Exploring String Representations for Event Data

David Woods
ADAPT Centre
Trinity College Dublin, Ireland
dwoods@tcd.ie

Tim Fernando
ADAPT Centre
Trinity College Dublin, Ireland
tim.fernando@tcd.ie

Carl Vogel
ADAPT Centre
Trinity College Dublin, Ireland
carl.vogel@tcd.ie

1 Introduction

This paper explores using string-based models to effectively represent event data such as might be found in a document annotated with TimeML. It is described how such data may be simply translated to strings, and how to infer information through operations on these strings. Strings are basic computational entities that can be more readily manipulated by machines than the infinite models of predicate logic.

Given a finite set A of fluents, a string $s = \alpha_1 \cdots \alpha_n$ of subsets α_i of A can be construed as a finite model consisting of n moments of time $i \in \{1, \ldots, n\}$ with α_i specifying all fluents (in A) that (as unary predicates) hold at i.

Throughout this paper, a fluent $a \in A$ will be understood as naming an event, and the powerset 2^A of A (consisting of subsets of A) will serve as an alphabet $\Sigma = 2^A$ of an *event-string* $s \in \Sigma^+$. Such strings are finite models of Monadic Second Order logic, which are specifically tailored to fluents, and which are amenable to finite state methods. We will further restrict them in Section 5, with a focus on using Allen Relations, in order to attack the issue of inference.

An event-string $\alpha_1 \cdots \alpha_n$ is read from left to right chronologically, so that any predicates which hold at the moment at index i are understood to have held before another moment indexed by j if and only if $i < j$. The precise duration of each moment is taken as unimportant in the current discussion, and thus the strings model an inertial world, whereby *change* is the only mark of progression from one moment to the next – “But neither does time exist without change” (Aristotle, *Physics IV*). Thus, if $\alpha_i = \alpha_{i+1}$ for any $1 \leq i < n$, then either α_i or α_{i+1} may be safely deleted from s without affecting the interpretation of the string, as the remaining symbol is simply taken as representing a longer moment. This operation of removing repetition from the event-string is known as *block compression* (Fernando, 2016). The inverse of this process introduces repeated elements in an event-string for greater flexibility in manipulating strings. These operations are detailed in Section 2.

2 Superposition and Block Compression

With two strings s and s' of the same length n built from an alphabet Σ, the powerset of some fixed set A, the *superposition* $s \,\&\, s'$ of s and s' is their componentwise union:

$$\alpha_1 \cdots \alpha_n \,\&\, \alpha'_1 \cdots \alpha'_n \;:=\; (\alpha_1 \cup \alpha'_1) \cdots (\alpha_n \cup \alpha'_n)$$

For convenience of notation, we will use boxes rather than curly braces $\{\ \}$ to represent sets in Σ, such that each symbol α in a string s corresponds to exactly one box. For example, with $a, b, c, d \in A$:

$$\boxed{a}\boxed{c} \,\&\, \boxed{b}\boxed{d} \;=\; \boxed{a,b}\boxed{c,d} \;\in\; \Sigma^2$$

Extending superposition to languages L and L' over the same alphabet is a simple matter of collecting the superpositions of strings of equal length from each language:

$$L \,\&\, L' \;:=\; \bigcup_{n\geq 0} \{s \,\&\, s' \mid s \in L \cap \Sigma^n \text{ and } s' \in L' \cap \Sigma^n\}$$

For example, $L \,\&\, \boxed{}^* = L$. If L and L' are regular languages computed by finite automata with transitions $\rightarrow$ and $\rightarrow'$, then the superposition $L \,\&\, L'$ is a regular language computed by a finite automaton with transitions $\Rightarrow$ formed by running $\rightarrow$ and $\rightarrow'$ in lockstep according to the rule

$$\frac{q \overset{\alpha}{\rightarrow} r \qquad q' \overset{\alpha'}{\rightarrow'} r'}{(q,q') \overset{\alpha \cup \alpha'}{\Rightarrow} (r,r')}$$

