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

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. A gap is identified, which the remaineder 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 Relations

The seminal work of James F. Allen's *Maintaining Knowledge about Temporal Intervals* has pushed the field since its publication in 1983, and indeed the framework he described drives some of the design decisions in the present work.

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[1]]

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

[1] 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.

it comes to temporal annotation, the international standard is TimeML, specifically ISO-TimeML. [NB[2]]

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.

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[3]]

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.

[2] Also talk about Tango and Verhagen's T-BOX.
[3] This section needs more—Prior's TL, FOL, SOL, Event Calculus?, Frames?, FST?

[NB[4]]

2.3.2 Boxer

Automatic parsing of text into DRSs is a popular task [NB[5]], 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 [Citation needed!] toolchain—however, this version had unfortunately not been made available for general use at the time of writing.

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

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

3 Finite-State Temporality

Finite-state methods may be applied to temporal semantics in an approach known as *finite-state temporality* (Fernando and Nairn, 2005) [NB[6]]. In this chapter, strings are demonstrated as a tool of choice in modelling sequences of times and events for use within this approach. The reasoning behind the creation of such temporal strings is shown, along with their mechanics, and a discussion on the granularity of temporal information, i.e. how much "zoom" is useful when looking at an event.

A number of operations are described for working with these strings, in particular *superposition* for combining the data from multiple strings. These manipulations will prove useful when reasoning about event relations, and also in maximising data density, as will be seen.

The strings are then shown as applied to creating a functional timeline from the temporal information in a piece of text, and also how they may be useful elsewhere, such as in the area of scheduling, using a variation of the Zebra Puzzle [Citation needed!].

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[7]]

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 ex-

[6] Probably need to go into more detail on what FST is as defined here. Definitely need to mention why finite-state as a whole is good. Maybe mention MSO in this opening bit?
[7] Should probably have a citation here?

ample, 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 by the string $\boxed{}\boxed{lt}\boxed{js, lt}\boxed{fa, js, lt}\boxed{js, lt}\boxed{lt}\boxed{}$[8] (an 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). Building this inertial world view into strings allows for certain flexibilities, as real duration does not need to be accounted for when, for example, superposing strings in order to determine the relations between the events they mention (see § 3.1.4, p. 16; § 4.3.1). Additionally, this inertiality[NB[9]]—coupled with the fact that strings are explicit as to whether a particular fluent holds or does not hold at any particular point in time—means that the classic issue of the frame problem is avoided (see McCarthy and Hayes (1969, pp. 30-31)).[10]

[NB[11]]

[NB[12]]

[8] 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).

[9] word?

[10] 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.

[11] Would be useful to discuss logical circumscription and show/discuss how strings are an application since fluents not in the vocabulary are not relevant to the string, and fluents not mentioned in a box are implicitly not true at that time point (but can be made true through superposition) – is this still circumscription?

[12] Definitely should be stated somewhere how realtime durations can still be represented through fluents representing times e.g. $\boxed{sleep\text{-}in, friday}\boxed{friday}$ or something similar

[NB[13]]

[NB[14]]

A *fluent* is a condition which may change over time, and is understood here[15] 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. [NB[16]]

How should events be labelled? Some say an event is defined in part by its participants [Citation needed!]. Many of the procedures defined in this work which use these strings operate primarily on a syntactic level rather than depending on the lexical semantics of individual words, in order to produce an approach that works broadly. As such, in most cases, simple labels suffice for the purposes of describing the mechanics used here—this follows TimeML's standard of using labels such as `"e1"` or `"t2"` for events and times. However, it will be useful to keep reference to a fuller picture of the data when possible, particularly when drawing inferences (see § 3.2.2 and § 4.3.2).

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 moments of time, with $i \in \{1, 2, \ldots, n\}$.

[13] Should also probably go into a small bit of detail about event calculus in lit. section

[14] Perhaps talk about forces, a la Tim's paper? since it couples with the inertia idea, i.e. nothing changes without a force to change it. It's probably too much to mention Newton's 1st law...?

