Entailments in finite-state temporality

Tim Fernando and Rowan Nairn

Computer Science Department, Trinity College Dublin

1 Introduction

The "surge in use of finite-state methods" ([10]) in computational linguistics has largely, if not completely, left semantics untouched. The present paper is directed towards correcting this situation. Techniques explained in [1] are applied to a fragment of temporal semantics through an approach we call *finite-state temporality*. This proceeds from the intuition of an event as "a series of snapshots" ([15]; see also [12]), equating snapshots with symbols that collectively form our alphabet. A sequence of snapshots then becomes a string over that alphabet, evoking comic/film strips. Jackendoff has, among others, objected to conceptualizing events in terms of snapshots ([8]). To counter these objections, we step up from events-as-strings to event-types-as-regular languages ([5, 6]), recognizing the need for variable granularity. Beyond the introduction of disjunction implicit in the step from a single string up to a set of strings, we obtain a useful logic from the regular operations and a careful choice of the snapshots (constituting our alphabet).

To a first approximation, a snapshot is simply a set of formulas, or fluents, described to hold at some point in time. These sets are *not* to be construed as exhaustive: the absence of a fluent φ from a set α need not mean that φ does not hold at the time described by α — only that α does not describe φ as holding at that time. Hence, a snapshot α is at least as informative as β if β is a subset of α, $\beta \subseteq \alpha$. We define a subsumption relation $\trianglerighteq$ in section 2 that extends this notion of underspecification to strings of the same length and more generally, to languages.

Building on $\trianglerighteq$, we modify Koskenniemi's *restriction* operator $=>$ ([1], page 64) in section 3 to express various entailments *as* regular languages. Examples include the constraints (1) and (2) on our alphabet.

$$\boxed{\varphi, \sim\varphi} \;\Rightarrow\; \emptyset \tag{1}$$

$$\Box \;\Rightarrow\; \boxed{\varphi} \mid \boxed{\sim\varphi} \tag{2}$$

We enclose snapshots such as $\boxed{\varphi, \sim\varphi}$ and $\boxed{\varphi}$ in boxes (rather than $\{\cdot\}$) to reinforce the movie strip analogy. We distinguish the empty language $\emptyset$ in (1) from the empty snapshot $\Box$ in (2), and write $|$ for non-deterministic choice (noting $L|\emptyset = L \neq L|\Box$ unless $\Box \in L$). Leaving the precise semantics of $\Rightarrow$ for section 3, the idea is that (1) says

> a snapshot never includes a fluent φ alongside its negation $\sim\varphi$

while (2) says

> every snapshot specifies that either φ or $\sim\varphi$ holds.

(1) is a constraint we will wish to apply universally, over every fluent φ. (2) is not, as an event need not decide every fluent φ.

For a concrete English example, consider the pair (a), (b), with speech time S put after the event described as past.

(a) It rained for a week (or more). $\boxed{0(x),\ \text{rain}}\boxed{\text{rain}}^*\boxed{\text{week}(x),\ \text{rain}}\Box^*\boxed{S}$

(b) It rained for a day (or more). $\boxed{0(x),\ \text{rain}}\boxed{\text{rain}}^*\boxed{\text{day}(x),\ \text{rain}}\Box^*\boxed{S}$

The fluents $0(x)$, $\text{week}(x)$ and $\text{day}(x)$ mark the passage of (respectively) 0 time, a week's time and a day's time since x. We can derive (b) from (a) through the "week-contains-day" constraint

$$\boxed{0(x)}\Box^*\boxed{\text{week}(x)} \quad \Rightarrow \quad \Box^+\boxed{\text{day}(x)}\Box^+$$

that beefs up the regular language in (a) to

$$\boxed{0(x),\ \text{rain}}\boxed{\text{rain}}^*\boxed{\text{day}(x),\ \text{rain}}\boxed{\text{rain}}^*\boxed{\text{week}(x),\ \text{rain}}\Box^*\boxed{S}$$

which in turn subsumes the regular language in (b).

