Strings for Temporal Annotation and Semantic Representation of Events

by

David Woods

A dissertation submitted

in fulfillment of the requirements

for the Degree of

Doctor of Philosophy

University of Dublin, Trinity College

March 2021

Supervised by: Dr Tim Fernando, Dr Carl Vogel

Declaration

I, the undersigned, declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

I, the undersigned, agree to deposit this thesis in the University's open access institutional repository or allow the Library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

David Woods

March 2021

Abstract

Acknowledgements

I would like to thank my parents, Margaret and Graham, who never doubted I would do my best; my brother, Fergus, often a much-needed reminder of normal life; Conor and Katie, who inspired me to start; Adelais, who encouraged me to finish; and all the members of DU Trampoline Club, who gave me a reason to stick around. I hope it was worth it!

My thanks go also to my supervisors: Tim, for being patient with me even when I was struggling, and Carl, who often managed to reassure me that I wasn't going down completely the wrong path.

This research is supported by Science Foundation Ireland (SFI) through the CNGL Programme (Grant 12/CE/I2267) in the ADAPT Centre (`https://www.adaptcentre.ie`) at Trinity College Dublin. The ADAPT Centre for Digital Content Technology is funded under the SFI Research Centres Programme (Grant 13/RC/2106) and is co-funded under the European Regional Development Fund.

Related Publications

During the course of my studies, I was an author on three papers that were accepted for publication, listed below.

- *Towards Efficient String Processing of Annotated Events* (Woods, Fernando, and Vogel, 2017), describing the use of strings to model temporal data such as could be found in text annotated with ISO-TimeML. Presented at the 13th Joint ISO-ACL Workshop on Interoperable Semantic Annotation in Montpellier, France.

- *Improving String Processing for Temporal Relations* (Woods and Fernando, 2018), discussing refinements to the previously described string-based model, such as varied granularity. Presented at the 14th Joint ISO-ACL Workshop on Interoperable Semantic Annotation, colocated with COLING 2018 in Santa Fé, New Mexico, USA.

- *MSO with tests and reducts* (Fernando, Woods, and Vogel, 2019), discussing differing string granularities in the context of tests within Monadic Second Order logic. Presented at the 14th International Conference on Finite-State Methods and Natural Language Processing in Dresden, Germany.

Contents

List of Figures

List of Tables

1 Introduction

This thesis will explore and describe the use of strings as models to represent temporal information—data concerning times and events—for use in computational systems which deal with knowledge-based reasoning in some way. Such systems rely on temporal information in order to perform accurately. For example, a question-answering system that is asked "Will it rain next week?" must be able to locate itself temporally such that it knows what the current time is, the relation between that time and the queried time, and whether raining events have been forecast during the period that it defines using the phrase 'next week'.

2 Relevant Literature

In this chapter, the existing literature related to the major topics of the thesis are reviewed and analysed. [NB[1]] A gap is identified, which the remainder of this work seeks to fill.

2.1 Times and Events

The concept of time has fascinated researchers for millenia, and as such, a great deal of work exists on the topic. What follows here focuses on the formal study of temporality in language.

2.1.1 Allen's Interval Algebra

Much of the present work draws its roots in James F. Allen's work *Maintaining Knowledge about Temporal Intervals* (1983), both directly and indirectly via the many systems which have built upon it in the years since it was first published. In this seminal paper, Allen described a framework for the use of temporal intervals as primitives to represent events and time periods, as opposed to points on the real line. There are four key criteria given as being of primary importance in guiding the design of this framework (1983, p. 833), all of which also feature in the system of representation described in the present work (see § 3.1):

1. The representation should allow for the fact that much temporal information is relative rather than precise.
2. Uncertainty of information should be allowed for, such as when the precise relation between two times is unknown (though constraints on the relation may exist).
3. The granularity of reasoning should be flexible—that is, capable of dealing with years and seconds in the same manner.

[1] Write more here later.

4. Reasoning should assume that states will persist unless there is some evidence to the contrary. In other words, *change* is the marker of progression of time.

Part of the motivation for using intervals rather than points is stated as follows:

> "There seems to be a strong intuition that, given an event, we can always 'turn up the magnification' and look at its structure. ... Since the only times we consider will be times of events, it appears that we can always decompose times into subparts. Thus the formal notion of a time point, which would not be decomposable, is not useful." (Allen, 1983, p. 834)

For example, taking just the event *Going-Home* from the sentence "I went home after work last night", it could be conceptually broken down into sub-events that make up parts of the whole: *Leaving-Work*, *Commuting*, and *Arriving-Home*. Each of these could be again further subdivided, repeatedly, as desired. However, if an event is given a specified time, such as "I went home at midnight", this seems to be a little trickier to deal with—'last night' is plainly referring to an interval of time during which the *Going-Home* event occurs, while 'midnight' appears to be an instantaneous moment of time. Nevertheless, the *Going-Home* event can still be subdivided in the same manner, which shows that even times that appear to intuitively points may be thought of as intervals as well.

Allen makes further argument for the use of intervals as primitive, disallowing zero-width time points, with an illuminating example involving a lightbulb being switched on: there must be an interval of time when the light was off, followed by an interval when it was on, but whether these intervals are open or closed presents another issue. If both intervals are open, then there is some point of time between the two when the light is neither on nor off; however, if both are closed, then there is some time point when the light is both on and off. This fault can be resolved by having the intervals be open at one end and closed at the other, although Allen calls this an artificial solution which "emphasizes that a model of time based on points on the real line does not correspond to

our intuitive notion of time" (1983, p. 834).

However, this argument that temporal intervals are counter-intuitive if they are open on one end and closed at the other has not been universally accepted—for example, the event calculus (Kowalski and Sergot, 1986; Miller and Shanahan, 1999; Mueller, 2008) describes a predicate $\mathit{Initiates}(e, f, t)$ which states that some event e occurs at a timepoint t, and the temporal proposition f is true after t. In the lightbulb example, this is equivalent to using intervals which are open on the lower end and closed at the higher, which can be interpreted as meaning the light is not on at the initial moment of switching it on, but is afterwards. Similarly, Fernando (2018) describes using an open left border and closed right border as a means for event representation.

Allen goes on to formally present and label the thirteen possible relations which may exist between two temporal intervals, which can be thought of as six invertible relations (before, meets, overlaps, during, starts, finishes) and one symmetric (equals). Figure 1, reproduced from Allen (1983, p. 835, Figure 2) shows these relations along with the symbols typically used to abbreviate them, and examples of some events represented graphically as strings of 'X' and 'Y' demonstrating the relations. These are often referred to as Allen's interval relations, or simply the *Allen relations*.

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

Figure 1: Allen interval relations (Allen, 1983, p. 835, Figure 2).

These relations form a cornerstone both within this work (see Table 7) and elsewhere—they are a fundamental part of the specification of ISO-TimeML (Pustejovsky et al., 2010) (see § 2.2.1), the international standard markup language for temporal annotation, where they are used, as one might expect[2], as the possible relation types between a pair of events tagged elsewhere in a document. Freksa (1992) also describes his larger set of *semi-interval* relations in terms of disjunctions of the Allen relations—see also § 3.1.3 and § 4.1.2.

The framework proposed by Allen uses a directed graph as its basis, in which the nodes represent intervals, and the arcs are labelled according to the relation (or, in the case of uncertainty, relations—see Table 1) between the intervals. It is assumed that complete information about the relations in the network is maintained—that is, there are no nodes without an edge between them, as the transitivities between the various relations are to be

[2]Albeit with slightly different labels and omitting *overlaps*—see Figure 3.

computed as necessary—for instance, on the addition of a new interval into the network.

To illustrate, given a graph representing some pair of events X and Y such that X is *before* Y (Figure 2 (a)), a third node representing the event Z is added, with the additional information that Y occurs *before* Z (Figure 2 (b)). The relation between X and Z can then be calculated, using the fact that it has been constrained from thirteen possibilities to just one due to the transitivity rules which apply to the Allen relations—in this case, $X < Y$ and $Y < Z$ results in $X < Z$ (Figure 2 (c))—see also Table 1.

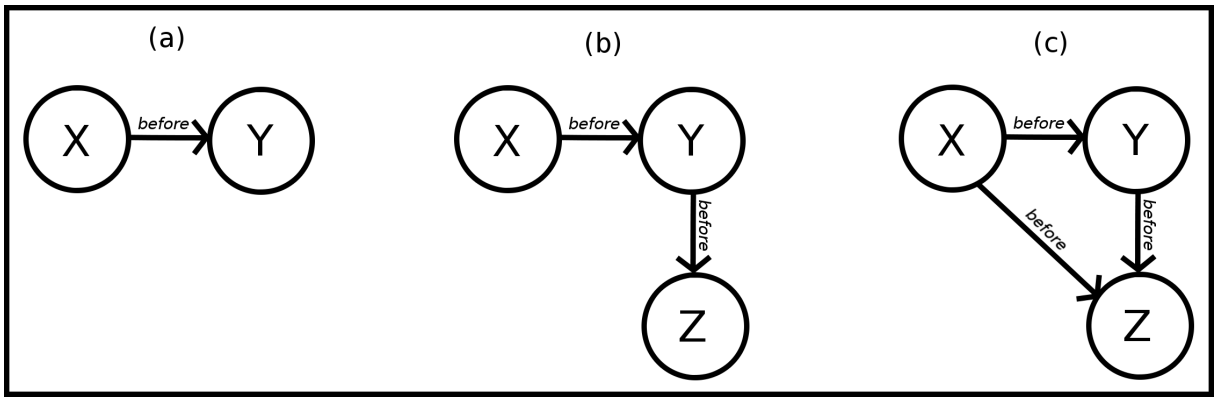


Figure 2: Simple graph network showing the computation of a transitivity.

If there is a single label on every arc in the graph (of which there should be $N(N-1)/2$ for N nodes), then the complete *temporal closure* has been calculated. This is a difficult (though desirable) goal given any real world data, as discourse often features incomplete or vague temporal information.[NB[3]]

Allen (1983, p. 836) also gives a 12×12 transitivity table (a fragment of which is reproduced below in Table 1) which gives the possible relations between two intervals A and C, given the relations **r1** between A and B, and **r2** between B and C (omitting the 'equal' relation). A smaller fragment of this table is given in Woods et al. (2017, p. 130) showing these transitivities interpreted as superpositions of strings (see Section 3). One

[3]See Verhagen (2005) for a discussion on temporal closure, write this in.

immediate advantage of the string approach comes from the fact that the strings may describe the relations between more than three events at once.

An algorithm is detailed (Allen, 1983, p. 835) for the propogation of constraints upon the addition of new information to the network, using the transitivity table to update all arcs in the network as necessary. While Allen notes that there are some issues with the algorithm (specifically he mentions the space requirement, which is somewhat high at $O(N^2)$ space for N temporal intervals, and the fact that the algorithm doesn't guarantee consistency in larger than three-node networks (a limit which is improved upon in this work—see § 3.2), and Verhagen (2005, p. 219) also notes an $O(N^3)$ time complexity), it does give an upper bound to the number of modifications that can be made to the network, regardless of the number of constraints added, as $13 \times \frac{(N-1)(N-2)}{2}$, and "the average amount of work for each addition is essentially linear (i.e., N additions take $O(N^2)$ time; one addition on average takes $O(N)$ time)" (1983, p. 837). This is still quite high, given that it is not unusual for documents of the TimeBank corpus to feature fifty intervals or more, and some are in the hundreds (Verhagen, 2005, p. 213).

B **r2** C / A **r1** B	<	d	o	m	s	f
'before' <	<	< o m d s	<	<	<	< o m d s
'during' d	<	d	< o m d s	<	d	d
'overlaps' o	<	o d s	< o m	<	o	o d s
'meets' m	<	o d s	<	<	m	o d s
'starts' s	<	d	< o m	<	s	d
'finishes' f	<	d	o d s	m	d	f

Table 1: Fragment of Allen's transitivity table (1983, p. 836, Figure 4).

(Allen, 1983, p. 838) also describes a method for grouping clusters of intervals which are fully computed in terms of the relations between them, termed "reference intervals" due to the fact that each interval I_i in the group $\{I_1, I_2, ..., I_n\}$ will reference some new interval

R_m (where m is an index on the number of reference intervals). The motivation for these is to reduce the space requirements while maintaining as much of the inferential power as possible. Allen shows how, since these reference intervals are just treated as normal intervals, they may themselves be grouped into a cluster, and thus allow for the creation of hierarchies of intervals which may be useful for scenarios such as when a path search algorithm is to be used. These are not currently implemented in the present work, but they may be useful as a way to address documents with high numbers of intervals.

Allen's work was and remains highly influential in the field as the approach is intuitive and straight-forward, as well as being easy to implement in various applications. One such proponent of interval relations has been TimeML, a markup language designed to "capture the richness of temporal and event related information in language" (Pustejovsky et al., 2005, p. 123).

2.1.2 Tense and Aspect

Reichenbach's (1947) theory of tense and aspect allows for the temporal placement of an event time in relation to a speech time and a reference time. The relative orderings of these three times gives rise to the categorisations of tense and aspect. [NB[4]]

2.2 Temporal Annotation

Ideally, a human using an artificially intelligent system won't have to consider annotation, except in the case when that itself is the goal. In any case, for now annotated text is a crucial source of data for working with models such as are presented in this work. When it comes to temporal annotation, the international standard is TimeML, specifically ISO-

[4]Bring up TEA here. Also worth mentioning Vendler and Dowty in relation to lexically telic/atelic verbs, which inform the stative-based approach of the strings and thus block compression. Tim mentioned that this is a difference from Schwer's S-Words.

TimeML. [NB[5]]

2.2.1 ISO-TimeML

TimeML (Pustejovsky et al., 2005) was initially designed with the goal of improving question-answering systems by marking up texts so as to give events explicit temporal locations.

TimeML vs ISO-TimeML: the ISO version is the standard. However, the TimeBank corpus uses the older version.

TLINK	**Allen**
SIMULTANEOUS	*equal* =
IDENTITY	*equal* =
BEFORE	*before* <
AFTER	*after* >
IBEFORE	*meets* m
IAFTER	*met by* mi
INCLUDES	*contains* di
IS_INCLUDED	*during* d
DURING	*during* d
DURING_INV	*contains* di
BEGINS	*starts* s
BEGUN_BY	*started by* si
ENDS	*finshes* f
ENDED_BY	*finished by* fi

Figure 3: Possible values of a TLINK's relType attribute and their Allen relation counterparts.

[5]Also talk about Tango and Verhagen's T-BOX.

2.2.2 TimeBank

The TimeBank corpus (Pustejovsky et al., 2006) collects news articles that were manually annotated with TimeML. While it does contain some inconsistencies as a result of human error, it remains one of the largest sources for documents marked up with TimeML.

2.3 Temporal Semantics

The following sections describe some of the existing ways that the semantics of times and events can be represented. [NB[6]]

2.3.1 Discourse Representation Theory

Kamp and Reyle (1993) introduced Discourse Representation Theory (DRT) as a framework for semantically representing information derived from a text. DRT separates discourse referents (the entities under discussion) from Discourse Representation Structure (DRS) conditions, which describe what is known about the referents.

[NB[7]]

2.3.2 Boxer

Automatic parsing of text into DRSs is a popular task [NB[8]], and Bos (2008) released the first version of the Boxer software which claimed to do this with over 95% coverage for semantic analysis of newswire texts, though it was noted in the release that performance for temporal data was not as strong as its other areas.

