MSO and Probability

Shuchita Kapoor

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Intelligent Systems)

Supervisor: Tim Fernando

September 2020

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Shuchita Kapoor

September 6, 2020

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Shuchita Kapoor

September 6, 2020

Acknowledgments

I would like to express my gratitude to my supervisor, Mr Tim Fernando, for guiding and supporting me throughout this research. Even in this dire situation, he helped me to overcome the obstacles that were faced during this work. I am thankful to him for always inspiring me and steering me in the right direction with immense knowledge in this field.

Shuchita Kapoor

University of Dublin, Trinity College
September 2020

MSO and Probability

Shuchita Kapoor, Master of Science in Computer Science

University of Dublin, Trinity College, 2020

Supervisor: Tim Fernando

Representing time has always been challenging in the domain of artificial intelligence. There are have been researchers working in natural language processing domain who have found ways to extract this information. This paper particularly considers the work presented by James Allen and the 13 temporal relations that he coined in his research. Further, it has been studied that representing these relations in string format of MSO (Monadic Second Order) logic brings out various characteristics of this kind of knowledge. This helps in processing the temporal information and finding new results in this field of research. In a research conducted by B*ü*chi, Elgot, Trakhtenbrot, it was proved that MSO logic is related to finite state methods. Thus, in this paper, we consider two case studies - Zebra Puzzle and Allen Relations, whose constraints are converted to MSO logic and further solved using superposition theorem. To understand how probabilities can be involved in these constraints, we consider the research done by James Allen for the Allen relations and extend it to calculate the conditional probability for the relations. Also, these Allen relations are equiprobable. Thus, to obtain a different probability for each relation, we assign weights to them and apply the equation given in Markov Logic Network. Finally, to get a uniform probability, appropriate weights are allotted to the relations.

Summary

Many researchers have worked on extracting the temporal information of the real world. In this paper, we consider the researches done by James Allen. This paper mainly revolves around the 13 temporal relations, which was given by him. To obtain more characteristics of these relations, it is represented in a string format which is further described in MSO (Monadic Second Order) logic. There are two formats which are considered - Border and Interior (Stative) given by David Dowty in his paper. Also, according to B*ü*chi, Elgot, Trakhtenbrot, it was observed that language defined using MSO logic is equivalent to regular languages, which can be expressed using finite automaton. Thus, this proved that MSO logic is related to finite state methods. This paper focuses on researching about MSO logic by using two case studies - Zebra Puzzle and Allen Relations. In research conducted by Tim Fernando, it was found that superposition theorem is equivalent to MSO conjunction. Thus, this theorem has been studied deeply by applying it to find the solution for Zebra Puzzle and obtain transitivity table for Allen relations. The Allen transitivity grid has been further used in calculating probabilities for these relations.

This paper concentrates on Allen Relations to understand how probabilities can be calculated for a constraint. To formulate equations for the Allen relations, they are categorised as - Long, Medium and Short relations by considering the border format of the strings. James Allen was able to calculate these probabilities according to different

categories. It even states that long relations are more probable than short and medium relations in an input string. This paper extends his theory and calculates conditional probabilities for the relations using the Allen transitivity table. Also, weights are applied to the relations and calculations are done using formulas provided in Markov Logic Network. Due to this, each relation gets a unique probability according to its weight. To get a uniform probability, appropriate weights are assigned to the relations according to the length of the input string. Hence, for long relations, the weight decreases as the input length increase, and it is vice versa in case of short and medium relations.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

James Allen, in his paper [1], states that time can be represented as an interval or a point in the real world. In artificial intelligence (AI), interpreting this temporal information is the most challenging and concerning part. Natural language processing is a sub-domain of AI which deals with extracting this kind of information into models. Thus, James Allen describes 13 temporal relations with which such knowledge can be captured easily. These relations have been widely used by many researchers in their papers (eg, [4], [5], [6] and many more). Even in this paper, we focus on these relations to understand how probabilities can be calculated for the constraints.

To further study and use temporal relations, they are converted into strings. These strings are converted to MSO (Monadic Second Order) Logic to understand their properties. The strings are described as two formats - Border and Interior (Also known as stative) - by David Dowty in his paper [7]. The border format is mostly used to categorise the Allen relations, which are used in the probability calculations. Converting the constraints of any Constraint Satisfaction Problem (such as Zebra Puzzle) into border format can be cumbersome. Thus, for such problems, the interior format of the strings are used. This paper mainly focusses on the border format for its computations.

1.1 Objective

This paper focuses on two main objectives or research questions -

- To understand finite state methods for reasoning the constraints.
- To learn how to include probabilities in the constraints.

Regular languages can be expressed using finite state methods. According to Büchi, Elgot, Trakhtenbrot theorem stated in [8], [9] and [10], it can be inferred that MSO languages are equivalent to regular languages. Thus, it can be seen that by learning how MSO applies to the constraints, we can achieve our first objective. Therefore, this paper directs on the research on MSO (Monadic Second Order) logic. For accomplishing this, two case studies are considered - Zebra Puzzle and Allen Relations. From the paper [11], superposition theorem is used to reason over the constraints which are defined using MSO formulas. To understand the working of this theorem, it was applied to solve the Zebra puzzle and to generate Allen Transitivity Grid using the Allen Relations.

To attain the second objective, Allen relations were used as a constraint. The Allen Transitivity Grid was considered to understand how probabilities can be calculated for these constraints. Lastly, Markov Logic Network was studied in-depth to find the probabilities for the constraints when weights are applied to them. Using these formulas, a uniform probability was calculated for the Allen relations by assigning appropriate weights to the relations.

1.2 Motivation

We came across the concept of MSO in the paper [11], where superposition theorem was used to reason over the terms formulated using MSO logic. The motive of this paper is to apply this theorem to solve the Zebra puzzle and find Allen transitivity grid for Allen relations and understand the notion of MSO. James Allen, in his paper [1], calculated probabilities for the Allen relations using the transitivity table. This paper

extends this theory and applies weights to find probabilities using Markov Logic. The aim of this paper is also to find appropriate weights for the Allen relations to get equal probabilities for all the relations.

1.3 Reader's Guide

The structure of the paper is as follows - Chapter 2 discusses the concepts of First Order Logic, MSO (Monadic Second Order Logic), Superposition theorem and MONA tool. This chapter also examines the Zebra puzzle, Allen Relations and Allen Transitivity Grid. It even presents prior work on finding probabilities for Allen relations. It introduces the concept of Markov logic. Chapter 3 gives the implementation methods used to solve the Zebra puzzle and find Allen transitivity grid for Allen relations using the superposition theorem. It also discusses the methodology to find probabilities for Allen relations with and without Markov logic. Chapter 4 concludes the paper by presenting the results and further work.

Chapter 2

Prior Work

2.1 First Order Logic

The first order logic quantifies over individual variables, constants, functions and predicates. Constants denote objects in the domain, such as people like Bob, Claire, etc. Variables range over such objects, for example, a variable can be used to describe the houses in zebra puzzle where, at a given time, it can be any of the houses from the five houses. Functions are mappings from a group of objects to objects, for example, MotherOf function gives a mapping between two objects. Predicates represent the relation between objects, for example, Friends states that two objects are friends [12].

If we consider a constraint which states that Englishman lives in the red house, we can represent it as follows -

$$\exists x(english(x) \wedge red(x)) \tag{2.1}$$

Here x is an individual variable which denotes a house and english(x) and red(x) are functions, which maps x to one of the five nationalities and five colours of the house respectively.

2.2 MSO (Monadic Second Order Logic)

The Monadic Second Order logic is an extension of First Order Logic because it quantifies over a set of variables, which are known as Monadic Predicates. Thus, if x_1, x_2 denote positions of the houses then set variables X denotes set of these positions [13]. Consider a constraint stating that five houses should always be beside each other. We can represent this as follows -

$$\exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 (\bigwedge_{i=1}^{4} x_i S x_{i+1} \wedge \forall y \bigwedge_{i=1}^{5} y = x_i) \tag{2.2}$$

Here xSy means x is to the left of y.

As shown in equation 2.2, x_1, x_2, x_3, x_4, x_5 denote the positions of the five houses which is quantified using existential quantifier. Since these set of variables are bound to the quantifier, it is also known as a sentence. If a language consists of such kind of sentences, the language is called as MSO-definable. According to B*ü*chi, Elgot, Trakhtenbrot theorem stated in [8], [9] and [10], this kind of language is also said to be a regular language. Since regular languages are represented using finite state methods, this theorem proves that MSO is related to it.

2.3 MONA

The MONA tool is an MSO project which was developed by Nils Klarlund and Anders Møller in Denmark [3]. This tool considers WS1S (Weak monadic Second-order theory of 1 Successor) and WS2S (Weak monadic Second-order theory of 2 Successor) formulas and converts them into finite state automata. These formulas quantify over finite subsets of variables. The WS1S reasons over properties of finite strings [14] whereas WS2S is a generalization of WS1S since it considers finite binary trees. These formulas are used to represent regular languages and thus is used in the tool. This tool converts the formulas into minimum DFAs (Deterministic Finite Automata) and GTAs (Guided Tree Automata) [3]. MONA tool is very efficient and fast in converting the formulas.

```
var2 P,Q;
P\Q = {0,4} union {1,2}; # the formula from Section 1

var1 x;
var0 A;

ex2 Q: x in Q
  & (all1 q:
      (0 < q & q <= x) =>
          (q in Q => q - 1 notin Q)
        & (q notin Q => q - 1 in Q))
  & 0 in Q;

A & x notin P;
```

Figure 2.1: even.mona [3]

A program written in MONA consists of any number of equations and definitions. The lines are terminated with a semicolon. Variables of first order and second order logic can also be declared. For example, in the figure 2.1, we can see that two second order variables are declared - P,Q - which are defined using set of positions. Thus, from the second line, P is assinged to {1,2,4} and Q is assigned to {5}. It also consists of first order variable x (which is a natural number) and a zeroth order variable A which is equivalent to a boolean variable. The existential quantifiable formula checks if x is even or not. The Q in this formula is a local variable which is only valid till the end of the formula and the Q denoted as second order variable is global variable which is valid till the end of the program.

When we run this program, a minimum deterministic automaton is obtained which satisfies all the formulas in the program. It also gives a counter example of a string with minimum length which rejects the formulas. As seen in the figure 2.2, the minimum length for counter example is 2 (including the first position). Similarly, a satisfying example of minimum length can be captured which accepts all the formulas in the program. In the figure 2.2, the minimum length for satisfying example is 8.

```
A counter-example of least length (1) is:
P               X X
Q               X X
x               X 1
A               0 X

P = {}
Q = {}
x = 0
A = false

A satisfying example of least length (7) is:
P               X 1110100
Q               X 000X0XX
x               X 0000001
A               1 XXXXXXX

P = {0,1,2,4}
Q = {}
x = 6
A = true
```

Figure 2.2: Output for even.mona [3]

2.4 Zebra Puzzle

The Zebra Puzzle is one of the famous classical logic problems. Usually, this problem comes in the family of Constraint Satisfaction Problems (CSP). Since Albert Einstein invented this riddle as a boy, it is also called Einstein's Riddle[15]. This puzzle is known for its difficulty, and sometimes it is claimed that only 2% of the population can solve this problem. There are many variants of this problem, but this paper deals with the version published in Life International magazine on December 17, 1962[15].

2.4.1 Constraints and Attributes

This puzzle states that there are five houses. Each house has the following attributes -

1. A unique colour (chosen from Red, Green, Blue, Yellow, and Ivory)
2. Inhabited by people of a different nationality (selected from Englishman, Spaniard, Japanese, Ukrainian, and Norwegian

3. Having a distinct pet (chosen from Dog, Horse, Snails, Fox and Zebra)
4. Smoking a different American cigarette (selected from Kools, Old Gold, Parliament, Lucky Strike, and Chesterfield)
5. Having a unique drink (chosen from Water, Tea, Orange Juice, Milk and Coffee).

