S-words, MSO and Superposition

Shravani Deepak Kulkarni

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.

[redacted]

Shravani Deepak Kulkarni

September 6, 2020

Permission to Lend and/or Copy

Acknowledgments

I would like to thank my supervisor, Professor Tim Fernando, whose expertise was invaluable in formulating the research questions and designing and implementing the solutions. The constant support in understanding the difficult concepts in the dissertation along with the insightful feedback pushed me to sharpen my thinking.

Shravani Deepak Kulkarni

University of Dublin, Trinity College
September 2020

S-words, MSO and Superposition

Shravani Deepak Kulkarni, Master of Science in Computer Science

University of Dublin, Trinity College, 2020

Supervisor: Tim Fernando

The S-Languages proposed by Schwer provide a way of temporal representation and reasoning using S-words. Unlike other time interval and point algebra that uses transitivity tables (Eg. Allen relations) for reasoning, the S-words uses the operations like Intersections, Projections and Mixes. The S-words also use the Join operation which is used for solving problems on time constraints. This is similar to the superposition - that can be extended to solve constraint satisfaction problems like the Zebra puzzle and also finding transitivity in Allen relations. The S-words are understood using the Monadic Second Order Logic (MSO) and are also compared with the other available notations for representing a temporal object and the relative positioning with other instants in time.

Summary

Temporal reasoning is a crucial area of Artificial intelligence that finds usage in many Natural Language Processing tasks. It gives information about how one instant of time is related to others. In this paper, I aim to understand S-words in terms of Monadic Second Order Logic (MSO).

Previously, temporal reasoning has seen much research which includes the Allen relations, which provides the thirteen possible relationships between intervals. Allen also provides the transitivity table to give information about the relationship between two intervals i and k when the relation between i, j and j, k is known. This kind of reasoning is different from the one proposed by Schwer in S-Languages (SLS), which uses the mix and join operations. The S-Languages use the S-words as a notation to represent relations which use onset and end of a time interval to represent as a word. For instance, $[\{i\}\{i\}\{j\}\{j\}]$ means *i starts and ends, then, j starts and ends* for intervals i and j. Conversely, the interior and the bordered representation use empty boxes and the left, right border (li and ri) respectively. The mix operation, in SLS, provides a shorter way of representing the S-words and the join operation is very useful in finding a solution for given a list of temporal constraints (Eg. the Six Train Problem). On the other hand, the superposition theorem, which is the MSO conjunction, can be extended to solve the Constraint Satisfaction Problems (CSPs) like the Zebra Puzzle. Superposition works on the concept that given two MSO strings, it finds all other strings that satisfy the two strings.

For the implementation, I use the Python programming language along with the TkInter library for the graphical user interface (GUI). I implement the superposition theorem and utilize it to solve the CSP - Zebra Puzzle and also simulate it in Tkinter. For achieving this, I first design all the MSO formulas for constraints and then convert it to an MSO string and apply superposition. In this process, I also understand the S-words in terms of MSO formulas. Additionally, I use superposition to compute the transitivity in the Allen relations, which results in the non-Allen relations of Rank 0, 1 and 2. I show these relations clearly in the implementation of an interactive Allen transitivity grid. I also study the bordered notations for these ranked relations that provide the shortest way of representing it using the bordered notations as it separates the left and right borders. Lastly, I also implement the solution for the Six train problem using MSO formulas and superposition on bordered strings. I also compare it with the solution using the Mix and Joins in S-words. The bordered strings provide a clearer way to represent the strings as compared to the mix operations in S-words. The mix operation, although gives a compressed format but also brings in the additional complexity.

As future work, the MONA tool can be utilized to represent the MSO formulas obtained throughout the dissertation. The MONA tool, developed by BRICS take MSO formulas for a problem in the program and give satisfying solutions for it.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In artificial intelligence, reasoning of temporal information is crucial. These find its importance in many areas including Natural language processing, summarizing, organizing etc. Hence, there have been various algebra frameworks proposed by taking time as interval and as points. These involve transitive tables to represent relationships between the objects representing time (eg. Allen relations [1]).

In this dissertation, I aim to understand S-words which are proposed by Schwer as a way of representing temporal information 'without tables' [2]. The goal being that, given possibly uncertain and inadequate information of the relationships between some temporal objects, what are the possible relationship scenarios between the two objects. While all other works in the domain deal with reasoning this in terms of transitivity tables and binary relations, the S-words proposed by intersection, shuffling and projections of languages.

1.1 Objective

The main objective of this dissertation is to understand S-words in terms of MSO (Monadic Second-Order Logic). The S-words, explained by Schwer and Durand [3] which are used to represent temporal information and also reason between them.

The S-words can be understood by the regular languages and regular languages can be expressed in terms of MSO. This is stated and proved by the Bücci, Elgot and Trakhtenbrot [4] [5] [6]. So, we can say that

Regular languages $\approx$ MSO sentences

For achieving this objective, it is essential to understand the concept of MSO and also know how MSO formulas are written. For understanding this, I took the case study of the Zebra Puzzle. Zebra puzzle is a constraint satisfaction problem with a list of 15 constraints about 5 houses. These constraints are converted to MSO formulas and then converted to MSO strings and superposed with each other to solve the puzzle. This superposition of strings or the MSO conjunction is then implemented to solve the problem.

As the S-words provide a way of representing time intervals, it is also important to understand other notations available for representing temporal information. These include the Allen relations which give temporal reasoning based on transitivity table as a contrast to the S-words. Various other notations for representing time are also studied like the interior and bordered notations. The idea is to compare the S-words with other existing works.

Another case study undertaken is the Six train problem, which is used by [3] to explain the S-words along with the Mix and Join operations to solve the constraints of the problem and find the solution. This join operation is also compared with the superposition theorem.

1.2 Motivation

I was already aware of regular languages and finite automata and how it is used to model computational problems. For instance, finite automata can generate a regular language to describe if a number is even or not. And, knowing about the Monadic second order logic which is an addition to the first-order logic intrigued me to know more about it.

Initially, I went through the paper on temporal representation [7] and got to know about the representation of time along with the concept of the superposition. This motivated me to try to utilize the superposition in solving problems, especially constraint satisfaction problems. The zebra puzzle problem was a very challenging problem to solve manually as it involved a lot of steps. Thinking that it could be solved by MSO and superposition was a big motivation.

Reading about temporal representation led me to the Allen relations and the thirteen Allen relations and various ways to represent the onset and end of intervals in the form of strings. Here, the S-words came into the picture, as they provide a way of representing intervals and lead me to this dissertation. The way it contrasted with the transitive reasonings available was really interesting and I wanted to dive deeper into its understanding.

1.3 Reader's Guide

Going further, the dissertation is organized into four more sections: Previous Work, Design and Implementation, Results and Conclusion.

The Literature section goes through the various concepts and case studies taken up during the dissertation that includes First-order logic, MSO, Superposition, Allen relations, String representations for temporal information (Interior strings, bordered strings and S-words), Zebra puzzle and Six train problem.

In the implementation section, all the approaches and the implementations that are done, are highlighted. Python programming language is used for all the implementations of the programs and the simulations are done using Python's TkInter library. The section also discusses the MSO formulas for Zebra puzzle and the six train problems and how those are used to solve the problems. Additionally, it also presents an understanding of S-words in terms of MSO.

The Results section summarizes all the understandings and findings throughout the process. It covers the comparison of S-words with other notations and also differentiates between the Joins in S-words with the superposition. Lastly, the conclusion section concludes with information on all the work done as a part of this dissertation.

Chapter 2

Prior Work

In Artificial Intelligence (AI), representation of time is really important. Especially in the field of Natural Language Processing (NLP), temporal representation has applications by the extraction of text and utilizing it for summaries and question and answers. Consider a question and answer system, a question asking the occurrence of a certain event requires accurate information about the thing into consideration. For instance, the question *When did the storm occur?* would require complete information of the time of the storm. Other problem-solving questions also require understanding the change in time.

There have been various works in the past to tackle the representation of time. They include the work by Allen [1], Vilain and Kautz [8] which explain time in the form of intervals and points respectively. Additionally, there has also been research about chains of intervals [9], semi-intervals or convex set of intervals [10], chain of points and non-convex intervals [11]. In all of these frameworks proposed, the objective is to find the relationship between intervals given some piece of information. Similar is a method proposed by Schwer [2] which will be further discussed in this section.

As I aim to understand S-words in terms of Monadic Second-Order Logic (MSO), the prior work in this area is also studied. Starting with the regular languages which we can say is the base for the MSO, to MONA - which is a tool to represent the MSO formulas [12].

Other case studies of the Zebra puzzle and the Six train problem are also discussed in detail with the already existing solutions for them.

2.1 First Order Logic

Firstly, starting with the basics, First-order logic is used to represent Language for describing properties of mathematical structures. We can say that it is a logical system that can deal with objects, their properties, and relations between them. It typically has the following structure:

1. Domain: The set of natural numbers could be a domain.
2. Predicate: A predicate takes one or more arguments, and is either true or false. $even(x)$ and $odd(x)$ can be some examples of predicates.
3. Function: A function can take one or more arguments. For instance, for any numbers x and y, $sum(x, y)$ and $square(x)$ can be functions.
4. Relation: The relation is between the elements of the domain. For example *less than, equals* or *divides*.

A simple example of a first-order logic formula is given in 2.1

$$\forall u.P_a(v) \Rightarrow \exists v.S(u, v) \wedge P_b(v) \tag{2.1}$$

Here,
$P_a(u)$ stands for *a is at position u*
$S(u, v)$ stands for *v is next to position u*

This formula represents *'Every a is followed by b'*.

2.2 Monodic Second Order Logic (MSO)

Second-order logic talks about *for all properties* as compared to the first-order logic that only takes the *for all elements* into account. It is said to be stronger than the first-order logic [13] as more information can be expressed with it. In this, along with the first-order variables $x, y, z \cdots$ that range over natural numbers, propositional operations $(\wedge, \vee, \neg \cdots)$ and the existential and universal quantifiers $(\exists x, \forall x)$, we also have the monadic predicates $X_1, X_2, X_3 \cdots$ for representing the relationships between

these numbers. In other words, In MSO, the first-order logic is extended to incorporate the Set variables $X_1, X_2, X_3 \cdots$ along with quantifications over them.
For instance, the equation 2.2 is a simple MSO formula.

$$\forall x_1 \cdots \forall x_n (X_1(x_1 \cdots x_n) \iff X_2(x_1 \cdots x_n)) \tag{2.2}$$

It is used to represent the simple equation $X_1 = X_2$ but it also adds additional quantifiers that can be used to represent more information.
Another example of an MSO formula could be the one given in equation 2.3. The formula is for *x is at first position*. You can see that the monadic predicate X is used to specify more information in the formula. Although it makes the formula longer, it can express more details.

$$\begin{aligned} \exists X.\forall x.(first(x) \Rightarrow X(x)) \\ \wedge(\forall y.S(x,y) \wedge X(x) \Rightarrow \neg X(y)) \\ \wedge(\forall y.S(x,y) \wedge \neg X(x) \Rightarrow X(y)) \\ \wedge(last(x) \Rightarrow \neg first(x)) \end{aligned} \tag{2.3}$$

The Bücci, Elgot and Trakhtenbrot theorem [4] [5] [6] states that the MSO formulas can be expressed as regular languages. In other words, finite automata are expressively equivalent to specifications in MSO.

2.3 The MONA tool

The MONA tool, developed by the researchers at BRICS, University of Aarhus, converts the MSO formulas to finite automata [12]. The WS1S (Weak MSO theory of 1 Successor) that is talked about, incorporates the first-order variables as well as the second-order variables. This along with the WS2S can be used for problems on linguistics. Although previous research has posed problems related to memory constraints and time complexities in the conversion of formulas into automata, MONA provides an efficient method which is tried and tested in various settings. Few of the major applications of MONA are hardware verifications and computational linguistics.

A simple code snippet written in MONA is given below:

$$var2\ A, B;$$

$$A \backslash B = \{0, 1, 2, 4\};$$

The above code snippet means that the set difference between A and B is $\{0, 1, 2, 4\}$, where A and B are second-order variables. The execution of this MONA program gives a counterexample and a satisfying example of the MSO formulas represented in the program. Also, the program can be extended to give DFA style automation.

The code is given in Listing 2.1 below is for even numbers and the output for this is given in Figure 2.1. The MSO formula for an even number is given below and is also represented in the MONA code. The program states that $U - V = \{0, 1, 2, 4\}$ where U and V are second-order variables. x is a first-order variable and A is a boolean variable.

$$\begin{aligned} \exists Q : x \in Q \wedge (\forall q : (0 < q <= x) \Rightarrow (q \in Q \Rightarrow q - 1 \notin Q) \\ \wedge (q \notin Q \Rightarrow q - 1 \in Q) \wedge 0 \in Q) \end{aligned}$$

Listing 2.1: Mona program for even.mona

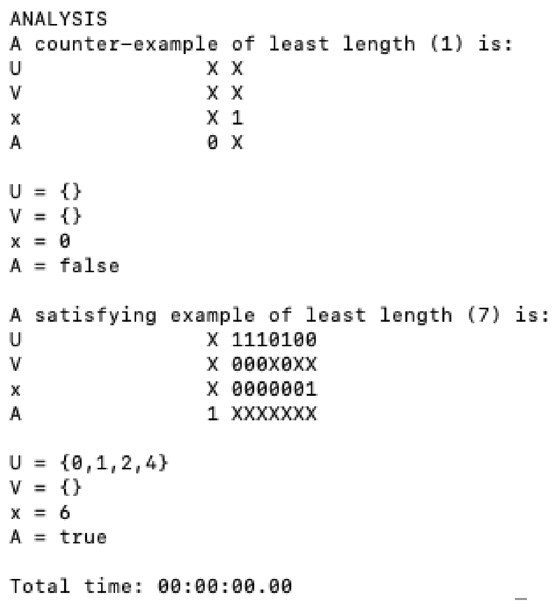
```
var2 U,V;
U\V = {0,1,2,4};

var1 x;
var0 A;

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

A & x notin U;
```

As can be seen from the output, the least number which satisfies the MSO formula



```
ANALYSIS
A counter-example of least length (1) is:
U               X X
V               X X
x               X 1
A               0 X

U = {}
V = {}
x = 0
A = false

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

U = {0,1,2,4}
V = {}
x = 6
A = true

Total time: 00:00:00.00
```

Figure 2.1: The output for even.mona

and the least number that does not satisfy the formula. As $U - V = \{0, 1, 2, 4\}$, we know that U can be $\{3, 5, 6 \cdots\}$ and the least even number in V is 6.