[15] Following the convention set out and used by McCarthy and Hayes (1969), Van Lambalgen and Hamm (2008), Fernando (2016b), among others.

[16] This para sucks

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. A string does not (necessarily) give any sense of real duration, due to the fact that it models an inertial world (see also § 3.1.4, p. 14). 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[NB[17]].

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

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

[17] this is more or less the same as in ISA-13 – should cite?

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 [NB[18]]. Languages allow for grouping strings based on some property, such as their source, which will permit the introduction of ambiguity as will be shown in more detail in § 4.3.1 (see also p. 17). For example,

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

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. 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.

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) [NB[19]]. MSO is a fragment of Second-Order

[18] Maybe mention this in § 3.1 rather than here? See how it goes

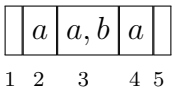
[19] In transfer I just pointed the reader towards Libkin (2004) for a discussion – should maybe go into at least some small detail in the main text this time? Or should that go into the lit review?

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.[Citation needed!]

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. (4) will serve for this purpose, with the positional indices shown underneath the string position with which it corresponds:

(4)



This is a string of length 5, which contains two events, a and b, where b occurs *during* a (see table 3). The linear ordering of a and b can be identified by the string positions in which each occurs (Fernando, 2016a, 2018):

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

where P_a and P_b are unary predicates, and their interpretations [NB[20]] are subsets of $\{1, 2, 3, 4, 5\}$, which is the set of all string positions for the string in eq. (4).

Generally, for any string $s = \sigma_1\sigma_2 \cdots \sigma_n$ with length $n \geq 0 \in \mathbb{N}$ and vocabulary $\mathcal{V}$

[20]check that the interpretation fn is used consistently

[NB[21]], the set $[n]$ of string positions is defined as:

$$[n] := \{1, 2, \ldots, n\} \tag{6}$$

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

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

$$[\![P_v]\!] := \{i \in [n] \mid v \in \sigma_i\} \tag{7}$$

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

$$S_n := \{(i, i+1) \mid i \in [n-1]\} \tag{8}$$

If $\text{MSO}_\mathcal{V}$ is the set of sentences of MSO whose vocabulary is limited to $\mathcal{V}$, then an $\text{MSO}_\mathcal{V}$ model $mod(s)$—which is described by the string s—is defined by the tuple:

$$mod(s) := \langle [n], S_n, \{[\![P_v]\!] \mid v \in \mathcal{V}\} \rangle \tag{9}$$

Given an arbitrary $\text{MSO}_\mathcal{V}$ model M, the string $str(M)$ which describes it can be obtained by inverting eq. (7) to get each set $\sigma_i \in str(M)$, for $i \in [n]$:

$$\sigma_i := \{v \in \mathcal{V} \mid i \in [\![P_v]\!]\} \tag{10}$$

[NB[22]][Citation needed! – Fernando 2016 regular goes into depth about a lot of this stuff]

[21] Should the vocabulary of all strings be $\mathcal{V}$ and the vocabulary of a particular sting be V?
[22] Really not sure how I feel about this last bit

[NB[23]]

There is a theorem due to Büchi, Elgot, and Trakhtenbrot (see, for example, Libkin (2004, p.124, Theorem 7.21)) which states that sentences φ of MSO capture regular languages, i.e. MSO-definability is equivalent to regularity.