The goal of this problem is to find in which house the Zebra lives and water is drunk. There are fifteen constraints given to solve this riddle[15] -

1. There are five houses.
2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old Gold smoker owns snails.
8. Kools are smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house.
11. The man who smokes Chesterfields lives in the house next to the man with the fox.
12. Kools are smoked in a house next to the house where the horse is kept.
13. The Lucky Strike smoker drinks orange juice.
14. The Japanese smokes Parliaments.
15. The Norwegian lives next to the blue house.

All the constraints of Zebra Puzzle can be represented using First Order Logic.

2.4.2 Prior Work

Many researchers solved the zebra puzzle using various AI techniques. One of the authors in [16] evaluated methods such as backtracking, backtracking and forward chaining (FC), backtracking and minimum remaining value (MRV), backtracking and minimum conflict, backtracking and minimum conflict and MRV, and new method created by the authors, to find the solution of the Einstein's problem. From the results, they found out that Backtracking and MRV technique is the best and fastest algorithm to solve the zebra puzzle. This technique first satisfies unit constraints and later selects constraints with the least number of nodes.

The authors in [17] used a Prolog approach to solve the zebra puzzle. They used a depth-first searching technique to find an optimized solution for this problem. The methodology was modified using three additional heuristics, which gave a faster and better outcome. These three heuristics are as follows -

- Constraints which resulted in a single branch should be chosen since it states which house contains the attribute.
- Since the solution involves branching, the nodes with zebra and water should be kept at the end.
- The constraints with the same kind of attributes should be grouped.

These papers were considered in selecting the constraints and applying superposition theorem on it to find an optimized solution for the zebra problem.

2.5 Superposition Theorem

Superposition Theorem is equivalent to the MSO Conjunction. As described in [11], its primary aspect is to join two strings using the component-wise union of the two strings. This is denoted by $\&_\circ$. Thus, for two strings - $\alpha 1 \cdots \alpha k$ and $\alpha' 1 \cdots \alpha' k$ of sets with the same length k, the component-wise union is:

$$(\alpha 1 \cdots \alpha k) \;\&_\circ\; (\alpha' 1 \cdots \alpha' k) \;:=\; (\alpha 1 \cup \alpha' 1) \cdots (\alpha k \cup \alpha' k) \tag{2.3}$$

To avoid complications arising from:

1. projections formed due to compressing d$\Box$ (removing empty strings from input).
2. the need to project output with the inputs.

we use inductive rules such as s0 and s1 as follows –

$$\frac{}{\&(\in,\in,\in)}(s0) \qquad\qquad \frac{\&(s,s',s'')}{\&(\alpha s,\alpha' s',(\alpha\cup\alpha')s'')}(s1) \tag{2.4}$$

Here s, s', s'' are the strings equal to $\alpha 1\cdots\alpha k$, $\alpha' 1\cdots\alpha' k$ and $\alpha'' 1\cdots\alpha'' k$ respectively.

Using s0 and s1, we can generate $\&_\circ$ as follows -

$$s \;\&_\circ\; s' \Longleftrightarrow \&(s,s',s'') \tag{2.5}$$

The component-wise intersection given in s1 superposes in lockstep. To solve this, using the interleaving/shuffling principle, two rules (d1 and d2) are generated –

$$\frac{\&(s,s',s'')}{\&(\alpha s,s',\alpha s'')}(d1) \qquad\qquad \frac{\&(s,s',s'')}{\&(s,\alpha' s',\alpha' s'')}(d2) \tag{2.6}$$

This paper focuses on using the Preposition 4 stated in [11] for solving Zebra puzzle and defining Allen Interval Relations using superposition.

The equations from this preposition, which we use in this research are -

$$\frac{\&(s,s',s'') \quad \alpha\cap\sum{}'\subseteq\alpha' \quad \alpha'\cap\sum\subseteq\alpha}{\&(\alpha s,\alpha' s',(\alpha\cup\alpha')s'')}(s1)_{\sum\sum'} \tag{2.7}$$

$$\frac{\&(s,s',s'') \quad \alpha\cap\sum{}'=\emptyset}{\&(\alpha s,s',\alpha s'')}(d1_{\sum'}) \qquad\qquad \frac{\&(s,s',s'') \quad \alpha'\cap\sum=\emptyset}{\&(s,\alpha' s',\alpha' s'')}(d2_{\sum}) \tag{2.8}$$

where $\sum$ is a finite set of strings.

2.6 Allen Interval Relations

James Allen, in his paper[1], has indicated that we can represent time as points as well as intervals. Considering below two sentences -
We found the letter at twelve noon.
We found the letter yesterday.
The 'twelve noon' in the first sentence states a precise point of time and 'yesterday' in the second sentence state an interval of time when the event of finding letter occurred. These sentences are an example of temporal precision. Many times, sentences do not consist of temporal precision. Usually, English sentences use temporal references in the form of tenses to relate two events in the sentences. For example, in the below two sentences -
We found the letter while John was away.
We found the letter after we made the decision.
The words 'while' and 'after' give a temporal reference by connecting two events.

Furthermore, He states that by using a temporal interval instead of a temporal point, it is easier to avoid irrelevant facts. Thus, 13 relations (shown in Table 2.1) were introduced to relate two time intervals. These relations are also known as Allen Interval Relation.

2.6.1 Allen Transitivity Grid

Usually, a network represents these relations, where the node represents individual intervals. The arc between the nodes depicts the relation between the two intervals. When a new interval relation is introduced in the network, transitivity rules are applied to compute other consequences between these intervals, which have been affected due to the new fact. For example, if a new relation (i is during j) is added to constraints such as j is before k, then it can be concluded that i must be before k [1]. Such type of transitivity relations can be seen in Table 2.3. Since eq relation gives the same relation as applied with, it has not been included in the table 2.3. Here (d,s,f) are also known as dur relation and (di,si,fi) are also known as con relation.

Relation	**Symbol**	**Symbol For Inverse**	**Pictorial Example**
X *before* Y	$<$	$>$	XXX YYY
X *equal* Y	$=$	$=$	XXX YYY
X *meets* Y	m	mi	XXXYYY
X *overlaps* Y	o	oi	XXX YYY
X *during* Y	d	di	XXX YYYYYY
X *starts* Y	s	si	XXX YYYYYY
X *finishes* Y	f	fi	XXX YYYYYY

Table 2.1: Thirteen Allen Interval Relations [1]

2.7 Temporal Representation of Strings

Using these relations, two intervals a and a' are shown as a string in [11]. The temporal representation for a overlaps a' can be pictured as -

	a	a, a'	a'	

(2.9)

This string is of length 5 which consists of following attributes -

BOX 1 The string starts with an empty box for times before a.

BOX 2 The second box contains a since the string a has started in that time and a' is yet to start.

BOX 3 Since the two strings overlap each other, so in this box a ends and a' starts.

BOX 4 This box contains a' because in this time a' has ended.

BOX 5 This empty box represents times after a'.

If a third interval a'' overlaps both the previous intervals, then the string in (2.9) has its length increased to 7 with following changes -

- Since a'' overlaps with a and a', it starts before a. Thus the first box is splitted as $\boxed{\ }$ and a''. Similarly, the third box is splitted as $\boxed{a, a', a''}$ and $\boxed{a, a'}$.
- The second box gets an additional a'' because it has begun before a.

This string is represented as follows -

$$\boxed{\ }\boxed{a''}\boxed{a, a''}\boxed{a, a', a''}\boxed{a, a'}\boxed{a'}\boxed{\ } \tag{2.10}$$

2.7.1 Interior and Border Representation of Strings

Strings defined in these theorems can be represented in interior format as well as border format [11] [7]. Interior format is pretty straight forward in which strings are appended with empty boxes to consider other times before and after these strings. In this paper, this format is used to denote constraints of the Zebra Puzzle. It is denoted as shown in equation 2.9. This equation or interior format of the string can be magnified further to achieve border format of the string which is represented as [1] -

$$\boxed{l(a)}\boxed{a, l(a')}\boxed{a, a', r(a)}\boxed{a', r(a')}\boxed{\ } \tag{2.11}$$

Here $l(a)$ and $l(a')$ depict the left (open) border for a and a' respectively, and $r(a)$ and $r(a')$ depict the right (closed) border for a and a' respectively.

2.7.2 Reduct

Reduct is a function defined to find a string with respect to a particular interval. It uses component-wise intersection which can be stated as -

$$\rho_A(\alpha_1 \cdots \alpha_n) := (\alpha_1 \cap A) \cdots (\alpha_n \cap A) \tag{2.12}$$

where A is a set and string s consists of strings $\alpha_1 \cdots \alpha_n$.

Thus, if this formula is applied on equation 2.10 to get reduct for $\{\alpha, \alpha'\}$, it is depicted as -

| | | a | a, a' | a, a' | a' | | (2.13)

2.7.3 Compression

Compression is used to remove repeated intervals (called as stutters) in a string. This is known as block compression with which we can obtain a stutterless string. It uses the following formula-

$$bc(\alpha_1 \cdots \alpha_n) := \begin{Bmatrix} \alpha_1 \cdots \alpha_n & if\ n{<}2 \\ bc(\alpha_2 \cdots \alpha_n) & else\ if \alpha_1 = \alpha_2 \\ \alpha_1 bc(\alpha_2 \cdots \alpha_n) & otherwise \end{Bmatrix} \tag{2.14}$$

Thus, $\alpha_1 \cdots \alpha_n$ can be called stutterless if $\alpha_i \neq \alpha_{i+1}$.

Compression can also be used to remove the empty boxes in equations such as 2.9, 2.10 and 2.11. This is given as follows -

$$d_\square(\alpha s) := \begin{Bmatrix} d_\square(s) & if\ \alpha = \square \\ \alpha d_\square(s) & otherwise \end{Bmatrix} \tag{2.15}$$

2.7.4 Translation of Border and Interior strings

An interior string can be converted to a border string and vice versa using the preposition 1 and preposition 2 as stated in the paper [11]. Thus,to map interior strings $\alpha_1 \cdots \alpha_n$ to border stings $\beta_1 \cdots \beta_n$, we use following formulas -

$$\beta_i := \{l(a) \mid a \in \alpha_{i+1} - \alpha_i\} \cup \{r(a) \mid a \in \alpha_i - \alpha_{i+1}\} \quad for\ i{<}n$$
$$\beta_n := \{r(a) \mid a \in \alpha_n\}$$

For example, the above equation can be applied on equation 2.9 in following steps -

- Since after the empty box a starts, the first box consists $l(a)$.
- Similarly, the next box after a has a, a' where a' starts and a continues. Hence, the next box consists of $l(a')$.

- The next box does not contain a but a' continues to be present. Thus, the next box consists of $r(a)$.
- The last box is an empty box. Hence, the next box consists of $r(a')$.
- The string ends with an empty box.

Together, this can be denoted as -

$$\boxed{l(a)}\boxed{l(a')}\boxed{r(a)}\boxed{r(a')}\boxed{\ } \tag{2.16}$$

This depicts an open left border and closed right border which can also be described as $(l(a), r(a)]$.

In order to get interior format from the equation 2.16, we use the following steps -

- Since the first box contains $l(a)$, it means a starts in this box.
- Similarly the second box contains $l(a')$ and it does not contain $r(a)$ which means a continues in this box. Hence, the next box consists of a, a'.
- The next box has $r(a)$ which means that a ends here and thus, this box contains only a' since there is no $r(a')$.
- Similarly, the next box consists of $r(a')$ which shows that $r(a')$ ends in this box. Therefore, it is an empty box.

Interior format always starts with an empty box and thus, by adding it in the first position, we obtain equation 2.9 from the above steps.

To obtain correct format of the strings, it is necessary that, for a string interval a, $l(a)$ and $r(a)$ should be present in that order.

