Structured Programming and Formal Methods Final Exam

Yu Chen

September 22, 2018

1 New Proposed Logic

I would like to build out a bridge between the binary certainty of first-order logic, with the probabilistic uncertainty of the real world. However, one of the challenges with nesting probability distributions within logic is that it becomes difficult to reason about posterior distributions and "updates" to the likelihoods (it is for this very reason why conjugate distributions are so useful, since they allow for extremely easy updates from prior to posterior distributions without using compute-heavy numerical solutions like sampling). Thus, I will extend the Propositional Logic of Unit 1 in the following ways:

- An additional connective called **quorom**, represented by the symbol γ, which evaluates to true when at least half of the formulas within it are true. For instance $\gamma(\phi, \phi, \neg\phi)$ will evaluate to **True**, $\gamma(\phi, \neg\phi, \neg\phi)$ will evaluate to **False**, and $\gamma(\phi, \phi, \neg\phi\neg\phi)$ will evalute to **True** since it is at least half. Not that we can use our Propositional logic and introduction rule to write this as $\gamma(\phi \wedge \phi \wedge \neg\phi \wedge ...)$. **Note: ϕ within both γ and ϵ as described below has a few small differences from ϕ within traditional propositional logic- read the section of non-valid formulas for further detaisl.**

- An additional connective called **partitionable**, represented by the symbol ϵ, which evaluates to true when a collection of formulas has an even, nonzero number of true formulas. For instance, $\epsilon(\phi_1, \phi_2, \neg\phi_3, \neg\phi_4)$ will evaluate to true, whereas $\epsilon(\phi_1, \neg\phi_2, \neg\phi_3, \neg\phi_4)$ will evaluate to false

- Finally, an additional shorthand connective for **exclusive or**, called χ. It is of the form $\phi\chi\phi$. For instance, $\phi\chi\phi$ evaluates to false, $\phi\chi\neg\phi$ evaluates to true, and $\neg\phi\chi\neg\phi$ evaluates to false.

Furthermore, I extend propositional logic into the realm of probabilistic modeling by introducing a beta distribution wrapper around the highest-level formula. A beta distribution is a key distribution within machine learning, used to model phenomenon bounded between 0 and 1 in output values (we use it heavily, for instance, to model clickthrough rate, likelihood of churn, etc.). Thus, it is a well-suited distribution- compared to a Gaussian (normal) distribution to model the probability of an event occur, since its support is $[0, 1]$. The two parameters of a beta distribution are positive, real-values α and β, with its expected value $E[X] = \frac{\alpha}{\alpha+\beta}$. For instance, if we model the probability of me answering correctly on a multiple choice question with a beta distribution $B(\alpha = 1, \beta = 1)$, this evaluates to correctly answering with a probability of $\frac{\alpha}{\alpha+\beta} = \frac{1}{1+1} = 0.5$.

With this new machinery, we can express gradients of probability other than a binary true or false. For instance, let's assign ϕ_1 to be **Yu Chen sleeps at least 8 hours tonight.** and ϕ_2 to be **Yu Chen is not cranky in the morning**, and $B(1, 3) = \phi_1 \wedge \phi_2$. In natural language, this amounts to **there is a 25 percent chance Yu Chen sleeps at least 8 hours tonight and is not cranky in the morning.**

However, the formula $((B(1, 3) = \phi_1) \vee \phi_2)$ is not well-formed, because we cannot assign a probability to a lower-level formula (only the top-most).

1.1 Examples of Non-Valid Formulas

All of the standard examples of wff apply here from Propositional Logic, so I will highlight only some of the key areas of distinction:

- $\phi\chi$ - χ requires another formula on its righthand side

- $\phi\epsilon\phi$ - ϵ requires formulas of the form $\phi \wedge \phi \wedge \ldots$

- $\gamma(\phi)$ - notice here that γ requires multiple formulas, bounded together with $\wedge$. This is the same behavior as ϵ. Also note carefully that within the scope of ϵ and γ, ϕ does not quite have the same meaning as in traditional Propositional Logic. For instance, $\gamma(\phi \wedge \ \neg\phi)$ will evaluate to True, because there is one true formula and one false one, so indeed at least half . But you could also simply evaluate $\phi \wedge \neg \wedge \phi$ to be false, first, and then substitute $\gamma(\neg\phi)$ in and return False. However, in both γ and ϵ, the $\wedge$ connector is not evaluated, and all formulas are treated as distinct, unevaluated entities until they leave the scope of either ϵ or γ. For instance, within the context of γ (inside of its parenthesis), the truth table for $p \wedge q$ does not apply, and $\phi \wedge \phi$ does not evaluate to ϕ, nor does $\phi \wedge \neg\phi$ evaluate to $\neg\phi$. This is extremely important for maintaining the structure of well-formed formulas in my new logic syntax!