[NB[24]]

Analogous Strings A string s will be said to be *analogous* [NB[25]] to some other string s' if the MSO models (see § 3.1.2) corresponding to each string are isomorphic—that is, s and s' are of equal length, there exists a bijective function f mapping the vocabulary of s to the vocabulary of s', and for every $v \in \mathcal{V}_s, [\![P_v]\!] = [\![P_{f(v)}]\!]$:

$$s \sim s' \iff mod(s) \cong mod(s') \tag{11}$$

For example, in eq. (12), $\boxed{\boxed{a}\boxed{b}}$ is analogous to $\boxed{\boxed{c}\boxed{d}}$, while in eq. (13) $\boxed{\boxed{a}\boxed{b}}$ is not analogous to $\boxed{\boxed{c}\boxed{c,d}\boxed{d}}$

$$\boxed{\boxed{a}\boxed{b}} \sim \boxed{\boxed{c}\boxed{d}} \tag{12}$$

$$\boxed{\boxed{a}\boxed{b}} \nsim \boxed{\boxed{c}\boxed{c,d}\boxed{d}} \tag{13}$$

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[26]. By employing this concept in conjunction with

[23] Need to say something about $[\![A]\!]_M$ being the interpretation of A relative to the model M

[24] more here

[25] I'm not sure where to put this section/para

[26] For example, the string on the left hand side of eq. (12) 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.

that of projection (see p. 22) and a set of reference strings—such as those modelling Allen's relations (see table 3)—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.

3.1.3 Granularity: Points vs Intervals vs Semi-intervals

A decision must be made in regards to what is regarded as 'primitive' in a string, in terms of whether a basic component is considered as instantaneous, or as having some duration. In the first view, only points exist, and so an event will have a beginning and an ending (left and right borders), each of which represented by a symbol in the string. In the second, entire intervals are used instead of points. Two points may only be related in three ways: $<$, $=$, $>$. Two intervals, however, may be related in thirteen different ways, precisely, the Allen Relations [Citation needed!].

There are advantages to each approach: ... e.g. intervals are intuitive and infinitely divisible and allow for more specific relation-labels; points are simpler and incomplete information doesn't break the system by forcing disjunctions.

A third option exists: semi-intervals, as described by Freksa (1992). By treating the borders of an event as intervals, this expands the number of available relations between two events to 31.

Ultimately, intervals are chosen for a number of reasons: primarily due to the fact that ISO-TimeML, the international standard for temporal annotation, uses intervals for

its TLINKs, but also they seem the most intuitive [NB[27]].

It's worth noting that there are translations available between the different granularities.

[NB[28]]

3.1.4 String Operations

Many many operations. The most important is **superposition** (and it's more advanced forms)! Also we have block-compression, reduct, vocabulary, projection, border translations. Note also that there are equivalent operations for strings which use points instead of intervals.

[NB[29]]

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 of s may be determined by taking the union of its components:

$$\mathcal{V}_s := \sigma_1 \cup \sigma_2 \cup \cdots \cup \sigma_n \tag{14}$$

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

$$\mathcal{V}_L := \bigcup\{\mathcal{V}_s \mid s \in L\} \tag{15}$$

[27] This is vague and subjective; defend it better
[28] Need to say something in here about the bounded finite intervals.
[29] Discuss the language level operations here too.

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. 5), 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):

$$\mathrm{bc}(s) := \begin{cases} s & \text{if } \mathit{length}(s) \leq 1 \\ \mathrm{bc}(\sigma s') & \text{if } s = \sigma\sigma s' \\ \sigma\mathrm{bc}(\sigma' s') & \text{if } s = \sigma\sigma' s' \text{ with } \sigma \neq \sigma' \end{cases} \tag{16}$$

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[30]:

$$\mathrm{bc}^{-1}(\mathrm{bc}(s)) := \sigma_1^+\sigma_2^+\cdots\sigma_n^+ \quad \text{if } \mathrm{bc}(s) = \sigma_1\sigma_2\cdots\sigma_n \tag{17}$$

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

For example:

(18) $$\mathrm{bc}^{-1}(\boxed{a}\boxed{c}) = \{\boxed{a}\boxed{c}, \boxed{a}\boxed{a}\boxed{c}, \boxed{a}\boxed{c}\boxed{c}, \boxed{a}\boxed{a}\boxed{c}\boxed{c}, \ldots\}$$

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 $\mathrm{bc}(s) = \mathrm{bc}(s')$. 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. 18).

