Record-based Automata for TTR

Joshua Prakash Manoharan, B.Eng

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

August 2021

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.

Joshua Prakash Manoharan

August 31, 2021

Permission to Lend and/or Copy

Acknowledgments

I would like to express my gratitude to my supervisor, Mr. Tim Fernando from the School of Computer Science at Trinity College Dublin (TCD), for guiding and supporting me throughout this research. Even in this dire situation, he helped me to overcome the obstacles that were faced during this work. I am deeply indebted to him for his insights and expertise in this area. I would also like to thank our course director Dr. John Dingliana for his constant support and guidance. Thanks are also due to my loved ones for supporting me throughout the entire process.

Joshua Prakash Manoharan

University of Dublin, Trinity College
August 2021

Record-based Automata for TTR

Joshua Prakash Manoharan, Master of Science in Computer Science

University of Dublin, Trinity College, 2021

Supervisor: Tim Fernando

Making sense of actions and events observed or expressed via disfluencies, a variety of types of non-sentential utterance, and partial comprehension in a systematic way is a challenge in the field of Natural Language Semantics. There have been several frameworks structured in the past to provide computational insights to extract this information. In this research work, we particularly talk in detail about Robin Cooper's framework for representing such hindrance through finite automata-based Type Theory with Records (TTR), which enables a uniform theory of grammar, semantics, and interaction to maintain past insights and paved the way for probabilistic based semantics. In addition, it's a proven fact that actions and events are closely related to time. Therefore, we also considered the 13 temporal relations created by James Allen in his research work. Later, the actions and events based temporal relations are constructed using superposition theorem by representing them in the form of MSO (Monadic Second Order) logic, which was related to finite-state methods as proved by Büchi, Elgot, Trakhtenbrot research work. This helped us to extract temporal information and to find out new results in this field of research. Finally, we implemented Gap-based string inference to retrieve all minimum possible strings required to construct string L from string L'.

Summary

Representing actions and events as observed by the agent of the environment in a way that will avoid set-theoretic paradoxes and provide a well-structured compositional semantic interpretation is an uphill task. Many age-old frameworks like semantic ontology, syntax-semantics interfaces based TFS (Typed Feature Structure) linguistic models like HPSG (Head-driven Phrase Structure Grammar)) and SCG (Sign-based Construction Grammar) are insufficient in capturing the bindings, functions, and cognitive aspects of the agent. Hence, this paper focuses entirely on defining a state of art framework over "Record-based Automata for TTR". To achieve this, actions and events are represented in the form of a string of records defined under the concept of string theory of events where events act as states and actions act as transition paths. In addition, the 13-temporal relations of events defined by Allen were placed as an icing over the constructed string theory of events. Thus, helping us to incorporate all these relations in our defined framework. To execute such incorporation, the research work done by Büchi, Elgot, Trakhtenbrot was considered. It states that the language defined under MSO is equivalent to regular languages, which can be expressed using finite-state methods. With such consideration, Tim Fernando's research work concluded that the superposition theorem is the same as MSO conjugation. Thus, this theorem is implied over constructed automata to extract projections via Allen composition. The next section of our implementation focused on assigning probabilities to the sequence of possible Allen relations unfolded over a period. Initially, uniform probabilities are assigned, and it is evident from the calculated probability that longer relations probabilities are smaller than shorter relations. This contradicts Allen's theory of probabilities over relations categorized as Long, Medium, and Short. James Allen state that the longer the relations are, the more probable they are. This paper extends beyond this contradiction and calculates the conditional probability for the events using the string count by assigning a uniform probability to each Allen relation. Finally, we talk about the implementation of Gap over strings using superposition theorem framed under MSO logic.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The research work began by initially analyzing the temporal events in the form of points or an interval in the real world as represented by James Allen [2]. Interpretation and construction of such events in Natural Language Semantics require representing them in the form of strings. Thus, it helps us to capture the in-depth concepts related to 13 relations defined by Allen. The string representation of events can be achieved in two ways, namely border representation and interior representation (Stative representation) as described by Dowty [3]. Simultaneously, research works are made under First-Order Logic, Second Order Logic, and on a restricted version of Second-order logic i.e. MSO upon which Tim Fernando's superposition theorem [4] was built. This resulted in a set of programs to extract finite temporality strings based on Future to Past/ Past to Future of the temporal relations that tend to hold between events expressed in text. To study further, our research work concentrated on the projections via the Allen Transitivity table which depicts all possible relations that can be derived from a given set of relational events.

To further study and use temporal relations, a formal semantic framework based on "type" theory titled TTR proposed by Robin Cooper was studied. This paper mainly focuses on representing the temporality strings which represent timelines of events in the form of finite automata defined inside TTR with the help of superposition theorem, as well as implementing projection via Allen composition. Later, research methodology was framed to extract probabilities associated with events as rolled out in time. At last, the intention of the thesis deviates from what is typical by concentrating more on the gap implementation of strings. To conclude, even though we fell short of some of our initial research plans like representing the developed framework in python-format using pyTTR, adding new weights to the relations, and calculating probabilities using the formula provided in Markov Logic Network, we managed to construct a state of art framework to represent actions and events along with their timeline probabilities was the major contribution towards the field of language semantics.

1.1 Objective

This paper focuses on 4 main objectives or research questions

- To represent string theory of events using finite automata within TTR and apply the methods followed in constructing the 13 Allen relations based on superposition theorem over the constructed automata.
- To implement projection over any two Allen relation paths, singled out from the constructed automata.
- To calculate the conditional probability associated along with each path from any state to the terminal state within the automata constructed.
- Gap implementation: To form a set gap (L, L') of strings representing minimal reasons to infer string (L) from string (L').

The first two objectives are obtained by combining the concepts of temporal relations, constraints attached, and the TTR framework proposed by James Allen and Robin Cooper. In addition, the concept of superposition over strings represented in the form of automata as structured by Tim Fernando using Monadic Second Order logic act as an anchor to combine the above-mentioned works.

To attain the third objective, the string count from the state of interest to the final terminal state was taken along with the string count of its parent state to the terminal state by considering uniform probability distribution over the selected path.

The final objective is achieved by analyzing the role played by MSO in constructing regular languages with the help of constraints as described by Büchi, Elgot, Trakhtenbrot in their research work [5], [6], and [7]. Thus, our research work concentrates both on MSO logic for automata as well as for regular languages represented in the form of strings.

1.2 Motivation

The entire research work was built upon the seed of "String Theory of Events" which was discussed in parts under this research paper [8]. In this theory, Tim Fernando claims that instead of perceiving an event as a static notion, it is also possible to analyse the changes taking place as an event unfolds in the form of strings. Therefore, the motive of this research paper is to apply this claim to capture a series of events in the "Record-based Automata for TTR" framework along with the actions that perceive it. Since these events are observed over a period of time, the next source of inspiration was drawn from the temporal representation of events proposed by James Allen in his paper [2]. This paper extends this theory to find probabilities associated with events described as a state within the finite automata of records. In addition, the superposition theorem over MSO logic as defined by Tim Fernando in his paper [4], acts as the building block for gap implementation.

1.3 Reader's Guide

The structure of the paper is as follows - Chapter 2 discusses the concepts of First-order Logic, Second-order logic and its variant MSO, followed by the role of MSO in constructing regular languages in terms of string and automata. Next sub sessions contain Finite Temporality explaining Allen Relation and Allen Transitivity Table, and various Computational framework tools for Linguistic Semantics including their pros and cons. This chapter also talks extensively about TTR and some basic string operations. Chapter 3 introduces the problem definitions and discusses the implementation methods used to construct "Record-based Automata for TTR" and extracting projection via Allen Composition. It also describes the methodology to find the conditional probabilities for events described inside the proposed framework, and for gap implementation. Chapter 4 concludes the paper by outlining the results and future work.

Chapter 2

Prior Work

2.1 First-order Logic

The First-order logic, also known as first-order predicate calculus, quantifies individuals or non-logical objects present in a sentence as a quantified variable. Additionally, it also makes use of predicates and functions to capture information conveyed in a sentence. Predicates deal with an entity in a domain of disclosure to represent all possible relations between variables. For example, consider a sentence Edison is dead. Here, Edison is an individual variable, and the word dead is a predicate. Functions define the relationship that exists between two or more objects.

Consider the following two sentences,

- Edison is a scientist, and he is dead.
- Tesla is a scientist, and he is dead.

We can represent these two sentences in first-order logic as

$$\exists x(scientist(x) \land dead(x) \tag{2.1}$$

Here, x is a quantified variable representing individuals, and the predicates are represented as a scientist(x) and dead(x). Functions can be used to map the variable x to either other professions like doctor, politician or to other state of life like living and unborn.

2.2 Second-order Logic

The second-order logic represents information present in a sentence as a notion of a domain of disclosure, which is an extension of first-order logic. The major drawback of first-order logic is that it can quantify only over objects. To quantify over relations, the demand for second-order logic becomes inevitable.

Suppose if we want to express the collision of life events involving Tesla and Edison in first-order logic,

$$lifeEvents(Tesla) \wedge lifeEvents(Edison) \tag{2.2}$$

But equation 2.2 fails to convey the exact intended meaning. In order to overcome this scenario we make use of second-order logic. Since it quantifies predicates, this information can be represented by

$$\exists P(collide(P) \wedge P(tesla) \wedge P(Edison)) \tag{2.3}$$

Here, successful quantification was made using second-order predicate 'collide' instead of using an additional predicate 'lifeEventsCollide'.

2.3 MSO (Monadic Second Order Logic)

The Monadic Second Order logic is a logical language with restriction implied over second order logic such that quantifies sentence over sets of no function variables containing only unary predicates. These predicates are often termed as Monadic Predicates (predicates having single argument) equivalent in expressive power to sets.

2.3.1 MSO: Strings

Any string representation can be expressed in MSO logic with a binary relation symbol S (successor) and a unary relational symbol P_σ for each symbol $\sigma \in \sum$ (a set of all alphabets). Such MSO$_{\sum}-$ model for string 'abbc' which is derived from a set $\sum = (a, b, c)$ is given by -

$$\langle D_4, S_4, [P_a], [P_b], [P_c] \rangle \tag{2.4}$$

Where,

D_4 $\langle$ Set representing positions from the given string $\rangle$:= $\{1, 2, 3, 4\}$
S_4 $\langle$Successor$\rangle$:= $\{\langle 1, 2\rangle, \langle 2, 3\rangle, \langle 3, 4\rangle\}$
$[P_a] = \{1\}$
$[P_b] = \{2, 3\}$
$[P_c] = \{4\}$

The MSO$_{\sum}-$ model constructed can be expressed in terms of existential quantifier as

$$\begin{aligned} (\exists x_1)(\exists x_2)(x_3)(\exists x_4)\ P_a(x_1) \wedge P_b(x_2) \wedge P_b(x_3) \wedge P_c(x_4) \wedge \\ S(x_1, x_2) \wedge S(x_2, x_3) \wedge S(x_3, x_4) \wedge \\ \neg(\exists (x)(S(x, x_1) \vee S(x_4, x)) \end{aligned} \tag{2.5}$$

Therefore, equation 2.5 is a fragment of MSO termed as EMSO (Existential Monadic Second Order Logic).

2.3.2 BET (Büchi, Elgot, Trakhtenbrot): Regular Expression = MSO

Büchi, Elgot, Trakhtenbrot in thier research work [5], [6], and [7] made a claim such that for any finite alphabet defined in $\sum$, there exist regular languages (a proper subset of $\sum^+$) derived as a set of strings definable in MSO$_{\sum}-$ models.

This leads to the conclusion of two basic principles,

- For every languages $L \subseteq \sum^{+}$, there is an MSO$_{\sum}-$ sentence φ, such that

$$L = \{s \in \sum^{+} \mid Mod(s) \models \varphi\} \tag{2.6}$$

- For every MSO$_{\sum}-$ sentence φ

$$\{s \in \sum^{+} \mid Mod(s) \models \varphi\} \; is\ regular \tag{2.7}$$

The principles defined are proved by induction on MSO$_{\sum}-$ formulas φ with free variables interpreted in an MSO$_{\sum}-$ model M by a function f given by,

$$M, f \models \varphi \tag{2.8}$$

2.3.3 MSO: A-Reduct (ρ_A)

Reduct in MSO is defined as a restriction over MSO model to extract language consisting of a particular interval of our interest. Therefore, the A-reduct of a regular expression defined in a MSO$_{\sum}-$ model M = $\langle \{1, \ldots, n\}, S_n, (P_a)_a \in \sum \rangle$ is -

$$M_A := \langle \{1, \ldots, n\}, S_n, (P_a)_{a \in A} \rangle \quad (an\ MSO_A - model) \tag{2.9}$$

such that,

$$str(M_A) = \rho_A(str(M)) \tag{2.10}$$

Similarly, a language L $\subseteq Pow(\sum)^{+}$ is A-based if for all s $\in Pow(\sum)^{+}$,

$$s \in L \iff \rho_A(s) \in L \tag{2.11}$$

Therefore, for a given $A \subseteq \sum$ and an MSO$_A$ – formula φ, than language

$$\mathcal{L}_{\sum}(\varphi) := \{s \in (2^{\Sigma})^{*} \mid \rho_A(s) \in \mathcal{L}_A(\varphi)\} \tag{2.12}$$

2.3.4 $\mathcal{R}$ – Extended Regular Expressions

For given a family $\mathcal{R}$ of relations R between strings, the $\mathcal{R}$ extended regular expressions E consist of the regular expressions closed under the following conditions,

- Intersection operation over $E \; and \; E'$
- Complement of the relation E.
- The inverse image R^{-1}E of E under R, given by

$$\langle R \rangle := \{s \mid (\exists s' \in E) sRs'\} \tag{2.13}$$

2.3.5 MSO: Automata

The role of MSO in automata can be described with the help of following figure,

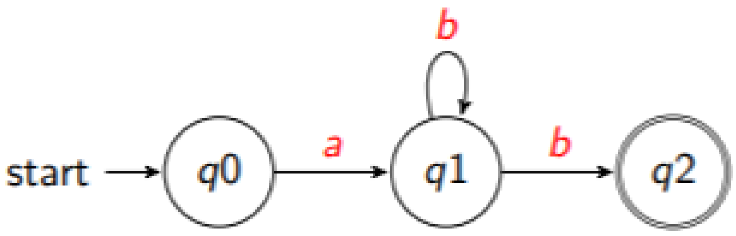


Figure 2.1: An automaton defined over alphabets {a, b} $\in \sum$

The accepting runs are,

$q_0 \overset{a}{\rightarrow} q_1 \overset{b}{\rightarrow} q_2 \in F$

$q_0 \overset{a}{\rightarrow} q_1 \overset{a}{\rightarrow} q_1 \overset{b}{\rightarrow} q_2 \in F$