A disadvantage of this operation is that it requires the string operands to be of equal length, which is an overly specific case. In order to generalise this procedure to strings of arbitrary lengths, we may manipulate the strings to introduce asynchrony. One such manipulation is that we can cause a string $s = \alpha_1 \cdots \alpha_n$ to *stutter* such that $\alpha_i = \alpha_i + 1$ for some integer $0 < i < n$. For example, $\boxed{a}\boxed{a}\boxed{a}\boxed{c}\boxed{c}$ is a stuttering version of $\boxed{a}\boxed{c}$. If a string does not stutter, it is *stutterless*, and we can transform a stuttering string to this state by using "block compression":

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

This function can be applied multiple times to a string, but the output will not change after the first application: $\mathrm{bc}(\mathrm{bc}(s)) = \mathrm{bc}(s)$. We can also use the inverse of this function to generate infinitely many stuttering strings:

$$\mathrm{bc}^{-1}(\boxed{a}\boxed{c}) = \boxed{a}\boxed{c} + \boxed{a}\boxed{a}\boxed{c} + \boxed{a}\boxed{c}\boxed{c} + \boxed{a}\boxed{a}\boxed{c}\boxed{c} + \cdots$$

We can say that any of the strings generated by this inverse block compression are bc-*equivalent*. Precisely, a string s' is bc-equivalent to a string s iff $s' \in \mathrm{bc}^{-1}\mathrm{bc}(s)$.

We can now define the *asynchronous superposition* $s \,\&_*\, s'$ of strings s and s' as the (provably) *finite* set obtained by block compressing the *infinite* language generated by superposing the strings which are bc-equivalent to s and s':

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

For example, $\boxed{a}\boxed{c} \,\&_*\, \boxed{b}\boxed{d}$ will comprise three strings:

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

In order to avoid generating all possible strings when using the inverse block compression, we introduce an upper bound to the length of the strings which will be superposed. It can be shown that with two strings of length n and n', the longest bc-unique string (one which has no shorter bc-equivalent strings) produced through asynchronous superposition will be of length $n + n' - 1$.

3 Upper Bound on Asynchronous Superposition

For all $s, s' \in \Sigma^*$, we define a finite set $s \,\hat{\&}\, s'$ of strings over Σ with enough of the strings in $\mathrm{bc}^{-1}\mathrm{bc}(s) \,\&\, \mathrm{bc}^{-1}\mathrm{bc}(s')$ to form $s \,\&_*\, s'$. The definition proceeds by induction on s and s', with

$$\begin{aligned} \epsilon \,\hat{\&}\, \epsilon &:= \{\epsilon\} \\ \epsilon \,\hat{\&}\, s &:= \emptyset \quad \text{for } s \neq \epsilon \\ s \,\hat{\&}\, \epsilon &:= \emptyset \quad \text{for } s \neq \epsilon \end{aligned}$$

and for all $\alpha, \alpha' \in \Sigma$,

$$\begin{aligned}\alpha s \;\hat{\&}\; \alpha' s' &:= (\alpha \cup \alpha')(\alpha s \;\hat{\&}\; s' \mid s \;\hat{\&}\; \alpha' s' \mid s \;\hat{\&}\; s') \\ &= \{(\alpha \cup \alpha')s'' \mid s'' \in (\alpha s \;\hat{\&}\; s') \cup (s \;\hat{\&}\; \alpha' s') \cup (s \;\hat{\&}\; s')\}\end{aligned}$$

Note that a string in $s \;\hat{\&}\; s'$ might stutter, even if neither of the operands s or s' do (*e.g.* $\boxed{a, c}\boxed{a, c} \in \boxed{a}\boxed{c} \;\hat{\&}\; \boxed{c}\boxed{a}$) – however, it can be made stutterless through block compression.

Proposition 1. *For all $s, s' \in \Sigma^+$ and all $s'' \in s \;\hat{\&}\; s'$,*

$$\text{length}(s'') \;\leq\; \text{length}(s) + \text{length}(s') - 1$$

Proposition 2. *For all $s, s' \in \Sigma^+$,*

$$s \;\hat{\&}\; s' \;\subset\; \mathrm{bc}^{-1}\mathrm{bc}(s) \;\&\; \mathrm{bc}^{-1}\mathrm{bc}(s')$$

and

$$\{\mathrm{bc}(s'') \mid s'' \in s \;\hat{\&}\; s'\} \;=\; s \;\&_*\; s'$$

Now, for any integer $k > 0$ and string $s = \alpha_1 \cdots \alpha_n$ over Σ, we introduce a new function pad_k which will generate the set of strings with length k which are bc-equivalent to s:

$$\begin{aligned}pad_k(\alpha_1 \cdots \alpha_n) \;&:=\; \alpha_1^+ \cdots \alpha_n^+ \;\cap\; \Sigma^k \\ &= \{\alpha_1^{k_1} \cdots \alpha_n^{k_n} \mid k_1, \ldots, k_n \geq 1 \text{ and } \sum_{i=1}^{n} k_i = k\} \\ &\subset\; \mathrm{bc}^{-1}\mathrm{bc}(\alpha_1 \cdots \alpha_n)\end{aligned}$$