Superposition: In its most basic form, the *superposition* $s \mathbin{\&} s'$ 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[31]:

(19) $$\sigma_1\sigma_2\cdots\sigma_n \mathbin{\&} \sigma'_1\sigma'_2\cdots\sigma'_n := (\sigma_1 \cup \sigma'_1)(\sigma_2 \cup \sigma'_2)\cdots(\sigma_n \cup \sigma'_n)$$

For example:

(20) $$\boxed{a}\boxed{b}\boxed{c} \mathbin{\&} \boxed{a}\boxed{c}\boxed{d} = \boxed{a}\boxed{b,c}\boxed{c,d}$$

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

(21) $$L \mathbin{\&} L' := \bigcup_{n\geq 0}\{s \mathbin{\&} s' | s \in L \cap \Sigma^n, s' \in L' \cap \Sigma^n\}$$

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

The result $L \,\&\, L'$ of superposing two languages L and L' is also a language [NB[32]], and if L and L' are regular languages, then $L \,\&\, L'$ is also regular. If L is accepted by the finite automaton $\langle Q, (2^{\mathcal{V}})^*, (q \xrightarrow{\sigma} r), q_0, F\rangle$ and L' is accepted by the finite automaton $\langle Q', (2^{\mathcal{V}'})^*, (q' \xrightarrow{\sigma'} r'), q'_0, F'\rangle$ then $L \,\&\, L'$ is computed by a finite automaton composed of the automata accepting each L and L': $\langle Q \times Q', (2^{\mathcal{V}\cup\mathcal{V}'})^*, ((q,q') \xrightarrow{(\sigma\cup\sigma')} (r,r')), (q_0, q'_0), F \times F'\rangle$. [NB[33]]

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. (22) below, the result of the superposition accounts for the alternate event sequences in the strings of the first input language.

(22) $$\{\boxed{a}\boxed{b}\boxed{c}, \boxed{a}\boxed{c}\boxed{b}\} \,\&\, \{\boxed{a}\boxed{c}\boxed{d}\} = \{\boxed{a}\boxed{b,c}\boxed{c,d}, \boxed{a}\boxed{c}\boxed{b,d}\}$$

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. (23) 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

[32] Don't forget to mention the connection to regular languages and FSA here
[33] expand on this and its importance

empty set.

(23) $(\boxed{a}\boxed{b}\ \&\ \boxed{c}\boxed{d})\ \&\ (\boxed{a}\boxed{b}\boxed{c}\ \&\ \boxed{a}\boxed{c}\boxed{d}) = \boxed{a,c}\boxed{b,d}\ \&\ \boxed{a}\boxed{b,c}\boxed{c,d} = undefined$

To achieve this, bc-equivalence is exploited and the *inverse block compression* operation (see eq. (17), p. 15) 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'[34]:

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

Now the strings in eq. (23) can be superposed using asynchronous superposition, as in eq. (25) below: [NB[35]]

(25) $$\boxed{a,c}\boxed{b,d}\ \&_*\ \boxed{a}\boxed{b,c}\boxed{c,d} = \{\boxed{a,c}\boxed{a,b,c}\boxed{b,c,d}, \boxed{a,c}\boxed{a,b,d}\boxed{b,c,d}, \boxed{a,c}\boxed{a,b,c}\boxed{a,c,d}\boxed{b,c,d}, \boxed{a,c}\boxed{b,c,d}\}$$

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 lan-

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

[35] double check the below output, not sure of prolog quality... (turing:www/super/superposition.pl)

guages, 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 $s \mathbin{\&_*} s'$, 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 $k > 0$:

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

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. (18):

$$
(27) \qquad pad_3(\boxed{a}\boxed{c}) = \{\boxed{a}\boxed{a}\boxed{c}, \boxed{a}\boxed{c}\boxed{c}\}
$$

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:

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

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. (20), 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 occuring in the same box together, which states that they were occurring at the same moment.