$\vdots$

regexp ab*b denoting the regular language {ab, abb, …}

From equation: 2.6 and 2.7, the regular language in the form of strings can be represented in MSO model by the following MSO-sentence

$$\begin{aligned} (x)(y)\; P_a(x) \wedge P_b(y) \wedge (\forall z)((z, x) \\ \wedge (z = x \vee P_b(z))) \end{aligned} \tag{2.14}$$

2.3.6 MSO: Accepting Runs

The accepting runs for a constructed automaton over a set of alphabets defined under $\sum$ can be viewed using the following assumptions,

- Subsumption $\trianglerighteq$, is the concept of deriving a set A from a super-set B, Consider the following equation,

$$\alpha_1, \ldots, \alpha_n \trianglerighteq \beta_1, \ldots, \beta_m \Longleftrightarrow n = m \; and \; \alpha_i \supseteq \beta_i \quad for \; 1 \; \leq \; i \; \leq n, \tag{2.15}$$

  Therefore,

$$s \trianglerighteq \rho_A(s) \; for \; all \; s \; and \; A \tag{2.16}$$

- The concept of prefix and suffix is given by,

$$\begin{aligned} s \text{ pref } s' &\Longleftrightarrow \; (\exists u) s = s'u \\ s \text{ suff } s' &\Longleftrightarrow \; (\exists u) s \; = \; us' \end{aligned} \tag{2.17}$$

Consider a given automaton A $= \langle Q, \rightarrow, q_0, F \rangle over \sum$, let

$$\mathcal{L}_+(A) = \{\rho_{\sum}(s) | s \in AcRuns_+(A)\}$$

than,

$$\begin{aligned} \text{AcRuns}_+(A) := \{(a_1, q_1), (a_2, q_2), \ldots, (a_n, q_n) \mid q_{i-1} \xrightarrow{a_i} q_i \\ fori \in D_n \; and \; q_n \in F\} \end{aligned} \tag{2.18}$$

$$= \; L_F \cap \; L_{q0 \rightarrow} \cap L_{\nrightarrow} \cap \; \langle \rho_{\sum} \rangle \; (\sum_{\sigma \in \sum} \sigma)^+ \; \cap \; \langle \rho_q \rangle (\sum_{q \in Q} q)^+ \tag{2.19}$$

Where,

$L_F := \langle \text{suff} \rangle \langle \trianglerighteq \rangle \sum_{q \in F} (Q)$

$L_{q0} := \langle \text{pref} \rangle \sum_{q_0 \xrightarrow{a} q} (a, q)$

$L_{\nrightarrow} := \langle \text{pref} \rangle \langle \text{suff} \rangle \langle \trianglerighteq \rangle \sum_{q \nrightarrow a q'} ((q), (a, q'))$

2.4 Finite Temporality

Time as a quantity plays a crucial role in representing events associated with the living organism in the field of linguistic semantics. The representation varies based on, how the time was perceived by a living organism during an event occurrence. James Allen in his paper [2], states that time can be represented in the form of points as well as intervals. Focusing on humans, they try to analyze events occurred in terms of fine-grained time format depicting precise points in time like yesterday, at midnight, and so on. But this might not be the case with other living organisms. Even though they possess a very minimal understanding of time progression and the interplay of events within it, species like birds repeat the event of migration by start flying north during spring and return to winter grounds during autumn. This signifies that their interpretation of time is mostly seasonal based, which is an example of reference type or intervals. Therefore, shifting our focus on representing events as a collection of time intervals, helped us to model events directly into logic by capturing the changes occurred when a new event starts or an old one disappears. Such a representation of events is termed as finite temporality string, incorporating information related to the timeline of events. The major concern associated with defining such a system is to distinguish time intervals associated with events. This issue can be solved by representing time intervals in the form of strings. Thus, strings defined takes the form of either interior or border format [3], [4]. Further sessions exploit finite temporality strings to derive various relations which are possible between two finite temporal events.

2.4.1 Allen Realtion

Before moving on to temporal relations, it is important to break down Allen's temporal logic system and assumptions made by him. In the system defined by Allen, temporal intervals are considered as primitives, while constraints defined over actions are viewed as a relation between intervals. Allen's work came as a direct interpretation of occurrences, which divided into two classes: processes and events in the field of linguistic semantics at the sentence level. Processes are the characteristic representation of an expression occurring, while events represent the term occurs. On interpreting occurrences, it is important to consider whether the events represented in the form of intervals are closed or open? Without this information, the defined logic becomes conceptually invalid to derive temporal relations that exist between two intervals. Therefore, Allen suggests that this relation should be defined as either open on the lower end and closed on the upper end. To illustrate this assumption, consider an event describing either a person is living or dead. Open intervals imply that there exists a time where the person is neither living nor dead and closed intervals suggest that the person can be both living and dead at the same time. Here, the closed interval does not make any sense. Therefore, a solution of half-opened interval at the upper end allows the relation of meeting between two ends to define a common endpoint. Thus, Allen states that by using temporal intervals and by making use of valid assumptions over the defined interval state (opened or closed) it is easier to avoid irrelevant facts and construct 13 temporal relations between any two-time intervals. These relations are also known as Allen Interval Relation as shown in the Table: 2.1.

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

Table 2.1: Thirteen Allen Relations symbols and its pictorial representations.

2.4.2 Allen Transitivity Table

Further study reveals that Allen Interval relation can also be viewed through the magnifying lens of relational algebra. Before moving forward, it is important to clarify why the notion of time points cannot be used as a base to represent the occurrence of an event. Consider an example given by Allen [2], "We found the letter at 12 noon". It clearly explains the fact that the letter was found at midday. But on inspecting the same sentence through a magnifying lens, we can also extract the spatial information's like "where the letter was found" or "how many letters were there". Therefore, Allen claims that usage of time points in representing events should be avoided. Later, such a claim transformed into one of the supporting pillars to represent actions and events in the form of automata, where the state represents events. The arc between the state describes the actions that occurred within a defined temporal interval. When a new temporal event defined in the format of an automaton is superposed over the previously defined one, the transitivity rules are implied to extract other temporal relations between these two superposed automaton as seen in the Table: 2.2.

Relation between 2 3 Relation between 1 2	<	>	d	di	o	oi	m	mi	s	si	f	fi
<	<	nil	< o m d s	<	<	< o m d s	<	< o m d s	<	<	< o m d s	<
>	nil	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	< oi d m f	>	>	>
d	<	>	d	nil	< o m d s	> oi mi d f	<	>	d	> oi d m f	d	< o m d s
di	< om di fi	> oi di mi di si	o oi d con =	di	o di fi	oi di si	oi di fi	oi di si	di fi o	di	di si oi	< di
o	<	> oi di mi si	o d s	< o m di si	< o m	o oi d con eq	<	oi di si	o	di fi o	d s o	< o m
oi	< o m di fi	>	oi d f	> oi mi di si	o oi d con eq	> oi mi	o di fi	>	oi d f	> mi oi	oi	oi di si
m	<	> oi mi di si	o d s	<	<	o d s	<	f fi eq	m	m	d s o	<
mi	< o m di fi	>	oi d f	>	oi d f	>	s si eq	>	d f oi	>	mi	mi
s	<	>	d	< o m di fi	< o m	oi d f	<	mi	s	s si eq	d	< m o
si	< o m di fi	>	oi d f	di	o di fi	oi	o di fi	mi	s si eq	si	oi	di
f	<	>	d	> oi mi di si	o d s	> oi mi	m	>	d	¿ oi mi	f	f fi eq
fi	<	> oi mi di si	o d s	di	o	oi di si	m	si oi di	o	di	f fi eq	fi

Table 2.2: Transitivity table for 12 Allen Relations excluding 'equal' relation.

2.5 Computational Framework for Linguistic Semantics

The following sections provides an overview of various state of art computational frameworks in the field of linguistics.

2.5.1 Semantic Ontology

Semantics is the study of words, sentences and phrases and the meaning that is drawn from them [9]. An ontology is a collection of relevant topics within a domain and their relationships [10]. So, in linguistics, semantic ontology is defined as study of languages, its syntax and concepts, and their corresponding word meanings. A wide variety of models available to analyze the semantic ontology. One such model is Possible world semantics is a method part of philosophical lexicon used to analyze of intentional phenomena, including modality, conditionals, tense and temporal adverbs, obligation, and reports of informational and cognitive content [11] and lead to development of new applications in the field of computer science and artificial intelligence. But this strategy holds some drawbacks like,

- Inability to finely grain down the information conveyed in an event/sentence.
- Issues in calculating the truth value associated with a sentence representing negations.

2.5.2 Syntax-Semantic Interface

Syntax semantics interface in the linguistic world is the study of how the syntax and semantics interconnects with one other. Syntax studies about the underlying structure how the words and phrases are ordered to form proper sentences. Semantics is the study of the how the words in sentences are interlinked to each other and interpreted, that is how we extract the true meanings from these words. So how syntax and semantics maps with one other is the most fascinating area in linguistics. For example, "I hates him" is grammatically incorrect because there is no connection between the subject and verb, this example is related to syntax whereas "Chomsky's "Colorless green ideas sleep furiously" [12] here the sentence is syntactically correct but semantically incorrect. Such representation paved way to construct HPSG (Head-driven Phrase Structure Grammar) [13], and SCG (Sign-based Construction Grammar) [14] over TFS based approaches. But these well-defined structure fails to cover certain fundamental representational aspects in the domain of linguistics.

Some include.

- The lack of proper binding of entire information conveyed within a sentence.
- The mis clarity prevails within a function definition

2.5.3 Type Theory & Perceptions

Type theory is an age-old concept defined first in the classical model of theoretic semantics [15], [16]. The underlying theory about type hovers over the ideas involving ontology-based classes of object creations to represent truth values, entities, possible worlds, and the entire function between their objects. The successor version of type theory, which is of our interest, is termed rich type theory. This theory was first proposed by Martin Lof in his research work [17]. Later, modifications were made to adapt rich type theory in various fields of linguistics like Natural Language Processing and Knowledge Representation. The rich type theory differs from classic type theory by considering both basic ontological categories and types related to categories of objects.

See figure 2.2, and 2.3, visualizing a part of the environment containing a tree, a human, and a bee.

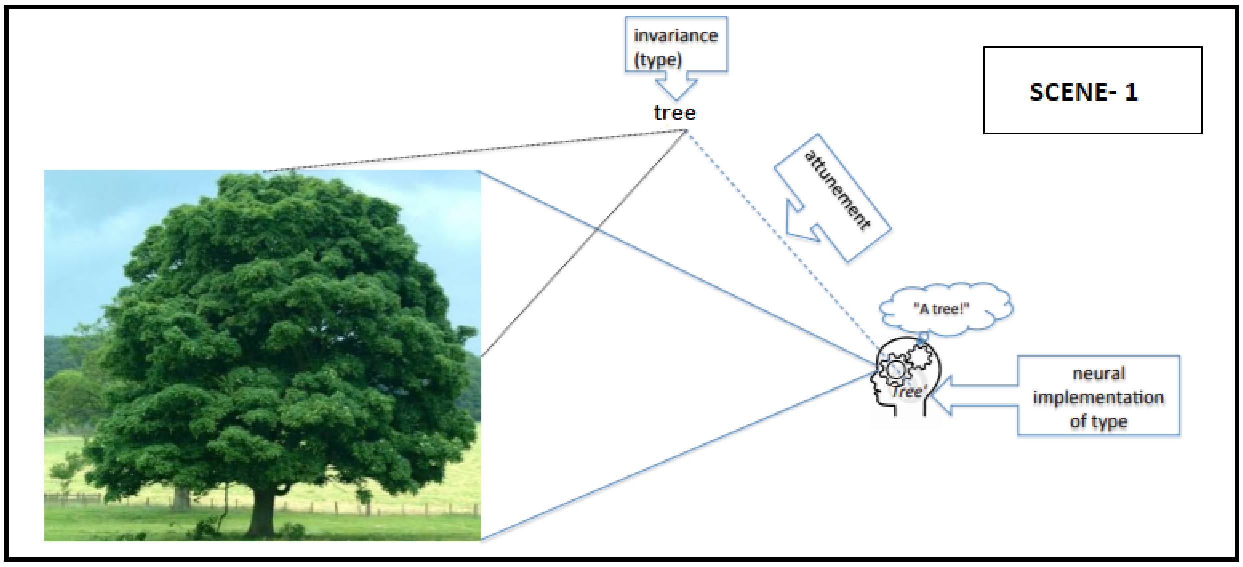


Figure 2.2: Simulation view of seeing a tree by a human

Scene- 1: Figure 2.2, describes the agent(human) makes a judgment that object(tree) is of type Tree based on characteristics green and having branches, in symbols, human: tree.