For example, $pad_4(\boxed{a}\boxed{c})$ will generate $\{\; \boxed{a}\boxed{a}\boxed{a}\boxed{c},\; \boxed{a}\boxed{a}\boxed{c}\boxed{c},\; \boxed{a}\boxed{c}\boxed{c}\boxed{c} \;\}$. We can use this new function in our calculation of asynchronous superposition, to limit the generation of strings from the inverse block compression step. Since we know from Proposition 1 that the maximum possible length we might need is $n + n' - 1$, we can use this value in the pad function to just generate the strings of that length, giving us a new definition of asynchronous superposition:

Corollary 3. *For any $s, s' \in \Sigma^+$ with nonzero lengths n and n' respectively,*

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

Neither $s \;\hat{\&}\; s'$ nor $pad_{n+n'-1}(s) \;\&\; pad_{n+n'-1}(s')$ need be a subset of the other, even though, under the assumptions of Corollary 3, both sets block compress to $s \;\&_*\; s'$.

4 Event Representation

Now we may use asynchronous superposition to generate the 13 strings in $\boxed{\;}\boxed{e}\boxed{\;} \;\&_*\; \boxed{\;}\boxed{e'}\boxed{\;}$, each of which corresponds to one of the unique interval relations in Allen (1983). No more than one of these relations may hold between any two fluents, and thus each of the 13 generated event-strings exists in a distinct possible "world". We use the empty box $\boxed{\;}$ as a string of length 1 (not to confused with the empty string ϵ, which is length 0) to bound events, allowing us to represent the fact that they are finite – they have a beginning and ending point. It is prudent to assume that we will deal only with finite event data, such that there are no fluents which do not have both an associated start-point and end-point. If such a fluent were to occur, it would be trivial to add it to every position in the string.

The bounding boxes represent the time before and after the event occurs, during which no other fluents $a \in A$ hold. The Allen Relation Strings are laid out below:

$e = e'$	$\boxed{\;}\boxed{e, e'}\boxed{\;}$	equal
$e\ \mathbf{s}\ e'$	$\boxed{\;}\boxed{e, e'}\boxed{e'}\boxed{\;}$	starts
$e\ \mathbf{si}\ e'$	$\boxed{\;}\boxed{e, e'}\boxed{e}\boxed{\;}$	starts (inverse)
$e\ \mathbf{f}\ e'$	$\boxed{\;}\boxed{e'}\boxed{e, e'}\boxed{\;}$	finishes
$e\ \mathbf{fi}\ e'$	$\boxed{\;}\boxed{e}\boxed{e, e'}\boxed{\;}$	finishes (inverse)
$e\ \mathbf{d}\ e'$	$\boxed{\;}\boxed{e'}\boxed{e, e'}\boxed{e'}\boxed{\;}$	during
$e\ \mathbf{di}\ e'$	$\boxed{\;}\boxed{e}\boxed{e, e'}\boxed{e}\boxed{\;}$	during (inverse)
$e\ \mathbf{o}\ e'$	$\boxed{\;}\boxed{e}\boxed{e, e'}\boxed{e'}\boxed{\;}$	overlaps
$e\ \mathbf{oi}\ e'$	$\boxed{\;}\boxed{e'}\boxed{e, e'}\boxed{e}\boxed{\;}$	overlaps (inverse)
$e\ \mathbf{m}\ e'$	$\boxed{\;}\boxed{e}\boxed{e'}\boxed{\;}$	meets
$e\ \mathbf{mi}\ e'$	$\boxed{\;}\boxed{e'}\boxed{e}\boxed{\;}$	meets (inverse)
$e < e'$	$\boxed{\;}\boxed{e}\boxed{\;}\boxed{e'}\boxed{\;}$	before
$e > e'$	$\boxed{\;}\boxed{e'}\boxed{\;}\boxed{e}\boxed{\;}$	after

These Allen Relations are included in the attributes of ISO-TimeML (Pustejovsky et al., 2010), the standard markup language used for event annotation in texts, as TLINKs. By extracting the TLINKs from an annotated document, and translating them to our event-string representation, we may begin to reason about the relationships between annotated events which do not have an associated TLINK in the markup. For example, the document may give us a relation between events e and e', and another relation between e' and e'', and from this we may infer the possible relations between e and e''.