Similarly, in eq. (25), 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):

$$\rho_A(\sigma_1\sigma_2\cdots\sigma_n) := (\sigma_1 \cap A)(\sigma_2 \cap A)\cdots(\sigma_n \cap A) \tag{29}$$

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:

$$\mathcal{V}_{\rho_A(s)} = A \tag{30}$$

$$(\forall a \in A)\ [\![P_a]\!]_{mod(s)} = [\![P_a]\!]_{mod(\rho_A(s))} \tag{31}$$

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:

$$\rho_{\{a,d\}}(\boxed{\ }\boxed{a,b}\boxed{a,b,c}\boxed{a,c,d}\boxed{a,e}\boxed{e}\boxed{\ }) = \boxed{\ }\boxed{a}\boxed{a}\boxed{a,d}\boxed{a}\boxed{\ }\boxed{\ } \tag{32}$$

The resulting string in eq. (32) 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[36]:

$$\text{bc}(\boxed{\ }\boxed{a}\boxed{a}\boxed{a,d}\boxed{a}\boxed{\ }\boxed{\ }) = \boxed{\ }\boxed{a}\boxed{a,d}\boxed{a}\boxed{\ } \tag{33}$$

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 & s'

[36] In the case of eqs. (32) and (33), the event a *contains* the event d, according to Allen's relations (Allen, 1983)[Figure reference needed!].

with respect to each of the strings' vocabularies is equal to the string itself:

$$\mathcal{V}_s \cap \mathcal{V}_{s'} = \varnothing \Longrightarrow \rho_{\mathcal{V}_s}(s \;\&\; s') = s \text{ and } \rho_{\mathcal{V}_{s'}}(s \;\&\; s') = s' \tag{34}$$

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

$$s \;\&\; s' = \boxed{a, c}\boxed{b, d} \tag{35}$$

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

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

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:

$$\pi_A(s) := \mathrm{bc}(\rho_A(s)) \tag{36}$$

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':

$$s \sqsupseteq s' \Longleftrightarrow \pi_{\mathcal{V}_{s'}}(s) = s' \tag{37}$$

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 $s = \mathrm{bc}(s)$ will project to itself, since the reduct of s with respect to its own vocabulary is s.

Borrowing from the examples in eqs. (32) and (33), the temporal data represented by the string on the right hand side of eq. (38) is also contained in the string on the left hand

side:

(38) $$\left[\,\middle|\, a,b \,\middle|\, a,b,c \,\middle|\, a,c,d \,\middle|\, a,e \,\middle|\, e \,\middle|\,\right] \sqsupseteq \left[\,\middle|\, a \,\middle|\, a,d \,\middle|\, a \,\middle|\,\right]$$

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:

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

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

(40) $$L \sqsupseteq L' \iff (\forall s \in L)(\forall s' \in L')\ s \sqsupseteq s'$$

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. 12). 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. 19) 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 $s \mathbin{\&_*} s'$ 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. (25) (p. 18), 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	$\not\sqsupseteq$	a,c	b,d		a,c	a,b,c	b,c,d	$\not\sqsupseteq$	a	b,c	c,d
a,c	a,b,d	b,c,d	$\not\sqsupseteq$	a,c	b,d		a,c	a,b,d	b,c,d	$\not\sqsupseteq$	a	b,c	c,d
a,c	a,b,c	a,c,d	b,c,d	$\not\sqsupseteq$	a,c	b,d	a,c	a,b,c	a,c,d	b,c,d	$\not\sqsupseteq$	a	b,c	c,d
a,c	b,c,d	$\not\sqsupseteq$	a,c	b,d			a,c	b,c,d	$\not\sqsupseteq$	a	b,c	c,d		

Table 1: Failed projections for eq. (25)

In fact, every string in eq. (25) shares the same vocabulary, so by eq. (39), 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[37]. This implication follows from eq. (39), since the vocabulary of every string $s'' \in s \mathbin{\&_*} s'$ is $\mathcal{V}_{s \cup s'} = \mathcal{V}_s = \mathcal{V}_{s'}$:

$$\mathcal{V}_s \cap \mathcal{V}_{s'} = \mathcal{V}_s = \mathcal{V}_{s'} \implies (\forall s'' \in s \mathbin{\&_*} s')\ \neg(s'' \sqsupseteq s \vee s'' \sqsupseteq s') \tag{41}$$

The implication of eq. (41) 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

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

generated string will be spurious and unable to project back—as seen in Table 1.