2.4 MSO Conjunction/Superposition

After understanding the MSO, understanding of the superposition is crucial. We can say that the superposition is MSO conjunction.

$$\textit{Superposition} \approx \textit{MSO Conjunction}$$

When two MSO formulas are converted to MSO strings, the MSO conjunction between these strings becomes the superposition of these strings. In other words, superposition of two strings generates strings that satisfy both the strings.

Now the question comes to mind that what are MSO strings. The MSO formulas can be used to generate strings. For example, consider the statement *Alice finishes work before Mary*, then the strings can be any of the following because we do not know when *Alice* and $Mary$ start eating. Here the subscript s stands for starts and f stands for finishes.

1. $Alice_s \Rightarrow Mary_s \Rightarrow Alice_f \Rightarrow Mary_f$
2. $Alice_s, Mary_s \Rightarrow Alice_f \Rightarrow Mary_f$
3. $Mary_s \Rightarrow Alice_s \Rightarrow Alice_f \Rightarrow Mary_f$

As mentioned in [7], the basic form of superposition is obtained by component-wise union $\&_\circ$ of two strings $\alpha_1 \cdots \alpha_k$ and $\alpha'_1 \cdots \alpha'_k$ when both the strings are of same length k.

$$(\alpha_1 \cdots \alpha_k)\&_\circ(\alpha'_1 \cdots \alpha'_k) := (\alpha_1 \cup \alpha'_1) \cdots (\alpha_k \cup \alpha'_k) \tag{2.4}$$

Superposition becomes different when the projections are not directly the reducts or when the output projects to inputs.
we use inductive rules such as s0 and s1 as follows –

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

s, s', s' are strings whose superposition is computed. Using s0 and s1, we can generate $\&_\circ$ as follows -

$$s\&_\circ s' \iff \&(s, s', s'')$$

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 –

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

This concept of superposition theorem has its usage in solving constraint satisfaction problems. For instance, consider two constraints c_1 and c_2, if we find the MSO strings for these constraints and then find the MSO conjunction of those to get the solution for the problem. To see how this can work, the Zebra Puzzle is solved using superposition and the details are given in the implementation section. The description of the puzzle is given in the next section.

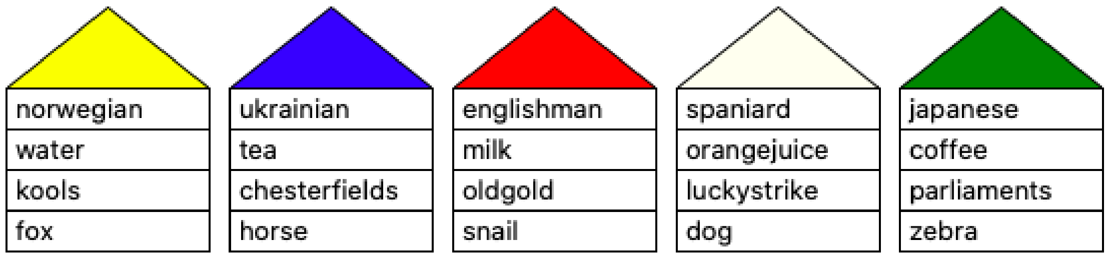


Figure 2.2: The Zebra Puzzle

2.5 The Zebra Puzzle

The Zebra Puzzle or Einstein's puzzle is a classic CSP (Constraint Satisfaction Problem) with a set of 15 constraints about 5 houses. There are various versions of the problem available over the web. We consider the one mentioned in [14], which has the following constraints.

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.

The problem is solved such that it arrives at a satisfactory answer to the question: *Who owns the Zebra?* One of the solutions is given in Figure 2.2.
Along with the constraints, additional information related to the attributes are also provided:

- Each of the five houses is painted a different colour and one of (Red, Green, Blue, Yellow or Ivory)

- The inhabitants are of different national extractions and one of (Englishman, Spaniard, Japanese, Ukrainian or Norwegian)

- The inhabitants own different pets and one of (Dog, Horse, Snails, Fox or Zebra)

- The inhabitants drink different beverages and one of (Kools, Old Gold, Parliament, Lucky Strike or Chesterfield)

- The inhabitants smoke different brands of cigarettes and one of (Water, Tea, Orange Juice, Milk or Coffee)

There are many ways to solve these CSPs. One of them is provided in [15] where the authors solve it using Prolog. In these problems, the order in which the constraints are satisfied is not important, so a simple reordering of constraints can result in reaching the solution faster. In [15], this reordering can be done by introducing certain *heuristics*. A heuristic is a measure of how close a state in the process is from the solution. In their implementation, the choose the constraints that give complete information, for example, Constraint 9 *(Milk is drunk in the middle house)* and 10 *(The Norwegian lives in the first house)*. On the other hand, other constraints result in more number of possibilities.

Another solution is also provided in [16] which provides an optimized way of solving the puzzle using backtracking and MRV (Minimum remaining value) along with minimum conflicts. This is done by choosing the best variable at each time. For achieving the best variable, by placing the attribute information about the same house together. For example, for the constraint - *Kools are smoked in the yellow house*: if Kools is associated with House1 ($Kools = S_1$) then Yellow is also associated with House1 ($Yellow = C_1$). Here S stands for Smoke and C stands for colour. This means that $S_1 = C_1$ is also added to the constraint list.

The details of the solution and the approach using superposition are provided in the implementation section.

2.6 Representation of temporal information

As understanding S-words was one of the main ideas of this dissertation, and S-words were a way of representing temporal information, I also looked into some of the other representations. The idea was to compare the S-words with the other notations and understand when it can be advantageous and when not.

As mentioned, there had been a lot of prior work concerning the representation and reasoning of temporal information which includes the work by Allen [17] on time intervals, by Vilain and Kautz [8] on time as points. The Allen relations are discussed in the next section.

2.6.1 Allen Relations

The paper by Allen [1] provides a method for representing temporal information using reference intervals. Allen designed these in such a way that it handles relative temporal knowledge as most of the temporal knowledge is relative.

Time can be represented as either points or intervals. For example, 'at noon' is a point and 'yesterday' is an interval. Allen describes the relations between times (point or interval) in thirteen different ways. These relations along with their symbols and examples are given in figure 2.3. In the figure, i and j are intervals and the thirteen relations possible between them are given.

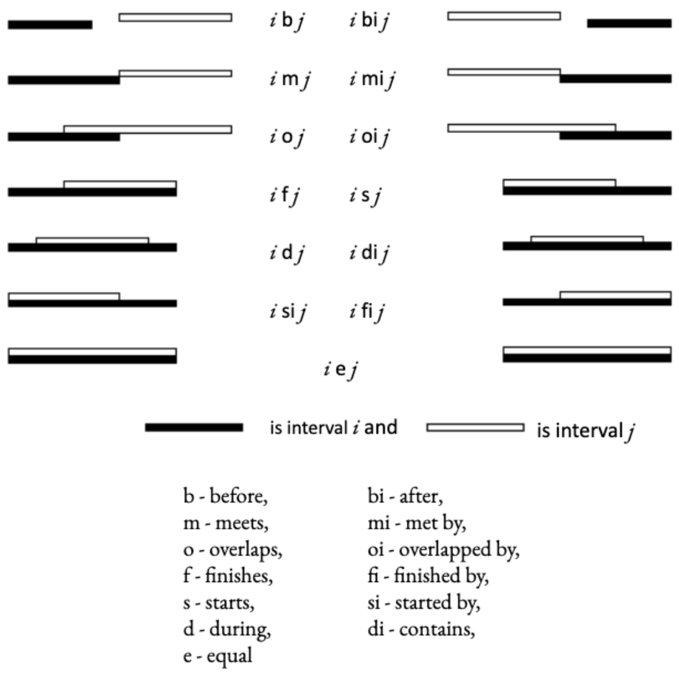


Figure 2.3: The 13 Allen Relations

2.6.2 Transitivity of Allen Relations

Allen also provided a transitivity table for these relations. Given a relation r1 between i and j and another relation r2 between j and k, the transitivity table gives all the possible relations between i and k. For instance, if i is before j and j is during k, i.e. i b j and j d k, then i b,d,m,o,s k. This means, five relations can exist between i and k (before, during, meets, overlaps and starts).

2.6.3 S-Languages

Schwer [2] introduces the concept of *S-Languages* which provides a mechanism of representing temporal information in the form of *S-words*. The aim was to propose a lighter, uniform and more intuitive representation.

As mentioned earlier, the Allen relations transitivity and binary relations to explain the temporal reasoning. However, Schwer proposed the *S-Languages* which use the intersection, projections and shuffling for the temporal reasoning. According to Schwer, using an alphabet to represent an interval and using as many occurrences based on if it is a point or interval, it is useful to describe the relationship between the other relations in the timeline. This is achieved without any transitivity tables in the case of S-Languages.

Any object in time can be thought of as event-like, lasting or repeating. These objects are expressed using S-letters that include alphabetical letters. The S-letters are combined to obtain the S-words and a set of S-words make the S-language. In the following example, w is an S-word of length 3 and L is S-language over two temporal objects i and j.

$$w = [\{j\}\{i\}\{j\}]$$

$$L = \{[\{j\}\{i\}\{i\}\{j\}], [\{i\}\{i\}\{j\}\{j\}], [\{i\}\{j\}\{i\}\{j\}]\}$$

if i is event-like, there is only one occurrence of it in the word. If it is lasting, two occurrences of it are required, one for the beginning and other for the end. For instance, in w, j is lasting and i is event-like. If it is lasting and iterating n times, there will be $2n$ occurrences of that object. L above is S-language with three s-words that describe - *i ends before j ends.*

When a temporal object is preceded by another temporal object in the S-word, it shows precedence. In $[\{j\}\{i\}\{j\}]$, $j < i < j$ and for the case of $[\{i, j\}]$, $i < i = j < j$ which means that i and j occur simultaneously. α called the alphabet is used to denote the set of temporal objects over which the S-words are made. For w mentioned above, $\alpha = \{i, j\}$

The S-words for all the Allen relations is provided in the table 2.4.

2.6.4 Interior and Border Notations

The previous section discussed the algebras for temporal representation and reasoning. The S-words also provide the representations for intervals. There are also other ways to depict the intervals and relationship between the intervals. Two of these are the interior and bordered representation. These notations can incorporate many intervals in them, unlike the Allen relations which mention about relationship between two intervals.

Interior

The interior notations make use of empty boxes. Consider a case where an interval i overlaps another interval j. This is represented as an interior string using the following notation (2.5) of length 5 [7]. Each box in the notation represents an instant of the total time.

$$\begin{array}{|c|c|c|c|c|}\hline & i & i,j & j & \\ \hline\end{array} \tag{2.5}$$

- $\boxed{}$ in the beginning, denotes the time before i.
- $\boxed{i}$ means the time when i is present but j has not started.
- The third box denotes the overlap of i and j.
- $\boxed{j}$ means the time when j is present but i has ended.
- $\boxed{}$ in the end, means the time after j ended.

This can also be denoted as s = [{}, {i}, {i,j}, {j}, {}].
If another interval k is to be added in this which overlaps a and b, s translates to s' = [{}, {k}, {i,k}, {i,j,k}, {i,j}, {j}, {}] of length 7.