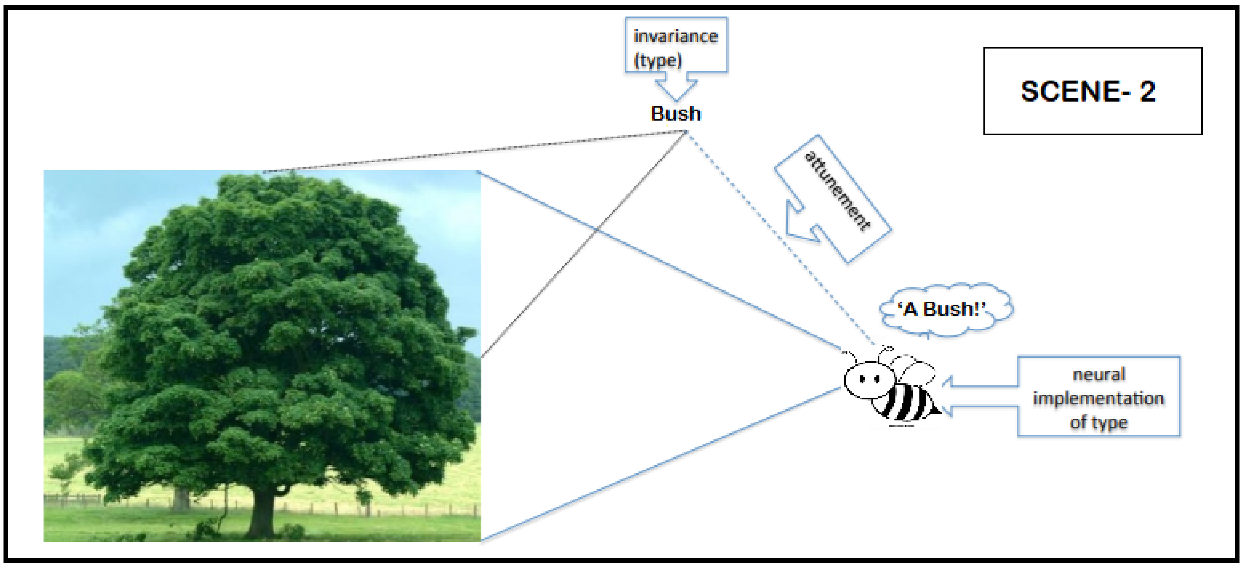


Figure 2.3: Simulation view of seeing a tree by a bee [1]

Scene- 2: Figure 2.3, describes the agent(bee) makes a judgment that object(tree) is of type Bush based on only characteristic green, in symbols, bee: Bush.

From the above-mentioned scenes, it's clear that judgments made by agents were purely based on individual perceptions that one possesses about the objects. Since perceptions vary from one agent to another, the type of objects perceived are varied. Therefore, Gibson in his paper [18], termed type as an invariance. To overcome such variation in perceptions, an agent must be attuned to certain universal patterns that define the object of particular interest.

2.5.4 TTR: Type Theory with Records

The TTR (type theory with records) is a computational framework proposed by Robin cooper in a series of works [19], [20], [21], [8], and [22], discussing its relation to various semantic theories. The defined TTR framework found its application in not only categorizing object types but also in defining the types of situations. The basic fundamental concept involved in TTR is characterizing a system of basic types as a pair consisting of a non-empty set of types, and a function to describe the patterns in which the objects are characterized into a particular type of interest.

In addition, the types defined under TTR are mathematical objects in their own right, they are not just sets of objects. The witness condition associated with type are not quantifiable. For example, Classifiers. One of the main advantage of TTR is that it can allow objects to be part of two or more distinct types i.e A(T1) = A(T2), a: T1 just in case a: T2.

This research work mainly concentrated on two subtypes of the type defined within TTR. One, the basic types, which are used to define objects. These types are used in capturing some of the basic object definition which are universally accepted. The basic types system contains a set of btypes and a function, A, which assigns a set of objects to each of the types in btype. For example, Ind("individual").

$$a : T \text{ iff } a \in A(T) \tag{2.20}$$

Second, the complex types (non-basic types) are constructed from other set of mathematical objects. One kind of complex type is ptypes, which are used to define situations /events. These types are constructed from predicates, and the objects are used as arguments to the predicate. The Predicates come along with an arity which tells you what kind of arguments the predicates have. A system of complex types will contain a set of ptypes, PType, in addition to basic types. Therefore, Ptypes are types based on a set of predicates with their articles. In addition, PType will also contain all the possible ptypes for a given predicate and assigned to the arity for the predicate elsewhere in the system. Consider predicates such as 'born' and 'hit', whose arity relation can be defined using following notion -

$$\begin{aligned} Arity(born) &= \langle Ind \rangle \\ Arity(hit) &= \langle Ind, Ind \rangle \end{aligned} \tag{2.21}$$

Another kind of complex type in TTR is the record type, types which consist of a collection of types indexed by labels. See the following figures 2.4, and 2.5 is a zoomed in version of the tree image in figure 2.2 or 2.3, describing a situation in which the event of an apple falling over newton is occurring. Here, the predicate is fall, the objects considered are apple, and Newton, the agents observing the events are human and bee.

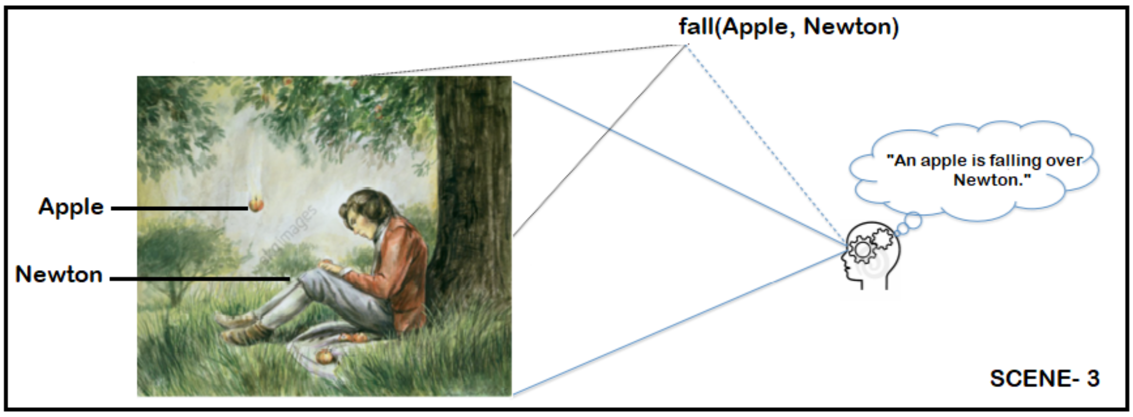


Figure 2.4: Simulation view of seeing an event of apple falling over newton by a human [1].

Scene- 3: Figure 2.4, describes the entire event viewed by an agent human can be defined within the TTR framework with the help of btype and pytype as,

$$\begin{bmatrix} Newton : Ind \\ Apple : Ind \\ e : Fall(Apple, Newton) \end{bmatrix} \tag{2.22}$$

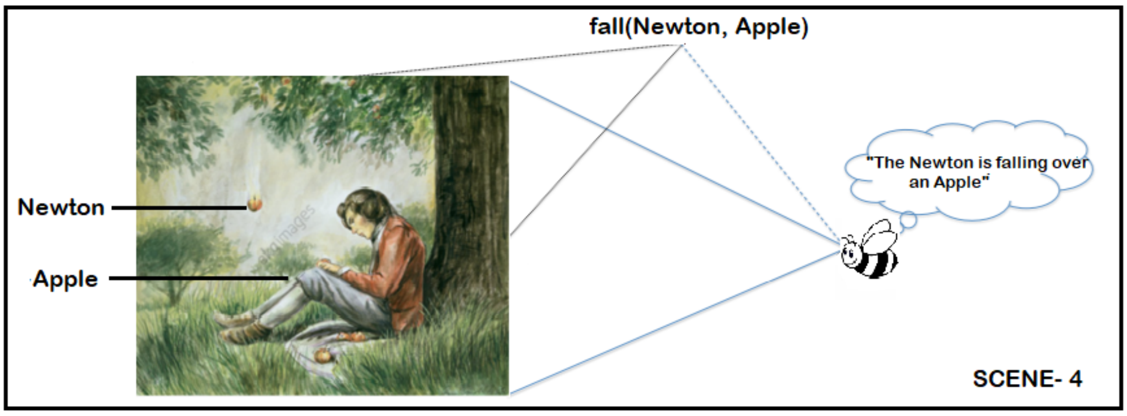


Figure 2.5: Simulation view of seeing an event of newton falling over an apple by a bee[1].

Scene- 4: Figure 2.5, describes what if the bee observes the same event but considered apple as newton and newton as an apple. Then, the equation 2.22 becomes,

$$\begin{bmatrix} Newton : Ind \\ Apple : Ind \\ e : Fall(Newton, Apple) \end{bmatrix} \tag{2.23}$$

To overcome such issues while representing information in TTR, we make use of record types.

$$\begin{bmatrix} Newton : Ind \\ c_{Human} : human(Newton) \\ Apple : Ind \\ c_{Fruit} : fruit(Apple) \\ e : Fall(Apple, Newton) \end{bmatrix} \tag{2.24}$$

2.5.5 String Theory of Events

Tim Fernando in his series of paper [23], [24], [25], and [26], proposed an event or situation cannot be viewed as a single static point because it fails to capture all the changes that are taking place when an event unfolds. Therefore, Tim made a proposal to capture the changes that unfolds within an event frame in the form of strings. This kind of representation helped us to perceive events in the form of regular languages and to implement the theories associated with MSO over it. Later, to incorporate the string theory of events in TTR, the research work made by Hopecraft [27] was considered. Hopecraft, suggested that in formal language theory one can make use of regulartypes: a type of string of objects, which is of the same kind as the strings defined in regular languages, to achieve the goal of representing sequence of actions captured in an event occurrence within the framework of TTR. The two main regular types which is of our interest are ‘a $\frown$ b’ represents concatenation operation between two objects ‘a’ and ‘b’ and T^+ is a type used to represent nonempty string of object type T.

From the figure 2.6 , ‘a’ signifies a person, ‘b’ signifies a dog and ‘c’ signifies a stick. The actions and events captured while playing the game of fetch can be represented in TTR by the following equation.

$$\begin{split} (pick\_up(a,c) \frown attract\_attention(a,b) \frown throw(a,c) \frown run\_after(b,c) \\ \frown pick\_up(b,c) \frown return(b,c,c))^+ \end{split} \tag{2.25}$$

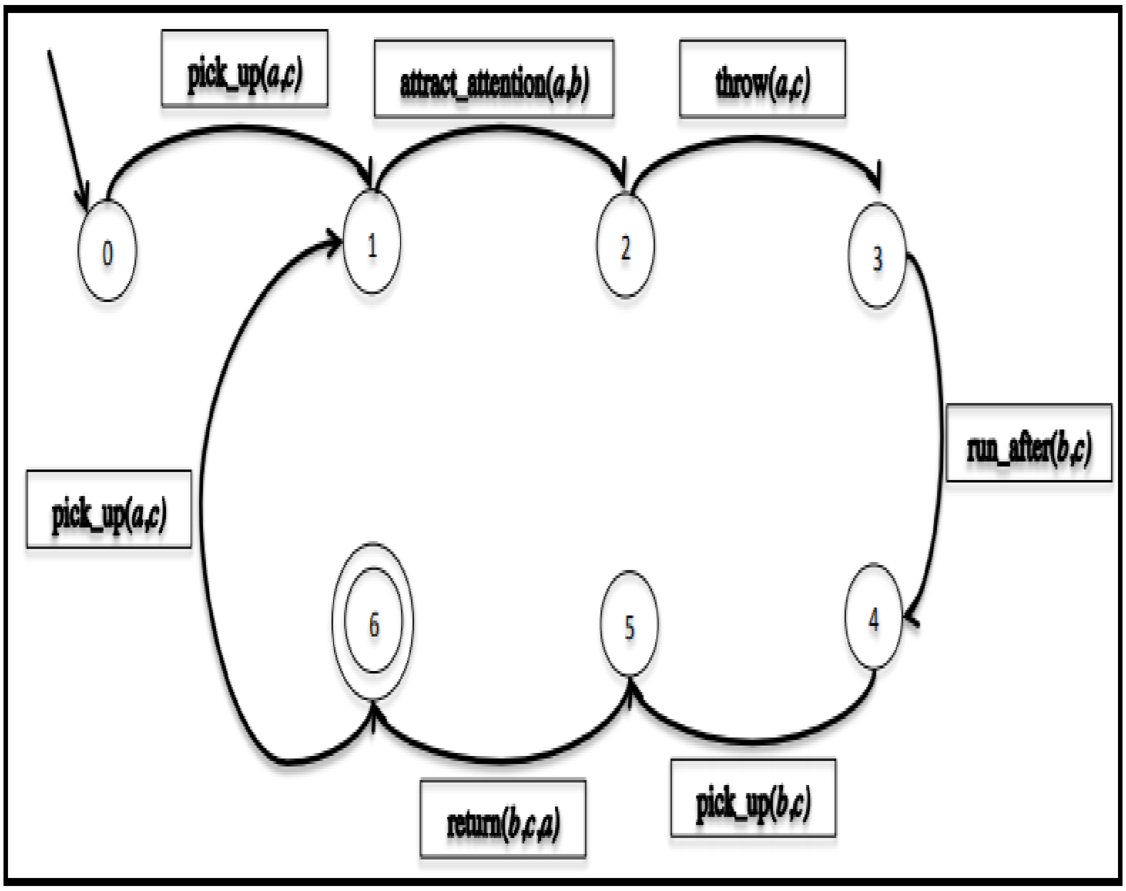


Figure 2.6: An automaton representation of “Game of Fetch”.

2.5.6 Inference from Partial Observation of Events

The next important concept to fuel our research work was the fact of predicting the whole event type to be unfolded by knowing only a part of it. Consider the changes associated with the life events as unborn, living, and dead. Now, a person named ‘A’ was categorized to be in the event category of unborn. Therefore, we can represent the sequence of events that can occur in the life of A like living, and dead as a prediction inside the function defined in the following equation -

$$\left[\lambda_r\right] = \begin{bmatrix} A : Ind \\ c_{human} : human(A) \\ e : Living(A) \frown dead(A) \end{bmatrix} \tag{2.26}$$

$$\left[\text{e: unborn(r.A)}\right]$$

2.6 String Operations

2.6.1 Reduct

The A$-$Reduct of a string ‘S’ is a function to find the intersection of ‘S’ component-wise with string ‘A’. It is denoted by the symbol $\rho_A(S)$.

$$\rho_A(\alpha_1\alpha_2\ldots\alpha_n) := (\alpha_1 \, \cap \, A)\ldots(\alpha_n \, \cap \, A) \tag{2.27}$$

This can be achieved by representing strings in MSO$_A-$ model as described in equation 2.10

2.6.2 Depad

Depad is an operation in which string S of points can be compressed by deleting all occurrences of the empty boxes in S. It is denoted by using the symbol $(d_\Box(S))$. This is given as follows -

$$d_\Box = \begin{cases} d_\Box(s), & \text{if } \alpha \; = \; \Box \\ \alpha d_\Box(s), & \text{Otherwise} \end{cases} \tag{2.28}$$

2.6.3 Projection

Projection is the combined implementation of both reduct and depad operation over string S. The projection execution follows a pattern of performing depad over the function of reduct. It is denoted by using the symbol $d_A(S)$.

$$d_A(S) := d(\rho_A(S)) \tag{2.29}$$

Chapter 3

Design and Implementation

3.1 Problem Definition: Unborn, Living, Dead

Events are closely related to time. Thus, events are categorized into either happened, happening or about to happen based on the tense of action or states observed. The number of events an agent can observe in a given ecosystem are infinite. Therefore, we singled out the stages involved in the life cycle of an organism like unborn, living, and dead as potential events for our study. Later, a finite non-empty set A, the set 3^A of functions from A to Z = {U, L, D} is isomorphic to set of triples (Unborn, Living, Dead) are considered. The defined set A is the union of disjoint subsets containing the event of U, L, and D. This is because, in reality organisms cannot hold the state of unborn, living, and dead at the same time.