Conversely, if the vocabularies of the input strings are disjoint, then every resulting string will project back, due to eq. (34):

$$\mathcal{V}_s \cap \mathcal{V}_{s'} = \varnothing \implies (\forall s'' \in s \mathbin{\&_*} s')\ s'' \sqsupseteq s \wedge s'' \sqsupseteq s' \tag{42}$$

This can be seen by returning to the example in eq. (35):

$$\boxed{a}\boxed{b} \mathbin{\&_*} \boxed{c}\boxed{d} = \{\boxed{a,c}\boxed{a,d}\boxed{b,d}, \boxed{a,c}\boxed{b,d}, \boxed{a,c}\boxed{b,c}\boxed{b,d}\} \tag{43}$$

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

Table 2: Preserved projections of eq. (43)

However, in general, the vocabularies of the input strings to asynchronous superposition may overlap without being equal. In this case, some number of the resulting strings may project back, and thus it is necessary to test whether each result is valid. This approach of generating then testing was initially taken in Woods et al. (2017)[38] to ensure that only valid strings were finally produced. Yet, this is not without issue either—consider the following example:

$$\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{} \mathbin{\&_*} \boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{} = \{\boxed{}\boxed{a,b}\boxed{}\boxed{b,c}\boxed{}, \boxed{}\boxed{b}\boxed{a,b}\boxed{}\boxed{b,c}\boxed{}, \boxed{}\boxed{b}\boxed{a}\boxed{}\boxed{b,c}\boxed{}, \boxed{}\boxed{b}\boxed{a}\boxed{c}\boxed{b,c}\boxed{}, \boxed{}\boxed{b}\boxed{a}\boxed{c}\boxed{b}\boxed{}, \ldots\} \tag{44}$$

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

The language result of eq. (44) contains 270 strings, and in fact, only one of these will project back to both of the input strings: namely, $\boxed{\boxed{a}\boxed{b}\boxed{c}}$. This makes an intuitive sense, since the inputs are "a before b" – $\boxed{\boxed{a}\boxed{b}}$ and "b before c" – $\boxed{\boxed{b}\boxed{c}}$, 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. (44) 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.

[NB[39]]

Vocabulary-constrained Superposition:

3.2 Applications

The following will outline some of the ways that the use of string-based FST can applied to problems in NLP.

3.2.1 Timelines from Texts

Strings, as entities 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—though not all languages or cultures perceive time in this way, it is common cross-linguistically to use some spatial reference points when discussing temporality [Citation needed! – see Sec 1 para 1 of: https://onlinelibrary.wiley.com/doi/full/10.1111/cogs.12804]. Regardless

[39] in woods and fernando 2018, integrated testing

of spatial orientation, this perception of time maps well to a sequential representation as is found with strings. Perceived directionality of time is not an issue, as it is a small modification to change the direction of a string.

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.

[NB[40]] [NB[41]]

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

Table 3: Allen interval relations in strings

3.2.2 Inferring New Information

A text which has been manually annotated for temporal information will typically mark up the most important relations between events, at least as the annotator saw them. By extracting and making plain the timeline, any inconsistencies are immediately revealed,

[40] dunno where this table goes yet, just fixing the broken reference
[41] decide about (consistently) using 'b' or '<' for the before relation

whether these are the fault of a problematic document or human error. Further, relations between times and events which were not previously obvious can be spelled out clearly.