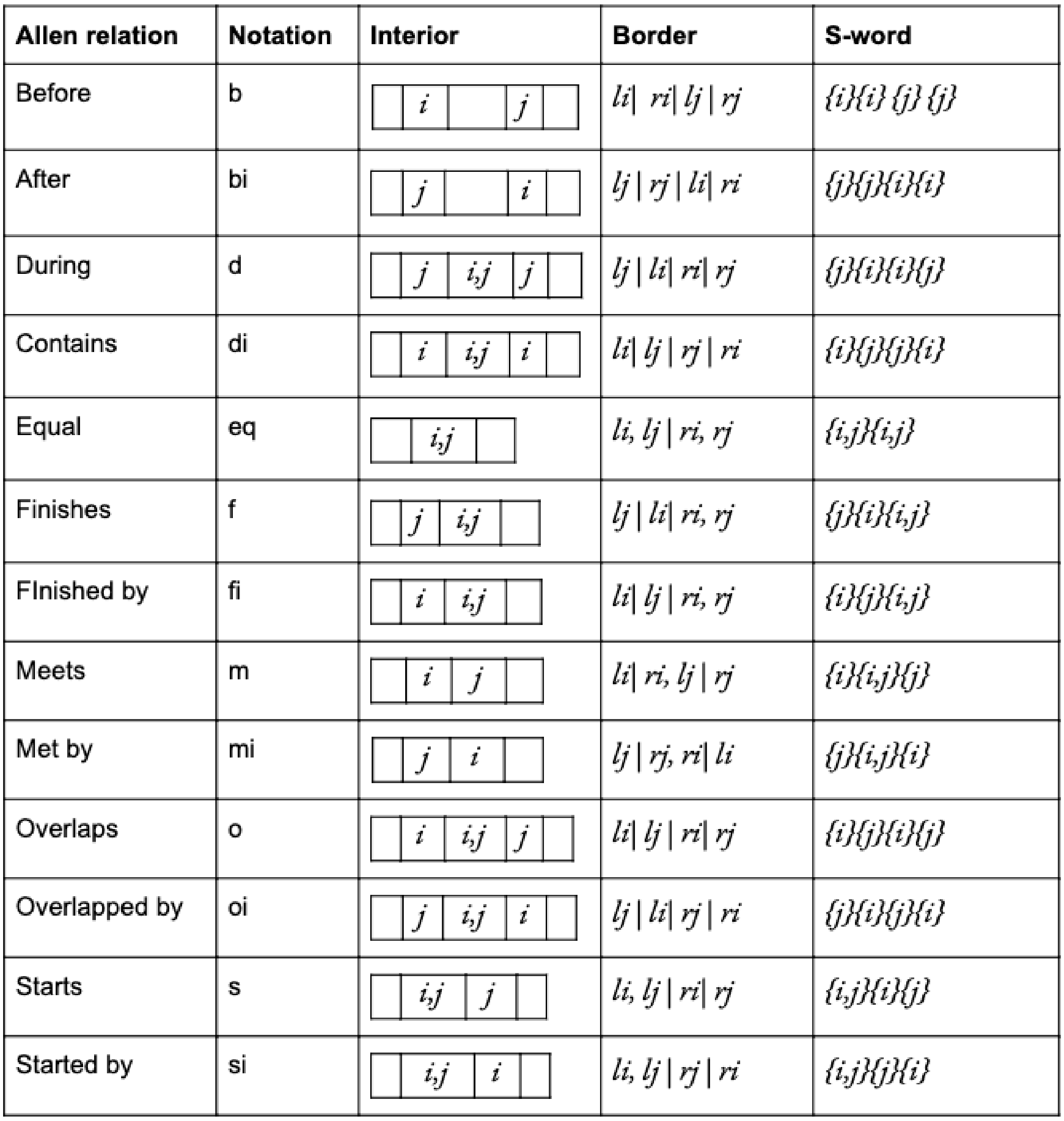
$$\begin{array}{|c|c|c|c|c|c|c|}\hline & k & i,k & i,j,k & i,j & j & \\ \hline\end{array} \tag{2.6}$$

Let us denote this string as s' = $(\alpha_1 \cdots \alpha_5)$
Let A be the set of intervals, then the A-reduct of s' is component-wise intersection of s with A. Therefore, {i,j}-reduct of s' is [{}, {}, {i}, {i,j}, {i,j}, {j}, {}]

$$\begin{array}{|c|c|c|c|c|c|c|}\hline & & i & i,j & i,j & j & \\ \hline\end{array} \tag{2.7}$$

This notation (2.7) can be compressed to obtain 2.5. This is known as block-compression (bc). For a given string $\alpha_1 \cdots \alpha_5$ remove every α_t such that $t < n$ and $\alpha_t = \alpha_{t+1}$, where n is the length of string. The block compression results into stutterless strings. A string is called as stutterless when there is no t such that $\alpha_t = \alpha_{t+1}$.

The interior representation of all the Allen relations is given in the table 2.4.



Allen relation	Notation	Interior	Border	S-word
Before	b	$\boxed{\ }\boxed{i}\boxed{\quad}\boxed{j}\boxed{\ }$	$li \mid ri \mid lj \mid rj$	$\{i\}\{i\}\{j\}\{j\}$
After	bi	$\boxed{\ }\boxed{j}\boxed{\quad}\boxed{i}\boxed{\ }$	$lj \mid rj \mid li \mid ri$	$\{j\}\{j\}\{i\}\{i\}$
During	d	$\boxed{\ }\boxed{j}\boxed{i,j}\boxed{j}\boxed{\ }$	$lj \mid li \mid ri \mid rj$	$\{j\}\{i\}\{i\}\{j\}$
Contains	di	$\boxed{\ }\boxed{i}\boxed{i,j}\boxed{i}\boxed{\ }$	$li \mid lj \mid rj \mid ri$	$\{i\}\{j\}\{j\}\{i\}$
Equal	eq	$\boxed{\ }\boxed{i,j}\boxed{\ }$	$li, lj \mid ri, rj$	$\{i,j\}\{i,j\}$
Finishes	f	$\boxed{\ }\boxed{j}\boxed{i,j}\boxed{\ }$	$lj \mid li \mid ri, rj$	$\{j\}\{i\}\{i,j\}$
FInished by	fi	$\boxed{\ }\boxed{i}\boxed{i,j}\boxed{\ }$	$li \mid lj \mid ri, rj$	$\{i\}\{j\}\{i,j\}$
Meets	m	$\boxed{\ }\boxed{i}\boxed{j}\boxed{\ }$	$li \mid ri, lj \mid rj$	$\{i\}\{i,j\}\{j\}$
Met by	mi	$\boxed{\ }\boxed{j}\boxed{i}\boxed{\ }$	$lj \mid rj, ri \mid li$	$\{j\}\{i,j\}\{i\}$
Overlaps	o	$\boxed{\ }\boxed{i}\boxed{i,j}\boxed{j}\boxed{\ }$	$li \mid lj \mid ri \mid rj$	$\{i\}\{j\}\{i\}\{j\}$
Overlapped by	oi	$\boxed{\ }\boxed{j}\boxed{i,j}\boxed{i}\boxed{\ }$	$lj \mid li \mid rj \mid ri$	$\{j\}\{i\}\{j\}\{i\}$
Starts	s	$\boxed{\ }\boxed{i,j}\boxed{j}\boxed{\ }$	$li, lj \mid ri \mid rj$	$\{i,j\}\{i\}\{j\}$
Started by	si	$\boxed{\ }\boxed{i,j}\boxed{i}\boxed{\ }$	$li, lj \mid rj \mid ri$	$\{i,j\}\{j\}\{i\}$

Figure 2.4: The interior, border and S-words notations for Allen relations

Bordered

The interior kind of notations does not take into account the recurring nature of the intervals a,b and c. These can be represented using bordered strings [18]. In a bordered string, each interval i is associated with a left border li and a right border ri. Adding the border gives more information about the string.

For the case where one interval a meets another interval b, it is written in bordered strings as s = [$\{li\}$, $\{lj\}$, $\{ri\}$, $\{rj\}$] and is represented as

$$\begin{array}{|l|l|l|l|}\hline li & lj & ri & rj \\ \hline\end{array} \tag{2.8}$$

This means that the order of occurrence is $li < (ri = lj) < rj$.
The border representation for Allen relations is given in the table 2.4. The 13 relations can be represented using strings of length between 2 and 4.
Another interval c when overlaps 2.9 gives a smaller representation than 2.6 of length 5 written as s' = [$\{lk\}$, $\{li\}$, $\{lj,rk\}$, $\{ri\}$, $\{rj\}$].

$$\begin{array}{|l|l|l|l|l|}\hline lk & li & lj,rk & ri & rj \\ \hline\end{array} \tag{2.9}$$

2.8 can be obtained when $\{i,\ j\}$-Reduct is obtained. In the case of bordered strings, the reduct is obtained by component-intersection of the string with the left and right borders of i and j i.e. $\{li,\ ri,\ lj,\ rj\}$ which results in 2.10. After this, □-removal is done to obtain the stutterless string.

$$\begin{array}{|l|l|l|l|l|}\hline & li & lj & ri & rj \\ \hline\end{array} \tag{2.10}$$

Translations

The interior and bordered strings can be translated to one another as per propositions mentioned in [7].

$$\begin{array}{|l|l|l|l|l|}\hline l0 & l2 & l1 & r0,r1 & r2 \\ \hline\end{array} \tag{2.11}$$

$$\Downarrow$$

$$\begin{array}{|l|l|l|l|l|}\hline & 0 & 0,2 & 0,1,2 & 2 \\ \hline\end{array} \tag{2.12}$$

Consider the bordered string 2.11. If we need to convert this string into the corresponding interior notation, the following steps are followed:

1. The first occurrence is a left border so an empty box is added in the beginning. As we know that the interior representation uses empty boxes for border representation.

2. The second box contains $l2$, but even this is not added in the box. As interval 0 has already started, the second box contains 0.

3. The third box in 2.11 contains the $l1$ but interval 2 has already started, so the box contains $0, 2$

4. The second last box contains right borders for intervals $r0, r1$ So the translation contains $0, 1, 2$ (2 is added because it has not ended yet).

5. The last one in the translation just contains the interval 2.

$$\boxed{4}\boxed{2}\boxed{1,2}\boxed{0,2}\boxed{0} \tag{2.13}$$

$$\Downarrow$$

$$\boxed{r4,l2}\boxed{l1}\boxed{r1,l0}\boxed{r2}\boxed{r0} \tag{2.14}$$

Similarly, the translations can also be applied from interior to border. The example for the same is given in equations 2.13 and 2.14.

2.7 Operations on S-Languages

Projection

The *S-projection* is similar to the *A-reduct* operation discussed in the previous section. The S-projection of w over a set of alphabet α is obtained by removing all the occurrences of the temporal objects in α from w.
For example, Let $w = [\{i, k\}\{i, j, k\}, \{j\}]$ and $\alpha = \{i, j\}$. Then, the projection $w_{|\alpha} = [\{i\}\{i, j\}, \{j\}]$

Mix operation

The *mix* operation on s-words or the shuffle was introduced in the paper [2] to giver s shorter representation of the S-words. This operation is denoted by X which considers all possible ordering of the S-words on which it is applied. For instance consider two s-words $L_1 = [\{i\}\{i\}]$ and $L_2 = [\{j\}\{j\}]$. The mix operation, also called as the S-expression of these is denoted by $[\{i\}\{i\}]$ X $[\{j\}\{j\}]$.
Let u and v be two words written on an alphabet $A*$.
The shuffle of u and v is the language,
$u \,\mathrm{X}\, v = \{\alpha_1\beta_1 \cdots \alpha_k\beta_k \in A*|\alpha_1, \beta_k \in A*, \alpha_2 \cdots \alpha_k, \beta_1 \cdots \beta_{k-1} \in A+, u = \alpha_1 \cdots \alpha_k, v = \beta_1 \cdots \beta_k\}$.
For instance let $A = \{x, y\}$, then
xx X $yy = \{xxyy, xyxy, yxxy, xyyx, yxyx, yyxx\}$.
This mix operation is expanded to obtain all the possible s-words by applying these two re-write rules until no new strings are obtained:

- Replace all $\{i\}\{j\}$ with $\{i, j\}$
- Replace all $\{i, j\}$ with $\{j\}\{i\}$

This results in the 13 Allen relations for the mix $[\{i\}\{i\}]$ X $[\{j\}\{j\}]$ as shown in Figure 2.5.

Join operation

The join operation [3] is similar to the concept of superposition discussed above. For two s-words S_1 and S_2 with certain temporal constraints, the joining of these two strings denoted by $S_1 \mathrm{J} S_2$ will contain s-words that satisfy the temporal constraints of both S_1 and S_2.

The join operation has two cases:

1. When $\alpha(S_1) \cap \alpha(S_2) = \phi$, the join operation is the same as the mix, i.e.

$$S_1 \mathrm{J} S_2 = S_1 \mathrm{X} S_2$$

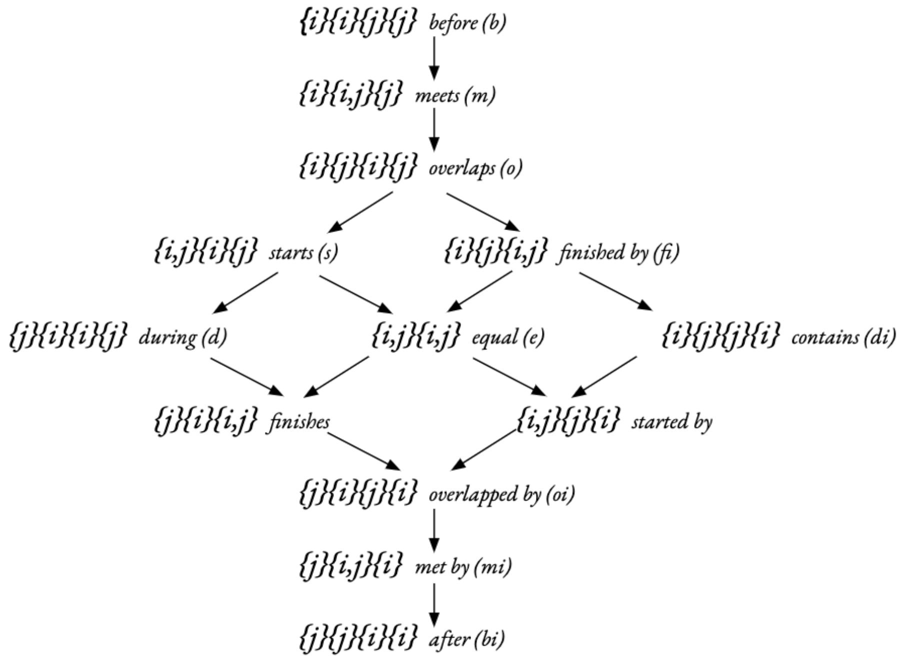


Figure 2.5: The mix operation in S-words

2. When $\alpha(S_1) \cap \alpha(S_2) \neq \phi$, we first find out if the two strings are compatible. Let $\beta = \alpha(S_1) \cap \alpha(S_2)$. The two strings are said to be compatible if their S-projections with respect to β are equal i.e. $S_{1|\beta} = S_{2|\beta}$. If they are not compatible, $S_1 \mathrm{J} S_2 = \{\}$

   If they are compatible, All the s-words h from $\alpha(S_1) \cup \alpha(S_2)$ which satisfy $S_{1|\alpha(S_1)} = S_1$ and $S_{2|\alpha(S_2)} = S_2$

$$S_1 = [\{i,k\}\{i,j\}\{k,m\}\{i,j,k\}]$$

$$S_2 = [\{n\}\{i,n,o\}\{n\}\{i,j\}\{o\}\{i,j,o\}\{n\}]$$

$$S_1 J S_2 = \{[\{n\}\{i,k,n,o\}\{n\}\{i,j\}\{k,m,o\}\{i,k,j,o\}\{n\}],$$

$$[\{n\}\{i,k,n,o\}\{n\}\{i,j\}\{k,m\}\{o\}\{i,k,j,o\}\{n\}],$$

$$[\{n\}\{i,k,n,o\}\{n\}\{i,j\}\{o\}\{k,m\}\{i,k,j,o\}\{n\}]\}$$

This can be written in short using the mix operation:

$$[\{n\}\{i,k,n,o\}\{n\}\{i,j\}] \cdot ([\{k,m\}] \text{ X } [\{o\}]) \cdot [\{i,j,k,o\}\{n\}]$$

The *Join* operation can be extended from s-words to s-languages. The join of two S-languages is the union of s-words from the first language to s-words from the second language.