There is a newer version of Boxer implemented as part of the Parallel Meaning Bank

[6] This section needs more—Prior's TL, FOL, SOL, Event Calculus?, Frames?, FST?

[7] Harry Bunt's slides from his DCLRS talk would be useful here, else ref "A semantic annotation scheme for quantication" (Bunt, 2019)

[8] Find the shared task from 2019—I'm sure it was on the PMB website, but I can't find it at present

[Citation needed!] toolchain—however, this version had unfortunately not been made available for general use at the time of writing.

3 Finite-State Temporality

Following the intuition that sequences of (possibly overlapping) events and time periods may be conceptualised in a manner akin to a strip of film, or a timeline, finite-state techniques may be applied to temporal semantics in an approach known as *finite-state temporality* (Fernando and Nairn, 2005). In this chapter, strings are demonstrated as a tool of choice in modelling sequences of times and events for use within this approach, justified by their interpretation as finite models of Monadic Second-Order Logic, which leads to an equivalence with regular languages (see § 3.1.2). The advantage of this is that strings can be accepted and parsed by finite-state automata (FSA), which are a well-known formalism and have found significant usage across many fields, including disciplines of Mathematics, Linguistics, and Computer Science (see, for example, Buchner and Funke (1993); Veanes et al. (2012)), as well as in modern software development (for instance, Khourshid (2015) provides a tool for using FSA in the creation of online web applications). Such widespread application of FSA is due to their flexibility and efficiency as a technology, with benefits such as deterministic recognition being linear according to the length of the input, the associated closure properties of regular lanugages, and the ability to compose several automata (Wintner, 2007).

The mechanics of these temporal strings are shown in detail, along with an explanation on the methods of their creation, and a discussion on the granularity of temporal information that should be included within a string—that is, what level of detail should be considered when representing events in this manner. A number of operations are subsequently described for working with these strings, in particular *superposition* for the composition of multiple strings and combining the data found within them. The availability of these manipulations will prove useful when reasoning about event relations, and

also in maximising data density, as will be seen in § 4.1 and § 4.3.

A description follows of how strings may be applied in the creation of timelines, representations which contain the temporal information—events, times, and the relations between them—as extracted from an annotated piece of text, which has uses in the areas of, for instance, automatic summary-generation, fact-checking, and question-answering systems. Additionally, the techniques which are laid out in § 3.1.4 may be augmented with additional constraints so that strings can be used to model scheduling restrictions—for instance, that in a given system some event a must occur before some other event b, but may not be occurring while c is occurring—and the superposition of these amounts to constraint satisfaction. A variation of the well-known Zebra Puzzle (also referred to as 'Einstein's Riddle', as it is occasionally attributed to Albert Einstein—see Stangroom (2009, p. 10)) which uses temporal properties in place of spatial constraints is provided, exemplifying this particular usage of strings.

3.1 Strings for Times and Events

A string is a basic computational entity, defined as a finite sequence of symbols selected from some finite alphabet. They are amenable to manipulation using finite-state methods, something lacking in the infinite models of predicate logic. [NB[9]]

Strings as described and used throughout this work are used to represent sequences of events and time periods such that the linear order and inter-relations of these events and times are clearly apparent, while unnecessary repetition of information is avoided. For example, where js = "John sleeps", fa = "The fire alarm sounds", and lt = "Last Tuesday", the sentence "John slept through the fire alarm last Tuesday" might be represented

[9] Should probably have a citation here?

by the string $\boxed{}\boxed{lt}\boxed{js, lt}\boxed{fa, js, lt}\boxed{js, lt}\boxed{lt}\boxed{}$[10] (a detailed explanation follows in § 3.1.1).

These strings model the concept of inertial worlds, wherein a state will persist unless and until it is altered. The intuition for viewing inertia as a default state seems to go at least as far back as Aristotle ("But neither does time exist without change" in *Physics IV*), and is known by the term *commonsense law of inertia* (Shanahan, 1997, p. 19). This notion is also present in the Event Calculus, which represents the effects of actions on fluents in order to reason about change (Kowalski and Sergot, 1986; Miller and Shanahan, 1999; Mueller, 2008), and in the Fluent Calculus (Thielscher, 1999), which asserts that a state will be unaltered after an event, except for just those conditions which the event changes. Building this inertial world view into strings allows for certain flexibilities, as real duration does not need to be accounted for[11] when, for example, superposing strings in order to determine the relations between the events they mention (see § 3.1.4, p. 30; § 4.3.1). Fernando (2018, p. 44) also directly connects the notion of "No change unless forced" with strings, using it to motivate the concept of actions creating change. Additionally, the built-in inertiality—coupled with the fact that strings are explicit as to whether a particular *fluent* holds or does not hold at any particular moment in time—means that, for string-based representations of events and times, the classic issue of the frame problem[12] is avoided (see McCarthy and Hayes (1969, pp. 30-31)).

A fluent is a condition which may change over time—thus a predicate such as $Sleeps(john)$ ("John sleeps") becomes the fluent $Sleeps(john, t)$, where t is the time

[10] It is worth noting that, although tense and aspect are often abstracted away from the string models, this is not a necessity, and speech and reference times may be represented in the same way as event times (Fernando, 2016a; Derczynski and Gaizauskas, 2013; Reichenbach, 1947).

[11] Real-time durations may still be represented using strings, but the depiction will not (necessarily) align with the assumption that one string should be longer than another if it features events that have a longer duration.

[12] The frame problem is an issue that can arise in first-order logic representations of the world, whereby specifying the conditions which change as the result of an action is not sufficient to entail that no other conditions have changed.

at which John is sleeping. Following the convention set out and used by McCarthy and Hayes (1969), Van Lambalgen and Hamm (2008), Fernando (2016b) (among others), a fluent may be understood here as naming a temporal proposition—some event, time period, or state which may change (which hereafter will also be referred to as an event, as in Pustejovsky et al. (2005)). Sets of fluents will be encoded as symbols so that any number of them may hold at once, and these symbols will make up the alphabet from which strings are created.

In most cases, simple identifiers suffice for the purposes of labelling fluents as they appear in a string—so, for example, $Sleeps(john, t)$ may be labelled as `"js"`, where the time t being represented by the fluent's position in the string (explained fully in the next section). This follows TimeML's standard (Pustejovsky et al., 2010) of using identification labels such as `"e1"` or `"t2"` for events and times. However, in some instances, particularly when drawing inferences (see **??** and § 4.3.2) using semantic roles or lexical semantics, it will be useful to use a fuller labelling system, and so a string $\boxed{\boxed{js}}$ would be rendered instead as $\boxed{\boxed{sleeps(john)}}$.

3.1.1 Creating Strings

In order to create a string, first it is necessary to fix a finite set $\mathcal{V}$ of symbols which represent the times and events under discussion, such that each $v \in \mathcal{V}$ will be understood as naming a *fluent* as a unary predicate which holds at a particular time. A string $s = \sigma_1\sigma_2 \cdots \sigma_n$ of subsets σ_i of $\mathcal{V}$ is interpreted as a finite model of n discrete, contiguous moments of time, with $i \in \{1, 2, \ldots, n\}$.

The set $\mathcal{V}$ will be known as a *vocabulary*, and the powerset of $\mathcal{V}$ will serve as a finite alphabet $\Sigma = 2^{\mathcal{V}}$ of a string $s \in \Sigma^*$. Accordingly, at each position i in the string s, the *component* σ_i will be a (possibly empty) set of the fluents which hold at that position.

The chronology of a string is read from left to right, and thus, each component of the string depicts one of the n moments similar to a snapshot, or a frame of a film reel, and specifies the set of exactly those fluents which hold simultaneously at the i^{th} moment[13]. A string does not (necessarily) give any indication of real-time duration, due to the fact that it models an inertial world, and thus if a symbol occurs in several string components, this is not implicative of the fluent associated with that symbol occurring multiple times, nor of the fluent's real-time duration being necessarily different to another fluent whose associated symbol only occurs in a single string component (see also § 3.1.4, p. 29). A fluent $a \in \sigma_i$ is understood to be occurring before another fluent $b \in \sigma_j$ if $i < j$ and $b \notin \sigma_i$; if $a \in \sigma_i$ and $b \in \sigma_i$, then a and b are understood as occurring at the same time.

For convenience of notation, boxes $\boxed{\cdot}$ are used instead of curly braces $\{\cdot\}$ to denotate the sets which make up each component σ_i of a string $s = \sigma_1\sigma_2\cdots\sigma_n$ (as in Fernando (2004, 2015, 2016b))—for example, the string $\{\}\{a\}\{a,b\}\{b\}\{\}$ is written as $\boxed{\ }\boxed{a}\boxed{a,b}\boxed{b}\boxed{\ }$. This lends to the intuition that the strings may be read as strips of film, or as panels of a comic, with the same narrative-style layout as a timeline[14]. An empty box $\boxed{\ }$ is drawn for the empty set $\varnothing$: this is a string of length 1, a moment of time during which no fluent $v \in \mathcal{V}$ holds. This should not be confused with the empty string ϵ, which has length 0, and contains no temporal information.

It will be said that for any event a occurring in a string $s = \sigma_1\sigma_2\cdots\sigma_n$, a may not *judder*—that is, if a appears in multiple positions of the string, then those positions are

[13] Conversely, the set $\mathcal{V} - \sigma_i$ specifies exactly those fluents which *do not* hold at the i^{th} moment. This relates to the notion of logical circumscription (McCarthy, 1980), wherein if a formula is not known to be true, then it must be false.

[14] It is worth pointing out here that, as each box is a set, the order of fluents appearing in it is immaterial—for instance, $\boxed{a,b}$ is exactly equivalent to $\boxed{b,a}$, where in both cases, a is co-occurring with b. This contrasts with $\boxed{a}\boxed{b}$ vs $\boxed{b}\boxed{a}$, where in the first case a is occurring before b, and in the second, the converse is true.

contiguous; there are no gaps between appearances of a in s:

$$a \in \sigma_i \wedge a \in \sigma_j \wedge i < j \implies (\forall k, i < k < j)\ a \in \sigma_k \tag{1}$$

If some string, such as in eq. (2a), features judder in this way, it is said to be invalid. Judder can, if necessary, be treated by subindexing an event—for example, a juddering event a becomes separate sub-events $\{a_1, a_2, a_3\}$, as in eq. (2b).

(2a) $*\boxed{a}\,\boxed{\ }\,\boxed{a}\,\boxed{\ }\,\boxed{a}$ – invalid

(2b) $\boxed{a_1}\,\boxed{\ }\,\boxed{a_2}\,\boxed{\ }\,\boxed{a_3}$ – valid

Sets of strings are *languages*, which allow for grouping of strings based on some property—such as their source—and which introduces a method for handling such cases of ambiguity and non-determinism as will be shown in more detail in § 4.3.1 (see also eq. (34), p. 31). For example,

$$L = \{\boxed{a}\boxed{b}\boxed{c}, \boxed{a}\boxed{c}\boxed{b}\} \tag{3}$$

where the language L contains two strings which represent different—albeit related—sequences of events.

Note that a language containing only a single string may be conflated with its sole member, and vice versa. For instance:

$$\boxed{a}\boxed{b}\boxed{c} \approx \{\boxed{a}\boxed{b}\boxed{c}\} \tag{4}$$

This is a useful admittance, particularly in regards to superposition and other string

operations (see § 3.1.4), when it may be necessary to, for example, superpose a string and a language, in which case the string is conflated with a language containing just that string. This allows for superposition to be extended beyond just two strings to any arbitrary number: $s \,\&_{\mathcal{X}}\, s' \,\&_{\mathcal{X}}\, s'' \,\&_{\mathcal{X}}\, \ldots$ (see p. 43).

The next section details how strings are expressions of finite models of Monadic Second-Order Logic, and how they are thus manipulable by finite-state automata.

3.1.2 Strings as MSO Models

Strings of symbols representing events as described in § 3.1.1 may be interpreted as finite models of Monadic Second-Order Logic (MSO)[15]. MSO is a fragment of Second-Order Logic that restricts quantification so as to be permitted solely for unary predicates, which is equivalent to quantification over sets—this is due to the fact that a unary predicate may be effectively described by the set of terms for which that predicate is true. That is, in a given model of MSO, if there exists some property P, then $[\![P]\!]$ is the set of individuals for which P holds—known as the *interpretation* of P relative to the given model—such that:

$$P(x) \iff x \in [\![P]\!] \tag{5}$$

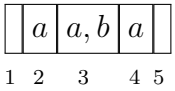
This may be construed for temporal representation by considering each predicate of a model as describing an event, and the terms which make that predicate true are the moments (relative to the model) during which the associated event is occurring.

This is most clearly illustrated by means of an example. The string below in eq. (6) will serve for this purpose, with the positional indices shown underneath the string position

[15] See Libkin (2004, ch. 7) for an in-depth introduction to the facets of MSO.

with which it corresponds:

(6)



This is a string of length 5, which contains two events, a and b, where b occurs *during* a (see table 7). The linear ordering of a and b can be identified by the string positions in which each occurs (Fernando, 2016a, 2018):

(7) $$[\![P_a]\!] = \{2, 3, 4\} \text{ and } [\![P_b]\!] = \{3\}$$

where P_a and P_b are unary predicates, and their interpretations are subsets of $\{1, 2, 3, 4, 5\}$, which is the set of all string positions for the string in eq. (6).

Generally, for any string $s = \sigma_1\sigma_2 \cdots \sigma_n$ with length $n \geq 0 \in \mathbb{N}$, the set $[n]$ of string positions[16] is defined as:

(8) $$[n] := \{1, 2, \ldots, n\}$$

Since s is restricted to being a finitely bounded string, $[n]$ is also finite, and thus the MSO model described by s must also be finite.

The vocabulary $\mathcal{V}$ is the set of fluents which appear in s, and for each $v \in \mathcal{V}$, the set of positions during which v (as a fluent) occurs $[\![P_v]\!]$ is a subset of $[n]$, and if there exists

[16] If $n = 0$, i.e. $s = \epsilon$, the empty string, then $[n] = \varnothing$, allowing a model to have an empty domain, as in Libkin (2004).

some $x \in [n]$ such that $x \in [\![P_v]\!]$, then P_v holds at x, as in eq. (5):

$$[\![P_v]\!] \subseteq [n] \tag{9}$$

$$P_v(x) \iff x \in [\![P_v]\!] \tag{10}$$

The successor relation which links each string position to the next is also defined:

$$S_n := \{(i, i+1) \mid i \in [n-1]\} \tag{11}$$

Now, for each $v \in \mathcal{V}$, the predicate P_v specifies all the string positions in which v occurs:

$$[\![P_v]\!] := \{i \in [n] \mid v \in \sigma_i\} \tag{12}$$

If $\text{MSO}_\mathcal{V}$ is the set of sentences (closed formulas, with no free variables) of MSO whose vocabulary is limited to subsets of $\mathcal{V}$, then an $\text{MSO}_\mathcal{V}$ model $mod(s)$—which is described by the string s—is defined by the tuple (Fernando, 2016a):

$$mod(s) := \langle [n], S_n, \{[\![P_v]\!] \mid v \in \mathcal{V}\} \rangle \tag{13}$$

Given an arbitrary $\text{MSO}_\mathcal{V}$ model M, the string $str(M)$ which describes it can be obtained by inverting eq. (12) to get each set $\sigma_i \in str(M)$, for $i \in [n]$, where $[\![P_v]\!]_M$ is the interpretation of the predicate P_v relative to the model M[17]:

$$\sigma_i := \{v \in \mathcal{V} \mid i \in [\![P_v]\!]_M\} \tag{14}$$

[17]In general, for a given string s, it will be convenient to write the interpretation of a property P relative to the $\text{MSO}_\mathcal{V}$ model described by s, $[\![P]\!]_{mod(s)}$, as simply $[\![P]\!]_s$.