But how do we go from the English sentences in (a) and (b) to their regular languages? The answer given in [6] appeals to common sense laws of inertia, formulated with the help of fluents $\mathsf{F}\varphi$ that assert some force is acting on φ.[1] These laws boil down in the present context to (3) and (4).

$$\boxed{\varphi}\Box \quad \Rightarrow \quad \Box\boxed{\varphi} \mid \boxed{\mathsf{F}\varphi}\Box \tag{3}$$

$$\Box\boxed{\varphi} \quad \Rightarrow \quad \boxed{\varphi}\Box \mid \boxed{\mathsf{F}\varphi}\Box \tag{4}$$

[1] Inertia has figured prominently in logical approaches to AI since McCarthy and Hayes described the *frame problem* for specifying the effects and non-effects of actions ([11]). Its importance in linguistic semantics (recognized since [2, 3]) is one of the basic messages of [14]. A recent investigation of it is [7], which applies methods from [13].

Space limitations prevent us from presenting a careful comparison of our approach with these and others in the literature — or, for that matter, from detailing applications of finite-state temporality to Vendler classes and a Reichenbachian system of tense and aspect ([4, 6]).

(3) can be read as

> φ persists unless some force is applied to it.

(4) is (3) run in reverse:

> φ persists backwards unless some force was applied to it.

Together, (3) and (4) characterize what [6] calls *inertially complete* strings. The operator $\Rightarrow$ above is (to the best of our knowledge) new, although it is an obvious outcome of marrying $\trianglerighteq$ with Koskenniemi's $=>$ (left out from [6]).

Whereas section 3 describes *what* it means to satisfy a $\Rightarrow$-constraint, section 4 focuses on *how* to satisfy such a constraint. Section 3 picks out the regular language of strings respecting the constraint; section 4 identifies a regular *relation* that updates an input language to a language respecting the constraint. If restriction $=>$ is the inspiration behind $\Rightarrow$ in section 3, *replacement* $->$ in [1] lurks near section 4. But within section 4, a rather different mechanism is employed for effecting transduction, viz *superposition* $\&$. We meet $->$ in section 5, where we compare languages that are in a precise (model-theoretic) sense semantically equivalent but may differ computationally. (That is, computing with one language may be costlier than computing with another semantically equivalent one.)

2 Subsumption and subsumption closure

Here we briefly introduce the finite-state setting in which we carry out temporal inference, drawing freely from [5] (where model-theoretic interpretations are outlined). As always when working with finite-state machines, we must specify an alphabet Σ. We assume a finite set Φ of fluents and build symbols out of sets of such fluents (setting $\Sigma = \mathsf{Pow}(\Phi)$[2]). These are our snapshots. Strings $\alpha_1 \cdots \alpha_n \in \Sigma^*$ of these describe temporal sequences where all the fluents in α_i are asserted to hold at the i^{th} time point.

Finite-state machines are defined by a tuple $\langle Q, q_0, F, \rightarrow \rangle$ with Q the set of states, q_0 the initial state, F the set of final states and $\rightarrow$ the transition relation. For acceptors, $\rightarrow$ is a subset of $Q \times \Sigma \times Q$, and for transducers, a subset of $Q \times (\Sigma \cup \{\epsilon\}) \times (\Sigma \cup \{\epsilon\}) \times Q$. We use the standard regular expression operators (e.g. Kleene star *) and define our own as we need

[2] We will place extra restrictions on Σ in section 3, illustrating the meaning of $\Rightarrow$ in the process (rather than explicitly introducing these constraints now).

them in terms of transformations of finite-state machines or in terms of more primitive operators.