The algorithm used for the joins in S-words is provided in [19]. However, this paper was only available in French and I could not go through the details of the implementations. Hence, throughout the paper, only the concepts are compared.

2.8 The Six Train Problem

The Train problem as mentioned in [3] is a constraint satisfaction problem that takes the time into consideration. In [3], the authors solve this problem using the S-words and the mix and join operations mentioned in the previous section.

The details of the problem are given as follows: Consider a set of 6 trains named A, B, C, D, E, F with the following set of temporal constraints.

1. A, B and E reach the platform at the same time
2. A leaves before B.
3. A leaves after or at the same time as C but before the arrival of D.
4. D and F arrive at the same time as B is leaving.
5. E and D leave at the same time.

The problem is to find the minimum number of tracks required at the station. Because of security reasons, we do not allow that a train to arrive on a track from which a train is currently leaving.

In the paper [3], the authors solve this problem using the the *Join* operation. All the constraints are converted into s-expressions. a, b, c, d, e, and f denote the six trains A,B,C,D,E,F.

1. $E1 = [\{a, b, e\}] \cdot ([\{a\}] \text{ X } [\{b\}] \text{ X } [\{e\}])$

2. $E2 = ([\{a\}] \text{ X } [\{b\}]) \cdot [\{a\}\{b\}]$
3. $E3 = (([\{a\}] \text{ X } [\{c\}\{c\}]) \cdot [\{a\}\{d\}\{d\}]) \cup (([\{a\}] \text{ X } [\{c\}]) \cdot [\{a,c\}\{d\}\{d\}])$
4. $E4 = [\{b\}\{b,d,f\}] \cdot ([\{f\}] \text{ X } [\{d\}])$
5. $E5 = ([\{e\}] \text{ X } [\{d\}]) \cdot [\{d,e\}]$

Here, $\alpha = \{a,b,c,d,e,f\}$. And all the possible s-words are denoted by the following expression

$$[\{a\}\{a\}] \text{ X } [\{b\}\{b\}] \text{ X } [\{c\}\{c\}] \text{ X } [\{d\}\{d\}] \text{ X } [\{e\}\{e\}]X[\{f\}\{f\}]$$

To solve all the constraints, the join of all the expressions is done: $E = E1$ J $E2$ J $E3$ J $E4$ J $E5$ which gives the final S-Expression as (according to the algorithm by [3]). This is done by a tool that is developed by Schwer and Durand called the SLS.

$$\begin{gathered} E = ([\{c\}\{c\}]X[\{a,b,e\}]) \cdot [\{a\}] \cdot [\{b,d,f\}] \cdot ([\{f\}]X[\{e,d\}]) \cup \\ ([\{a,b,e\}]X[\{c\}]) \cdot [\{a,c\}] \cdot [\{b,d,f\}] \cdot ([\{f\}]X[\{e,d\}]) \end{gathered} \tag{2.15}$$

Out of the expression E, it is broken down to $[\{c\}\{c\}\{a,b,e\}\{a\}\{b,d,f\}] \cdot ([\{f\}] \text{ X } [\{e,d\}])$. This means that, C arrives and then goes back (1 track), the A, B and E arrive (2 more tracks), then A leaves, B leaves and D, F arrive (1 more track). Hence, minimum of 4 tracks are required.

A solution of the Train problem using superposition theorem is provided in the implementation section.

Chapter 3

Design and Implementation

As the theory for the superposition was already available, the important part was implementing it in Python. This was done using the intersections, unions and other set operations available in Python. This superposition function was then utilized to design and implement python programs with GUI for the following:

1. The simulation of the Zebra puzzle.
2. The computations of the Allen transitivity relations.
3. An interactive Allen transitivity table along with rank0, rank1 and rank2 relations along with finding the shorter string representations for them.
4. The solution to the Six train problem.

Along with this, The MSO formulas were also designed along with the presuppositions into consideration for executing the superposition on them. The details of all the design and implementations are given in this section.

3.1 Implementation of Superposition Theorem

The implementation of the Superposition theorem that is discussed in the previous section, was the most important part. Although the theory related to that was already available the actual implementation took a lot if understanding and testing. I decided to go with python function for the implementation.

The first task was to determine what will be the input to the function and what will be returned. As mentioned, the superposition will be working with MSO strings, there would be two MSO strings as input and an array of superposed strings would be returned that satisfies the given two strings.

The algorithm that was designed is given in the Algorithm 1. Here, superpose(s1, s2, voc1, voc2) is a recursive function where s1 and s2 are two input strings and voc1 and voc2 are the vocabularies of s1 and s2 respectively. A vocabulary of a string is the list of all the contents in the strings. For instance, the vocabulary of {color: red, shape: square} will be [colour, red, shape, square].

The algorithm also makes use of set operations like union and intersection. At each step in the superposition, the string is divided into head and then the remaining and superposition is applied to each part recursively. The algorithm is given in Algorithm 1.

3.2 Solving the Zebra Puzzle

3.2.1 MSO Formula

The Zebra Puzzle as discussed in the previous section, was solved using the superposition theorem. As a first step, all the constraints for the Zebra puzzle were analyzed and were converted to their corresponding MSO formula as given below:

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.
   $\exists x(ukrainian(x) \wedge tea(x))$

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
```

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.
    $\exists x(norwegian(x) \wedge \exists x_2 \exists x_3 \exists x_4 \exists x_5(xSx_1 \wedge x_2 S x_3 \wedge x_3 S y_4 \wedge x_4 S x_5))$

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 stands for 'x is to the left of y'.

3.2.2 Approach

Using the MSO formulas given above, the constraints were then represented into a set with five attributes which are shown as given below. The string notations here are the interior string notations as discussed in the literature review. For a smooth superposition, the strings are padded to make them of equal length.

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 -

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

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 given below -

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

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

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

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.

A variant of superposition theorem (Proposition 4 in [7]) is used to solve the zebra puzzle given as:

Given:
attributes Att := {color, nation, drink, smoke, pet}
domains D(color) := {red, blue, green, yellow, ivory}

$\vdots$

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:

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

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' is maintained. 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 -

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

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

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, 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 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 $(s, \square)$.

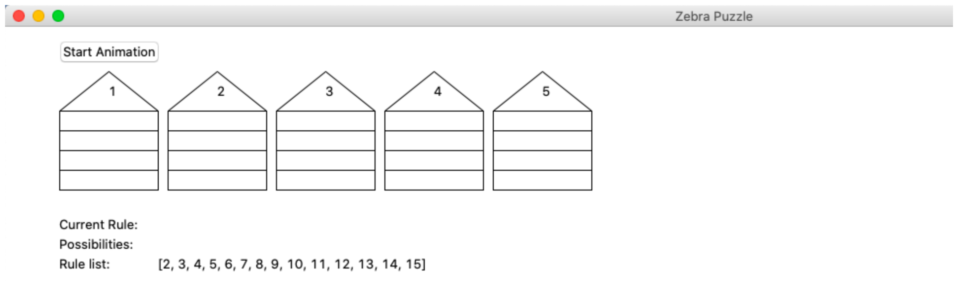
3.2.3 Simulation using Python TkInter

The simulation of the implementation of the solution of the Zebra Puzzle was done using Python's TkInter library [20]. It is a graphical user interface library available in python that allows users to render elements like shapes, text boxes, buttons etc.

For the implementation, as the superposition function required the strings, each string was represented using a python dictionary and it was denoted as given below:

Listing 3.1: String representation for the zebra puzzle

```
{
```



Current Rule:
Possibilities:
Rule list: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

Figure 3.1: The Zebra Puzzle - Initial state

```
  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':''}
}
```

As can be seen, the string contains five keys labelled: 1,2,3,4,5 which represent the five houses. Each of the houses has five attributes labelled with the keys: 'color', 'nationality', 'drink', 'smoke' and 'animal'. Initially, all the attribute values are empty as no constraints are satisfied.

Listing 3.2: List of rules with the information

```
{
  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'}],
}
```

The following approach is followed to solve the given problem using TkInter.

1. As a first step, all the rules are converted to smaller dictionaries with the information that they give. This is given in Listing 3.2. There are 3 things determined by manually looking at the rules:
   The house number it is describing (if available, otherwise 0),
   If the rule is reversible or non-reversible (A rule is reversible if they give information like *next to house, besides this house* etc.), and
   The information that the rule gives.

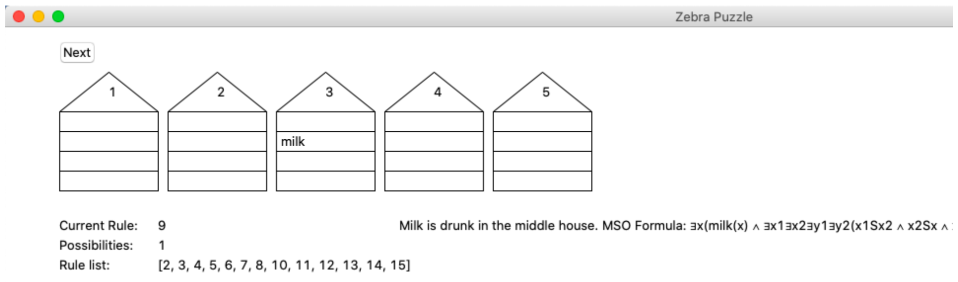
2. A rule list is maintained consisting of all the rules. Rule 1 is not taken in this list because *'There are five houses'* is already taken into consideration in the already created string mentioned in Listing 3.1. The initial state is as shown in Figure 3.1. Here you can see the rule list and a graphical representation of 5 houses.

3. There is a *Start Animation* button provided which takes one rule at a time from the list of rules and then computes the string for that rule. For instance, Rule 9 states *'Milk is drunk in the middle house'*. The rule list is used here, to compute the string. For Rule 9 the string then becomes as given in Listing 3.3. Here, *'drink' :' milk'* can be seen in house 3.

4. Rule 9 and Rule 10 gives only one possibility of the string as compared to the others which give more than one possibilities. Hence, one of these rules is chosen for the next step.

5. The obtained string is superposed with the current state string to give the possible strings that satisfy both the strings. This is done via the superposition function which is implemented. This superposed string is then represented in the TkInter GUI as shown in Figure 3.2.

6. The next button is provided in the UI for continuing the simulation. Whenever more than one possibilities are obtained, a *Show other possibility* button is also provided as shown in Figure 3.3.