1.2 Examples of Well-Formed Formulas and Natural Language Translations

- $C\chi A$ (C is **Yu Chen gets a 90 on his final exam** and A is **Yu Chen does not submit his exam**: *Either Yu Chen gets a 90 on his final exam or Yu Chen does not submit his final exam.*
- $B(1,9) = \gamma(I\wedge, J \wedge K)$ where I is **Ivan is ready for the test**, J is **John is ready for the test**, and K is **Kevin is ready for the test**: **There is an estimated 10 percent chance that at least two out of Kevin, Ivan, and John are ready for the exam.** Note here that$B(1,9)$ still implies a fair bit of uncertainty, but $B(1000, 9000)$ would imply almost near certainty that the probability is around 10 percent.
- $B(1,1) = (\gamma(A, D, F) \wedge \epsilon(A, D, F) \rightarrow G)$ (A is **Aaron is happy**, D is **Derek is happy**, and F is **Fred is happy**, and G is **my grandmother will bake cookies.**: **If two, and only two, out of Aaron, Derek, and Fred are happy, then there is a 50 percent change grandmother will bake cookies.**

1.3 When To Use

You would use this new type of logic when needing to introduce uncertainty around events. For instance, you aren't fully sure if something is going to happen, but from empirical observation (and collected data), you can gather a distribution for likelihoods conditioning on past events. In the last example above, maybe you have observed grandmother's behavior many times when two of the three individuals (Aaron, Derek, and Fred) are happy. And you have noticed that half the time she will bake cookies. This will be a valid model to express these empirical observations.

You may also use this type of logic when you only need a certain threshold of truthfulness (for instance, you don't need everything to be true from a set of formulas- you just need a certain number of formulas to be true. For instance, you might use this to represent the terminal conditions for a game, where you need only two out of three game conditions to be true to be the champion.

2 Alternative Logic: Markov Logic Networks

.

Note: *I find the concepts below extremely interesting- there's always been a divide between the world of machine learning and the world of traditional computer science. The model described below is one of the best examples I have seen where cross-disciplinary ideas are merged together in a fruitful way.* The link to the original paper is provided here: Markov Logic Networks by Pedro Domingos and Matthew Richardson.

2.1 The Need

In the types of logic that we have studied within class (propositional and predicate logic), the state expression is a binary true or false variable. Even in more complex temporal logic, such as LTL and CTL, where the state of certain nodes is dependent upon an added dimension of time, a certain node can only ultimately feature a state P, or its negation, $\neg P$. As Pedro Domingo and Matthew Richardson states in *Markov Logic Models*,

> A first-order KB can be seen as a set of **hard constraints** on the set of possible worlds: **if a world violates even one formula, it has zero probability**.

As a data scientist, I frequently need to model both a temporal dimension, as well as the likelihood of certain events happening. Thus, while we explored LTL and CTL, I immediately recognized strong parallels to probabilistic graphical modelling as a mechanism for describing the relationship of different phenomenon in our world. In a very brief sense, there is a need for softening the restrictions of first-order predicate logic so that event probabilities are not discretized as all or none:

The basic idea in MLNs is to soften these constraints: **when a world violates one formula in the KB it is less probable, but not impossible**. The fewer formulas a world violates, the more probable it is. Each formula has an associated weight that reflects how strong a constraint it is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal.

2.2 Overview

The MLN is constructed using a set of pairs (F_i, w_i) where F_i is a first-order logic formula, and w_i is any real number. The model also contains a set of constants $C = \{c_1, c_2, ..., c_C\}$. Together, they form a certain Markov network $M_{L,C}$ where:

- For each predicate that appears within the set $M_{L,C}$, there is a binary node with a value of either 1 for true and 0 for false
- For each formula F_i, there is a "feature" that is of value 1 if the ground formula is true and 0 if it is false.

You can think of the C constant set as the input parameters of the model, while the overall behavior is encoded in its binary nodes, features, and (F_i, w_i) sets:

> Given different sets of constants, it will produce different networks, and these may be of widely varying size, but all will have certain regularities in structure and parameters, given by the MLN (e.g., all groundings of the same formula will have the same weight).