The subsumption relation $s \trianglerighteq s'$ on strings s and s' says s explicitly contains all the information in s', and is defined by

$$\alpha_1 \cdots \alpha_n \trianglerighteq \beta_1 \cdots \beta_k \quad \text{iff} \quad n = k \text{ and for } 1 \leq i \leq n,\ \beta_i \subseteq \alpha_i \ .$$

We extend $\trianglerighteq$ to languages, taking L to be more informative than L' if every string in L subsumes some string in L'

$$L \trianglerighteq L' \quad \text{iff} \quad (\forall s \in L)(\exists s' \in L')\ s \trianglerighteq s' \ .$$

These definitions allow us to routinely conflate strings with singleton languages, following customary practice. The *subsumption closure of* L, $L_{\underline{\trianglerighteq}}$, consists of all strings that subsume a string in L

$$L_{\underline{\trianglerighteq}} \quad =_{def} \quad \{s \in \Sigma^* \mid s \trianglerighteq L\} \ = \ \left\{s \in \Sigma^* \mid (\exists s' \in L)\ s \trianglerighteq s'\right\} \ .$$

$L_{\underline{\trianglerighteq}}$ will be useful later on. For now, note that

$$\begin{aligned} L_{\underline{\trianglerighteq}} &= \bigcup_{s \in L} s_{\underline{\trianglerighteq}} \\ (L|L')_{\underline{\trianglerighteq}} &= L_{\underline{\trianglerighteq}} \mid L'_{\underline{\trianglerighteq}} \\ (LL')_{\underline{\trianglerighteq}} &= L_{\underline{\trianglerighteq}} L'_{\underline{\trianglerighteq}} \end{aligned}$$

and

$$L \trianglerighteq L' \quad \text{iff} \quad L \subseteq L'_{\underline{\trianglerighteq}} \ .$$

Moreover, $\cdot_{\underline{\trianglerighteq}}$ is a regular operation: if L is accepted by the finite-state machine $\langle Q, q_0, F, \rightarrow \rangle$, then $L_{\underline{\trianglerighteq}}$ is accepted by $\langle Q, q_0, F, \rightsquigarrow \rangle$, where

$$q \overset{\beta}{\rightsquigarrow} q' \quad \text{iff} \quad (\exists \alpha \subseteq \beta)\ q \overset{\alpha}{\rightarrow} q' \ .$$

The predicate-labeled transitions described in [16] provide us with a method for more efficiently dealing with the potentially expensive $\trianglerighteq$-closure operation. Instead of multiplying the number of transitions between any two states we simply replace a transition-label α with a predicate P that is satisfied by all and only the supersets of α.

$$P = (\lambda \beta)\ \beta \supseteq \alpha$$

Van Noord and Gerdemann have shown that all the standard regular operators, including determinization and minimization, can be generalized to predicate-labeled transitions. As a bonus this lets us relax the restriction of a finite set of fluents: the predicate P is true of any superset of α. We never have to explicitly specify the complete alphabet. In fact, our alphabet Σ is just $\square_{\underline{\trianglerighteq}}$.

3 Entailments encoded as regular languages

Our formulation of rules is inspired by the constructions available in the Xerox toolkit for morphological analysis ([1]). We will introduce analogs to two such constructions. The first is the *restriction operator* => which places constraints on an individual language (via intersection).

Restrictions take the form $L \Rightarrow L' \_$ or $L \Rightarrow \_\ L'$. These rules constrain the occurrence of substrings of type L to the contexts in which they are respectively preceded or followed by a string of type L'. They can be translated into regular expressions which accept all and only those strings that adhere to the rule. ($\overline{L}$ is the complement $\Sigma^* - L$ of L.)

$$L \Rightarrow L' \_ \quad =_{def} \quad \overline{\overline{\Sigma^*\ L'}\ L\ \Sigma^*} \tag{5}$$

$$L \Rightarrow \_\ L' \quad =_{def} \quad \overline{\Sigma^*\ L\ \overline{L'\ \Sigma^*}} \tag{6}$$

Read (5) as

> never have a substring that doesn't end with an L' before one that begins with an L

and (6) as

> never have a substring that doesn't begin with an L' after one that ends with an L.