Let us consider a person named ‘a’, whose event states can be defined as a finite set E = {U, L, D}. The term A-story was coined to describe all possible life events that can unfold in a’s life. Therefore, A-story is a path from {U, ∅, ∅} to {∅, ∅, D}. For a path to be considered as a valid A-story, it should satisfy all the real world constrains, namely an organism in the state of unborn cannot reach the state of dead directly without being in the state of living. On taking this condition into account, the A-story for person ’a’ can be represented with the help of finite automaton as seen in the figure 3.1-

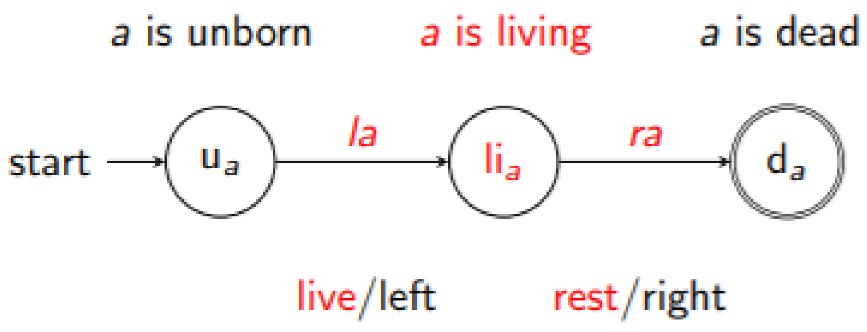


Figure 3.1: An automaton representing A-story of a person named 'a'

Our main motive is to capture the changes associated with life events as described by state and action pair strings in terms of Record-based automata that can be defined within the framework of TTR. Since states and actions describing events are represented in the form of strings and closely related to time, it is possible to incorporate the concept of finite temporality defined by Allen over Record-based automata to extract all the 13 relations. See figure 3.1, it displays a directed graph representation of the possible transition of states for a two-person event A = {a, b}. The 13 unique paths from the start node to the end node describing A-stories corresponding to 13 Allen Relations is described in Table: 3.1 -

Relation	**Path**	**Inverse Relation**	**Path**
a before b	$\boxed{\text{la}}\boxed{\text{ra}}\boxed{\text{lb}}\boxed{\text{rb}}$	b before a	$\boxed{\text{lb}}\boxed{\text{rb}}\boxed{\text{la}}\boxed{\text{ra}}$
a equal b	$\boxed{\text{la, lb}}\boxed{\text{ra, rb}}$	b equal a	$\boxed{\text{la, lb}}\boxed{\text{ra, rb}}$
a meets b	$\boxed{\text{la}}\boxed{\text{ra, lb}}\boxed{\text{rb}}$	b meets a	$\boxed{\text{lb}}\boxed{\text{la, rb}}\boxed{\text{ra}}$
a overlaps b	$\boxed{\text{la}}\boxed{\text{lb}}\boxed{\text{ra}}\boxed{\text{rb}}$	b overlaps a	$\boxed{\text{lb}}\boxed{\text{la}}\boxed{\text{rb}}\boxed{\text{ra}}$
a during b	$\boxed{\text{la}}\boxed{\text{lb}}\boxed{\text{rb}}\boxed{\text{ra}}$	b during a	$\boxed{\text{lb}}\boxed{\text{la}}\boxed{\text{ra}}\boxed{\text{rb}}$
a finishes b	$\boxed{\text{la}}\boxed{\text{lb}}\boxed{\text{ra, rb}}$	b finishes a	$\boxed{\text{lb}}\boxed{\text{la}}\boxed{\text{ra, rb}}$
a starts b	$\boxed{\text{la, lb}}\boxed{\text{ra}}\boxed{\text{rb}}$	b starts a	$\boxed{\text{la, lb}}\boxed{\text{rb}}\boxed{\text{ra}}$

Table 3.1: Thirteen unique paths describing Allen Relations.

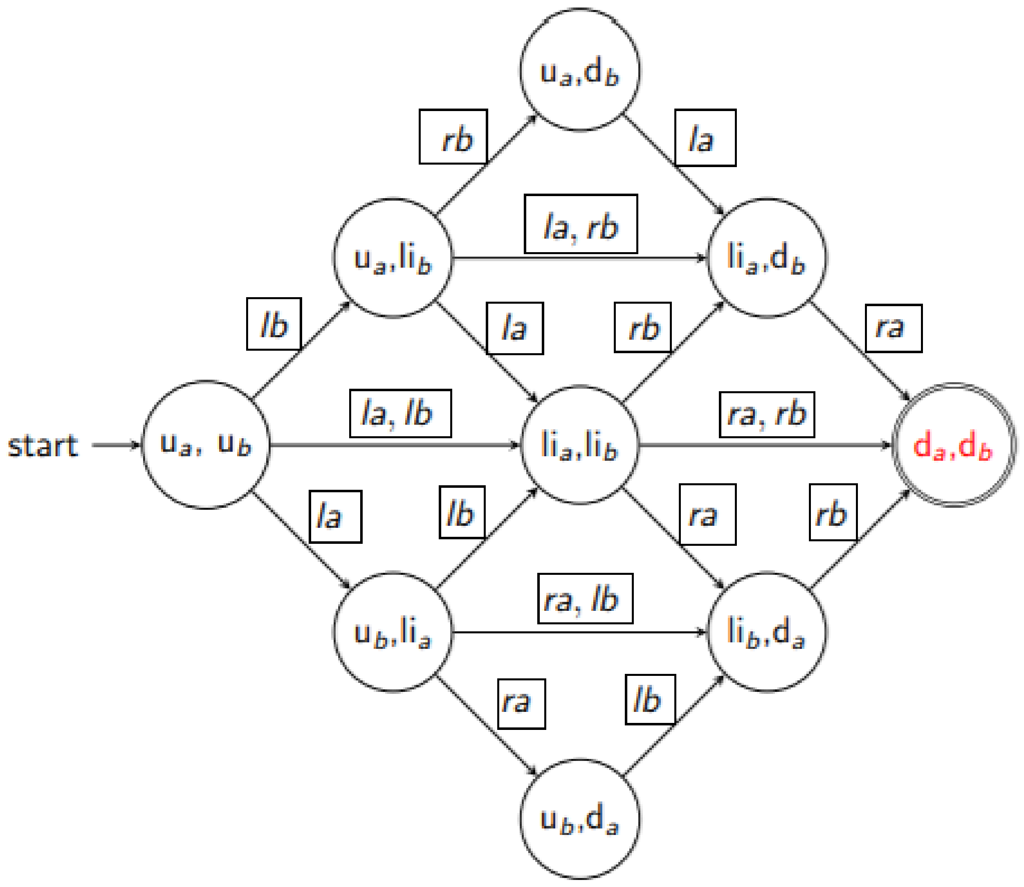


Figure 3.2: Automata representing combined A-stories of two persons named 'a' and 'b'

3.2 Record-based Automata for TTR with Allen Relations

3.2.1 Superposition Theorem: Automata

The A-stories for 'n' number of events are constructed using superposition theorem of automata, which make use of component-wise union based on constrains, followed by depad operation. From listing 3.1, it is apparent that the superposition function takes in the two-object representation of S-word transformed automata, which are to be merged. Within the superposition function, the vocabulary associated with the automata, which is a set containing all possible actions are extracted. In addition, the information about the start states and the Sword object definitions were also extracted. This extracted information is given as input to a component-wise union function responsible for position-based union operation over the vocabularies considered and we also check for satisfaction of relevant constraints. The component-wise union function is a recursive function that checks for all possible paths that can be extracted based on the defined constraints. After component-wise union operation, it is necessary to perform depad operation over the extracted path string, which is done to remove null loop insertions.

Consider two finite automata -

$$\text{M} = \langle \rightarrow, F, q_0 \rangle$$
$$\text{M'} = \langle \rightarrow', F', q_0' \rangle$$

Let M & M' be $\langle \rightsquigarrow, F \times F', (q_0, q_0') \rangle$ with transitions $\rightsquigarrow$ generated alognside a set P of pairs of states including (q_0, q_0') as follows -

$$\frac{P(q,q') \quad q \xrightarrow{\alpha} r \quad q' \xrightarrow{\alpha'} r' \quad \alpha \cap A' \subseteq \alpha' \quad \alpha' \cap A \subseteq \alpha}{P(r,r') \quad (q,q') \overset{\alpha \cup \alpha'}{\rightsquigarrow} (r,r')} \; (cu) \tag{3.1}$$

$$\frac{P(q,q') \quad q \xrightarrow{\alpha} r \quad \alpha \cap A' = \square}{P(r,q') \quad (q,q') \overset{\alpha}{\rightsquigarrow} (r,q')} \; (d1) \tag{3.2}$$

$$\frac{P(q,q') \quad q' \xrightarrow{\alpha'} r \quad \alpha' \cap A = \square}{P(q,r') \quad (q,q') \overset{\alpha'}{\rightsquigarrow} (q,r')} \; (d2) \tag{3.3}$$

For instance,

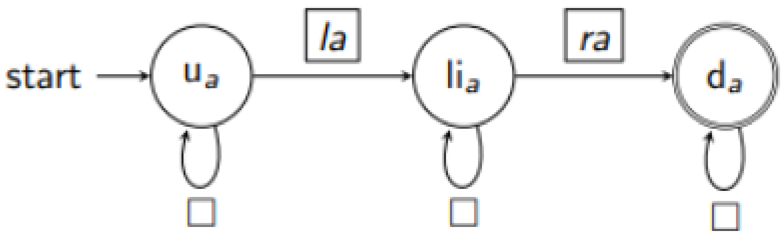


Figure 3.3: An automaton representing A-story of a person ‘a’

From figure 3.3 , one can derive the following regular expression -

$$\boxed{}^* \boxed{\text{la}} \boxed{}^* \boxed{\text{ra}} \boxed{}^* = \boxed{\text{la}}\boxed{\text{ra}} + \boxed{}\boxed{\text{la}}\boxed{\text{ra}} + \ldots$$

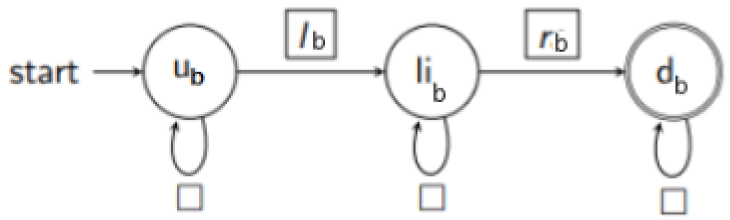


Figure 3.4: An automaton representing A-story of a persons ‘b’

From figure 3.4, one can derive the following regular expression -

$$\boxed{\ }^{*}\boxed{\text{lb}}\boxed{\ }^{*}\boxed{\text{rb}}\boxed{\ }^{*} = \boxed{\text{lb}}\boxed{\text{rb}} + \boxed{\ }\boxed{\text{lb}}\boxed{\text{rb}} + \dots$$

On applying Componentwise union $\sqcup$ between string of the same length, followed by de-pad operations.

$$\boxed{\text{la}}\boxed{\text{ra}} \sqcup \boxed{\text{lb}}\boxed{\text{rb}} = \boxed{\text{la, lb}}\boxed{\text{ra, rb}} \quad \text{‘=’ Relation}$$
$$\boxed{\ }\boxed{\text{la}}\boxed{\text{ra}} \sqcup \boxed{\text{lb}}\boxed{\text{rb}}\boxed{\ } = \boxed{\text{lb}}\boxed{\text{la, rb}}\boxed{\text{ra}} \quad \text{‘mi’ Relation}$$
$$\vdots$$

After $\sqcup$, undo $\square$ -loop insertion

$$\text{depad(s)} := \text{s with } \square \text{ deleted.}$$

Similarly, all other relations of Allen are formed using the superposition theorem.

