Middle East Technical University Department of Computer Engineering

# CENG 424

Logic for Computer Science

Fall 2024-2025

Assignment 5/Term Project

Name SURNAME: Aly Asad GILANI
Student ID: 2547875

## Answer 1

Before writing the pseudocode, I will first define each relational and functional formula I will use so it is clear what I am doing.

1. **Basic Transitions:**
   Firstly, I will define each transition given in the original NWFSM as a premise of the form:

   $$T(q, p, \sigma, w)$$

   where there is a transition from state $q$ to state $p$ on input symbol $\sigma$, and $w$ is the weight of that transition.

2. **Starting a Path:**
   Since we will be given $s$ the starting state, all our paths should start from $s$. Therefore, I will add this rule to the premises:

   $$P(s, \epsilon, 0)$$

   This means that to start a path, we need to start at state $s$, with empty input, and 0 weight. This will be used with the next rule to find paths.

3. **Rule of Transitivity:**
   To the premises, I will add a rule of the form:

   $$P(x.y, \sigma_1, w_1) \land T(y, z, \sigma_2, w_2) \to P(x.y.z, \sigma_1\sigma_2, w_1 + w_2)$$

   This means that if there is a path with previous state(s) $x$ to state $y$ with input $\sigma_1$ and weight $w_1$, and there is a transition from state $y$ to state $z$ with input $\sigma_2$ and weight $w_2$, then through transitivity, there is a path from previous state(s) $x.y$ to $z$ with input $\sigma_1\sigma_2$ concatenated with each other, and weight $w_1 + w_2$ added to each other. This rule allows us to find a path containing the input string. The variable $x$ can be any number of previous states defining a path, even 0. $y$ and $z$ will be 1 state each.

4. **Finished Path:**
   Once a path has traced the input string (*input*) on the NWFSM, it should finish and save its route and total weight. For this, I will add the following premise:

   $$P(x, input, w) \rightarrow Finished(x, w)$$

   This checks if the path has correctly traversed the input the input string, then saves the route it took ($x$) and the total weight of that route ($w$).

   For example for an input route $x_1$ and its weight $w_1 = 5$, and another input route $x_2$ and its weight $w_2 = 8$, $maxW(w_1, w_2)$ will return $w_2 = 8$, and $maxP(x_1, w_1, x_2, w_2)$ will return $x_2$ since its weight is greater. These functions wont be explicitly defined in the premises, but will be used in the formation of other premises.

5. **Extracting the goal:**
   In the end, we should be left with one (or more of the same weight) path that has the highest weight among all others. We should extract the answer from that path:

   $$Finished(x_1, w_1) \land \forall_{x_2,w_2}(Finished(x_2, w_2) \land w_1 \geq w_2) \rightarrow goal(x_1, w_1)$$

   This means that our goal is a path whose weight is greater than or equal to all other paths.

Now, we can finally write the pseudocode to represent any given NWFSM and input string into a collection of Relational Logic premises:

**Algorithm 1** Generating Relational Logic Premises

```
procedure RelLogic(NWFSM $N = \{Q, \Sigma, \delta, s, w\}$, input string $inStr$)

    Premises = {}

    for each transition $t = (q, \sigma, q') \in \delta$ do
        Add $T(q, q', \sigma, w(t))$ to Premises
    end for

    Add $P(s, \epsilon, 0)$ to Premises

    Add $P(x.y, \sigma_1, w_1) \land T(y, z, \sigma_2, w_2) \rightarrow P(x.y.z, \sigma_1\sigma_2, w_1 + w_2)$ to Premises

    Add $P(x, inStr, w) \rightarrow Finished(x, w)$ to Premises

    Add $Finished(x_1, w_1) \land \forall_{x_2,w_2}(Finished(x_2, w_2) \land w_1 \geq w_2) \rightarrow goal(x_1, w_1)$ to Premises

    return Premises
end procedure
```

# Answer 2

For this question, I will divide the answer into two parts: first I will represent the problem in relational logic sentences using my pseudocode, then I will solve it to find the highest weighted path.