So far, everything is still being expressed in terms of binaries (0 and 1). The probabilistic element of MLNs comes into play when a log-linear probability distribution is defined for the Markov network. A Markov network is an **undirected graphical model expressing relationships defined by a joint distribution of a set of variables** $X = x_1, x_2, x_3, ..., x_n \in \chi$ **along with a set of potential functions** ϕ_k. Potential functions output non-negative, real values for each clique. But we often map them into log-linear functions to make the easier to work with ($\prod_k \phi_k(x_{\{k\}}) \rightarrow exp(\sum_j w_k f_k(x))$. This leads us to base equation we use use to make inferences from using Markov Logic:

$$P(X = x) = \frac{1}{Z}(\sum_i w_i f_i(x)) \tag{1}$$

The Z is a **partition function**, which behaves similar to the **normalization constant** for probability identities like Bayes Rule to ensure that we get a well-formed probability distribution where $0 \leq P(X = x) \leq 1$ and $\sum_x^X P(x) = 1$. This leads us to the base formula that connects traditional first-order logic with the world of machine learning and probabilistic modeling:

$$P(X=x) = \frac{1}{Z}\exp\left(\sum_i w_i n_i(x)\right) = \frac{1}{Z}\prod_i \phi_i(x_{\{i\}})^{n_i(x)} \tag{3}$$

Here, $n_i(x)$ is the number of true groundings for F_i, x_i is a vector of states (true or false) of the atoms in F_i.

2.3 Core Similarities Between MLN and Previously Studied Logics

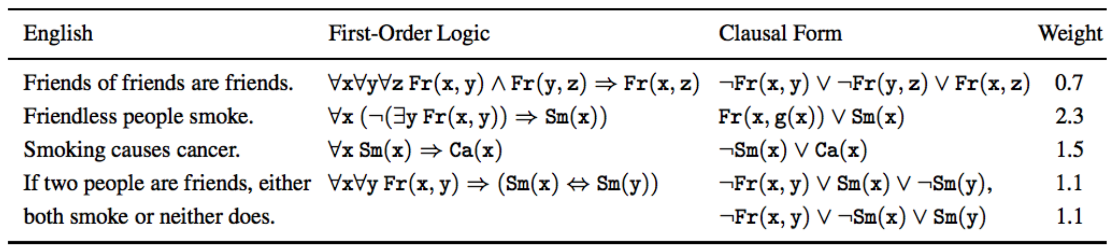
- Both MLN and First-Order Logics share the **same basic syntax (well-formed formulas), a knowledge base, and clauses**.
- **Free or unquantified variables** are treated as being **"quantified" at the top level of the formula**. For instance, at the top level of a MLN, we evaluate the logic as $P(X = x)$, where the free variable X is quantified as x.
- Although the syntax to represent the semantic is different, both MLN and Predicate Logic account for quantifiers ($\forall_x$, although in many cases within probabilistic modelling this is expressed using $\prod_x^X$ or $\sum_x^X$ notations.
- Despite the fact that Markov Logic Networks advertise itself as being non-binary, more probabilistic models of real-world phenomenon, much of its internal machinery still rests upon a root state evaluating to either true or false. For instance, the n in the MLN probability formula described above rests upon evaluation of N atoms for their "truthness".
- Both MLNs and the logics that we have studied in this course provide for the injection of "environment" variables into the world.

2.4 Core Differences

The main conceptual ways that MLNs are different are summarized and proved below: MLNs represent a distribution of likelihoods, whereas all of the logics that we have studied in this course evaluate ultimately to a true or false state.

Let's pretend that we have set of first-order logic clauses, and assigned weights:

Table I. Example of a first-order knowledge base and MLN. `Fr()` is short for `Friends()`, `Sm()` for `Smokes()`, and `Ca()` for `Cancer()`.



English	First-Order Logic	Clausal Form	Weight
Friends of friends are friends.	$\forall x \forall y \forall z\, \mathtt{Fr}(x,y) \wedge \mathtt{Fr}(y,z) \Rightarrow \mathtt{Fr}(x,z)$	$\neg\mathtt{Fr}(x,y) \vee \neg\mathtt{Fr}(y,z) \vee \mathtt{Fr}(x,z)$	0.7
Friendless people smoke.	$\forall x\, (\neg(\exists y\, \mathtt{Fr}(x,y)) \Rightarrow \mathtt{Sm}(x))$	$\mathtt{Fr}(x,g(x)) \vee \mathtt{Sm}(x)$	2.3
Smoking causes cancer.	$\forall x\, \mathtt{Sm}(x) \Rightarrow \mathtt{Ca}(x)$	$\neg\mathtt{Sm}(x) \vee \mathtt{Ca}(x)$	1.5
If two people are friends, either	$\forall x \forall y\, \mathtt{Fr}(x,y) \Rightarrow (\mathtt{Sm}(x) \Leftrightarrow \mathtt{Sm}(y))$	$\neg\mathtt{Fr}(x,y) \vee \mathtt{Sm}(x) \vee \neg\mathtt{Sm}(y)$,	1.1
both smoke or neither does.		$\neg\mathtt{Fr}(x,y) \vee \neg\mathtt{Sm}(x) \vee \mathtt{Sm}(y)$	1.1