2.8 Probability of Allen Relations

Since there are 13 Allen relations, it can be said that the probability for each relation is $\frac{1}{13}$ using the principle of indifference. But from the transitivity table, it is evident that

$(l,r]R(l',r']$	S_R	R^{-1}	$S_{R^{-1}}$
$(l,r]b(l',r']$	$l \mid r \mid l' \mid r'$	bi	$l' \mid r' \mid l \mid r$
$(l,r]d(l',r']$	$l' \mid l \mid r \mid r'$	di	$l \mid l' \mid r' \mid r$
$(l,r]o(l',r']$	$l \mid l' \mid r \mid r'$	oi	$l' \mid l \mid r' \mid r$
$(l,r]m(l',r']$	$l \mid r,l' \mid r'$	mi	$l' \mid r',l \mid r$
$(l,r]s(l',r']$	$l,l' \mid r \mid r'$	si	$l,l' \mid r' \mid r$
$(l,r]f(l',r']$	$l' \mid l \mid r,r'$	fi	$l \mid l' \mid r,r'$
$(l,r]e(l',r']$	$l,l' \mid r,r'$	e	

Table 2.2: Categories of Allen Interval Relations [2]

some relations are more probable. To understand this, Allen relations are segregated into three categories. The border format of the relations is considered to get the length of a string, according to which it is divided as long, medium and short strings as shown in table 2.2. Thus, the three categories can be represented as [2]-

- $\{R \in AR \mid length(S_R) = 4\} = \{b, d, o, bi, di, oi\}$
- $\{R \in AR \mid length(S_R) = 3\} = \{m, s, f, mi, si, f\}$
- $\{R \in AR \mid length(S_R) = 2\} = \{e\}$

Each category has a different probability. To calculate this, an input string of length n with specific Allen relation is considered. As this length increases, the probability of short and medium relations to be considered as specific Allen relation decreases. Since there are 6 long relations, the probability of these relations to be considered in the input string becomes $\frac{1}{6}$.

2.9 Markov Logic Network

Markov logic Network is usually used as a template for creating the Markov networks. A Markov network is also known as Markov random field and is [12]-

- model for the joint distribution of a set of variables (we consider the Allen relations as variables in this paper).
- composed of an undirected graph with nodes representing the Allen interval relations.

- represented using log-linear models to calculate probabilities for the set of variables.

Markov logic network is a set of pairs (F_i, w_i), where F_i is a formula in first order logic and w_i is a real number (known as weights) [12]. The probability distribution over x possible worlds defined by Markov Logic Network can be presented as [12]-

$$P(X = x) = \frac{1}{Z}\ exp\ (\sum_i w_i n_i(x)) \tag{2.17}$$

Here $n_i(x)$ is the number of formulas that satisfy the first order formula in F_i and Z is the partition function which is a summation of all first order formulas.

B r2 C A r1 B	<	>	d	di	o	oi	m	mi	s	si	f	fi
"before" <	<	no info	< o m d s	<	<	< o m d s	<	< o m d s	<	<	< o m d s	<
"after" >	no info	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	>	>
"during" d	<	>	d	no info	< o m d s	> oi mi d f	<	>	d	> oi mi d f	d	< o m d s
"contains" di	< o m di fi	> oi di mi si	o oi dur con =	di	o di fi	oi di si	o di fi	oi di si	di fi o	di	di si oi	di
"overlaps" o	<	> oi mi di si	o d s	< o m di fi	< o m	o oi dur con =	<	oi di si	o	di fi o	d s o	< o m
"over-lapped-by" oi	< o m di fi	>	oi d f	> oi di mi si	o oi dur con =	>oi mi	o di fi	>	oi d f	oi > mi	oi	oi di si
"meets" m	<	> oi mi di si	o d s	<	<	o d s	<	f fi =	m	m	d s o	<
"met-by" mi	< o m di fi	>	oi d f	>	oi d f	>	s si =	>	d f oi	>	mi	mi
"starts" s	<	>	d	< o m di fi	< o m	oi d f	<	mi	s	s si =	d	<m o
"started-by" si	< o m di fi	>	oi d f	di	o di fi	oi	o di fi	mi	s si =	si	oi	di
"finishes" f	<	>	d	> oi mi di si	o d s	> oi mi	m	>	d	> oi mi	f	f fi =
"finished-by" fi	<	> oi mi di si	o d s	di	o	oi di si	m	sioi di	o	di	f fi =	fi

Table 2.3: Transitivity Table for Twelve Relations without (=) [1]

Chapter 3

Design and Implementation

3.1 Superposition Theorem

From the algorithm 1, it is apparent that the function takes in two strings, which are to be merged, and their vocabularies, which is the union of all the attributes in a string, and gives an output string which is the union of the two input strings. Thus, component-wise union is done to obtain the merged string. Using the component-wise intersection between a string and vocabulary of the other string, we check that the relevant constraints are satisfied. It is a recursive algorithm which finds maximum number of possible strings that satisfy the constraints. This theorem is quiet efficient to solve the Zebra Puzzle and find transitivity grid for Allen Relations because it takes less amount of time and space to store intermediate results.

Algorithm 1: superpose(s1, s2, voc1, voc2)

```
if s1 and s2 are empty then
    return [ ];
else
    temp = [ ];
    if length of s1 > 0 then
        h1 = s1[0], t1 = s1[1: ];
        h1v2 = h1 intersection voc2;
        if h1v2 is empty then
            for res in superpose(t1,s2,voc1,voc2) do
                temp.append([h1] + res)
            end
        end
        if length of s2 > 0 then
            h2 = s2[0], t2 = s2[1: ] ;
            h3 = h1 union h2 ;
            if h1v2 is subset of h2 then
                h2v1 = h2 intersection voc1;
                if h2v1 is subset of h1 then
                    for res in superpose(t1,t2,voc1,voc2) do
                        temp.append([h3] + res)
                    end
                end
            end
        end
    end
    if length of s2 > 0 then
        h2 = string2[0];
        t2 = string2[1: ];
        h2v1 = h2 intersection voc1;
        if h2v1 is empty then
            for res in superpose(s1,t2,voc1,voc2) do
                temp.append([h2] + res)
            end
        end
    end
    return temp;
end
```

3.2 Zebra Puzzle

3.2.1 MSO Representation

Before solving the zebra puzzle, the constraints were converted to MSO(Monadic Second Order) formulas, which are defined as follows -

1. There are five houses.
   $\exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 (\bigwedge_{i=1}^{4} x_i S x_{i+1} \wedge \forall y \bigwedge_{i=1}^{5} y = x_i)$

2. The Englishman lives in the red house.
   $\exists x(english(x) \wedge red(x))$

3. The Spaniard owns the dog.
   $\exists x(spaniard(x) \wedge dog(x))$

4. Coffee is drunk in the green house.
   $\exists x(coffee(x) \wedge green(x))$

5. The Ukrainian drinks tea.
   $\exists x(ukrainian(x) \wedge tea(x))$

6. The green house is immediately to the right of the ivory house.
   $\exists x \exists y(xS_y \wedge ivory(x) \wedge green(y))$

7. The Old Gold smoker owns snails.
   $\exists x(oldGold(x) \wedge snails(x))$

8. Kools are smoked in the yellow house.
   $\exists x(kools(x) \wedge yellow(x))$

9. Milk is drunk in the middle house.
   $\exists x(milk(x) \wedge \exists x_1 \exists x_2 \exists y_1 \exists y_2 (x_1 S x_2 \wedge x_2 S x \wedge x S y_1 \wedge y_1 S y_2))$

10. The Norwegian lives in the first house.
    $\exists x(norwegian(x) \wedge \exists x_2 \exists x_3 \exists x_4 \exists x_5 (x S x_1 \wedge x_2 S x_3 \wedge x_3 S y_4 \wedge x_4 S x_5))$

11. The man who smokes Chesterfields lives in the house next to the man with the fox.
    $\exists x(chesterfields(x) \wedge \exists y(fox(y) \wedge (xSy \wedge ySx)))$

12. Kools are smoked in a house next to the house where the horse is kept.
    $\exists x(kools(x) \wedge \exists y(horse(y) \wedge (xSy \wedge ySx)))$

13. The Lucky Strike smoker drinks orange juice.
    $\exists x(luckyStrike(x) \wedge orangeJuice(x))$

14. The Japanese smokes Parliaments.
    $\exists x(japanese(x) \wedge parliaments(x))$

15. The Norwegian lives next to the blue house.
    $\exists x(norwegian(x) \wedge \exists y(blue(y) \wedge (xSy \wedge ySx)))$

Here xSy means x is to the left of y.

3.2.2 Set Representation

Using the MSO strings, the constraints were depicted as a set of five attributes, which are shown as follows -

1. Each house is a set of color, nationality, drink, smokes, and pet. It is represented as -

$$House \begin{bmatrix} c \\ n \\ d \\ s \\ p \end{bmatrix} := \boxed{\text{(color, c), (nation, n), (drink, d), (smoke, s), (pet, p)}}$$

2. Since the position of attributes in this constraint is not fixed, 5 strings can be

created by changing the position of the attributes as follows -

$$pad_5(\begin{bmatrix} red \\ english \\ - \\ - \\ - \end{bmatrix}) = \{\square^k \boxed{\text{(color,red),(nation,english)}} \square^{4-k} \mid 0 \leq k \leq 4\}$$

3. In the similar manner, 5 strings can be created for this constraint by padding empty boxes at different positions -

$$pad_5(\begin{bmatrix} - \\ span \\ - \\ - \\ dog \end{bmatrix})$$

4. Similarly, $pad_5(\begin{bmatrix} green \\ - \\ coffee \\ - \\ - \end{bmatrix})$ 5. $pad_5(\begin{bmatrix} - \\ ukrainian \\ tea \\ - \\ - \end{bmatrix})$

6. $pad_5(\begin{bmatrix} ivory \\ - \\ - \\ - \\ - \end{bmatrix} \begin{bmatrix} green \\ - \\ - \\ - \\ - \end{bmatrix})$ 7. $pad_5(\begin{bmatrix} - \\ - \\ - \\ oldGold \\ snail \end{bmatrix})$ 8. $pad_5(\begin{bmatrix} yellow \\ - \\ - \\ kools \\ - \end{bmatrix})$

9. The attributes in this constraint is given a fixed house number. Thus, they are

represented as follows -

$$pad_5(\begin{bmatrix}-\\-\\-\\-\\-\end{bmatrix}\begin{bmatrix}-\\-\\-\\-\\-\end{bmatrix}\begin{bmatrix}-\\-\\milk\\-\\-\end{bmatrix}\begin{bmatrix}-\\-\\-\\-\\-\end{bmatrix}\begin{bmatrix}-\\-\\-\\-\\-\end{bmatrix}) = \boxed{\boxed{\boxed{\text{(drink,milk)}}\boxed{}}\boxed{}}$$

10. In the same way, the attributes in this constraint are fixed and they are represented as -

$$pad_5(\begin{bmatrix}-\\norwegian\\-\\-\\-\end{bmatrix}\begin{bmatrix}-\\-\\-\\-\\-\end{bmatrix}\begin{bmatrix}-\\-\\-\\-\\-\end{bmatrix}\begin{bmatrix}-\\-\\-\\-\\-\end{bmatrix}\begin{bmatrix}-\\-\\-\\-\\-\end{bmatrix}) = \boxed{\boxed{\text{(nation,norwegian)}}\boxed{}\boxed{}\boxed{}\boxed{}}$$

11. Since the attributes can be selected in two ways, it is represented as -

$$pad_5(\begin{bmatrix}-\\-\\-\\chester\\-\end{bmatrix}\begin{bmatrix}-\\-\\-\\-\\fox\end{bmatrix}) \cup pad_5(\begin{bmatrix}-\\-\\-\\-\\fox\end{bmatrix}\begin{bmatrix}-\\-\\-\\chester\\-\end{bmatrix})$$

12. In the similar manner, the attributes in this constraint are represented -

$$pad_5(\begin{bmatrix}-\\-\\-\\kools\\-\end{bmatrix}\begin{bmatrix}-\\-\\-\\-\\horse\end{bmatrix}) \cup pad_5(\begin{bmatrix}-\\-\\-\\-\\horse\end{bmatrix}\begin{bmatrix}-\\-\\-\\kools\\-\end{bmatrix})$$

13. $pad_5(\begin{bmatrix} - \\ - \\ oj \\ luckyStr \\ - \end{bmatrix})$ 14. $pad_5(\begin{bmatrix} - \\ japan \\ - \\ parliam \\ - \end{bmatrix})$

15. $pad_5(\begin{bmatrix} - \\ norwegian \\ - \\ - \\ - \end{bmatrix}\begin{bmatrix} blue \\ - \\ - \\ - \\ - \end{bmatrix}) \cup pad_5(\begin{bmatrix} blue \\ - \\ - \\ - \\ - \end{bmatrix}\begin{bmatrix} - \\ norwegian \\ - \\ - \\ - \end{bmatrix})$

Here pad_5 indicates the following -

$$pad_5(s) := \{s' \in \square^* s \square^* \mid length(s') = 5\}$$

Since attributes in constraint 9 and 10 have a single defined string, we first use these strings to reach the goal state.

