CSC 520 Summer 2024
HW #1
25 points/10 extra credit points

Submit a .zip archive that can contain these files:

- A plain text .txt file that affirms the honor code. For individuals: On my honor as an SFSU student, I, *<name>*, have neither given or received inappropriate help with this homework assignment. I understand that I will be asked to redo the assignment in person if this submission presents any question of an honor code violation.
  For groups: On our honor as SFSU students, we, *<names>*, have neither given or received inappropriate help with this homework assignment. All group members participated in this work, and all concur with the submission. We understand that we will be asked to redo the assignment in person if this work presents any question of an honor code violation.
- A .py Python source code file with your answer for problem 1.
- A .jff file containing your solution for problem 2.

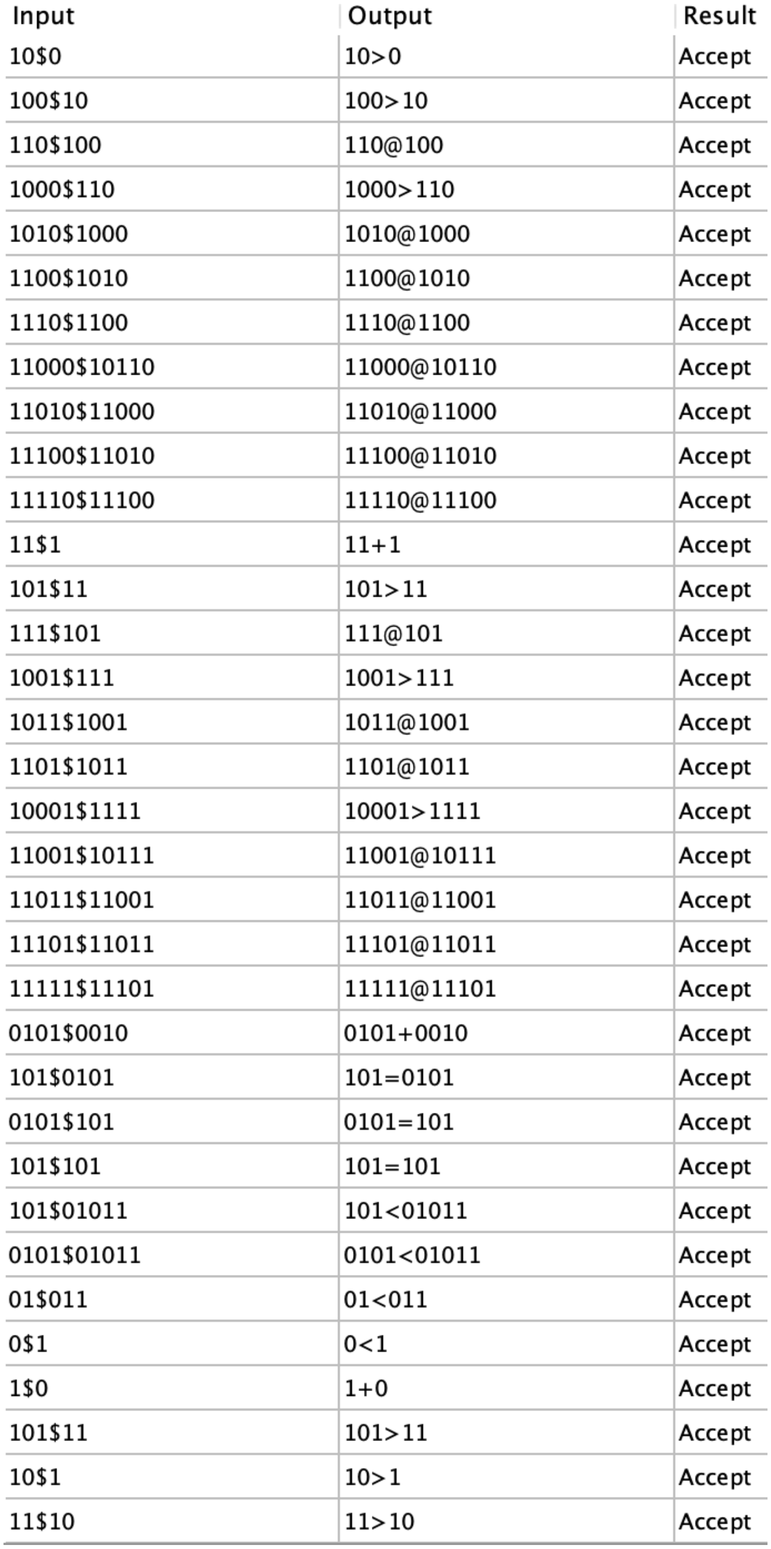
1. **(12.5 points. Include a modified version of computesLen-template.py in your .zip submission)**
ComputesLen(*P*) == `'yes'` if $\forall_I$(P(*I*) returns |*I*| encoded as a string); otherwiseComputesLen(*P*) returns `'no'`. That is, ComputesLen decides if a SISO Python function returns the length of its input as a string. For example:

```
def ƒ1(inString): return str(len(inString))
def ƒ2(inString): return 'CAGT'
ComputesLen(rf('ƒ1.py') == 'yes'
ComputesLen(rf('ƒ2.py') == 'no'
```