There is a fundamental theorem which was independently put forward by Büchi, Elgot, and Trakhtenbrot (Fernando, 2016a, p. 30)[18] which states that sentences φ of MSO capture regular languages—that is, *the MSO-definability of a language is equivalent to its regularity*. This leads to the following definition in Fernando (2018, p. 35) of the set of regular languages over the powerset $2^{\mathcal{V}}$ of the vocabulary $\mathcal{V}$[19], given by the sentences φ of $\text{MSO}_{\mathcal{V}}$:

$$\{s \in (2^{\mathcal{V}})^* \mid mod(s) \models \varphi\} \tag{15}$$

Thus, languages of temporal strings are regular, and as such are open to manipulation and reasoning using finite-state techniques, due to the equivalence between regular languages and finite automata according to Kleene's theorem (see, for example, Yu (1997, p. 41)). For instance, the entailment of one regular language by another is decidable, which is not the case for First-Order Logic (Trakhtenbrot, 1953; Elgot and Rabin, 1966). This translates to being able to determine whether one string entails another, which is a powerful feature for ascertaining the relations which appear in a particular string, and for reasoning with them (see § 3.1.4, p. 36; § 4.3.1).

The fact that strings can be construed as models of MSO gives rise to a convenient method of comparison between strings, whereby the set of temporal relations (see Table 7, p. 48) found in one string s may be said to be equal to the set of relations in another string s' if s and s' are analogous.

Analogous Strings A string s will be said to be *analogous* to some other string s' if the MSO models corresponding to each string can be said to be, in a sense, isomorphic—that

[18] A proof is given in Libkin (2004, p.124, Theorem 7.21).

[19] The powerset $2^{\mathcal{V}}$ is used here in place of the usual $\mathcal{V}$, due to the fact that strings may allow any number of fluents $v \in \mathcal{V}$ to hold at once.

is, s and s' are of equal length, there exists a bijective function f mapping between the vocabulary of s and the vocabulary of s', and for every fluent $v \in \mathcal{V}_s$, its image $f(v) \in \mathcal{V}_{s'}$ appears in only those same strings positions in s' as v appears in s:

(16) $$mod(s) \cong mod(s') \iff$$
$$\text{length}(s) = \text{length}(s') \wedge (\exists f : \mathcal{V}_s \leftrightarrow \mathcal{V}_{s'})((\forall v \in \mathcal{V}_s)[\![P_v]\!] = [\![P_{f(v)}]\!])$$

(17) $$mod(s) \cong mod(s') \iff s \sim s'$$

For example, in eq. (18), $\boxed{\ }\boxed{a}\boxed{\ }\boxed{b}\boxed{\ }$ is analogous to $\boxed{\ }\boxed{c}\boxed{\ }\boxed{d}\boxed{\ }$, while in eq. (19) $\boxed{\ }\boxed{a}\boxed{\ }\boxed{b}\boxed{\ }$ is not analogous to $\boxed{\ }\boxed{c}\boxed{c,d}\boxed{d}\boxed{\ }$

(18) $$\boxed{\ }\boxed{a}\boxed{\ }\boxed{b}\boxed{\ } \sim \boxed{\ }\boxed{c}\boxed{\ }\boxed{d}\boxed{\ }$$

(19) $$\boxed{\ }\boxed{a}\boxed{\ }\boxed{b}\boxed{\ } \nsim \boxed{\ }\boxed{c}\boxed{c,d}\boxed{d}\boxed{\ }$$

This definition of analogy can be lifted to languages L and L' simply by testing if all strings in L are analogous with all strings in L':

(20) $$L \sim L' := (\forall s \in L, s' \in L')\ s \sim s'$$

Determining whether strings are analogous is useful when ascertaining the relations between events appearing in a string. If the relations between events in a string s are known, and another string s' can be shown to be analogous to s, then the relations between the events that appear in s' are also known[20]. By employing this concept in conjunction with that of projection (see p. 36) and a set of reference strings—such as those modelling

[20]For example, the string on the left hand side of eq. (18) says that event a is *before* event b, and since the string on the right hand side is analogous, THEN event c must be *before* event d.

Allen's relations (see table 7)—it becomes simple to determine which relations appear in a string.

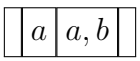
Further, if a pair of strings are shown to be analogous, this can make any calculations or processing involving these strings to be more efficient: any set of operations applied to one of the strings would produce the same result if applied to the other, so there is no need to apply them to both. This is expanded upon further in § 3.2.1, see p. 48.

3.1.3 Granularity: Points and Intervals

The vocabulary from which a string's alphabet is constructed is a set of fluents, or temporal propositions—for example "John sleeps" $Sleeps(john, t)$, where $Sleeps$ is a predicate and $john$ is an individual for which $Sleeps$ holds true for some time t. However, it has not yet been made explicit whether t should be an instantaneous point in time, or some temporal interval which has a non-zero duration. The need for this distinction is presented below.

In order for a string such as in eq. (21) to be valid, where some event a occurs in multiple positions in the string, a must be an interval: a period of time with a beginning and an ending, and a is occurring between these.

(21)



This allows for a to be represented via subdivisions[21] across the components within the string, such that the event a is understood as beginning at its first (leftmost) occurrence, and ending after its last (rightmost)—as long as it appears in a component, the event is occurring at that moment. While a point is instantaneous and therefore indivisible,

[21] An interval may be subdivided theoretically *ad infinitum*, though the constraint of a finite vocabulary prevents this from occurring within a string.

an interval may be split into an arbitrary number of contiguous subintervals in this way, which allows for the representation of other events co-occurring with a specific subdivision of an interval, as in eq. (21), where a is occurring during two string positions $[\![P_a]\!] = \{2, 3\}$, while b is only occurring during a single string position, $[\![P_b]\!] = \{3\}$. In this example, the real-time interval duration ι of position 3 is equal to that of the event b, which is also the span of time during which a and b are co-occurring. The summed duration ι' of positions 2 and 3 is equal to that of the event a, and so the duration of position 2 (the span of time in which a is occurring, but b has not begun) is equal to $\iota' - \iota$.

If a and b represent points of time, then the string in eq. (21) is invalid—since a point is instantaneous, it may not occur at multiple string positions, which are discrete moments of time. Additionally, it is not possible for one point to have a shorter duration than another, and thus the fact that b occurs at the end of a is lost. Allen (1983) also discusses the logical and physical issues with allowing events to be representable as instantaneous points, but there is also a cognitive issue, discussed in Freksa (1992), where events must have some non-zero duration in order to be perceivable (Hamblin, 1972).

That is not to say that this information cannot be represented in a string using points, but instead, a translation from events to *event borders* must occur, where event borders are points representing the beginnings and endings of events. Fernando (2018) uses $l(a)$ as the (open) left border and $r(a)$ as the (closed) right border of some event a—the *granularity* is altered to focus on the event borders (as points) rather than on the events (as intervals). In keeping with the analogy of strings being similar to strips of film, the change of granularity is akin to altering the level of 'camera zoom' to focus on details which were previously considered simply as parts of the whole.

This change does introduce additional complexity to the vocabulary, with the new set

$\breve{\mathcal{V}}$ constructed from the old $\mathcal{V}$ (Fernando, 2018, p. 37):

$$L = \{\ l(v) \mid v \in \mathcal{V}\ \}$$

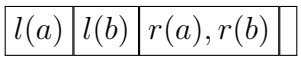
$$R = \{\ r(v) \mid v \in \mathcal{V}\ \}$$

$$\breve{\mathcal{V}} := L \cup R \qquad \text{if } L, R, \text{ and } \mathcal{V} \text{ are pairwise disjoint.} \tag{22}$$

The cardinality of $\breve{\mathcal{V}}$ is thus twice that of $\mathcal{V}$, with two symbols required to represent the same information of a single interval. The translation from a string $s = \sigma_1\sigma_2\cdots\sigma_n$ whose symbols are events (intervals) to a string $\breve{s} = \breve{\sigma_1}\breve{\sigma_2}\cdots\breve{\sigma_n}$ whose symbols are event borders (points) is given as (Fernando, 2018, p. 38):

$$\breve{\sigma_i} := \begin{cases} \{r(a) \mid a \in \sigma_i\} & \text{if } i = n \\ \{l(a) \mid a \in \sigma_{i+1} - \sigma_i\} \cup \{r(a) \mid a \in \sigma_i - \sigma_{i+1}\} & \text{if } i < n \end{cases} \tag{23}$$

The symbol $l(a)$ appearing in $\breve{\sigma_i}$ says that the event a is not occurring at position i, but is occurring at position $i+1$. The symbol $r(a)$ appearing in $\breve{\sigma_i}$ says that a is occurring at position i, but is not occurring at position $i+1$. Using event borders in this way, the string in eq. (21) would be translated to be drawn as:



(24)

Points are, in some ways, simpler than intervals—for instance, two points may be related in just three ways ($<$, $=$, and $>$), while a pair of intervals have thirteen possible relations between them (precisely, the set of relations given by Allen's interval algebra (Allen, 1983), see Table 7, p. 48). Additionally, focusing on the borders of events opens up a pathway

to representing incomplete information, by omitting one of the beginning or ending of an event from the vocabulary of a string, it is left unspecified, which is often the case in natural language. For example, in eq. (25) below, it is known that the events a and b begin simultaneously, but it is not known when a ends:

(25) $\boxed{l(a), l(b)}\boxed{r(b)}\boxed{}$

This situation cannot be simply represented using ordinary intervals, since for any symbol in a string's vocabulary, its appearance or non-appearance in a string component indicates explicitly whether or not the event is occurring at that moment, with no option for 'possibly occurring'.

This work, however, will primarily consider intervals as the basic unit of representation for fluents. When looking at a particular string component, it is more straightforward to tell whether some event a is occurring at that moment if the symbol a appears within that component, than to have to either check backwards and forwards through the string to see if the event has begun ($l(a)$ appears somewhere to the left of the component of interest) and not yet ended ($r(a)$ does not appear to the left of the component of interest), or to have to perform a translation. This also more closely mirrors the intuition of the analogy with panels of a comic strip or frames of a reel of film, such that all events which are occurring at any particular moment are 'visible' within that panel or frame (string component). There is further a higher level of descriptiveness that can be achieved with thirteen relations between a pair of intervals, as opposed to requiring a set of four points to achieve the same.

Chiefly, however, among reasons to generally consider intervals as primitive is the fact that ISO-TimeML, the international standard for temporal annotation Pustejovsky et al.

(2010), considers events and event-like entities to be intervals, and uses relations which are based in Allen (1983)'s set of interval relations. Further, Freksa (1992, p. 201) agrees with Allen from a cognitive perspective that events should not be "represented by points on the real line", and additionally describes the concept of *semi-intervals*—using intervals to represent the beginnings and endings of events—which allows for treating incomplete information similar to using points (as in eq. (25) above), and expanding the number of possible relations to a set of 31 options. Nonetheless, semi-intervals do, again, introduce an additional level of complexity over plain intervals, and are discussed more completely in § 4.1.2.

It is worth noting here that, in general, in this work it is also assumed that any text which will be used as a source for the creation of strings will feature only *finite events*. The fluents which represent these events will therefore hold for a finite amount of time, and thus intervals appearing in strings will be *bounded* (Allen and Ferguson, 1994). That is, for some event a, there is some time *immediately before* a holds during which a does not hold, and similarly, there is some time *immediately after* a holds during which a does not hold. This fact is represented through the use of bounding empty sets, and so any given string will both begin and end with an empty set, drawn as an empty box $\square$[22]. The framework does not, in fact, require this assumption to be true—compare the bounded (finite) interval a, drawn as $\square\,|a|\,\square$, and the non-bounded (infinite) interval b, drawn as $|b|$—and indeed, it is convenient to go beyond it when discussing the handling of incomplete data, although this moves away from plain intervals to semi-intervals (see § 4.1.2).

The following section describes a number of operations which may be used with interval-based strings, though it should be pointed out that for each operation there is an

[22]Equivalently, as a formula of MSO: $(\forall v \in \mathcal{V})\ \neg P_v(1) \wedge \neg P_v(n)$.

equivalent which may be used for a string which treats event borders as basic instead of events.

3.1.4 String Operations

A number of operations are available for the manipulation and description of strings which represent sequences of times and events, and these are detailed below. Key among these is *superposition* (basic, asynchronous, and vocabulary-constrained), which may be used to combine the information from multiple strings, but additionally defined are: an operation to find the vocabulary of arbitrary strings; block compression, which removes duplicate string components; reduct, which filters a string to only contain a specified set of events; and projection, whereby a string can be said to contain the same temporal data as another. Many of these operations are also available at the language level, in general by performing the particular desired operation on each string within the language.

Vocabulary: The *vocabulary* $\mathcal{V}$ of a string or language is the set of fluents which appear in it. For an arbitrary string $s = \sigma_1\sigma_2\cdots\sigma_n$, the vocabulary $\mathcal{V}_s$ of s may be determined by taking the union of the string's components:

$$\mathcal{V}_s := \sigma_1 \cup \sigma_2 \cup \cdots \cup \sigma_n \tag{26}$$

and the vocabulary of a language is just the union of the vocabularies of the strings it contains:

$$\mathcal{V}_L := \bigcup\{\mathcal{V}_s \mid s \in L\} \tag{27}$$

The vocabulary of a string is a key factor in the calculation of many other operations, notably projection (p. 36) and vocabulary-constrained superposition (p. 41).

Block Compression: Since the length of a string $s = \sigma_1\sigma_2\cdots\sigma_n$ does not reflect its real duration (due to the understanding that strings model intertial worlds—see § 3.1 p. 13), it is also not required that the length of time represented by any σ_i is equal to that represented by any σ_j, for $i \neq j$. Similarly, if a fluent symbol $v \in \mathcal{V}$ from the vocabulary appears in both σ_i and σ_{i+1}, this does not imply that the event represented by v has a duration twice as long as if it had only appeared in σ_i. Indeed, the symbol v may appear in any number of consecutive positions in s without affecting the interpretation of the real length of time of the event it represents. Further, if the string features a repeating component, i.e. $\sigma_i = \sigma_{i+1}$ for any $1 \leq i < n$, the interpretation of the string is not affected by the deletion of one of either σ_i or σ_{i+1}. So for example, the interpretation of the string $\boxed{a}\boxed{a}\boxed{a,b}\boxed{b}\boxed{b}$ is equal to the interpretation of the string $\boxed{a}\boxed{a,b}\boxed{b}$. A string featuring such repetitions is said to contain *stutter*.