## 2.1 Representing in Relational Logic

(i) First, our premises are empty {}.

(ii) From lines 5-7 from my pseudocode, I start to add the transitions as premises:

   (a) $T(q_0, q_1, a, w(q_0, a, q_1)) = T(q_0, q_1, a, 4)$
   (b) $T(q_0, q_2, b, 2)$
   (c) $T(q_0, q_3, a, 3)$
   (d) $T(q_0, q_3, b, 4)$
   (e) $T(q_1, q_1, b, 4)$
   (f) $T(q_1, q_3, \epsilon, 2)$
   (g) $T(q_2, q_0, a, 2)$
   (h) $T(q_2, q_3, a, 4)$
   (i) $T(q_3, q_1, b, 1)$
   (j) $T(q_3, q_2, \epsilon, 7)$

(iii) From line 9 from my pseudocode, I add the starting path to premises. Since it is given that $s = q_0$, we substitute:

$$P(q_0, \epsilon, 0)$$

Converting to clausal form using INSEADO:

$$\{P(q_0, \epsilon, 0)\}$$

(iv) From line 11 from my pseudocode, I add the rule of transitivity to premises:

$$P(x.y, \sigma_1, w_1) \land T(y, z, \sigma_2, w_2) \rightarrow P(x.y.z, \sigma_1\sigma_2, w_1 + w_2)$$

Converting to clausal form using INSEADO:

$$\neg(P(x.y, \sigma_1, w_1) \land T(y, z, \sigma_2, w_2)) \lor P(x.y.z, \sigma_1\sigma_2, w_1 + w_2)$$

$$\neg P(x.y, \sigma_1, w_1) \lor \neg T(y, z, \sigma_2, w_2) \lor P(x.y.z, \sigma_1\sigma_2, w_1 + w_2)$$

$$\{\neg P(x.y, \sigma_1, w_1), \neg T(y, z, \sigma_2, w_2), P(x.y.z, \sigma_1\sigma_2, w_1 + w_2)\}$$

(v) From line 13 from my pseudocode, I add the finished path to premises. Since the input string is *abb*, we substitute it into the formula:

$$P(x, abb, w) \to Finished(x, w)$$

Converting to clausal form using INSEADO:

$$\neg P(x, abb, w) \lor Finished(x, w)$$

$$\{\neg P(x, abb, w), Finished(x, w)\}$$

(vi) From line 15 from my pseudocode, I add the premise to extract the goal:

$$Finished(x_1, w_1) \land \forall_{x_2,w_2}(Finished(x_2, w_2) \land w_1 \geq w_2) \to goal(x_1, w_1)$$

Converting to clausal form using INSEADO:

$$\neg(Finished(x_1, w_1) \land \forall_{x_2,w_2}(Finished(x_2, w_2) \land w_1 \geq w_2)) \lor goal(x_1, w_1)$$

$$\neg Finished(x_1, w_1) \lor \neg\forall_{x_2,w_2}(Finished(x_2, w_2) \land w_1 \geq w_2) \lor goal(x_1, w_1)$$

Since we need to compare our best path with every other path, I will not simplify the middle term further to avoid loss of generality or accuracy in the logical representation. We need to make sure that the comparison is explicit. We can let the

$$\{\neg Finished(x_1, w_1), \neg\forall_{x_2,w_2}(Finished(x_2, w_2) \land w_1 \geq w_2), goal(x_1, w_1)\}$$

## 2.2 Finding highest weighted path using Relational Resolution

Now that we have found the relational logic premises for our problem, we are ready to solve it using relational resolution.

**Note #1:** I will add an additional premise to perform logical comparisons like $a > b$ over a set, which most tools generally have.