Prove that ComputesLen is undecidable by completing yesViaLen() and alterYesToLen() in computesLen-template.py. **Submissions that define any function other than these two, or which code alterYesToLen() to return a function, will not be graded.**

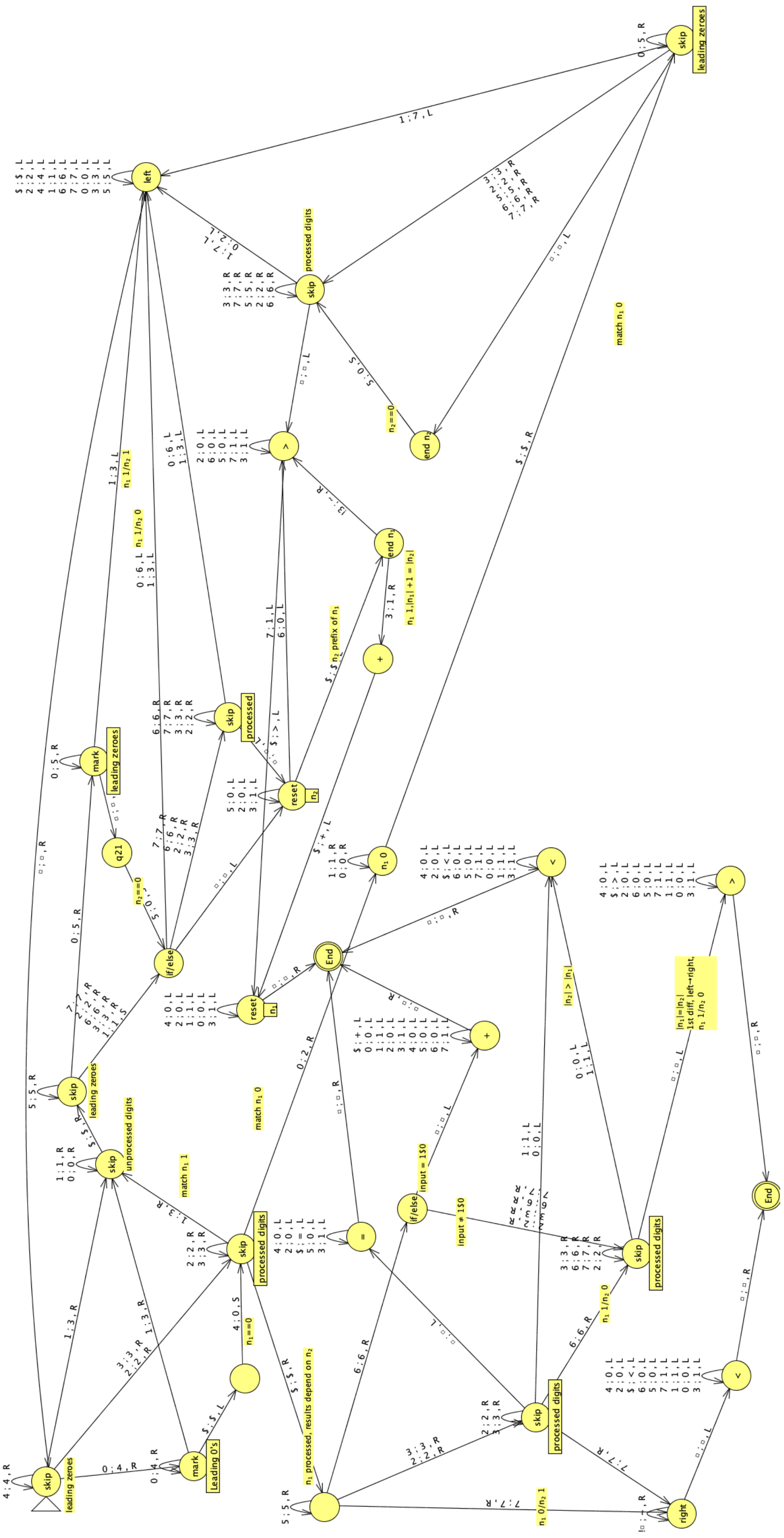
2**. (12.5 points/10 extra points for a minimal correct solution. Include a modified version of compareNumbers-template.jff in your .zip submission)**

