Review Sheet for Final of CS375

Spring, 2024

Things you need to know about the final exam:

- Final exam will be an in-person exam. The time and date are: 10:30am - 12:30pm, 04/29/2024 (Monday)

- If you need extra time for your final exam (and you have a letter from the DRC to verify your eligibility), please let me know by email before 04/27/2024. You don't take the exam in class, you take the exam on the 3rd floor of the Davis Marksbury Building (DMB). Please stop by my office (Room 303, DMB) before 12:58pm on 04/29/2024 (Monday) so I can show you where to take the exam. Your exam starts at 1:00pm (or a time good for you).

Final Exam will cover:

1. **C-F Languages & Pushdown Automata II**

   (slides 43 -)

2. **C-F Languages & Pushdown Automata III**

3. **C-F Languages & Pushdown Automata IV**

4. **Turing Machines I**

5. **Turing Machines II**

5. **Turing Machines III**

   (slides 1 - 26)

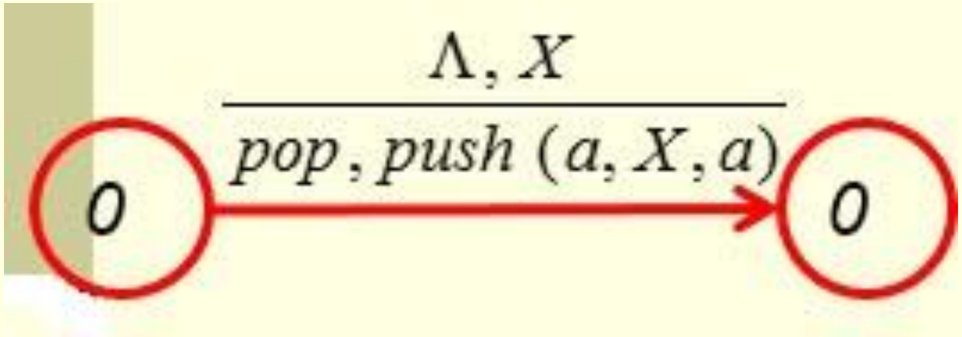
There will be 18-20 questions on this exam.

1. Know how to find a grammar for a language L by first building an empty-stack PDA to accept L and then transforming the PDA to a CFG. Know how to do this for the example in slide 48 of the notes “Context -free languages and Pushdown automata - II”.

2. Know NPDAs are more powerful than DPDAs by being able to show that even palindromes can be recognized by the NPDA in slide 62 of the notes “Context-free languages and Pushdown automata - II” but cannot be recognized by any DPDAs.

3. Know that Final-state acceptance and empty-stack acceptance are equivalent only for NPDAs. They are *not* equivalent for DPDAs. For DPDAs, the class of languages defined by final-state acceptance is bigger. **WHY?** (see slides 69-71 of the notes “Context-free languages and Pushdown automata - II”)

4. Know how to convert the following type 3 PDA instruction to a CFG production



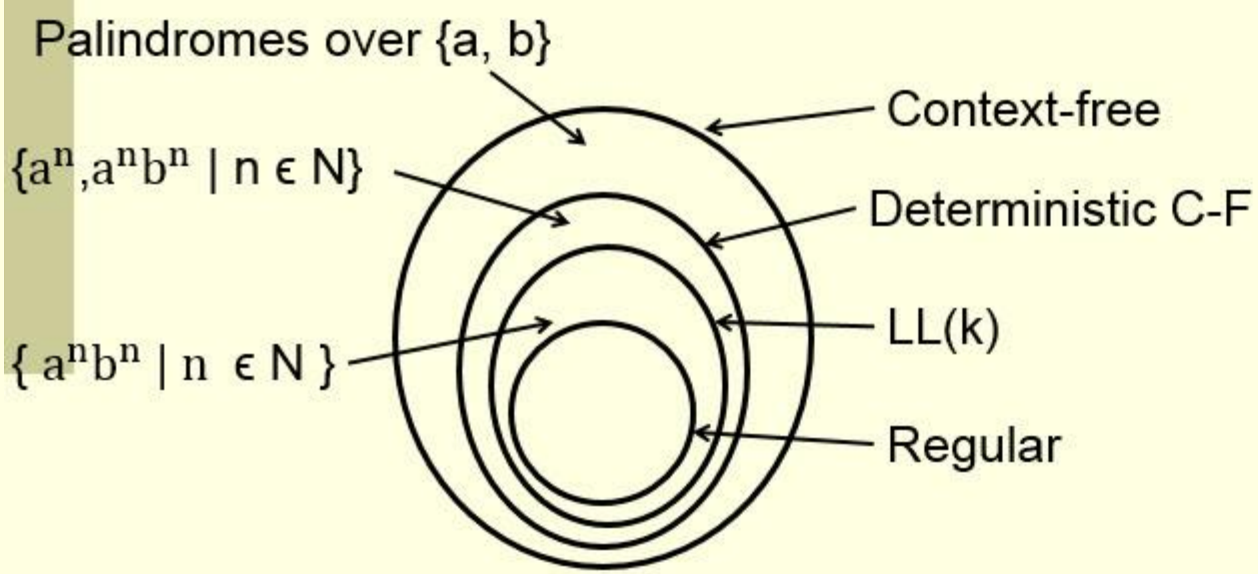
5. Know that CF languages are exactly those languages that are accepted by (non-deterministic) PDAs.

6. Know what is a parse tree, yield of a parse tree, left-most derivation, parsing, top-down parsing, and what is an LL(k) grammar/language.