As a result, the *block compression* $\mathrm{bc}(s)$ of a string s may be introduced, which removes any stutter present in s. This is defined as (Fernando, 2015; Woods et al., 2017):

(28)
$$\mathrm{bc}(s) := \begin{cases} s & \text{if } \mathit{length}(s) \leq 1 \\ \mathrm{bc}(\sigma s') & \text{if } s = \sigma\sigma s' \\ \sigma\mathrm{bc}(\sigma' s') & \text{if } s = \sigma\sigma' s' \text{ with } \sigma \neq \sigma' \end{cases}$$

Stutter may also be induced in a string which is *stutterless* (it does not contain stutter)

by using the inverse of block compression, which will generate infinitely many strings[23]:

$$\text{bc}^{-1}(\text{bc}(s)) := \sigma_1^+\sigma_2^+\cdots\sigma_n^+ \quad \text{if bc}(s) = \sigma_1\sigma_2\cdots\sigma_n \tag{29}$$

For example:

$$\text{bc}^{-1}(\boxed{a}\boxed{c}) = \{\boxed{a}\boxed{c}, \boxed{a}\boxed{a}\boxed{c}, \boxed{a}\boxed{c}\boxed{c}, \boxed{a}\boxed{a}\boxed{c}\boxed{c}, \ldots\} \tag{30}$$

Since these strings all block compress to the same string, they can be said to be equivalent under block compression. Specifically, strings s and s' are bc-*equivalent* iff $\text{bc}(s) = \text{bc}(s')$. This ability to generate infinitely many strings which have an equivalent interpretation allows for varying the length of a string as will be required in order to form a useful notion of superposition (see p. 32).

Superposition: In its most basic form, the *superposition* s & s' of two strings $s = \sigma_1\sigma_2\cdots\sigma_n$ and $s' = \sigma'_1\sigma'_2\cdots\sigma'_n$ of equal length n is simply their component-wise union[24]:

$$\sigma_1\sigma_2\cdots\sigma_n \;\&\; \sigma'_1\sigma'_2\cdots\sigma'_n := (\sigma_1 \cup \sigma'_1)(\sigma_2 \cup \sigma'_2)\cdots(\sigma_n \cup \sigma'_n) \tag{31}$$

For example:

$$\boxed{a}\boxed{b}\boxed{c} \;\&\; \boxed{a}\boxed{c}\boxed{d} = \boxed{a}\boxed{b,c}\boxed{c,d} \tag{32}$$

[23] If the string s features stutter, then $\text{bc}^{-1}(s)$ will not contain any strings with a length shorter than s, including $\text{bc}(s)$. To capture all possible bc-equivalent strings, s is block compressed before the inverse is applied.

[24] The vocabulary of the resulting string is, as might be expected, the union of the vocabularies of the original strings: $\mathcal{V}_{s\ \&\ s'} = \mathcal{V}_s \cup \mathcal{V}_{s'}$.

This is easily extended to pairs of languages L & L' by collecting the superpositions of strings of equal lengths in each language:

$$L \;\&\; L' := \bigcup_{n \geq 0} \{s \;\&\; s' | s \in L \cap \Sigma^n, s' \in L' \cap \Sigma^n\} \tag{33}$$

The result L & L' of superposing two languages L and L' is also a language, and if L and L' are regular languages (due to strings being finite models of Monadic Second-Order Logic and the theorem due to Büchi, Elgot, and Trakhtenbrot—see § 3.1.2, p. 21), then L & L' is also regular (Fernando, 2004; Woods et al., 2017, p. 126). If L is accepted by the finite automaton $\langle Q, (2^{\mathcal{V}_L})^*, (q \xrightarrow{\sigma} r), q_0, F \rangle$ and L' is accepted by the finite automaton $\langle Q', (2^{\mathcal{V}_{L'}})^*, (q' \xrightarrow{\sigma'} r'), q'_0, F' \rangle$ then L & L' is computed by a finite automaton composed of the automata accepting each L and L': $\langle Q \times Q', (2^{\mathcal{V}_L \cup \mathcal{V}_{L'}})^*, ((q, q') \overset{(\sigma \cup \sigma')}{\rightarrow} (r, r')), (q_0, q'_0), F \times F' \rangle$.

Using languages provides more flexibility than strings alone, since non-determinism can be accounted for through variations between strings within a language. For example, in eq. (34) below, the result of the superposition accounts for the alternate event sequences in the strings of the first input language.

$$\{\boxed{a}\boxed{b}\boxed{c}, \boxed{a}\boxed{c}\boxed{b}\} \;\&\; \{\boxed{a}\boxed{c}\boxed{d}\} = \{\boxed{a}\boxed{b, c}\boxed{c, d}, \boxed{a}\boxed{c}\boxed{b, d}\} \tag{34}$$

This may reflect a situation where there is uncertainty as to the correct order of events—in this case a language is useful to collect all of the possible alternatives, which can then be still be superposed with other languages.

Asynchronous Superposition: In order to extend this operation further, it is necessary to remove the restriction that only strings of equal length may be superposed together. This is desirable so as to allow arbitrary numbers of events to appear in strings, and to superpose strings which may be of unknown length. For example, the operation in eq. (35) below cannot be calculated, and even if the strings were instead singleton members of languages and those languages were superposed, the result would just be the empty set.

$$(35) \qquad (\boxed{a}\boxed{b} \;\&\; \boxed{c}\boxed{d}) \;\&\; (\boxed{a}\boxed{b}\boxed{c} \;\&\; \boxed{a}\boxed{c}\boxed{d}) = \boxed{a,c}\boxed{b,d} \;\&\; \boxed{a}\boxed{b,c}\boxed{c,d} = undefined$$

To achieve this, bc-equivalence is exploited and the *inverse block compression* operation (see eq. (29), p. 30) is leveraged. Since, by inducing stutter in a string, infinitely many new strings of greater or equal length can be generated which are bc-equivalent to the starting string, it is effectively possible to force a pair of strings to be of equal length.

So, the *asynchronous superposition* $s \;\&_* \; s'$ of two strings s and s' is initially defined as the language obtained by applying block compression to the results of superposition between the languages which are respectively bc-equivalent to each of s and s'[25]:

$$(36) \qquad s \;\&_* \; s' := \{\mathrm{bc}(s'') \mid s'' \in \mathrm{bc}^{-1}(\mathrm{bc}(s)) \;\&\; \mathrm{bc}^{-1}(\mathrm{bc}(s'))\}$$

Now the strings in eq. (35) can be superposed using asynchronous superposition, as in

[25]Note that $\mathrm{bc}(s) = \mathrm{bc}(s') \iff s \in \mathrm{bc}^{-1}(\mathrm{bc}(s'))$

eq. (37) below:

$$\boxed{a,c}\boxed{b,d} \;\&_*\; \boxed{a}\boxed{b,c}\boxed{c,d} = \{\boxed{a,c}\boxed{a,b,c}\boxed{b,c,d}, \boxed{a,c}\boxed{a,b,d}\boxed{b,c,d}, \boxed{a,c}\boxed{a,b,c}\boxed{a,c,d}\boxed{b,c,d}, \boxed{a,c}\boxed{b,c,d}\} \tag{37}$$

However, one slightly problematic aspect of this definition is the fact that bc^{-1} maps from a string to an infinite language. While this is not an issue from a theoretical standpoint, since $\&_*$ collects the set of block compressed strings from the superposition of these languages, from a practical and computational standpoint anything infinite is inconvenient.

In order to tackle this back to something finite and to avoid generation of large amounts of redundant information, in Woods et al. (2017, p. 127) an upper bound of $n + n' - 1$ is established for the maximum length of any string produced via asynchronous superposition $s \;\&_*\; s'$, where n and n' are the (nonzero) lengths of the strings s and s', respectively. This work additionally introduces the operation pad_k, which will perform inverse block compression on a string, but will only produce strings of a given length $k > 0$:

$$\begin{aligned} pad_k(\mathrm{bc}(s)) &:= \sigma_1^+\sigma_2^+\cdots\sigma_n^+ \cap \Sigma^k \quad \text{if } \mathrm{bc}(s) = \sigma_1\sigma_2\cdots\sigma_n \\ &= \{\sigma_1^{k_1}\sigma_2^{k_2}\cdots\sigma_n^{k_n} \mid k_1,\ldots,k_n \geq 1, \sum_{i=1}^{n} k_i = k\} \end{aligned} \tag{38}$$

The language produced by padding a string is a proper subset of the language produced by performing inverse block compression on that same string. For example, using the same string as eq. (30):

$$pad_3(\boxed{a}\boxed{c}) = \{\boxed{a}\boxed{a}\boxed{c}, \boxed{a}\boxed{c}\boxed{c}\} \tag{39}$$

By using this padding operation in place of the inverse block compression in an updated definition of asynchronous superposition, setting k to be the upper bound derived from the lengths of the input strings, the issue of going beyond finite sets is avoided without losing any of the power of using bc-equivalence:

$$s \mathbin{\&_*} s' := \{\mathrm{bc}(s'') \mid s'' \in pad_{n+n'-1}(s) \mathbin{\&} pad_{n+n'-1}(s')\} \tag{40}$$

It's worth noting here that neither basic superposition $\&$ nor asynchronous superposition $\&_*$ place any importance on the semantic content contained within the strings over which they operate. That is to say, they are entirely syntactical operations, and any meaningful information represented by a given string is liable to be lost once it has been superposed with some other string.

For instance, in eq. (32), the second of the operand strings has the event c appearing in a box to the left of (before) the event d, whereas the result has c and d occurring in the same box together, which states that they were occurring at the same moment.

Similarly, in eq. (37), the first input string has events a and c appearing in the same box, while the second input string has a appearing in a box before c. While this should seem like a contradiction which should not have viable results, the operation instead produces a set of four strings, the second and third of which are invalid according to eq. (1): in $\boxed{a, c}\boxed{a, b, d}\boxed{b, c, d}$ the event c occurs in positions 1 and 3 but not position 2, and in $\boxed{a, c}\boxed{a, b, c}\boxed{a, c, d}\boxed{b, c, d}$ the event b occurs in positions 2 and 4 but not in position 3.

This stands in contrast to the concept of bc-equivalence, where strings have an equal interpretation regardless of how much or how little stutter is present. By using superposition as it is currently presented, information is often, in fact, lost rather than gained

when strings are combined. The next two operations, *reduct* and *projection*, aim to assist in the resolution of this issue.

Reduct: It will be useful to be able to alter the vocabulary of a string—in particular, to shrink it—so as to control which events are mentioned. If a string contains, for instance, five events, but only two of these are relevant to the application, there is sense in being able to focus on those two.

As such, for any set A, the *A-reduct* ρ_A of a string $s = \sigma_1\sigma_2\cdots\sigma_n$ is defined as the componentwise intersection of s with A (Fernando, 2016a; Woods and Fernando, 2018):

$$\rho_A(\sigma_1\sigma_2\cdots\sigma_n) := (\sigma_1 \cap A)(\sigma_2 \cap A)\cdots(\sigma_n \cap A) \tag{41}$$

The resulting new string has a vocabulary of A, but the remaining fluents are still in the same relative positions to each other as in the original:

$$\mathcal{V}_{\rho_A(s)} = A \tag{42}$$

$$(\forall a \in A)\ [\![P_a]\!]_{mod(s)} = [\![P_a]\!]_{mod(\rho_A(s))} \tag{43}$$

For example, with the string $s = \boxed{\ }\boxed{a,b}\boxed{a,b,c}\boxed{a,c,d}\boxed{a,e}\boxed{e}\boxed{\ }$ and $A = \{a, d\}$, the A-reduct of s is:

$$\rho_{\{a,d\}}(\boxed{\ }\boxed{a,b}\boxed{a,b,c}\boxed{a,c,d}\boxed{a,e}\boxed{e}\boxed{\ }) = \boxed{\ }\boxed{a}\boxed{a}\boxed{a,d}\boxed{a}\boxed{\ }\boxed{\ } \tag{44}$$

The resulting string in eq. (44) contains only the events of interest (those mentioned in A), without loss of information. That is, the relative ordering of the events in the result string is the same as that in the input. The result string can additionally be block compressed

to derive the simplest representation of the information it contains[26]:

(45) $$\mathrm{bc}(\boxed{}\boxed{a}\boxed{a}\boxed{a,d}\boxed{a}\boxed{}\boxed{}) = \boxed{}\boxed{a}\boxed{a,d}\boxed{a}\boxed{}$$

It's worth noting that for any pair of strings s and s' with equal length and disjoint vocabularies, the reduct of the result of basic (not asynchronous) superposition $s \mathbin{\&} s'$ with respect to each of the strings' vocabularies is equal to the string itself:

(46) $$\mathcal{V}_s \cap \mathcal{V}_{s'} = \varnothing \Longrightarrow \rho_{\mathcal{V}_s}(s \mathbin{\&} s') = s \text{ and } \rho_{\mathcal{V}_{s'}}(s \mathbin{\&} s') = s'$$

For example, with $s = \boxed{a}\boxed{b}$ and $s' = \boxed{c}\boxed{d}$, $\mathcal{V}_s = \{a, b\}$ and $\mathcal{V}_{s'} = \{c, d\}$:

(47) $$s \mathbin{\&} s' = \boxed{a,c}\boxed{b,d}$$
$$\rho_{\{a,b\}}(\boxed{a,c}\boxed{b,d}) = \boxed{a}\boxed{b}$$
$$\rho_{\{c,d\}}(\boxed{a,c}\boxed{b,d}) = \boxed{c}\boxed{d}$$

Projection: This process is streamlined through the use of *projection*, where the A-projection π_A of a string s is simply the block compressed reduct of s relative to A:

(48) $$\pi_A(s) := \mathrm{bc}(\rho_A(s))$$

[26] In the case of eqs. (44) and (45), the event a *contains* the event d, according to Allen's relations (Allen, 1983)[Figure reference needed!].

It will be said that a string s *projects to* another string s', $s \sqsupseteq s'$, if the $\mathcal{V}_{s'}$-projection of s is equal to s':

$$s \sqsupseteq s' \iff \pi_{\mathcal{V}_{s'}}(s) = s' \tag{49}$$

If s projects to s', then all of the information represented within s' is also represented in s—s effectively 'contains' s'. Trivially, any block compressed string $s = \text{bc}(s)$ will project to itself, since the reduct of s with respect to its own vocabulary is s.

Borrowing from the examples in eqs. (44) and (45), the temporal data represented by the string on the right hand side of eq. (50) is also contained in the string on the left hand side:

$$\left[\,\middle|\, a,b \,\middle|\, a,b,c \,\middle|\, a,c,d \,\middle|\, a,e \,\middle|\, e \,\middle|\,\right] \sqsupseteq \left[\,\middle|\, a \,\middle|\, a,d \,\middle|\, a \,\middle|\,\right] \tag{50}$$

It is worth noting that, if strings s and s' share the same vocabulary, but are not equal, then neither can s project to s' nor s' project to s—this scenario suggests that s and s' are incompatible, that they describe contradictory sequences of the same events:

$$\mathcal{V}_s = \mathcal{V}_{s'} \wedge s \neq s' \implies \neg(s \sqsupseteq s' \vee s' \sqsupseteq s) \tag{51}$$

A language L can be said to project to another language L' if every string $s \in L$ projects to every string $s' \in L'$:

$$L \sqsupseteq L' \iff (\forall s \in L)(\forall s' \in L')\ s \sqsupseteq s' \tag{52}$$

This notion of projection is particularly useful, allowing for temporal reasoning when

used in conjunction with the concept of *analogous* strings (see § 3.1.2, p. 21). Particular events can be simply extracted from larger, more complex strings, and compared against reference to determine the relations between and ordering of the events of interest.

Importantly, projection will also be used to enrich the asynchronous superposition (see p. 34) operation, injecting the currently-lacking 'semantic-ness' by ensuring that all results of superposing a pair of strings project to each of their input strings.

Generate and Test: The predominant issue with asynchronous superposition $\&_*$ as it stands is that it needs not *preserve projections*—that is, data can become lost or 'corrupted' when combining strings. If a string generated by superposition does not project back to both of the strings that were superposed to generate it, then information has effectively been lost, as it has become impossible to return to the original event relation data. Only those result strings in $s \;\&_*\; s'$ which do project back can be said to be a valid result of the superposition, in terms of preserving the temporal information.