```
Current Rule:    9                    Milk is drunk in the middle house. MSO Formula: ∃x(milk(x) ∧ ∃x1∃x2∃y1∃y2(x1Sx2 ∧ x2Sx ∧
Possibilities:   1
Rule list:       [2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15]
```

Figure 3.2: The Zebra Puzzle - first step with Rule 9

7. When at a stage, no further solutions can be obtained, the program backtracks to another possibility. This is shown in Figure 3.4.

8. The final solution is shown in Figure 3.5.

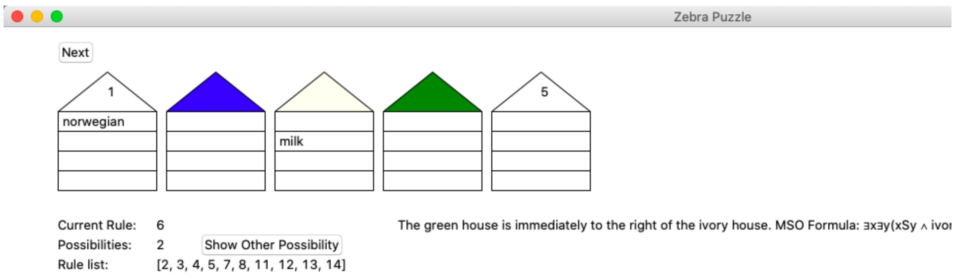
Listing 3.3: String representation for Rule 9 in the zebra puzzle

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

3.3 Allen Transitivity Table

The Allen transitivity table is a table which gives transitivity information about the Allen relations. The superposition theorem can be used in finding the transitivity relations as well. If the Allen relations r_1 and r_2 are converted to their corresponding MSO strings and superposition is done on them, we will get strings that satisfy r_1 and r_2.

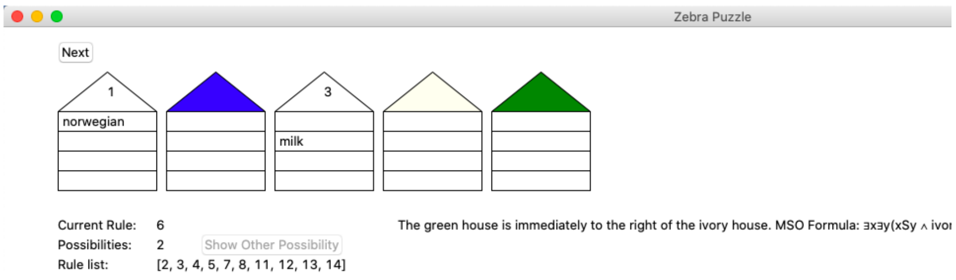
So, as a first step, it was important to convert the Allen relations into the MSO formulas. The approach is taken to design the formulas are given in the next sections



Current Rule: 6
Possibilities: 2 Show Other Possibility
Rule list: [2, 3, 4, 5, 7, 8, 11, 12, 13, 14]

The green house is immediately to the right of the ivory house. MSO Formula: ∃x∃y(xSy ∧ ivor

(a) Possibility 1

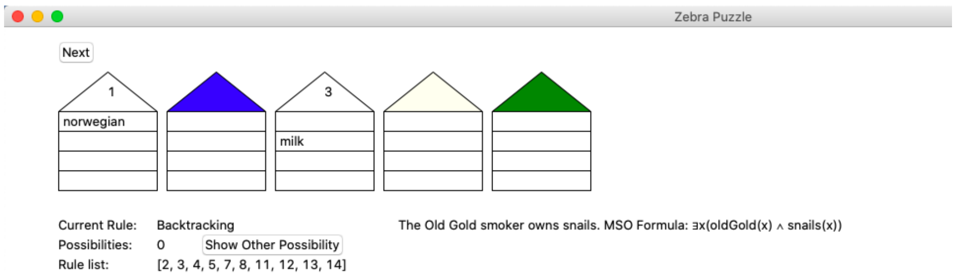


Current Rule: 6
Possibilities: 2 Show Other Possibility
Rule list: [2, 3, 4, 5, 7, 8, 11, 12, 13, 14]

The green house is immediately to the right of the ivory house. MSO Formula: ∃x∃y(xSy ∧ ivor

(b) Possibility 2

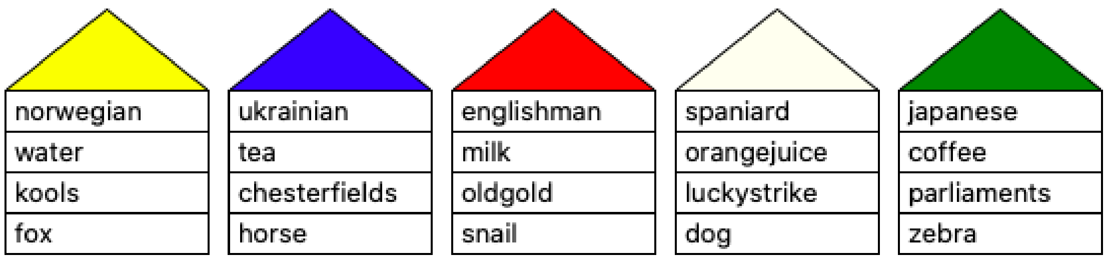
Figure 3.3: Two possibilities for Rule 6 in Zebra puzzle



Current Rule: Backtracking
Possibilities: 0 Show Other Possibility
Rule list: [2, 3, 4, 5, 7, 8, 11, 12, 13, 14]

The Old Gold smoker owns snails. MSO Formula: ∃x(oldGold(x) ∧ snails(x))

Figure 3.4: The Zebra Puzzle - Backtracking step



norwegian	ukrainian	englishman	spaniard	japanese
water	tea	milk	orangejuice	coffee
kools	chesterfields	oldgold	luckystrike	parliaments
fox	horse	snail	dog	zebra

Figure 3.5: The Zebra Puzzle - Final solution

3.3.1 Presuppositions shaping the sample space

Interval = non-empty set without holes

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

Here, $P_c(x)$ stands for *c is ar position x*
Bounded interval = period

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

For every Allen relation R,

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

3.3.2 MSO for Allen relations

For all strings s such that

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

an Allen relation R can be expressed as an MSO-formula $\chi_R[a,b]$

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

The formula $\chi_R[a,b]$ is

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

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	

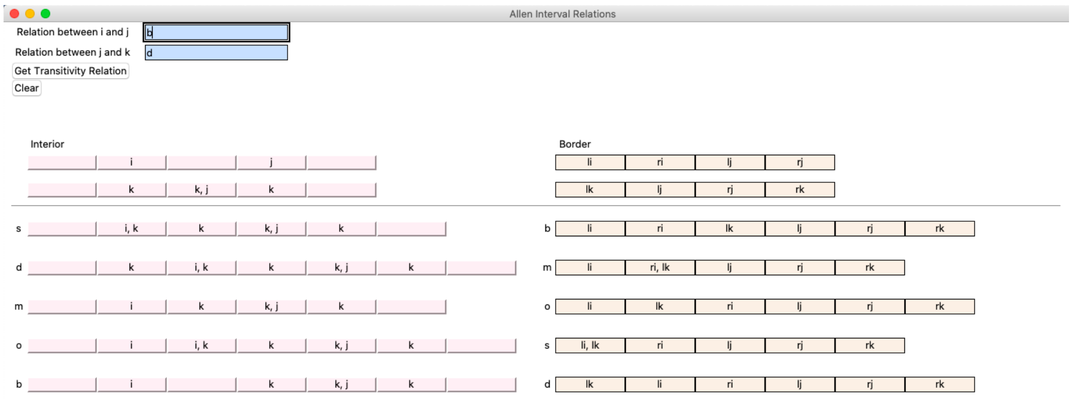
where

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

3.3.3 Simulation using Python TkInter

To simulate the behaviour of transitivity, I again used TkInter to take two Allen relation and compute the transitivity relations between them. When I say transitivity between them, it means that for two Allen relations r_1 and r_2 between i and j and j and k respectively the transitivity relations are the possible relations between i and k. The screenshots of the simulation are given in Figure 3.6.

The program was also extended to take input for the actual strings in the border format and then compute the superposition between them and display those strings in both bordered and interior format. This is shown in Figure 3.6.



Relation between i and j: b

Relation between j and k: d

```
Interior                              Border
   |i|   |j|                          li | ri | lj | rj
   |k|k, j|k|                         lk | lj | rj | rk

s  |i, k|k|k, j|k|                 b  li | ri | lk | lj | rj | rk
d  |k|i, k|k|k, j|k|               m  li | ri, lk | lj | rj | rk
m  |i|k|k, j|k|                    o  li | lk | ri | lj | rj | rk
o  |i|i, k|k|k, j|k|               s  li, lk | ri | lj | rj | rk
b  |i| |k|k, j|k|                  d  lk | li | ri | lj | rj | rk
```

Figure 3.6: Transitivity relation implementation in Tkinter

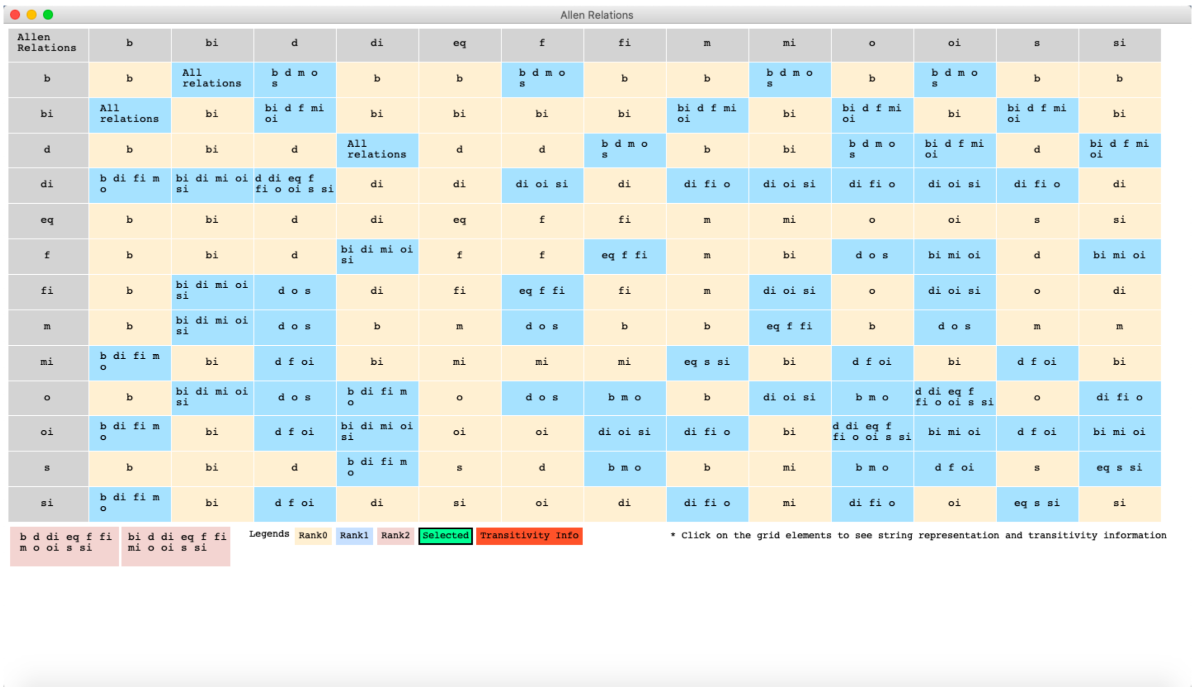
3.4 Non-Allen Relations in the Allen Transitivity Table

If you recall the Allen transitivity table as discussed, the transitivity of the thirteen Allen relations gives rise to some Allen relations and some combinations of Allen relations. I also tried to understand the behaviour of these combinations of Allen relations and how it can be represented in the interior, bordered and S-words format.

The program implemented in the previous section was utilized to construct the transitivity table in TkInter. In this implementation, again, superposition was used to compute the transitivity relations. The screenshot is given in Figure 3.7

3.4.1 Rank 0

Let us call the 13 Allen relations as the Rank 0 relations. The transitivity between the 13 Allen relations results in some Allen relation and some combination of Allen relation. The Rank 0 relations are shown in yellow tiles in Figure 3.7. The bordered string representation for the Rank 0 strings is given in Table 3.1.
Consider three intervals i, j, k such that $i\ r_1\ j$ and $j\ r_2\ k$, then $i\ r_3\ k$
r_1, r_2 and one of the Allen relations
If $r_1 = b$, $r_2 = b$ Then, $r3 = b$, which is Rank 0 relation.
If $r_1 = b$, $r_2 = d$ Then, $r3 = b, d, m, o, s$, which is a combination of Allen relations.


Allen Relations

Allen Relations	b	bi	d	di	eq	f	fi	m	mi	o	oi	s	si
b	b	All relations	b d m o s	b	b	b d m o s	b	b	b d m o s	b	b d m o s	b	b
bi	All relations	bi	bi d f mi oi	bi	bi	bi	bi	bi d f mi oi	bi	bi d f mi oi	bi	bi d f mi oi	bi
d	b	bi	d	All relations	d	d	b d m o s	b	bi	b d m o s	bi d f mi oi	d	bi d f mi oi
di	b di fi m o	bi di mi oi si	d di eq f fi o oi s si	di	di	di oi si	di	di fi o	di oi si	di fi o	di oi si	di fi o	di
eq	b	bi	d	di	eq	f	fi	m	mi	o	oi	s	si
f	b	bi	d	bi di mi oi si	f	f	eq f fi	m	bi	d o s	bi mi oi	d	bi mi oi
fi	b	bi di mi oi si	d o s	di	fi	eq f fi	fi	m	di oi si	o	di oi si	o	di
m	b	bi di mi oi si	d o s	b	m	d o s	b	b	eq f fi	b	d o s	m	m
mi	b di fi m o	bi	d f oi	bi	mi	mi	mi	eq s si	bi	d f oi	bi	d f oi	bi
o	b	bi di mi oi si	d o s	b di fi m o	o	d o s	b m o	b	di oi si	b m o	d di eq f fi o oi s si	o	di fi o
oi	b di fi m o	bi	d f oi	bi di mi oi si	oi	oi	di oi si	di fi o	bi	d di eq f fi o oi s si	bi mi oi	d f oi	bi mi oi
s	b	bi	d	b di fi m o	s	d	b m o	b	mi	b m o	d f oi	s	eq s si
si	b di fi m o	bi	d f oi	di	si	oi	di	di fi o	mi	di fi o	oi	eq s si	si

b d di eq f fi m o oi s si

bi d di eq f fi mi o oi s si

Legends Rank0 Rank1 Rank2 Selected Transitivity Info

* Click on the grid elements to see string representation and transitivity information

Figure 3.7: Allen Transitivity table implementation in Tkinter

Listing 3.4: Function for obtaining Rank 0 relations

```
def rank0():
    return ['b','bi','d','di','eq','f','fi','m','mi','o','oi','s','si']
```

The implementation of rank0 function is shown in 3.4.

Table 3.1: Rank 0 Allen relations

Relation between i and k	**Bordered string representation**
b	[ri][lk]
bi	[rk][ri]
d	[lk][li][ri][rk]
di	[li][lk][rk][ri]
eq	[li,lk][ri,rk]
f	[lk][li][ri,rk]
fi	[li][lk][ri,rk]
m	[ri,lk]

mi	[li,rk]
o	[li][lk][ri][rk]
oi	[lk[li][rk][ri]
s	[li,lk][ri][rk]
si	[li,lk][rk][ri]

3.4.2 Rank 1

The combinations of Allen relations obtained by Rank 0 relations in the last step is what we call as Rank 1 relations. The blue tiles in Figure 3.7 represent the Rank 1 relations. 14 such relations are obtained and these are also shown in Table 3.2. The implementation of the rank1 function is provided in Listing 3.5. Here the function $tt()$ is used to get the transitivity relations.

Listing 3.5: Function for obtaining Rank 1 relations