Listing 3.1: Code snippet capturing important function definitions in superposing two automaton.

```
# Main function definition to superpose two S—words automatons.
def sup(sfa1,sfa2):
    voc1 = sfa1.voc()
    voc2 = sfa2.voc()
    temp = sfa()
    temp.stateD = { 0: {1:sfa1.init, 2:sfa2.init} }
    newp = {(sfa1.init,sfa2.init)}
    done = set()
    while not(newp==set()):
            nextp = set()
            for q in newp:
                morecu = rulecu(sfa1,sfa2,voc1,voc2,q,temp)
                mored1 = ruled1(sfa1,voc2,q,temp)
                mored2 = ruled2(sfa2,voc1,q,temp)
                morep = morecu.union(mored1.union(mored2))
                nextp.update(morep)
            done.update(newp)
            newp = nextp.difference(done)
        for i in temp.stateD:
            if temp.stateD[i][1] in sfa1.final and
            temp.stateD[i][2] in sfa2.final:
                temp.final.add(i)
        # To unwind temp.stateD using sfa1, sfa2
        temp.stateD = unwindSD(temp.stateD,sfa1.stateD,sfa2.stateD)
        # Union operation
        temp.labels = sfa1.labels.union(sfa2.labels)
        return temp

# To add cu to p [side-effects on temp]
def rulecu(sfa1,sfa2,voc1,voc2,q,temp):
        morep = set()
        moretrans = {}
        if (q[0] in sfa1.trans  and  q[1] in sfa2.trans):
            for a1 in sfa1.trans[q[0]]:
                for a2 in sfa2.trans[q[1]]:
                    s1 = sfa1.actD[a1]
                    s2 = sfa2.actD[a2]
                    s = s1.union(s2)
                    # To check for a subset of a set.
                    if contain(s1,voc2,s2) and contain(s2,voc1,s1):
                        for r1 in sfa1.trans[q[0]][a1]:
                            for r2 in sfa2.trans[q[1]][a2]:
                                r = (r1,r2)
                                morep.add(r)
                                temp.actD[s2s(s)] = s
                                # Append operation
                                add2stateD(temp,r)
                                # To decode state values
                                qcode = decodeState(temp,q)
                                di = {s2s(s): {decodeState(temp,r)}}
                                if qcode in temp.trans:
                                    # To merge two dictionary
                                    temp.trans[qcode] =
                                    dicMerge(temp.trans[qcode], di)
                                else:
                                    temp.trans[qcode] = di
        return morep

# To add d1 to p [side-effects on temp]
def ruled1(sfa1,voc2,q,temp):
        morep = set()
        moretrans = {}
        if q[0] in sfa1.trans:
            for a1 in sfa1.trans[q[0]]:
                s = sfa1.actD[a1]
                if contain(s,voc2,set()):
                    for r1 in sfa1.trans[q[0]][a1]:
                        r = (r1,q[1])
                        morep.add(r)
                        temp.actD[s2s(s)] = s
                        add2stateD(temp,r)
                        qcode = decodeState(temp,q)
                        di = {s2s(s): {decodeState(temp,r)}}
                        if qcode in temp.trans:
                            temp.trans[qcode] = dicMerge
                            (temp.trans[qcode], di)
                        else:
                            temp.trans[qcode] = di
        return morep

# To add d2 to p [side-effects on temp]
def ruled2(sfa2,voc1,q,temp):
        morep = set()
        moretrans = {}
        if q[1] in sfa2.trans:
            for a2 in sfa2.trans[q[1]]:
                s = sfa2.actD[a2]
                if contain(s,voc1,set()):
                    for r2 in sfa2.trans[q[1]][a2]:
                        r = (q[0],r2)
                        morep.add(r)
                        temp.actD[s2s(s)] = s
                        add2stateD(temp,r)
                        # To decode state values.
                        qcode = decodeState(temp,q)
                        di = {s2s(s): {decodeState(temp,r)}}
                        if qcode in temp.trans:
                            temp.trans[qcode] =
                            dicMerge(temp.trans[qcode], di)
                        else:
                            temp.trans[qcode] = di
        return morep
```

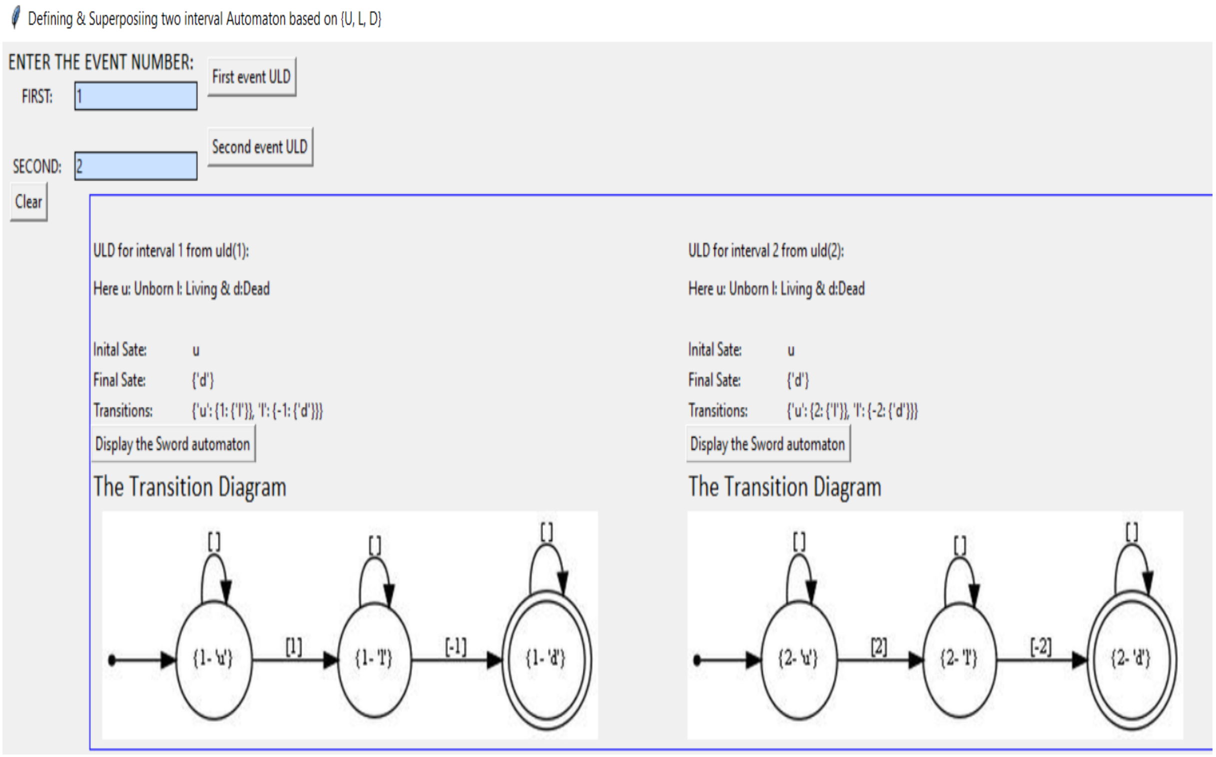
3.2.2 Simulation

The solution implementation was done with the help of a modified version of visual automata package in python [28]. The visual automata package provides a predefined python code to construct circle to represent states, an indicator to represent start and terminal states, and an arc to represent the transition path between states. In addition, it also provides a function to highlight the paths accepted by the automata. The constructed automata can be viewed using tkinter library [29] in the form of a jpeg image. This library also provides a graphical user interface to build input text boxes and buttons to stimulate problems. The implementation of Record-based finite for TTR with Allen Relations can be achieved by following steps -

1. The initial step is to define A-story based on {U, L, D} for two individuals, for this we created two input text boxes that can take numerals as strings. Here, the numbers taken can be considered as names associated with an individual. The A-story constructed over {U, L, D} associated with these two events are obtained by

pressing buttons ‘First event ULD’ and ‘Second event ULD’ respectively.

2. The defined A-story for two events, are not in the form of records. To transform them into the form of record, we made use of S-word transformation function. This function adds record format to the states and a null transition was added over each state. Such transformation can be achieved by clicking the button ‘Display the Sword automaton’. See figure 3.5 -


Defining & Superposiing two interval Automaton based on {U, L, D}

ENTER THE EVENT NUMBER:
FIRST: 1
SECOND: 2

ULD for interval 1 from uld(1):
Here u: Unborn l: Living & d:Dead

```
Inital Sate:     u
Final Sate:      {'d'}
Transitions:     {'u': {1: {'l'}}, 'l': {-1: {'d'}}}
```

ULD for interval 2 from uld(2):
Here u: Unborn l: Living & d:Dead

```
Inital Sate:     u
Final Sate:      {'d'}
Transitions:     {'u': {2: {'l'}}, 'l': {-2: {'d'}}}
```

Figure 3.5: Automata representing S-word transformed A-story for persons ‘1’ and ‘2’

3. The superposition theorem was implied over these two S-word transformed automata, to get the A-story for a finite set consisting of events A = {1, 2}. This event is triggered by using the button ‘Display Superposed automaton of event 1 and event 2’.

4. Next, a text box that can take sequence of states as input along with a function to check for its acceptance over the superposed automaton are defined. The button ‘Check for Acceptance’ was used to trigger the acceptance function as seen in the figure 3.6 -

Consider the following sequence of states represented in the form of string -

State sequence accepted: "{1: 'u', 2: 'u'}.{1: 'l', 2: 'u'}.{1: 'd', 2: 'l'}.{1: 'd', 2: 'd'}"
State sequence rejected: "{1: 'u', 2: 'u'}.{1: 'l', 2: 'u'}.{1: 'd', 2: 'l'}.{1: 'd', 2: 'd'}"

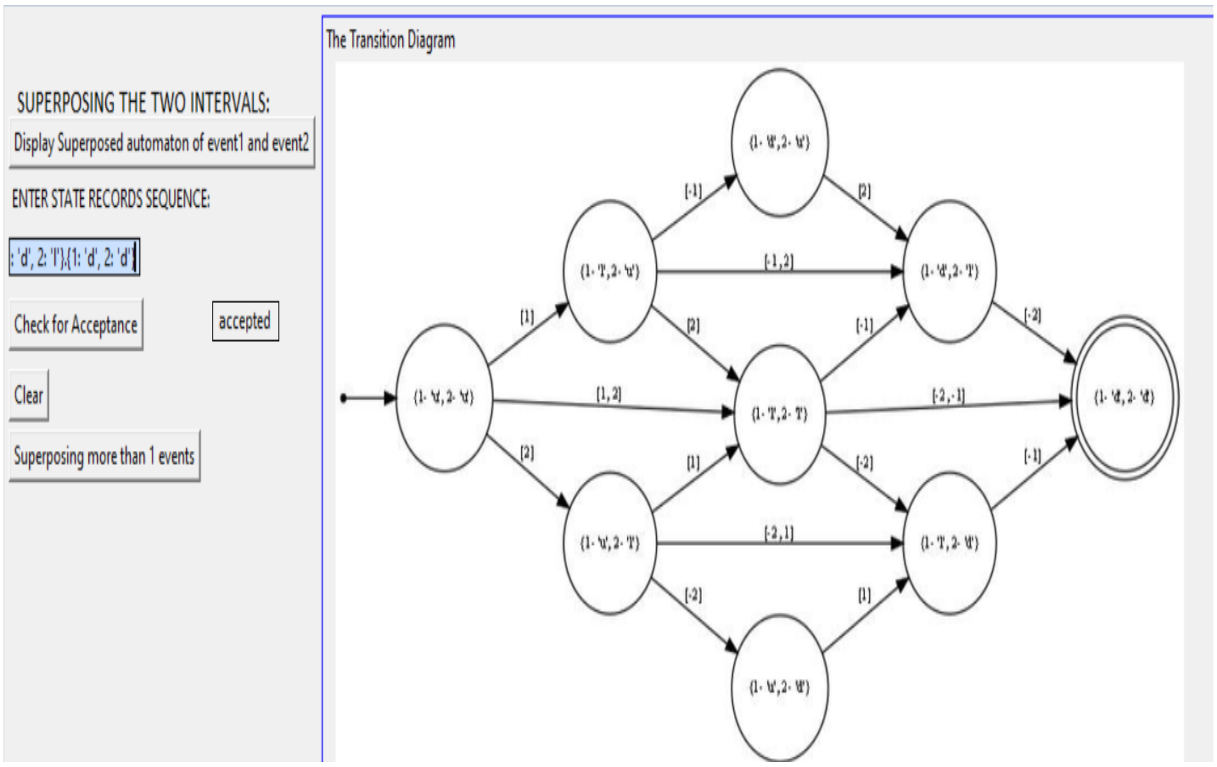


Figure 3.6: Automata representing combined A-stories of persons '1' and '2' along with state acceptance message.

5. To directly superpose more than one event at a time, a new window titled "Automaton for more than two events" was triggered using the 'Superposing more than 1 events' button. The new window was divided into two halves. See figure 3.7, the first half contains two text boxes, one to provide the input for total number of events to be considered, two- provide the state information to visualize a part of the entire constructed automata.

6. The other half contains an input textbox that can take sequence of transition paths to check for its acceptance. If its accepted, the path gets highlighted. If not, a null state was created to signify that the path is rejected.

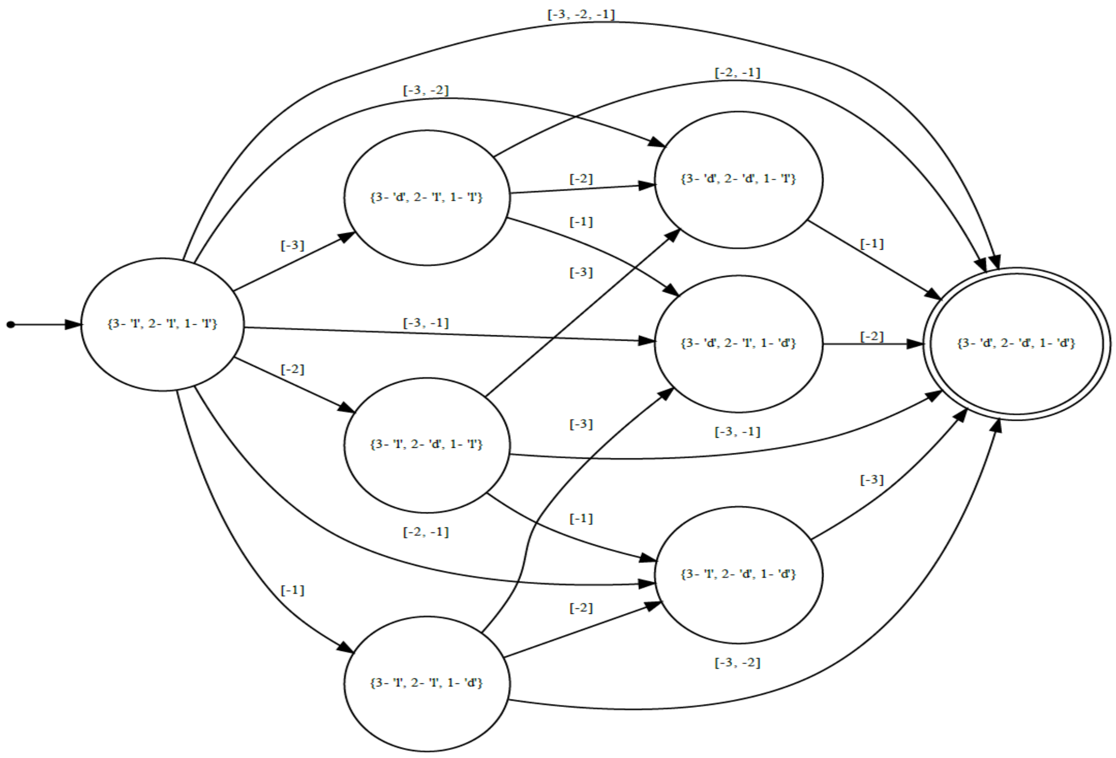


Figure 3.7: Automata representing a part of A-stories for number of events ‘3’ with starting state as {3- ’l’, 2- ’l’, 1- ’l’}.