3.2.3 Superposition Theorem

A variant of superposition theorem (Proposition 4 in [11]) is used to solve the zebra puzzle, which is defined as follows:

Given:
attributes Att := {color, nation, drink, smoke, pet}
domains D(color) := {red, blue, green, yellow, ivory}
⋮
D(pet) := {dog, fox, horse, snail, zebra}

The attributes and domains combined form records -
records R := $\{r : Att \to \bigcup_{a \in Att} D(a) \mid (\forall a \in Att) r(a) \in D(a)\}$

Sample space S_5 is given by -

$$S_5 := \{\alpha_1 \cdots \alpha_5 \mid (\exists r_1 \cdots r_5 \in \Omega_5)(\forall i \in [5]) \;\; \alpha_i \subseteq r_i \tag{3.1}$$

where

$$\Omega_5 := \{r_1 \cdots r_5 \in R^5 \mid (\forall a \in Att)(\forall i \in [4])(\forall j \;\; s.t. \;\; i<j \leq 5) r_i(a) \neq r_j(a)\}$$

and $[n] := \{1, \cdots, n\}$

Subrecords $R_0 := \{\alpha \mid (\exists r \in R)\alpha \subseteq r\}$
$= \bigcup_{A \subseteq Att} \{\alpha : A \to \bigcup_{a \in A} D(a) \mid (\forall a \in A)\alpha(a) \in D(a)\}$

R-Superposition:

$$\frac{}{\&_R(\in, \in, \in)} \qquad \frac{\&_R(s, s', s'') \qquad \alpha \cup \alpha' \in R_0}{\&_R(\alpha s, \alpha' s', (\alpha \cup \alpha')s'')}(\dagger) \tag{3.2}$$

where $\&_R := \bigcup_{n \geq 0} \{(\alpha_1 \cdots \alpha_n, \alpha'_1 \cdots \alpha'_n, \beta_1 \cdots \beta_n) \in R_0^n \times R_0^n \times R_0^n \mid (\forall i \in [n])\alpha_i \cup \alpha'_i = \beta_i\}$

Here $\alpha \cup \alpha' \in R_o$ ensures that, when two records are combined, the values of attributes in α and α' does not collide. If same attributes have different values, then this condition will fail.

To check if the final string has unique attribute-value pairs for all houses, we define (R,X,X')-superposition as -

$$\frac{\&_R^{X,X'}(s, s', s'') \qquad \alpha \cup \alpha' \in R_0 \qquad \alpha \cap X' \subseteq \alpha' \qquad \alpha' \cap X \subseteq \alpha}{\&_R^{X,X'}(\alpha s, \alpha' s', (\alpha \cup \alpha')s'')} \tag{3.3}$$

where $\&_R^{X,X'} := \bigcup_{n \geq 0} \{(\alpha_1 \cdots \alpha_n, \alpha'_1 \cdots \alpha'_n, \beta_1 \cdots \beta_n) \in R_0^n \times R_0^n \times R_0^n \mid (\forall i \in [n])\alpha_i \cup \alpha'_i = \beta_i\}$ and $\alpha_i \cap X' \subseteq \alpha'_i$ and $\alpha'_i \cap X \subseteq \alpha_i$

Here X and X' are finite sets containing attribute value pair in the form of $\{a, v\}$ where $a \in Att$ and $v \in D(a)$. These sets are generally defined as vocabularies for s

and s' where vocabulary for $s = \alpha_1 \cdots \alpha_n$ is written as -

$$voc(\alpha_1 \cdots \alpha_n) := \alpha_1 \cup \cdots \cup \alpha_n \tag{3.4}$$

By applying one of the constraint(φ) from Φ (list of all the constraints) on a string s, the strings generated by $S_5[\varphi]$ is superposed with s which is denoted as -

$$S_5[s, \varphi] := \{s'' \mid (\exists s' \in S_5[\varphi])\&_R^{voc(s),voc(s')}(s, s', s'') \tag{3.5}$$

So, the goal of this superposition is to pick a constraint $\varphi \in \Phi$ such that the number of strings generated (s'' in $S_5[s, \varphi]$) is minimum given a node (s, Φ), where $s \in S_5$ and Φ is a finite set of constraints to satisfy.

Thus, we start with $(\square^5, L)$ where L is list of constraints to be satisfied and we superpose till we get $(s, \square)$.

3.2.4 Simulation

This implementation was done using Python TkInter library [18]. This library provides a graphical user interface to simulate problems. Thus, we can add shapes, textboxes and buttons to visually depict a scenario. The solution for Zebra puzzle was found using the following steps -

1. An initial state is defined using a dictionary which depicts the five houses with attributes as follows -

Listing 3.1: String representation for Initial state in the zebra puzzle

```
{
 1:{'color':'','nationality':'','drink':'','smoke':'','animal'
    ↪ :''},
 2:{'color':'','nationality':'','drink':'','smoke':'','animal'
    ↪ :''},
 3:{'color':'','nationality':'','drink':'','smoke':'','animal'
    ↪ :''},
```

```
 4:{'color':'','nationality':'','drink':'','smoke':'','animal'
    ↪ :''},
 5:{'color':'','nationality':'','drink':'','smoke':'','animal'
    ↪ :''}
}
```

2. To apply the constraints on the current state, a dictionary is defined -

Listing 3.2: Constraints in the zebra puzzle

```
{
 2: [0, 'nr', {'color': 'red', 'nationality': 'englishman'}],
 3: [0, 'nr', {'nationality': 'spaniard', 'animal': 'dog'}],
 4: [0, 'nr', {'color': 'green', 'drink': 'coffee'}],
 5: [0, 'nr', {'nationality': 'ukrainian', 'drink': 'tea'}],
 6: [0, 'nr', {'color': 'ivory'}, {'color': 'green'}],
 7: [0, 'nr', {'smoke': 'oldgold', 'animal': 'snail'}],
 8: [0, 'nr', {'smoke': 'kools', 'color': 'yellow'}],
 9: [3, 'nr', {'drink': 'milk'}],
 10: [1, 'nr', {'nationality': 'norwegian'}],
 11: [0, 'r', {'smoke': 'chesterfields'}, {'animal': 'fox'}],
 12: [0, 'r', {'smoke': 'kools'}, {'animal': 'horse'}],
 13: [0, 'nr', {'smoke': 'luckystrike', 'drink': 'orangejuice'
    ↪ }],
 14: [0, 'nr', {'smoke': 'parliaments', 'nationality': '
    ↪ japanese'}],
 15: [0, 'r', {'nationality': 'norwegian'}, {'color': 'blue'
    ↪ }],
}
```

It takes first input as house number given in the constraint. The second input indicates if the constraint is 'nr' or 'r' stating that if the constaint is 'r' then it involves two houses where the specified constraint can be in any of the house. The third input is the string format for the constraint using the dictionary defined in above listing. Also, a rule list is maintained to keep a track of which constraints were left to apply to the current state.

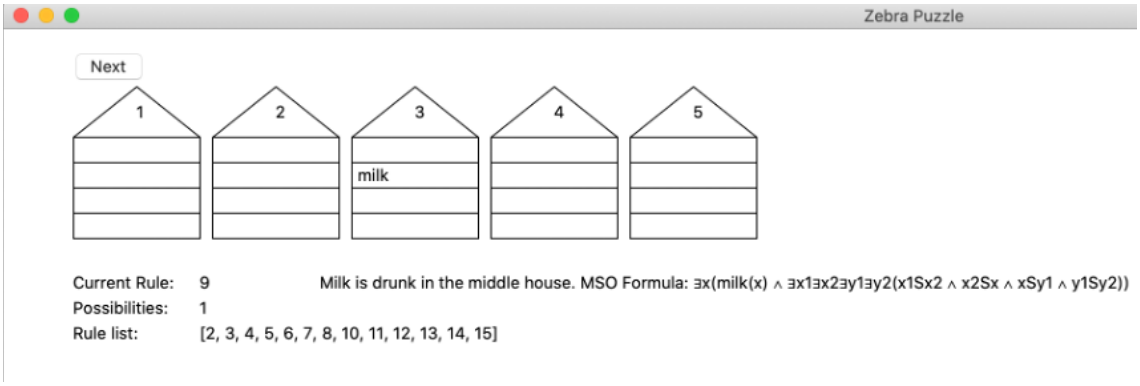

Figure 3.1: Initial State for Zebra Puzzle

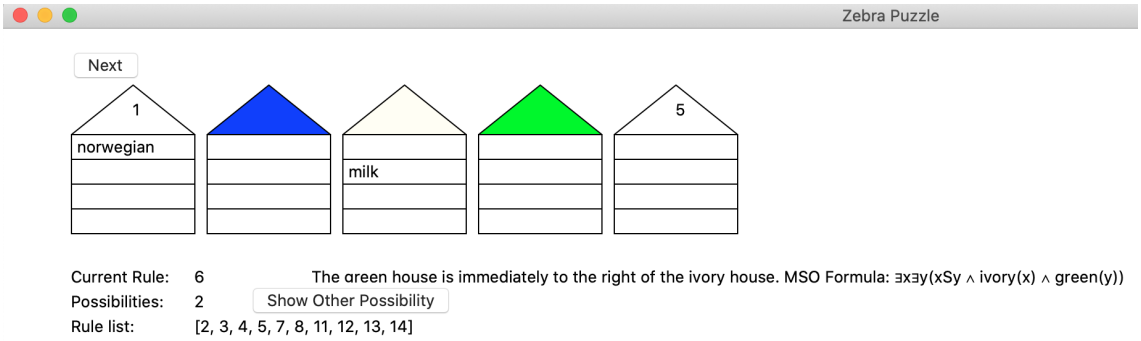

Figure 3.2: More than one possible strings in Zebra Puzzle

3. Superposition theorem is applied on all the constraints from the rule list and the current state to obtain the maximum number of strings that satisfy each constraint. The constraint with minimum number of possible obtained strings after applying this theorem is chosen and applied to the current state. Thus, From figure 3.1, it is evident that the first rule which is applied to the current state is constraint 9 (Milk is drunk in the middle house) since it gives only one possible string after superposition that satisfies the constraint. The current state is updated with this constraint and the new state is as follows -

Listing 3.3: String representation for Rule 9 in the zebra puzzle

```
{
 1:{'color':'','nationality':'','drink':'','smoke':'','animal'
    ↪ :''},
 2:{'color':'','nationality':'','drink':'','smoke':'','animal'
```

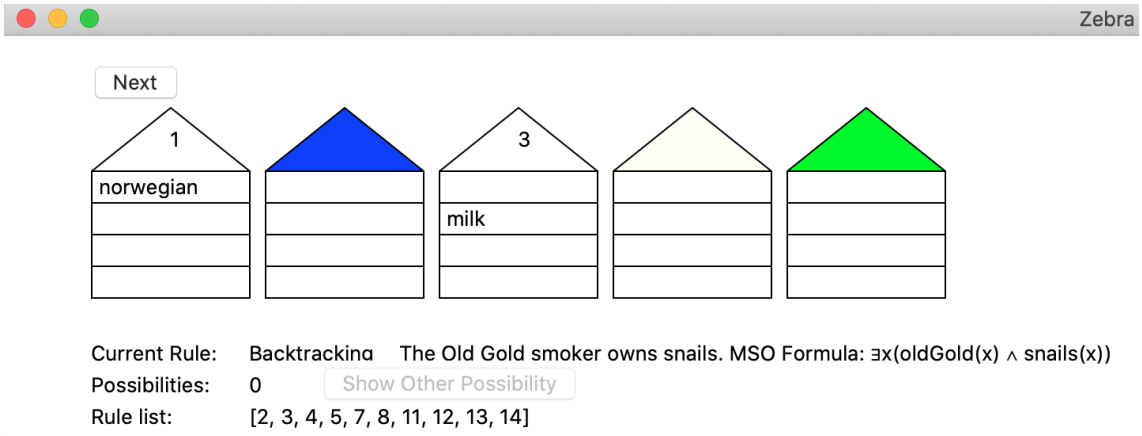

Figure 3.3: Backtracking to previous state in Zebra Puzzle

```
    ↪ :''},
  3:{'color':'','nationality':'','drink':'milk','smoke':'','
    ↪ animal':''},
  4:{'color':'','nationality':'','drink':'','smoke':'','animal'
    ↪ :''},
  5:{'color':'','nationality':'','drink':'','smoke':'','animal'
    ↪ :''}
}
```

4. Proceeding further, minimum two possible strings are obtained for the constraint 6 (The green house is immediately to the right of the ivory house) which is shown in 3.2. Thus, using the button "Show Possibility", we can view the other possibility on the current state of zebra puzzle. However, the program applies the first possibility on the current state.