The input to TM M is $s_1\$s_2$, where s_1 and s_2 are strings of *1's* and *0's*. Neither s_1 *or* $s_2 = \varepsilon$. When s_1 and s_2 are interpreted as the binary numbers n_1 and n_2 respectively, with leading zeroes ignored, the template output is $s_1{=}s_2$ if $n_1 = n_2$; $s_1{+}s_2$ if $n_1 = 2*n_2 + 1$; $s_1{>}s_2$ if $n_1 > n_2$ and $n_1 \neq 2*n_2 + 1$; and $s_1{<}s_2$ if $n_1 < n_2$. The new feature to implement is that when $|s_1| = |s2|$, ignoring leading zeroes, and $n_1 = n_2 + 2$, the output is $s_1@s_2$. A correct implementation of M will have the transducer results below for the inputs in test_cases.txt (also included with the homework materials).



Input	Output	Result
10$0	10>0	Accept
100$10	100>10	Accept
110$100	110@100	Accept
1000$110	1000>110	Accept
1010$1000	1010@1000	Accept
1100$1010	1100@1010	Accept
1110$1100	1110@1100	Accept
11000$10110	11000@10110	Accept
11010$11000	11010@11000	Accept
11100$11010	11100@11010	Accept
11110$11100	11110@11100	Accept
11$1	11+1	Accept
101$11	101>11	Accept
111$101	111@101	Accept
1001$111	1001>111	Accept
1011$1001	1011@1001	Accept
1101$1011	1101@1011	Accept
10001$1111	10001>1111	Accept
11001$10111	11001@10111	Accept
11011$11001	11011@11001	Accept
11101$11011	11101@11011	Accept
11111$11101	11111@11101	Accept
0101$0010	0101+0010	Accept
101$0101	101=0101	Accept
0101$101	0101=101	Accept
101$101	101=101	Accept
101$01011	101<01011	Accept
0101$01011	0101<01011	Accept
01$011	01<011	Accept
0$1	0<1	Accept
1$0	1+0	Accept
101$11	101>11	Accept
10$1	10>1	Accept
11$10	11>10	Accept