Within traditional first-order logic, the likelihood of friendless people not smoking is 0. However, within any MLN, this statement is refined: **the probability of 5 friendless people not smoking is $e^{(2.3*5)}$ less likely than the probability of all friendless people smoking (ground formula)**. Generalizing this, the probability of n friendless people not smoking is $e^{2.3n}$ less likely than the ground truth. You may notice that the MLN is similar in many respects to network graph analysis, and for my purposes can be quite useful for social network modeling.

The authors Richardson and Domingos demonstrate that if there is a zero-arity predicate R_h for each variable X_h in a MLN, where $R_h()$ is true if X_h is true, we can construct formulas that is a conjunction of all the different h literals. The weight of each formula is $logP(X_1, X_2, ...X_h)$ (the joint probability of variables). For any given unique state of the MLN, only one particular formula can be true: for instance, if $X = \{X_1 = F, X_2 = F, X_3 = T\}$, then $R = \{\neg R_1(), \neg R_2(), R_3()\}$. In this scheme, the summation of the probabilities of all unique states will equal 1, which is one of the prerequisites for a well-defined probability distribution. Indeed, traditional first-order logic is a special case of the MLN where the weights for equal for all states and tend towards infinity (limit $w \to \inf$).

For example, let's take a very simple formula $\forall_x R(x) \to S(x)$, with a weight of w and only one constant in the set $C = \{A\}$. The possible permutations, or worlds, available, are $\{R(A), S(A)\}, \{R(A), \neg S(A)\}, \{\neg R(A), S(A)\}, \{\neg R(A), \neg S(A)\}$. Let's walk through each of these worlds and calculate their probabilities:

$\{R(A), \neg S(A)\}$ maps to $T \to F$ in our inference tables, which does not satisfy the the formula. Thus, the number of true groundings $n = 0$:

$$P(X = x) = \frac{1}{Z} exp(\sum_{i=1}^{F} w_i n_i(x)) \tag{2}$$

$$P(X = x) = \frac{1}{Z} exp(0) \tag{3}$$

$$P(X = x) = \frac{1}{Z} \tag{4}$$

The other three worlds are all satisfied, and thus will have a probability of $P(X = x) = \frac{e^w}{Z}$. Since $Z = \sum_{x \in \chi} \prod_k \phi_k(x_k)$ and functions as a normalization constant, it will be equal to $3e^w + 1$ (the 1 comes from the one unsatisfied world, and the $3e^w$ is a summation of the other satisfied worlds).

Notice that here, for the worlds that are satisfied, for any weight vector w that is positive, they will consistently be more likely than the non-satisfied world $\{R(A), \neg S(A)\}$. However, this non-satisfied world still retains a non-zero probability of occurring. It can also be shown that the probability of $S(A)$ being true, given that $R(A)$ is true, is $P(S(A)|R(A)) = \frac{1}{(1+e^{-w})}$. Again, to reinforce the link between traditional first-order logic and probabilistic modelling, if $w \to \inf$, then this probability is 1.