For example, in eq. (37) (p. 33), none of the result strings contain the information that is represented in the input strings. This can be verified by testing whether any string in the result set projects to either of the inputs, which they do not:

a,c	a,b,c	b,c,d	$\not\supseteq$	a,c	b,d		a,c	a,b,c	b,c,d	$\not\supseteq$	a	b,c	c,d
a,c	a,b,d	b,c,d	$\not\supseteq$	a,c	b,d		a,c	a,b,d	b,c,d	$\not\supseteq$	a	b,c	c,d
a,c	a,b,c	a,c,d	b,c,d	$\not\supseteq$	a,c	b,d	a,c	a,b,c	a,c,d	b,c,d	$\not\supseteq$	a	b,c	c,d
a,c	b,c,d	$\not\supseteq$	a,c	b,d			a,c	b,c,d	$\not\supseteq$	a	b,c	c,d			

Table 2: Failed projections for eq. (37).

In fact, every string in eq. (37) shares the same vocabulary, so by eq. (51), it is not possible for any to project to any other.

In the case where the vocabularies of the two strings are identical, then none of the results will project to the original strings[27]. This implication follows from eq. (51), since the vocabulary of every string $s'' \in s \mathbin{\&_*} s'$ is $\mathcal{V}_{s \cup s'} = \mathcal{V}_s = \mathcal{V}_{s'}$:

$$\mathcal{V}_s \cap \mathcal{V}_{s'} = \mathcal{V}_s = \mathcal{V}_{s'} \implies (\forall s'' \in s \mathbin{\&_*} s')\ \neg(s'' \sqsupseteq s \vee s'' \sqsupseteq s') \tag{53}$$

The implication of eq. (53) can be used to avoid unuseful superpositions: since the operation $\&_*$ has the potential to generate a large number of new strings, which can become costly from a computational perspective, it is prudent to test ahead of time whether every generated string will be spurious and unable to project back—as seen in Table 2.

Conversely, if the vocabularies of the input strings are disjoint, then every resulting string will project back, due to eq. (46):

$$\mathcal{V}_s \cap \mathcal{V}_{s'} = \varnothing \implies (\forall s'' \in s \mathbin{\&_*} s')\ s'' \sqsupseteq s \wedge s'' \sqsupseteq s' \tag{54}$$

This can be seen by returning to the example in eq. (47):

$$\boxed{a}\boxed{b} \mathbin{\&_*} \boxed{c}\boxed{d} = \{\boxed{a,c}\boxed{a,d}\boxed{b,d}, \boxed{a,c}\boxed{b,d}, \boxed{a,c}\boxed{b,c}\boxed{b,d}\} \tag{55}$$

$\boxed{a,c}\boxed{a,d}\boxed{b,d}$	$\sqsupseteq$	$\boxed{a}\boxed{b}$		$\boxed{a,c}\boxed{a,d}\boxed{b,d}$	$\sqsupseteq$	$\boxed{c}\boxed{d}$
$\boxed{a,c}\boxed{b,d}$	$\sqsupseteq$	$\boxed{a}\boxed{b}$		$\boxed{a,c}\boxed{b,d}$	$\sqsupseteq$	$\boxed{c}\boxed{d}$
$\boxed{a,c}\boxed{b,c}\boxed{b,d}$	$\sqsupseteq$	$\boxed{a}\boxed{b}$		$\boxed{a,c}\boxed{b,c}\boxed{b,d}$	$\sqsupseteq$	$\boxed{c}\boxed{d}$

Table 3: Preserved projections of eq. (55).

However, in general, the vocabularies of the input strings to asynchronous superposition

[27]This assumes that $s \neq s'$. In the case where $s = s'$, there is exactly one string $s'' \in s \mathbin{\&_*} s'$ such that $s = s' = s''$.

may overlap without being equal. It might be assumed that, in this case, some number greater than 0 but less than all of the resulting strings may project back—however, this assumption does not hold in fact. A counter-example is readily found:

(56) $$\boxed{a}\boxed{b} \;\&_* \; \boxed{b}\boxed{c} = \{\boxed{a,b}\boxed{a,c}\boxed{b,c}, \boxed{a,b}\boxed{b,c}, \boxed{a,b}\boxed{b}\boxed{b,c}\}$$

$\boxed{a,b}\boxed{a,c}\boxed{b,c}$	$\not\supseteq$	$\boxed{a}\boxed{b}$		$\boxed{a,b}\boxed{a,c}\boxed{b,c}$	$\not\supseteq$	$\boxed{b}\boxed{c}$
$\boxed{a,b}\boxed{b,c}$	$\not\supseteq$	$\boxed{a}\boxed{b}$		$\boxed{a,b}\boxed{b,c}$	$\not\supseteq$	$\boxed{b}\boxed{c}$
$\boxed{a,b}\boxed{b}\boxed{b,c}$	$\not\supseteq$	$\boxed{a}\boxed{b}$		$\boxed{a,b}\boxed{b}\boxed{b,c}$	$\not\supseteq$	$\boxed{b}\boxed{c}$

Table 4: Failed projections for eq. (56).

Thus it is necessary to test whether each result of a superposition is valid when neither eq. (53) nor eq. (54) are true. This approach of generating then testing was initially taken in Woods et al. (2017)[28] to ensure that only valid strings were finally produced. Yet, this is not without issue either—consider the following examples:

(57)

$$\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{} \;\&_* \; \boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{} = \{\boxed{}\boxed{a,b}\boxed{}\boxed{b,c}\boxed{}, \boxed{}\boxed{b}\boxed{a,b}\boxed{}\boxed{b,c}\boxed{}, \boxed{}\boxed{b}\boxed{a}\boxed{}\boxed{b,c}\boxed{},$$
$$\boxed{}\boxed{b}\boxed{a}\boxed{c}\boxed{b,c}\boxed{}, \boxed{}\boxed{b}\boxed{a}\boxed{c}\boxed{b}\boxed{}, \ldots\}$$

(58)

$$\boxed{}\boxed{a}\boxed{b}\boxed{c}\boxed{} \;\&_* \; \boxed{}\boxed{d}\boxed{c}\boxed{b}\boxed{} = \{\boxed{}\boxed{a,d}\boxed{a,c}\boxed{b}\boxed{c}\boxed{}, \boxed{}\boxed{d}\boxed{a,d}\boxed{b,d}\boxed{b,c}\boxed{}, \boxed{}\boxed{d}\boxed{c}\boxed{b}\boxed{a,b}\boxed{b}\boxed{c}\boxed{},$$
$$\boxed{}\boxed{a}\boxed{b}\boxed{c}\boxed{}\boxed{d}\boxed{c}\boxed{b}\boxed{}, \boxed{}\boxed{a}\boxed{b}\boxed{c,d}\boxed{c}\boxed{b,c}\boxed{b}\boxed{}, \ldots\}$$

The language result of eq. (57) contains 270 strings, and in fact, only one of these will project back to both of the input strings: namely, $\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{}$. This makes an intuitive

[28] Albeit, the testing algorithm used there was based on matching string positions rather than projections.

sense, since the inputs are 'a before b' – $\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{}$ and 'b before c' – $\boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{}$, and this result string is the only possibility where the linear ordering of the events a, b, and c is retained.

Using projection to test each of the generated strings in eq. (57) and rejecting those which fail to project back to the inputs will produce the singular correct result, though it is rather inefficient: 269 (over 99%) of the generated results must be discarded in this example. The language result of eq. (58) contains 257 strings, and this time 100% of them must be discarded: not one of the results projects back to both of the inputs. Clearly, the computational effort required to produce so many non-viable strings is entirely wasted, and so a modified approach is required to avoid this issue.

In Woods and Fernando (2018), a new version of superposition is defined which integrates projection-based testing into the generation process. This prevents problematic strings from ever being produced, improving on the efficiency of asynchronous superposition.

Vocabulary-Constrained Superposition: In order to define *vocabulary-constrained superposition* $\&_{\mathfrak{w}}$, begin by fixing an infinite set of fluents Θ. Then for any string s, the set of finite subsets of Θ is $Fin(\Theta)$ such that $s \in Fin(\Theta)^*$. Given a pair of finite subsets of Θ, $\Sigma \in Fin(\Theta)$ and $\Sigma' \in Fin(\Theta)$, an operation $\&_{\Sigma,\Sigma'} : (Fin(\Theta)^* \times Fin(\Theta)^*) \to 2^{Fin(\Theta)^*}$ is defined, mapping a pair of strings s and s' to a language $s \;\&_{\Sigma,\Sigma'}\; s'$ as follows, where ϵ is the empty string (of length 0)[29]:

[29]It follows from this definition that any string in $s \;\&_{\Sigma,\Sigma'}\; s'$ will have a length less than $n + n'$ where n and n' are the lengths of s and s', respectively, which is the same upper bound found in Woods et al. (2017) (see p. 33).

(59a) $$\epsilon \mathbin{\&_{\Sigma,\Sigma'}} \epsilon := \{\epsilon\}$$

(59b) $$\epsilon \mathbin{\&_{\Sigma,\Sigma'}} s := \varnothing \qquad \text{if } s \neq \epsilon$$

(59c) $$s \mathbin{\&_{\Sigma,\Sigma'}} \epsilon := \varnothing \qquad \text{if } s \neq \epsilon$$

and with $\sigma \in Fin(\Theta), \sigma' \in Fin(\Theta)$

(59d)

$$\sigma s \mathbin{\&_{\Sigma,\Sigma'}} \sigma' s' := \begin{cases} \{(\sigma \cup \sigma')s'' \mid s'' \in L(\sigma, s, \sigma', s', \Sigma, \Sigma')\} & \text{if } \Sigma \cap \sigma' \subseteq \sigma \text{ and } \Sigma' \cap \sigma \subseteq \sigma' \\ \varnothing & \text{otherwise} \end{cases}$$

where

(59e) $$L(\sigma, s, \sigma', s', \Sigma, \Sigma') := (\sigma s \mathbin{\&_{\Sigma,\Sigma'}} s') \cup (s \mathbin{\&_{\Sigma,\Sigma'}} \sigma' s') \cup (s \mathbin{\&_{\Sigma,\Sigma'}} s')$$

If $\Sigma = \Sigma' = \varnothing$, then the condition in the first case of eq. (59d) ($\Sigma \cap \sigma' \subseteq \sigma$ and $\Sigma' \cap \sigma \subseteq \sigma'$) holds vacuously, and $\&_{\Sigma,\Sigma'}$ becomes effectively identical to asynchronous superposition $\&_*$. Otherwise, this condition can be used to eject those strings which do not project back to both s and s', according to Proposition 1 and Corollary 2 in Woods and Fernando (2018, p. 81), reproduced below.

Proposition 1. *For all $\Sigma \in Fin(\Theta), \Sigma' \in Fin(\Theta)$ and $s \in Fin(\Theta)^*, s' \in Fin(\Theta)^*$, $s \mathbin{\&_{\Sigma,\Sigma'}} s'$ selects those strings from asynchronous superposition $s \mathbin{\&_{\varnothing,\varnothing}} s'$ which project*

to both the Σ-projection of s and the Σ'-projection of s':

$$s \;\&_{\Sigma,\Sigma'}\; s' \;=\; \{s'' \in s \;\&_{\varnothing,\varnothing}\; s' \mid s'' \sqsupseteq \pi_{\Sigma}(s) \wedge s'' \sqsupseteq \pi_{\Sigma'}(s')\} \tag{60}$$

Corollary 2. *For all $s \in Fin(\Theta)^*, s' \in Fin(\Theta)^*$ such that s and s' are stutterless, if $\Sigma = \mathcal{V}_s$ and $\Sigma' = \mathcal{V}_{s'}$, then $s \;\&_{\Sigma,\Sigma'}\; s'$ selects those strings from asynchronous superposition $s \;\&_{\varnothing,\varnothing}\; s'$ which project to s and s'*:

$$s \;\&_{\Sigma,\Sigma'}\; s' = \{s'' \in s \;\&_{\varnothing,\varnothing}\; s' \mid s'' \sqsupseteq s \wedge s'' \sqsupseteq s'\} \tag{61}$$

According to Corollary 2, *vocabulary-constrained superposition* $\&_{\mathcal{V}}$ can be used to preserve temporal information under projection during superposition:

$$s \;\&_{\mathcal{V}}\; s' := s \;\&_{\mathcal{V}_s,\mathcal{V}_{s'}}\; s' \tag{62}$$

Now, this new form of superposition can be used for the same example as eq. (57), and only the one valid result will be produced:

$$\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{} \;\&_{\mathcal{V}}\; \boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{} = \{\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{}\} \tag{63}$$

Note that this result is still a language (a set of strings), and that there may be more than one string in this language, depending on the input strings. Where each input's vocabulary has a cardinality of 2, and the intersection of their vocabularies has cardinality of 1, then the number of result strings from the vocabulary-constrained superposition of the inputs corresponds with the transitivity table in Allen (1983, Fig. 4). For example, eq. (64) shows the string 'a overlaps b' – $\boxed{}\boxed{a}\boxed{a,b}\boxed{b}\boxed{}$ superposed with 'b during c' – $\boxed{}\boxed{c}\boxed{b,c}\boxed{c}\boxed{}$, and according to Allen's transitivity table there should be three results, corresponding to 'a during c' – $\boxed{}\boxed{c}\boxed{a,c}\boxed{c}\boxed{}$, '$a$ overlaps c' – $\boxed{}\boxed{a}\boxed{a,c}\boxed{c}\boxed{}$, and '$a$ starts c' – $\boxed{}\boxed{a,c}\boxed{c}\boxed{}$, and

in fact this is the result shown by Table 5.

(64) $\left[\left|a\right|a,b\right|b\left|\right]\ \&_{\mathcal{U}}\ \left[\left|c\right|b,c\right|c\left|\right] = \{\left[\left|c\right|a,c\right|a,b,c\right|b,c\left|c\right|\right], \left[\left|a\right|a,c\right|a,b,c\right|b,c\left|c\right|\right],$
$\left[\left|a,c\right|a,b,c\right|b,c\left|c\right|\right]\}$

$$\left[\left|c\right|a,c\right|a,b,c\right|b,c\left|c\right|\right] \sqsupseteq \left[\left|c\right|a,c\right|c\left|\right]$$
$$\left[\left|a\right|a,c\right|a,b,c\right|b,c\left|c\right|\right] \sqsupseteq \left[\left|a\right|a,c\right|c\left|\right]$$
$$\left[\left|a,c\right|a,b,c\right|b,c\left|c\right|\right] \sqsupseteq \left[\left|a,c\right|c\left|\right]$$

Table 5: Projections of eq. (64) matching Allen's transitivities.

In Woods and Fernando (2018, p. 82) some simple benchmark tests were run, comparing the time (in milliseconds[30]) taken to compute the superpositions of a number of pairs of strings, using each of asynchronous superposition (generating then testing) and vocabulary-constrained superposition (testing while generating). These figures indicate a notable increase in the efficiency of time to calculate the correct results in using vocabulary-constrained superposition. Each of the strings in Table 7 (p. 48) is superposed with itself and each of the others (e.g. *before* with *before*, *before* with *after*, *before* with *meets*, ..., *finished by* with *started by*, *finished by* with *finished by*), while varying the vocabularies of the operand strings as follows: first, both strings had the same vocabulary $\{a, b\}$; second, the strings shared one fluent in common, $\{a, b\}$ and $\{b, c\}$; finally, the strings had disjoint vocabularies, $\{a, b\}$ and $\{c, d\}$. A fragment of these tests is shown in Table 6.