7. Know how to find an LL(1) or LL(2) grammar for a given language such as $\{ a^n b \mid n \in \mathbf{N} \}$ and $\{ a^{m+n} b^m c^n \mid m,n \in \mathbf{N} \}$

8. Know how to find an LL(k+1) grammar for a given language that is not LL(k). Know how to do this for $\{ a^n b \mid n \in \mathbf{N} \}$.

9. Know the following structure and know where the set of non-deterministic C-F languages is. Know that $\{\ a^{n+k} b^n \mid k, n \in \mathbf{N}\ \}$ is nondeterministic context-free. So it has no LL(*k*) grammar for any *k*.



10. Know how to do *left-factoring* for an LL(*k*) grammar to obtain an equivalent LL(*n*) grammar with $n < k$.

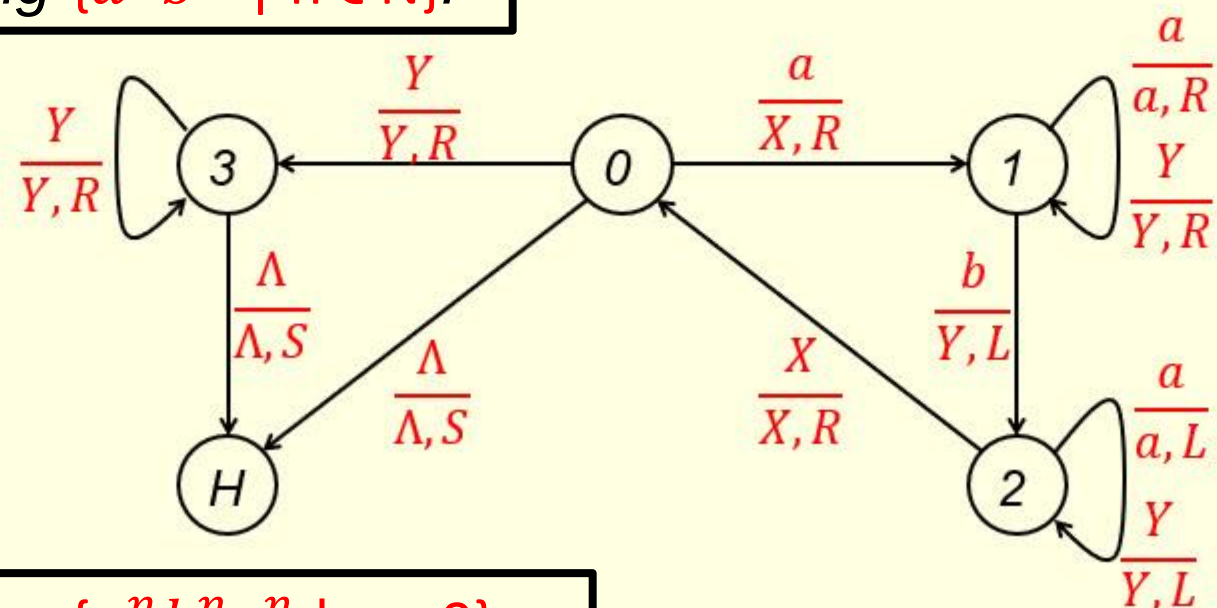
11. Know why left recursive grammars are not LL(k) for any k and know how to remove direct left recursion and indirect left recursion.

12. Know how to do top-down parsing of LL-languages

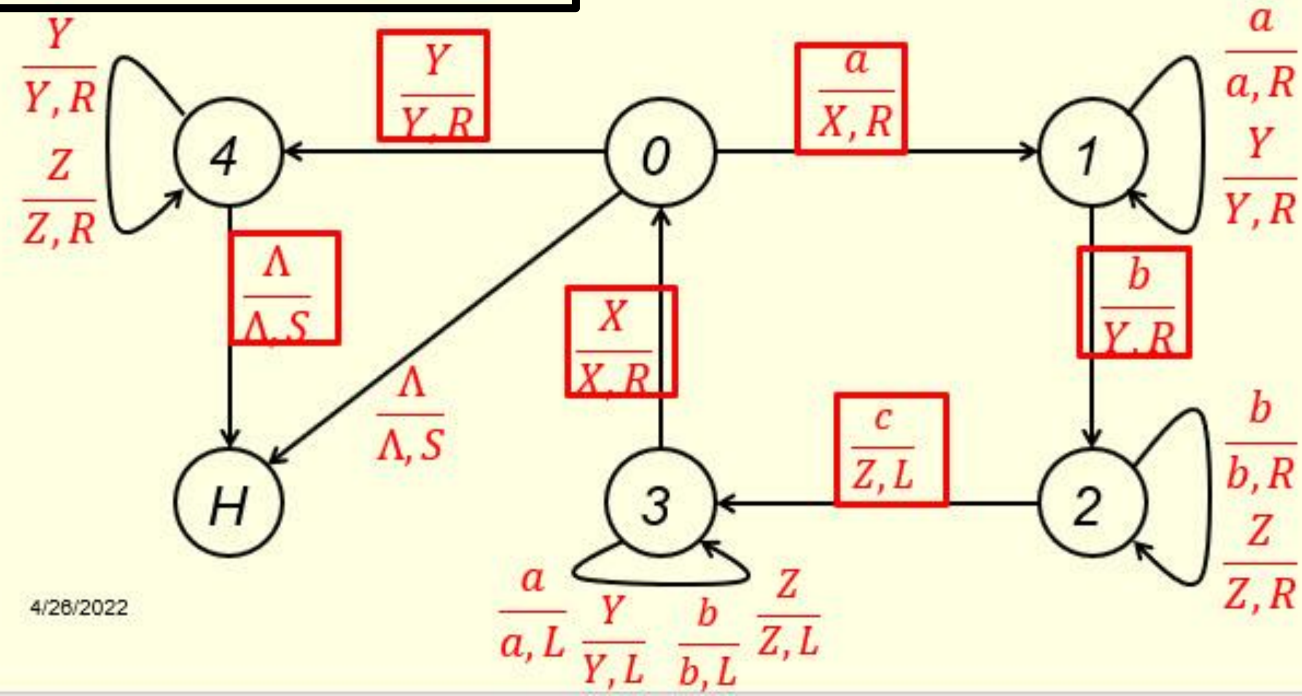
13. What is a Turing Machine (TM)? Know how to design a TM to accept a language such as $\{a^n b^n \mid n \in N\}$, $\{a^n b^n c^n \mid n > 0\}$ or even $\{a^n b^{n+m} c^n \mid n, m > 0\}$.