5. If at a later stage, no possible strings is obtained from superposition theorem that satisfies the constraint, then it is backtracked to the state where there were more than one possible string (as shown in figure 3.3).

6. Thus, on backtracking and using the other possible string we get the final solution which gives an empty rule list as shown in figure 3.4. Here, it can be seen that the last rule that was applied was constraint 11 (The man who smokes Chesterfields

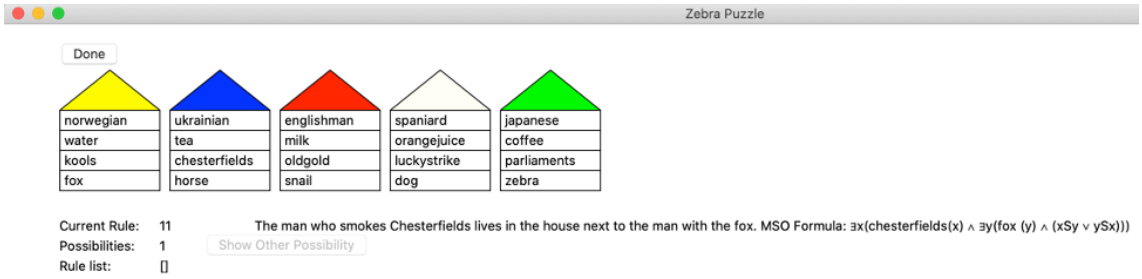

Figure 3.4: Final solution for Zebra Puzzle

lives in the house next to the man with the fox) after which it was deduced that zebra lives in the green house and water is drunk in the yellow house.

3.3 Allen Transitivity Grid

3.3.1 MSO Logic Representation

The transitivity relation between two Allen interval relations can be obtained using the superposition theorem given in algorithm 1. In order to apply the theorem on Allen relations, some presuppositions are considered to generalize the sample space which are -

1. Intervals are non empty sets without holes where -

$$\begin{aligned} \text{interval}[c] &:= \exists x P_c(x) \wedge \neg\exists y \ \text{hole}_c(y) \\ \text{hole}_c(x) &:= \exists y(y < x \wedge P_c(y)) \ \wedge \ \exists z(x < z \wedge P_c(z)) \wedge \neg P_c(x) \end{aligned}$$

   where $P_c(x)$ represents position of c at x

2. Bounded intervals are equivalent to periods which is denoted as -

$$\begin{aligned} \text{period}[c] &:= \text{interval}(c) \wedge \neg(\text{first}[c] \vee \text{last}[c]) \\ \text{first}[c] &:= \exists x(P_c(x) \wedge \neg\exists y(ySx)) \\ \text{last}[c] &:= \exists x(P_c(x) \wedge \neg\exists y(xSy)) \end{aligned}$$

So for every Allen relation R,

$$aRb \text{ presupposes } \text{period}[a] \wedge \text{period}[b]$$

Thus, Allen Relation for a string s can be defined as an MSO formula $\chi_R[a, b]$ as follows:

$$aRb \text{ in } s \iff s \models \chi_R[a, b] \tag{3.6}$$

where

$$s \models \text{period}[a] \wedge \text{period}[b],$$

The formula for $\chi_R[a, b]$ can be given as -

$$\exists x \exists y \exists x' \exists y' (P_{l(a)}(x) \wedge P_{r(a)}(y) \wedge P_{l(b)}(x') \wedge P_{r(b)}(y') \wedge <_R (x, y, x', y')) \tag{3.7}$$

where

$$\begin{aligned} P_{l(a)}(u) &:= \exists v(uSv \wedge P_a(v)) \wedge \neg P_a(u) \\ P_{r(a)}(u) &:= P_a(u) \wedge \neg \exists v(uSv \wedge P_a(v)) \end{aligned}$$

and $<_R (x, y, x', y')$ is given by

R	$<_R (x, y, x', y')$	R^{-1}	$<_{R^{-1}} (x, y, x', y')$
b	$y < x'$	bi	$y' < x$
d	$x' < x \wedge y < y'$	di	$x < x' \wedge y' < y$
o	$x < x' \wedge x' < y \wedge y < y'$	oi	$x' < x \wedge x < y' \wedge y' < y$
m	$y = x'$	mi	$y' = x$
s	$x = x' \wedge y < y'$	si	$x = x' \wedge y' < y$
f	$x' < x \wedge y = y'$	fi	$x < x' \wedge y = y'$
e	$x = x' \wedge y = y'$	e	

where

$$\begin{aligned} u < v &:= \exists X(\text{closed}(X) \wedge X(v) \wedge \neg X(u)) \text{ where} \\ \text{closed}(X) &:= \forall x \forall y(X(x) \wedge xSy \supset X(y)) \end{aligned}$$

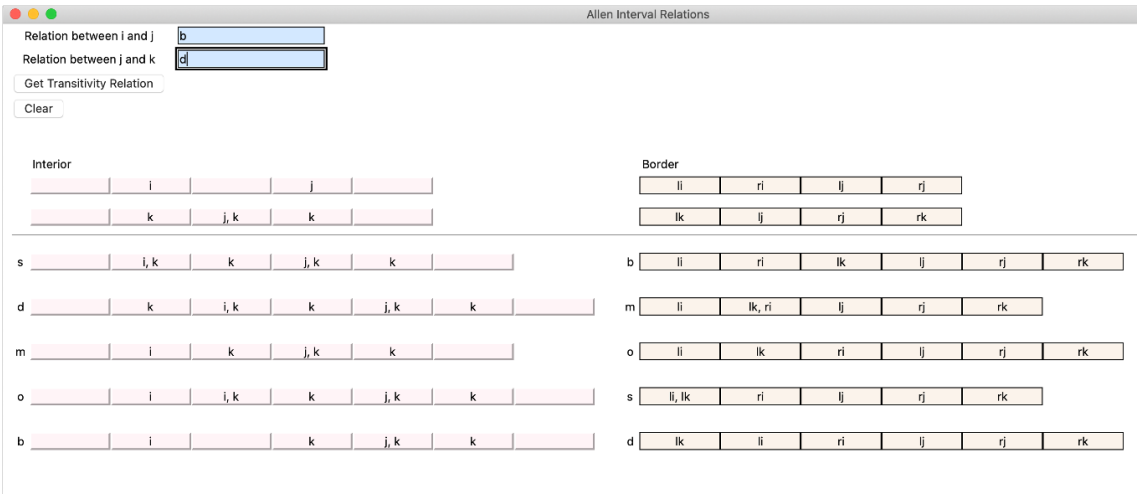

Figure 3.5: Simulation for Allen Relation

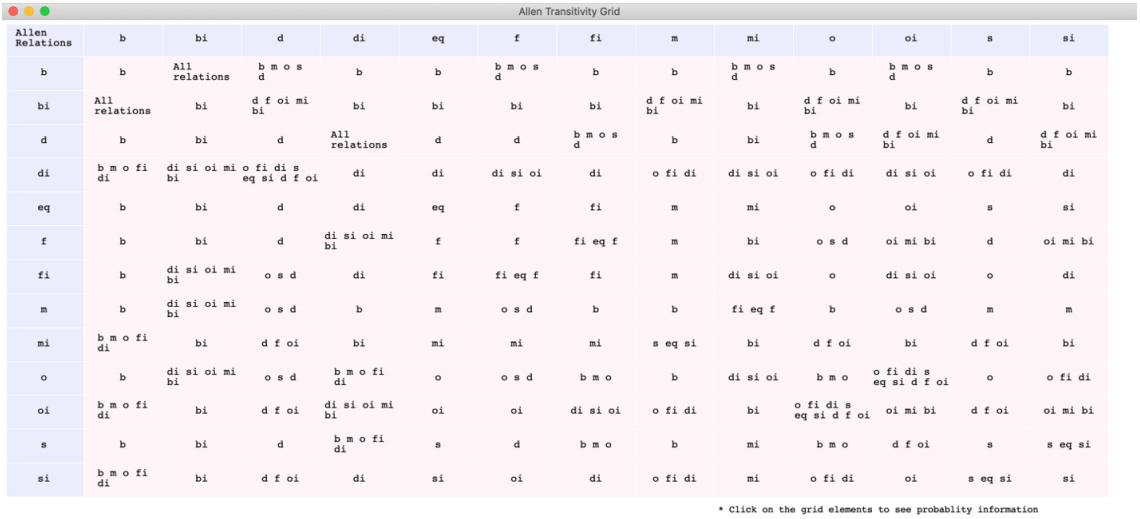

Figure 3.6: Allen Transitivity Grid

3.3.2 Simulation

These formulas were also implemented using Python TkInter library. From the figure 3.5 we see that by accepting the two relations between i,j intervals and j,k intervals, we get the transitive relations between i,k in interior as well as border format. This is done by first converting the two relations into strings of required format and applying the superposition theorem discussed in section 3.1 on the strings. By applying this theorem on every pair of relations, the transitivity table can be obtained as shown in the figure 3.6.

3.4 Probabilities of Allen Relations

3.4.1 Calculation

To calculate the probabilities of Allen Relations using the transitivity table, the border format of the strings is considered. A sample space of n points is used to calculate probabilities between three intervals I, I', I'' where

$$I = (f(x), f(y)],\ I' = (f(x'), f(y')],\ I'' = (f(x''), f(y'')]$$

Here $f(x), f(y)$ denote the left and right borders for interval I.
Similarly, $f(x'), f(y'), f(x''), f(y'')$ denotes the left and right borders for interval I' and I'' respectively.

If n = 6 and positions for left and rights borders of I, I', I'' are as follows -

$$\{(x, 1), (x', 2), (x'', 1), (y, 2), (y', 3), (y'', 5)\}$$

This can be visualized as a string which is shown as -

x, x''	y, x'	y'		y''	

with Allen Relations as -

$$I\ meets\ I' \qquad x{<}y = x'{<}y'$$

$$I' \; during \; I'' \qquad x''<x'<y<y''$$

$$I \; starts \; I'' \qquad x = x''<y<y''$$

Thus, if two relations R and R' is defined as an entry in Allen transitivity table, then the transitivity t between these two relations can be given as -

$$t(R, R') = \{R'' \in Allen \mid there\, are\, intervals\, I, I', I''s.t. IRI', I'R'I''\, and\, IR''I''\} \quad (3.8)$$

In the previous example, there is a world of 6 points where $IRI', I'R'I''$ *and* $IR''I''$. So, if $R'' \in t(R, R')$, we can define the cardinality or length of these three strings as -

$$l(R, R', R'') := least\; n\; with\; an\; n - world\; where\; IRI', I'R'I''\; and\; IR'I'' \qquad (3.9)$$

Hence, in the previous example, since the borders for I, I', I'' fill only 4 positions in 6-world, the $l(meets, during, starts)$ is 4.