As asynchronous superposition is a commutative, associative, homomorphic binary relation over event-strings, we may superpose arbitrary numbers of event-strings: $s_1 \&_* \cdots \&_* s_n$. Via Proposition 1, we know that superposing n unconstrained bounded event-strings will generate strings of maximum length $2n + 1$:

Given that each string $s_i \in \{s_1, \ldots, s_n\} = \boxed{\;}\boxed{e_i}\boxed{\;}$, with each $e_i \in A$
For $n = 2$: $s_1 \&_* s_2$
From Proposition 1, the maximum length of the result is $3 + 3 - 1 = 5 = 2(2) + 1$
Assume true for $n = p$, thus the maximum length of $s_1 \&_* \cdots \&_* s_p$ is $2(p) + 1$
Prove for $n = (p + 1)$
$s_1 \&_* \cdots \&_* s_{p+1} = s_1 \&_* \cdots \&_* s_p \&_* s_{p+1}$
$= s_{1 \ldots p} \&_* s_{p+1}$
From Proposition 1, the maximum length of the result is $(2(p) + 1) + 3 - 1$
$= 2(p) + 3$
$= 2(p + 1) + 1$
Thus true for $(p + 1)$, and by induction, true for any $n \geq 2$

Note, however, that superposing even a relatively small number of unconstrained bounded events leads to a massive blowup in the number of outcomes, or possible worlds. Current methods to compute the result of more than five of these events have so far been unsuccessful, reaching the memory limits of the available hardware:

$$\begin{aligned}
&2 \text{ bounded events} \rightarrow 13 \text{ outcomes}\\
&3 \text{ bounded events} \rightarrow 409 \text{ outcomes}\\
&4 \text{ bounded events} \rightarrow 23917 \text{ outcomes}\\
&5 \text{ bounded events} \rightarrow 2244361 \text{ outcomes}
\end{aligned}$$

Clearly, simply superposing bounded events in this manner is not feasible, as it is unreasonable to expect that any given document should contain five or fewer events. In order to avoid generating such a large number of computed event-strings, it is necessary to add constraints where appropriate to limit the strings that may be considered allowable for a particular context.

Interestingly, because each unconstrained bounded event-string contains exactly one fluent, we may determine the maximum possible length of a string generated by superposition, $2n + 1$, from the size of the set A of fluents, where $n = |A|$. By keeping track of $|A|$, we ensure that the length of the string will always be finite, opening up the possibility of using methods from constraint satisfaction, due to the finite search space.

5 Constraints on Event-Strings

Two approaches to constraints may be implemented, which are not mutually exclusive. The first is to prevent unwanted strings from being generated, based on the nature of the operand strings, and the second is to remove disallowed strings from the set of outputs. The former approach is preferred from a computational standpoint, as there is less data to store and process. For either, we define some properties of what we may consider to be a *well-formed event-string*.

We assume that every fluent we encounter has exactly one beginning and one ending – that is, that events do not *resume* once they have ended. Events of the same type may stop and start frequently, but by assuming that every instance of an event will have a uniquely identifying fluent, we can discard any strings which feature such a resumption. In this way, fluents are *interval-like*. We define the function ρ_a on strings of sets to component-wise intersect with $\{a\}$ for any $a \in A$ (Fernando, 2016)

$$\rho_a(\alpha_1 \cdots \alpha_n) := (\alpha_1 \cap \{a\}) \cdots (\alpha_n \cap \{a\})$$

Applying block compression to an event-string which has been reduced with ρ_a should produce a single string: $\boxed{}\boxed{a}\boxed{}$. For example,

$$\mathrm{bc}(\rho_{ei9}(\boxed{}\boxed{ei9}\boxed{t1, ei9}\boxed{t1}\boxed{})) = \boxed{}\boxed{ei9}\boxed{}$$

Additionally, fluents may be referred to multiple times by different TLINKs in an annotated document, and we assume that they will be *consistent* within the context of that document *i.e.* if a relation holds between e and e', and a relation holds between e' and e'', then both instances of e' refer to the same fluent. In this case, if a relation also holds between e and e'', then this relation should not contradict the other two relations. For example, if $e > e'$ and $e' > e''$, then it should be impossible for a well-formed event-string to also have the relation $e < e''$, as this would break the interval-like fluent constraint mentioned above.