14. A TM is not built with a stack, but it can build implicit stacks. How many implicit stacks can a TM build?

TM accepting $\{a^n b^n \mid n \in N\}$:



TM accepting $\{a^n b^n c^n \mid n > 0\}$:



15. Know how to design a TM to add a given natural integer to 2, 4, or 8 (in binary form).

16. Know how to design a TM that can move a string one unit to the left/right or two units to the left/right.

17. The '**P vs. NP**' problem is a major unsolved problem in CS. If it turns out that P = NP (i.e., **all problems can be solved in polynomial time**) then there is no need to build quantum computers. Why?

18. Know how to design a TM that can perform addition on 2 given integers *m* and *n* or three given integers *m, n* and *p* (in unary form).

Add given integer to 8:

1	0	0	0

\+

	1	0	1	1	0	1	Λ	

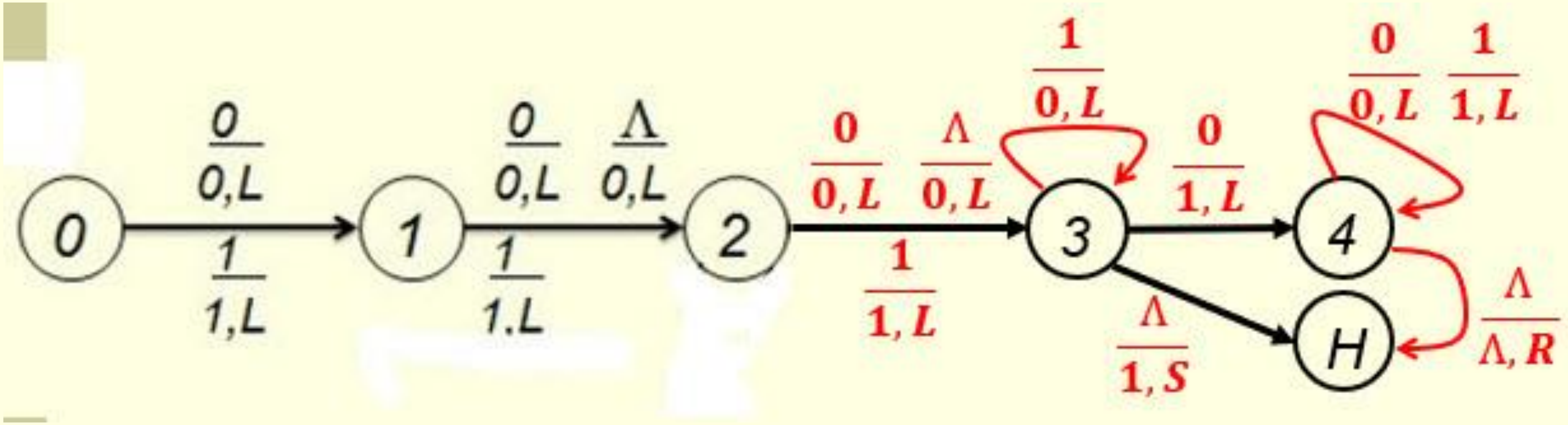
= ?

Move three cells left
(0, 0, 0, L, 1)
(0, 1, 1, L, 1)
(1, 0, 0, L, 2)
(1, 1, 1, L, 2)
(1, Λ, 0, L, 2)
(2, 0, 0, L, 3)
(2, 1, 1, L, 3)
(2, Λ, 0, L, 3)

Add 1:	
(3, 0, 1, L, 4)	Move left
(3, 1, 0, L, 3)	Carry
(3, Λ, 1, S, halt)	Done

Find left end of the string:	
(4, 0, 0, L, 4)	
(4, 1, 1, L, 4)	
(4, Λ, Λ, R, halt)	Done

State Transition Diagram:



Move string 2 units to the left:

Find a or b to move:

$(0, a, \Lambda, L, 11)$	found a
$(0, b, \Lambda, L, 21)$	found b
$\boxed{(0, \Lambda, \Lambda, L, 41)}$	no more a's or b's

Move to left end of output:

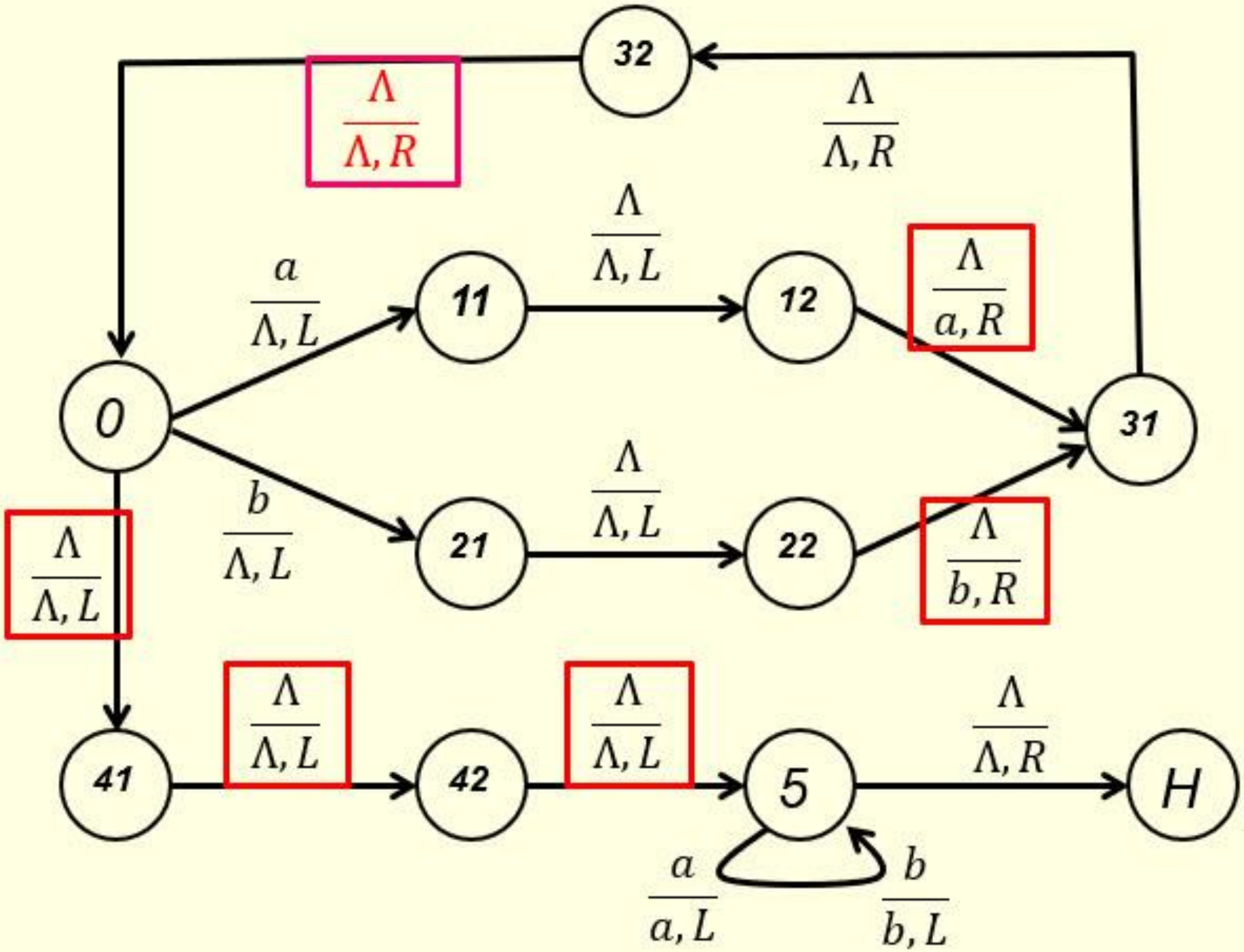
$\boxed{(41, \Lambda, \Lambda, L, 42)}$	skip Λ
$\boxed{(42, \Lambda, \Lambda, L, 5)}$	skip Λ
$(5, a, a, L, 5)$	skip a
$(5, b, b, L, 5)$	skip b
$(5, \Lambda, \Lambda, R, halt)$	Done

Write a or b:

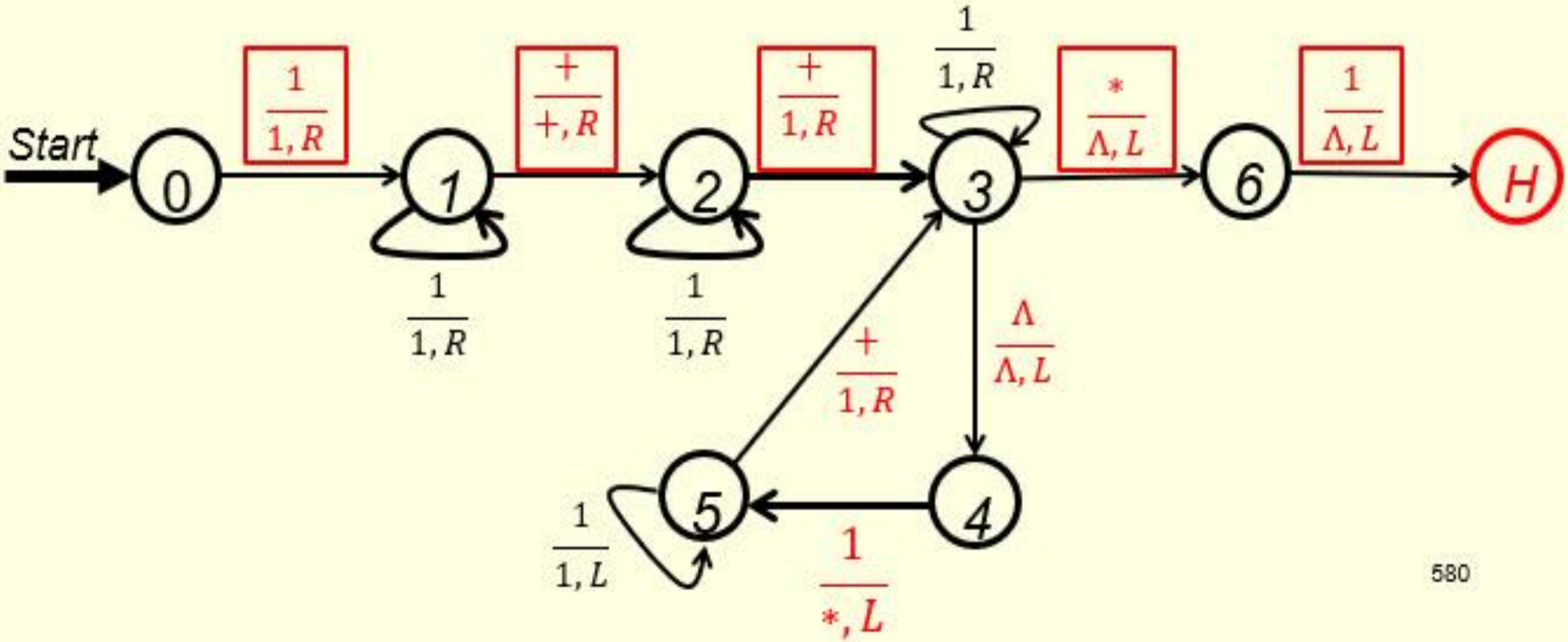
$(11, \Lambda, \Lambda, L, 12)$	skip Λ
$\boxed{(12, \Lambda, a, R, 31)}$	write a
$(21, \Lambda, \Lambda, L, 22)$	skip Λ
$\boxed{(22, \Lambda, b, R, 31)}$	write b
$(31, \Lambda, \Lambda, R, 32)$	skip Λ
$\boxed{(32, \Lambda, \Lambda, R, 0)}$	skip Λ