```
def rank1():
    rank1_strings = []
    rank0_strings = rank0()
    rank1_dictionary = []
    for r1 in rank0_strings:
        for r2 in rank0_strings:
            temp,s = tt(r1,r2)
            if temp not in rank1_strings:
                if len(temp) != 1:
                    temp.sort()
                    if not in(temp,rank1_strings):
                        rank1_dictionary.append({
                        "r" : (r1, r2),
                        's': temp
                        })
                        rank1_strings.append(temp)
    return rank1_strings,rank1_dictionary
```

Table 3.2: Rank 1 Allen relations

Relation between i and k	**Bordered string representation**
b,bi,d,di,eq,f,fi,m,mi,o,oi,s,si	-
b,d,m,o,s	[ri][rk]
b,di,fi,m,o	[li][lk]
b,m,o	o;o
bi,d,f,mi,oi	[lk][li]
bi,di,mi,oi,si	[rk][ri]
bi,mi,oi	oi;oi
d,di,eq,f,fi,o,oi,s,si	di;d
d,f,oi	[lk][li][rk]
d,o,s	[lk][ri][rk]
di,fi,o	[li][lk][ri]
di,oi,si	[li][rk][ri]
eq,f,fi	[ri,rk]
eq,s,si	[li,lk]

All these relations R can be pictured by a string s_R with vocabulary $\subseteq \{li, lk, ri, rk\}$ in that

$L_R = s_R$ & [li] [ri] & [lk] [rk]

except for [b,m,o], [bi,mi,oi] and [d,di,eq,f,fi,o,oi,s,si]. The representations for this introduces non determinism and is given in Table 3.3.

Unwinding <>introduces non-determinism (internalized by <>). This means that [<i>,<k>] can take either the for [i][k] or [k][i] or [i, k]

$$[< i >, < k >] \rightarrow [i][k] + [k][i] + [i, k]$$

Table 3.3: Rank 2 Allen relations

Relation between i and k	**Notation**	**Bordered string representation**
b,m,o	o;o	[li] [<ri>, <lk>] [rk]
bi,mi,o	oi;oi	[lk] [<rk>, <li>] [ri]
d,di,eq,f,fi,o,oi,s,si	di;d	[<li>,<lk>] [<ri>,<rk>]

These non-deterministic representations are somewhat similar to the mix operations in S-words. Like the mix operation, here as well, the intervals in angular brackets can be expanded to include all the possibilities of occurrences.

3.4.3 Rank 2

Similar to the Rank 1 relations, Rank 2 relations are also obtained by transitivity between Rank 1 relations. Only two such relations are obtained and are shown in Table 3.4. The implementation of the rank2 function is provided in Listing 3.6.
After the Rank 2 relations, no further relations of higher ranks are obtained.

Table 3.4: Rank 2 Allen relations with non-deterministic notations

Relation between i and k	**Bordered string representation**
b,d,di,eq,f,fi,m,o,oi,s,si	[li][rk]
bi,d,di,eq,f,fi,mi,o,oi,s,si	[lk][ri]

Listing 3.6: Function for obtaining Rank 2 relations

```
def rank2():
    rank0_strings = rank0()
    rank1_strings,rank1_dictionary = rank1()
    rank2_strings = []
    rank2_dictionary = []
    for r1s in rank1_strings:
        for r2s in rank1_strings:
            temp = []
            for r1 in r1s:
                for r2 in r2s:
                    r,s = tt(r1,r2)
                    temp.extend(r)
            temp.sort()
            temp2 = list(set(temp))
            temp2.sort()
            if not in(temp2,rank2_strings)
            and not in(temp2,rank1_strings)
```