These last two points are interesting in particular, as they lead to a specific kind of superposition between strings $s, s' \in \Sigma^+$ when some symbol $\alpha \in s$ is equal to some other symbol $\alpha' \in s'$. In this scenario, the symbols must unify when superposing the strings, in order to create a well-formed event-string in accordance with the above two constraints. To achieve this, when a symbol α in s is also present in s', and the asynchronous superposition of these strings is desired, padding is carried out as normal, but superposition is only permitted of those results of padding in which the indices of the matching symbols are equal:

$$\alpha_1 \cdots \alpha_n \,\&\, \alpha'_1 \cdots \alpha'_n \;:=\; \{(\alpha_1 \cup \alpha'_1) \cdots (\alpha_n \cup \alpha'_n) \mid \alpha_i = \alpha'_j \Rightarrow i = j\}$$

Finally, we may also introduce simple binary constraints if external information is available (*e.g.* "the string may not begin with e"), and these might be simply intersected with the result of a superposition: $(s \mathbin{\&_*} s') \cap C$, where C represents the constraints to be applied.

6 Grounding

The TIMEBANK Corpus (Pustejovsky et al., 2003) provides a large number of documents annotated with TimeML, from which we may extract the event data – in particular the TLINKs. These elements in the markup indicate the relation holding between two fluents which was found in the text of the document. Though not every fluent will necessarily be linked with another in this manner, a majority will be. As mentioned in Section 4, attempting to generate all of the possible worlds becomes difficult when using just the unconstrained bounded event-strings alone, as there are just too many (rarely, if ever, five or fewer). Instead, we begin by looking at just those fluents which are linked to another by Allen Relation.

As each relation corresponds exactly to one possible model, we translate them immediately to the appropriate event-strings, and superpose these according to the constraints mentioned in Section 5. This allows us to avoid simply superposing based on the fluents, and bypasses having to generate the initial 13 possibilities. In this way, we may generate a much smaller set of possible outcomes from a larger number of bounded events.

In order to give a more concrete understanding, let us take a pair of examples of TLINKs from a TimeML document:

```
<TLINK lid="l9" relType="IS_INCLUDED" timeID="t1"
    relatedToEventInstance="ei9" origin="USER"/>
<TLINK lid="l10" relType="BEFORE" eventInstanceID="ei9"
    relatedToEventInstance="ei10" origin="USER"/>
```

The TLINK nodes will have either a `timeID` or `eventInstanceID` attribute, as well as either a `relatedToTime` or `relatedToEventInstance` attribute, as well as a `relType` attribute. The Allen Relation specified by `relType` will correspond to one of the 13 event-strings given in Section 4, which will in turn determine which formula the new event-string will satisfy. The examples above give the event-strings $\boxed{}\boxed{ei9}\boxed{t1, ei9}\boxed{t1}\boxed{}$, and $\boxed{}\boxed{ei9}\boxed{}\boxed{ei10}\boxed{}$, respectively[1].

A drawback here is that for this to be effective, it relies on the events being heavily constrained by their interrelations. If there are too few TLINKs relative to the number of events, we still run into the problem of combinatorial explosion.