2.5 Usage for Inference

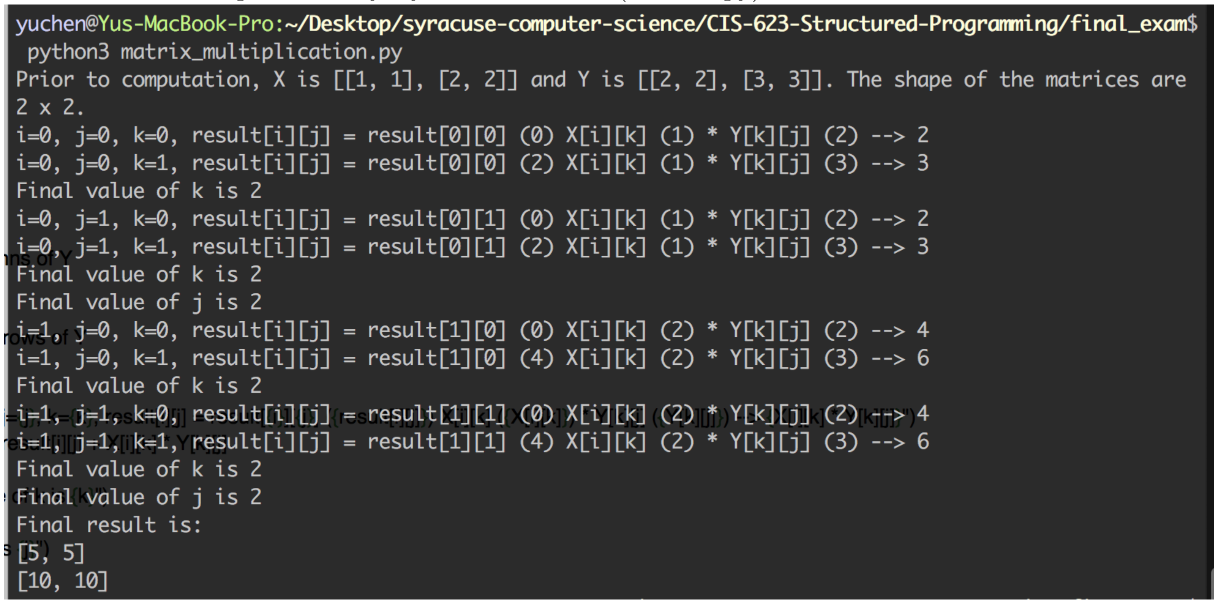
Using MLN, we can answer questions like *what is the probability of formula F holding assuming that formula G holds?*. The core formula used for this is a reformulation of the bedrock of machine learning and probabilistic analysis (Bayes Rule):

$$\begin{aligned} P(F_1|F_2, L, C) &= P(F_1|F_2, M_{L,C}) \\ &= \frac{P(F_1 \wedge F_2|M_{L,C})}{P(F_2|M_{L,C})} \\ &= \frac{\sum_{x \in \mathcal{X}_{F_1} \cap \mathcal{X}_{F_2}} P(X=x|M_{L,C})}{\sum_{x \in \mathcal{X}_{F_2}} P(X=x|M_{L,C})} \end{aligned} \tag{4}$$

In this definition, L is the MLN, C is the set of environment constants, X_{F_i} is the set of all possible words where the formula F_i holds. Ultimately, $P(X = x|M_{L,C})$ is defined by the probability equation (Z) from earlier in this section. In layman's terms, you can read this equation as ***the probability that the state template X assumes a particular configuration x given a Markov model of all functions F, their weights, and a constant set of environment variables C***.

3 Matrix Multiplication

Here is the final output from my Python executable (matmul.py):



```
yuchen@Yus-MacBook-Pro:~/Desktop/syracuse-computer-science/CIS-623-Structured-Programming/final_exam$
 python3 matrix_multiplication.py
Prior to computation, X is [[1, 1], [2, 2]] and Y is [[2, 2], [3, 3]]. The shape of the matrices are
2 x 2.
i=0, j=0, k=0, result[i][j] = result[0][0] (0) X[i][k] (1) * Y[k][j] (2) --> 2
i=0, j=0, k=1, result[i][j] = result[0][0] (2) X[i][k] (1) * Y[k][j] (3) --> 3
Final value of k is 2
i=0, j=1, k=0, result[i][j] = result[0][1] (0) X[i][k] (1) * Y[k][j] (2) --> 2
i=0, j=1, k=1, result[i][j] = result[0][1] (2) X[i][k] (1) * Y[k][j] (3) --> 3
Final value of k is 2
Final value of j is 2
i=1, j=0, k=0, result[i][j] = result[1][0] (0) X[i][k] (2) * Y[k][j] (2) --> 4
i=1, j=0, k=1, result[i][j] = result[1][0] (4) X[i][k] (2) * Y[k][j] (3) --> 6
Final value of k is 2
i=1, j=1, k=0, result[i][j] = result[1][1] (0) X[i][k] (2) * Y[k][j] (2) --> 4
i=1, j=1, k=1, result[i][j] = result[1][1] (4) X[i][k] (2) * Y[k][j] (3) --> 6
Final value of k is 2
Final value of j is 2
Final result is:
[5, 5]
[10, 10]
```

```python
if __name__ == "__main__":

   # 2x2 matrix
   X = [[1,1],
       [2,2]]

   # 2x2 matrix
   Y = [[2,2],
       [3,3]]
   N = 2
   # result is 2x2
   result = [[0,0],
            [0,0]]
```

```
k = 0
j = 0
i = 0

# iterate through rows of X
while i < N:
   # iterate through columns of Y
   j = 0
   while j < N:
      # iterate through rows of Y
      k = 0
      while k < N:
         print(f"i={i}, j={j}, k={k}, result[i][j] = result[{i}][{j}] ({result[i][j]}) X[i][k] ({X[i][k]}) *
             Y[k][j] ({Y[k][j]}) --> {X[i][k] * Y[k][j]}")
         result[i][j] = result[i][j] + X[i][k] * Y[k][j]
         k = k + 1
      print(f"Final value of k is {k}")
      j = j + 1
   print(f"Final value of j is {j}")
   i = i + 1
for r in result:
   print(r)
```

