configuration is the successor of the previous odd-indexed configuration. The PDA accepts if all of these checks for valid successors of configurations pass and if the final configuration is an accepting one. This PDA accepts a string of the form ww iff M accepts x.

Note that this PDA may accept strings of the form $y\$z\$$ where $y \neq z$ even if M does not accept x. This is because our construction relies on the separte check that the input is of the form ww (that can't be done by a PDA) to ensure that the two passes over the history are checking the *same* history.

5. 34

Let $X = \{\langle M, w \rangle |$ where M is a single tape TM that never modifies the portion of the tape containing the input w $\}$. Show that X is not decidable by a reduction from A_{TM}.

Hint: you may find it easier to work with $\overline{X}$.

Solution: We show $A_{\text{TM}} \leq_m \overline{X}$. Here is the reduction: $f(\langle M, w \rangle) = \langle M'_w, w \rangle$, where M'_w is defined as follows: on any input it moves past the input, writes \$ followed by w, and then it simulates M on w never going to the left of \$. If M accepts w, M'_w moves to the left of \$, and writes anything on the original input, and accepts.

The correctness of the reduction follows from the fact that if M does not accept w, M' will never move to the left of \$, and if M does accept w, then M' writes something on the original input.

5. 35

5.35:

Say that a variable A in CFG G is *necessary* if it appears in every derivation of some string w. Let

$$\text{NECESSARY}_{CFG} = \{\langle G, A\rangle | A \text{ is a necessary variable in } G\}.$$

1. Show that NECESSARY_{CFG} is Turing-recognizable.

   Let M be a TM that behaves as follows:

   - On input $\langle G, A\rangle$:
     - Construct CFG $G \setminus A$ by removing the nonterminal A and any productions that mention it from G.
     - Enumerate strings w generated by G. For each such string, simulate a decider for A_{CFG} to test whether w is generated by $G \setminus A$. If w is *not* generated by $G \setminus A$ then *accept*, otherwise continue the search.
   - On other input, *reject*.

The language $L(G \setminus A)$ consists of all and only those strings $w \in L(G)$ that have derivations that do not use A. If A is necessary for G, then some string $w \in L(G)$ cannot be derived without the use of A, hence $w \notin L(G \setminus A)$. Eventually, M will identify such a w and will accept. On the other hand, if A is not necesssary for G, then $L(G) = L(G \setminus A)$ and M will loop. Thus, M recognizes NECESSARY_{CFG}.

Note that it is *not* possible to prove this part by enumerating all strings $w \in L(G)$ and checking all derivations of each such string to see whether or not A is used, because in general a string $w \in L(G)$ can have *infinitely many* derivations. So the checking procedure might never terminate.

2. Show that NECESSARY_{CFG} is undecidable.

We show $\text{ALL}_{CFG} \leq_m \overline{\text{NECESSARY}}_{CFG}$. Since we already know that ALL_{CFG} is undecidable, this will show that $\overline{\text{NECESSARY}}_{CFG}$ is undecidable, hence NECESSARY_{CFG} must also be undecidable (a language is (un)decidable if and only if its complement is (un)decidable).

The reduction f is computed as follows:

- On input $\langle G \rangle$:
  - Construct G' by adding to G a new nonterminal A, together with productions: $S \to A$, $A \to \epsilon$, and $A \to aA$ for each $a \in \Sigma$.
  - Output $\langle G', A \rangle$.

Note that the grammar G' constructed by f is always such that $L(G') = \Sigma^*$. Thus, if $L(G) = \Sigma^*$, then A is not necessary for G', because every string $w \in \Sigma^*$ can already be derived from G, hence also from G' by a derivation not using A. Also, if $L(G) \neq \Sigma^*$, then A *is* necessary for G', because if $w \notin L(G)$ then w can only be derived from G' by a derivation that uses A. To summarize, if $\langle G \rangle \in \text{ALL}_{CFG}$, then $\langle G', A \rangle \notin \text{NECESSARY}_{CFG}$, and if $\langle G \rangle \notin \text{ALL}_{CFG}$, then $\langle G', A \rangle \in \text{NECESSARY}_{CFG}$. Hence, f is a reduction of ALL_{CFG} to $\overline{\text{NECESSARY}}_{CFG}$, as claimed.