For example, $ab \Rightarrow \_r$ means that all substrings ab must be followed by an r. The language includes strings such as *abracadabra* and xyz, but not abc or dab. The subset of a language that adheres to the rule can then be found simply by intersecting it with the language defined by the rule.

In our case, we are dealing with strings of snapshots so we want to make sure that the rules not only constrain the context of L but also any language which subsumes L. To this end, we change (5) to (7) and (6) to (8):

$$L \Rightarrow L' \_ \quad =_{def} \quad \overline{\overline{(\Box^*\ L')_{\trianglerighteq}}\ (L\ \Box^*)_{\trianglerighteq}} \tag{7}$$

$$L \Rightarrow \_\ L' \quad =_{def} \quad \overline{(\Box^*\ L)_{\trianglerighteq}\ \overline{(L'\ \Box^*)_{\trianglerighteq}}} \tag{8}$$

A string s belongs to the language (7) if and only if

$$(\forall \text{ strings } l, c, r \text{ such that } lcr = s \text{ and } c \trianglerighteq L)\ l \trianglerighteq \Box^* L'$$

and to the language (8) if and only if

$$(\forall \text{ strings } l, c, r \text{ such that } lcr = s \text{ and } c \trianglerighteq L)\ r \trianglerighteq L'\Box^* \ .$$

To these, we add a third construction which, although of limited use in the Xerox setting, is quite serviceable, given the underspecification we build into our symbols.

$$L \Rightarrow L' \quad =_{def} \quad \overline{\Box_{\trianglerighteq}{}^{*}\ (L_{\trianglerighteq} \cap \overline{L'_{\trianglerighteq}})\ \Box_{\trianglerighteq}{}^{*}} \tag{9}$$

Read (9) as

> any substring that subsumes L must also subsume L'.

A string s belongs to the language (9) if and only if

$$(\forall \text{ strings } l, c, r \text{ such that } lcr = s \text{ and } c \trianglerighteq L)\ c \trianglerighteq L' \ .$$

Take for example, (1)

$$\boxed{\varphi, \sim\varphi} \ \Rightarrow \ \emptyset \ .$$

This translates to

$$\begin{aligned}
\overline{\Box_{\trianglerighteq}{}^{*}\ \left(\boxed{\varphi, \sim\varphi}_{\trianglerighteq} \cap \overline{\emptyset_{\trianglerighteq}}\right)\ \Box_{\trianglerighteq}{}^{*}} \ &= \ \overline{\Box_{\trianglerighteq}{}^{*}\ \left(\boxed{\varphi, \sim\varphi}_{\trianglerighteq} \cap \Box_{\trianglerighteq}{}^{*}\right)\ \Box_{\trianglerighteq}{}^{*}} \\
&= \ \overline{\Box_{\trianglerighteq}{}^{*}\ \boxed{\varphi, \sim\varphi}_{\trianglerighteq}\ \Box_{\trianglerighteq}{}^{*}} \\
&= \ \overline{\left(\Box^{*}\ \boxed{\varphi, \sim\varphi}\ \Box^{*}\right)_{\trianglerighteq}}
\end{aligned}$$

which is simply a way of saying that no symbol in a string should contain both φ and its negation. That is, the language expressed by (1) is just

$$\{\alpha \in \Sigma \mid \boxed{\varphi, \sim\varphi} \not\trianglelefteq \alpha\}^{*}.$$

Letting φ range over every fluent, we get the restricted alphabet

$$\Sigma_1 \quad =_{def} \quad \bigcap_{\varphi \in \Phi} \{\alpha \in \Sigma \mid \boxed{\varphi, \sim\varphi} \not\trianglelefteq \alpha\} \ = \ \{\alpha \in \Sigma \mid (\forall \varphi \in \alpha)\ \sim\varphi \notin \alpha\} \ .$$