A note before I begin my proof: $\forall_{0\leq i,j<N}$ is shorthand for $0 < i \wedge i \leq N \wedge 0 < j \wedge j \leq N$. These quantifiers are needed because values for i and j outside of these values will yield index errors (for example, it makes no sense accessing $x[4][-2]$).

4 Entire Proof of Total Correctness

$(PRECONDITION) x = [[1,1],[2,2]]; y = [[2,2],[3,3]];$

implies $\rightarrow \forall_{\mathbf{0 \le K < 2}}(\mathbf{0 = 0}) \land \mathbf{0 \le 2}$

$R = [[0,0],[0,0]]$

$\forall_{\mathbf{0 \le K < 2}}(\mathbf{R[0][0] = 0}) \land \mathbf{0 \le 2}$

$N = 2$

summation identity simplifies to $\forall_{\mathbf{0 \le K < N}}(\mathbf{R[0][0] = 0}) \land \mathbf{0 \le N}$

$$\forall_{\mathbf{0 \le K < N}}(\mathbf{R[0][0]} = \sum_{\mathbf{K}}^{\mathbf{0-1}} \mathbf{x[0][K] * y[K][0]}) \land \mathbf{0 \le N}$$

$k = 0$

$$\forall_{\mathbf{0 \le K < N}}(\mathbf{R[0][0]} = \sum_{\mathbf{K}}^{\mathbf{0-1}} \mathbf{x[0][K] * y[K][0]}) \land \mathbf{0 \le N}$$

$j = 0$

$$\forall_{\mathbf{0 \le 0, j, K < N}}(\mathbf{R[0][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[0][K] * y[K][j]}) \land \mathbf{0 \le N}$$

$i = 0$

(inv., var. of outer loop) $\forall_{\mathbf{0 \le i, j, K < N}}(\mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j]}) \land \mathbf{0 \le N - i}$

while(i < N)

(inv., var., and guard of outer loop) $\forall_{\mathbf{0 \le i, j, K < N}}(\mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j]}) \land \mathbf{i \ne N} \land \mathbf{0 \le N - i = E_0}$

implied postcondition) $\forall_{\mathbf{0 \le i, K < N}} \mathbf{R[i][0]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][0]} \land \mathbf{0 \le N}$

$j = 0$

(inv., var. of middle loop) $\forall_{\mathbf{0 \le i, j, K < N}} \mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j]} \land \mathbf{0 \le N - j}$

while (j < N)

(inv., var., and guard of middle loop) $\mathbf{j \ne N} \land \forall_{\mathbf{0 \le i, j, K < N}} \mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{N-1}} \mathbf{x[i][k] * y[k][j]} \land \mathbf{0 \le N - j = E_1}$

implies $\rightarrow K \le k = 0 \land \mathbf{R[i][j] = 0} \land \mathbf{0 \le N}$

$$\mathbf{K \le 0} \land \mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{0-1}} \mathbf{x[i][K] * y[K][j]} \land \mathbf{0 \le N} \land \mathbf{0 \le 0}$$

$k = 0$

(invariant and variant) $\mathbf{K \le k} \land \mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j]} \land \mathbf{0 \le N - k} \land \mathbf{0 \le k}$

while(k < N){

(inv., guard, variant) $\mathbf{K \le k} \land \mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j]} \land \mathbf{k \ne N} \land \mathbf{0 \le N - k = E_2} \land \mathbf{0 \le k}$

$$\text{(implied)}\forall_{\mathbf{0\le i,j<N,K<k+1}}\mathbf{R[i][j] + X[i][k] * Y[k][j]} = \sum_{\mathbf{K}}^{\mathbf{k}} \mathbf{x[i][K] * y[K][j]} \land \mathbf{0 \le N - k - 1 < E_2}$$

R[i][j] = R[i][j] + X[i][k] * Y[k][j]

$$\forall_{\mathbf{0\le i,j<N,K<k+1}}\mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k}} \mathbf{x[i][K] * y[K][j]} \land \mathbf{0 \le N - k - 1 < E_2}$$