	$\triangle = *$	$\triangle = \mathcal{U}$	Decrease in time
$\left[\left\|a\right\|\right\|b\left\|\right] \&_{\triangle} \left[\left\|b\right\|\right\|a\left\|\right]$	0.3207ms	0.0180ms	94.39%
⋮	⋮	⋮	⋮
$\left[\left\|a\right\|\right\|b\left\|\right] \&_{\triangle} \left[\left\|b\right\|\right\|c\left\|\right]$	0.3207ms	0.0659ms	79.45%
⋮	⋮	⋮	⋮
$\left[\left\|a\right\|\right\|b\left\|\right] \&_{\triangle} \left[\left\|c\right\|\right\|d\left\|\right]$	22.4016ms	5.3616ms	76.07%
⋮	⋮	⋮	⋮

[30]The mean time of 1001 runs is given. The testing environment was Node.js v10.0.0 (64-bit) on Ubuntu 16.04 using an Intel i7-6700 CPU with 16GB of memory.

Table 6: Fragment of speed comparison between $\&_*$ and $\&_{vc}$.

The mean percentage decrease for all pairs with identical vocabularies using vocabulary-constrained superposition was 74.28%. With one shared fluent, the mean decrease was 34.11%, and with no fluents in common, it was 64.51%.

The main operations for string-based finite-state temporality have now been defined: block compression, reduct, projection, and (vocabulary-constrained) superposition. These operations may be leveraged in a number of ways in order to create useful applications.

3.2 Applications

Having discussed how to create and manipulate strings which represent temporal data—the linear order and inter-relations of a set of events—the following will explore some of the possible applications of this technology, including the ability to create timelines from annotated texts, to verify that the narrative structure of an annotated text is internally consistent with regards to the relations it depicts between its events, and to infer information not explicitly stated in an annotated text based on the event relations that can be extracted. These abilities can be used as part of tooling for the automatic aiding of creation of temporal annotation in new texts, for automated generation of summaries of a text, or fact-checking via corroboration of event sequences between sources.

Additionally, strings can be used as a tool for other, related applications which deal with sequential data through the use of external constraints. An example of such an application is given via a solution to a variant of the well-known Zebra Puzzle which models temporal constraints (that is, scheduling constraints) rather than spatial ones.

3.2.1 Timelines from Texts

Strings, as entities which are comprised of sequential components, have an intuitive comparison to a traditionally linear view of time. That is, that events which have not yet occurred are ahead of us, and events which occurred in the past are behind us—although not all languages or cultures perceive time in this way, it is common cross-linguistically to use some spatial reference points when discussing temporality (even absent vision, see

Bottini et al. (2015))—Lakoff and Johnson (2008, pp. 42–43), for example, discusses the metaphor of "Time is a moving object", where the future is perceived as moving towards us, and Mitchell (1980, p. 542) argues that "we literally cannot 'tell time' without the mediation of space". Often, time is visualised as a line which travels along a three-dimensional axis, with events appearing as points or spans along the line, although according to Rosenberg and Grafton (2013, p. 14), the particular modern definition of a *timeline* as a "single axis and a regular, measured distribution of dates ... is not even 250 years old".

Regardless of the spatial orientation or directionality, this perception of time as coming from somewhere and going to somewhere else maps well to a sequential representation. The core concept being that, if two events exist at different moments in time, then they can be put in some kind of spatial ordering corresponding to the temporal ordering. The strings described in § 3.1 are used to model sequences of events in such a way that they can be read in a manner similar to a series of snapshots or film reel, or like the panels of a comic: each 'image' or moment of time features all of the events which are occurring at that time (relative to some fixed vocabulary of events).

An example of timelines in use is via Gantt charts, also known as harmonograms, which are a kind of bar chart that—like strings—display events over time. They are often used as a visual aid to show project schedules or similar temporal data (Kumar, 2005). The vertical axis shows the events mentioned in the chart, and the horizontal axis represents time intervals, with the width of the bars showing the duration of each event, and the beginnings and endings also illustrated by the bars' horizontal placement. See, for example, Figure 4 below.

A chart like this is a useful visual display tool, though it can become a little unwieldy with large numbers of events. Compare the same temporal data in Figure 4 shown in a single string in eq. (65):

(65)

	a	a, b	a, b, c	b, c	b	b, d	b, d, e	e	f	

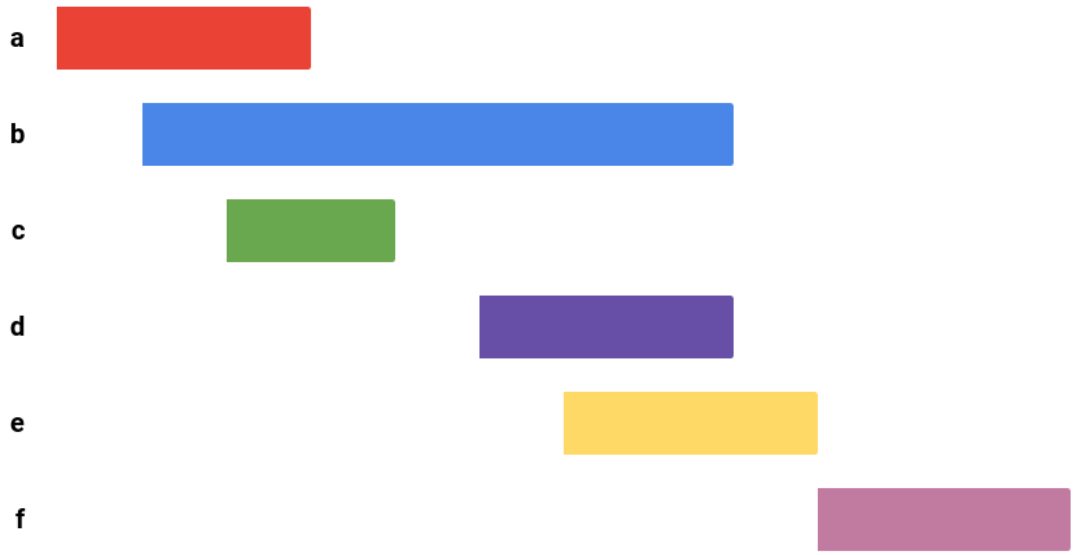


Figure 4: A Gantt chart featuring six events.

No matter how many events may be of interest, if all beginnings, endings, and durations are known (such as they could be displayed in a Gantt chart), then they can be represented in a single string, with reducts and projections available in order to focus an analysis on a subset of events if desired, demonstrating a compact method of representing the timeline. However, in general, a text does not provide

A lot of information can be derived from a text document, depending on what one is looking for. Here, the focus is on isolating the events and times that are mentioned or implied in the text. By extracting this information, a timeline is built which gives a picture of the content of the document, which may reveal insights not obvious when looking at the text as a whole.

A point of interest is that the thirteen interval relations given by the interval algebra of Allen (1983) fall out of the vocabulary-constrained superposition of a pair of strings which each feature a single and different finitely-bounded event:

(66) $$\mathcal{AR} := \{<, >, m, mi, o, oi, d, di, s, si, f, fi, =\}$$

(67) $$\boxed{\boxed{a}} \;\&_{ix}\; \boxed{\boxed{b}} = \{\mathcal{S}_\bullet(a, b) \mid \bullet \in \mathcal{AR}\}$$

Each string $\mathcal{S}_\bullet(a, b)$ of the result set features one relation $a \bullet b$, as shown in Table 7, reproduced from Woods and Fernando (2018, p. 79, Table 1).

$\bullet$	$a \bullet b$	$\mathcal{S}_\bullet(a,b)$	$\bullet^{-1}$	$a \bullet^{-1} b$	$\mathcal{S}_{\bullet^{-1}}(a,b)$
$<$	a before b	$\boxed{\boxed{a}\boxed{b}}$	$>$	a after b	$\boxed{\boxed{b}\boxed{a}}$
m	a meets b	$\boxed{\boxed{a}\boxed{b}}$	mi	a met by b	$\boxed{\boxed{b}\boxed{a}}$
o	a overlaps b	$\boxed{\boxed{a}\boxed{a,b}\boxed{b}}$	oi	a overlapped by b	$\boxed{\boxed{b}\boxed{b,a}\boxed{a}}$
d	a during b	$\boxed{\boxed{b}\boxed{a,b}\boxed{b}}$	di	a contains b	$\boxed{\boxed{a}\boxed{b,a}\boxed{a}}$
s	a starts b	$\boxed{\boxed{a,b}\boxed{b}}$	si	a started by b	$\boxed{\boxed{b,a}\boxed{a}}$
f	a finishes b	$\boxed{\boxed{b}\boxed{a,b}}$	fi	a finished by b	$\boxed{\boxed{a}\boxed{b,a}}$
$=$	a equals b	$\boxed{\boxed{a,b}}$			

Table 7: Allen interval relations in strings.

Using the notion of analogous strings (see § 3.1.2, p. 21), any block compressed string which has a vocabulary of cardinality 2 can be compared to the strings in Table 7. If such a string $s \sim \mathcal{S}_\bullet(a,b)$ for some $\bullet \in \mathcal{AR}$, then the events appearing in s can also be said to be related by $\bullet$. For example, $\boxed{\boxed{c}\boxed{d,c}} \sim \mathcal{S}_{\text{fi}}(a,b)$, and thus the relation between the events c and d is 'c finished by d'.

This is easily extended beyond strings featuring just two fluents. The relations between the events appearing in the string $s = \boxed{\boxed{a}\boxed{b}\boxed{c}}$ can be determined by taking its block compressed reduct relative to the subsets of the vocabulary which have cardinality 2—in this case, a is before b, b is before c, and a is before c. Once these relations have been calculated, the relations between the events in another string $s' = \boxed{\boxed{d}\boxed{e}\boxed{f}}$ are immediately available on analogy $s \sim s'$.

While this is a relatively simple example, it can be extended for strings featuring any arbitrary number of events, and perhaps more usefully it can be used to shortcut superpositions and other string operations. For instance, given the pair of strings $s = \boxed{\boxed{a}\boxed{a,b}\boxed{b}}$ and $s' = \boxed{\boxed{c}\boxed{b,c}\boxed{c}}$, the vocabulary-constrained superposition is calculated as in eq. (64):

$$s \mathbin{\&_{vc}} s' = \{\boxed{\boxed{c}\boxed{a,c}\boxed{a,b,c}\boxed{b,c}\boxed{c}}, \boxed{\boxed{a}\boxed{a,c}\boxed{a,b,c}\boxed{b,c}\boxed{c}}, \boxed{\boxed{a,c}\boxed{a,b,c}\boxed{b,c}\boxed{c}}\} \tag{68}$$

Now, given two more strings $t = \boxed{}\boxed{x}\boxed{x,y}\boxed{y}\boxed{}$ and $t' = \boxed{}\boxed{z}\boxed{y,z}\boxed{z}\boxed{}$ such that $s \sim t$ and $s' \sim t'$, the generated results of superposing t and t' will also be analogous to the results in eq. (68), and so there is no need to calculate $t \;\&_{\mathscr{w}}\; t'$. Since the strings are analogous, there is a bijective mapping [NB[31]] between the vocabularies $f : (\mathcal{V}_s \cup \mathcal{V}_{s'}) \leftrightarrow (\mathcal{V}_t \cup \mathcal{V}_{t'})$, and applying this function f to the results in eq. (68) gives the same result as calculating the superposition $t \;\&_{\mathscr{w}}\; t'$:

(69) $$f(s \;\&_{\mathscr{w}}\; s') = \{\boxed{}\boxed{z}\boxed{x,z}\boxed{x,y,z}\boxed{y,z}\boxed{z}\boxed{}, \boxed{}\boxed{x}\boxed{x,z}\boxed{x,y,z}\boxed{y,z}\boxed{z}\boxed{},$$
$$\boxed{}\boxed{x,z}\boxed{x,y,z}\boxed{y,z}\boxed{z}\boxed{}\}$$

(70) $$= \; t \;\&_{\mathscr{w}}\; t'$$

Again, this is a small example, but with larger numbers of events and more complex strings, leveraging the power of analogous strings has the potential to massively reduce the computational cost to calculate superpositions.

3.2.2 Scheduling (Zebra Puzzle)

Scheduling is a problem. The Zebra/Einstein Puzzle is another problem. They may seem dissimilar, but in fact, there are a number of parallels. Both problems can be viewed as constraint satisfaction problems. Although the Zebra Puzzle concerns physical constraints, these can be modelled as sequences for which one can use strings to represent. If a 'house' is conceptualised as a box in a string i.e. a set, the elements contained within are the properties of that house according to the puzzle. For example, $\{red, english, zebra, oj, kools\}$. The left/right spatial relations are thought of in the same way as the previous/next boxes in a string.

Below is presented a variant of the puzzle using clues pertaining to temporal relations instead of spatial ones.

The clues to the puzzle are as follows:

1. There are five weekdays.

[31] ref a definition for analogous strings, also define analogous languages

2. The foggy day is mild.

3. I am tired on the warm day.

4. There is a traffic jam on the overcast day.

5. It is cold on the day with little traffic.

6. It is overcast the day after it snows.

7. I'm sad the day that I'm reading.

8. It rains the day I have printing to do.

9. The traffic is average in the middle of the week.

10. It's freezing at the beginning of the week.

11. The stapling is done the day before or the day after I'm happy.

12. Printing happens the day before or the day after I'm angry.

13. There is a lot of traffic the day that filing happens.

14. Shredding happens the day it's hot.

15. It's freezing the day before or the day after the weather is clear.

Table 8: Temporal Zebra puzzle clues in English.

Using these clues, it should be possible to answer these questions:

- What day is there no traffic?
- What day am I curious?

The puzzle makes three assumptions: first, that the values as presented are *discrete* rather than continuous—'freezing' and 'cold', for example, are similar concepts (indeed, if the weather is freezing, then it is also cold), but for the purposes of this puzzle they are treated as being separate and unrelated; second, that each value lasts for only one day—if

it rains on one of the days, it will not rain on any other; and third, each day only has one value per attribute—for example, only one task can be performed on any given day.