For Allen Relations R, R' *and* R'', let

$$K_n(R, R', R'') := number\; of\; n - worlds\; where\; IRI', I'R'I''\; and\; IR'I'' \qquad (3.10)$$

If $R'' \notin t(R, R')$ then there will be no n-worlds where $IRI', I'R'I''$ *and* $IR'I''$ which makes $K_n(R, R', R'') = 0$. Otherwise, $K_n(R, R', R'')$ is numbers of subsets of n-worlds of cardinality $l(R, R', R'')$ and is represented as -

$$K_n(R, R', R'') = \binom{n}{(l(R, R', R'')} for R'' \in t(R, R') \qquad (3.11)$$

Here $\binom{n}{(l(R, R', R'')}$ indicates n choose $(l(R, R', R'')$.

The probability of n-worlds where $IRI', I'R'I''$ *and* $IR'I''$ is $K_n(R, R', R'')$ divided by the cardinality of the set of n-worlds. Since we select two points for $R, R' and R''$ from n points, the cardinality can be computed as -

$$\binom{n}{2}^3 = \frac{n^3(n-1)^3}{8} \tag{3.12}$$

By combining equations 3.11 and 3.12, the probability can be calculated as follows -

$$p_n(R, R', R'') = \frac{8}{n^3(n-1)^3} \binom{n}{(l(R, R', R''))} \; for \; R'' \in t(R, R') \tag{3.13}$$

which is 0 if $R'' \notin t(R, R')$.

Using equation 3.13, we can calculate the conditional probabilities -

$$p_n(R''|R, R') = \frac{p_n(R, R', R'')}{p_n(R, R')} = \frac{K_n(R, R', R'')}{K_n(R, R')} \tag{3.14}$$

where,

$$p_n(R, R') = \sum_{R'' \in t(R,R')} p_n(R, R', R'')$$

or

$$K_n(R, R') = \sum_{R'' \in t(R,R')} K_n(R, R', R'')$$

3.4.2 Simulation

The above equations were simulated using Python TkInter library and the Allen Transitivity Grid (shown in figure 3.6). As shown in figure 3.7, if R is b and R' is d, then R'' is one of the 5 relations (b,m,o,s,d). The probability can be calculated using the following code -

Listing 3.4: Probability Calculation

```
def showProbabilities(r1,r2,root):
    global choice, len_n
    if int(choice.get())==2:
        max = int(len_n.get())
```

```
else:
    max = maxLength()
totLen = calculateProbR1R2(max)
for i,s in enumerate(result):
    f = Frame(root, height=20, width=200)
    f.pack_propagate(0) # don't shrink
    if(i>5):
        f.place(x=400, y=0+((j)*20))
    else:
        j=0
        f.place(x=0, y=0+((i)*20))
    prob = 8 / (pow(max,3)*pow(max-1,3))
    prob1 = prob * nCr(max,len(s))
    label = Label(f, text="p("+r1+","+r2+","+allInv(proj(s,{l
        ↪ (0),r(0),l(2),r(2)}),0,2)+") = "+str(round(prob1, 4))
        ↪ ,font=("Courier", 13))#label
    label.pack(fill=BOTH, expand=1)
    f1 = Frame(root, height=20, width=200)
    f1.pack_propagate(0) # don't shrink
    if(i>5):
        f1.place(x=600, y=0+((j)*20))
        j=j+1
    else:
        f1.place(x=200, y=0+((i)*20))
    prob = nCr(max,len(s)) / totLen
    label = Label(f1, text="p("+allInv(proj(s,{l(0),r(0),l(2),r
        ↪ (2)}),0,2)+"|"+r1+","+r2+") = "+str(round(prob, 4)),
        ↪ font=("Courier", 13))#label
    label.pack(fill=BOTH, expand=1)
```

The steps taken in this calculation are -

- Firstly, $(l(R, R', R'')$ is calculated for each R'' relation by using the strings obtained from superposition theorem (shown in figure 3.6)-
  tt(b,d) = [b,m,o,s,d]
  b = [{li}, {ri}, {lk}, {lj}, {rj}, {rk}] (length = 6)

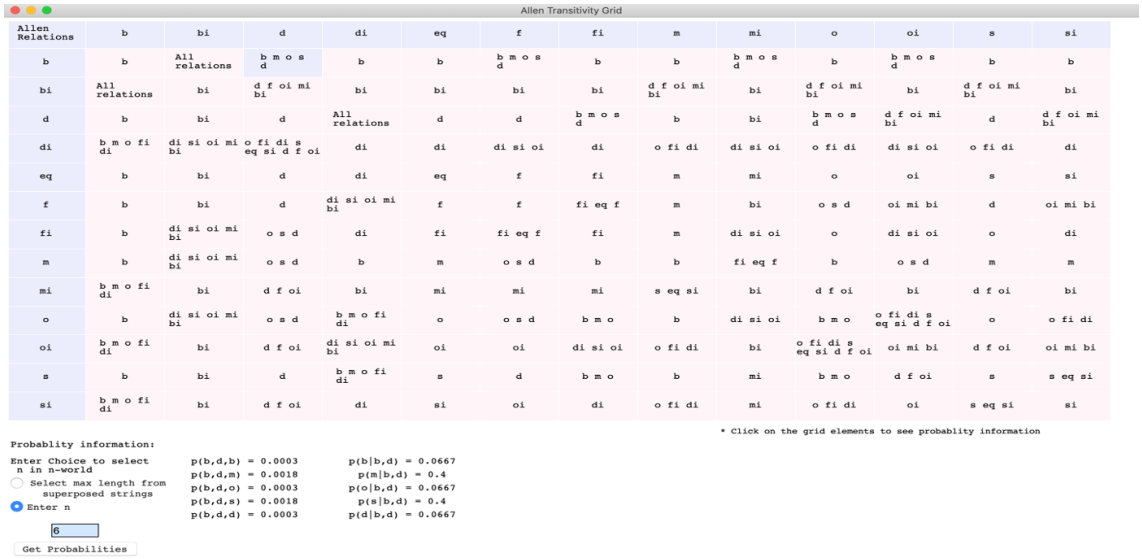

Figure 3.7: Probability of Allen Relations

m = [{li}, {ri, lk}, {lj}, {rj}, {rk}] (length = 5)
o = [{li}, {lk}, {ri}, {lj}, {rj}, {rk}] (length = 6)
s = [{li, lk}, {ri}, {lj}, {rj}, {rk}] (length = 5)
d = [{lk}, {li}, {ri}, {lj}, {rj}, {rk}] (length = 6)

- The number of points (or n-world) is either taken as an input or maximum length is considered from the length of above 5 strings (in this case,n=6).

- The input length n (=6) and the length of the relations (=either 5 or 6 depending on the relation) are applied to the equation 3.13 as follows -

$$p_6(b,d,b) = \frac{8}{6^3 5^3}\binom{6}{6} = 0.000296296296296296$$

$$p_6(b,d,m) = \frac{8}{6^3 5^3}\binom{6}{5} = 6p_6(b,d,b) = 0.0017777777777777779$$

- The conditional probability for these relations can be calculated using equation 3.14 -

$$K_n(R, R') = \binom{6}{6} + \binom{6}{5} + \binom{6}{6} + \binom{6}{5} + \binom{6}{6} = 1 + 6 + 1 + 6 + 1$$

$$p_6(b|b, d) = \frac{1}{1 + 6 + 1 + 6 + 1} = \frac{1}{15}$$

$$p_6(m|b, d) = \frac{6}{1 + 6 + 1 + 6 + 1} = \frac{6}{15}$$

3.5 Probability of Allen Relations using Markov Logic Network

3.5.1 Calculation

The above implementation of calculating probabilities for Allen Relations was done without any weight applied to the relations. If different probability is required for each relation, a weight is associated with every Allen relation and probabilities are calculated using Markov Logic Network. To apply the equation 2.17, we need to calculate the partition function (Z). For achieving that, we define a softmax function which is used to normalize weights so that the probability lies between 0 and 1 [19]. Using this function, the weights are adjusted by subtracting it from the maximum of all weights. If there are m number of weights such that $w_1, ..., w_m$ are mapped to m non-negative numbers which sum upto 1, it is calculated as -

$$\begin{aligned} sm(w_1, ..., w_m) &:= (e^{w_1}/z, ..., e^{w_m}/z) \quad where \quad z := \sum_{i=1}^{m} e^{w_i} \\ &= sm(y_1, ..., y_m) \quad where \quad y_i := w_i - max\{w_1, ..., w_m\} \end{aligned} \tag{3.15}$$

An n-sample (also amounts to 5-tuple) function is defined to obtain positions of the borders of the two intervals and number of points in the world (as done in section 3.4). It is defined as -

$$f : \{x, y, x', y'\} \to [n] \tag{3.16}$$

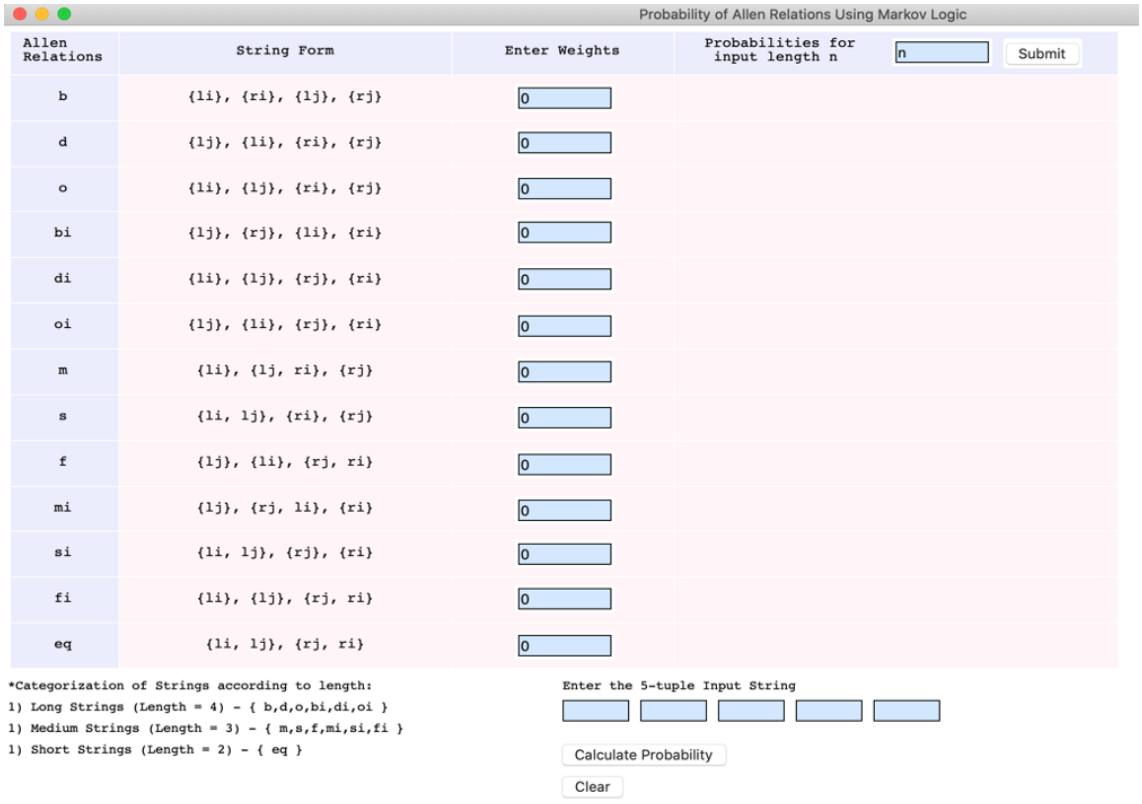

Figure 3.8: Inputs for Calculating Probability using Markov Logic

such that

$$f(x)<f(y) \quad and \quad f(x')<f(y')$$

Here $f(x), f(y), f(x'), f(y')$ denote the borders for the two intervals.