This can be extended over multiple documents by keeping track of the semantics of the elements in a string—being able to map between the symbols and their meanings—and linking documents which refer to the same events. This can provide a way for checking consistency between reports which originate from different sources, or a way for augmenting an existing timeline with new data.

Additionally, conclusions can be drawn about whether certain inferences can be made based on existing data by determining the gap between premise strings and strings representing the question statement. That is, deriving the information that could be added to the premises to make the conclusion consistent with the known information (see § 4.3.3).

3.2.3 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.

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[42]]. 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

[42] example here please

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[43]]). 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 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[44]]. It should be noted that several combinations of relations produce a disjunction of relations, which follows from the fact that these combinations of $a \cdot b$ and $b \cdot c$ don't provide enough information to determine the exact relation $a \cdot c$. 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

[43] See transfer p6 – maybe include the graph of $n(n-1)/2, n \geq 2$?
[44] Defend this – I think there is something in ISA13 paper – or remove

event c $\boxed{}\boxed{a}\boxed{}\boxed{c}\boxed{}$ and that event b also occurred before c $\boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{}$ 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 $a \bullet b$ with a precision of $\frac{1}{13}$ 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 box; similarly, $\omega(a)$ represents a negation of the fluent a conjoined with a formula stating that a was true in a previous box.

(45) $$\alpha_a(x) := \exists y(x < y \wedge y \in P_a \wedge \forall z(z < y \rightarrow z \notin P_a))$$

(46) $$\omega_a(x) := \exists y(y < x \wedge y \in P_a \wedge \forall z(y < z \rightarrow z \notin P_a))$$

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

(47) $$s = \boxed{}\boxed{a}\boxed{b}\boxed{}$$

becomes [NB[45]]

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

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 compositions of Allen relations [NB[46]] [NB[47]].

Many of the Freksa relations can be captured using the semi-intervals [NB[48]] in the box-notation.

For example, the relation *older* corresponds with a disjunction of five Allen relations: *before*, *meets*, *overlaps*, *contains*, and *finished by*. The similarities between these five relations becomes more apparent when they are translated to use semi-intervals instead:

[45] decide on semin-interval translation fn symbol
[46] expand, clarify "intuitive", decide between Allen/Allen's, try to use less "relations"?
[47] include table or graphic of Freksa's relations, eg p18 or p21
[48] duh, he designed them that way

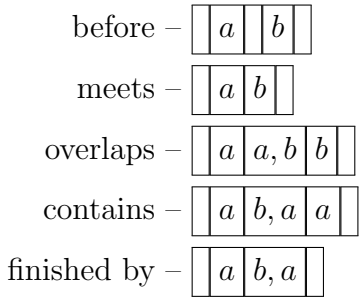


Figure 1: The Freksa relation *older*

Figure 2: 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 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[49]]

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 4: The Freksa relations and the strings they project to

[49] 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	$\alpha(a), \alpha(b) \mid \alpha(b) \mid \;\wedge\; \mid \omega(a) \mid \omega(a), \omega(b)$
younger and survives	bi, mi, oi	$\alpha(a), \alpha(b) \mid \alpha(a) \mid \;\wedge\; \mid \omega(b) \mid \omega(a), \omega(b)$

Table 5: 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[50]], 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

[50] compare the relations

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.

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:

(49) $$\boxed{\boxed{a}\boxed{b}} \;\&_{\mathfrak{w}}\; \boxed{\boxed{b}\boxed{c}} = \boxed{\boxed{a}\boxed{b}\boxed{c}}$$

The constraint that event a occurs before event b holds in $s = \boxed{\boxed{a}\boxed{b}}$, and also in the resulting string[51] $s'' = \boxed{\boxed{a}\boxed{b}\boxed{c}}$. 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:

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

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'.

[51] 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.

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

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[52]]

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

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

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.

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[53]]

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.

[53] 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.

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.

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.

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.

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

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

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

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.

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., 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.

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

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

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.

Appendices

Python Code