For each of the given clues, a constraint can be constructed from one or more strings. Additional constraints can be formed to represent the external assumptions, and by superposing these constraints together, a solution to the puzzle can be found. The strings corresponding to each clue are as follows:

1. $\boxed{mon}\boxed{tue}\boxed{wed}\boxed{thu}\boxed{fri}$
2. $\{\boxed{\boxed{fog, mild}}, \boxed{\boxed{fog, mild}}, \boxed{\boxed{fog, mild}}\}$
3. $\{\boxed{\boxed{tired, warm}}, \boxed{\boxed{tired, warm}}, \boxed{\boxed{tired, warm}}\}$
4. $\{\boxed{\boxed{jammed, overcast}}, \boxed{\boxed{jammed, overcast}}, \boxed{\boxed{jammed, overcast}}\}$
5. $\{\boxed{\boxed{cold, little}}, \boxed{\boxed{cold, little}}, \boxed{\boxed{cold, little}}\}$
6. $\{\boxed{\boxed{snow}\boxed{overcast}}, \boxed{\boxed{snow}\boxed{overcast}}, \boxed{\boxed{snow}\boxed{overcast}}\}$
7. $\{\boxed{\boxed{sad, reading}}, \boxed{\boxed{sad, reading}}, \boxed{\boxed{sad, reading}}\}$
8. $\{\boxed{\boxed{rain, printing}}, \boxed{\boxed{rain, printing}}, \boxed{\boxed{rain, printing}}\}$
9. $\boxed{\boxed{\boxed{average}}}$
10. $\boxed{freezing}$
11. $\{\boxed{\boxed{stapling}\boxed{happy}}, \boxed{\boxed{stapling}\boxed{happy}}, \boxed{\boxed{stapling}\boxed{happy}},$
    $\boxed{\boxed{happy}\boxed{stapling}}, \boxed{\boxed{happy}\boxed{stapling}}, \boxed{\boxed{happy}\boxed{stapling}}\}$
12. $\{\boxed{\boxed{printing}\boxed{angry}}, \boxed{\boxed{printing}\boxed{angry}}, \boxed{\boxed{printing}\boxed{angry}},$
    $\boxed{\boxed{angry}\boxed{printing}}, \boxed{\boxed{angry}\boxed{printing}}, \boxed{\boxed{angry}\boxed{printing}}\}$
13. $\{\boxed{\boxed{lots, filing}}, \boxed{\boxed{lots, filing}}, \boxed{\boxed{lots, filing}}\}$
14. $\{\boxed{\boxed{shredding, hot}}, \boxed{\boxed{shredding, hot}}, \boxed{\boxed{shredding, hot}}\}$

15. {$\boxed{}\boxed{freezing}\boxed{clear}$, $\boxed{}\boxed{freezing}\boxed{clear}\boxed{}$, $\boxed{freezing}\boxed{clear}\boxed{}$,
$\boxed{}\boxed{clear}\boxed{freezing}$, $\boxed{}\boxed{clear}\boxed{freezing}\boxed{}$, $\boxed{clear}\boxed{freezing}\boxed{}$}

Table 9: Temporal Zebra puzzle clues as strings.

For most of the clues, the constraint is formed from a language: when event a and b occur on the same day, they will appear in the same box, but it's unknown whether they occur at the beginning of the week ($\boxed{a, b}\boxed{}$), the end of the week ($\boxed{}\boxed{a, b}$), or somewhere in the middle ($\boxed{}\boxed{a, b}\boxed{}$), and so all of the possibilities appear together.

Formally, there are five Attributes (Weather, Temperature, Traffic, Tasks, Mood), each of which is a set of Values[32]. The vocabulary $\mathcal{V}$ of the puzzle is the union of the Attributes:

- Weather = $\{rain, clear, fog, snow, overcast\}$
- Temperature = $\{freezing, cold, mild, warm, hot\}$
- Traffic = $\{none, little, average, lots, jammed\}$
- Tasks = $\{printing, stapling, reading, filing, shredding\}$
- Mood = $\{happy, angry, sad, tired, curious\}$

The external constraints are formalised as follows:

"Each Value only lasts for one day."

$$(\forall v \in \mathcal{V})\ x \in [\![P_v]\!] \implies (\forall v' \in \mathcal{V})\ v' \neq v \wedge x \notin [\![P_{v'}]\!] \tag{71}$$

"Each day only contains one Value for each Attribute."

$$(\forall A \in Attributes)\ x \in [\![P_v]\!] \wedge v \in A \implies (\forall v' \in A)\ v' \neq v \wedge x \notin [\![P_{v'}]\!] \tag{72}$$

One further constraint is needed due to the nature of superposition allowing for the result

[32]The five day names as seen in the first clue of Table 9 are not required for the puzzle: a string of five empty boxes suffices. However, the names are included here purely for convenience of reading.

of superposing two strings to be longer than either of the input strings:

"There are only 5 days."

(73) $(\forall v \in \mathcal{V})\ x \in [\![P_v]\!] \Longrightarrow 1 \leq x \leq 5$

Now, superposing all of the languages in Table 9 and taking into account the constraints in eqs. (71) to (73), the (singular) result string is generated (each Value is displayed here abbreviated to its first two letters):

(74) $\boxed{ra, fr, pr, ha}\boxed{cl, co, li, st, an}\boxed{fo, mi, av, re, sa}\boxed{sn, wa, lo, fi, ti}\boxed{ov, ho, ja, sh}$

Finally, superposing this string with languages representing the questions that were asked, a string containing the full "week schedule" is obtained (eq. (75a)), and by taking the reduct of this string relative to $\{none, curious\}$ the solution to the puzzle can be found (eq. (75b), and superposed with a string of weekday names for ease of reading in eq. (75c)):

- $\{\boxed{}\boxed{none}, \boxed{}\boxed{none}\boxed{}, \boxed{none}\boxed{}\}$
- $\{\boxed{}\boxed{curious}, \boxed{}\boxed{curious}\boxed{}, \boxed{curious}\boxed{}\}$

(75a) $s = \boxed{ra, fr, pr, ha, \underline{no}}\boxed{cl, co, li, st, an}\boxed{fo, mi, av, re, sa}\boxed{sn, wa, lo, fi, ti}\boxed{ov, ho, ja, sh, \underline{cu}}$

(75b) $\rho_{\{none,curious\}}(s) = \boxed{\underline{none}}\boxed{}\boxed{}\boxed{}\boxed{\underline{curious}}$

(75c) $\rho_{\{none,curious\}}(s)\ \&\ \boxed{mon}\boxed{tue}\boxed{wed}\boxed{thu}\boxed{fri} = \boxed{mon, \underline{none}}\boxed{tue}\boxed{wed}\boxed{thu}\boxed{fri, \underline{curious}}$

Thus the answer is that "there is no traffic on the first day of the week", and "I am curious on the last day of the week". This result is reproduced in Table 10 below for the sake of completeness.

	Mon	Tue	Wed	Thu	Fri
Weather	*rain*	*clear*	*fog*	*snow*	*overcast*
Temperature	*freezing*	*cold*	*mild*	*warm*	*hot*
Traffic	*none*	*little*	*average*	*lots*	*jammed*
Task	*printing*	*stapling*	*reading*	*filing*	*shredding*
Mood	*happy*	*angry*	*sad*	*tired*	*curious*

Table 10: Solution to Temporal Zebra puzzle as in eq. (75a).

[NB[33]]

[33] how does this connect with scheduling?

4 Methods

This chapter details the approaches that were taken to produce the results.

4.1 Extracting Strings from Annotated Text

The primary source of data is the TimeBank corpus which is one of the largest available collections of documents which are annotated with TimeML. Events are marked up with an <EVENT> tag, and times with the <TIMEX3> tag. Within TimeML, they are treated as intervals (see § 3.1.3) which becomes relevant to the present work in the way they are related.

4.1.1 TLINKs

TimeML primarily uses <TLINK> tags to annotate relations between marked up events and times. These serve as the basis for initial string creation, as they provide the two intervals that are being related, as well as the relation between them. The relation set used is specific to TimeML, but has its roots in the set of Allen Relations—see table [Figure reference needed!] below.

A translation is built from <TLINK> to string, using the tag's attributes to provide the data, and the Allen Relations as an intermediary step, as per table [Figure reference needed!]. For example: [NB[34]]. Any events and times that feature in the document but are not mentioned in a <TLINK> are kept aside for now, as there is not a given connection from these events to the rest of the timeline.

After creating strings from the <TLINK>s, the next step is to start building the timeline, using the superposition technique described in § 3.1.4. This allows for connections to be built between events and times that were not explicitly related by the annotation. If every time and event in the text is related to every other time and event, there is *temporal closure* in the text (see § 2.1.1 [NB[35]]). For there to be temporal closure in a TimeML document would be a large amount of work for an annotator, as for N events/times there

[34] example here please

[35] See transfer p6 – maybe include the graph of $n(n-1)/2, n \geq 2$?

would be $N(N-1)/2$ relations between them, which increases quickly: at $N = 10$ there are 45 relations, at $N = 50$ there are 1225 relations. This becomes unwieldy for a human to annotate, especially given that it is generally understood by the human reader that if A precedes B, and B precedes C, then A also precedes C, so it feels unnecessary to include this latter relation. However, an automated system does not have this inherent knowledge, and needs a way to calculate the implied relations.

The various relations can be combined according to the table in Allen (1983) [Citation needed! – get the page and figure number], reproduced in § 2.1.1 [Figure reference needed!], and again below using strings in table [Figure reference needed!]. The advantage to using strings is that the extension beyond three events is simple [NB[36]]. It should be noted that several combinations of relations produce a disjunction of relations, which follows from the fact that these combinations of $a \bullet b$ and $b \bullet c$ don't provide enough information to determine the exact relation $a \bullet c$. It may be possible to narrow this down, though, with the addition of further relations connecting other intervals $(d, e, f, ...)$ with some or all of a, b, c.

4.1.2 Handling Incomplete Data

When dealing with any temporal information, there is often a lack of specificity that means temporal closure cannot be calculated. For example, knowing that event a occurred before event c, as in $\boxed{}\boxed{a}\boxed{}\boxed{c}\boxed{}$, and that event b also occurred before c, as in $\boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{}$, does not impart any information on the relation between a and b. If this is all of the available data, then it is only possible to choose this relation $a \bullet b$, $\bullet \in \mathcal{AR}$ with a precision of 1 in 13 chance. However, further information which includes other relations for each of a and b individually may eventually reveal their connection.

See also, using semi-intervals to simplify disjunctions of Allen Relations. Minimal sets of maximal strings.

Freksa relations using semi-intervals: the appearance of $\alpha(a)$ within a box represents a negation of the fluent a conjoined with a formula stating that a will be true in a subsequent

[36] Defend this – I think there is something in ISA13 paper – or remove

box; similarly, $\omega(a)$ represents a negation of the fluent a conjoined with a formula stating that a was true in a previous box.

$$\alpha_a(x) := \exists y(x < y \ \wedge \ y \in [\![P_a]\!] \ \wedge \ \forall z(z < y \Longrightarrow z \notin [\![P_a]\!])) \tag{76}$$

$$\omega_a(x) := \exists y(y < x \ \wedge \ y \in [\![P_a]\!] \ \wedge \ \forall z(y < z \Longrightarrow z \notin [\![P_a]\!])) \tag{77}$$

Note that it is convenient to write $\alpha(a)$ for $\alpha_a(x)$ when using the box-notation.

A string may be translated to one using semi-intervals by placing $\alpha(f)$ in every box preceding one in which a fluent f appears, and $\omega(f)$ in every box succeeding it, repeating this for each $f \in A$.

Thus a string $s = \boxed{\ }\boxed{a}\boxed{b}\boxed{\ }$ becomes [NB[37]]

$$\lceil s \rceil = \boxed{\alpha(a), \alpha(b)}\boxed{\alpha(b)}\boxed{\omega(a)}\boxed{\omega(a), \omega(b)} \tag{78}$$

in which each of the fluents originally appearing in the string s have also been removed in . It should be noticed that strings which use semi-intervals do not (necessarily) feature empty boxes at each end. This is due to the fact that a semi-interval need not be finitely bounded. In fact

This mechanism allows for partially known information to be represented using strings. For example, the string $\boxed{\alpha(a), \alpha(b)}\boxed{\ }$ represents the knowledge that the events labelled a and b both begin at the same moment, without stating anything about when they each finish—they may end simultaneously, a may finish before b, or b may finish before a. Which of these states is true is unknown without further data.

Semi-intervals allow for representing a greater range of event relations than the Allen relations do alone. Freksa [Citation needed!] describes 18 new relations which are made up of disjunctions of Allen's relations, based on the concept of cognitive neighbourhoods. That is, groupings of relations that make intuitive sense and can be derived from compo-

[37] decide on semin-interval translation fn symbol

sitions of Allen relations [NB[38]] [NB[39]].

Many of the Freksa relations can be captured using the semi-intervals [NB[40]] in the box-notation.

For example, the relation *older* corresponds with a disjunction of five Allen relations: *before*, *meets*, *overlaps*, *contains*, and *finished by*.

before	$\boxed{\ }\boxed{a}\boxed{\ }\boxed{b}\boxed{\ }$
meets	$\boxed{\ }\boxed{a}\boxed{b}\boxed{\ }$
overlaps	$\boxed{\ }\boxed{a}\boxed{a,b}\boxed{b}\boxed{\ }$
contains	$\boxed{\ }\boxed{a}\boxed{b,a}\boxed{a}\boxed{\ }$
finished by	$\boxed{\ }\boxed{a}\boxed{b,a}\boxed{\ }$

Table 11: The Freksa relation *older*.

The similarities between these five relations becomes more apparent when they are translated to use semi-intervals instead:

before	$\boxed{\alpha(a),\alpha(b)}\boxed{\alpha(b)}\boxed{\alpha(b),\omega(a)}\boxed{\omega(a)}\boxed{\omega(a),\omega(b)}$
meets	$\boxed{\alpha(a),\alpha(b)}\boxed{\alpha(b)}\boxed{\omega(a)}\boxed{\omega(a),\omega(b)}$
overlaps	$\boxed{\alpha(a),\alpha(b)}\boxed{\alpha(b)}\boxed{\ }\boxed{\omega(a)}\boxed{\omega(a),\omega(b)}$
contains	$\boxed{\alpha(a),\alpha(b)}\boxed{\alpha(b)}\boxed{\ }\boxed{\omega(b)}\boxed{\omega(a),\omega(b)}$
finished by	$\boxed{\alpha(a),\alpha(b)}\boxed{\alpha(b)}\boxed{\ }\boxed{\omega(a),\omega(b)}$

Table 12: The Freksa relation *older* using semi-intervals, before block compressed reduct.

Each of these five strings projects (using a block compressed reduct) to the string $\boxed{\alpha(a),\alpha(b)}\boxed{\alpha(b)}\boxed{\ }$, meaning the relation *a older b* can be represented using just this one string, instead of five. This does raise the question, however, of whether the tradeoffs are worth it: a great reduction in the cardinality of the timeline set can be achieved, but at the cost of using a more complex vocabulary and reducing the precision of the known information.

Unfortunately, only 10 of the 18 Freksa relations can be described using a single string without further complicating the vocabulary which may appear within a box in

[38] expand, clarify 'intuitive', decide between Allen/Allen's, try to use less 'relations'?
[39] include table or graphic of Freksa's relations, eg p18 or p21
[40] duh, he designed them that way

a string. Since all of these new relations are disjunctions of Allen relations, it follows that it may be acceptable to include disjunctions of semi-intervals inside a box, such as $\boxed{\alpha(a) \vee \alpha(b)}$, which is interpreted as one might expect: either $\alpha(a)$ appears in the box, or $\alpha(b)$ does, or they both do. This allowance admits a further 5 Freksa relations to be described in single strings. Of the remaining 3 relations, the *unknown* relation may also be described using a single string, but it is trivial, since it encompasses a disjunction of all 13 Allen relations, and is formed by a simple disjunction of all possible semi-inteval symbols: $\boxed{\alpha(a) \vee \alpha(b) \vee \omega(a) \vee \omega(b) \vee \epsilon}$, or even more simply, with a string consisting of just a single empty box: $\boxed{\ }$.