Thus, to select two intervals from n sample space,it is denoted as -

$$\binom{n}{2} \times \binom{n}{2} = 6(\binom{n}{4} + \binom{n}{3}) + \binom{n}{2} \tag{3.17}$$

Since there are three categories of Allen Relations (discussed in section 2.8), three cases are defined to describe the Allen relation -

CASE 1 There are 6 long Allen relation b,d,o,bi,di,oi (length = 4) which can be described

as $\binom{n}{4}$ n-samples.

CASE 2 There are 6 medium Allen relation m,s,f,mi,si (length = 3) which can be described as $\binom{n}{3}$ n-samples.

CASE 3 There is a single short Allen relation eq (length = 2) which can be described as $\binom{n}{2}$ n-samples.

Therefore, if the 13 Allen Relations are assigned a weight w_j, then -

$$for\ 1 \leq j \leq 6, \binom{n}{4}\ n - samples\ have\ weight\ w_j$$

$$for\ 7 \leq j \leq 12, \binom{n}{3}\ n - samples\ have\ weight\ w_j$$

and

$$for\ j = 13, \binom{n}{2}\ n - samples\ have\ weight\ w_j$$

3.5.2 Simulation

This implementation has also been simulated using Python TkInter library. The inputs taken (shown in figure 3.8) are -

- Weights for all the Allen relations
- Number of n points (which is the length of the input string that denotes the two intervals)
- n-sample (or 5-tuple) function

In the figure 3.9, we can see that, by assigning no (or 0) weights to Allen relations, the probabilities over n-samples calculated is equivalent to the ones calculated in [2]

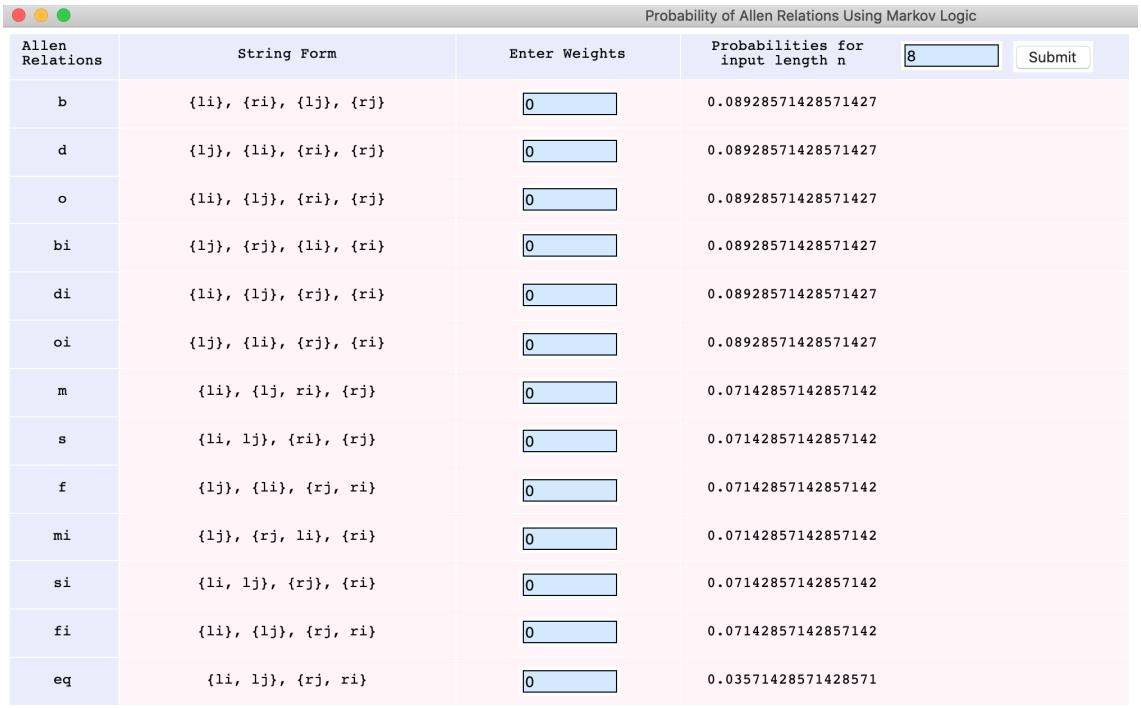

Figure 3.9: Probability using Markov Logic with weight 0

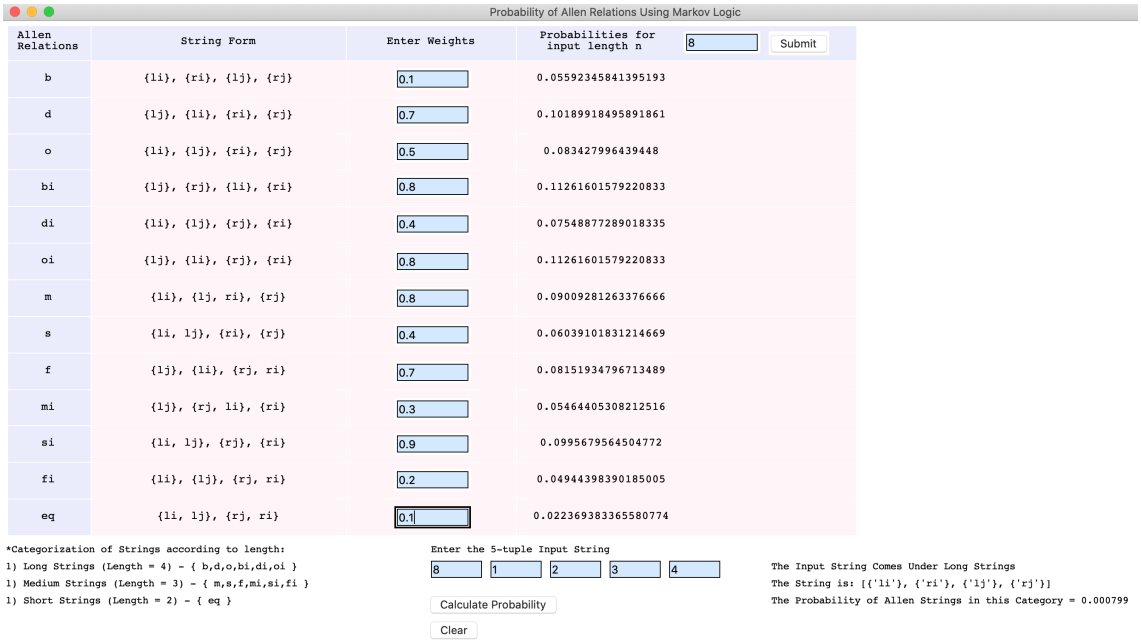

Figure 3.10: Probability using Markov Logic with different weights

since all relations become equiprobable.

From the figure 3.10, it is evident that if different weights are applied to the relations, a different probability is calculated for each relation. For example, for the first relation b , the calculation is done using following steps -

- All the weights are normalized using the softmax function given in equation 3.15. The max weight is 0.9 with which all the weights are recalculated -
  b := exponential of (0.1 - 0.9) = 0.4493
  d := exponential of (0.7 - 0.9) = 0.8187
  o := exponential of (0.6 - 0.9) = 0.7408
  bi := exponential of (0.8 - 0.9) = 0.9048
  di := exponential of (0.4 - 0.9) = 0.6065
  oi := exponential of (0.8 - 0.9) = 0.9048
  m := exponential of (0.8 - 0.9) = 0.9048
  s := exponential of (0.4 - 0.9) = 0.6065
  f := exponential of (0.7 - 0.9) = 0.8187
  mi := exponential of (0.3 - 0.9) = 0.5488
  si := exponential of (0.9 - 0.9) = 1
  fi := exponential of (0.2 - 0.9) = 0.4965
  eq := exponential of (0.1 - 0.9) = 0.4493

- Using these adjusted weights, the partition function is calculated according to the category of the relation -

  $$Z = \binom{8}{4} * 0.4493 + \binom{8}{4} * 0.8187 + \binom{8}{4} * 0.7408 + \binom{8}{4} * 0.9048 + \binom{8}{4} * 0.6065 + \binom{8}{4} * 0.9048 + \binom{8}{3} * 0.9048 + \binom{8}{3} * 0.6065 + \binom{8}{3} * 0.8187 + \binom{8}{3} * 0.5488 + \binom{8}{3} * 1 + \binom{8}{3} * 0.4965 + \binom{8}{2} * 0.4493$$

  $$= 567.3402$$

- Since b is a string under long category, the probability is calculated as -

(adjusted weight for b (0.4493) / Z (567.3402)) * $\binom{8}{4}$

= 0.55923

The code snippet for these functions is as follows -

Listing 3.5: Partition Function

```
def zNorm(q,n):
    z = 0
    for x in q[0:6]:                # long 'b','d','o','bi','di','oi'
        z += math.comb(n,4)*x
    for x in q[6:12]:               # medium 'm','s','f','mi','si','fi'
        z += math.comb(n,3)*x
    z += math.comb(n,2) * q[12] # short 'eq'
    return z
```

Listing 3.6: Probability for Allen Relations

```
def allenWs(n):
    get_input_weights()
    q = np.exp(weights - np.max(weights))
    return q / zNorm(q,n)

def wn2prob(n):
    global temp
    p = allenWs(n)
    temp=[]
    for x in p[0:6]:                # long
        temp.append(math.comb(n,4)*x)
    for x in p[6:12]:               # medium
        temp.append(math.comb(n,3)*x)
    temp.append(math.comb(n,2) * p[12])
```

To get the probability of a single relation according to its category, we use the n-sample function in which input length n is given in the first box which is same as

that used while calculating the probabilities for all samples. From the rest of the four boxes (which depict the positions of border of the two intervals), the Allen relation is determined. To compute the probability of this relation, the probability over n-samples is divided by the appropriate case described above.

For example, in the figure 3.10, the input length n is 8 and the positions of the borders of the intervals states the probability needs to be calculated for relation b which comes under long category (table 2.3). Thus the probability is 0.55923 (probability of b over 8 samples) divided by $\binom{n}{4}$ (description of long relations) which is equal to 0.000799. The code snippet for the following is -

Listing 3.7: Probability for Each Allen Relation

```
def show_probability(root):
    input_string = get_input_strings()
    rel = alRel(input_string)
    wn2prob(input_string[0])
    if rel>=0 and rel<=6:
        Label(root, text="The␣Input␣String␣Comes␣Under␣Long␣Strings
            ↪ ",font=("Courier", 12)).place(x=0,y=0)
        Label(root, text="The␣String␣is:␣"+str(allen('i','j',rel)),
            ↪ font=("Courier", 12)).place(x=0,y=20)
        probability = temp[rel]/math.comb(input_string[0],4)
        Label(root, text="The␣Probability␣of␣Allen␣Strings␣in␣this␣
            ↪ Category␣=␣"+str(round(probability, 6)),font=("
            ↪ Courier", 12)).place(x=0,y=40)
    elif rel>=7 and rel<=11:
        Label(root, text="The␣Input␣String␣Comes␣Under␣Medium␣
            ↪ Strings",font=("Courier", 12)).place(x=0,y=0)
        Label(root, text="The␣String␣is:␣"+str(allen('i','j',rel)),
            ↪ font=("Courier", 12)).place(x=0,y=20)
        probability = temp[rel]/math.comb(input_string[0],3)
        Label(root, text="The␣Probability␣of␣Allen␣Strings␣in␣this␣
            ↪ Category␣=␣"+str(round(probability, 6)),font=("
```

```
        ↪ Courier", 12)).place(x=0,y=40)
elif rel==12:
    Label(root, text="The Input String Comes Under Short
        ↪ Strings",font=("Courier", 12)).place(x=0,y=0)
    Label(root, text="The String is: "+str(allen('i','j',rel)),
        ↪ font=("Courier", 12)).place(x=0,y=20)
    probability = temp[rel]/math.comb(input_string[0],2)
    Label(root, text="The Probability of Allen Strings in this
        ↪ Category = "+str(round(probability, 6)),font=("
        ↪ Courier", 12)).place(x=0,y=40)
else:
    Label(root, text="No Categorization for the Input String",
        ↪ font=("Courier", 12)).place(x=0,y=0)
```

Chapter 4

Conclusion

In conclusion, we were able to use finite state methods to reason over the constraints, which were Allen relations in our case. To do that, we first researched on First order and MSO Logic. In the course of the research, strings were represented using border format (except in the case of zebra puzzle which was represented using interior format). To understand the two representations of the strings (interior and border formats), temporal representation of the strings was studied. Also, as seen in the paper, MSO logic is equivalent to regular languages (which is expressed using finite state methods). Therefore, MSO formulas were used to represent the constraints of the Zebra Puzzle and Allen Relations.

