25.8(f)

$S \rightarrow S_1\ 0\ S_1$

$S_1 \rightarrow 1\ S_1\ 0 \mid 0\ S_1 1 \mid S_1\ S_1 \mid 0\ S_1 \mid S_1\ 0 \mid\ \epsilon$

26.5(d)

Pseudocode code:

1. Goto *, check if bit on the right is 0, otherwise reject. (check if it is an empty string)
2. Goto *, move one step to the right at a time until the end of the input, check all bits in the input if they are 0, if find a single 1 before the end of the string (before '␣' is found), reject.
3. Goto *, check if the first bit is a 0, then move right one bit, check if next bit is '␣', if so, accept.
4. Return *
5. Move right to the first unmarked bit before blank, if a blank is reached before any unmarked bit, accept. Otherwise, mark the location.
6. Move right to the first unmarked bit before blank. if a blank is reached before any unmarked bit, reject.
7. Move right to the first unmarked bit before blank, if a blank is reached before any unmarked bit, reject. Otherwise, mark the location.
8. Move right to the first unmarked bit before blank, if a blank is reached, go to step 4. Otherwise go to step 6.

We must mark every other unmarked bit, if we reach the end of the string and there are still some unmarked bits left, go back to the starting point of the string to find the first unmarked bit from left to right. If the last unmarked bit is followed by a ⊔, then we know the number of 0s is in power of 2.

{q0} {1,⊔} → {E} {} {} //Error checking for empty string and 1

{q0} {*} → {q1} {} {R} //move to first bit

{q1} {⊔} → {q2} {} {L}

{q1} {1} → {E} {} {}

{q1} {0} → {q1} {} {R} //go through the string for error checking

{q2} {0} → {q2} {} {L} //move back to the first bit

{q2} {∗} → {q3} {} {R}

{q3} {} → {q4} {X} {R} //If it is unmarked

{q3} {*X*} → {q3} {} {R} //if it is marked, we need to find unmarked valid bit

{q4} {X} →{q4} {} {R}

{q4} {} →{q5} {} {R}

{q4} {⊔} →{A} {} {} //If the last unmarked bit is followed by a ⊔, accept

{q5} {X} → {q5} {} {R}

{q5} {} → {q6} {X} {R}

{q5} {⊔} → {E} {} {} //If the last unmarked bit is not followed by a ⊔, error

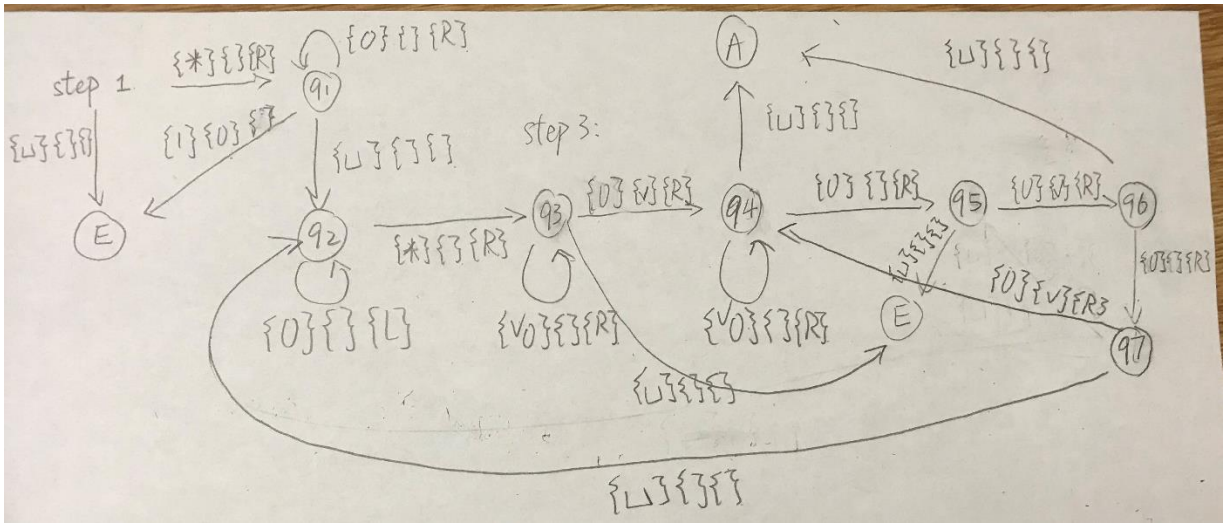
{q6} {X} → {q6} {} {R}

{q6} {⊔} → {A} {} {} //If the last unmarked bit is followed by a ⊔, accept

{q6} {} → {q7} {} {R}

{q7} {} → {q4} {} {R} //if this bit is unmarked, repeating steps

{q7} {⊔} → {q2} {} {L}



26.8(f)

Step1: Goto *

Step2: Move to the right. If the next bit is marked, halt.

Step3: Move right until a blank or a marked bit is reached

Step4: Move to the left. Mark the location and read the bit. Mark this bit

Step5: Move right until reach first unmarked bit. Insert the bit that has been read into current bit. Mark this bit and goto step 1.

27.4(b)

```
for (n is a prime in set ℕ)
        if (n, n + 2 are not prime)
        print(“Hello”)
        break;
```

27.20

(a) B is undecidable

(b) B is uncertain

A is reducible to B, A is decidable, we cannot know whether B is decidable or undecidable. Not enough information to determine.

(c) A is uncertain

A is reducible to B, B is undecidable, we cannot know whether A is decidable or undecidable, not enough information to determine.

(d) A is decidable

27.45

(a)

3 * d1 + 4 * d2

(b)

d1 d1 d3 d4 d4 d2 d2

27.46

Since the length of the strings are fixed, we only need to compare if every single input from the bottom matches the top.

Turing machine for variant of PCP

Input: string a # string b

Check:

The length of top and bottom if they are the same, if not, reject.

Repeat:

Compare value at the same index of each strings until the end is reached

Reject:

Any value from the bottom and the top don’t match