```
                and not in(temp2, rank0_strings):
                    if len(temp) != 1:
                        rank2_dictionary.append({
                            'r' : (r1s, r2s),
                            's': temp2
                        })
                        rank2_strings.append(temp2)
    return rank2_strings, rank2_dictionary
```

The bordered representation provides the shortest way of the representation because of the left and the right borders. Hence, it was considered in this section to represent the higher ranked relations. The S-words will require the two occurrences of each interval *i* and *k* which will increase the length of the strings. On the other hand, the interior representation will require empty boxes that will also increase the length of the strings.

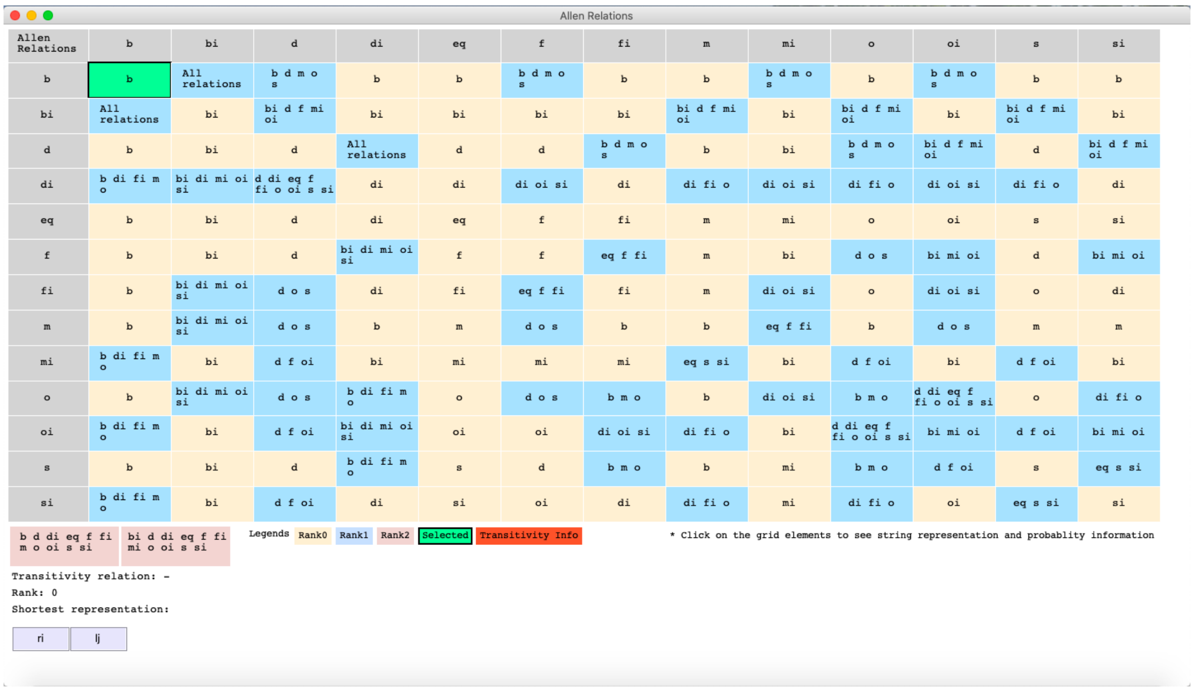
3.4.4 Interactive transitivity table using TkInter

Using Python's TkInter library, I developed an interactive transitivity table as shown in Figure 3.7. In this table, the tiles are organized into Rank 0, Rank 1 and Rank 2 relations and each of them are clickable. The link to the Python programs and the execution details are given in the appendix section. The program provides the following information:

1. Clicking on a Rank 0 relation gives its notation. This is shown in Figure 3.8. The tile 'b' is clicked (shown in green). Also shown below is the shortest notation to represent it in bordered strings.

2. Clicking on two Rank 0 relations highlights the relation which is a result of the transitivity of those two clicked relations. Additionally, the string notation is provided for the new highlighted relation. In Figure 3.9, 'b' and 'd' are clicked (shown in green) and the relation 'b,d,m,o,s' is highlighted (in red), which is a result of transitivity of relations 'b' and 'd'.

3. Similar is the case for Rank 1 and Rank 2 relations. Clicking on Rank 1 relation highlights the two Rank 0 relations from which this relationship might have



Allen Relations	b	bi	d	di	eq	f	fi	m	mi	o	oi	s	si
b	b	All relations	b d m o s	b	b	b d m o s	b	b	b d m o s	b	b d m o s	b	b
bi	All relations	bi	bi d f mi oi	bi	bi	bi	bi	bi d f mi oi	bi	bi d f mi oi	bi	bi d f mi oi	bi
d	b	bi	d	All relations	d	d	b d m o s	b	bi	b d m o s	bi d f mi oi	d	bi d f mi oi
di	b di fi m o	bi di mi oi si	d di eq f fi o oi s si	di	di	di oi si	di	di fi o	di oi si	di fi o	di oi si	di fi o	di
eq	b	bi	d	di	eq	f	fi	m	mi	o	oi	s	si
f	b	bi	d	bi di mi oi si	f	f	eq f fi	m	bi	d o s	bi mi oi	d	bi mi oi
fi	b	bi di mi oi si	d o s	di	fi	eq f fi	fi	m	di oi si	o	di oi si	o	di
m	b	bi di mi oi si	d o s	b	m	d o s	b	b	eq f fi	b	d o s	m	m
mi	b di fi m o	bi	d f oi	bi	mi	mi	mi	eq s si	bi	d f oi	bi	d f oi	bi
o	b	bi di mi oi si	d o s	b di fi m o	o	d o s	b m o	b	di oi si	b m o	d di eq f fi o oi s si	o	di fi o
oi	b di fi m o	bi	d f oi	bi di mi oi si	oi	oi	di oi si	di fi o	bi	d di eq f fi o oi s si	bi mi oi	d f oi	bi mi oi
s	b	bi	d	b di fi m o	s	d	b m o	b	mi	b m o	d f oi	s	eq s si
si	b di fi m o	bi	d f oi	di	si	oi	di	di fi o	mi	di fi o	oi	eq s si	si

b d di eq f fi m o oi s si	bi d di eq f fi mi o oi s si

Legends Rank0 Rank1 Rank2 Selected Transitivity Info

* Click on the grid elements to see string representation and probablity information

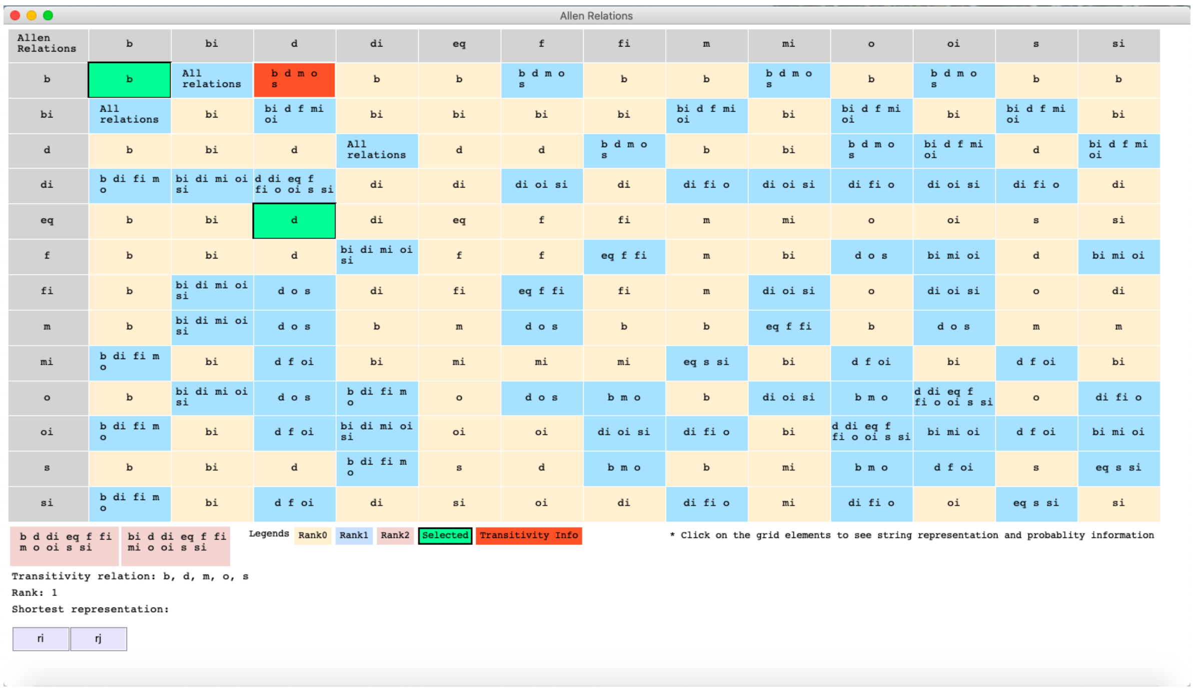
Transitivity relation: -

Rank: 0

Shortest representation:

ri lj

Figure 3.8: Allen Transitivity table - clicking one Rank 0 relation 'b'



Allen Relations	b	bi	d	di	eq	f	fi	m	mi	o	oi	s	si
b	b	All relations	b d m o s	b	b	b d m o s	b	b	b d m o s	b	b d m o s	b	b
bi	All relations	bi	bi d f mi oi	bi	bi	bi	bi	bi d f mi oi	bi	bi d f mi oi	bi	bi d f mi oi	bi
d	b	bi	d	All relations	d	d	b d m o s	b	bi	b d m o s	bi d f mi oi	d	bi d f mi oi
di	b di fi m o	bi di mi oi si	d di eq f fi o oi s si	di	di	di oi si	di	di fi o	di oi si	di fi o	di oi si	di fi o	di
eq	b	bi	d	di	eq	f	fi	m	mi	o	oi	s	si
f	b	bi	d	bi di mi oi si	f	f	eq f fi	m	bi	d o s	bi mi oi	d	bi mi oi
fi	b	bi di mi oi si	d o s	di	fi	eq f fi	fi	m	di oi si	o	di oi si	o	di
m	b	bi di mi oi si	d o s	b	m	d o s	b	b	eq f fi	b	d o s	m	m
mi	b di fi m o	bi	d f oi	bi	mi	mi	mi	eq s si	bi	d f oi	bi	d f oi	bi
o	b	bi di mi oi si	d o s	b di fi m o	o	d o s	b m o	b	di oi si	b m o	d di eq f fi o oi s si	o	di fi o
oi	b di fi m o	bi	d f oi	bi di mi oi si	oi	oi	di oi si	di fi o	bi	d di eq f fi o oi s si	bi mi oi	d f oi	bi mi oi
s	b	bi	d	b di fi m o	s	d	b m o	b	mi	b m o	d f oi	s	eq s si
si	b di fi m o	bi	d f oi	di	si	oi	di	di fi o	mi	di fi o	oi	eq s si	si

b d di eq f fi m o oi s si	bi d di eq f fi mi o oi s si

Legends Rank0 Rank1 Rank2 Selected Transitivity Info

* Click on the grid elements to see string representation and probablity information

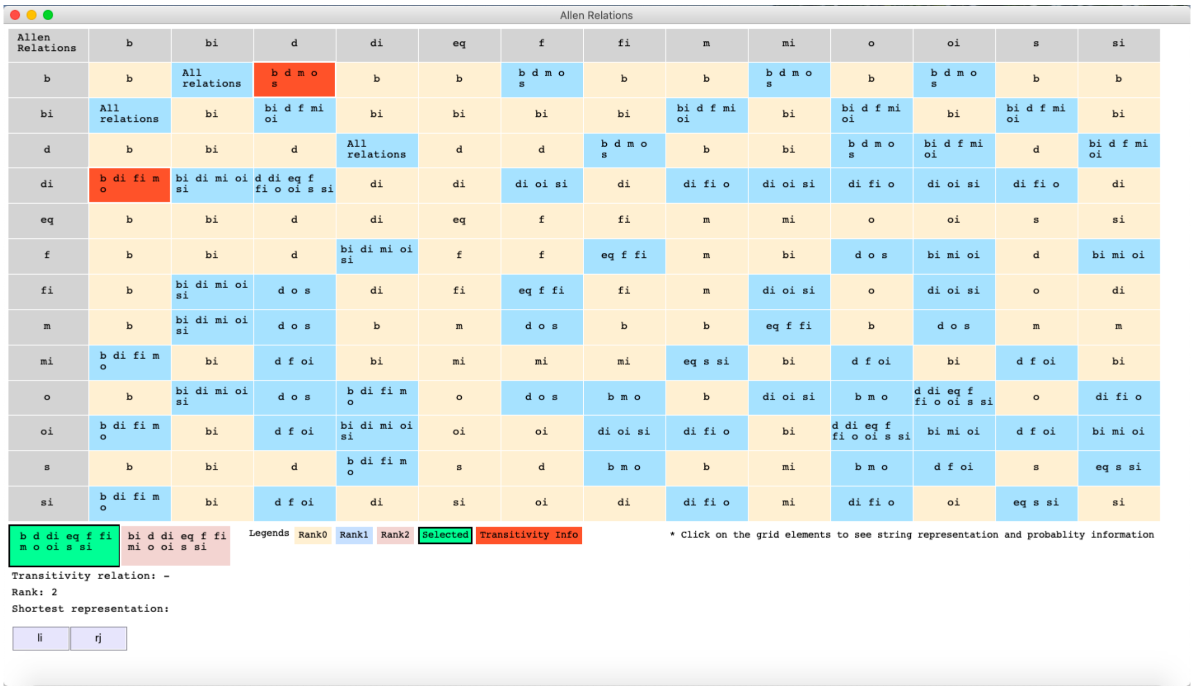
Transitivity relation: b, d, m, o, s

Rank: 1

Shortest representation:

ri rj

Figure 3.9: Allen Transitivity table - clicking two Rank 0 relation 'b' and 'd' highlights 'b,d,m,o,s'



Allen Relations

Allen Relations	b	bi	d	di	eq	f	fi	m	mi	o	oi	s	si
b	b	All relations	b d m o s	b	b	b d m o s	b	b	b d m o s	b	b d m o s	b	b
bi	All relations	bi	bi d f mi oi	bi	bi	bi	bi	bi d f mi oi	bi	bi d f mi oi	bi	bi d f mi oi	bi
d	b	bi	d	All relations	d	d	b d m o s	b	bi	b d m o s	bi d f mi oi	d	bi d f mi oi
di	b di fi m o	bi di mi oi si	d di eq f fi o oi s si	di	di	di oi si	di	di fi o	di oi si	di fi o	di oi si	di fi o	di
eq	b	bi	d	di	eq	f	fi	m	mi	o	oi	s	si
f	b	bi	d	bi di mi oi si	f	f	eq f fi	m	bi	d o s	bi mi oi	d	bi mi oi
fi	b	bi di mi oi si	d o s	di	fi	eq f fi	fi	m	di oi si	o	di oi si	o	di
m	b	bi di mi oi si	d o s	b	m	d o s	b	b	eq f fi	b	d o s	m	m
mi	b di fi m o	bi	d f oi	bi	mi	mi	mi	eq s si	bi	d f oi	bi	d f oi	bi
o	b	bi di mi oi si	d o s	b di fi m o	o	d o s	b m o	b	di oi si	b m o	d di eq f fi o oi s si	o	di fi o
oi	b di fi m o	bi	d f oi	bi di mi oi si	oi	oi	di oi si	di fi o	bi	d di eq f fi o oi s si	bi mi oi	d f oi	bi mi oi
s	b	bi	d	b di fi m o	s	d	b m o	b	mi	b m o	d f oi	s	eq s si
si	b di fi m o	bi	d f oi	di	si	oi	di	di fi o	mi	di fi o	oi	eq s si	si

b d di eq f fi m o oi s si	bi d di eq f fi mi o oi s si

Legends Rank0 Rank1 Rank2 Selected Transitivity Info

* Click on the grid elements to see string representation and probablity information

Transitivity relation: -

Rank: 2

Shortest representation:

Figure 3.10: Allen Transitivity table - clicking Rank 2 relation 'b,d,di,eq,f,fi,m,o,oi,s,s' highlights the two relations from which it can be obtained

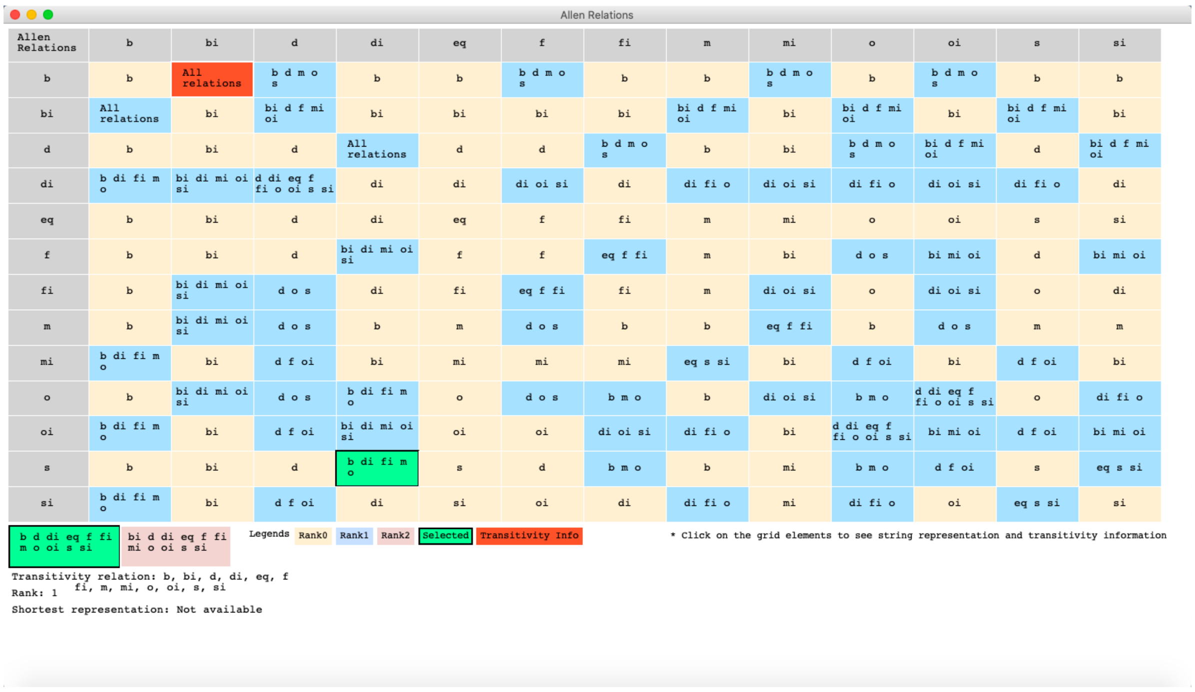
occurred. Clicking on two Rank 1 relations highlights the relation which is a result of transitivity of the two clicked relations. Some of the screenshots for this are provided in Figure 3.10 and 3.11.

3.5 Solving the Six Train Problem

The approach to solving the six train problem was similar to the one followed for the Zebra puzzle. The main idea was to compare it with the solution obtained in [3] by SLS using the mix and join operations.

3.5.1 MSO Formula and understanding S-words

As a first step, all the constraints of the six train problem were converted to the MSO formulas.



Allen Relations	b	bi	d	di	eq	f	fi	m	mi	o	oi	s	si
b	b	All relations	b d m o s	b	b	b d m o s	b	b	b d m o s	b	b d m o s	b	b
bi	All relations	bi	bi d f mi oi	bi	bi	bi	bi	bi d f mi oi	bi	bi d f mi oi	bi	bi d f mi oi	bi
d	b	bi	d	All relations	d	d	b d m o s	b	bi	b d m o s	bi d f mi oi	d	bi d f mi oi
di	b di fi m o	bi di mi oi si	d di eq f fi o oi s si	di	di	di oi si	di	di fi o	di oi si	di fi o	di oi si	di fi o	di
eq	b	bi	d	di	eq	f	fi	m	mi	o	oi	s	si
f	b	bi	d	bi di mi oi si	f	f	eq f fi	m	bi	d o s	bi mi oi	d	bi mi oi
fi	b	bi di mi oi si	d o s	di	fi	eq f fi	fi	m	di oi si	o	di oi si	o	di
m	b	bi di mi oi si	d o s	b	m	d o s	b	b	eq f fi	b	d o s	m	m
mi	b di fi m o	bi	d f oi	bi	mi	mi	mi	eq s si	bi	d f oi	bi	d f oi	bi
o	b	bi di mi oi si	d o s	b di fi m o	o	d o s	b m o	b	di oi si	b m o	d di eq f fi o oi s si	o	di fi o
oi	b di fi m o	bi	d f oi	bi di mi oi si	oi	oi	di oi si	di fi o	bi	d di eq f fi o oi s si	bi mi oi	d f oi	bi mi oi
s	b	bi	d	b di fi m o	s	d	b m o	b	mi	b m o	d f oi	s	eq s si
si	b di fi m o	bi	d f oi	di	si	oi	di	di fi o	mi	di fi o	oi	eq s si	si

b d di eq f fi m o oi s si

bi d di eq f fi mi o oi s si

Legends Rank0 Rank1 Rank2 Selected Transitivity Info

* Click on the grid elements to see string representation and transitivity information

Transitivity relation: b, bi, d, di, eq, f fi, m, mi, o, oi, s, si

Rank: 1

Shortest representation: Not available

Figure 3.11: Allen Transitivity table - clicking 'b,d,di,eq,f,fi,m,o,oi,s,s' and 'b,di,fi,m,o' highlights 'All relations'

Identify each train X with an interval $(l(X), r(X)]$ when X is at the platform so that

$$l(X) \text{ is } X\text{'s arrival (reaching) time}$$

and

$$r(X) \text{ is } X\text{'s departure (leaving) time.}$$

(Unfortunately, l clashes with *r*eaching, and r clashes with *l*eaving.)
Constraints 1-5 become:

1. A, B, E, reach the platform at the same time.

   $\exists x(P_{l(A)}(x) \wedge P_{l(B)}(x) \wedge P_{l(E)}(x))$

   String $\boxed{l(A), l(B), l(E)}$

2. A leaves before B.

   $\exists x \exists y(x < y \wedge P_{r(A)}(x) \wedge P_{r(B)}(y))$

String $\boxed{r(A)}\boxed{r(B)}$

3. A leaves after or at the same time as C but before the arrival of D.

   $\exists x \exists y \exists z (P_{r(A)}(x) \wedge P_{r(C)}(y) \wedge (y < x \vee x = y) \wedge P_{l(D)}(z) \wedge x < z)$

   Strings $(\boxed{r(C)}\boxed{r(A)} + \boxed{r(A), r(C)})$ & $\boxed{r(A)}\boxed{l(D)}$

   $= (\boxed{r(C)}\boxed{r(A)} + \boxed{r(A), r(C)})\boxed{l(D)}$

4. D and F arrive at the same time as B is leaving.

   $\exists x (P_{l(D)}(x) \wedge P_{l(F)}(x) \wedge P_{r(B)}(x)$

   String $\boxed{l(D), l(F), r(B)}$

5. E and D leave at the same time.

   $\exists x (P_{r(E)}(x) \wedge P_{r(D)}(x))$

   String $\boxed{r(E), r(D)}$

where $u < v$ is as defined in "MSO for Allen Relations"

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

and we also have the background constraints (presuppositions defining context/sample space)

$$\boxed{l(A)}\boxed{r(A)} \ \& \ \boxed{l(B)}\boxed{r(B)} \ \& \ \boxed{l(C)}\boxed{r(C)} \ \& \ \boxed{l(D)}\boxed{r(D)} \ \& \ \boxed{l(E)}\boxed{r(E)}$$

The MSO formulas for the left and right borders correspond to the onset and end of the intervals in S-words which can be understood as per the above details.

3.5.2 Solution using Superposition

As a next step, all the MSO formulas were converted to their corresponding bordered string notations. These are given below for each of the five constraints:

$S1 = [\{lA, lB, lE\}]$
$S2 = [\{rA\}, \{rB\}]$
$S3 = [\{rC\}, \{rA\}, \{lD\}]$ or $[\{rC, rA\}, \{lD\}]$
$S4 = [\{lD, lF, rB\}]$
$S5 = [\{rE, rD\}]$

Here, A, B, C, D, E, F stands for the six trains and l is used to represent the arrival of the train and r is used to portray the departure of the train.

Along with these 5 strings, six other strings need to be added for the background constraints mentioned in the previous section.

$S6 = [\{lA\}, \{rA\}]$
$S7 = [\{lB\}, \{rB\}]$
$S8 = [\{lC\}, \{rC\}]$
$S9 = [\{lD\}, \{rD\}]$
$S10 = [\{lE\}, \{rE\}]$
$S11 = [\{lF\}, \{rF\}]$

The python function for superposition is utilized here to find the solutions. All the strings and superposed with one another to get more strings. All these obtained strings are passed to a function that computes the minimum tracks required. The program yields 24 solutions of length between 4 and 7. Each of them requires a minimum of 4 tracks. Thus, the answer is 4 tracks. The code snippet in 3.7 shows the implementation. The *superpose*() function here is the superposition function seen in the earlier sections.

The *get_tracks*() implementation in 3.8 is used for obtaining the minimum number of tracks required for a given string.

Listing 3.7: Implementation of solution for Six train problem using Superposition