Turning to (2), we have

$$\begin{aligned}
\Box \Rightarrow \boxed{\varphi} \mid \boxed{\sim\varphi} \ &= \ \overline{\Box_{\trianglerighteq}{}^{*}\ \left(\Box_{\trianglerighteq} \cap \overline{(\boxed{\varphi} \mid \boxed{\sim\varphi})_{\trianglerighteq}}\right)\ \Box_{\trianglerighteq}{}^{*}} \\
&= \ \overline{\Box_{\trianglerighteq}{}^{*}\ \{\alpha \in \Sigma \mid \text{not } (\alpha \trianglerighteq \boxed{\varphi} \text{ or } \alpha \trianglerighteq \boxed{\sim\varphi})\}\ \Box_{\trianglerighteq}{}^{*}} \\
&= \ \{\alpha \in \Sigma \mid \varphi \in \alpha \text{ or } \sim\varphi \in \alpha\}^{*} \ .
\end{aligned}$$

Allowing φ to range over all fluents, and imposing all instances of (1) together with (2), we get the sub-alphabet

$$\begin{aligned}\Sigma_m &=_{def} \bigcap_{\varphi\in\Phi} \{\alpha \in \Sigma_1 \mid \varphi \in \alpha \text{ or } \sim\varphi \in \alpha\} \\ &= \{\alpha \in \Sigma_1 \mid (\forall \varphi \in \Phi)\ \varphi \in \alpha \text{ or } \sim\varphi \in \alpha\} \ .\end{aligned}$$

Clearly, Σ_m consists of the elements α in Σ_1 that are $\subseteq$-maximal (in Σ_1)

$$(\forall \beta \in \Sigma_1) \quad \alpha \subseteq \beta \text{ implies } \alpha = \beta \ .$$

Next are the inertial laws (3) and (4)

$$\begin{aligned}\boxed{\varphi}\Box &\Rightarrow \Box\boxed{\varphi} \mid \boxed{\mathsf{F}\varphi}\Box \\ \Box\boxed{\varphi} &\Rightarrow \boxed{\varphi}\Box \mid \boxed{\mathsf{F}\varphi}\Box \ .\end{aligned}$$

Having seen (9) at work on (1) and (2), we trust that the reader will agree that (3) says that

> if at some point, φ holds but $\mathsf{F}\varphi$ does not, then φ will hold at the next point.

(4) is similar, and coupled with (3), makes φ "inertial." Note that (3) and (4) admit strings such as

$$\boxed{\varphi, \mathsf{F}\varphi\,|\,\varphi}$$

in which a force is applied to φ without any evident effect on φ. We can (if we are so inclined) rule out such vacuous forces by imposing the additional constraint

$$\boxed{\varphi, \mathsf{F}\varphi}\Box \quad \Rightarrow \quad \Box\boxed{\sim\varphi}$$

which would effectively strengthen (3) and (4) to

$$\begin{aligned}\boxed{\varphi}\Box &\Rightarrow \Box\boxed{\varphi} \mid \boxed{\mathsf{F}\varphi\,|\,\sim\varphi} \\ \Box\boxed{\varphi} &\Rightarrow \boxed{\varphi}\Box \mid \boxed{\mathsf{F}\varphi, \sim\varphi}\Box\end{aligned}$$

respectively. Be that as it may, let us fix a set $\mathsf{Inr} \subseteq \Phi$ of *inertial* fluents, and assume that $\Phi - \mathsf{Inr}$ includes the set

$$\mathsf{FInr} \ =_{def} \ \{\mathsf{F}\varphi \mid \varphi \in \mathsf{Inr}\}$$

of non-inertial force fluents. Writing IC_φ for the intersection of the languages (3) and (4), we can characterize the *inertially complete* strings of [6] as the elements of the regular language

$$IC \;\; =_{def} \;\; \bigcap_{\varphi \in \mathsf{Inr}} IC_\varphi \; .$$

(Recall that we are assuming Φ is finite, and hence so is $\mathsf{Inr} \subseteq \Phi$.)