Superposition theorem was used to solve the Zebra puzzle and find the transitivity table for Allen relations because it is equivalent to MSO conjunction. This theorem gave a quicker result and took less amount of space to store the partial states. Formulating the MSO logical formulas and applying the superposition theorem was a bit tedious task since the research work was at a preliminary stage.

After understanding these case studies and theorems, the second objective of this paper was to find probabilities for these constraints. Firstly, the probabilities of Allen relations in the transitivity table was calculated using the logic given by James Allen. The calculation for the conditional probabilities of the transitivity relations given in this paper was an extension of above logical equations. The probabilities were also

Allen Relations	String Form	Enter Weights	Probabilities for input length n 8 Submit
b	{l0}, {r0}, {l1}, {r1}	0.452	0.07692989371458926
d	{l1}, {l0}, {r0}, {r1}	0.452	0.07692989371458926
o	{l0}, {l1}, {r0}, {r1}	0.452	0.07692989371458926
bi	{l1}, {r1}, {l0}, {r0}	0.452	0.07692989371458926
di	{l0}, {l1}, {r1}, {r0}	0.452	0.07692989371458926
oi	{l1}, {l0}, {r1}, {r0}	0.452	0.07692989371458926
m	{l0}, {l1, r0}, {r1}	0.675	0.07691885111985297
s	{l0, l1}, {r0}, {r1}	0.675	0.07691885111985297
f	{l1}, {l0}, {r1, r0}	0.675	0.07691885111985297
mi	{l1}, {l0, r1}, {r0}	0.675	0.07691885111985297
si	{l0, l1}, {r1}, {r0}	0.675	0.07691885111985297
fi	{l0}, {l1}, {r1, r0}	0.675	0.07691885111985297
eq	{l0, l1}, {r1, r0}	1.368	0.07690753099334652

Figure 4.1: Equal Probability of Allen Relations for n=8

calculated using the Markov Logic network. For achieving this, firstly, Markov network was studied, after which we understood the Markov Logic Network. In these formulas, weights are assigned to the constraints which give different probabilities for different relations (as seen in the design and implementation chapter). This logic is often used to get an uniform probability for all the 13 relations. But this also proved to be a challenging part of the research.

4.1 Results

As seen from the section 3.4, even though the probabilities for all Allen relations is equiprobable since no weight is given to the relation, a uniform probability is not obtained for the relations due to different categories of the strings. To get a uniform probability for the relations, appropriate weights are allocated to the relations as shown in the figure 4.1.

The weights of the relations and the length of the sample space (input string or n

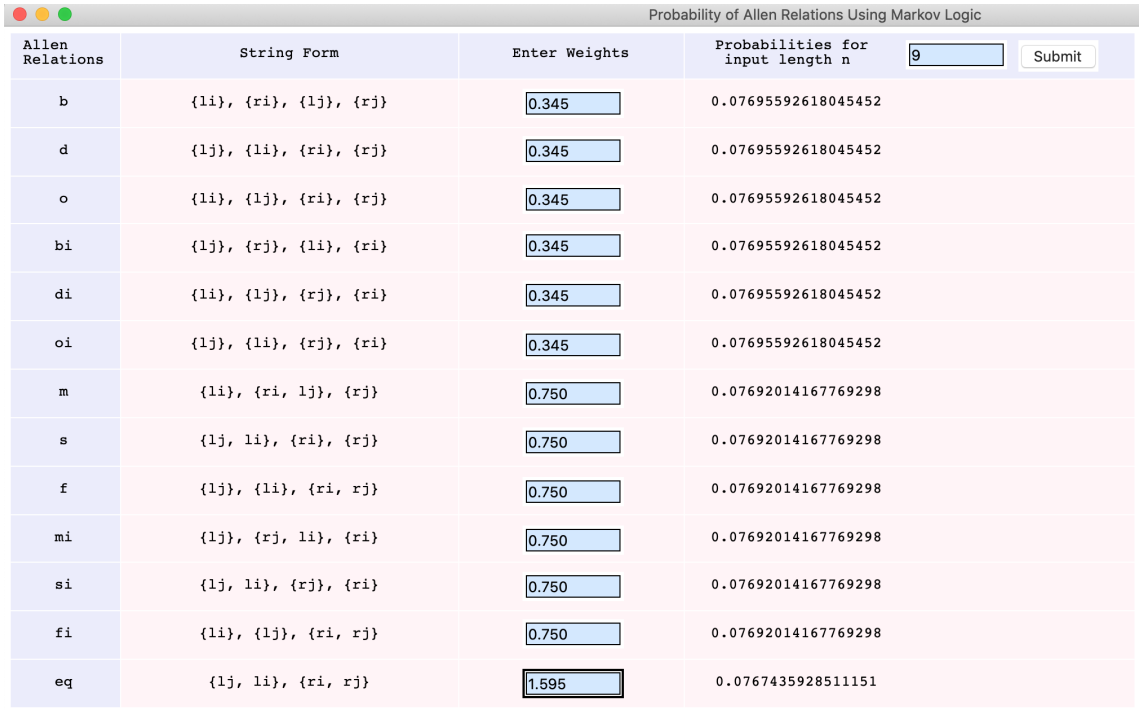

Figure 4.2: Equal Probability of Allen Relations for n=9

points) are correlated. As discussed in section 2.8, the long Allen relations become more probable as the length n increases. Due to this fact, to maintain a uniform probability, the following steps are taken -

- Decrease the weights for long Allen relation strings as n increases
- Increase the weights for medium and short Allen relation strings as n increases.

Thus from figures 4.1 and 4.2, we can see that -

- For n=8, weights for -
  Long relations = 0.452, Medium relations = 0.675, Short relations = 1.368
- For n=9, weights for -
  Long relations = 0.345, Medium relations = 0.750, Short relations = 1.595

4.2 Further Work

As seen in the prior work, formulating the MSO formulas can be done with ease in the MONA tool. This paper discusses how it can be done, but it was not used when implementing the two case studies. Since the structure and coding style of this tool is quite different, it required more amount of time and effort to understand and build the formulas. Thus, the implementation of the MSO logical formulas in MONA has been kept as a further work.

The probabilities using the Markov logic network was calculated for Allen relations by assigning different weights to them. A similar logic can be applied to find a solution for the Zebra Puzzle. As a further study, one can understand how appropriate weights can be employed to the constraints so that the probability computed for the solution is higher than that of other expected results combined or taken individually. Additionally, the partition function used in the calculation for Markov Logic network can also be researched particularly for Zebra Puzzle. With that, a generalised understanding of the applied weights can be assessed and applied to any Constraint Satisfaction Problem.

Bibliography

[1] J. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, pp. 510–521, 01 2013.

[2] T. Fernando and C. Vogel, "Prior probabilities of allen interval relations over finite orders," in *Proceedings of the 11th International Conference on Agents and Artificial Intelligence - Volume 2: NLPinAI*, pp. 952–961, 01 2019.

[3] N. Klarlund and A. Møller, *MONA Version 1.4 User Manual*. BRICS, Department of Computer Science, University of Aarhus, January 2001. Notes Series NS-01-1. Available from `http://www.brics.dk/mona/`.

[4] L. Liu, S. Wang, B. Hu, Q. Qiong, J. Wen, and D. Rosenblum, "Learning structures of intervalbased bayesian networks in probabilistic generative model for human complex activity recognition," *Pattern Recognition*, vol. 81, p. 545–561, 2018.

[5] M. Verhagen, R. Gaizauskas, F. Schilder, M. Hepple, J. Moszkowicz, and J. Pustejovsky, "The tempeval challenge: Identifying temporal relations in text," *Language Resources and Evaluation*, vol. 43(2), p. 161–179, 2009.

[6] J. Allen and G. Ferguson, "Actions and events in interval temporal logic," *Journal of Logic and Computation*, vol. 4(5), p. 531–579, 1994.

[7] D. Dowty, *Word Meaning and Montague Grammar: The Semantics of Verbs and Times in Generative Semantics and in Montague's Ptq*. Studies in Linguistics and Philosophy, Springer Netherlands, 1979.

[8] J. R. Büchi, "Weak second-order arithmetic and finite automata," *Mathematical Logic Quarterly*, vol. 6, no. 1-6, pp. 66–92, 1960.

[9] C. C. Elgot, "Decision problems of finite automata design and related arithmetics," *Trans. Amer. Math. Soc.*, vol. 98, pp. 21–51, 1961.

[10] B. A. Trakhtenbrot, "Finite automata and logic of monadic predicates (in russian)," *Dokl. Akad. Nauk SSSR*, vol. 140, p. 326–329, 1961.

[11] T. Fernando, "Temporal representations with and without points," in *Logic and Algorithms in Computational Linguistics 2018 (LACompLing2018)*, (Cham), pp. 45–66, Springer International Publishing, 2020.

[12] P. Domingos and D. Lowd, *Markov Logic: An Interface Layer for Artificial Intelligence*, vol. 3. 06 2009.

[13] M. Mukund, "Regular languages: From automata to logic and back," 2011. Available from `https://www.cmi.ac.in/ madhavan/papers/pdf/regular-languages-2011.pdf`.

[14] T. Fiedor, L. Holik, P. Janků, O. Lengál, and T. Vojnar, "Lazy automata techniques for ws1s," pp. 407–425, Springer, Berlin, Heidelberg, 03 2017.

[15] Wikipedia contributors, "Zebra puzzle — Wikipedia, the free encyclopedia," 2020. [Online].

[16] S. Salavati, S. Hajjarzadeh, and M. Mazloom, "An optimized method for solving zebra puzzle," in *Second International Conference on Computer and Electrical Engineering*, pp. 448–451, 2009.

[17] M. Gregor, K. Zábovská, and V. Smataník, "The zebra puzzle and getting to know your tools," in *2015 IEEE 19th International Conference on Intelligent Engineering Systems (INES)*, pp. 159–164, 2015.

[18] Wikipedia contributors, "Tkinter," 2020. [Online].

[19] Wikipedia contributors, "Softmax function — Wikipedia, the free encyclopedia," 2020. [Online].

Appendix

The code for all the simulations described in the previous sections are available on github using the link - `https://github.com/shuchi03/MSO_And_Probability` .

There are four programs in the repository -

1. Zebra_Puzzle.ipynb: This program is to run the simulation for the Zebra puzzle.
2. Allen_transitivity_relations.ipynb: This program is to run the simulation for Allen relations and to see how to get transitivity relations.
3. Probabilities_using_Allen_transitivity_grid.ipynb: This program is to calculate conditional probabilities for Allen relations using the transitivity table.
4. Probability_with_markov_logic_network.ipynb: This program is to calculate probabilities for Allen relations according to their categories, weights and equation given in Markov Logic Network.

To run these programs, following steps should be followed -

- Upgrade python version to the latest one (3.8.5).
- Download dependencies using pip3 for -
  tkinter = 8.5
  numpy = 1.19.1
- Run the programs in jupyter-lab