compareNumbers-template.jff


skip
leading zeroes
4 ; 4 , R
0 ; 4 , R
mark
Leading 0's
0 ; 4 , R
$; $, L
4 ; 0 , S
n₁==0
1 ; 3 , R
3 ; 3 , R
2 ; 2 , R
1 ; 3 , R
skip
unprocessed digits
1 ; 1 , R
0 ; 0 , R
$; $, R
skip
leading zeroes
5 ; 5 , R
7 ; 7 , R
2 ; 2 , R
6 ; 6 , R
3 ; 3 , R
1 ; 1 , S
0 ; 5 , R
mark
leading zeroes
0 ; 5 , R
□ ; □ , L
q21
n₂==0
5 ; 0 , S
□ ; □ , R
1 ; 3 , L
n₁ 1/n₂ 1
0 ; 6 , L
1 ; 3 , L
n₁ 1/n₂ 0
left
$; $, L
2 ; 2 , L
4 ; 4 , L
1 ; 1 , L
6 ; 6 , L
7 ; 7 , L
0 ; 0 , L
3 ; 3 , L
5 ; 5 , L
skip
processed digits
2 ; 2 , R
3 ; 3 , R
1 ; 3 , R
match n₁ 1
match n₁ 0
0 ; 2 , R
if/else
7 ; 7 , R
6 ; 6 , R
2 ; 2 , R
3 ; 3 , R
skip
processed
6 ; 6 , R
7 ; 7 , R
3 ; 3 , R
2 ; 2 , R
0 ; 6 , L
1 ; 3 , L
□ ; □ , L
reset
n₁
4 ; 0 , L
2 ; 0 , L
1 ; 1 , L
0 ; 0 , L
3 ; 1 , L
□ ; □ , R
reset
n₂
5 ; 0 , L
2 ; 0 , L
3 ; 1 , L
□ ; □ , L
$; > , L
7 ; 1 , L
6 ; 0 , L
>
2 ; 0 , L
6 ; 0 , L
5 ; 0 , L
7 ; 1 , L
3 ; 1 , L
$; $, L
n₂ prefix of n₁
end n₁
3 ; 1 , R
n₁ 1,|n₁| +1 = |n₂|
|3 ; ~ , R
+
$; + , L
skip
processed digits
3 ; 3 , R
7 ; 7 , R
5 ; 5 , R
2 ; 2 , R
6 ; 6 , R
1 ; 7 , L
0 ; 2 , L
□ ; □ , L
end n₂
n₂==0
5 ; 0 , S
skip
leading zeroes
0 ; 5 , R
1 ; 7 , L
3 ; 3 , R
2 ; 2 , R
5 ; 5 , R
6 ; 6 , R
7 ; 7 , R
□ ; □ , L
$; $, R
match n₁ 0
n₁ 0
1 ; 1 , R
0 ; 0 , R
End
5 ; 5 , R
$; $, R
n₁ processed, results depend on n₂
6 ; 6 , R
=
4 ; 0 , L
2 ; 0 , L
$; = , L
5 ; 0 , L
3 ; 1 , L
□ ; □ , R
if/else
input = 1$0
input ≠ 1$0
□ ; □ , L
+
$; + , L
0 ; 0 , L
1 ; 1 , L
2 ; 0 , L
3 ; 1 , L
4 ; 0 , L
5 ; 0 , L
6 ; 0 , L
7 ; 1 , L
□ ; □ , R
□ ; □ , R
<
4 ; 0 , L
2 ; 0 , L
$; < , L
6 ; 0 , L
5 ; 0 , L
7 ; 1 , L
0 ; 0 , L
1 ; 1 , L
3 ; 1 , L
3 ; 3 , R
2 ; 2 , R
skip
processed digits
2 ; 2 , R
3 ; 3 , R
□ ; □ , L
1 ; 1 , L
0 ; 0 , L
6 ; 6 , R
n₁ 1/n₂ 0
2 ; 2 , R
3 ; 3 , R
6 ; 6 , R
7 ; 7 , R
skip
processed digits
3 ; 3 , R
6 ; 6 , R
7 ; 7 , R
2 ; 2 , R
0 ; 0 , L
1 ; 1 , L
|n₂| > |n₁|
□ ; □ , L
|n₁|=|n₂|
1st diff, left→right,
n₁ 1/n₂ 0
>
4 ; 0 , L
$; > , L
2 ; 0 , L
6 ; 0 , L
5 ; 0 , L
7 ; 1 , L
1 ; 1 , L
0 ; 0 , L
3 ; 1 , L
□ ; □ , R
7 ; 7 , R
n₁ 0/n₂ 1
7 ; 7 , R
right
!□ ; ~ , R
□ ; □ , L
<
4 ; 0 , L
2 ; 0 , L
$; < , L
6 ; 0 , L
5 ; 0 , L
7 ; 1 , L
1 ; 1 , L
0 ; 0 , L
3 ; 1 , L
□ ; □ , R
End