4 Entailments implemented as regular relations

Given a language L that may or may not satisfy a constraint C, such as (1), (2), (3) or (4), what language can we extract from L that is certain to satisfy C? The intersection $L \cap C$ returns all the strings in L that respect C, and is a good answer for $C = (1)$. For $C = (2), (3)$ or (4), however, fluents appear on the right side of $\Rightarrow$, suggesting that a better answer is

$$L_{\trianglerighteq} \cap C \;\; = \;\; \{s \in C \mid (\exists s' \in L)\; s \trianglerighteq s'\}$$

in which we take the subsumption closure of L before intersecting it with C. Under the model-theoretic account developed in [5], a language L is instantiated in a model iff $L_{\trianglerighteq}$ is. Building on this, the case $C = (2)$ will prove useful in the next section.

Moving on to (3) and (4), the right hand sides of (3) and (4) present a choice between adding φ or $\mathsf{F}\varphi$, over which we may want some control. With this in mind, we recall (from [5]) the binary *superposition* operation & on languages that overlays comic strips of equal length

$$L \& L' \;\; =_{def} \;\; \bigcup_{n \geq 0} \left\{ (\alpha_1 \cup \beta_1) \cdots (\alpha_n \cup \beta_n) \mid \alpha_1 \cdots \alpha_n \in L,\; \beta_1 \cdots \beta_n \in L' \right\} \; .$$

& is commutative and associative, with $L \& L' \trianglerighteq L$ — that is, $L \& L' \subseteq L_{\trianglerighteq}$. Now, back to constraints (1), (3) and (4), which collectively define the regular language ${\Sigma_1}^* \cap IC$. Instead of turning L into

$$L_{\trianglerighteq} \;\cap\; {\Sigma_1}^* \;\cap\; IC \; ,$$

we can opt either to add only inertial fluents

$$(L \;\&\; \mathsf{Pow}(\mathsf{Inr})^*) \;\cap\; {\Sigma_1}^* \;\cap\; IC \tag{10}$$

or only force fluents

$$(L \;\&\; \mathsf{Pow}(\mathsf{FInr})^*) \;\cap\; {\Sigma_1}^* \;\cap\; IC \tag{11}$$

(or to variants of (10) and (11) with Inr reduced to its fluents occurring in L). In the case of (10), we are assuming the only forces at work are those that are explicitly mentioned, compelling us to add at most inertial fluents. (11) is the exact opposite: assuming that the only inertial fluents that hold are the ones listed, then the only fluents we can add to form an inertially complete string are fluents from FInr. The defeasible inertial inferences commonly described are given by (10), the idea being to minimize the force fluents $\mathsf{F}\varphi$. Abductive explanations, on the other hand, correspond to (11), assuming the inertial fluents are taken to be the observations which we are called upon to explain (via force fluents). In the next section, we equate (10) and (11) with what [6] calls the *inertial completion* and *minimal inertial cover*, respectively. But first, let us make good on the claim behind the title of this section — that entailments are implemented here as regular relations.

Given a finite-state machine $\langle Q, q_0, F, \rightarrow\rangle$ accepting L, the relation

$$\&_L \;=_{def}\; \bigcup_{n\geq 0}\{(\alpha_1\cdots\alpha_n, \gamma_1\cdots\gamma_n) \in \Sigma^n \times \Sigma^n \mid (\exists \beta_1\cdots\beta_n \in L)\ \gamma_i = \alpha_i \cup \beta_i \text{ for } 1 \leq i \leq n\}$$

is computed by the finite-state transducer $\langle Q, q_0, F, \rightsquigarrow\rangle$, where

$$q \overset{\alpha:\gamma}{\rightsquigarrow} q' \quad \text{iff} \quad (\exists \beta \in \Sigma)\ \gamma = \alpha \cup \beta \text{ and } q \overset{\beta}{\rightarrow} q' \ .$$

Writing Δ_L for the identity relation $\{(s,s) \mid s \in L\}$ on L, $\circ$ for relational composition, and $\mathsf{image}(R)$ for $\{s' \mid (\exists s)\ sRs'\}$, note that

$$L\&L' \;=\; \mathsf{image}(\Delta_L \circ \&_{L'})$$

and that $L_{\trianglerighteq} = L\&\ \Sigma^*$.