$k = k + 1$

$$\text{(invariant and variant)}\forall_{\mathbf{0\le i<N,j<N,K<k}}\mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j]} \land \mathbf{0 \le N - k < E_2}$$

} - close inner loop

$$\text{(inv., guard negation)}\forall_{\mathbf{0\le i<N,j<N,K<k}}\mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j]} \land \mathbf{k = N}$$

$$\text{(implies)}\forall_{\mathbf{0\le i<N,j+1<N,K<k}}(\mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j+1]}) \land \mathbf{k = N} \land \mathbf{0 \le N - j - 1 < E_1}$$

j = j + 1

$$\text{invariant and variant}\forall_{\mathbf{0\le i<N,j<N,K<k}}(\mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j]}) \land \mathbf{k = N} \land \mathbf{0 \le N - j < E_1}$$

} - close middle loop

$$\text{(inv., guard negation middle loop)}\forall_{\mathbf{0\le i<N,j<N,K<k}}(\mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j]}) \land \mathbf{k = N} \land \mathbf{j = N}$$

$$\text{(impl. postcondition)}\forall_{\mathbf{0\le i+1,j<N,K<k}}(\mathbf{R[a][b]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[a][K] * y[K][b]}) \land \mathbf{j = N} \land \mathbf{k = N} \land \mathbf{0 \le N - i - 1 < E_0}$$

i = i + 1

$$\text{(invariant and variant)}\forall_{\mathbf{0\le i,j<N,K<k}}(\mathbf{R[a][b]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[a][K] * y[K][b]}) \land \mathbf{j = N} \land \mathbf{k = N} \land \mathbf{0 \le N - i < E_0}$$