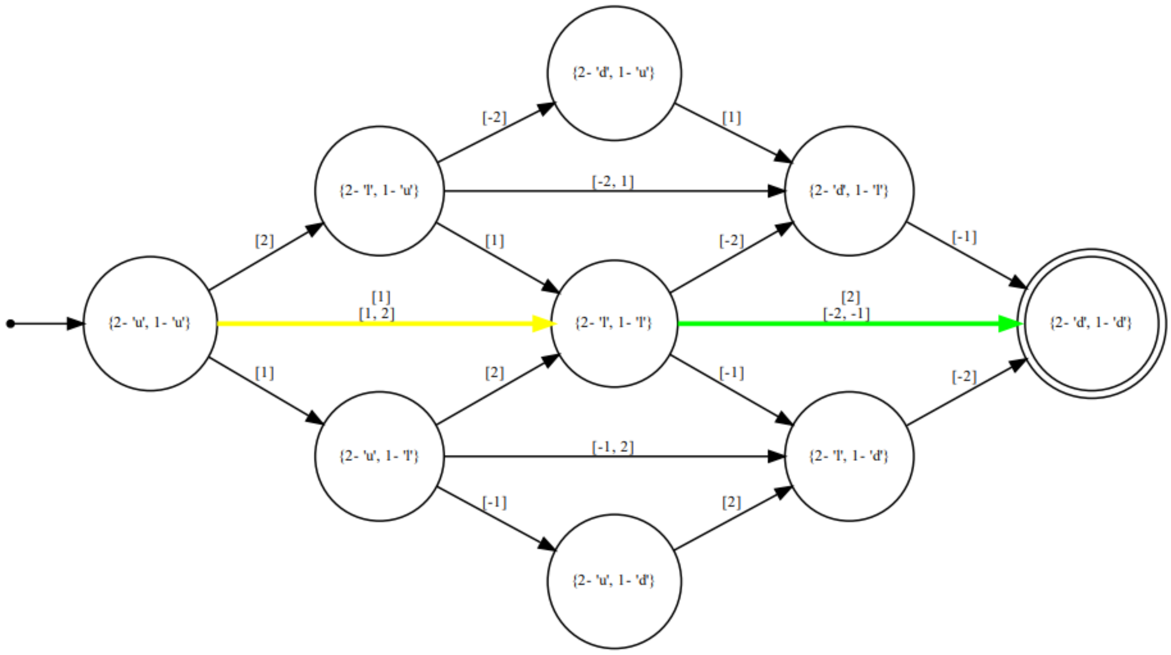


Figure 3.8: Automata containing highlighted path [1, 2].[-2, -1] accepted for number of events ‘2’.

3.3 Projection via Allen Composition

3.3.1 Superposition Theorem: Languages

The transitivity relation between two Allen interval relations can be obtained using the superposition theorem given in listing 3.1.

Consider a given set X, an S-string over X is a string s of non-empty subsets of X.

Let $\mathcal{S}(X)$ be the set of S-strings over X

$$\mathcal{S}(X) = \{depad(s) \mid \in (2^X)^*\} \tag{3.4}$$

Assuming M and M' accept S-strings over A and A' respectively -

$$\text{L(M)} \subseteq \mathcal{S}(A)$$
$$\text{L(M')} \subseteq \mathcal{S}(A')$$

then, the language L(M&M') accepted by M&M' consists of S-strings s over $A \cup A'$ such that,

$$\begin{aligned} \text{depad}(\rho_A(s)) \in \mathcal{L}(M) \\ \text{depad}(\rho'_A(s)) \in \mathcal{L}(M') \end{aligned} \tag{3.5}$$

where $\rho_X(s)$ is s intersected componentwise with X.

3.3.2 Simulation

1. In a new window titled, 'Projection via Allen Composition' two input boxes were provided. One to enter the Allen relation exist between the events 1 and 2. The other one is used to specify the Allen relation that exist between the events 2 and 3. For understanding purpose lets consider events 1, 2, 3, and 4 as the name of persons in real world.

2. 'Display Automaton' button is used to visualize the A-stories for the two Allen relations specified. Later, these two automata are superposed and displayed with the help of superposition theorem for automata and displayed when the button

‘Display Superposition Automaton’ is triggered as seen in the figure 3.9

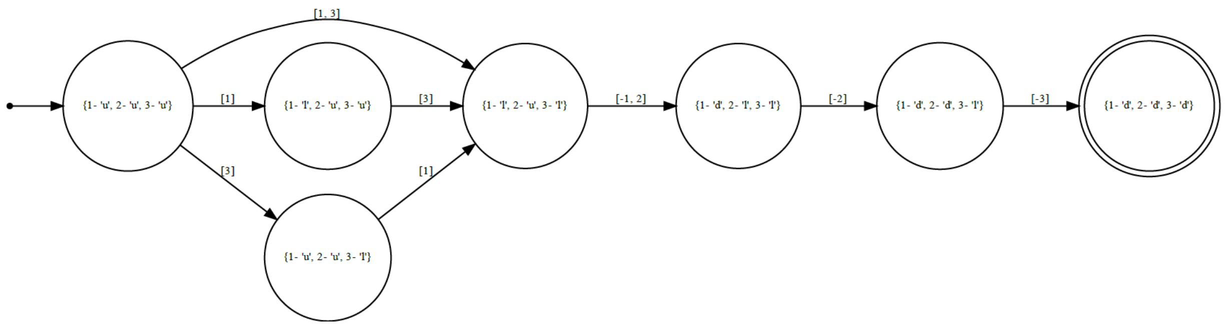
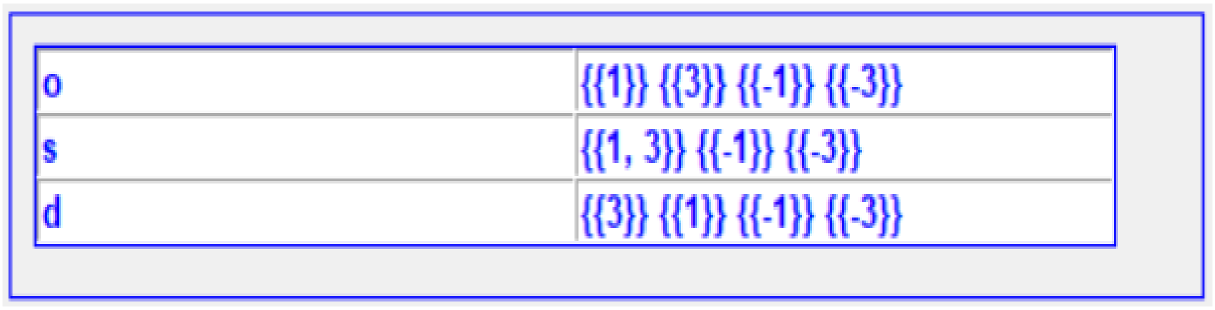


Figure 3.9: Superposed automata representing transitivity relation between events ‘meet’{1, 2} and ‘during’{2, 3}

3. Later, a text box was provided to enter two events of our interest, whose relation must be extracted from the superposed automaton. On providing events of our interest like ‘1 3’, will fetch all possible Allen relation between them like overlaps, starts, during and display them in the form of table. Using this method, one can construct the entire Allen transitivity table.



o	{{1}} {{3}} {{-1}} {{-3}}
s	{{1, 3}} {{-1}} {{-3}}
d	{{3}} {{1}} {{-1}} {{-3}}

Figure 3.10: Table representing all possible relations and their corresponding path between events ‘1’ and ‘3’.

3.4 Probability of Events

3.4.1 Construction of Unborn, Living, Dead

Instead of superposition theorem, it is also possible to construct A-story for ‘n’ events based on certain rules defined over partial observation of events. Consider the A-story for number of events equal to 1.

$$(\{a\},\ \emptyset,\ \emptyset)\ \rightsquigarrow\ (\emptyset,\ \{a\},\ \emptyset) \rightsquigarrow (\emptyset,\ \emptyset,\ \{a\}) \tag{3.6}$$

The constructed {U, L, D} automata based on equation 3.6 contain the left most node (start state) and the rightmost node (terminal/ accepting state). One of the fundamental principles considered was, an A-story associated with a particular event must consider all states justifying two basic facts, one it is not possible to move directly from the state of unborn to dead without being in the state of living. Second, once the event is dead, it is not possible to apply the action of resurrection over it. With these basic principles in mind and by incorporating the theory of partial observations of event, three basic rules were drawn out to construct the A-story based {U, L, D} automata for ‘n’ number of events. Consider A as triple {U, L, D} picks out the unborn, living, and dead under the transitions given by -

$$(U, L, D)\ \rightsquigarrow\ (U^{'}, L^{'}, D^{'})\ \iff\ U^{'} \subseteq U\ and\ L \neq L^{'}\ and\ D \subseteq D^{'} \subseteq L \subseteq D \tag{3.7}$$

From equation 3.7, the three basic rules are defined as following -

- Under transition, those events which are in the group of unborn can stay as such or move to the state of living.
- Those events which are in the state of living can not be continued to stay in the state of living that is it has to be moved to the state of dead.
- The group of dead is the terminal state, and it can only accept events from the state of living.

Based on these rules, A-story for two-person event A = a, b can be represented with the help of automata by labelling the transition with (la, lb) to represent the action of living and (ra, rb) to represent the action of dead as visualized in the figure 3.2 -

3.4.2 Calculation of Uniform Probability

For this we considered the A-story associated with an event number 2 that is A = {a, b}. The constructed automata can contain $P^{|A|}$ states with transition label '.' applied between states. Later, transition probabilities are assigned based on the number of transitions possible for a given state over the transition path from the start state of unborn to the end sate of dead. The A-story built can be represented with the help of right stochastic matrix M, with dimension $P^{|A|} \times P^{|A|}$ and the rows and columns represent the states defined within the constructed A-story. Let us consider a set P_A to hold all the states within the constructed A-story than M_{ij} represent the transition probability of going from state $p_i \in P_A$ to $p_j \in P_A$ and $\sum_j M_{ij} = 1$. Since the transition between states are directed and multiple transitions are avoided the constructed matrix will be sparse and convey less meaning. The assigned probabilities are viewed in the form of automaton in the figure 3.11.

The matrix distribution of uniform probability is given by

$$\begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where P_A, contains states representing the rows and columns in the above defined uniform probability distribution matrix -

$P_A = (\{\text{a, b}\}, \emptyset, \emptyset), (\{a\}, \{b\}, \emptyset), (\{b\}, \{a\}, \emptyset), (\{a\}, \emptyset, \{b\}), (\emptyset, \{b\}, \emptyset), (\{b\}, \emptyset, \{a\}),$
$(\emptyset, \{a\}, \{b\}), (\emptyset, \{b\}, \{a\}), (\emptyset, \emptyset, \{a, b\}), .$

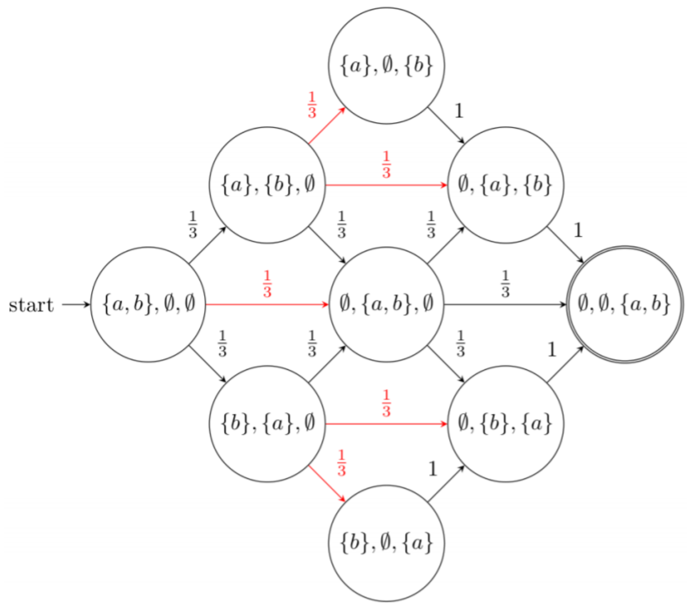


Figure 3.11: A-stories of two events {a, b} with uniform probability distribution.