An additional issue in computation of the superposition of events arises from unordered data leading to a much less efficient calculation of final results. For example, $\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{} \mathbin{\&_*} \boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{} \mathbin{\&_*} \boxed{}\boxed{c}\boxed{}\boxed{d}\boxed{}$ and $\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{} \mathbin{\&_*} \boxed{}\boxed{c}\boxed{}\boxed{d}\boxed{} \mathbin{\&_*} \boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{}$ should produce the same, single output: $\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{}\boxed{d}\boxed{}$, which they do. However, due to the respective orderings, the first sequence will arrive at that conclusion much faster as $\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{} \mathbin{\&_*} \boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{}$ has one outcome ($\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{}$), which can immediately be asynchronously superposed with $\boxed{}\boxed{c}\boxed{}\boxed{d}\boxed{}$ to produce the final output. However, $\boxed{}\boxed{a}\boxed{}\boxed{b}\boxed{} \mathbin{\&_*} \boxed{}\boxed{c}\boxed{}\boxed{d}\boxed{}$ has 321 output strings, each of which must be individually asynchronously superposed with $\boxed{}\boxed{b}\boxed{}\boxed{c}\boxed{}$, only to come to the same conclusion, as only one of these results is consistent.

One potential way to work around this pitfall is a grouping and ordering stage, where initially only events linked by some relation may be superposed, and only after the operand strings have been sorted to some optimal order, whereby the event-strings with the most shared fluents are grouped. It may be prudent to only perform superposition at all on event-strings which may be linked through one or more relations or shared fluents. In this way, new, underspecified events may be formed from the output strings.

[1] Using https://www.scss.tcd.ie/∼dwoods/timeml/ to quickly extract TLINKs from a TimeML document and translate them to event-strings

Consider the scenario with $e_1, \ldots, e_8 \in A$, and the following Allen Relations:

$$\begin{aligned} e_1 &< e_4 \\ e_1 &\ \mathbf{m}\ e_2 \\ e_2 &\ \mathbf{di}\ e_3 \\ e_5 &\ \mathbf{s}\ e_7 \\ e_8 &> e_5 \end{aligned}$$

Let us cluster the fluents as follows: for each fluent $a \in A$, fix a set $P = \{a\}$, and a set S whose members are these sets P.

$$S = \{\{e_1\}, \{e_2\}, \{e_3\}, \{e_4\}, \{e_5\}, \{e_6\}, \{e_7\}, \{e_8\}\}$$

Next, check for an Allen Relation between each pair of fluents a and a' – if a relation exists, add the fluent a' to the set P.

$$S' = \{\{e_1, e_4, e_2\}, \{e_2, e_3\}, \{e_3\}, \{e_4\}, \{e_5, e_7\}, \{e_6\}, \{e_7\}, \{e_8, e_5\}\}$$

Finally, for each pair of sets P, Q, if $|P \ \cap \ Q| > 0$, form $R = P \ \cup \ Q$, adding R to the set S'' and discarding P, Q. Add the remaining sets from S' to S''.

$$S'' = \{\{e_1, e_4, e_2, e_3\}, \{e_5, e_7, e_8\}, \{e_6\}\}$$

We might form the underspecified event groups E_1 and E_2 to refer to these first two clusters, at which point we may freely treat these groups as normal bounded events, and perform asynchronous superposition on their event-strings, as well as with that of e_6 – reducing the number of inputs from 8 to 3.

Additionally, various weightings might be considered as a method of priority-ordering in the case of a large A, such as the number of component events in an underspecified event groups, or the number of relations linking to a particular event.

7 Conclusion

We have explored in this work the possibility of using strings as basis for modelling event data, motivated by their nature as computational entities. The operation of asynchronous superposition was described for composing strings which represent finite, bounded events, as well as its limits in terms of blowup when the operation is repeated in sequence. The problem is addressed by constraining the strings which may be superposed, with the 13 unique Allen Relations forming the main part of these, as these can be found in annotated corpora such as TIMEBANK, using the TimeML standard.

Future work on this topic will examine using alternative model-bases to approach the same issue, such as using finite state automata, or a hybrid string/FSA approach, as well as the potential of employing methods from distributed computing in order to tackle the combinatorial explosion that occurs in asynchronously superposing unconstrained bounded events.

References

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

Fernando, T. (2016). Prior and temporal sequences for natural language. *Synthese 193*(11), 3625–3637.

Pustejovsky, J., P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim, D. Day, L. Ferro, et al. (2003). The TIMEBANK Corpus. In *Corpus Linguistics*, Volume 2003, pp. 647–656. Lancaster, UK.

Pustejovsky, J., K. Lee, H. Bunt, and L. Romary (2010). ISO-TimeML: An International Standard for Semantic Annotation. In *LREC*, Volume 10, pp. 394–397.