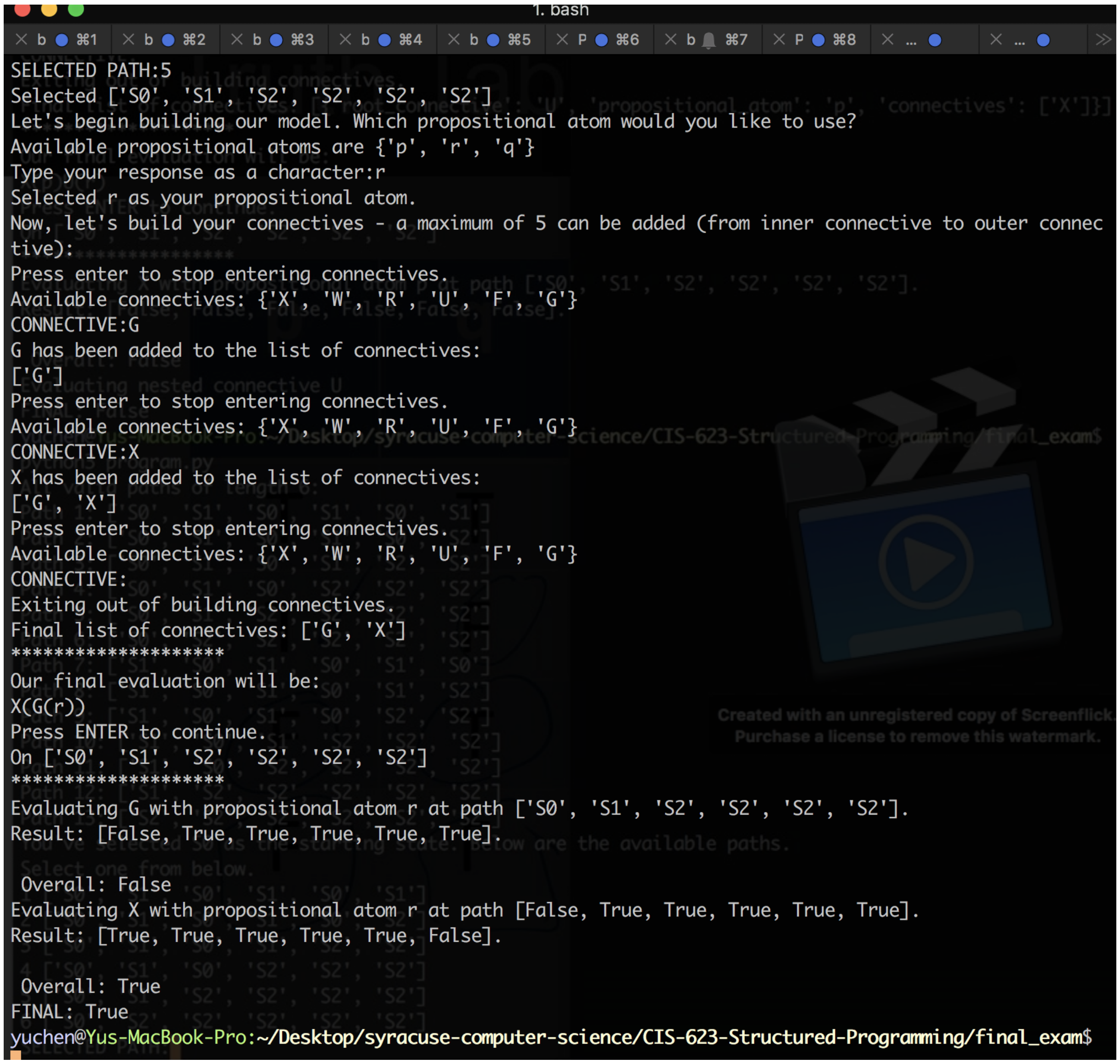
} - close outer loop

$$\text{(inv., guard negation)}\forall_{\mathbf{0\le i,j<N,K<k}}(\mathbf{R[i][j]} = \sum_{\mathbf{K}}^{\mathbf{k-1}} \mathbf{x[i][K] * y[K][j]}) \land \mathbf{j = N} \land \mathbf{k = N} \land \mathbf{i = N}$$

$$\text{implied POSTCONDITION}\forall_{\mathbf{0\le a<i,b<j,K<k}}(\mathbf{R[a][b]} = \sum_{\mathbf{k}}^{\mathbf{N-1}} \mathbf{x[a][k] * y[k][b]})$$

5 LTL and CTL Program

The relevant source code file to examine is **program.py**. The program dynamically evaluates the formulas provided, accepting user input. It relies upon a predefined world (the same as in the lecture notes). I have prepared a Youtube video demonstrating of the program here: https://youtu.be/cRyxGBGRuSY. A screenshot of a sample output is below (XGr, with path π from the lecture notes!):



```
SELECTED PATH:5
Selected ['S0', 'S1', 'S2', 'S2', 'S2', 'S2']
Let's begin building our model. Which propositional atom would you like to use?
Available propositional atoms are {'p', 'r', 'q'}
Type your response as a character:r
Selected r as your propositional atom.
Now, let's build your connectives - a maximum of 5 can be added (from inner connective to outer connec
tive):
Press enter to stop entering connectives.
Available connectives: {'X', 'W', 'R', 'U', 'F', 'G'}
CONNECTIVE:G
G has been added to the list of connectives:
['G']
Press enter to stop entering connectives.
Available connectives: {'X', 'W', 'R', 'U', 'F', 'G'}
CONNECTIVE:X
X has been added to the list of connectives:
['G', 'X']
Press enter to stop entering connectives.
Available connectives: {'X', 'W', 'R', 'U', 'F', 'G'}
CONNECTIVE:
Exiting out of building connectives.
Final list of connectives: ['G', 'X']
********************
Our final evaluation will be:
X(G(r))
Press ENTER to continue.
On ['S0', 'S1', 'S2', 'S2', 'S2', 'S2']
********************
Evaluating G with propositional atom r at path ['S0', 'S1', 'S2', 'S2', 'S2', 'S2'].
Result: [False, True, True, True, True, True].

 Overall: False
Evaluating X with propositional atom r at path [False, True, True, True, True, True].
Result: [True, True, True, True, True, False].

 Overall: True
FINAL: True
yuchen@Yus-MacBook-Pro:~/Desktop/syracuse-computer-science/CIS-623-Structured-Programming/final_exam$
```

6 Natural Deduction

The sequent $\Box(p \wedge q) \rightarrow (\Box p \wedge \Box q)$ can be proved quite easily via one of the core equivalences defined in lecture 10. One of the distributive laws states

$$\Box(p \wedge q) \equiv \Box(p) \wedge \Box(q)$$

Alternatively, you can prove this without any of the distributive laws for modal logic:

$$\boxed{\begin{array}{r} \Box(p \wedge q) \text{ assumption (1)} \\ \Box(p) \text{and elimination, 1 (2)} \\ \Box(q) \text{and elimination, 1 (3)} \\ \Box(p) \wedge \Box(q) \text{and introduction, 2-3 (4)} \end{array}} \qquad \begin{array}{r} (5) \\ (6) \\ (7) \\ (8) \end{array}$$

$$\Box(p \wedge q) \rightarrow (\Box p \wedge \Box q) \rightarrow \text{ introduction, 1-4)}$$