```
E1 = [{'la','lb','le'}]
E2 = [{'ra'},{'rb'}]
E3 = [{'rc'},{'ra'},{'ld'}]
E4 = [{'ld','lf','rb'}]
E5 = [{'re','rd'}]
```

```
E6 = [{'la'},{'ra'}]
E7 = [{'lb'},{'rb'}]
E8 = [{'lc'},{'rc'}]
E9 = [{'ld'},{'rd'}]
E10 = [{'le'},{'re'}]
E11 = [{'lf'},{'rf'}]

E = [E2,E3,E4,E5,E6,E7,E8,E9,E10,E11]
pr = [E1]

for e in E:
    sat = []
    for strings in pr:
        for s in superpose(strings, e):
            if s not in sat:
                sat.append(s)
    pr = satisfied
```

Listing 3.8: Implementation for obtaining number of tracks

```
def get_tracks(s):
    tracks = {
        't1':'',
        't2':'',
        't3':'',
        't4':'',
        't5':'',
        't6':'',
    }
    unique_tracks = set()
    to_empty = False
    empty_loc = []
    for s1 in s:
        for s2 in s1:
            if s2[0] == 'l':
                loc = get_track_key(tracks, '')
                tracks[loc] = s2[1]
                unique_tracks.add(loc)
            else:
                empty_loc.append(get_track_key(tracks, s2[1]))
                to_empty = True
        if to_empty:
            for e in empty_loc:
                tracks[e] = ''
    return len(unique_tracks)
```

All the 24 solutions that are obtained are given below:

1. $[\{lA, lE, lB\}, \{lC\}, \{rC\}, \{rA\}, \{lF, rB, lD\}, \{rE, rD\}, \{rF\}]$
2. $[\{lA, lE, lB\}, \{lC\}, \{rC\}, \{rA\}, \{lF, rB, lD\}, \{rE, rF, rD\}]$
3. $[\{lA, lE, lB\}, \{lC\}, \{rC\}, \{rA\}, \{lF, rB, lD\}, \{rF\}, \{rE, rD\}]$
4. $[\{lE, lB, lA, lC\}, \{rC\}, \{rA\}, \{lF, rB, lD\}, \{rE, rD\}, \{rF\}]$
5. $[\{lE, lB, lA, lC\}, \{rC\}, \{rA\}, \{lF, rB, lD\}, \{rE, rF, rD\}]$
6. $[\{lE, lB, lA, lC\}, \{rC\}, \{rA\}, \{lF, rB, lD\}, \{rF\}, \{rE, rD\}]$
7. $[\{lC\}, \{lA, lE, lB\}, \{rC\}, \{rA\}, \{lF, rB, lD\}, \{rE, rD\}, \{rF\}]$
8. $[\{lC\}, \{lA, lE, lB\}, \{rC\}, \{rA\}, \{lF, rB, lD\}, \{rE, rF, rD\}]$
9. $[\{lC\}, \{lA, lE, lB\}, \{rC\}, \{rA\}, \{lF, rB, lD\}, \{rF\}, \{rE, rD\}]$
10. $[\{lC\}, \{lE, lB, lA, rC\}, \{rA\}, \{lF, rB, lD\}, \{rE, rD\}, \{rF\}]$
11. $[\{lC\}, \{lE, lB, lA, rC\}, \{rA\}, \{lF, rB, lD\}, \{rE, rF, rD\}]$
12. $[\{lC\}, \{lE, lB, lA, rC\}, \{rA\}, \{lF, rB, lD\}, \{rF\}, \{rE, rD\}]$
13. $[\{lC\}, \{rC\}, \{lA, lE, lB\}, \{rA\}, \{lF, rB, lD\}, \{rE, rD\}, \{rF\}]$
14. $[\{lC\}, \{rC\}, \{lA, lE, lB\}, \{rA\}, \{lF, rB, lD\}, \{rE, rF, rD\}]$
15. $[\{lC\}, \{rC\}, \{lA, lE, lB\}, \{rA\}, \{lF, rB, lD\}, \{rF\}, \{rE, rD\}]$

16. $[\{lA, lE, lB\}, \{lC\}, \{rA, rC\}, \{lF, rB, lD\}, \{rE, rD\}, \{rF\}]$

17. $[\{lA, lE, lB\}, \{lC\}, \{rA, rC\}, \{lF, rB, lD\}, \{rE, rF, rD\}]$

18. $[\{lA, lE, lB\}, \{lC\}, \{rA, rC\}, \{lF, rB, lD\}, \{rF\}, \{rE, rD\}]$

19. $[\{lE, lB, lA, lC\}, \{rA, rC\}, \{lF, rB, lD\}, \{rE, rD\}, \{rF\}]$

20. $[\{lE, lB, lA, lC\}, \{rA, rC\}, \{lF, rB, lD\}, \{rE, rF, rD\}]$

21. $[\{lE, lB, lA, lC\}, \{rA, rC\}, \{lF, rB, lD\}, \{rF\}, \{rE, rD\}]$

22. $[\{lC\}, \{lA, lE, lB\}, \{rA, rC\}, \{lF, rB, lD\}, \{rE, rD\}, \{rF\}]$

23. $[\{lC\}, \{lA, lE, lB\}, \{rA, rC\}, \{lF, rB, lD\}, \{rE, rF, rD\}]$

24. $[\{lC\}, \{lA, lE, lB\}, \{rA, rC\}, \{lF, rB, lD\}, \{rF\}, \{rE, rD\}]$

These strings are the same as the ones obtained by the mix and join operations in S-words as represented in Equation 2.15. The paper [3] mentions that 24 S-words are obtained between the length 5 and 7. However, I feel this could have been a print mistake as I was able to find a solution of length 4.

Chapter 4

Conclusion

With the importance of temporal reasoning and representation various methods were studied including the Allen relations, S-words, Interior notations and bordered strings. The S-words by Schwer were understood in terms of Monadic second order logic (MSO) as S-words could already be understood in terms of regular languages and regular languages can be expressed in MSO (Bücci, Elgot and Trakhtenbrot theorem). Throughout the process, superposition was also implemented in Python to be used in solving constraint satisfaction problems. The concept of superposition was also compared with the Join operation in S-words.

4.1 Results

With the aim of understanding S-words, various concepts were understood throughout the process. These included the MSO, Superposition, Allen relations, Interior and Bordered strings and S-words. Additionally, two case studies of the Zebra puzzle and the Six train problem were also discussed along with their solutions.

The implementation of the superposition theorem proved out to be very useful in finding the solution to the Zebra puzzle and the Six train problem. Not only that, it was also useful in obtaining the transitivity in Allen relations. This helped in understanding that it is useful in solving problems related to temporal constraints (Six train problem and Allen relations) as well as the ones with general constraints (Zebra puzzle).

However, representation of these strings in different formats for the superposition was crucial as each came with their own advantages and disadvantages. For instance, the Six train problem was solved easily by the bordered string representation and S-words whereas the Zebra puzzle was better with interior representations. On the other hand, the transitivity in Allen relations was equally complex in both interior and bordered representations.

The higher ranked non-allen relations (Rank 0, Rank 1 and Rank 2) in the Allen transitivity table along with their representations were also studied that helped in realizing that there could be shorter notations to represent them. Here usage of the bordered representation proved to be useful in comparison to the S-words as it drastically reduced the length of the notation.
The usage of MSO formulas helped in understanding the S-words better.

$$\exists x(P_{l(A)}(x))$$

where it stands for *'the left border of A is at position x'*, which means that the interval A begins at the instant x.

The MSO formulas also helped in understanding all the constraints better and with the help of MSO conjunction, i.e. superposition, the CSPs like the Zebra puzzle and the Six train problems were solved. This can be extended to solve any constraint satisfaction problem after all its constraints are converted to the MSO formulas.

The different notations for representing intervals were studied: Interior, Border and S-words. The bordered notation provides a clear representation in terms of understanding because both the borders are marked with l and r. The interior and S-words, on the other hand, make use of empty boxes and two occurrences of intervals respectively to differentiate borders, which makes them a little confusing to understand. The mix operation denoted by 'X' in the S-words gives a very compressed representation to represent many S-words. We saw the single expression in the six train problem which represents the 24 S-words.

The constraints give information about either the arrival of the train or the departure of the trains. In the case of border strings, it is easier to represent them as they are differentiated as $'la'$ and $'ra'$. Whereas, in the case of s-languages, it requires the occurrence of two 'a' to represent the arrival and the departure respectively which gives rise to more possibilities.

For instance, the 5th constraint - E and D leave at the same time can be expressed as :
$[\{rE, rD\}]$ in the case of border strings.
and ([{e}] X [{d}]) · [d,e] in the case of s-words which result in three strings -

1. $[\{e\}],[\{d\}],[\{d,e\}]$

2. $[\{e\},\{d\}],[\{d,e\}]$

3. $[\{d\}],[\{e\}],[\{d,e\}]$

Here the X represents the uncertainty of the arrival of e and d. Although this mix operation (X) provides a shorter way of representation, we need to add strings which we do not have information on which adds additional complexity.

In the case of border strings, few more constraints need to be added to make sure that the left border (lA) occurs strictly before the right border (rA). This increases the total number of initial strings for superposition.

4.2 Further work

As a future work, the MSO formula utilized in finding the solutions of the constraint satisfaction problems - Zebra puzzle and Six train problem, can be implemented in the MONA tool. The MONA tool is used for representing MSO formulas and it could be extended to solve the constraint satisfaction problem. Due to time constraints, this was not explored during the dissertation.

Bibliography

[1] J. F. ALLEN, "Maintaining knowledge about temporal intervals," in *Communications of the ACM*, vol. 26, 1983.

[2] S. R. Schwer, "Temporal reasoning without transitive tables," *CoRR*, vol. abs/0706.1290, 2007.

[3] I. A. Durand and S. R. Schwer, "Reasoning about qualitative temporal information with s-words and slanguages," in *Proceedings of the 1st European Lisp Symposium*, 05 2008.

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

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

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

[7] 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.

[8] M. Vilain and H. Kautz, "Constraint propagation algorithms for temporal reasoning," in *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, p. 377–382, AAAI Press, 1986.

[9] E. Y. Kandrashina, "Representation of temporal knowledge," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 1*,

IJCAI'83, (San Francisco, CA, USA), p. 346–348, Morgan Kaufmann Publishers Inc., 1983.

[10] C. Freksa, "Temporal reasoning based on semi-intervals.," in *Artificial Intelligence, Volume 87, Issues 1–2*, p. 387, 11 1996.

[11] P. Ladkin, "Time representation: A taxonomy of internal relations.," in *Proceedings of the 5th National Conference on Artificial Intelligence.*, pp. 360–366, 01 1986.

[12] 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/`.

[13] J. Väänänen, "Second-order and higher-order logic," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, fall 2020 ed., 2020.

[14] Wikipedia contributors, "Zebra puzzle — Wikipedia, the free encyclopedia," 2020. [Online; accessed 26-July-2020].

[15] M. Gregor, K. Zábovská, and V. Smataník, "The zebra puzzle and getting to know your tools," in *IEEE 19th International Conference on Intelligent Engineering Systems*, 09 2015.

[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] J. F. Allen, "An interval-based representation of temporal knowledge," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'81, (San Francisco, CA, USA), p. 221–226, Morgan Kaufmann Publishers Inc., 1981.

[18] 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.

[19] S. Schwer, “Traitement de la temporalité des discours : une analysis situs,” 07 2006.

[20] Wikipedia contributors, “Tkinter,” 2020. [Online; accessed 26-July-2020].

Appendix

All the implementation codes are available on github at the link:
https://github.com/shrau94/MSO_S_Words_Superposition
The repository is for the implementation of all the codes and the details for the installations and executions are given below.

Requirements and Installations

The implementations are done in Python notebook with the python version 3.7.4. All the dependent libraries are given below:

1. Install Python v3.7.4 or above.
2. Install the TkInter library v8.5 or above using the below command.
pip install tkinter
2. Install Jupyter Lab for the execution of the codes.

Executing the codes

Run the files in Jupyter Lab or any other editor for ipynb files.

Details of the programs

1. zebra_puzzle_simulation.ipynb

   This program utilizes the superposition theorem to solve the Zebra Puzzle problem (a Constraint Satisfaction Problem). It also provides a GUI and simulation of all the steps and illustrates the MSO formulas used.

2. allen_relations_transitivity_superposition.ipynb

This file provides the implementation for obtaining the transitivity between two Allen relations using superposition theorem.

3. interractive_transitivity_grid.ipynb

   The interactive transitivity grid for Allen relations is provided along with the Rank 0, Rank 1 and Rank 2 relations.

4. six_train_problem.ipynb

   Another CSP with temporal constraints - The Six Train Problem is also solved using superposition and the results are compared with the results obtained by Join operations in S-words.