**Note #2:** Since it is not explicitly stated in the text that empty transitions at the end of a path are not accepted, the program will run assuming that they are accepted to find ALL the possible paths.

| | | |
|---|---|---|
| 1. | $\{T(q_0, q_1, a, 4)\}$ | Premise from (ii) |
| 2. | $\{T(q_0, q_2, b, 2)\}$ | Premise from (ii) |
| 3. | $\{T(q_0, q_3, a, 3)\}$ | Premise from (ii) |
| 4. | $\{T(q_0, q_3, b, 4)\}$ | Premise from (ii) |
| 5. | $\{T(q_1, q_1, b, 4)\}$ | Premise from (ii) |
| 6. | $\{T(q_1, q_3, \epsilon, 2)\}$ | Premise from (ii) |
| 7. | $\{T(q_2, q_0, a, 2)\}$ | Premise from (ii) |
| 8. | $\{T(q_2, q_3, a, 4)\}$ | Premise from (ii) |
| 9. | $\{T(q_3, q_1, b, 1)\}$ | Premise from (ii) |
| 10. | $\{T(q_3, q_2, \epsilon, 7)\}$ | Premise from (ii) |
| 11. | $\{\{P(q_0, \epsilon, 0)\}\}$ | Premise from (iii) |
| 12. | $\{\neg P(x.y, \sigma_1, w_1), \neg T(y, z, \sigma_2, w_2), P(x.y.z, \sigma_1\sigma_2, w_1 + w_2)\}$ | Premise from (iv) |
| 13. | $\{\neg P(x, abb, w), Finished(x, w)\}$ | Premise from (v) |
| 14. | $\{\neg Finished(x_1, w_1), \neg\forall_{x_2,w_2}(Finished(x_2, w_2) \wedge w_1 \geq w_2), goal(x_1, w_1)\}$ | Premise from (vi) |
| 15. | *Comparison operations* | Premise from Note #1 |
| 16. | $\{\neg T(q_0, z, \sigma_2, w_2), P(q_0.z, \sigma_2, w_2)\}$ | 11, 12 |
| 17. | $\{P(q_0.q_1, a, 4)\}$ | 1, 16 |
| 18. | $\{\neg T(q_1, z, \sigma_2, w_2), P(q_0.q_1.z, a\sigma_2, 4 + w_2)\}$ | 12, 17 |
| 19. | $\{P(q_0.q_1.q_1, ab, 8)\}$ | 5, 18 |
| 20. | $\{\neg T(q_1, z, \sigma_2, w_2), P(q_0.q_1.q_1.z, ab\sigma_2, 8 + w_2)\}$ | 12, 19 |
| 21. | $\{P(q_0.q_1.q_1.q_1, abb, 12)\}$ | 5, 20 |
| 22. | $\{Finished(q_0.q_1.q_1.q_1, 12)\}$ | 13, 21, Path #1 |
| 23. | $\{\neg T(q_1, z, \sigma_2, w_2), P(q_0.q_1.q_1.q_1.z, abb\sigma_2, 12 + w_2)\}$ | 12, 21 |
| 24. | $\{P(q_0.q_1.q_1.q_1.q_3, abb, 14)\}$ | 6, 23 |
| 25. | $\{Finished(q_0.q_1.q_1.q_1.q_3, 14)\}$ | 13, 24, Path #2 |
| 26. | $\{\neg T(q_3, z, \sigma_2, w_2), P(q_0.q_1.q_1.q_1.q_3.z, abb\sigma_2, 14 + w_2)\}$ | 12, 24 |
| 27. | $\{Finished(q_0.q_1.q_1.q_1.q_3.q_2, 21)\}$ | 13, 26, Path #3 |

To avoid an extremely long derivation, I will skip over the intermediate steps required to derive the rest of the ”Finished paths” as we have already seen how paths 1-3 are derived using them. I will write the resulting paths and work from there:

| | | |
|---|---|---|
| 28. | $\{Finished(q_0.q_1.q_1.q_3.q_1, 11)\}$ | Path #4 |
| 29. | $\{Finished(q_0.q_1.q_1.q_3.q_1.q_3, 13)\}$ | Path #5 |
| 30. | $\{Finished(q_0.q_1.q_1.q_3.q_1.q_3.q_2, 20)\}$ | Path #6 |
| 31. | $\{Finished(q_0.q_1.q_3.q_1.q_1, 11)\}$ | Path #7 |
| 32. | $\{Finished(q_0.q_1.q_3.q_1.q_1.q_3, 13)\}$ | Path #8 |
| 33. | $\{Finished(q_0.q_1.q_3.q_1.q_1.q_3.q_2, 20)\}$ | Path #9 |
| 34. | $\{Finished(q_0.q_1.q_3.q_1.q_3.q_1, 10)\}$ | Path #10 |
| 35. | $\{Finished(q_0.q_1.q_3.q_1.q_3.q_1.q_3, 12)\}$ | Path #11 |
| 36. | $\{Finished(q_0.q_1.q_3.q_1.q_3.q_1.q_3.q_2, 19)\}$ | Path #12 |
| 37. | $\{Finished(q_0.q_3.q_1.q_1, 8)\}$ | Path #13 |
| 38. | $\{Finished(q_0.q_3.q_1.q_1.q_3, 10)\}$ | Path #14 |
| 39. | $\{Finished(q_0.q_3.q_1.q_1.q_3.q_2, 17)\}$ | Path #15 |
| 40. | $\{Finished(q_0.q_3.q_1.q_3.q_1, 7)\}$ | Path #16 |
| 41. | $\{Finished(q_0.q_3.q_1.q_3.q_1.q_3, 9)\}$ | Path #17 |
| 42. | $\{Finished(q_0.q_3.q_1.q_3.q_1.q_3.q_2, 16)\}$ | Path #18 |
| 43. | $\{\neg\forall_{x_2,w_2}(Finished(x_2, w_2) \land 21 \geq w_2), goal(q_0.q_1.q_1.q_1.q_3.q_2, 21)\}$ | 14, 27 |
| 44. | $\{goal(q_0.q_1.q_1.q_1.q_3.q_2, 21)\}$ | 15, 43 |

In the last step, the comparison operator compared the current path with every other path to make sure it is the greatest.

As we have arrived at our goal, we see that the highest weighted path for the given NWFSM $N_1$ with input *abb* is $\boldsymbol{q_0 \xrightarrow{a,4} q_1 \xrightarrow{b,4} q_1 \xrightarrow{b,4} q_1 \xrightarrow{\epsilon,2} q_3 \xrightarrow{\epsilon,7} q_2}$ with a weight of **21**.

**Note:** The above is the highest weighted path considering the NWFSM can accept empty transitions even after the input is exhausted. Otherwise, the highest weighted path would be: $\boldsymbol{q_0 \xrightarrow{a,4} q_1 \xrightarrow{b,4} q_1 \xrightarrow{b,4} q_1}$ with a weight of **12**.

# Answer 3

The best class/type of tool is most probably **Logic Programming Languages**, such as **Prolog** that we have studied in the CENG242 Programming Language Concepts course. Its input structure is quite similar to the relational logic premises that we make.

## 3.1 Changes

(i) Most of the premises can be translated directly into "facts". For example, we had a premise $T(q_0, q_1, a, 4)$. We can translate this into Prolog as:

$$trans(q0, q1, a, 4).$$

(ii) For all the implications, we can make a "rule" in Prolog. For example, we had the implication $P(x, abb, w) \rightarrow Finished(x, w)$ in the premises. We can translate this into Prolog as:

$$finished(X, W) \text{ :- } path(X,'abb', W).$$

(iii) In the premises, we need to search over all the other paths to make sure the currently selected one has the greatest weight. We can easily add that as a premise in Prolog as well by using the "findall" predicate along with the comparison operators like ">=" to achieve this goal.

(iv) In the end, we can convert all our premises into "facts" and "rules" in Prolog then query our "goal" to get the answer.

## 3.2 How do the tools work

Logic programming languages, especially Prolog, work similar to the methods we have studied in this course. It tries to find the goal by making a tree-like depth-first search, by applying its "rules" until it exhausts each path in the tree to get the required answer. If a path is exhausted, it backtracks to query the next path and so on. This is similar to how we approach relational resolution. If we need the goal, we solve backwards from it until we achieve the answer.