of Kentucky 188

State Transition Diagram:

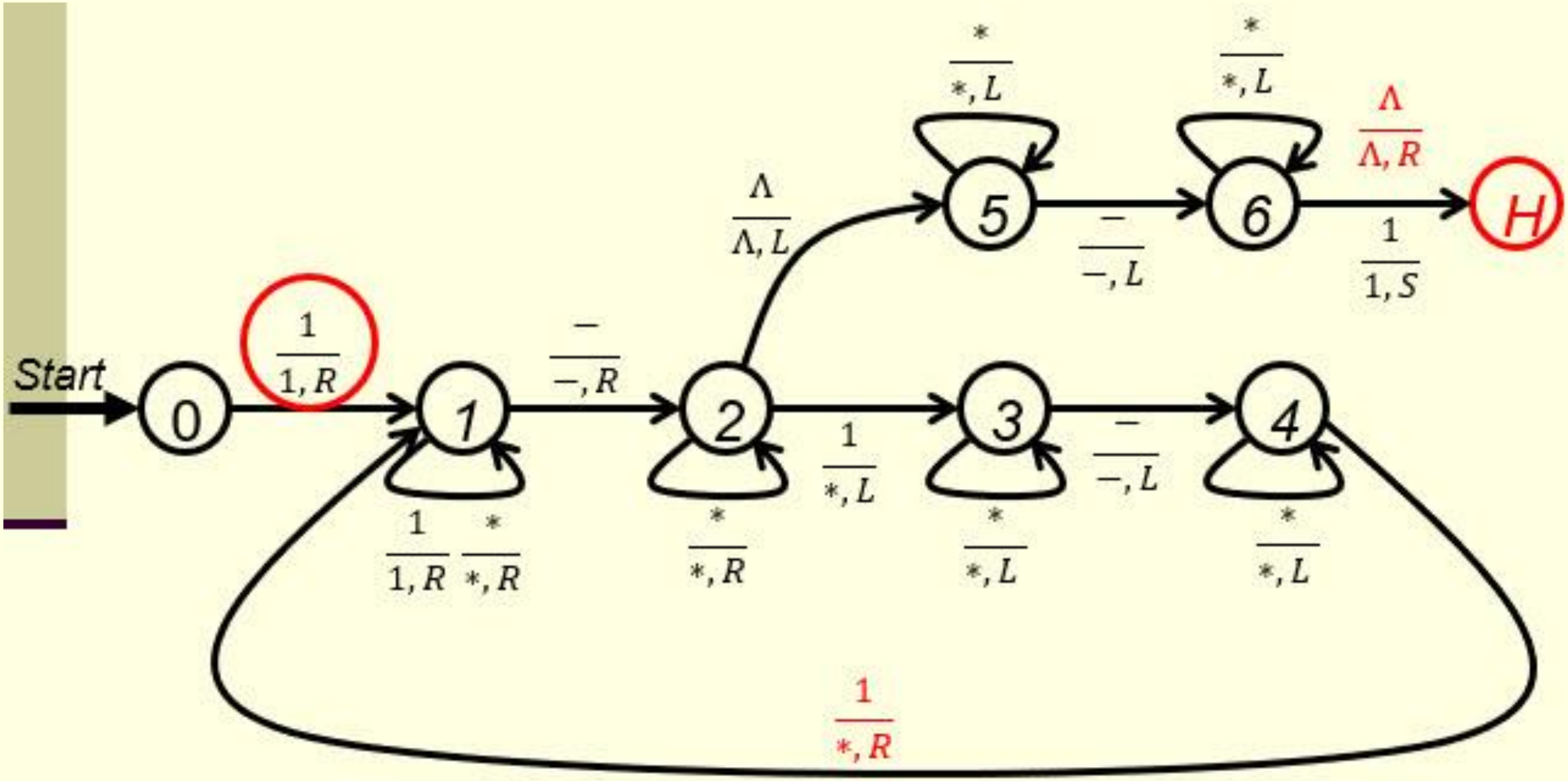


Addition on three numbers:

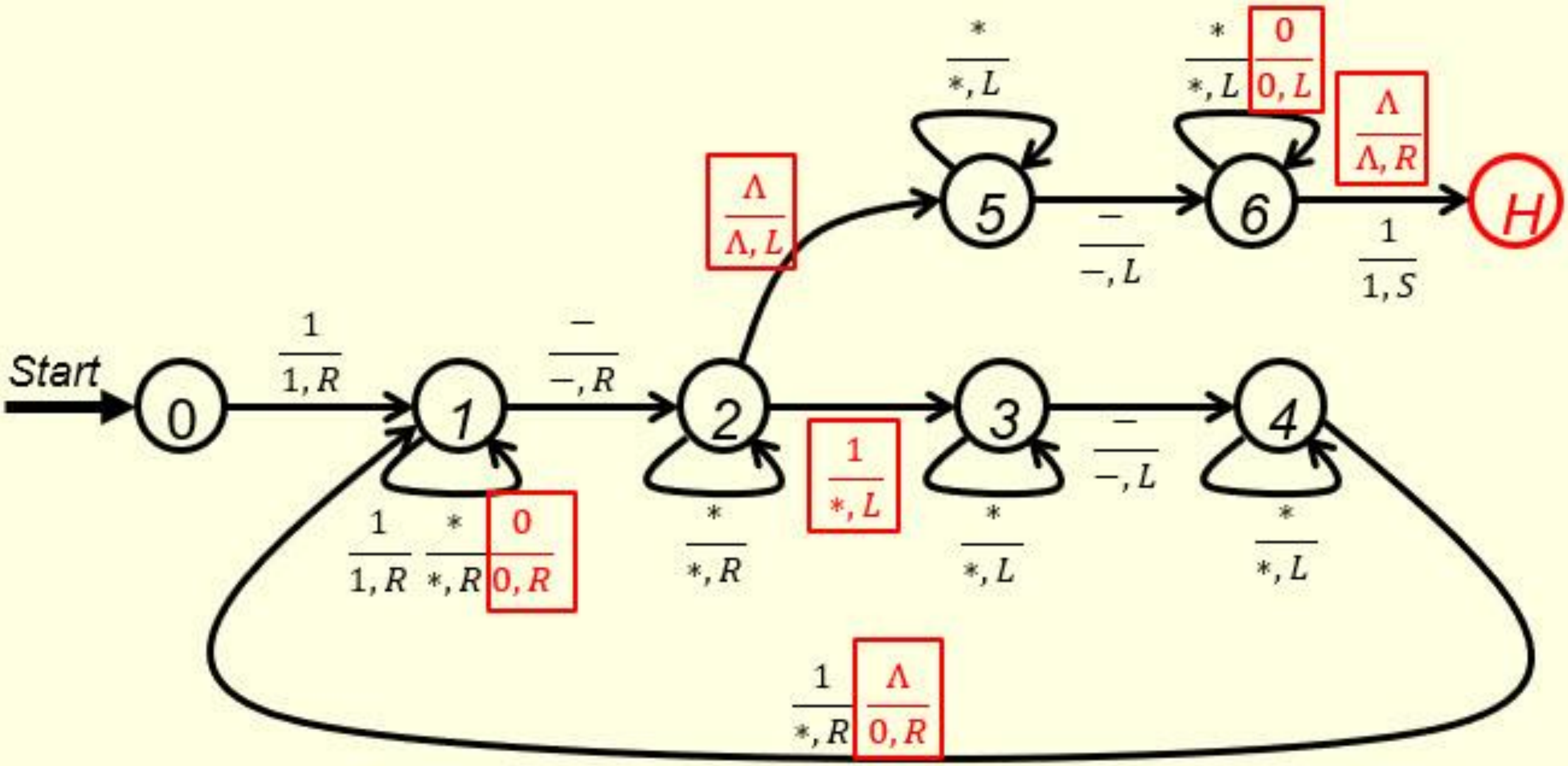


19. Know how to design a TM that can perform subtraction on 2 given integers *m* and *n* in unary form (*m* is not required to be bigger than or equal to *n*).