5 Comparing (underspecified) representations

The second construct we poach from the Xerox toolkit is the *replace operator* $->$ which maps an *upper* language to a *lower* language. In its simplest form, $->$ provides a language satisfying (3) and (4) along the lines of (11). Form

$$\alpha\beta \;\; -> \;\; (\alpha \cup \boxed{\mathsf{F}\varphi})\beta \qquad \text{for } \varphi \in \alpha - \beta \tag{12}$$

assuming for convenience, $\mathsf{F}\sim\varphi = \mathsf{F}\varphi$ and $\sim\sim\varphi = \varphi$. (12) replaces all occurrences of $\alpha\beta$ such that $\varphi \in \alpha - \beta$ by $(\alpha \cup \boxed{\mathsf{F}\varphi})\beta$. Applying all instances of (12) for $\varphi \in \mathsf{Inr}$ produces *essentially* the same result as (11) — "essentially"

that is, according to the following notion of semantic equivalence. Given a language L, let us impose the constraints (1), (2) on L to define

$$L_m \quad =_{def} \quad L_{\trianglerighteq} \cap {\Sigma_m}^*$$

(whence $(L\&L')_m = L_m \cap {L'}_m$, etc). Now, let $\equiv_m$ be the equivalence relation on languages L and L' induced by $\cdot_m$

$$L \equiv_m L' \quad \text{iff} \quad L_m = {L'}_m \ .$$

Under the model-theoretic interpretations given in [5], $L \equiv_m L'$ means L is instantiated in a model iff L' is. Now, returning to (11) and (12), these are "essentially" the same in that for any given language L, the language (11) is $\equiv_m$-equivalent to $\mathsf{image}(\Delta_L \circ R)$ where R is the relation given by (12) and (1). Indeed, the latter is what [6] calls the *minimal inertial cover of* L.

What about the language (10) obtained from a language L in order to satisfy (1), (3) and (4) by adding only inertial fluents? A first attempt at deriving a $\equiv_m$-equivalent language via the replace operator is given by (13), (14).

$$\alpha\beta \quad -> \quad \alpha(\beta \cup \boxed{\varphi}) \qquad \text{for } \varphi \in \alpha,\ \mathsf{F}\varphi \notin \alpha \tag{13}$$

$$\alpha\beta \quad -> \quad (\alpha \cup \boxed{\varphi})\beta \qquad \text{for } \varphi \in \beta,\ \mathsf{F}\varphi \notin \alpha \tag{14}$$

In contrast to (12), however, we may need to apply (13) and (14) repeatedly, as the addition of an inertial fluent φ may trigger further inertial inferences down the line. (Consider $\boxed{\varphi}\Box\Box$.) It would help if we could the turn a finite-state transducer of a relation R into a finite-state transducer for the reflexive-transitive closure (with respect to relational composition, not concatenation) of R, but this is not, it seems, possible for arbitrary regular relations R.

As it turns out, however, Xerox provides more sophisticated ways of applying $->$, which will do for (10). The regular relation

$$L \quad -> \quad L' \ \backslash / \ L''\_$$

says

> replace L substrings with a string from L' as long as the *lower*-side string is preceded by an L'' and leave everything else unchanged

while

$$L \quad -> \quad L' \ \backslash / \ \_L''$$

says the same, if we substitute "preceded" by "followed." The formal definition of these operators is complex, requiring more space than we have available here. The full definition is available in [9]. As Karttunen says, the advantage of these (and similarly restriction) is that they "encode in a compact way a useful condition that is difficult to express in terms of the more primitive operators" ([10]).