Since, each path from the start state to the end state represent finite temporality strings based on Allen relation, it is possible to calculate the probabilities associated with each relation. Therefore, from figure 3.11, the arcs in red are used to classify paths from $(\{a,b\},\emptyset,\emptyset) to (\emptyset,\emptyset,\{a,b\})$ in 2 flavours, assuming transition probabilities are assigned uniformly (either 1 or 1/3 for case A = {a, b})

- 7 with a red arc: probability $(\frac{1}{3})^2 = \frac{1}{9}$ for relations bi, mi, s, eq, si, m, b
- 6 with no red arc: probability $(\frac{1}{3})^3 = \frac{1}{27}$ for relations oi, f, d, di, fi, o

3.4.3 Calculation of Conditional Probability

Further study related to probabilities, are carried out by taking number of different strings possible from a given state as a parameter. For instance, from the start state in the figure 3.2, the total number of ways one can reach the terminal state is 13. Here, number of ways signifies the total number of possible strings. Therefore, the probability associated with the start state can be given as 1/13.
Similarly, we considered $t(q, q^{'})$ be the transition probability between two states q and q' given by $p(next(Q) = q^{'}|Q = q)$ as -

$$\text{t}(\square, \text{x}) = \begin{cases} \frac{5}{13} & \text{for } x \in \boxed{\text{la}} + \boxed{\text{lb}} \\ \frac{3}{13} & \text{for } x = \boxed{\text{la, lb}} \end{cases}$$

$$\text{t}(\boxed{\text{lb}}, \text{x}) = \begin{cases} \frac{1}{5} & \text{for } x \in \boxed{\text{la, lb, rb}} + \boxed{\text{lb, rb}} \\ \frac{3}{5} & \text{for } x = \boxed{\text{la, lb}} \end{cases}$$

$$\text{t}(\boxed{\text{la, lb}}, \text{x}) = \begin{cases} \frac{1}{3} & \text{for } x \in \boxed{\text{ra, rb, la, lb}} + \boxed{\text{ra, la, lb}} + \boxed{\text{rb, la, lb}} \end{cases}$$

We recover string probabilities $\frac{1}{13}$ by multiplying $t(q, q^{'})^{'}s$

For instance, consider relation '$(a = b)$' has probability

$$t(\Box, \boxed{\text{la, lb}})t(\boxed{\text{la,lb}}, \boxed{\text{la,lb,ra,rb}}) = \frac{3}{13} \times \frac{1}{3}$$

which is nothing but $\frac{1}{13}$

Similarly, consider relation '$(b < a)$' has probability

$$t(\Box, \boxed{\text{lb}})t(\boxed{\text{lb}}), \boxed{\text{lb,rb}})t(\boxed{\text{la, lb,rb}}t(\boxed{\text{la,lb,rb}}, \boxed{\text{la,lb,ra,rb}}) = \frac{5}{13} \times \frac{1}{5} \times \frac{1}{1} \times \frac{1}{1}$$

which is also equal to $\frac{1}{13}$. Therefore, the conditional probability defined has an uniform distribution over all the 13 Allen relation paths that exist within the constructed automaton over A-story containing {a, b}

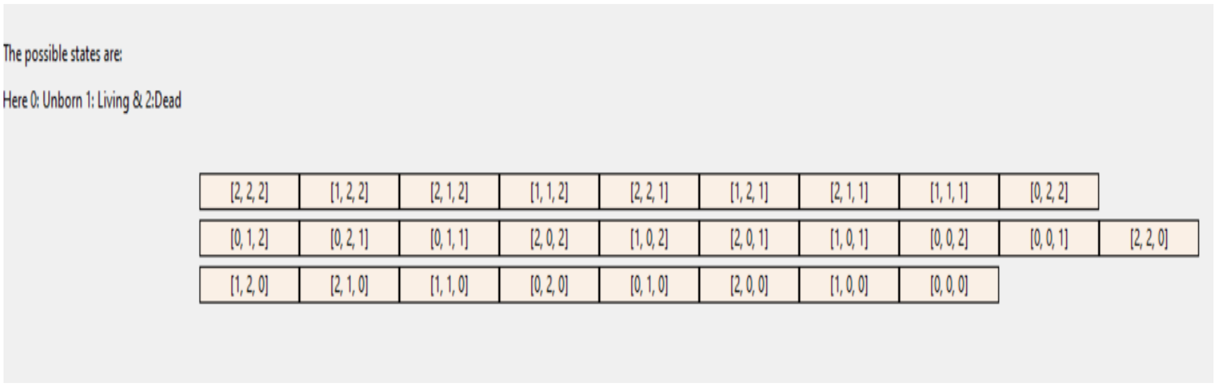
Listing 3.2: Conditional Probability calculation between two successive states within the {U, L, D} automaton.

```
# To retieve & Display the conditional Probability
# between the given two states:
def retrieveProbability(root):
    clearCanvas(root)
    state1 = childState.get()
    state2 = parentState.get()
    n = totalEvents.get()
    ans = transPr(state2,state1,n)
    breakValue.set("The Conditional probability between {} and {}
    is given by:".format(state1,state2))
   #label
    Label(root, textvariable = breakValue).place(x = 0, y = 25)
    Label(root, text = ans, font="Calibri 20").place(x=5, y=50)
# To calculate the conditional probability
# between two successive states/events in the automaton.
def transPr(q0,q1,n):
    temp = count(n)
    if q1 in str(temp[q0]):
        c0 = temp[q0][0]
        c1 = temp[q1][0]
        return(str(c1) + "/" + str(c0))
    else:
        return(0)
```

3.4.4 Simulation

For simulation purpose we made use of Tkinter library with the modified version of visual-automata package.

1. In the window titled "Conditional Probability Calculation for a given number of events in {U, L, D}, a text box input was provided to enter a numerical value greater than 1 specifying the total number of events to be considered in defining {U, L, D} based A-story. On clicking the 'Get the list of states in the automaton' button, one can produce a list of states possible the specified number of events. For n = 3, in total there are 27 states possible, which is given by figure 3.12 -



The possible states are:

Here 0: Unborn 1: Living & 2:Dead

```
[2, 2, 2]  [1, 2, 2]  [2, 1, 2]  [1, 1, 2]  [2, 2, 1]  [1, 2, 1]  [2, 1, 1]  [1, 1, 1]  [0, 2, 2]
[0, 1, 2]  [0, 2, 1]  [0, 1, 1]  [2, 0, 2]  [1, 0, 2]  [2, 0, 1]  [1, 0, 1]  [0, 0, 2]  [0, 0, 1]  [2, 2, 0]
[1, 2, 0]  [2, 1, 0]  [1, 1, 0]  [0, 2, 0]  [0, 1, 0]  [2, 0, 0]  [1, 0, 0]  [0, 0, 0]
```

Figure 3.12: All possible states for A-stories of three events {a, b, c}

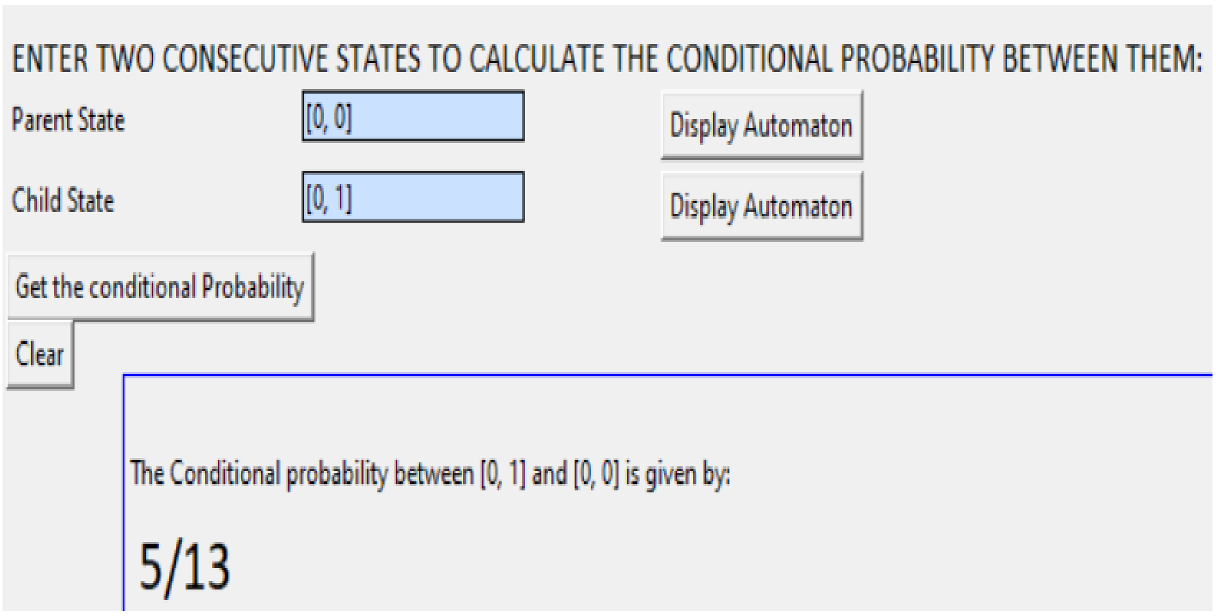
2. Two more text boxes named 'Parent State' and 'Child State' are provided to enter the states between which the probability must be calculated. The 'Display Automaton' button displays the total possible paths to reach the terminal state from the

state specified in the above two text boxes.

3. 'Get the Conditional Probability' button can be triggered to get the probability that exist between the states captured in the above two text boxes.

   For number of event 2, the parent state = [0, 0] signifying both 'a' and 'b' are unborn.
   Child State = [1, 0] signifying 'a' is 'born' and 'b' is 'unborn'.

   The probability is calculated as 5/13 as seen in the figure 3.13. Because, the number of path available from state [0, 0] to terminal state of both 'a' and 'b' died i.e. [2, 2]. Similarly, there are 5 unique strings available from state [1, 0] to the terminal state [2, 2].


ENTER TWO CONSECUTIVE STATES TO CALCULATE THE CONDITIONAL PROBABILITY BETWEEN THEM:

Parent State [0, 0]

Child State [0, 1]

The Conditional probability between [0, 1] and [0, 0] is given by:

5/13

Figure 3.13: The conditional probability calculation between any two successive states inside the {U, L, D} automata.

3.5 Gap Implementation

3.5.1 Superposition Theorem: String

From the listing 3.2, the main superposition function takes in two strings, which are to be merged. In addition, the vocabularies, which is the union of all the attributes in the inputted strings are also provided as an argument. Over this, component-wise union operations were performed in a recursive manner based on certain constraints. All the string combinations that satisfy the constraints are outputted. In general, the superposition algorithm defined will produce 13 strings representing the 13 Allen Relation, which is possible between two events represented in the form of strings. Getting such an output, completely depends upon the combination of input string considered. This theorem acts as the base for gap implementation between two strings L and L'.

While computing superposition, we generate

$$\&_{A,A'} \subseteq \mathcal{L}_A \times \mathcal{L}_{A'} \times \mathcal{L}_{A\cup A'}, \text{ so that}$$
$$L\&_{A,A'};\ L^{'} = \{s^{''} \mid (\exists s \in L)(\exists s^{'} \in L^{'})\&_{A,A'}(s, s', s'')\}$$

Consider, $\alpha \subseteq A, \alpha^{'} \subseteq A^{'}$(making the side conditions vacuous if $A^{'} = \emptyset$ and we use following rules s0, s1, d1, d2 to implement superposition upon which gap implementation was built.
The inductive rule s0 and s1, are used to implement superposition (&) of strings s, s', and s" based on Preposition 4 stated in [4].

$$\frac{}{\&_{A,A'}(\in, \in, \in)} \quad (s0) \tag{3.8}$$

$$\frac{\&_{A,A'}(s, s^{'}, s^{''})}{\&A, A'(\alpha s, \alpha' s', (\alpha \cup \alpha')s''} \quad \alpha \cap A^{'} \subseteq \alpha^{'} \quad \alpha^{'} \cap A \subseteq \alpha \quad (s1) \tag{3.9}$$

The superposition strings obtained will result in lockstep, which can be solved by rules d1 and d2 based on shuffling principles.

$$\frac{\&_{A,A'}(s, s^{'}, s^{"}}{\&A, A'(\alpha s, s^{'}, \alpha s'')} \quad \alpha \cap A^{'} = \emptyset \quad (d1) \tag{3.10}$$

$$\frac{\&_{A,A'}(s, s^{'}, s^{''}}{\&A, A'(s, \alpha' s', \alpha' s'')} \quad \alpha^{'} \cap A = \emptyset \quad (d1) \tag{3.11}$$

Listing 3.2: Code snippet containing superpose(String1, String2, Voc1, voc2)

```
def superpose(string1, string2, voc1, voc2):
    if len(string1) == len(string2) == 0:
        return [string1]
    else:
        temp = []
        if len(string1) > 0:
            h1 = string1[0]
            t1 = string1[1:]
            h1v2 = h1.intersection(voc2)
            if len(h1v2) == 0:
                for res in superpose(t1,string2,voc1,voc2):
                    temp.append([h1] + res)
            if len(string2) > 0:
                h2 = string2[0]
                t2 = string2[1:]
                h3 = h1.union(h2)
                if h1v2.issubset(h2):
                    h2v1 = h2.intersection(voc1)
                    if h2v1.issubset(h1):
                        for res in superpose(t1,t2,voc1,voc2):
                            temp.append([h3] + res)
        if len(string2) > 0:
            h2 = string2[0]
            t2 = string2[1:]
            h2v1 = h2.intersection(voc1)
            if len(h2v1) == 0:
                for res in superpose(string1,t2,voc1,voc2):
                    temp.append([h2] + res)
        return temp
```

Calculating gap of string L and L'

We form gap(L, L') in three steps, which we can mechanize provided L and L' are finite.