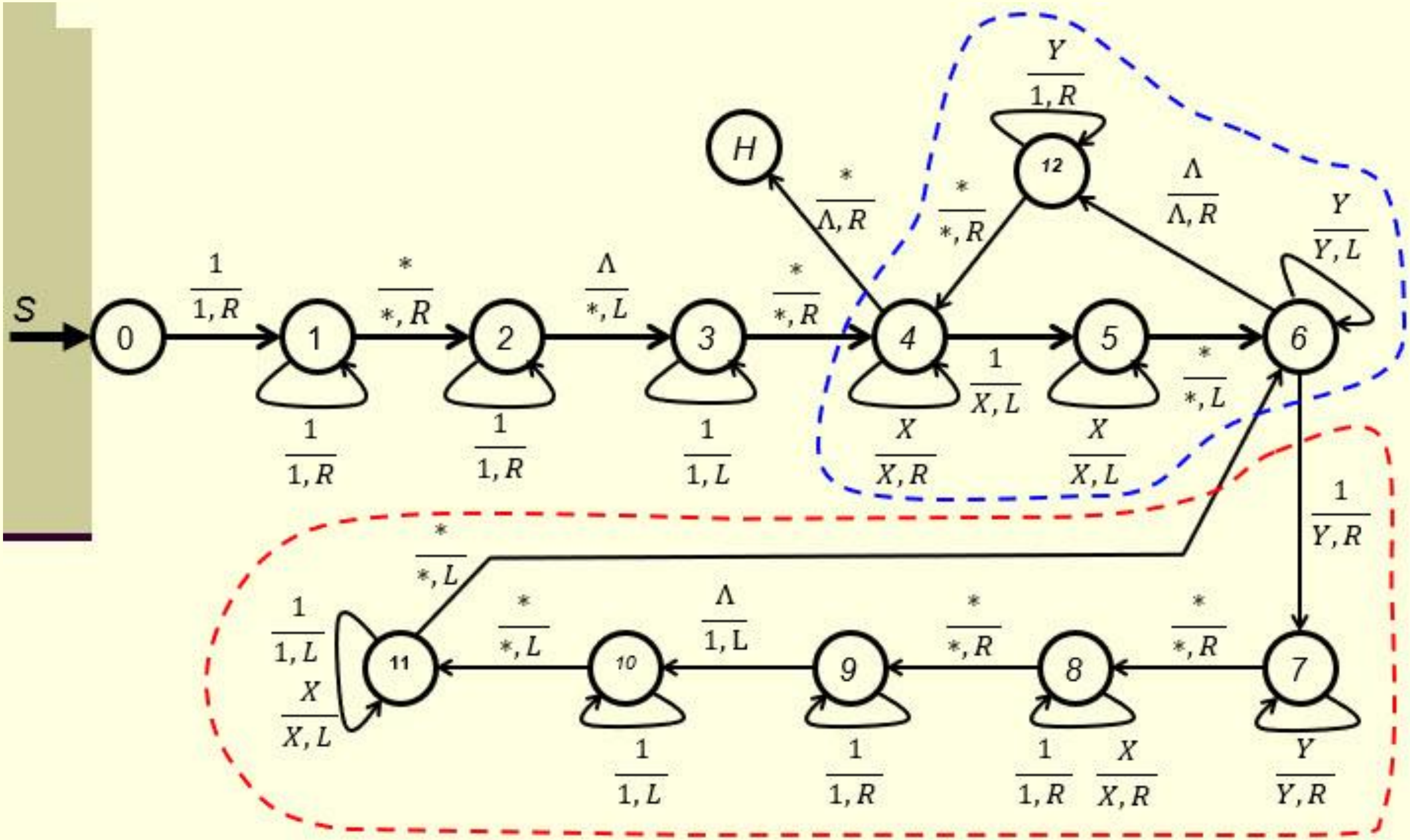
Subtraction when $m \geq n$:



Subtraction when $m \geq n$ *or* $m < n$:

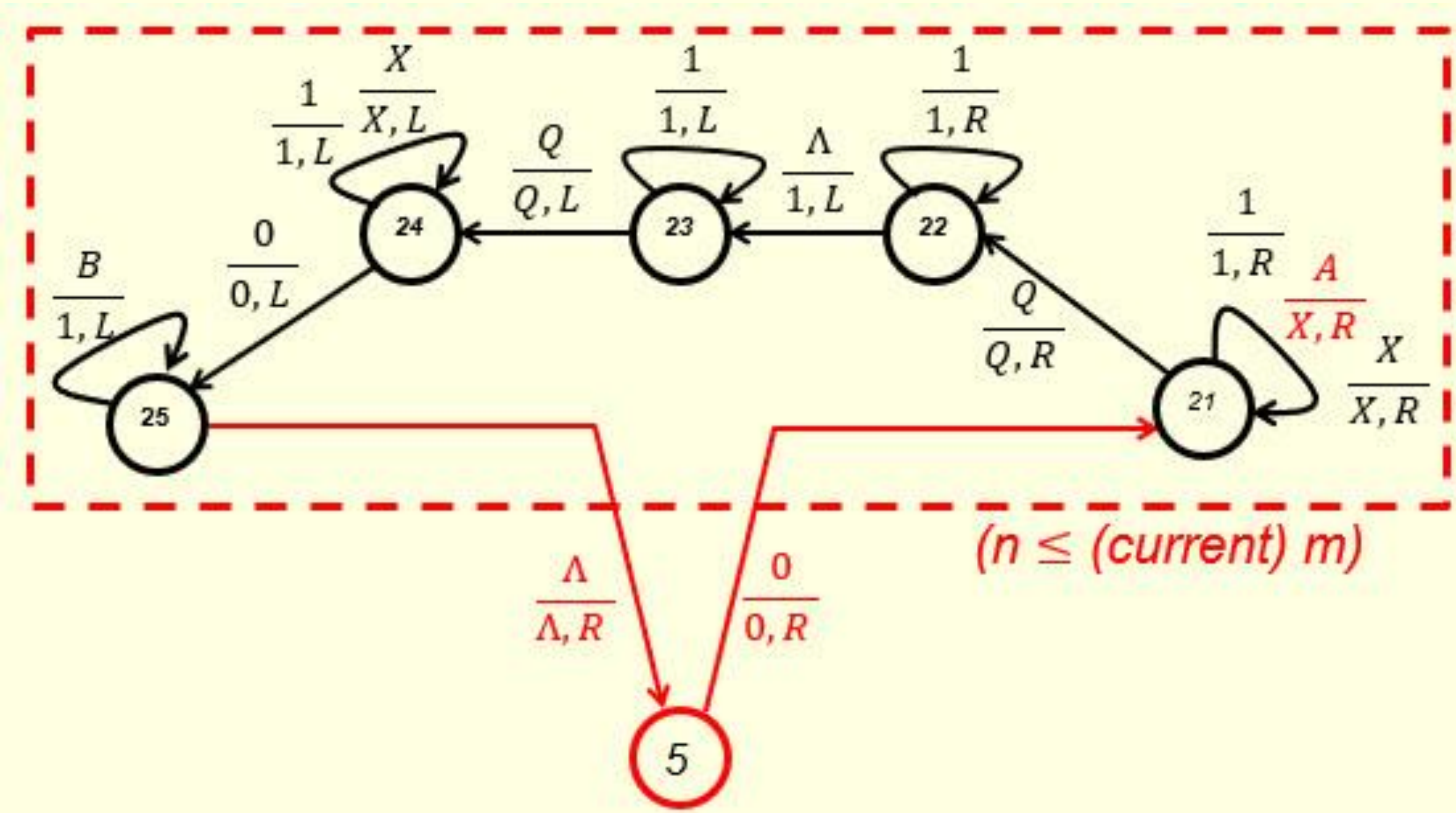


20. Know how to design a TM that can perform multiplication on 2 given integers *m* and *n* in unary form.



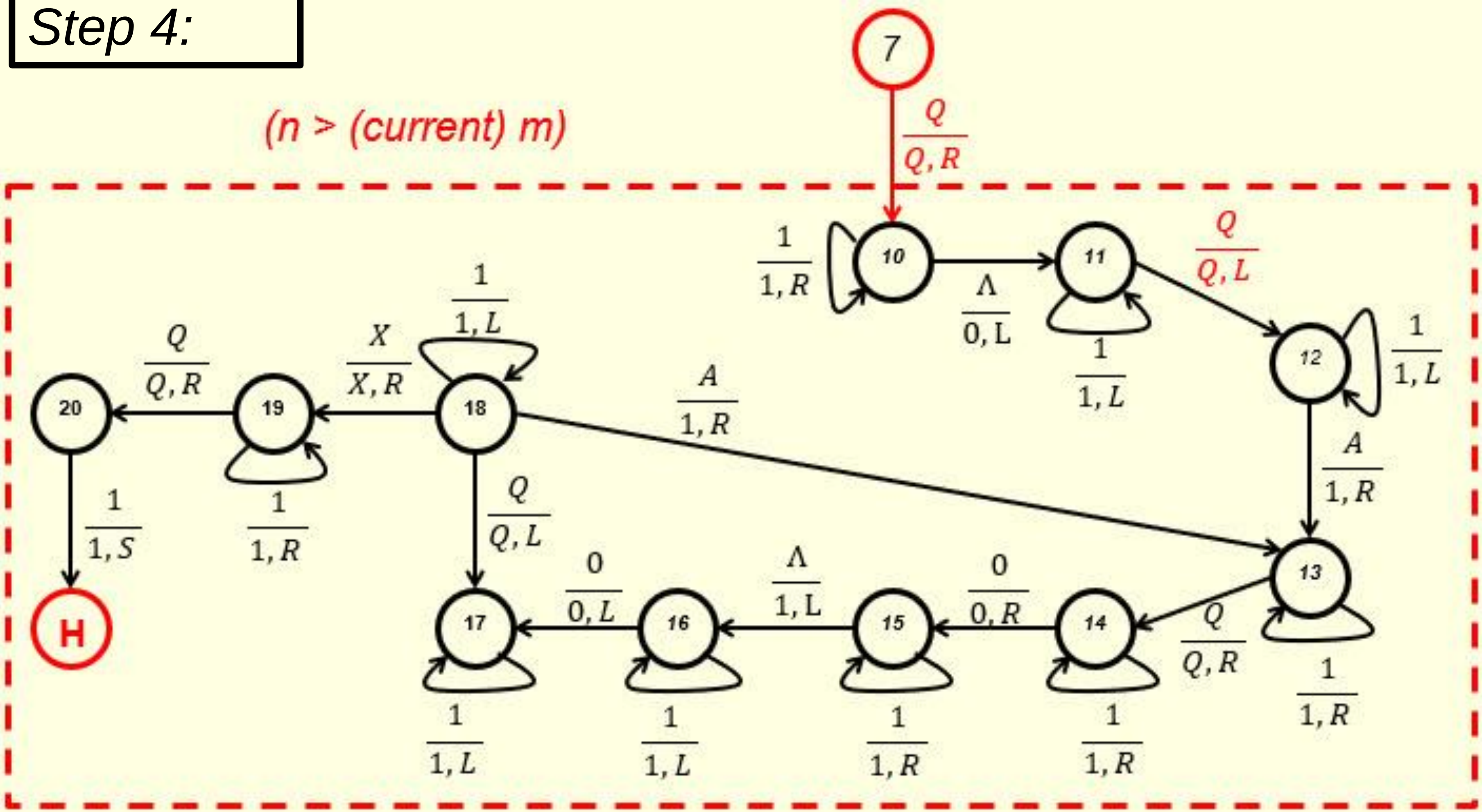
21. Know how to design a TM that can perform division on 2 given positive integers *m* and *n* in unary form, especially on the portions that perform step 3 and step 4.

Step 3:



21. Know how to design a TM that can perform division on 2 given positive integers *m* and *n* in unary form, especially on the portions that perform step 3 and step 4.

Step 4:



22. The Church-Turing Thesis has two versions.

Version 1:

A problem can be solved by an algorithm if and only if it can be solved by a Turing machine.

Version 2:

Anything that is intuitively computable can be computed by a Turing machine.

The first version is an if and only if statement, the second statement is not. Does this mean the other direction of the second version is not true?

No. the other direction of version 2 is intuitively true, so there is no need to include the other direction.

23. Church-Turing Thesis is not a theorem, but a thesis. **Why?**

> Because nobody can prove it, and nobody can disprove it either.

24. **Must** know how to do each question of HW6. HW7, HW8, HW9, and HW10.

The End