Armed with these constructs, we can sharpen (13) and (14) to

$$\beta \; -> \; \left(\beta \cup \boxed{\varphi}\right) \; \backslash/ \; \alpha\_ \qquad \text{for } \varphi \in \alpha,\ \mathsf{F}\varphi \notin \alpha \tag{15}$$

$$\alpha \; -> \; \left(\alpha \cup \boxed{\varphi}\right) \; \backslash/ \; \_\beta \qquad \text{for } \varphi \in \beta,\ \mathsf{F}\varphi \notin \alpha \tag{16}$$

respectively. (15) and (16) return what [6] calls *inertial completions*. We have yet to compare the finite-state machine constructed in [6] to what Xerox makes of (15), (16). There are any number of metrics relative to which implementations of a language or relation can be compared (number of states and transitions, time to compile, etc). And the matter is further complicated by the fact that we may wish to compare machines for $\equiv_m$-equivalent languages. These languages may be far from identical, as the approach via & in section 4 makes clear, tending, as it does, to add more fluents. On the positive side, & provides a very general method for turning restrictions into regular relations (as illustrated by (10), (11) and their variants described parenthetically). The question arises: to what extent can we rewrite restrictions $\Rightarrow$ to replacements $->$, yielding $\equiv_m$-equivalent lower languages? This is one of several matters we hope to explore in the future.

Acknowledgements

We thank our IWCS referees for their helpful comments, a good few of which we regret we could not address here, but will keep in mind as we pursue the work further. Rowan Nairn is supported by the Irish Research Council for Science, Engineering and Technology, funded by the National Development Plan.

References

[1] K. R. Beesley and L. Karttunen. *Finite State Morphology.* CSLI Publications, Stanford, 2003.

[2] D. R. Dowty. *Word Meaning and Montague Grammar.* Reidel, Dordrecht, 1979.

[3] D. R. Dowty. The effects of aspectual class on the temporal structure of discourse: semantics or pragmatics? *Linguistics and Philosophy*, 9:37–61, 1986.

[4] T. Fernando. Reichenbach's E, R and S in a finite-state setting. Sinn und Bedeutung 8, Frankfurt, 2003 (`www.cs.tcd.ie/Tim.Fernando/wien.pdf`).

[5] T. Fernando. A finite-state approach to events in natural language semantics. *Journal of Logic and Computation*, 14(1):79–92, 2004.

[6] T. Fernando. Inertia in temporal modification. In *Proc. Semantics and Linguistic Theory XIV* (Northwestern University, 2004). To appear Cornell Linguistics Circle Publications (`www.cs.tcd.ie/Tim.Fernando/nw.pdf`).

[7] F. Hamm and M. van Lambalgen. Event calculus, nominalisation, and the progressive. *Linguistics and Philosophy*, 26(4):381–458, 2003.

[8] R. Jackendoff. The proper treatment of measuring out, telicity, and perhaps even quantification in English. *Natural Language and Linguistic Theory*, 14:305–354, 1996.

[9] L. Karttunen. The replace operator. In E. Roche and Y. Schabes, editors, *Finite-state language processing*, Language, Speech and Communication, pages 117–147. MIT Press, Cambridge (MA), 1997.

[10] L. Karttunen. Applications of finite-state transducers in natural language processing. In *Implementation and Application of Automata*, LNCS 2088, pages 34–46. Springer-Verlag, 2001.

[11] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In M. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[12] J. Pustejovsky. The syntax of event structure. *Cognition*, 41:47–81, 1991.

[13] M. Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, Cambridge (MA), 1997.

[14] M. Steedman. *The Productions of Time*. Draft, `ftp://ftp.cogsci.ed.ac.uk/pub/steedman/temporality/temporality.ps.gz`, 2000. Subsumes 'Temporality,' in J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, 895–935, Elsevier North Holland, 1997.

[15] C. Tenny. Grammaticalizing aspect and affectedness. Dissertation, Department of Linguistics and Philosophy, MIT, 1987.

[16] G. van Noord and D. Gerdemann. Finite state transducers with predicates and identity. *Grammars*, 4(3), 2001.