Below is a table of the 18 Freksa relations, the Allen relations they comprise, and the string which they will project to. [NB[41]]

Freksa	Allen	string
unknown	b, bi, m, mi, s, si, f, fi, d, di, o, oi, e	$\boxed{\ }$
older	b, m, o, di, fi	$\boxed{\alpha(a), \alpha(b)}\boxed{\alpha(b)}\boxed{\ }$
younger	bi, mi, oi, d, f	$\boxed{\alpha(a), \alpha(b)}\boxed{\alpha(a)}\boxed{\ }$
head to head	s, si, e	$\boxed{\alpha(a), \alpha(b)}\boxed{\ }$
tail to tail	f, fi, e	$\boxed{\ }\boxed{\omega(a), \omega(b)}$
survived by	b, m, s, d, o	$\boxed{\ }\boxed{\omega(a)}\boxed{\omega(a), \omega(b)}$
survives	bi, mi, si, di, oi	$\boxed{\ }\boxed{\omega(b)}\boxed{\omega(a), \omega(b)}$
born before death	b, m, s, si, f, fi, d, di, o, oi, e	$\boxed{\alpha(a)}\boxed{\ }\boxed{\omega(b)}$
died after birth	bi, mi, s, si, f, fi, d, di, o, oi, e	$\boxed{\alpha(b)}\boxed{\ }\boxed{\omega(a)}$
precedes	b, m	$\boxed{\alpha(b) \vee \omega(a)}$
succeeds	bi, mi	$\boxed{\alpha(a) \vee \omega(b)}$
contemporary	s, si, f, fi, d, di, o, oi, e	$\boxed{\alpha(a) \vee \alpha(b)}\boxed{\ }\boxed{\omega(a) \vee \omega(b)}$
older contemporary	o, fi, di	$\boxed{\alpha(a), \alpha(b)}\boxed{\alpha(b)}\boxed{\ }\boxed{\omega(a) \vee \omega(b)}$
younger contemporary	oi, f, d	$\boxed{\alpha(a), \alpha(b)}\boxed{\alpha(a)}\boxed{\ }\boxed{\omega(a) \vee \omega(b)}$
surviving contemporary	di, si, oi	$\boxed{\alpha(a)}\boxed{\ }\boxed{\omega(b)}\boxed{\omega(a), \omega(b)}$
survived by contemporary	d, s, o	$\boxed{\alpha(b)}\boxed{\ }\boxed{\omega(a)}\boxed{\omega(a), \omega(b)}$
older and survived by	b, m, o	...
younger and survives	bi, mi, oi	...

Table 13: The Freksa relations and the strings they project to.

[41] tidy up ordering of allens

The last two relations on this list cannot be represented using a single string, and are instead must use conjunctions of pairs of strings:

Freksa	Allen	strings
older and survived by	b, m, o	$\boxed{\alpha(a),\alpha(b)}\boxed{\alpha(b)}\boxed{\ } \wedge \boxed{\ }\boxed{\omega(a)}\boxed{\omega(a),\omega(b)}$
younger and survives	bi, mi, oi	$\boxed{\alpha(a),\alpha(b)}\boxed{\alpha(a)}\boxed{\ } \wedge \boxed{\ }\boxed{\omega(b)}\boxed{\omega(a),\omega(b)}$

Table 14: The remaining Freksa relations and the strings they project to.

While having to use a conjunction of strings may seem like a confounding issue which prevents all the relations from being representable by a single string, it is in fact less problematic than allowing disjunction inside a box, in some ways.

4.2 Enhancing Strings using DRT

Automatic DRS-parsing of plain text is a task which has recently seen new approaches [Citation needed! – shared task, see fsmnlp paper sec 1]. It is possible to leverage the temporal information that features in a DRS to create strings. While the relations generally found using this approach are less specific than those found in TimeML [NB[42]], the advantage of using DRSs is that elements such as event participants are described.

Boxer (Bos, 2008) is one available tool for parsing a text into one or more DRSs, though the temporal information in this version struggles a bit. A newer version of the tool is used as part of the Parallel Meaning Bank [Citation needed!]toolchain, although it has not been made publicly available at present.

4.2.1 Parallel Meaning Bank

Under the assumption that the version of Boxer that is used in PMB would become available at some point, here is a description of PMB, and how the present work utilises the data within it as a demonstration of using text that has been parsed into DRSs can be transformed into strings for temporal reasoning.

Discourse relations are also a thing here.

[42] compare the relations

4.2.2 VerbNet and WordNet

Two resources which are used in PMB and that can be leveraged are VerbNet, which supplies the semantic roles in a DRS, and WordNet, which is used to specify the particular sense of a verb (or other word). The version of Boxer used in the PMB includes this information in its output.

This data can be useful when trying to make links between events which have not been given a specific relation. For example, finding the closest hypernym between two verbs, or checking verbs which share participants may help to inform the building of relations (see § 4.3.2).

4.3 Reasoning with Strings

Strings of events are used to reason about the temporal information they contain: to infer a linear ordering and new relations between the events that were not previously stated. There are several methods which can be employed depending on the specific results that are desired.

Superposition of strings creates new strings which feature all of the constraints of their 'parent' strings, and new relations can be derived by taking the projections of these strings in relation to an intersection of the parent vocabularies. Using resources such as VerbNet and WordNet (see § 4.2.2), the lexical semantics can be leveraged in order to suggest links between events which were not previously explicitly related. Additionally, residuals are employed to determine what relations may need to hold in order for other inferences to be made.

4.3.1 Superposition and Projection

Having extracted strings from an annotated document, such as one of those in the TimeBank [Citation needed!] corpus, superposition can be used to work out the relations which are not yet made explicit. Strings model sets of constraints between the events they mention, and when two strings are superposed, all of these constraints also hold in

the resulting language's strings. This can be shown through an example:

(79) $$\boxed{\boxed{a}\boxed{b}} \,\&_{ux}\, \boxed{\boxed{b}\boxed{c}} = \boxed{\boxed{a}\boxed{b}\boxed{c}}$$

The constraint that event a occurs before event b holds in $s = \boxed{\boxed{a}\boxed{b}}$, and also in the resulting string[43] $s'' = \boxed{\boxed{a}\boxed{b}\boxed{c}}$. In order to prove this, take the *projection* [Figure reference needed!– ref section 3.1] of s'' with respect to the vocabulary of s, and see that it is equal to s:

(80)
$$\begin{aligned}\pi_{voc(s)}(s'') &= \text{bc}(\rho_{voc(s)}(\boxed{\boxed{a}\boxed{b}\boxed{c}})) \\ &= \text{bc}(\boxed{\boxed{a}\boxed{b}}) \\ &= \boxed{\boxed{a}\boxed{b}} = s\end{aligned}$$

The string s'' thus *projects* to the string s, which says that the information in s is contained within s''. Similarly, the constraint that event b occurs before event c holds in both $s' = \boxed{\boxed{b}\boxed{c}}$ and in s''; s'' projects to s'.

Since s'' projects to both to both s and s', s'' also models the constraints that are represented by both s and s', using a single string rather than two strings. However, s'' also projects to the string $\boxed{\boxed{a}\boxed{c}}$, which represents the constraint that event a occurs before event c. This constraint was not represented in either s or s', but is represented in s''. Thus, a new relation between events has been discovered and made explicit.

4.3.2 Event Ontology

Event hierarchy and using verb/word nets to figure out how events might relate to each other, e.g. using verb roles/participants to forge connections

[43] Note that the result of a superposition is a language, but if its cardinality is 1, it can be conflated with its sole member string.

4.3.3 Residuals and Gaps

Finding out what additional information would be required on top of some set of premises in order to make some other conclusion(s) true.

5 Implementation

Various approaches were trialled using different programming languages, including Prolog, JavaScript (Node.js), and Python. Ultimately, Python was chosen for its speed, wide cross-platform availability, and availability of compatible tooling.

Ultimately a widely-useable package was created that could be used either on the web or as a standalone application, as this gave the broadest opportunity to demonstrate the use of the developed technology.

It combined data and technologies from a number of sources to create a package that could be used to reason about temporal information, and augment annotation-based data in a way that is both efficient and simple-to-use [NB[44]]

5.1 Back-end Pipeline

This is the main stuff. Lots of python code all interacting with the data and PMB and Boxer (and the failure of old boxer and why the output is basically theoretical).

Describe all the moving parts, but mostly go into detail about the different API layers (i.e. multiple languages, tooling exposing only certain parts, NLTK having only some parts up to scratch), and the code that I personally wrote to stitch them all together, since that's basically the only sort-of valuable thing in this entire project.

5.2 Front-end Interface

Brython application with some JS to grease the wheels – why not TKinter? because HTML and CSS are convenient and widely used tools and allow for a web-based interface as well as an offline tool (electron for packaging? maybe?) why not just use js frontend and python in the backend? python generators are hard, and don't play nice when you want to use 'next()' to use an effectively 'pausable' function and get a new value.

[44] hard claim to make without a study backing it up...

6 Evaluation

Evaluation was performed on the basis of three main components: asserting that the results produced were not invalid (i.e. did not cause contradictions internally within a string/language or externally within a knowledge-base), performing inferences using the FRACAS semantic test suite, and verifying that all implementation code was tested so as to produce exactly and only the expected results. [NB[45]]

6.1 Timeline Validity

If data is superposed or otherwise manipulated, no information is lost or falsified in the process.

6.2 FRACAS Semantic Test Suite

A de facto standard for testing semantic inferencing.

6.3 Correctness of Code

Python test suites.

[45] how and why does this add *value* – data is compressed? human readable/intuitive?

7 Conclusion

What have I spent the last four and a half years of my life doing?

Bibliography

Allen, J. F. (1983). Maintaining Knowledge About Temporal Intervals. *Communications of the ACM*, 26(11):832–843.

Allen, J. F. and Ferguson, G. (1994). Actions and events in interval temporal logic. *Journal of logic and computation*, 4(5):531–579.

Aristotle. The Internet Classics Archive: Physics IV. `http://classics.mit.edu/Aristotle/physics.4.iv.html`. Accessed: 2018-08-12.

Bos, J. (2008). Wide-Coverage Semantic Analysis with Boxer. In *Proceedings of the 2008 Conference on Semantics in Text Processing - STEP '08*, pages 277–286.

Bottini, R., Crepaldi, D., Casasanto, D., Crollen, V., and Collignon, O. (2015). Space and time in the sighted and blind. *Cognition*, 141:67–72.

Buchner, A. and Funke, J. (1993). Finite-state automata: Dynamic task environments in problem-solving research. *The Quarterly Journal of Experimental Psychology*, 46(1):83–118.

Derczynski, L. and Gaizauskas, R. (2013). Empirical Validation of Reichenbach's Tense Framework. In *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013)*, pages 71–82.

Elgot, C. C. and Rabin, M. O. (1966). Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *The Journal of Symbolic Logic*, 31(2):169–181.

Fernando, T. (2004). A Finite-State Approach to Events in Natural Language semantics. *Journal of Logic and Computation*, 14(1):79–92.

Fernando, T. (2015). The Semantics of Tense and Aspect: A Finite-State Perspective. In Lappin, S. and Fox, C., editors, *The Handbook of Contemporary Semantic Theory*, number August, pages 203–236. John Wiley & Sons.

Fernando, T. (2016a). On Regular Languages Over Power Sets. *Journal of Language Modelling*, 4(1):29–56.

Fernando, T. (2016b). Prior and Temporal Sequences for Natural Language. *Synthese*, 193(11):3625–3637.

Fernando, T. (2018). Intervals and Events with and without Points. In *Workshop on Logic and Algorithms in Computational Linguistics 2018 (LACompLing2018)*, pages 34–46, Stockholm.

Fernando, T. and Nairn, R. (2005). Entailments in finite-state temporality. In *Proceedings of the Sixth International Workshop on Computational Semantics*, pages 128–138, Tilburg University.

Fernando, T., Woods, D., and Vogel, C. (2019). MSO with tests and reducts. In *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing*, pages 27–36.

Freksa, C. (1992). Temporal Reasoning Based on Semi-Intervals. *Artificial Intelligence*, 54:199–227.

Hamblin, C. L. (1972). Instants and intervals. In *The Study of Time*, pages 324–331. Springer.

Kamp, H. and Reyle, U. (1993). *From Discourse to Logic; An Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and DRT*. Dordrecht: Kluwer.

Khourshid, D. (2015). XState. `https://xstate.js.org/docs/about/concepts.html#finite-state-machines`. Accessed: 2021-03-10.

Kowalski, R. and Sergot, M. (1986). A Logic-based Calculus of Events. *New Generation Computing*, 4(1):67–95.

Kumar, P. P. (2005). Effective use of gantt chart for managing large scale projects. *Cost engineering*, 47(7):14.

Lakoff, G. and Johnson, M. (2008). *Metaphors we live by*. University of Chicago press.

Libkin, L. (2004). Monadic Second-Order Logic and Automata. In *Elements of Finite Model Theory*, pages 113–140. Springer-Verlag, Berlin, Heidelberg.

McCarthy, J. (1980). Circumscriptiona form of non-monotonic reasoning. *Artificial intelligence*, 13(1-2):27–39.

McCarthy, J. and Hayes, P. J. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence*, pages 463–502. Edinburgh University Press.

Miller, R. and Shanahan, M. (1999). The Event Calculus in Classical LogicAlternative Axiomatizations. *Electronic Transactions on Artificial Intelligence*, 3:77–105.

Mitchell, W. J. (1980). Spatial form in literature: Toward a general theory. *Critical Inquiry*, 6(3):539–567.

Mueller, E. T. (2008). Event Calculus. In *Foundations of Artificial Intelligence*, volume 3, pages 671–708.

Pustejovsky, J., Knippen, R., Littman, J., and Saurí, R. (2005). Temporal and Event Information in Natural Language Text. *Language Resources and Evaluation*, 39(2-3):123–164.

Pustejovsky, J., Lee, K., Bunt, H., and Romary, L. (2010). ISO-TimeML: An International Standard for Semantic Annotation. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, pages 394–397.

Pustejovsky, J., Verhagen, M., Sauri, R., Littman, J., Gaizauskas, R., Katz, G., Mani, I., Knippen, R., and Setzer, A. (2006). TimeBank 1.2 LDC2006T08. Web Download: `https://catalog.ldc.upenn.edu/LDC2006T08`. Philadelphia: Linguistic Data Consortium.

Reichenbach, H. (1947). *Elements of symbolic logic*. New York: Macmillan.

Rosenberg, D. and Grafton, A. (2013). *Cartographies of time: A history of the timeline*. Princeton Architectural Press.

Shanahan, M. (1997). *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT press.

Stangroom, J. (2009). *Einstein's Riddle: Riddles, Paradoxes, and Conundrums to Stretch Your Mind*. Bloomsbury USA.

Thielscher, M. (1999). From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial intelligence*, 111(1-2):277–299.

Trakhtenbrot, B. A. (1953). On recursive separability. *Doklady Akademii Nauk SSSR*, 88(6):953–956. In Russian.

Van Lambalgen, M. and Hamm, F. (2008). *The Proper Treatment of Events*, volume 6. John Wiley & Sons.

Veanes, M., Hooimeijer, P., Livshits, B., Molnar, D., and Bjorner, N. (2012). Symbolic finite state transducers: Algorithms and applications. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 137–150.

Verhagen, M. (2005). Temporal closure in an annotation environment. *Language Resources and Evaluation*, 39(2-3):211–241.

Wintner, S. (2007). Finite-state technology as a programming environment. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 97–106. Springer.

Woods, D. and Fernando, T. (2018). Improving String Processing for Temporal Relations. In *Proceedings of the 14th Joint ISO-ACL Workshop on Interoperable Semantic Annotation (ISA-14)*, pages 76–86.

Woods, D., Fernando, T., and Vogel, C. (2017). Towards Efficient String Processing of Annotated Events. In *Proceedings of the 13th Joint ISO-ACL Workshop on Interoperable Semantic Annotation (ISA-13)*, pages 124–133.

Yu, S. (1997). Regular Languages. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages, Volume. 1: Word, Language, Grammar*, pages 41–110. Springer-Verlag New York, Inc., New York, NY.

Appendices

Python Code