- To form the superposition

$$L\&L' := \{s^{''} \mid (\exists s \in L)(\exists s^{'} \in L^{'}\&(s, s^{'}, s''))\} \tag{3.12}$$

- In this step, we calculate the projections that, superposed with L', project to L.

$$\begin{aligned} p(L, L^{'}) \;&:=\; \{d_A(s^{''}) \mid s^{''} \in L\&L^{'} \; and \\ A \;\subseteq\; voc(s^{''}) \; and \; (\forall s^{'} \;\in\; L^{'})&(\forall r \; s.t \; \&(d_a(s^{''}, s', r)) \; r \; projects \; to \; L\} \end{aligned} \tag{3.13}$$

  where "r projects to L" means $(\exists s \in L)$ $d_voc(s)(r) = s$. Next, for any set $\hat{L}$ of any strings of sets, we collect its minimal strings in

$$min\hat{L} \;:=\; \{s \in \hat{L} \mid not(\exists A \subset voc(s))d_A(s) \in \hat{L}\} \tag{3.14}$$

- To remove superfluous strings from p(L, L'), by implementing,

$$gap(L, L^{'}) := min \; p(L, L^{'}). \tag{3.15}$$

This is because strings describing events are not asymmetric,
gap($\boxed{a}\boxed{c}$, $\boxed{e}\boxed{c}$) $\neq gap(\boxed{e}\boxed{c}, \boxed{a}\boxed{c})$

3.5.2 Simulation

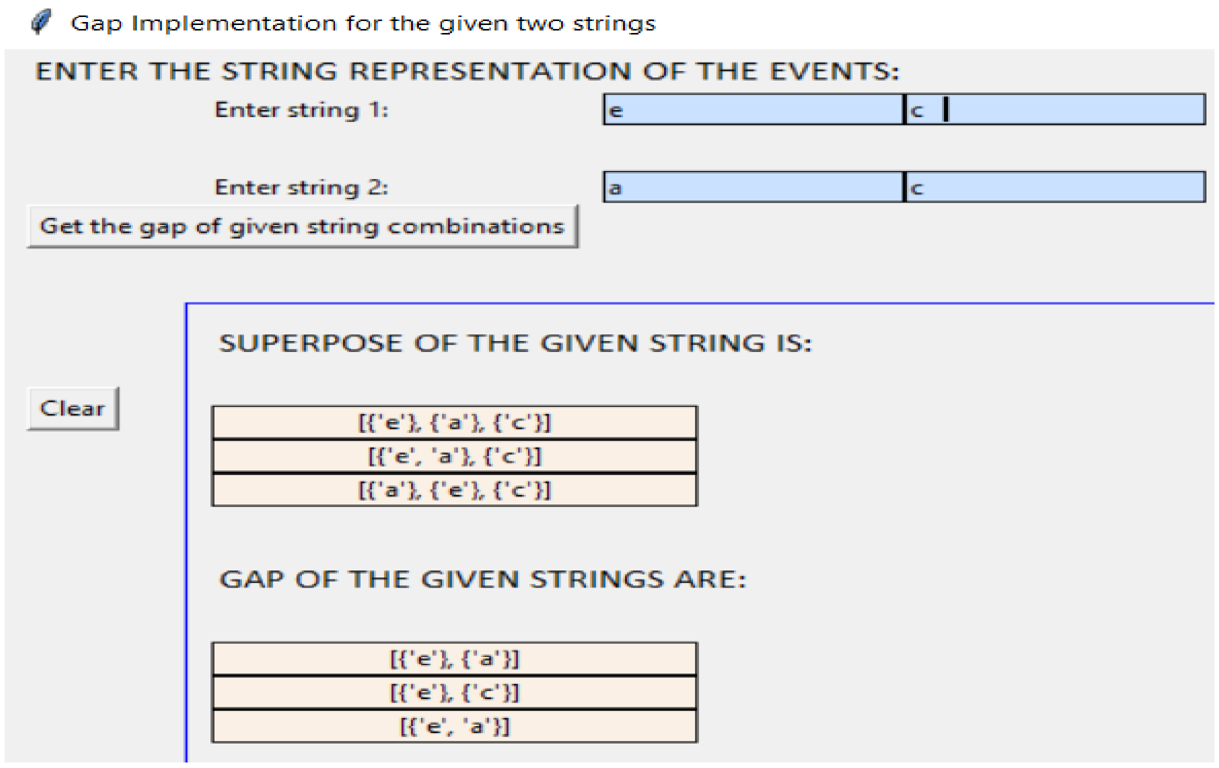
The entire gap implementation was done using tkinter library, here the string combination of L and L' were given as string inputs using two text boxes. The main function of gap implementation was triggered using the button 'Get the gap of given string combination '. On click, the function of superposition was triggered, and all combination of superposed strings are obtained, which is followed by projection function that results in minimal representation of strings to infer L from L'.

For example, To infer $\boxed{a}\boxed{c}$ from $\boxed{e}\boxed{c}$, it is enough to assume one of the three strings

$$\boxed{a,e} + \boxed{a}\boxed{e} + \boxed{a}\boxed{c}$$

whereas to infer $\boxed{e|c}$ from $\boxed{a|c}$, assume one of the three strings as seen in the figure 3.14 -

$$\boxed{a,e} + \boxed{e|a} + \boxed{e|c}$$



Gap Implementation for the given two strings

ENTER THE STRING REPRESENTATION OF THE EVENTS:

Enter string 1: e | c

Enter string 2: a | c

Get the gap of given string combinations

Clear

SUPERPOSE OF THE GIVEN STRING IS:

```
[{'e'}, {'a'}, {'c'}]
[{'e', 'a'}, {'c'}]
[{'a'}, {'e'}, {'c'}]
```

GAP OF THE GIVEN STRINGS ARE:

```
[{'e'}, {'a'}]
[{'e'}, {'c'}]
[{'e', 'a'}]
```

Figure 3.14: Gap Implementation of string $\boxed{e|c}$ and $\boxed{a|c}$

Chapter 4

Conclusions

To conclude, we fell short of our initial research principle, which is to represent actions and events as temporal based finite automata within the framework of TTR. But we were able to define a problem statement over the actions intertangled with the events of life followed by representing them in their temporal form of records within a finite automaton. To do that, extensive research work was made over the three foundational concepts involving constrains over MSO, finite temporality relations and finally, over TTR a computational framework for linguistics. In the course of research work, theories associated with MSO were extensively used in interpretating strings accepted by a finite state machine as regular languages and in defining superposition theorems. To interpret time associated with events temporal relations was studied. Therefore, superposition theorem for automata based on MSO constraints and the temporal relations were used to represent events as a string of records within the TTR framework.

In addition, the superposition theorem for automata was also used to implement the superposition of two languages derived from the automaton of life events. Such implementation helped us to find the transitivity table via Allen compositions. Understanding, the role of MSO in achieving projection over regular language extracted from an automaton was bit ambiguous. Therefore, it took little while to grasp the core concept of MSO than expected.

After constructing the A-story for ‘n’ number of events, the third objective was to define a methodology to calculate the probabilities. For this, a uniform probability was assigned initially based on the number of paths projected outward for a particular state within the A-story automata constructed. Later, a conditional probability was calculated for each state based on the number of unique paths available from that state to the terminal state.

Finally, we visualized these calculated probabilities for various A-stories defined over 'n' number of events.

At last, we implemented a novel concept of gap implementation, which helps us to identify minimal representation of strings. Overall, this project provided a clear overview of how one can interpret actions and events in the form of string-based records constructed over automata and how this well-established proposal can be represented inside the framework of TTR.

4.1 Results

From section: 3.4.2, one of the important insights we got by calculating the Allen relational probabilities was that it contradicts the result obtained from the research work of Tim and Vogel's, which deals with finding probability based on number of intervals. For instance, the probability associated with Allen relations as calculated by Tim and Vogel in their work [30] coincide with the statement proposed by James Allen stating long relation are more probable than short and medium relation in an input string. But our research work contradicts this statement by assigning equal probabilities to both shorter relation like 'equal' and longer relation like 'before'. Even though, the method and parameter considered to calculate the probability varies between our method and the works of Tim and Vogel, the contradiction in the probability obtained holds a significant importance. In addition, our method of calculating probabilities shows some additional contradict results. For example, consider two people, Bolt and Nadal living at the same time. Now, given the fact of living at the same time, the probability of Bolt born before Nadal is more likely than being born at the same time. This is because, born at the same time signifies both are born in the same month (probability is 1/12) or on same date (probability is 1/365). Therefore, on adding up information the probability of being born at the same time decreases dramatically. This contradiction, made us to calculate conditional probabilities as discussed in section 3.4.3 -

4.2 Future Work

This research work covers almost all the aspects of the proposed concept of representing string of events and its associated actions within the framework of TTR. But there are still certain unsolved objectives that needed to be addressed. For instance, the transition paths (signifies actions) are represented using numerical values, which must be in the form of records in order to capture it in TTR. This is one area, which need to be addressed in the future work.

Second area of future work involves in the implementation of Markov Logic Network by assigning weights over the probabilities associated with temporal events. On doing so one can interpret the temporal events captured in Time Bank corpus [31], which holds information about timeline of events outlined in over 183 articles with the help of Time ML schema [32]. For example, consider unborn and dead represent two events, one "The occurrence of earthquake of magnitude greater than 7" followed by event two "The catastrophe that unfolds". In uniform probabilities both the events hold the same probability, which is not appropriate. Therefore, by adding weights we can alter the probability of unborn (earthquake occurrence) to be minimal and the earthquake occurring 'before' catastrophe to be maximum. But if it occurred the probability of dead (catastrophe that unfolds) is maximum and the 'inverse relation of before' between these two events to be minimal.

The final possibility of future work lies within the framework of TTR. Since, the current version of TTR fails to capture the event probabilities, we can define a methodology to capture the probability associated with temporal events within the TTR framework. On moving forward, our research work laid a wonderful platform upon which several concepts can be drawn out in the field of computational linguistics.

Bibliography

[1] Google Inc., “Newton-apple & Bee images.” “`https://www.google.com/`”. Accessed: 25-July-2021.

[2] J. F. Allen, “Maintaining knowledge about temporal intervals,” *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[3] D. R. Dowty, *Word meaning and Montague grammar: The semantics of verbs and times in generative semantics and in Montague’s PTQ*, vol. 7. Springer Science & Business Media, 2012.

[4] T. Fernando, “Temporal representations with and without points,” in *Logic and Algorithms in Computational Linguistics 2018 (LACompLing2018)*, pp. 45–66, Springer, 2020.

[5] J. R. Buchi, “Weak second-order arithmetic and finite automata,” *Journal of Symbolic Logic*, vol. 28, no. 1, 1963.

[6] C. C. Elgot, “Decision problems of finite automata design and related arithmetics,” *Transactions of the American Mathematical Society*, vol. 98, no. 1, pp. 21–51, 1961.

[7] B. A. Trakhtenbrot, “Finite automata and the logic of single-place predicates,” in *Doklady Akademii Nauk*, vol. 140, pp. 326–329, Russian Academy of Sciences, 1961.

[8] R. Cooper and J. Ginzburg, “Type theory with records for natural language semantics,” *The handbook of contemporary semantic theory*, pp. 375–407, 2015.

[9] M. Konstantinovsky, “What does it mean when someone says ‘that is just semantics’?.” “`https://people.howstuffworks.com/semantics.htm`”. [online] Accessed: 15-January-2021.

[10] Ontotext Inc., “What-are-Ontologies.” “`https://www.ontotext.com/knowledgehub/fundamentals/what-are-ontologies/`”. [online] Accessed: 15-January-2021.

[11] J. Perry, "Semantics, possible worlds," *Routledge Encyclopedia of Philosophy. London: Routledge. URL http://www. rep. routledge. com/article U*, vol. 39, 1998.

[12] Wikipedia contributers, "Colorless green ideas sleep furiously." [online] Accessed: 25-January-2021.

[13] I. A. Sag, T. Wasow, E. M. Bender, and I. A. Sag, *Syntactic theory: A formal introduction*, vol. 92. Center for the Study of Language and Information Stanford, CA, 1999.

[14] L. A. Michaelis, "Sign-based construction grammar," *The Oxford handbook of construction grammar*, pp. 133–152, 2013.

[15] R. Montague, "The proper treatment of quantification in ordinary english," in *Approaches to natural language*, pp. 221–242, Springer, 1973.

[16] R. Montague and R. H. Thomason, "Formal philosophy. selected papers of richard montague," *Erkenntnis*, vol. 9, no. 2, 1975.

[17] P. Martin-Löf and G. Sambin, *Intuitionistic type theory*, vol. 9. Bibliopolis Naples, 1984.

[18] J. Gibson, "The ecological approach to visual perception. lawrence erblbaum associates," *Inc., Hillsdale*, 1986.

[19] R. Cooper, "Austinian truth, attitudes and type theory," *Research on Language and Computation*, vol. 3, no. 2, pp. 333–362, 2005.

[20] R. Cooper, "Records and record types in semantic theory," *Journal of Logic and Computation*, vol. 15, no. 2, pp. 99–112, 2005.

[21] R. Cooper, "Type theory and semantics in flux," *Handbook of the Philosophy of Science*, vol. 14, pp. 271–323, 2012.

[22] R. Cooper, "Adapting type theory with records for natural language semantics," in *Modern perspectives in type-theoretical semantics*, pp. 71–94, Springer, 2017.

[23] T. Fernando, "A finite-state approach to events in natural language semantics," *Journal of Logic and Computation*, vol. 14, no. 1, pp. 79–92, 2004.

[24] T. Fernando, "Situations as strings," *Electronic Notes in Theoretical Computer Science*, vol. 165, pp. 23–36, 2006.

[25] T. Fernando, “Finite-state descriptions for temporal semantics,” in *Computing meaning*, pp. 347–368, Springer, 2008.

[26] T. Fernando, “The semantics of tense and aspect,” *The handbook of contemporary semantic theory*, pp. 203–236, 2015.

[27] J. E. Hopcroft, R. Motwani, and J. D. Ullman, “Introduction to automata theory, languages, and computation,” *Acm Sigact News*, vol. 32, no. 1, pp. 60–65, 2001.

[28] L. L. Uberg, “Visual-automata 1.1.1.” “`https://pypi.org/project/visual-automata/`”. Accessed: 15-May-2021.

[29] Wikipedia contributors, “Wikipedia: Tkinter-library..” [online] Accessed: 15-May-2021.

[30] T. Fernando and C. Vogel, “Prior probabilities of allen interval relations over finite orders.,” in *ICAART (2)*, pp. 952–961, 2019.

[31] J. Pustejovsky, P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim, D. Day, L. Ferro, *et al.*, “The timebank corpus,” in *Corpus linguistics*, vol. 2003, p. 40, Lancaster, UK., 2003.

[32] J. Pustejovsky, “Iso-timeml and the annotation of temporal information,” in *Handbook of Linguistic Annotation*, pp. 941–968, Springer, 2017.

Appendix

Github Link: https://github.com/Josh-repository/Dissertation_Work
This repository contains a record of all the processes, codes, and its development timeline tracked from the beginning of the research work.

The programs that are closely assigned with our research work are,

1. Allen_Relation[Future to Past].ipynb & Allen_Relation[Past to Future].ipynb: It contains the code for determining all the Allen Relations.
2. Superposing_Automata(over an alphabet of sets),ipynb: It contains the code to construct A-stories for 'n' number of events based on automata superposition theorem.
3. U,L,D_Automata_Construction.ipynb: It contains the code for implementing our third objective of constructing A-stories for 'n' number of events based on equation 3.7 defined over partial observation of events principle.
4. Gap_Implementation.ipynb: It contains the the code to implement the concept of gap between two strings.
5. Final_ProjectDemo.ipynb: It contains the code to trigger the solution for all the four objectives within a single run-time environment.

To run the above mentioned programs, following steps should be followed -

- Upgrade python version to the latest one (3.9.0) and graphviz = 2.49.0
- Download dependencies using pip3 for -
  tkinter = 8.5
  visual-automata = 1.1.1
  Pillow = 8.3.1
  pdf2image 1.16.0
- Run the programs in jupyter-lab