Unit A4: Another Binary Adder Example, 2/6/05

After reviewing your TM binary adders, I came up with my own version. The main idea is similar to Jared's multiple-tape TM (but without using multiple tapes). I also borrowed Eric's idea of using the little endian format (but with the input formatting following the exercise sheet, i.e., 1+01=11 for 1+2=3, without using additional tape symbols for "carry"). If one still wants to get input in the big endian, it would be possible to add a preprocessor to convert the format, using Scott's binary number reverser.

Here is the explanation of the main mechanism. The TM would compare the leftmost bit of each segment. In the following example, there is nothing wrong with the part "0 + 1 = 1":

```
...␣01+11=101␣␣...
```

If it were "0 + 1 = 0", the TM can immediately tell the input is not acceptable, by terminating there. The TM will also need to record whether there will be carry. This can be done by using a distinct set of states. Thus, the TM distinguishes the following four acceptable cases: "0 + 0 = 0", "0 + 1 = 1", "1 + 0 = 1", "1 + 1 = 0" (with carry). Since the only information the TM needs for the next round is the presence/absence of carry, the processed bits can be replaced. Using the following replacement pattern, the tape configuration becomes analogous to the original configuration (except that there are repetitive delimiters, '+' and '=').

```
...␣␣1++1==01␣␣...
```

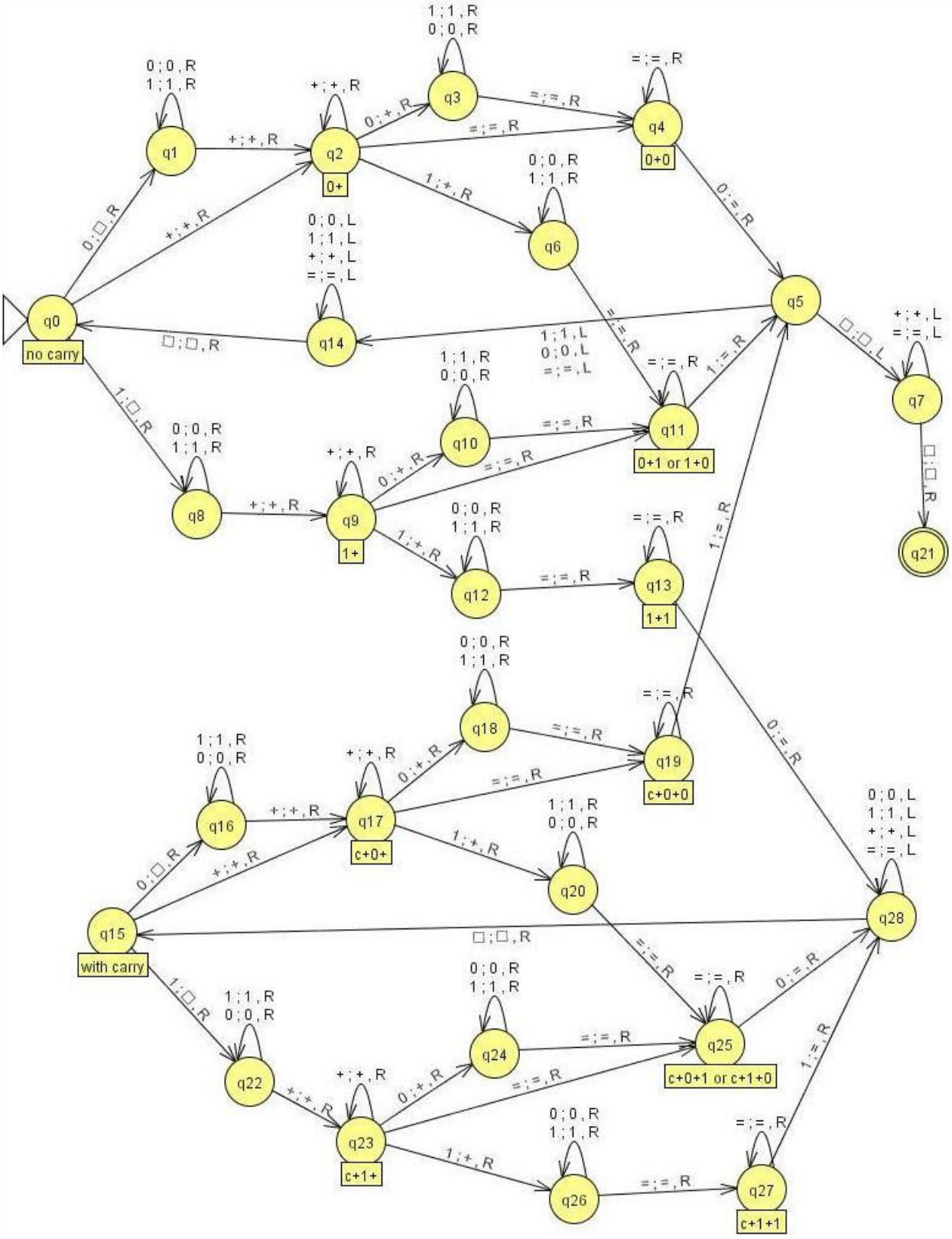
Then, we have an analogous, but smaller problem (cf. divide and conquer). The current focus "1 + 1 = 0" is correct if we process the carry bit in the next around. This TM contains two similar submachines: one for the case with no carry and the other for the one with carry. As done in Eric's machine, it is possible to encode the distinction on the tape (with fewer states, but more complex tape operation). Since the argument bits are already consumed, the TM interprets missing bit as 0. Then, "carry + 0 + 0 = 1".

```
...␣␣␣+++==1␣␣...
```

If the input survived this much, all the bits are consumed. The TM can then check for this by scanning only '+' and '='.

```
...␣␣␣+++===␣␣...
```

The transition diagram is on the next page, and the code (nk-adder.jff) is available in our JFLAP folder (http://www.tcnj.edu/~komagata/csc460/05s/JFLAP/). One unusual aspect of this version is that the input "+=0" will also be accepted because missing arguments across '+' are interpreted as 0. I tested with various inputs, but I have no doubt you can come with problematic inputs. If you test-drive this TM and identify a bug (or be convinced that it is correct), you will receive some reward.


q0
no carry
q1
q2
0+
q3
q4
0+0
q5
q6
q7
q8
q9
1+
q10
q11
0+1 or 1+0
q12
q13
1+1
q14
q15
with carry
q16
q17
c+0+
q18
q19
c+0+0
q20
q21
q22
q23
c+1+
q24
q25
c+0+1 or c+1+0
q26
q27
c+1+1
q28
0;0,R
1;1,R
+;+,R
=;=,R
0;+,R
1;+,R
0;□,R
□;□,R
1;□,R
0;=,R
1;=,R
0;0,L
1;1,L
+;+,L
=;=,L
1;1,L
0;0,L
□;□,L

// End